圧縮テクスチャについて

ゲームのお仕事された人はご存知だと思いますが、ゲームで使用されるテクスチャには

- 圧縮テクスチャ
- 非圧縮テクスチャ

があります。でも知らない人も多いと思うので、説明します。ここを理解してないとアーティストは泣き、プログラマは叫び、プロジェクトが死ぬかもしれませんので。

「圧縮テクスチャって PNG とか JPG ちゃうのん？」
と思われるかもしれませんが、ちょっと違います。

PNG とか JPG は、読み込む時に展開(解凍)され、結局は 24bit だか 32bit になります。
つまり**グラボに投げる段階で、RAW データ(なんの圧縮処理もしてない)と同じ状況**になっちまいます。
だから、僕らが日常的に使っているテクスチャデータはゲーム的には非圧縮テクスチャと思っていいです。

大雑把に言うと、グラボ的には PNG だろうが JPG だろうが BMP だろうが TIFF だろうが同じなわけです。とりあえずこの点をまずは押さえておきましょう。

じゃあ、圧縮テクスチャって何なの？って話ですが、ターゲットプラットフォームによって変わってきますが
大体よく使用されるのが

- DXTC(S3TC)：主に DirectX 系で使われる
- ETC：主に Android 系で使われる(Google で検索すると高速道路のアレがヒットする)
- PVRTC：主に iOS 系で使われる

などです。
ちなみに全部最後に"TC"ってついてますが、これは"Texture Compression"の略です。
他にも BC とか ATITC とかいろいろありますが、きりがないので、この三つの話です。
あと「主に」って書いたのは、いろいろ事情が絡んでて、〇〇なら絶対これ！みたいな言い方ができないんです。

なお、日本語 Wikipedia 解説があるのが DXTC のみっぽいので、圧縮テクスチャの勉強を始めるなら DXTC からやるといいと思います(ちなみに BC は DXTC とほぼ同じだけど DX11 から DXTC→BC と変わった。でも検索しづらいので DXTC でいいでしょう)
目的と機能は同じなのでさっさと統一してほしいところですが、それぞれ企業がライセンス持ってるのでそういうわけにもいかないっぽいのです。

これらの圧縮テクスチャの特徴は

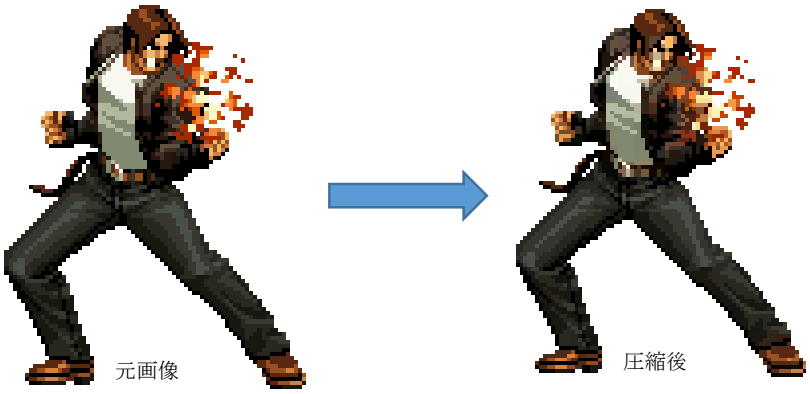
- **非可逆圧縮である(劣化が避けられない)**
- ４x４ブロックを２つの代表色から再現する
- 圧縮状態でグラボにそのまま投げられる(ハードウェアでサポートされている)
- アセット上(HDD)での圧縮率は PNG や JPG に大きく劣る

です。ゲームに使用する以外の利点は…あまりないんじゃないかな。

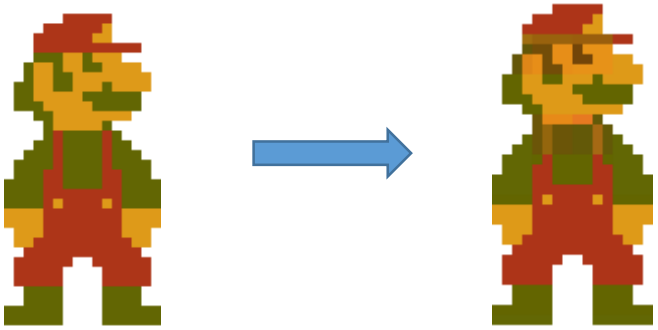
とりあえず、プログラマ、アーティストともに「劣化する」ってのは覚えておいてください。
特に「ドット絵」などのような 1 ピクセルでも色が変わると致命的な画像には向いていませんね。
ちなみに実際に圧縮した画像がこちら

顔の色が劣化しているのが、わかるだろう？あとよく見ると服とか炎もね。ちなみにファミコン版のスーマリだとこう…

違いがわかりにくいですが、赤が結構緑に侵食されてますね。
で、**ここまでだとドット屋の敵にしか思えない**んですが、一応この圧縮、圧縮時の設定で品質の調整ができます。今回の例では最低品質にしてみました。次は最高品質で確認してみましょう。

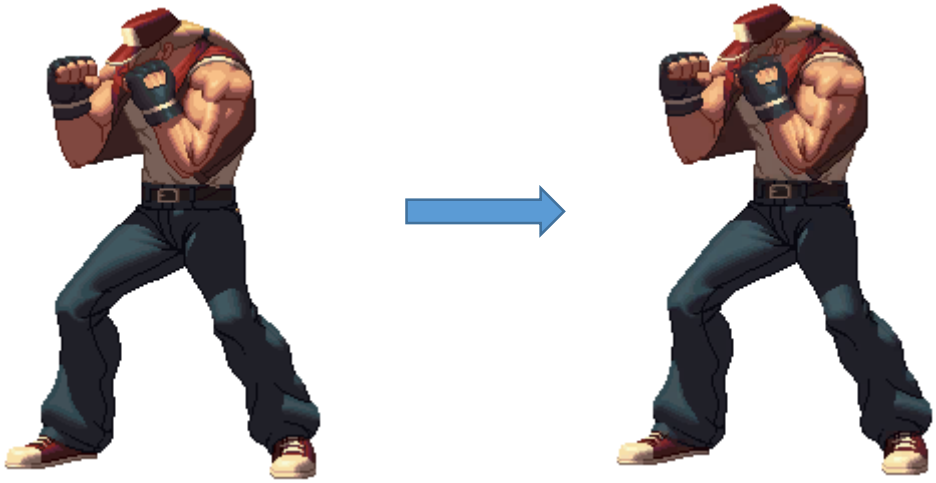


うーん。さっきよりかはマシだけどって感じですね。ちなみにマリオはこうなります。



これはひどい。これはドット屋さん激おこですわ。

ちなみに KOFXIII くらいの解像度になると



かなり劣化具合が分かりづらくはなります。左肩にちょっと劣化が見られるくらいですね。これらの事から、**解像度の小さいドット絵には使用しないほうが良い**です。別方面から負荷を減らしていきましょう。(ちなみにドット絵にこだわる某社では上のでもアーティストさんはキレます。注意しよう！)

とりあえずここで覚えておいてほしいのは「**同じ圧縮形式でも設定により品質に差が出る**」です。ちなみに JPG と違って、**品質が良かろうが悪かろうが圧縮後のサイズは同じ**です。違いは圧縮にかかる時間だけ(展開にかかる時間は一定)なのでゲームの場合は通常は品質を最高にしておきます。また、DXTC に関しては DirectX に最初から付属しているものより nvCompress という nVidia のツールで圧縮した方が品質がいいように思えます。

ちなみに、DXTC,ETC,PVRTC ともに「2 つの代表色からルックアップテーブルを作って、そのインデックスで 4x4 のピクセル色を決定する」という特性上、同じような問題は発生します。

時間がないのでルックアップテーブルについての細かい内容はここでは割愛します。余裕があれば後で解説します。

圧縮テクスチャは劣化するからが使い物にならないかというと、そうではなくて、自然物のテクスチャや、写真系や、解像度のデカい背景などでは劣化が目立たないため、そちらに使用するとよいでしょう。エッジが大事な奴には向いてないと思います。

ここまでだとデメリットばかりに見えますが、前に説明したように、圧縮状態でグラボに投げる事ができるため、**解凍による CPU 負荷＆メモリコストや、メモリからグラボへの転送**

コストが軽減できます。これは割と大きいと思います。特にこの辺はスマホにおいてはバカにならないコストと関わってきます。メリットとデメリットを理解して使用していきましょう。

DXTC には 5 種類のフォーマットがあり、それぞれに特性があります。大きく分けると DXT1 とそれ以外(2～5)という分け方にります。

違いは
DXT1：アルファなし圧縮→1 ピクセル当たり 4 ビット
DXT2～5：アルファあり圧縮→1 ピクセル当たり 8 ビット

となります。ちなみに DXT1 にはアルファ付き DXT1 ってのもあって、そいつの場合は 1 ビットアルファであるため「画素を抜くか抜かないか」の指定しかできない。
ところで、4x4 ブロックごとに圧縮するアルゴリズムであるため、123×331 のような中途半端な画像サイズは使用できない。いいね？
☆最後に、アルファ付き DXT1 は別の事情で「DXT2～5 より画質が劣ることがある」ので、これも覚えておきましょう。特に暗い部分の質が悪いです。仕様ビット数が中途半端であるため、ハードウェアとの相性が悪いためです。そういう理由で「アルファ付き DXT1」はあまり使われていません。

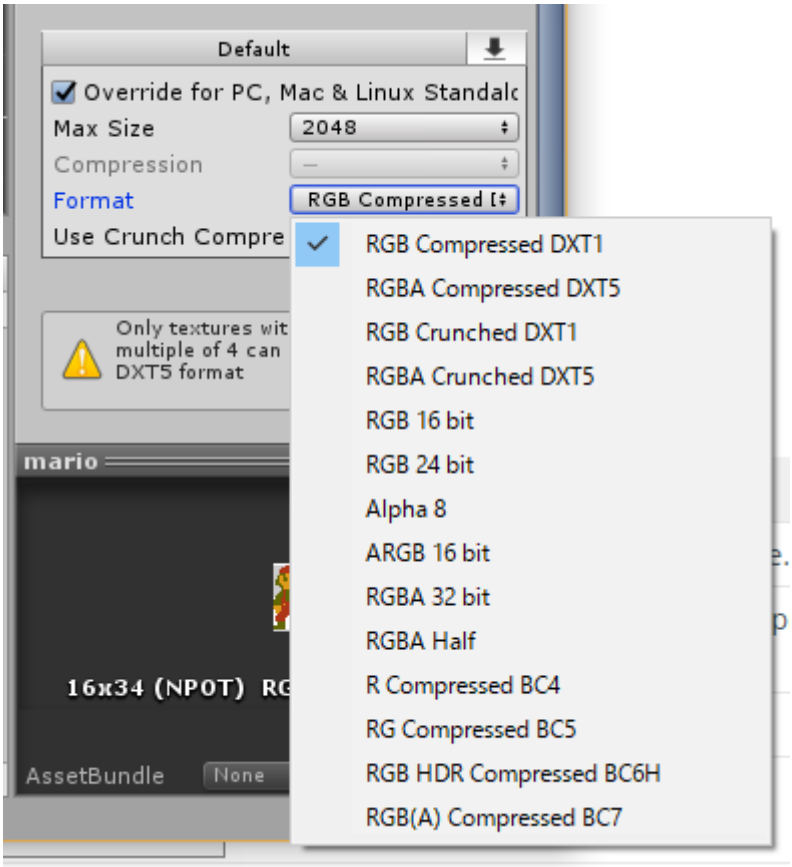
Unity における圧縮テクスチャ

まずマニュアルを見てみましょう。
https://docs.unity3d.com/jp/540/Manual/class-TextureImporter.html
下の方に
「プラットフォームごとのオーバーライド」と書いてある部分です。

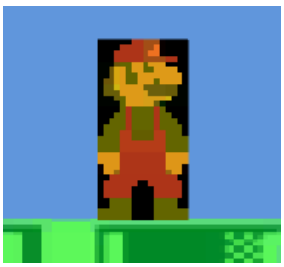
Texture Format	テクスチャの内部表示を設定します。サイズを上げると画質が下がるというように、サイズと画質のトレードオフとなります。
Compressed	圧縮された RGB テクスチャです。これは、デフューズテクスチャの最も一般的な形式になります。ピクセルあたり 4 ビット (25
16 bit	低画質 True Colorです。16 段階の赤、緑、青、アルファがあります。
Truecolor	これが一番画質のよい形式です。 256x256 サイズのテクスチャで 256KBになります。
Crunched	Crunch は DXT テクスチャ圧縮のうえ、不可逆圧縮形式です。実行時、テクスチャは CPU で DXT 形式に変換され、ランタイムに GP ード時に使用するスペース量をできるだけ少なくしたいときに役立ちます。Crunch テクスチャは、圧縮するのにかなり時間がかか

ん？
という事は、textureformat を compressed にすると DXT1 状態になるという事か？

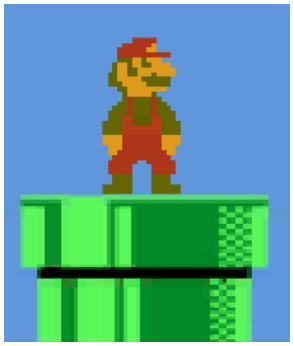
検証①：さっきのマリオ画像を読み込んでみる
あれ…？



なんかも一元の compress ってのが評判悪かったのか、きちんと DXT1 と DXT5 って明記されてるね。DXT1(アルファなし)で読み込むと、当然



抜けません(きたない)。
これを DXT5 にすると…抜けます。



ちなみにこれも非常にわかりづらいですが、超劣化してます。ボタン吹っ飛んでます。

ヤバいですね。ボタンの吹っ飛び具合が自然すぎて
最初気が付きませんでした(;´Д`)
これでは使い物になりません。

ということで、もう一つ検証。

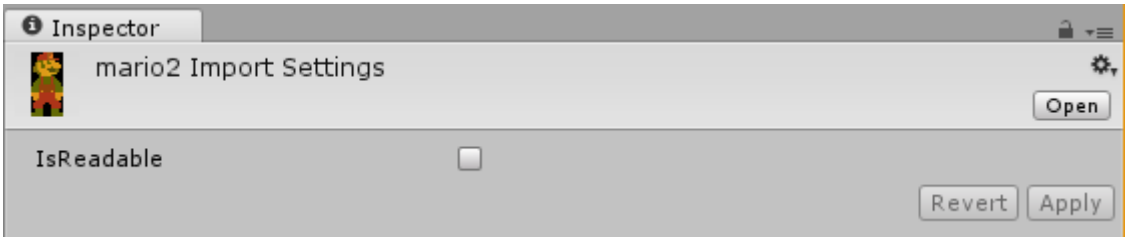
検証②：あらかじめ DXT5 で圧縮テクスチャにしたものをインポートしてみる。
何故かというと、Unity の内部の圧縮品質は調整できないので、事前にこちらで調整できればそっちの方がいいので。

こいつを使ってみます。

こいつも品質たいがい悪いですが、ボタン吹っ飛ぶよりマシです。
結果…

どうやら「画像ファイル」という認識はしてくれてますが、テクスチャアセットとして扱ってくれません。

インスペクタを見ると…

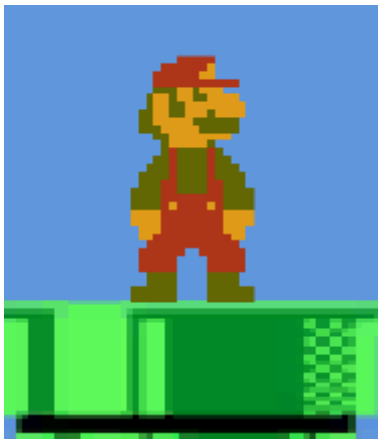


(´；ω；`)ｳｯ…

いや、プログラマなら何とかしようぜ。ってことで、動的に読み込むことを考えてみましたが…どうも労力に見合わなさそうなので、今回はやりません。

ということで、結論としては、他のフォーマットを検証してみたほうが建設的なようです。先ほど気になったのが BC7 です(対応ハードウェア少なそうですが)。

検証③：BC7 を試してみる



＼(^o^)／

Oh!!Good!!

ということで調査してみました。

調査結果：BC7 について

1 ピクセル当たり 8 ビット。つまり圧縮率的には DXT2～5 と同じようです。もうこれでええやん。と思うんですが一応現状におけるデメリットをあげておきます。

- 圧縮スピードが遅い
- 対応ハードウェアが少ない

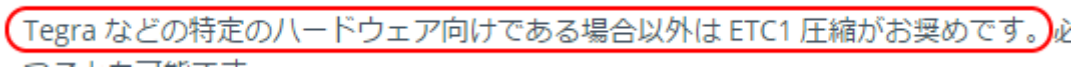
この辺もいずれ時間が解決してくれるとは思うんですが…現状のスマホゲーム開発に悩んでる人には何の助けにもならないね。仕方ないね。

ちなみに今回は DXTC を中心に解説してますが、プラットフォームがスマホ系だと、またちょっとややこしいです。

スマホ対応に向けて

とりあえず Android 対応の事を考えてみましょう。

https://docs.unity3d.com/jp/540/Manual/class-TextureImporterAndroid.html
Unity マニュアル→プラットフォーム別情報→Android→Android の 2D テクスチャオーバーライドを読んでみると…



はい…。
ちなみに「ホイール欲しいハンドル欲しい」情報(2011/02/16)から抜粋しますが

```
iOS
  PowerVR SGX           PVRTC

Android
  Qualcomm Adreno       ETC1, ATITC, 3DC, PALETTE
  ATI Imageon           ETC1, ATITC, 3DC, PALETTE
  Imagination PowerVR   ETC1, PVRTC
  NVIDIA Tegra2         ETC1, S3TC(DXT), LATC
  ZiiLabs ZMS-08 HD     ETC1, S3TC(DXT), PALETTE
```

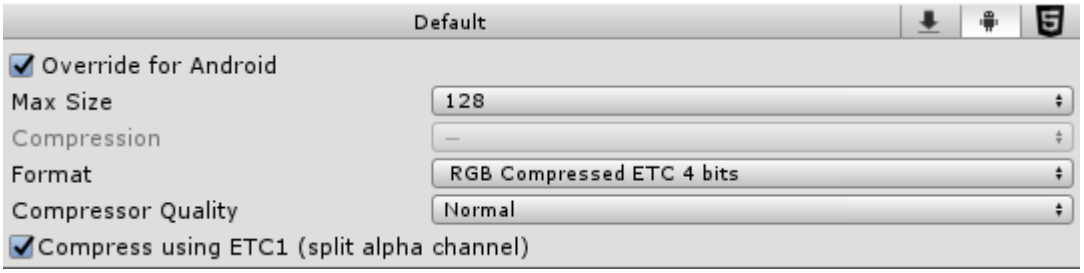
こういう状況のようです。たしかにここから 6 年たってますので状況は結構変わっていると思いますが、6 年間くらい同じスマホ使ってる人もいるわけですし、無視はできないと思います。なお、現在のこういう状況の分かりやすい表が見つからなかったので、知ってる人は教えてください。ちなみに僕はスマホほとんどいじらないので良く分かってないです。

ちなみに「Unity で作ったアプリの画像が**一部の Android 端末で表示されない**件について」という記事が 2016 年 12 月に出るあたり、まだまだこの手の問題はなくなっていないものと思われます。
http://qiita.com/monry/items/0349a2705b884f94ba1a

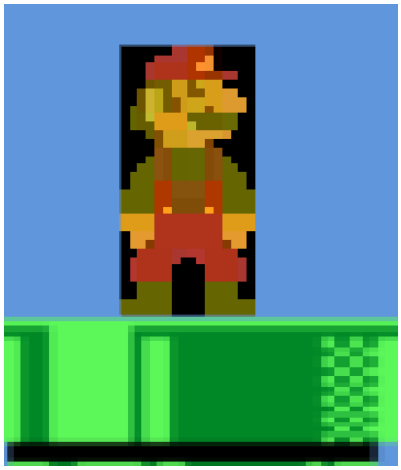
ということで無難にいきたい人は ETC1 を使用するのがいいでしょう。

で、何の呪いか、ETC1 で圧縮出力しようとすると Unity 落ちるん(そして立ち上がらなくなった)ですけど、呪いでしょうか(´；ω；`)

ともかく、ETC1 で出力してみます。
検証③：Android 用に ETC で出力



そうすると

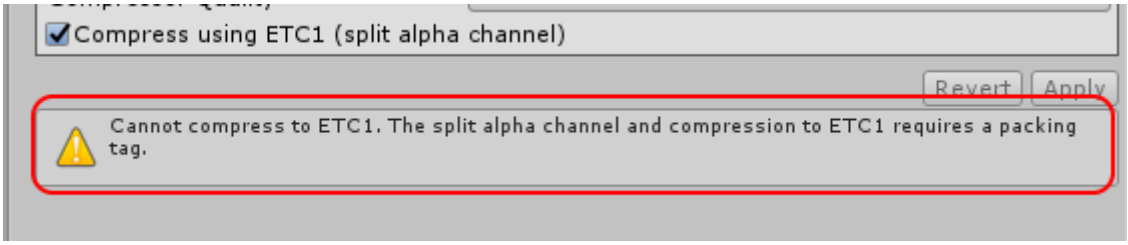


一応表示できてますね(ボタン残ってる)。EC1 にはアルファがないのでこの通りです(きちゃない)。

ところで

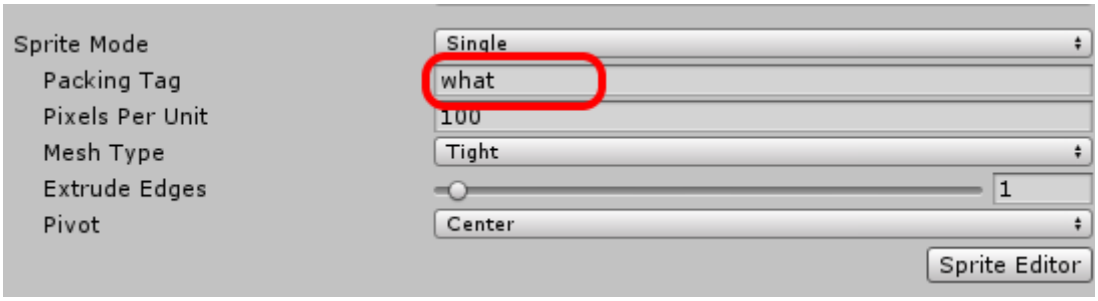
「Unity では、Android 用の（パッキングタグで指定され作成された）アトラスに含まれているアルファ付きのテクスチャに ETC1 を使用することができます。テクスチャの“Compress using ETC1” チェックボックスを設定することで選択できます。これにより Unity は、作成されるアトラスにアルファありとアルファ無しの 2 つのテクスチャに分割します。その後、レンダリングパイプラインの最終工程でその 2 つを合成します。」

とあるので、このチェックボックスを押してみますが、透明にはなりません。

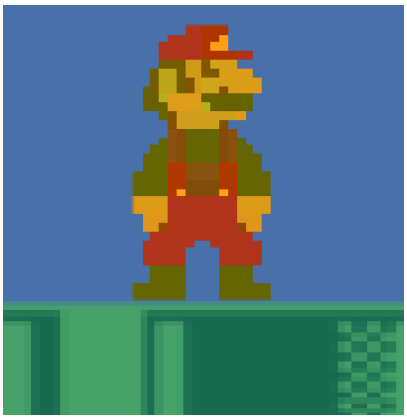


よく見るとこんなエラーメッセージが。確かに説明にあるように、パッキングタグが必要とあります。

試しにパッキンタグを適当に入れてみましょう。

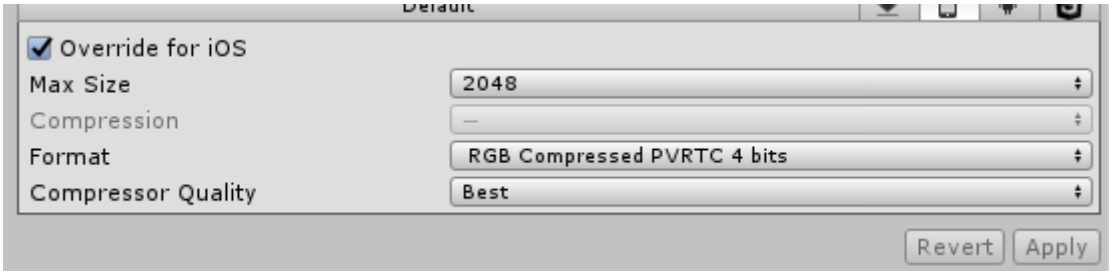


そうすると、汚いですが、きちんと抜けました。

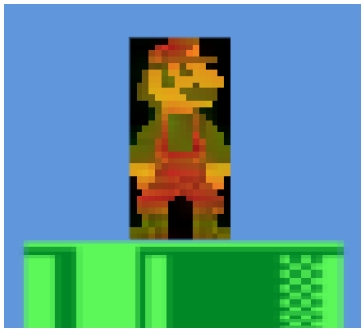


あとは iOS ですが、こいつらに関しては、PVRTC が使えるので、ETC ほど悩むことはないのではないでしょうか。

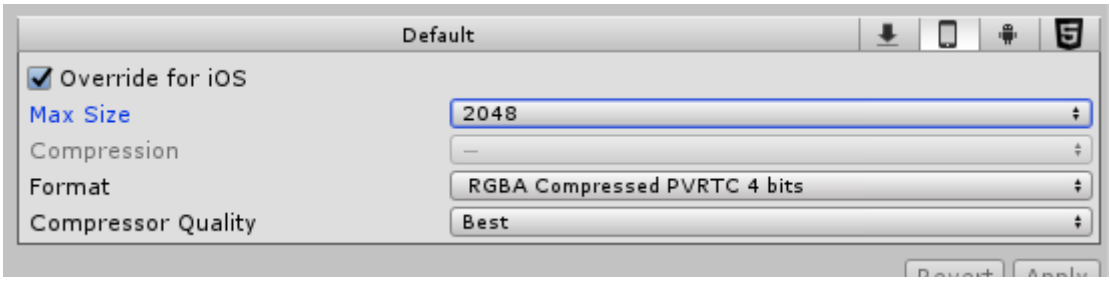
検証④：iOS 用に PVRTC で出力
まず、アルファなし版。



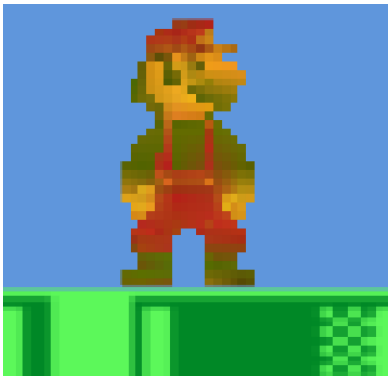
これで出力すると…にょわー！！！



なんじゃあこりゃあ！！
いや、落ち着け、気をしっかり持つんだ。アルファ付きだったらマシかもしれない。



ハァ…ハァ…どうよ！！



こんなのってないよ！！ひどすぎるよ！！！

結論

というわけで、ここまでの結論

「サイズの小さいドット絵に対して今しばらくは圧縮テクスチャを使用すべきではない」
これは確かにそうですが、大きな背景や写真系テクスチャには有効なので**圧縮テクスチャの存在**とメリットデメリットは是非覚えておきましょう。

さて、それではドット絵のサイズ改善はあきらめなければならないのか？後編へ続く！！