圧縮テクスチャが使えないなら…
シェーダ書けばいいじゃない
圧縮テクスチャ解決編

内容

さて、前回までで 2D のドット絵において、現状としては「圧縮テクスチャは適切な選択肢ではない」ことが分かりました。
ではいつものように PNG でやればいいのか？って話ですが、その発想しか出ないのであれば 2D ゲームプログラマとしてはちょっと残念かも。

さっきの話でも言いましたが PNG では結局フルカラー分(RGBA=8×4=32bit のピクセル数分)のメモリを消費してしまうからです。どうにかならないのでしょうか？

ここで旧時代の遺物を掘り起こします。それは…**パレットテクスチャ**です

パレットテクスチャって何？

簡単に言うと、予め「カラーテーブル」を用意しておきます。なお０番は通常は「透明」にします。

それぞれの色を RGBA として指定するのではなく、インデックス…つまり番号で指定します。
通常、番号は 0～255 で指定します。このため、テクスチャはグレースケールで済むわけです。
つまり A8 だの R8 というように 8bit で指定できます。

歴史的経緯

ところで、このパレットテクスチャが廃れてしまった理由はあるんです。
DX9 以降の GPU においては、フルカラーが当たり前みたいな状況になったためパレットテクスチャの意義はなくなってきたんです。
昔の GPU ではパレットもハードウェア解決してくれてましたが、時代の流れでその機能もなくなってしまいました。

そんな状況でパレットを実装しようとすると逆に手間はかかるし、手間がかかるということ

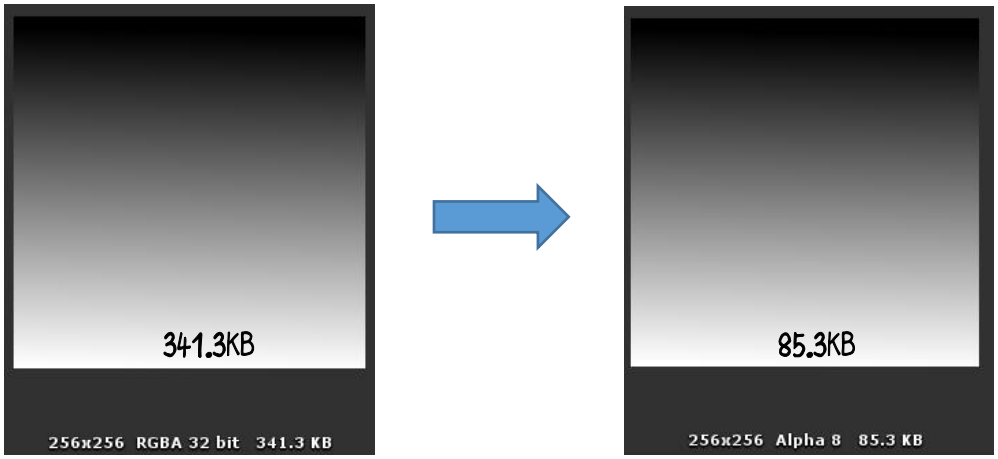
は遅くなることを意味します。

というわけで、いつのころからかパレットテクスチャは廃れてしまいました。ところが現代の状況では、モバイル 2D ゲームのためにテクスチャ容量を削減するためのコストが結構バカにならなくなってきました。

そこでまず思いつくのが圧縮テクスチャなんですが、これはお分かりの通りかなりキッツい劣化をします。そこでパレットテクスチャの復権です。

問題解決のために温故知新

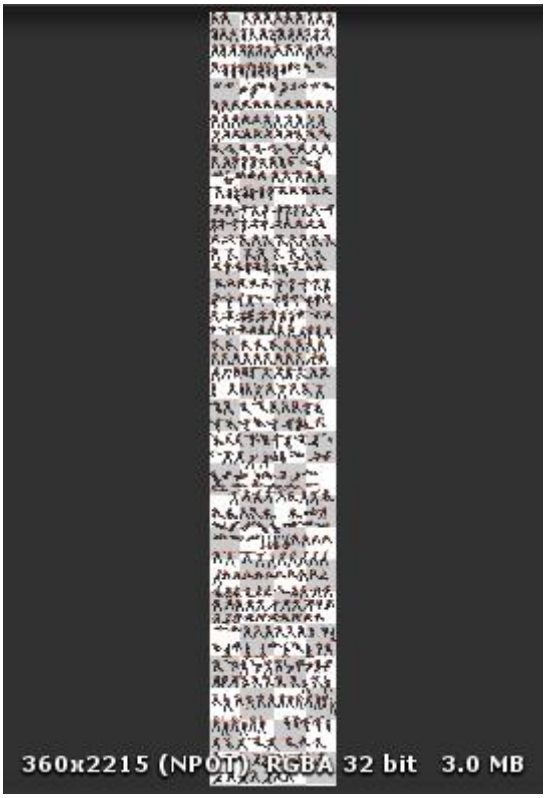
パレットテクスチャを使用してみます。なお、思考実験として 256×256×RGBA 表現について考えてみます。どれくらい圧縮できるでしょうか？



4 分の 1 になることが分かりますね。しかしパレットテクスチャが必要です。256 色であれば 16×16 フルカラーあればいいので

フルカラーにしても 1.3KB ですね。ということは 3.94 分の 1 ということになります。ちょっと考えると、これは画像がでかければデカイほど、圧縮率が上がることを意味しています。ちなみにテリー•ボガードなら

ちなみに360×2215になるわけです。そのまま使用すると3.2MBくらいになります。アーケードでもその状況だと結構キツイですよ。



そもそも単位(KB→MB)が変わっちゃいましたね。そのレベルです。この場合ほぼ4分の1になります。そう言うてもいいんじゃないかな。

さて、とはいえじゃあこの考えを実装にまで持っていくのが容易かというと(面倒じゃないかというと)そうでもなかった…OTL。

パレットとインデックステクスチャに分解

まず、ここでちょっと詰まった。
そうだ、通常の場合だとドット絵を書いてもパレット情報自体がないんじゃないか？あったとしても、それを画像化しないとどの道 Unity では使えないじゃん。

という事で、必要なデータとしては

- パレットデータ(フルカラー)
- インデックスデータ(A8)

である。
これは分かってるが A8 のデータを作ったところで Unity はそんなもんデフォルトで読み込ん

でくれない！…でも、インポータを作りたくない！(ていうか発表に間に合わん)。
という事で、C#でツール作った(ギリギリすぎんよ～)。

47 分で作った。
C#は暫くいじってなかったので苦労した(憔悴)。

ちなみにアルゴリズムは単純で、ハッシュテーブルに色情報入れて重複無いように並べりゃパレットの出来上がり。ねっ、簡単でしょ？

インデックスデータは、その色情報から、色が格納されているインデクスを取り出し、
col=Color.FromARGN(index,index,index,index);
として格納。

よく考えたら、Unity 側にグレースケールをアルファとして使用するパラメータがあったため、アルファ部分はいらなかった…訴訟も辞さない。

もうね、ツールのコーディングひどすぎるので、お見せできません。本当にすまない。

で、このツールにより出力されるファイルは

- 元ファイル名_idx.png(インデックスデータ)
- 元ファイル名_plt.png(パレットデータ)

としました。

出力されたデータは以下のようになりました。

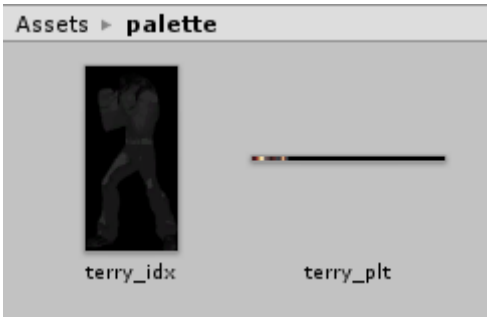
terry_idx.png

terry_plt.png

うっすら見えてるかな？うっすらグレーがインデックスデータで、見るからにパレットが右の奴です。

Unity にインデックスとパレットを取り込み

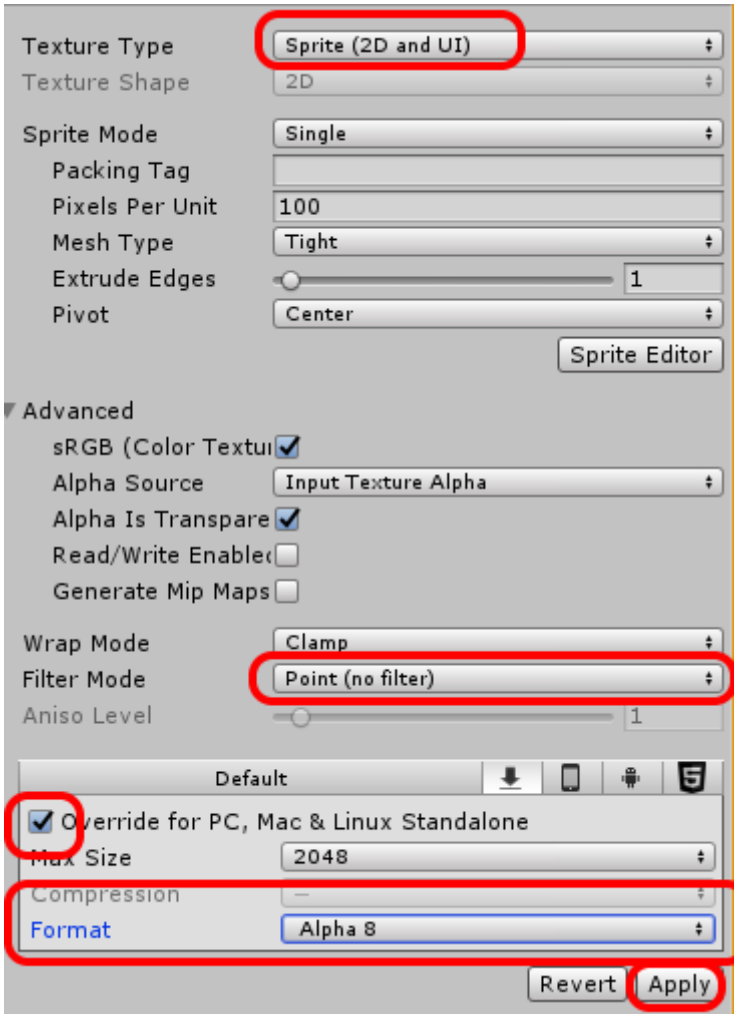
まずはそれぞれをそのままドラッグアンドドロップしてしまいましょう。
取り込んだばかりだとこんな状態です。



で、ここからちょっと面倒です。インデックスとパレットのインポート設定をしっかりやらないと変にハマります(ハマりました)。

インデックス側インポート設定

まずインデックス側の設定です。



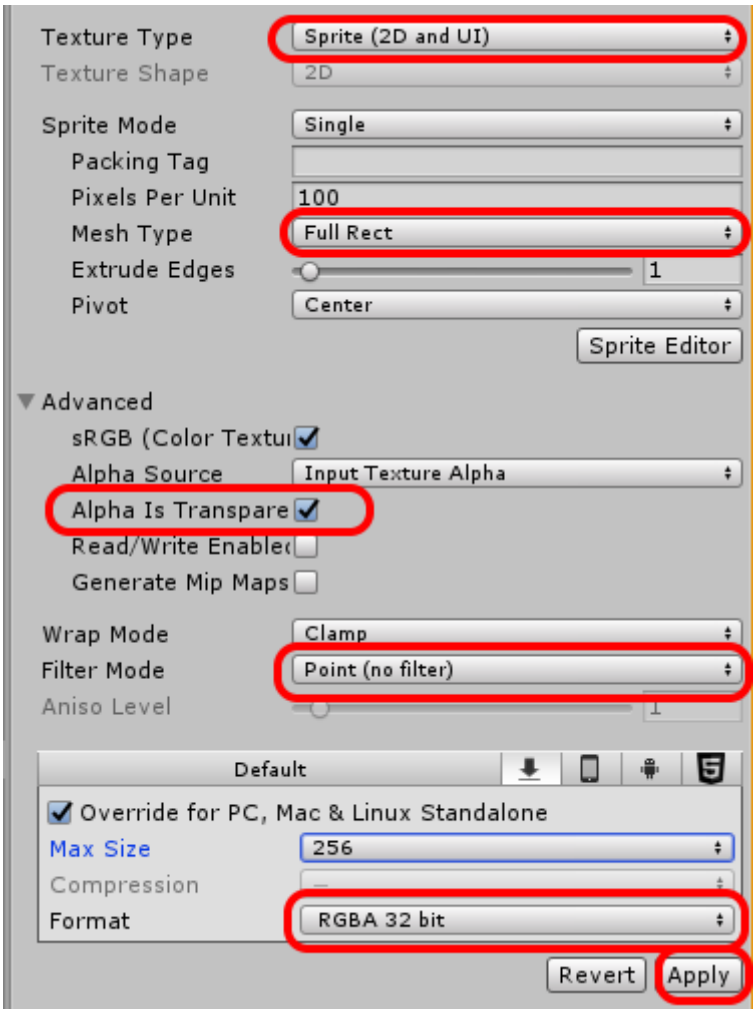
スプライトとして使用するなら TextureType は Sprite にしておきましょう。
次に、FilterMode の部分は Point にしておきます。

今回のデータはインデックスデータなので Bilinear にして補間されてしまうと大変なことになるのが…わかるだろう？

で、次に Override for ～ の部分にチェックを入れ、Format の部分をいじります。デフォルトだと DXT5 だの PVRTC だのという圧縮テクスチャになってるんですが…
次にこれを Alpha8 にします。これをしないと今までわざわざやってきたことの意味がなくなるからです。
Alpha8 にすることでデータサイズが元の画像の 4 分の 1 になります(GPU 側でも 4 分の 1)

最後に Apply して終わりです。

パレット側インポート設定



ほぼ似たような設定だけど、パレット側で忘れちゃいけないのがAlphaIsTranspareです。これ忘れると、透明度が透明じゃなくなるので注意(インデックス側は別にどっちでもいい)

そしてこれも補間されては困るためFilteModeはpointにしておきます。

で、ここがインデックスと全然違う所ですが、Override for ～にチェックを入れた後、フォーマットを指定するのですが、ここはためらわずにRGBA32bitにしてください。

やったらApplyで。

さて、取り込み設定はできたので、次にシェーダとマテリアルを用意します。

シェーダ作成

Unity 触ってても シェーダやったことない人が多いかもしれませんが、サーフェスシェーダくらいは触っておいた方が良いかと思います。

とはいえ、今回は表面の材質はほぼ関係なく、テクスチャを表示できればいいので
右クリ→Create→Shader→Unlit Shader あたりで作りましょう。シンプルなんで。で、名前を palette かなんかにしておきましょう。

こんなのができるはず

ダブルクリックするとシェーダファイルが開きます。

上から見ていくと、

```
Properties
{
        _MainTex ("Texture", 2D) = "white" {}
}
```

って書いてあるんですが、ここが外部からテクスチャを指定する部分なので、ここをちょっと書き換えます。

```
Properties
{
        _MainTex("index", 2D) = "white" {}
        _PaletteTex("palette", 2D) = "white" {}
}
```

メインの奴の名前を変えて、パレットテクスチャを追加します。

次にその下の Sub Shader 以降ですが、まず最初にタグをいじる必要があります。タグについては

https://docs.unity3d.com/jp/540/Manual/SL-SubShaderTags.html
↑の公式ドキュメントを参照しましょう。

元はこう書いてありますが
Tags { "RenderType"="Opaque" }

これはスプライト向きではないのでこうします。
Tags{ "Queue" = "Transparent" "IgnoreProjector" = "True" "RenderType" = "Transparent" }
ちなみに、Transparent はアルファありの時には指定します。次の IgnoreProjector はパース無視するかどうか。RenderType も Transparent です。

こいつはスプライト用なので Z バッファを OFF にして、アルファを有効にします。
また、LOD なんちゃらは消していいです。2D には関係ありません。

ので、LOD の行を削除して以下のコードを追加します。

```
ZWrite Off
Blend SrcAlpha OneMinusSrcAlpha
```

あとは、頂点シェーダとピクセルシェーダを以下のように書き換えます。

```
sampler2D _MainTex;
sampler2D _PaletteTex;

v2f vert (appdata v)
{
	v2f o;
	o.vertex =  UnityObjectToClipPos(v.vertex);
	o.uv = v.uv;// TRANSFORM_TEX(v.uv, _MainTex);
	return o;
}

fixed4 frag (v2f i) : SV_Target
{
	// sample the texture
	fixed4 idxf = tex2D(_MainTex, i.uv);
```

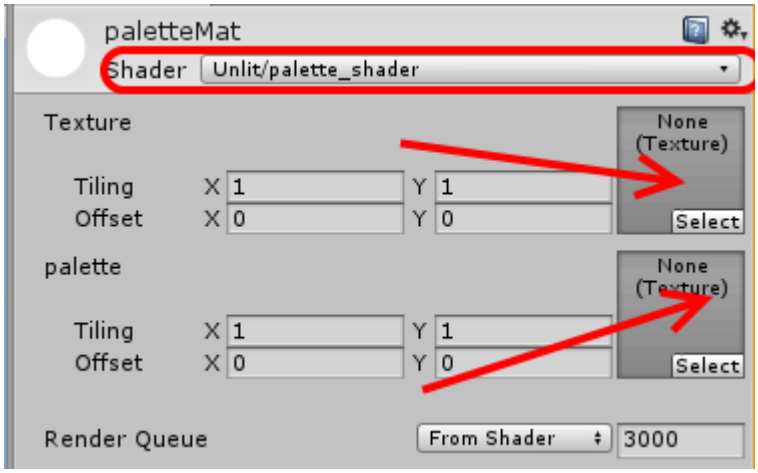
```
        fixed4 col = tex2D(_PaletteTex, float2(idxf.a+(0.5/256.0),0.5));
        return col;
}
```

色々試行錯誤した結果なので、これもうわかんねえな。

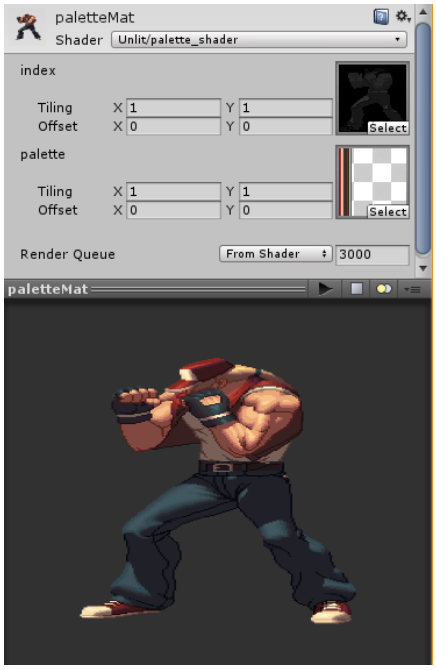
ちなみに 0.5 を足しているのは、ピクセルの中心に寄せるためです。

さて、こうやってできたシェーダを利用するにはマテリアルを作ります。
create→Material で
paletteMat かなんかそんな名前にします。

で、そのマテリアルのシェーダをさっき作った palette シェーダを指定すると、二つ設定すべきテクスチャが見えます。



ここにそれぞれ、先ほどインポート設定したインデックスとパレットを設定します。
そうすれば…



やったぜ!
いや、まだまだ。

最後にスプライトとして利用するために今度はヒエラルキーから Create->2D->Sprite でスプライトオブジェクトを作ります。

そのマテリアルを、このマテリアルで設定すれば…

はいこのとおり。

最後に

ちょっと今回は突貫工事だったので、かなりガバガバでした。ツールとかシェーダ自体は今後改善していこうと思います。

なお、今回の資料とツールは
https://goo.gl/VbLV6r

に上げておきましたが、資料には版権ものが結構あるので、個人的利用に留めてください。