工程の手戻りを最小限に
2Dエンジン活用における傾向と対策

小高 輝真(株式会社ウェブテクノロジ)
東田 弘樹(フリーランス・プログラマ)

CEDEC 2014 で行った講演のスライドです。
http://cedec.cesa.or.jp/2014/session/ENG/6589.html

このファイルは http://cedil.cesa.or.jp/ および
http://www.webtech.co.jp/blog/game-develop/7179/ で公開しています。

Ver. 2014/9/19

OPTPiX

WebTechnology®

CEDEC 2014 で行った講演のスライドです。
http://cedec.cesa.or.jp/2014/session/ENG/6589.html
このファイルは http://cedil.cesa.or.jp/ および
http://www.webtech.co.jp/blog/game-develop/7179/ で公開しています。

工程の手戻りを最小限に

2Dエンジン活用における傾向と対策

セッションの内容

ゲームエンジンを活用して2Dゲームアプリを開発する際に、事前の調査をしっかり行うことで、手戻りを起こさないための情報を提供します。
UnityとCocos2d-xで動作する同じ仕様のゲーム風アプリを使い、各種端末で性能比較を行います。
Unity、Cocos2d-x上で2Dゲームを開発する際のTipsをご紹介します。

受講スキル

- 小規模チームで、スマホ向け2Dゲーム開発に関わるプログラマ
- 幅広いユーザーに向けたスマホ向け2Dゲームの開発者

受講者が得られる知見

- 2Dゲームエンジンで、性能見積をするための実際的な情報(テストアプリのソース付)
- 主に画像の取扱に関するTips

対象プラットフォーム Android, iOS

WebTechnology®

1

アジェンダ

第１部　2Dゲームエンジンとは

- 主なゲームエンジン、特にUnityとCocos2d-xについてのご紹介

第２部　スプライト描画負荷のベンチマーク

- Cocos2d-x v3とUnityの描画性能差
- Cocos2d-x v2とv3の描画性能差
- 機種による描画性能差
- バッチについて

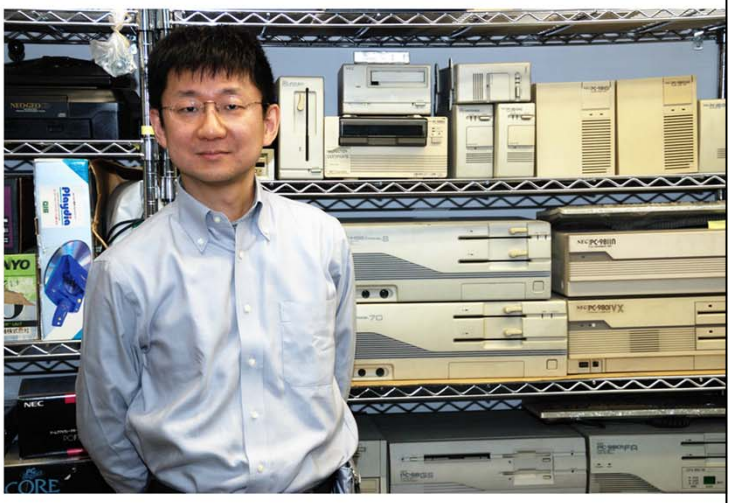
第３部

- 8つのTips

講演者紹介

小高　輝真

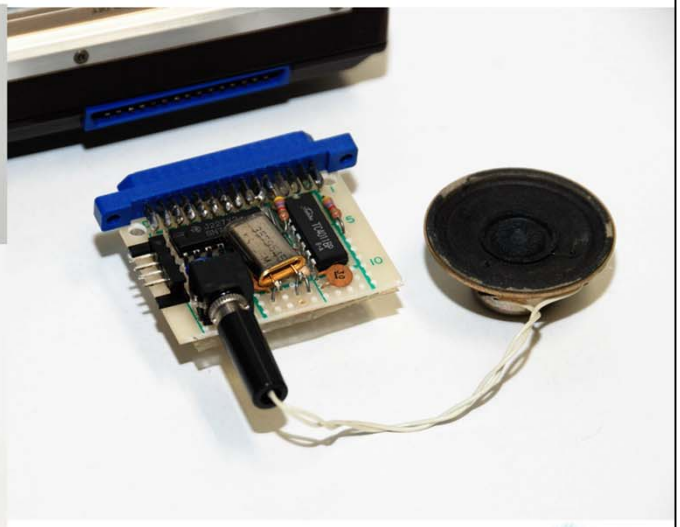
株式会社ウェブテクノロジ　代表取締役
元プログラマ
元テクニカルライタ
CEDEC運営委員
スポンサープログラムWGリーダー

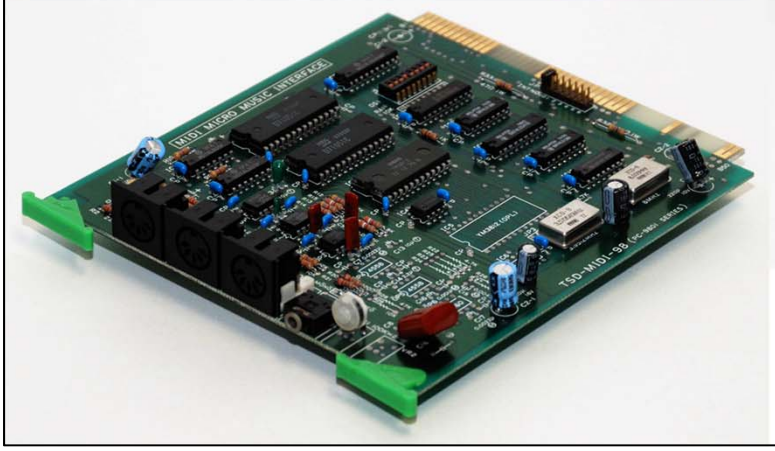


写真は、GameBusiness.jp
「OPTPiXはこうして生まれた！
ウェブテクノロジ設立物語（前編）」より
http://www.gamebusiness.jp/article.php?id=9350

http://www.gamebusiness.jp/article.php?id=9350

講演者紹介　小高　輝真





WebTechnology®

小高は、ゲームは作ったことはないのですが、近いところではサウンドのシーケンサ(自動演奏ソフト)を作ったことがあります。
高校生のときにPC-1500というポケコンを改造して、PSGを繋いで、ハンドアセンブルでシーケンサを作ったのが最初です。
大学生のときには、PC98用のFM音源ボードを設計して、そのシーケンサというかドライバを作りました。
ドライバは全部割り込みで動くようになっていて、一太郎を使っている裏で音楽を奏でることができました。

講演者紹介　小高　輝真


PC-9801
スーパーテクニック
小高輝真・清水和文・遠水祐　共著
アスキー出版局


アンドキュメンテイド
UNDOCUMENTED
9801/9821
1
インプレス

講演者紹介

東田 弘樹

フリーランス・プログラマ

ゲームボーイの時代頃よりゲームプログラマとして従事。
フリーランスになってからゲーム以外のプログラム開発案件にも関わりつつ
最近はゲーム開発を支援する裏方として、ツール開発などに従事しています。

第１部

2Dゲームエンジンとは

2Dゲームエンジンとは

このセッションでは、2Dゲームエンジンについて

「2Dゲームを作成するために使いやすいゲームエンジン」

と定義

3Dエンジンは除くが、Unityは現在2Dゲーム制作にも広く使われており、
Unityには2D機能も提供されているため、このセッションの対象とする

2Dゲームエンジンのメリット

- マルチプラットフォーム対応
  ゲームエンジンがプラットフォームごとの差異を吸収してくれるので、
  １ソースでマルチプラットフォームの開発が行える

- 高速化
  ハードウェアの性能を出すにはOpenGLの使用が必須
  しかし、OpenGLで2Dの表現をするのは案外面倒
  ゲームエンジンが肩代わりしてくれる

- 時間節約
  2Dであれば、比較的定型の処理が多い
  ゲームエンジンが必要な機能を備えている

代表的な2Dゲームエンジン

Unity

Cocos2d-x

Starling

Sprite Kit

Corona SDK

Marmalade SDK

…

ここからは、代表的な２Dゲームエンジンを見ていきます。
ここでご紹介するもの以外に、多数のゲームエンジンが存在していますが、
現時点で採用例が多いのはUnityと、最近注目されてきたCocos2d-xです。

Unity

主な特徴

- プロトタイプの開発が早い。独自エディタを使っての開発が主
- 2Dに3Dの表現を組み合わせることができる
- 自分が作りたいゲームにあったAsset(サードパーティ製のプラグインのようなもの)が見つかれば、非常に生産性が高くなる
- 対応プラットフォーム: iOS, Android, Windows, Mac, Linux, Webブラウザ, PlayStation® 4/3/Vita, Xbox ONE/360, Wii U™
- ライセンス等: Unity Free 無料(制限あり)、 Unity Pro 162,000円
- 開発言語: C#, JavaScript, Boo
- IDE: Unity エディタ, Mono Develop 4（替わりにVisualStudioも使用可 ※UnityVS）
- 物理エンジン: あり
- パーティクル: あり
- http://japan.unity3d.com/

http://japan.unity3d.com/
独自のUnityエディタというのが用意されていて、これがさながらゲームツクールのような機能を備えています。
プログラムを極力書かなくても済むように、というような設計思想が基本としてあるようです。

特徴的なのが、アセットストアです。AppStoreやGooglePlayのリソース版のようなもので、有償、無償を含め、テクスチャデータやモデルデータ、プログラムのライブラリみたいなものまで、
様々なものがここで公開されていて開発にこれらを流用するができます。Unityエディタからアセットストアを直接呼び出せて、アセットの組み込みもすぐ行えます。ただ、ライセンス形態も様々ですので、本番に使いたいものは事前にしっかりと確認してください。

Booはちょっとなじみのない言語ですが、Pythonに似た構文の言語です。
開発環境としては、Unityエディタが中心になりますが、Unityエディタ自体はコードの編集機能を備えていません。そこでコード編集はMonoDevelopという別のIDEを使うことになります。プログラマは、UnityエディタとMonoDevelopを行ったり来たりすることになりますね。

Windows環境で、VisualStudioをお使いの方は、MonoDevelopの替わりにVisualStudioを使えるようにするプラグインがマイクロソフトから無償で配布されています。VisualStudioに慣れていらっしゃる方は、こちらの方が扱いやすくなると思います。

Cocos2d-x

主な特徴

- 2Dゲームエンジン
  C++での開発に馴れたコンシューマ・ゲーム開発経験者には使いやすい
- 対応プラットフォーム: iOS 5.0以降, Android 2.3以降, Windows 7以降, OS X 10.6以降, Windows Phone 8
- ライセンス等: 無償, オープンソース(MITライセンス)
- 開発言語: C++(C++11), Lua/JavaScriptを使っての開発も可
- 使用可能IDE: Visual Studio 2012以降, Xcode 4.6以降
- 物理エンジン: あり(Chipmunk2D, Box2D)
- パーティクル: あり
- http://www.cocos2d-x.org/

上記はCocos2d-x v3の情報です

http://www.cocos2d-x.org/

開発は、中国のチュコン・テクノロジーズが中心になって行われています。

無償公開されていて、ソースコードはGitHubで公開されています。開発はかなり活発に行われています。

シンプルで、分かりやすい設計になっていますので、ゲーム開発経験者であれば扱いやすいのではないかと思います。

開発言語はC++で、v3からC++11の機能を積極的に活用してコードが書かれています。

エンジンのコードはC++で書かれていますが、それをLuaやJavaScriptから呼び出すことのできるインターフェイスも用意されていています。

ですので、ところによりC++とLuaを使い分けて開発の効率化を図ることも可能です。

Starling

主な特徴

- AIR/Flash上で動く
- 対応プラットフォーム: AIR/Flashが動くもの(iOS, Android, Windows, Mac, Linux, Webブラウザ)
- ライセンス等: 無償利用可, オープンソース
- 開発言語: ActionScript 3
- 使用可能IDE :Flash Builder 4.7, FlashDevelop
- 日本語情報: かなり少ない
- 物理エンジン: 標準ではなし
- パーティクル: あり
- http://gamua.com/starling/
- GPUで描画（Stage3Dを使用）
- AIRというと別途ランタイムのインストールが必要そうだが、iOS, Androidはアプリの中にランタイムが組み込まれる
- 気になること…Adobe がAIR/Flashをいつまでサポートするのか

http://gamua.com/starling/

AIRというと、ユーザーに別のラインタイムをインストールさせなきゃならないんじゃないかと思われがちですが、現在は、iOS・Androidともに、アプリのパッケージの中にラインタイムも一緒に組み込むことができます。

ですのでユーザーから見ると、普通のアプリと変わらず扱うことができます。

Sprite Kit

主な特徴

- iOS, Mac OS X標準の2Dグラフィック・フレームワーク
- 対応プラットフォーム: iOS 7.0以降, OS X 10.9以降
- ライセンス等: 無償利用可, ソース非公開
- 開発言語: Objective-C(C++, C), Swift
- 使用可能IDE：Xcode
- 日本語情報: 若干。日本語書籍は1冊(2014/8現在)
- 物理エンジン: あり
- パーティクル: あり
- https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html
- 気になること… Androidとのマルチプラットフォーム開発ができない

https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html

冒頭で、マルチプラットフォーム対応が利点だ、といっておきながらこれは例外です。

他の２Dゲームエンジンと遜色なく、プラットフォームベンダー自ら提供しているものなので、ここでとりあげました。

iOS7に搭載されている今のバージョンでも、cocosなどと同様のことが可能ですが、次のiOS ８では更に機能強化が図られていてなかなか興味深いです。

例えば、ライトというものが、導入されます。ゲームエディタなのか、シーンエ

ディタなのか詳細は分かりませんが、そのようなエディタがXcodeの方に搭載されてくるようです。

Corona SDK

Corona SDK

主な特徴

- クラウドコンパイルのため環境構築がラク
- ライトゲームの作成に向いている
- 対応プラットフォーム: iOS 5.1以降, Android 2.2以降, Windows Phone 8(予定)
- ライセンス等: US$16.00/月～
- 開発言語: Lua ※ライセンス次第でネイティブ言語も合わせて使用可
- 使用可能IDE：Corona Editor（Sublime Text用プラグイン）
- 日本語情報: かなり少ない
- 物理エンジン: あり(Box2D)
- パーティクル: あり
- http://coronalabs.com/
- 気になること
  - クラウドコンパイルのため、サービス継続性のリスク
  - ソースをクラウドにアップロードする必要があるため、それが許容できるかどうか
  - Enterpriseライセンスではオフラインビルドも可能

http://coronalabs.com/

特徴は、基本的にクラウドサービスというところ。ソースコード自体の編集はローカルで行いますが、コンパイルなどはサーバーを通して行われます。

Corona SDKを入れるだけで、あとはAndroidのSDKを入れたりとか、そういう面倒な手間がなくすぐに開発がはじめられるのが利点です。

ただ、クラウドサービスという点が許容できるのかが判断の分かれ目になりそうです。

Marmalade SDK

主な特徴

- 2D/3Dの開発が可能
- 対応プラットフォーム: iOS 5.0以降, Android 2.2以降, Windows Phone 8, BlackBerry, TIZEN, Windows, Mac（Win/MacはUS$499/月ライセンスから可）
- ライセンス等: フリー版あり（サポート限定）, US$15/月～
- 開発言語: C++
- 使用可能IDE: Visual Studio, Xcode
- 日本語情報: かなり少ない
- https://www.madewithmarmalade.com/
- ミドルウェアとの組み合わせで、LuaもしくはObjective-Cを使った開発も可
- Objective-Cを使った組み合わせでは、iOS向けに作ったプロジェクトをAndroidに移植するような用途にも使える

https://www.madewithmarmalade.com/

一時期ドコモさんが採用するんじゃないか、という噂がありましたが、正式発表は今もありません。

２Dだけでなく、3Dも扱うことができます。

ミドルウェアの組み合わせでLuaやObjective-Cを使った開発もできます。

少し興味深いのが、iOSネイティブで作られたプロジェクトをAndroidに移植するためのミドルウェアを備えているところです。

代表的な2Dゲームエンジン

Unity
Cocos2d-x

Starling

Sprite Kit

Corona SDK

Marmalade SDK

…

Unity

- 最新版はv4.5.3、v4.6がオープンβ中。v5はまもなく登場！
- 情報量とサポート
  - Unityテクノロジーズジャパン合同会社のサポートが受けられる
  - 日本語書籍は20冊以上（ただ3D機能についてが中心）
  - ネット上にも日本語情報が溢れている

Unity

- v4.6で、UIエディタが搭載
- Unity Cloud Build（β）
  - Git/svnのリポジトリを登録しておくと、自動的にビルド、配信(テスト向け)まで行ってくれる
  - Unity Pro限定

今までは、標準搭載されているGUIの部品はスクリプトからのみしか扱えなかったりと、ちょっと使い勝手が悪かったのですが、これがUIエディタが搭載されることで解消されそうです。
Unity Cloud Buildは、β扱いですが、ビルド、いわゆるパッケージ化と、テストアプリの配信をやってくれるクラウドサービスが準備されている模様です。ただし、Unity Pro限定の機能。
通常はUnityエディタを使って各プラットフォーム向けにパッケージ化しなければならないので、これが回数が増えてくると結構手間なので、それを自動化できるのは嬉しいです。

Cocos2d-x

- 最新版はv3.2、v2はv2.2.5
- 情報量
  - 日本語書籍は6冊程度。ただし、v2のものがほとんど
  - ネット上の日本語情報はかなり増えてきている

バージョン3は、アルファ版が昨年から公開されていましたが、正式版になったのは今年の4月です。
それからはハイペースで3.1、3.2とバージョンが上がってきています。

Cocos2d-x

v3になって何が変わったか？

- Objective-C的パターンからC++11へ
- 新しいレンダラーの導入
  - 自動バッチング、自動カリング、グローバルZオーダー導入

- ラベルの改善
- イベント処理の改善
  - タッチ処理がより扱いやすく
- 3Dモデルが表示可能に（v3.1から）

https://github.com/cocos2d/cocos2d-x/blob/cocos2d-x-3.0/docs/RELEASE_NOTES.md#user-content-highlights-of-v30

https://github.com/cocos2d/cocos2d-x/blob/cocos2d-x-

3.0/docs/RELEASE_NOTES.md#user-content-highlights-of-v30

Cocos2d-x

気になるところ…

サウンドの機能が弱い、v3でも変わらず

- 最低限のAPIしか用意されていない
- 音ゲーのようなもをお考えの方は事前確認を
- ミドルウェアに頼る手も　CRI ADX2 for Smartphone

サウンドの機能が弱いという弱点があります。一応、BGMと効果音を鳴らすAPIは備わっていますが、必要最小限の機能。
例えばBGMをフェード・アウトさせた後に止めるとか、そういう処理がありません。
後々になって、ええって？ことがないように、サウンドに力を入れたい方は事前に機能をご確認ください。

スプライト描画負荷のベンチマーク

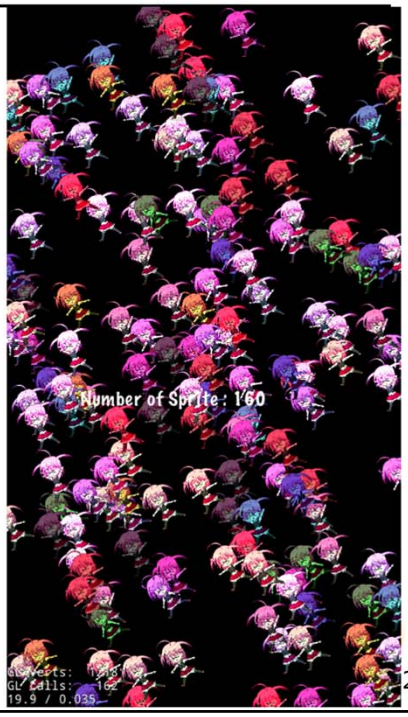
ここで使用したテストアプリのソースファイルは、
http://www.webtech.co.jp/blog/game-develop/7179/
で公開しています。

ここで使用したテストアプリのソースファイルは、

http://www.webtech.co.jp/blog/game-develop/7179/ で公開しています。

基本的なスプライト描画負荷テストアプリ

- 画面上にスプライトを増やしていき、その描画負荷を計測
- スプライトはアニメーションなし
- 画面タッチで描画スプライト数が増加
- 使用テクスチャの枚数により２パターンを検証
  - 1枚のテクスチャのみで描画
  - 16枚のテクスチャを順番に切り替えて描画
- テクスチャは、96x96ピクセル、RGBA4444(.pvrに格納)
- アプリ名: 1TexSpr, 16TexSpr
- Unity v4.5.2, Cocos2d-x v3.2, v2.2.5(Batchあり・なし)で検証



OPTPiX

基本的なスプライト描画負荷テストアプリ

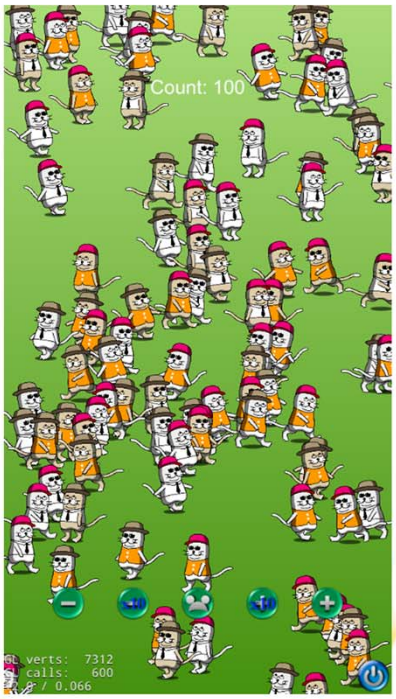
96 x 96
(RGBA4444) x1枚

96 x 96
(RGBA4444) x16枚

使っているテクスチャです。

ゲーム風アプリ

- 画面上に多関節アニメするキャラクターを増やしていき、
  その描画負荷を計測
- キャラクターは11パーツ（スプライト）で構成
- ボタンでキャラクターを増減可能
- サウンドあり(BGMをMP3で再生)
- 使用テクスチャの構成により３パターンで検証
  - １枚のみ
  - 白ネコ、黒ネコのテクスチャ２枚
  - 白ネコ、黒ネコ、洋服、帽子、めがね、
    などテクスチャ８枚
- テクスチャフォーマットは、RGBA4444(.pvrに格納)
- アプリ名: Multi
- Unity v4.5.2, Cocos2d-x v3.2, v2.2.5(Batchあり・なし)で検証
  ※テクスチャ８枚構成のv2 Batchありは未計測



OPTPiX

26

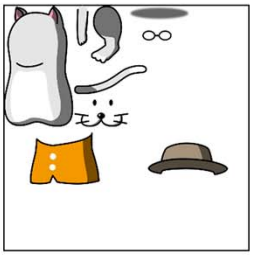
二つ目のテストアプリは、実際のゲームを意識して、複数テクスチャでアニメーションするキャラクタを表示する、スプライト描画負荷テストアプリです。

ゲームと付けるには、はばかられるのでゲーム風としています。

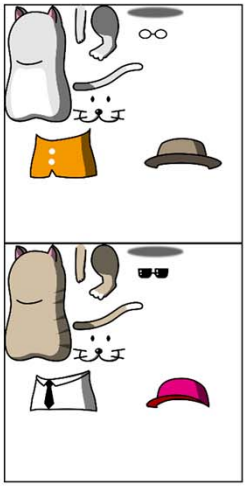
この環境では音が聞こえませんが、BGMも再生しています。

なお、このアニメーションはSpriteStudio で作っています。

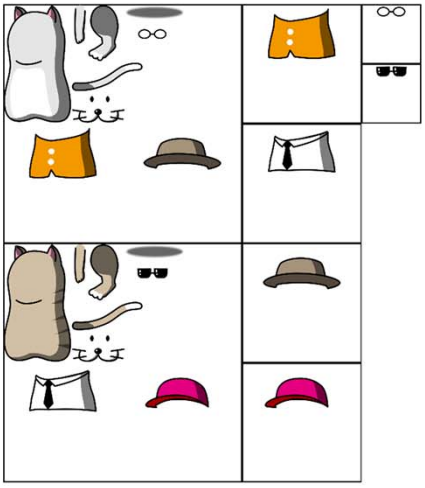
ゲーム風アプリ



1024 x 1024
(RGBA4444) x1枚



1024 x 1024
(RGBA4444) x2枚



8枚　※装備品は単体画像側を使用
1024 x2枚, 512 x4枚, 256 x2枚

27

使っているテクスチャです。

スプライト描画性能比較　検証端末

	端末名	発売日	CPU	画面解像度	備考
1	Xperia arc (docomo SO-01C)	2011/03	Qualcomm Snapdragon MSM8255 1GHz Qualcomm Adreno 205 GPU	480×854	Android 2.3.4
2	Nexus 5 (LG-D821)	2013/11	Qualcomm Snapdragon MSM8974 2.26GHz(クアッドコア) Qualcomm Adreno 330 GPU	1920×1080	Android 4.4.0
3	iPhone 4	2010/6	Cortex-A8(Single) 800MHz PowerVR SGX535	960x640	
4	iPhone 4S	2011/10	Coretex-A9(Dual) 800MHz PowerVR SGX543MP2(Dual)	960x640	
5	iPhone 5	2012/9	ARMv7s(Dual) 1.3GHz PowerVR SGX543MP3(Triple)	1136x640	

古いほうは、現在でも実際にユーザーが使っていそうな下限の機種を選んだつもりです。

スプライト描画性能比較

比較内容

1. Cocos2d-x v3とUnityの描画性能差(2パターン)

2. Cocos2d-x v2とv3の描画性能差(3パターン)

3. 機種による描画性能差(4パターン)

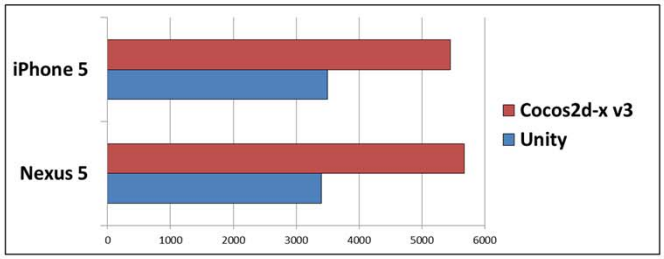
2D描画性能を比較するためにいくつか要素の異なるスプライトを１フレームにいくつ描画できるかを計測
計測方法は、描画するスプライトを徐々に増やしていき、30FPS を維持できる最大数を求めた

スプライト描画性能比較(Part 1)

Cocos2d-x v3とUnityの描画性能差(1)

- 使用アプリ: 1TexSpr (基本的なスプライト描画負荷テストアプリ)
- テストの内容：
  - 1枚のテクスチャ(96x96、RGBA4444)から、スプライトを多数描画するのみ
  - 非常にプリミティブな機能の計測
- テスト結果：

	Cocos2d-x v3	Unity
iPhone 5	5450個	3500個
Nexus 5	5670個	3400個



- 考察：
  - **iPhone 5では、Cocos2d-xがUnityの約1.6倍、Nexus 5では約1.7倍のパフォーマンス**
  - iPhone 5とNexus 5はほぼ互角(UnityはiPhone 5が3%優位、Cocos2d-x v3はNexus 5が4%優位)

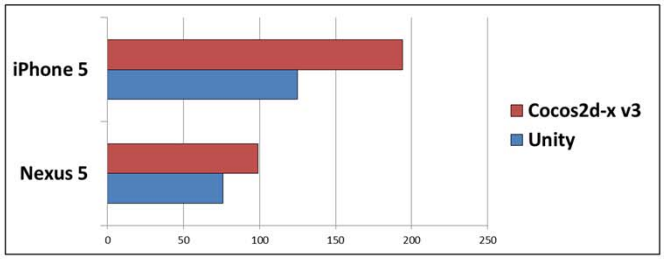
巷でよく行われているベンチマークは、このようなものだと思います。
Cocosのほうが、Unity の約1.6倍程度の描画性能という結果が出ています。
iPhone 5とNexus 5はほぼ同じ結果。

スプライト描画性能比較(Part 1)

Cocos2d-x v3とUnityの描画性能差(2)

- 使用アプリ: Multi (ゲーム風アプリ)
- テストの内容：
  - 2枚のテクスチャ(1024x1024、RGBA4444)から、複数スプライトで構成されたキャラクタを多数描画
  - UIボタン、サウンドあり
  - 実際のゲームアプリに近い構成
- テスト結果：

	Cocos2d-x v3	Unity
iPhone 5	194個	125個
Nexus 5	99個	76個



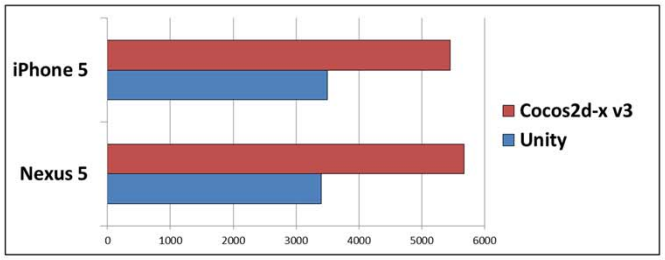
- 考察：
  - **iPhone 5では、Cocos2d-xが Unityの約1.6倍、Nexus 5では約1.3倍のパフォーマンス**
  - この条件ではiPhone 5がかなり優位(Unityで約1.6倍、Cocos2d-x v3で約2倍のパフォーマンス)

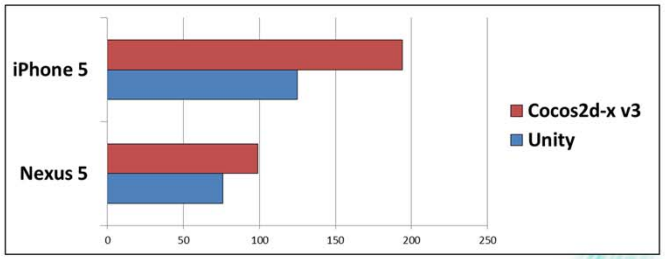
ゲーム風アプリのほうで測定した結果です。
こちらのほうが、より実際のゲームアプリに近い結果を示していると言えるでしょう。
iPhone 5が非常によい結果を示していて、逆にNexus 5のほうがかなり低い結果が出ています。

スプライト描画性能比較(Part 1)

Cocos2d-x v3とUnityの描画性能差

- 特定のプラットフォーム、特定の機種だけでパフォーマンスチェックしていると、手戻り発生の可能性あり





OPTPiX

WebTechnology®

32

計測の条件によって、同じアプリでもこれだけの違いが出ることがわかりました。

実際、iPhone で開発していて、あとからAndroidにもっていくとパフォーマンスが出なくて苦労する、という話をお客さんから聞いたことがあります。

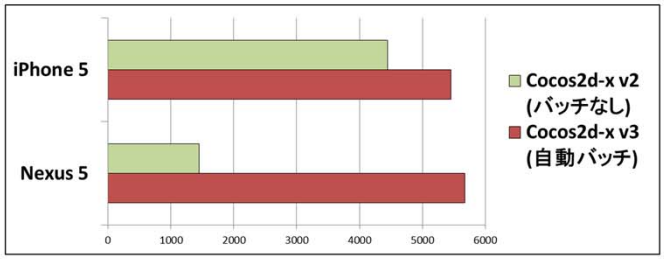
特定のプラットフォーム、特定の機種 だけでパフォーマンスチェックしていると、手戻り発生の可能性が高まります。

スプライト描画性能比較(Part 2)

Cocos2d-x v2とv3の描画性能差(1)

- 使用アプリ: 1TexSpr (基本的なスプライト描画負荷テストアプリ)
- テストの内容：
  - 1枚のテクスチャ(96x96、RGBA4444)から、スプライトを多数描画するのみ
  - 非常にプリミティブな機能の計測
- テスト結果：

	Cocos2d-x v2 (バッチなし)	Cocos2d-x v3 (自動バッチ)
iPhone 5	4444個	5450個
Nexus 5	1450個	5670個



- 考察：
  - **Nexus 5では、v3がv2(バッチなし)の約3.9倍のパフォーマンス。バッチの効果は絶大**
  - **iPhone 5では、v3がv2(バッチなし)の約1.2倍でしかない。というより、バッチなしでも高速**

Cocos v2とv3の描画性能差の計測結果です。
基本的なスプライト描画負荷テストアプリを使って、v2ではバッチなし、v3は自動バッチという条件。

Nexus 5では、Cocos v3のほうが圧倒的にパフォーマンスが高いという結果。
これは、バッチの効果と考えられます。

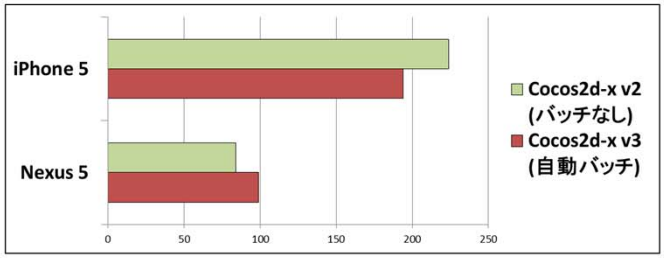
ところが、iPhone 5では、v2のバッチなしでもかなり高いパフォーマンスを出しています。
これは、Power VRのタイルベース遅延レンダリング（TBDR）が効いているからだと考えられます。

スプライト描画性能比較(Part 2)

Cocos2d-x v2とv3の描画性能差(2)

- 使用アプリ: Multi (ゲーム風アプリ)
- テストの内容：
  - 2枚のテクスチャ(1024x1024、RGBA4444)から、複数スプライトで構成されたキャラクタを多数描画
  - UIボタン、サウンドあり
  - 実際のゲームアプリに近い構成
- テスト結果：

	Cocos2d-x v2 (バッチなし)	Cocos2d-x v3 (自動バッチ)
iPhone 5	224個	194個
Nexus 5	84個	99個



- 考察：
  - **iPhone では、v2(バッチなし)がv3の+15%、Nexus 5 では逆にv3がv2(バッチなし)の+18%のパフォーマンス**
  - **実際のゲームアプリに近い構成では、v3の優位が揺らいでいる**

Cocos v2とv3の描画性能差の計測の続き、ゲーム風アプリの測定結果です。
前回と同様、v2はバッチなし、v3は自動バッチという条件です。

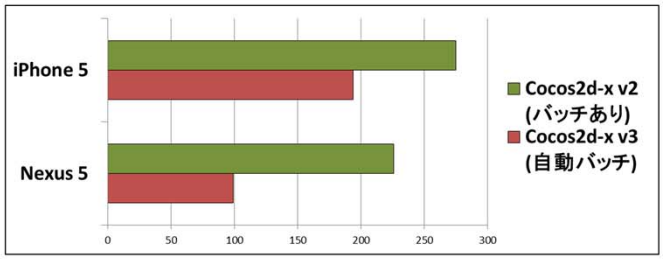
iPhone 5はバッチなしでも充分な好成績でしたが、今度はバッチなしが自動バッチを上回るという結果に。
Nexus 5のほうも、さきほどよりは差が小さい。

スプライト描画性能比較(Part 2)

Cocos2d-x v2とv3の描画性能差(3)

- 使用アプリ: Multi (ゲーム風アプリ)
- テストの内容：
  - 2枚のテクスチャ(1024x1024、RGBA4444)から、複数スプライトで構成されたキャラクタを多数描画
  - UIボタン、サウンドあり
  - 実際のゲームアプリに近い構成
- テスト結果：

	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)
iPhone 5	275個	194個
Nexus 5	226個	99個



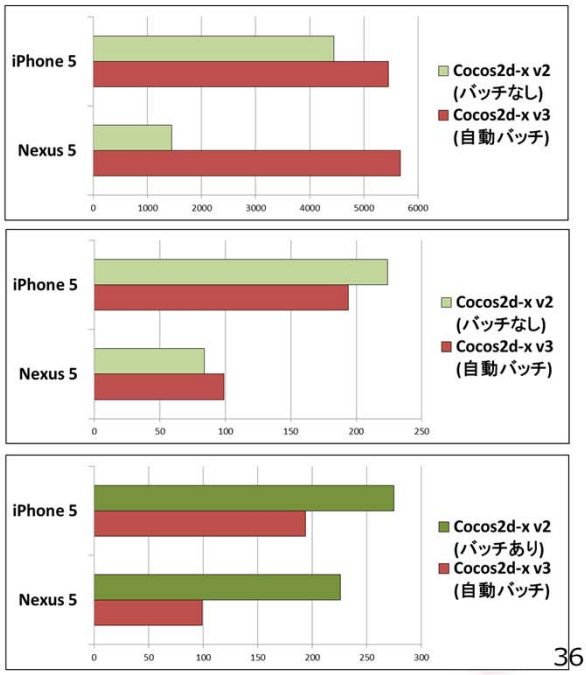
- 考察：
  - **iPhone では、v2(バッチあり)がv3の約1.4倍、Nexus 5 では約2.3倍のパフォーマンス**
  - **Cocs2d-x v2で、手動で理想的なバッチを組めれば、v3よりも高パフォーマンスを出せる**
  - **ただし、このアプリのv2 バッチありでは必ずバッチが効くという理想状態を計測しているので、注意**

Cocos v2とv3の描画性能差の計測の続きです。ゲーム風アプリのほうで、測定。
前回と変わって、v2はバッチあり、v3は自動バッチという条件。

スプライト描画性能比較(Part 2)

Cocos2d-x v2とv3の描画性能差

- Cocos2d-x v3で導入された自動バッチは、複雑な描画の場合は、必ずしもバッチなしv2より高性能になるとは限らない



OPTPiX

36

Cocos v2とv3の描画性能差を見てきました。

右の３つのグラフは、それぞれスケールが違うので、ひとつのグラフの中の比率だけを比べて見てください。

結論としては、iPhone 5では、GPUであるPowerVRの特性で、バッチなしでもかなり高い性能が出ることがわかりました。

Nexus 5では、手動で最適なバッチを組めばそれなりのパフォーマンスが出ますが、v3の自動バッチではあまり高い性能は出しにくいようです。

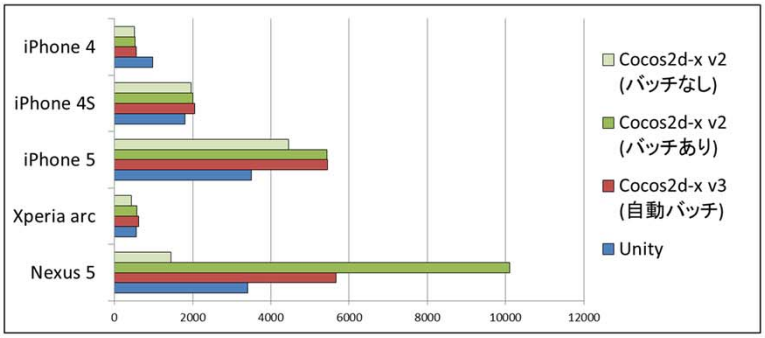
バッチを組みやすいアプリでは、v2で手動バッチが最も良いパフォーマンスを出せることがわかりました。
ただ、ゲームの種類によっては手動でバッチを組むのは結構大変ですので、v2手動バッチのこの結果は、あくまでもピーク性能と考えた方が良さそうです。

スプライト描画性能比較(Part 3)

機種による描画性能差

- 使用アプリ: 1TexSpr (基本的なスプライト描画負荷テストアプリ; テクスチャ1枚)
- テスト結果：

	Cocos2d-x v2 (バッチなし)	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)	Unity
iPhone 4	510個	520個	560個	980個
iPhone 4S	1960個	2000個	2050個	1800個
iPhone 5	4444個	5430個	5450個	3500個
Xperia arc	440個	580個	620個	560個
Nexus 5	1450個	10100個	5670個	3400個



3番目の計測では、機種による描画性能差を見てみます。
最初は、基本的なスプライト描画負荷テストアプリの結果から。

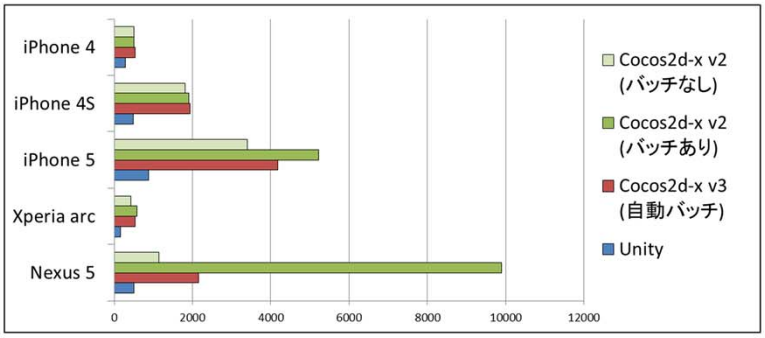
この数年のスマホの進化が激しいことが見て取れます。
古い世代の機種では、CocosとUnityの描画性能差が小さいことがわかります。
特に、iPhone 4では、Unityが一番良い結果を出しているのは興味深いところ。
気になるのは、Nexus 5がCocos v2バッチありの条件だけ突出して高い点。

スプライト描画性能比較(Part 3)

機種による描画性能差

- 使用アプリ: 16TexSpr (基本的なスプライト描画負荷テストアプリ; テクスチャ16枚)
- テスト結果：

	Cocos2d-x v2 (バッチなし)	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)	Unity
iPhone 4	510個	505個	530個	280個
iPhone 4S	1820個	1900個	1930個	490個
iPhone 5	3400個	5220個	4180個	880個
Xperia arc	425個	580個	540個	165個
Nexus 5	1150個	9890個	2150個	500個



次も、基本的なスプライト描画負荷テストアプリ。
16枚のテクスチャを使うというちょっと極端な場合を測定してみました。

前の結果と比べて目立つのは、Unityのパフォーマンスが大きく落ち込んでいること。
とはいえ、これは極端な条件での測定なので、実際にこのことが問題になることは少ないでしょう。

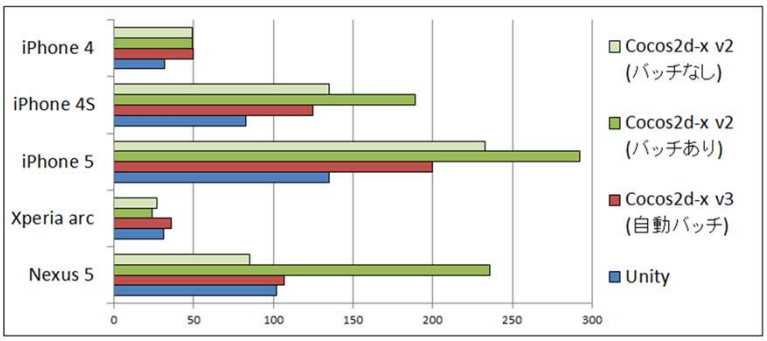
やはりここでもNexus5のCocos v2バッチありが飛び抜けています。

スプライト描画性能比較(Part 3)

機種による描画性能差

- 使用アプリ: Multi (ゲーム風アプリ; **テクスチャ1枚**)
- テスト結果：

	Cocos2d-x v2 (バッチなし)	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)	Unity
iPhone 4	49個	49個	50個	32個
iPhone 4S	135個	189個	125個	83個
iPhone 5	233個	292個	200個	135個
Xperia arc	27個	24個	36個	31個
Nexus 5	85個	236個	107個	102個



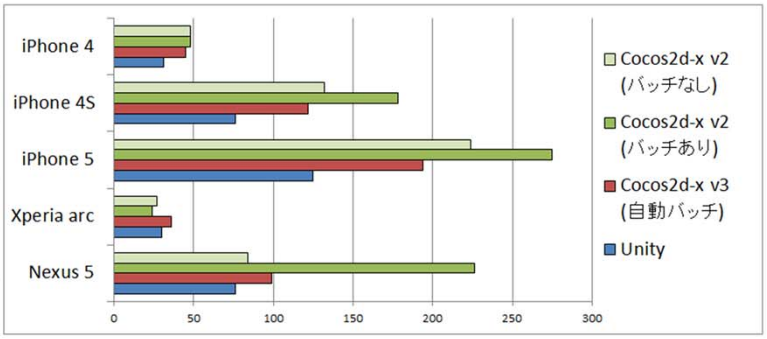
機種による描画性能差の計測の続きです。
ゲーム風アプリで測定。
テクスチャ1枚の場合と、テクスチャ2枚の場合を測定。こちらは1枚の場合。

スプライト描画性能比較(Part 3)

機種による描画性能差

- 使用アプリ: Multi (ゲーム風アプリ; **テクスチャ2枚**)
- テスト結果：

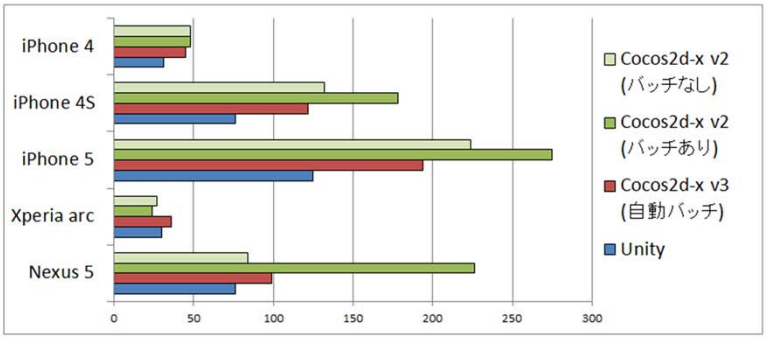
	Cocos2d-x v2 (バッチなし)	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)	Unity
iPhone 4	48個	48個	45個	31個
iPhone 4S	132個	178個	122個	76個
iPhone 5	224個	275個	194個	125個
Xperia arc	27個	24個	36個	30個
Nexus 5	84個	226個	99個	76個



こちらは、テクスチャ2枚の場合。
テクスチャ1枚の場合と、大きく傾向は変わりません。
若干気になるところとしては、Nexus 5のUnityではテクスチャが増えたときのペナルティが大きい傾向があること。

スプライト描画性能比較(Part 3)

- スマホ世代間の性能差は結構大きい
- Cocos2d-xに比べると、Unityはテクスチャの枚数がパフォーマンスにより影響を与える



OPTPiX

WebTechnology®

41

ここまで見てきてわかることとしては、スマホ世代間の性能差は結構大きいということです。
特にここ数年で性能がどんどん上がってきているので、３～４年前の機種と比較すると結構差が付いていることが分かります。
Unityは、扱うテクスチャが増えてくるとCocosに比べてよりパフォーマンスが落ちやすくなる傾向にある印象です。

全体を通して気になるのは、Nexus 5のCocos v2 バッチありの結果が突出しているところです。
残念ながらこの原因について、今回は究明できませんでした。
このように、ベンチマークテストをすると、たまにこういう突出した値が計測されることがあります。
こういうピーク性能を全体的な性能と誤解してしまうと、大きな手戻りの原因になってしまいます。
今回のように、複数の条件、複数の機種でテストして、こういうピーク値に惑わされないように気をつけるのが肝心です。

スプライト描画性能比較

手戻りしないために

- スマホの世代による性能差はわりと大きい
- スマホの性能は、今後もしばらく向上が予想されるので、リリース時点の状況を予測して、どこまで古い機種をサポートするのかを良く検討した方が良い
- Cocos2d-xのほうがパフォーマンスが安定している。Unityは扱うテクスチャの枚数が増えるとパフォーマンスがより落ちる傾向にある、ただ本来3Dエンジンであることを加味すれば十分健闘している
- パフォーマンスが決定的に重要な場合はCocos2d-xを、そうでなければ開発のしやすさ優先でエンジンを選ぶのが良さそう
- iPhoneで先行して開発していると、Androidで性能が出ない場合がある。
  定期的に、ターゲット範囲の底辺の機種でパフォーマンスを確認しておく

ドローコールとは？

- GPU に対して描画開始を命令すること
- この呼び出し回数が多ければ、多いほど描画負荷が上がる
- バッチングを活用することで、ドローコールを減らすことができる

OPTPiX WebTechnology® 43

先程、バッチあり、なしでパフォーマンスが大きく変化することを見てきましたが、それはドローコールが影響するためです。

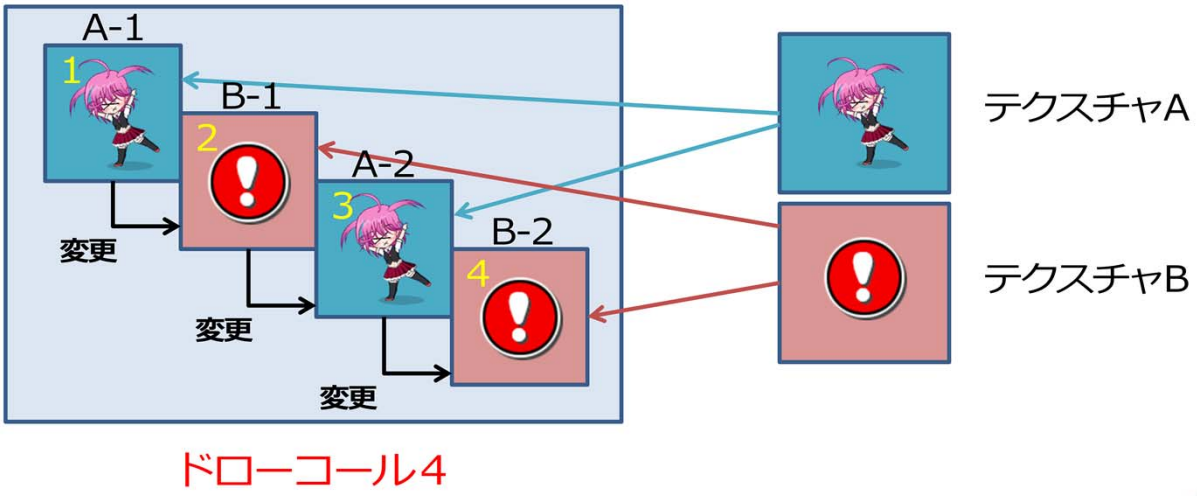
ドローコールとは簡単に言うと GPU に対し、描画開始の命令を出すことです。

一般的に、この回数が多ければ、多いほど描画のための負荷が高くなります。

このため世間では、ドローコールを少なく抑えるための努力が行われます。

ドローコールが増える要因

GPUステートの変更(テクスチャやシェーダの変更)によって増える



ドローコール4

OPTPiX

WebTechnology®

44

ドローコールが増えると負荷が高くなると説明しましたが、正確に言うと、「ドローコールを増やす要因となる処理」もまた大きな負荷になります。

その主な要因が、GPU に対してステートチェンジを要求することです。

これは多くの場合、描画に使うテクスチャの変更、シェーダの変更等によって発生します。

この図は、テクスチャA、Bの２枚のテクスチャを交互に使って、４枚のスプライトを描画する場合のドローコールを示したものです。

テクスチャA を指定し、スプライトA1 を描画、ここで１回めのドローコールが発生します。

次に、テクスチャBに変更し、スプライトB1 を描画、ここでドローコールが２つ目です。

再び、テクスチャAに戻し、スプライトA2 を描画、ドローコールは３、

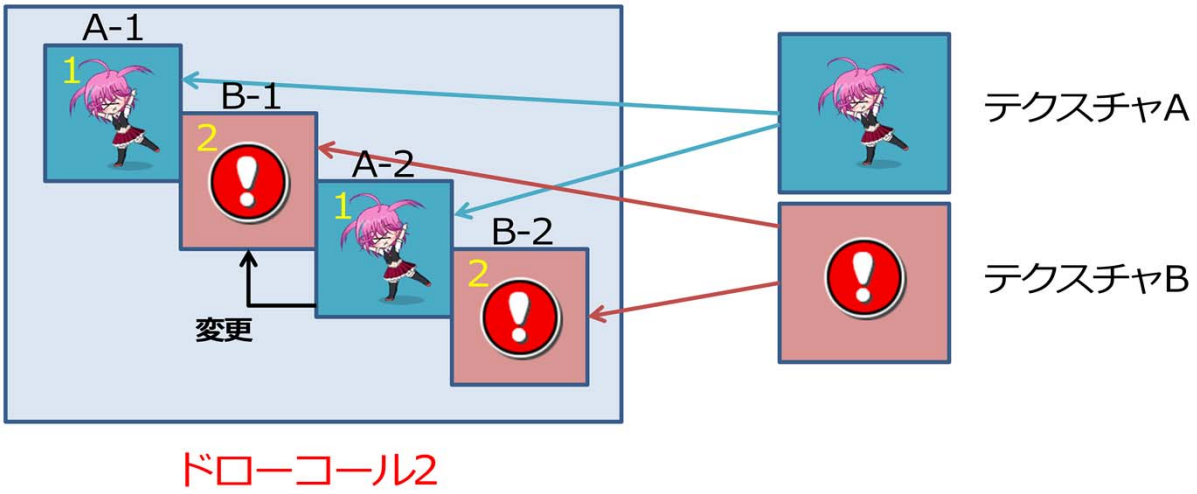
同じく、テクスチャBに変更し、スプライトB2 を描画すると、

最終的にドローコールは４回になります。

テクスチャの枚数がもっと増えて、なんの配慮もせず描画していくとドローコールはうなぎ登りとなり、アプリのパフォーマンスは大幅に下がることになります。

バッチングとは？

同じテクスチャを使ったものを一度にまとめて描画する



ドローコール2

OPTPiX

WebTechnology®

45

そこで、同じテクスチャを使ったものを一度にまとめて描画することで、ドローコールを減らす工夫が重要になります。

さきのケースの場合、描画順番を変更し、スプライトA1、A2の情報を連結することで２回に減らせます。

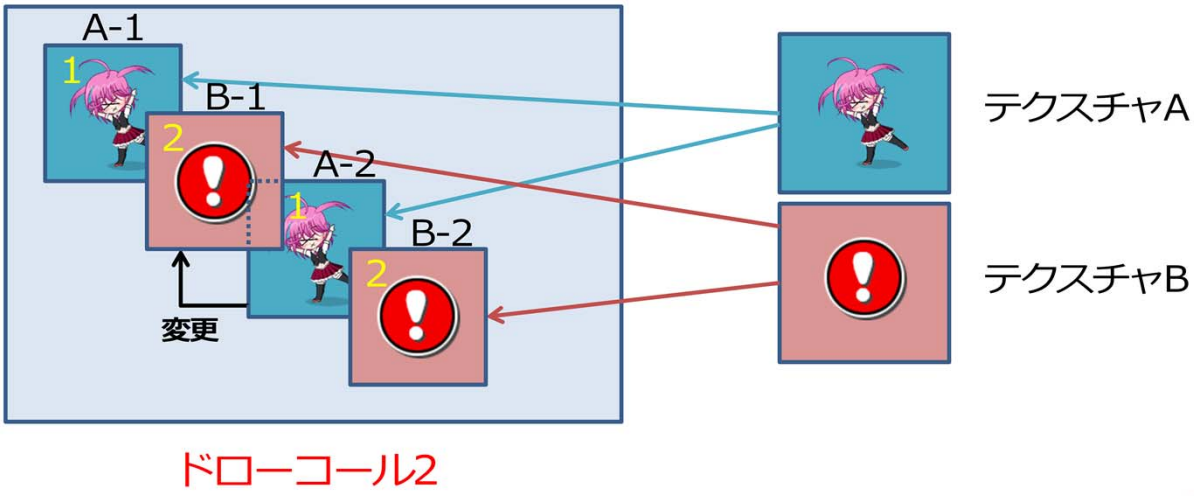
テクスチャA を指定し、スプライトA1、A2 を描画し、最初のドローコールが発生します。

次に、テクスチャBに変更し、 スプライトB1、B2 を描画すれば、ドローコールは２回で済みます。

このように最適化することをバッチングといいます。

バッチングと描画順序の関係

重なっている状態でバッチングすると描画順が狂う



ドローコール2

OPTPiX

WebTechnology®

46

ここで、問題が起こります。

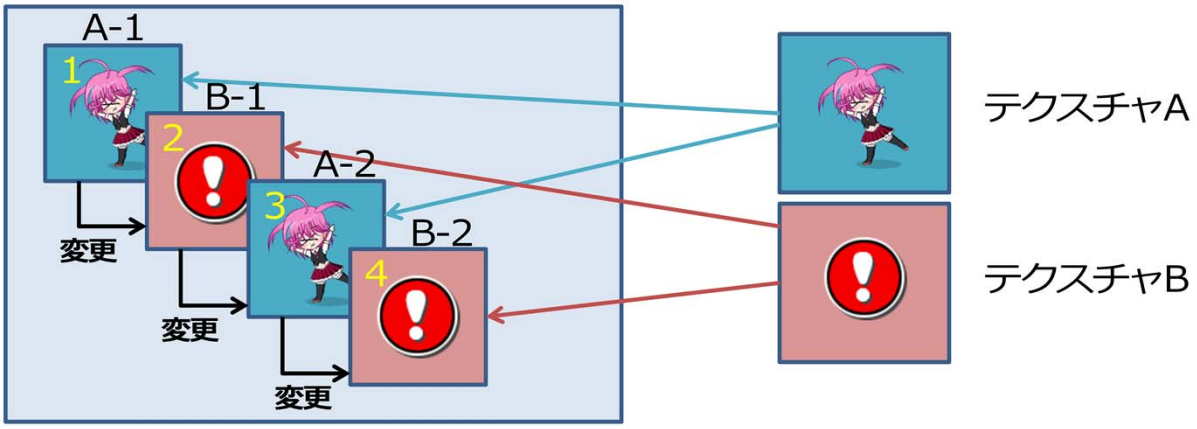
さきのケースのように描画順序を変更しても見た目が破綻しない場合は問題ありませんが、

図のように、スプライト同士が重なり合っている場合は、スプライトA2が、後から描かれたスプライトB1の後ろに隠れてしまいます。

このようにスプライト間で前後関係を守る必要がある場合は、入れ替えることができず、バッチングができません。

バッチングと描画順序の関係

描画順を守るためバッチングできない



ドローコール4

OPTPiX

WebTechnology®

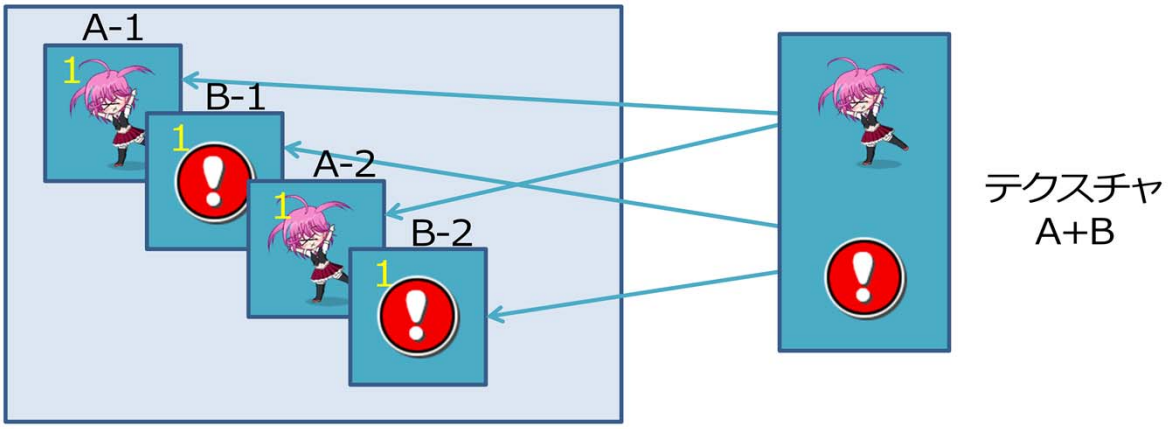
47

描画順を守ろうとするとバッチングができないため、結局、逐次テクスチャを切り替えるような処理に戻すことになります。

その結果、ドローコールが４になります。

テクスチャのアトラス化

バッチングできるようにテクスチャを１枚に合体する



ドローコール1

※ただし、現実的には必ずしも一枚のテクスチャに全ての画像を収められるとは限らない

OPTPiX WebTechnology® 48

では、どうすればよいのかというと、テクスチャの切り替えが発生しないよう１つにまとめてしまえばよいのです。

一般にテクスチャのパッキング、またはアトラス化と呼ばれるものです。

とはいえ、使用できるテクスチャサイズの制限などから、すべてのスプライトを１つにすることはできませんから、なるべく描画の順番が確定している単位でアトラス化するのが良いでしょう。

シューティングゲームで例えるなら、奥から順に、BG、地上物、飛行物、エフェクト、スコア表示、といった分け方になるでしょう。

バッチングのまとめ

- Unity, Cocos2d-x Ver.3系では自動バッチングに対応している
- Cocos2d-x Ver.2系では手動でバッチングする必要がある
- 自動、手動問わずバッチングを有効にする工夫が重要

- 参考情報
  http://japan.unity3d.com/unite/unite2014/files/DAY1-1700-room3-WebTechnologyAndKLab.pdf

OPTPiX WebTechnology® 49

ところで、これまで説明したバッチングをゲームエンジン側で自動的に行なってくれるものがあります。

具体的には Unity、Cocos v3 で対応されています。

Cocos v2 ではこの機能が無いため、プログラマが描画順序を考慮して、バッチング対象のスプライトをまとめてから、描画命令を発行する必要があります。

いずれにしても、自動、手動を問わず、効果的にバッチングするための工夫が重要なのには変わりありません。

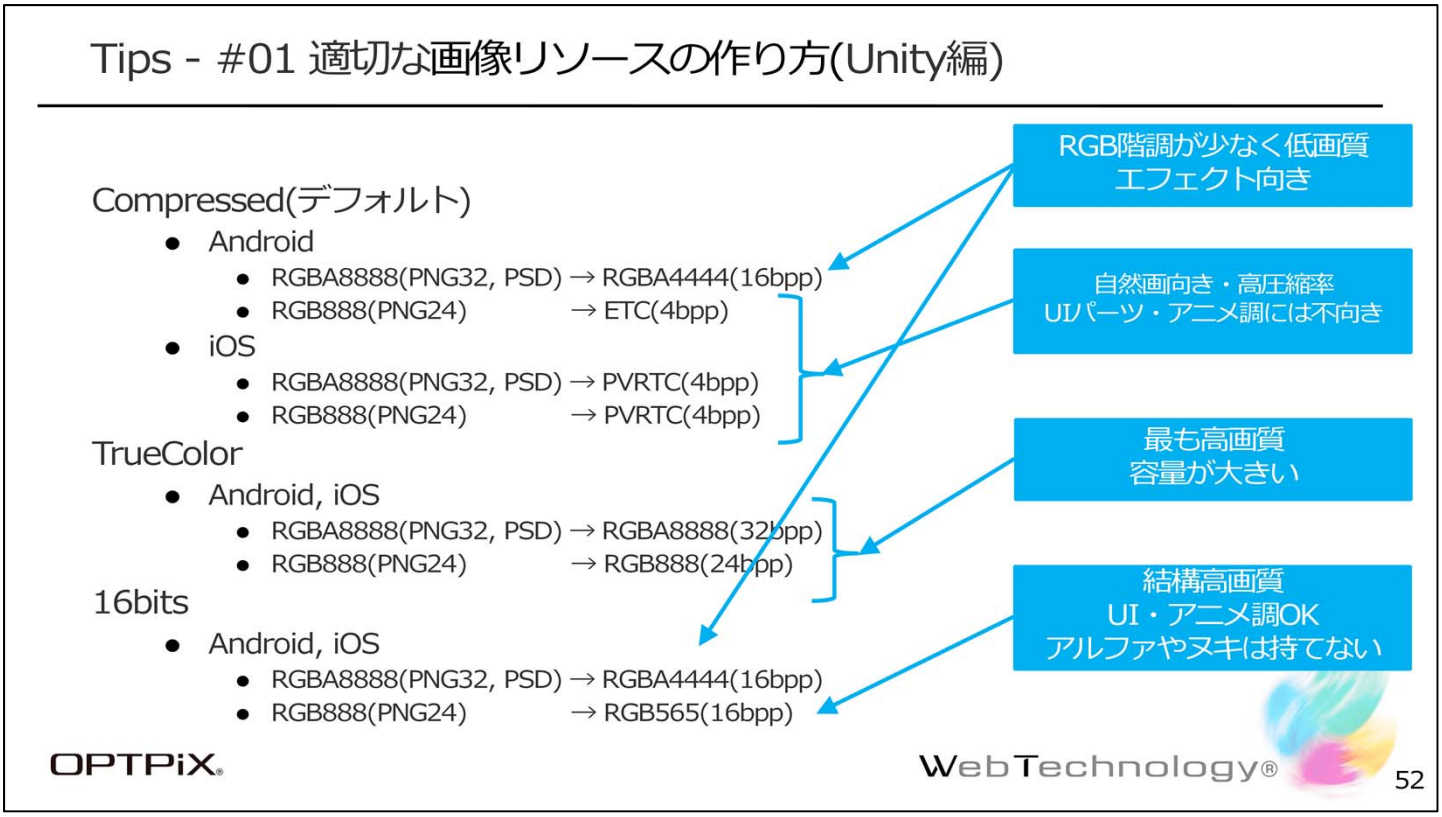
http://japan.unity3d.com/unite/unite2014/files/DAY1-1700-room3-WebTechnologyAndKLab.pdf

第３部　Tips

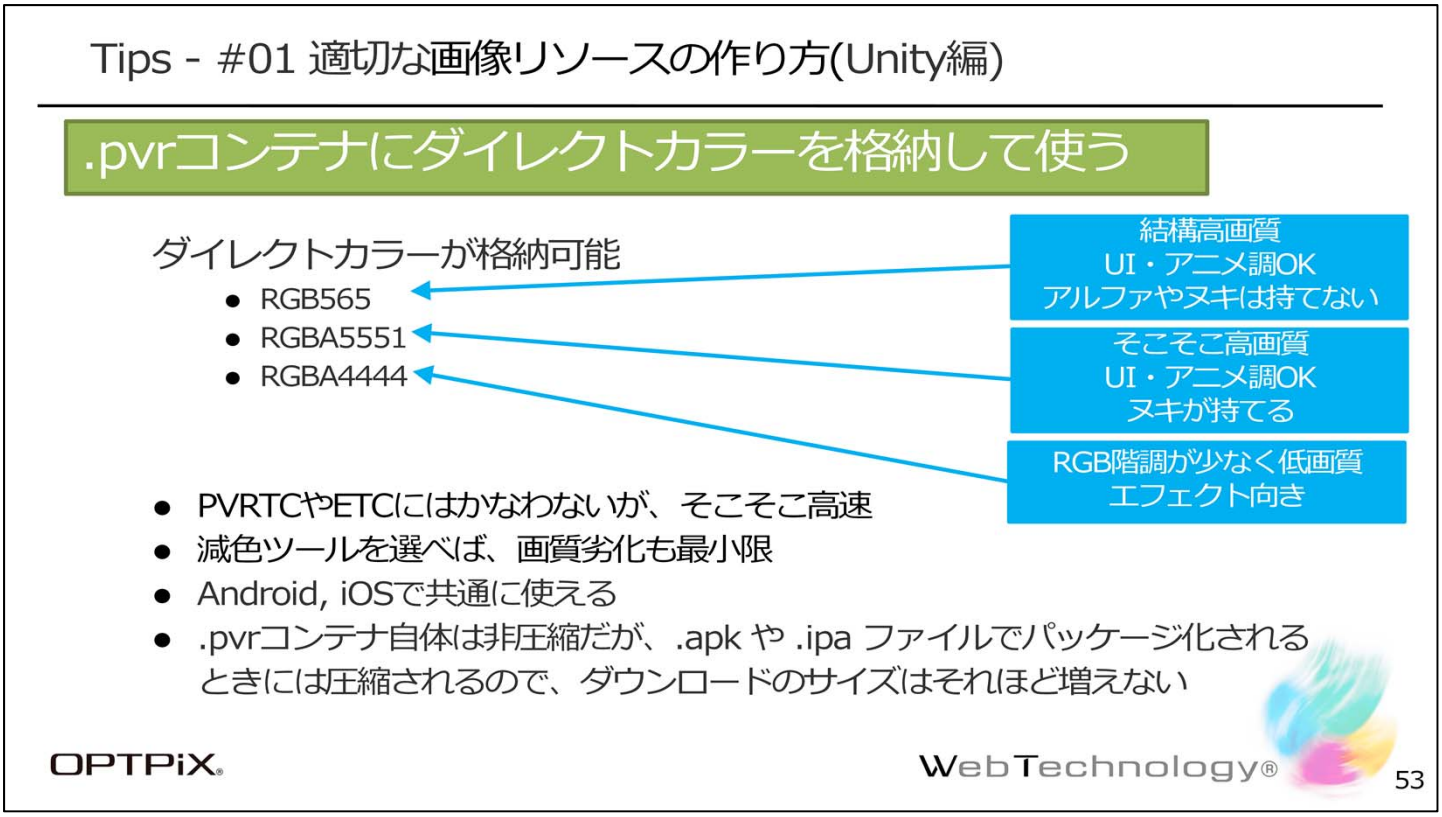
Tips - #01 適切な画像リソースの作り方(Unity編)
Tips - #02 適切な画像リソースの作り方(Cocos2d-x編)
Tips - #03 2DゲームでPVRTC,ETCを使う場合の注意点
Tips - #04　.pvrファイルをSpriteとして使う方法(Unity)
Tips - #05　どの範囲の機種をサポートするべきか
Tips - #06　Androidの画面解像度
Tips - #07 カタログスペックと実測値の乖離
Tips - #08 プロファイラ

Unity で、描画性能や画質を向上させるためのノウハウ

Unityはとても親切にできているので、とりあえず画像を放り込むだけで、ちゃんと表示してくれます。
PNGはもちろんのこと、Photoshopで作った.psdファイルでもそのまま読み込めるという優れもので、このことが売りにもなっています。
ただ、一見動作はしていますが、そのままだと性能低下や画質低下を招くことがあります。



Unityに画像を読み込ませたときの動作を見ていきます。
Unityは、デフォルトでは、Compressed というモードになっています。
この場合、画像は決め打ちで変換するので、UIパーツやアニメ調の画像には不向きな形式に変換されてしまったりします。
RGB階調が少なく低画質でエフェクト向きだとか、UIパーツ・アニメ調には不向きなフォーマットになってしまいます。
TrueColorや16bitsという指定をすることもできますが、容量が大きくなってしまうなど、それぞれ一長一短です。



解決方法として、.pvrコンテナにダイレクトカラーを格納するという方法があります。
それなりに高画質なRGB565とか、ヌキが使えるRGBA5551とか、エフェクト向きのRGBA4444などが使えます。
予め、画像の種類に応じてこれらの形式に変換して、.pvrコンテナに格納した画像ファイルをUnityに読み込ませると良いでしょう。

.pvrコンテナにダイレクトカラーを格納して使う

.pvrコンテナを使うもう一つのメリット

- PSDやPNGでは、ビルドターゲット(iOS/Android)を切り替えるたびに画像ファイルのリソースをすべて変換する処理が入り、ファイル数が多いと数10分に及ぶこともある。
- .pvrコンテナの画像では、この自動変換は行われないため、**ビルドが高速化する。**

その他の参考情報

- 16bppはPNGで保存できないため、.pvrコンテナを使う必要がある
- Unityは、.pvrコンテナの画像は変換しない。TrueColor, 16bitsを指定しても無視される
- .pvr コンテナへのダイレクトカラー出力は、OPTPiX imésta などが対応している
- .pvr ファイルはVer.2、垂直Flip(垂直反転)を指定しておく
- .pvr コンテナのダイレクトカラーテクスチャは、縦横ドット数は任意に指定できる (PVRTCテクスチャは、2の累乗平方ドットのみ格納可能)
- どうしても16bppの画質では許容できない場合だけ、PNG24のファイルを使い、TrueColorを指定する

もう一つ、大きなメリットがあります。ビルドが高速になるというメリットです。
Unityでは、PSDやPNGを読み込ませると、ビルドターゲットを切り替えるたびに画像ファイルのリソースをすべて変換する処理が入ってしまうため、ファイル数が多いと数10分かかることがあります。
.pvrの画像ではこの自動変換が行われないので、ビルドが高速になります。

Cocos2d-x で、
描画性能や画質を向上させるためのノウハウ

- PNG32 をロード → 内部では RGBA8888
- 32bppのため、データサイズ大、パフォーマンス低

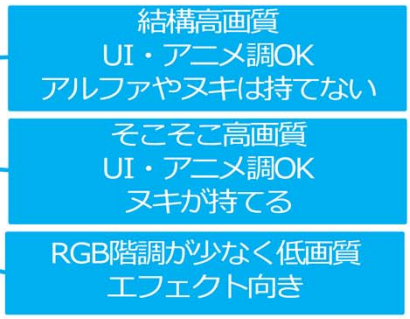
最も高画質
容量が大きい

次にCocosの場合を見てみます。
CocosもPNGファイルを直接読み込ませることができます。
ところが、フルカラー画像がそのまま使われるので、容量が大きく、描画性能も低くなってしまいます。

Tips - #02 適切な画像リソースの作り方(Cocos2d-x編)

.pvrコンテナにダイレクトカラーを格納して使う

- ダイレクトカラーの以下の形式が格納可能
  - RGB565
  - RGBA5551
  - RGBA4444



- PVRTCやETCにはかなわないが、そこそこ高速
- 減色ツールを選べば、画質劣化も最小限
- Android, iOSで共通に使える
- .pvrコンテナ自体は非圧縮だが、.apk や .ipa ファイルでパッケージ化されるときには圧縮されるので、ダウンロードのサイズはそれほど増えない

Cocosの場合も、Unityと同様に.pvrコンテナにダイレクトカラー画像を格納したファイルを使うことができます。
使える画像フォーマットもUnityの場合と同じで、2Dゲームで使いやすい形式から選ぶことができます。
.pvr コンテナにダイレクトカラーを保存できるツールはあまり多くありませんが、OPTPiX imésta などが対応しています。

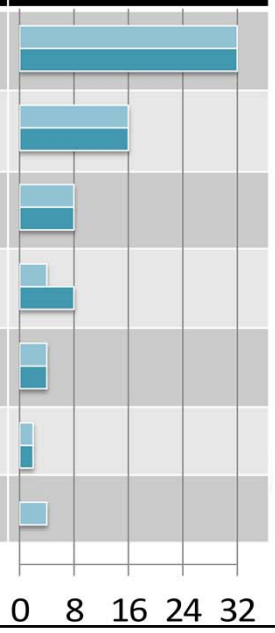
Tips - #03 2DゲームでPVRTC,ETCを使う場合の注意点

- PVRTC, ETCは、3Dテクスチャ用に開発された高圧縮フォーマットで、GPUの性能を最大に引き出すことができる
- 写真のような画像や、質感を表すようなテクスチャは、PVRTC,ETCともかなり高品質
- 動きが激しくてエッジやボヤけが気にならない場合は、PVRTC/ETC でも問題ない
- UIや、縁取りをはっきりとったアニメ調の絵では、輪郭がぼやけたり、ブロックノイズが発生する
- ETC はαが使えないので、ゲームでは使える場面が限られる
- iOSでは、PVRTCは必ず使用できるが、ETCは使用できない
- Androidでは、ETCは必ず使用できるが、PVRTCは一部の機種しか使用できない

PVRTC,ETCについて、おさらい
「圧縮テクスチャ(PVRTC・DXTC・ETC)における傾向と対策」からダイジェスト

テクスチャ形式ごとのbppと圧縮比

テクスチャ形式	アルファなし	アルファあり	サイズ
24bit / 32bitダイレクトカラー RGB888, RGBA8888	32bpp †	32bpp	
16bitダイレクトカラー RGB565,RGBA4444,RGB5551	16bpp	16bpp	
8bitインデックスカラー	8bpp	8bpp	
DXTC(S3TC)	4bpp	8bpp	
PVRTC 4bpp	4bpp	4bpp	
PVRTC 2bpp	2bpp	2bpp	
ETC	4bpp	使用不可	



OPTPiX

bpp：bit per pixel
† VRAM上では32bit必要

58

一番上のRGB888,RGBA8888が元画像。
一番小さいのがPVRTC 2bit per pixel。
ただ、PVRTC 2bppは、さすがにかなり劣化してしまいます。

圧縮テクスチャはもともと3Dポリゴンのテクスチャ用なので、左上の画像のような、こういう質感を表すような模様とか、写真のような画像の圧縮は得意です。

右下に行くに従って苦手になります。

アニメキャラやUIパーツのような、縁取りがクッキリした画像の圧縮は苦手です。

どのような圧縮結果が得られるのか、充分注意して使う必要があります。

PVRTC

オリジナルの画像

OPTPiX

WebTechnology®

60

良くあるこういうUIパーツ

PVRTC

PVRTC 4bpp

渗みを克服することは難しい

OPTPiX

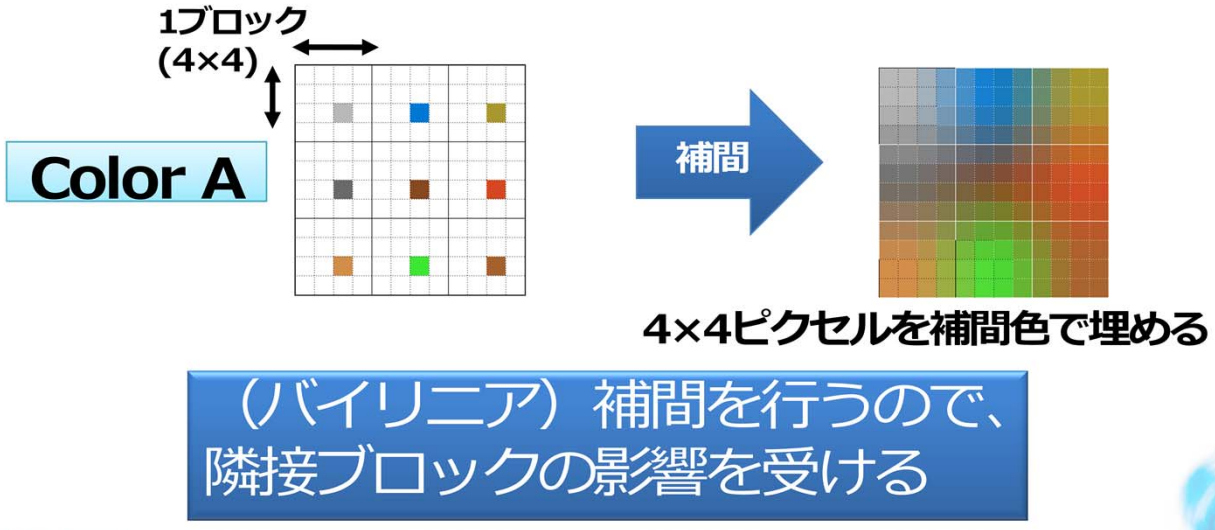
WebTechnology®

61

PVRTCで圧縮すると、文字の輪郭のところににじみのような汚れが出てしまいます。PVRTCでは、これはもうどうしようもありません。

PVRTCは、PowerVR Texture Compressionの略で、名前の通りPowerVR専用の圧縮形式。

PVRTCの圧縮・伸長方法



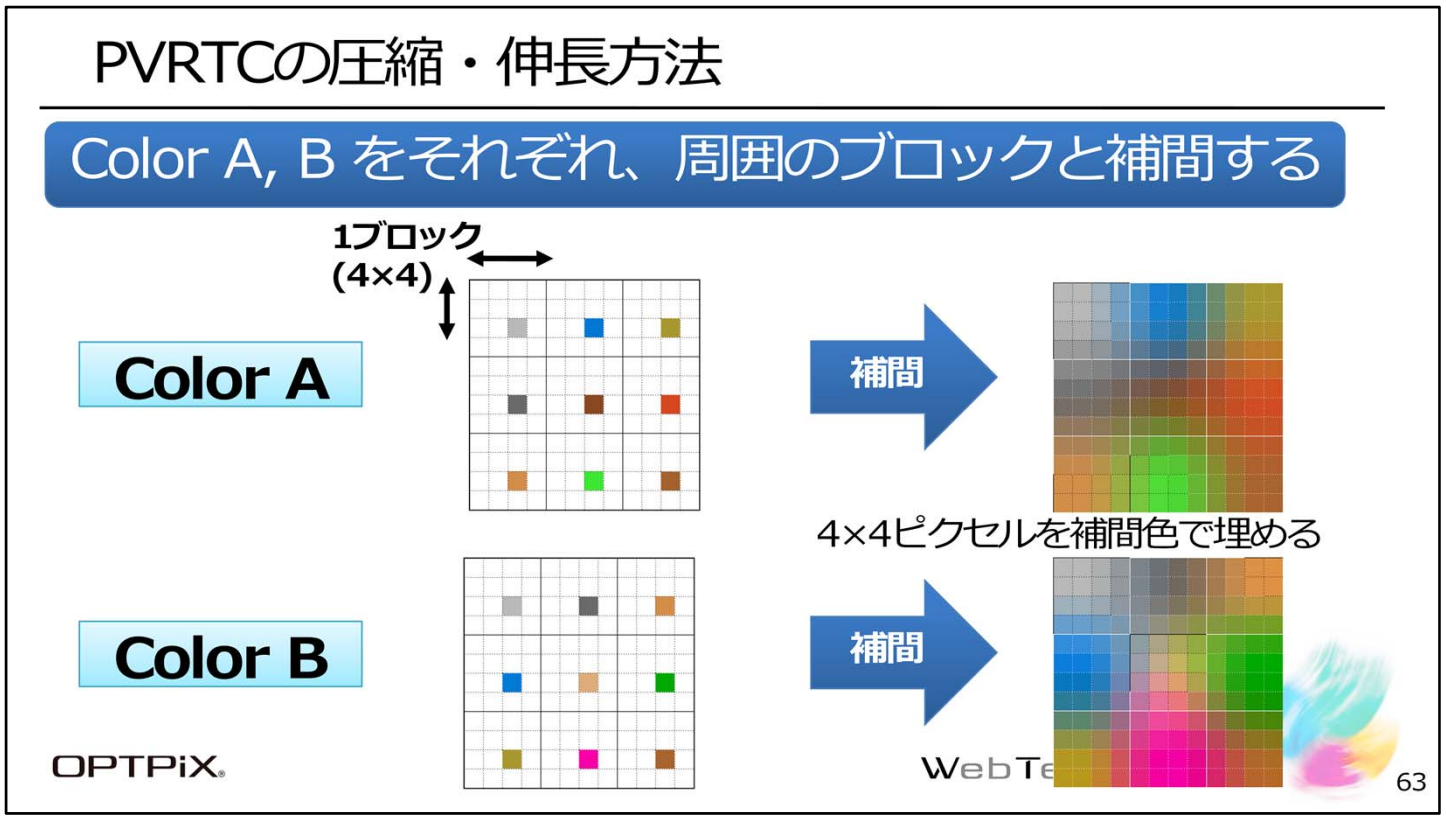
（バイリニア）補間を行うので、
隣接ブロックの影響を受ける

滲みの原因はPVRTCの圧縮方法に由来しています。
DXTCやETCも同じなのですが、PVRTCは、画像をこのように4x4ドットのブロックに区切って圧縮しています。

圧縮の仕組みを理解するために、ここでは伸長の手順を説明することにします。
このように、ひとつの4x4ブロックのなかには代表色が1色あります。
代表色は、こんなふうに、それぞれのブロックごとに、1色づつあるわけです。

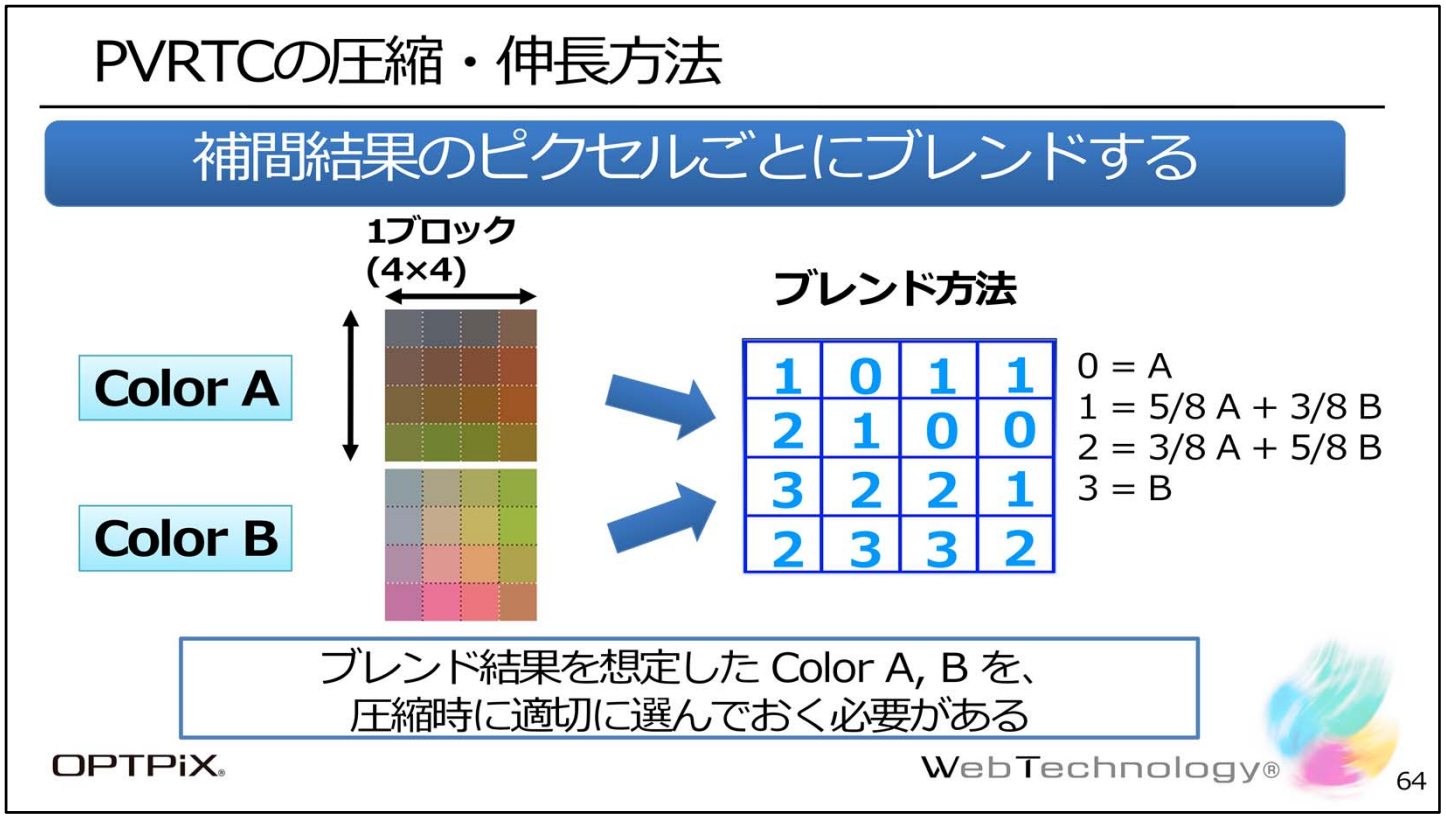
そして、この代表色を使ってバイリニア補間をかけて、右のような画像を作ります。

さすがにこれだけだとボケボケなので、ここで終わりではありません。



PVRTCでは、ひとつの4x4ブロックには、このように2つの代表色が設定できるようになっています。

このように、ボケボケの画像を2種類作ります。



PVRTCでは、このボケボケのColor AとColor Bの画像のほかに、もう一つ情報を持っています。

Color AとColor Bのどちらの画像の色を使うか、というブレンド方法の情報です。

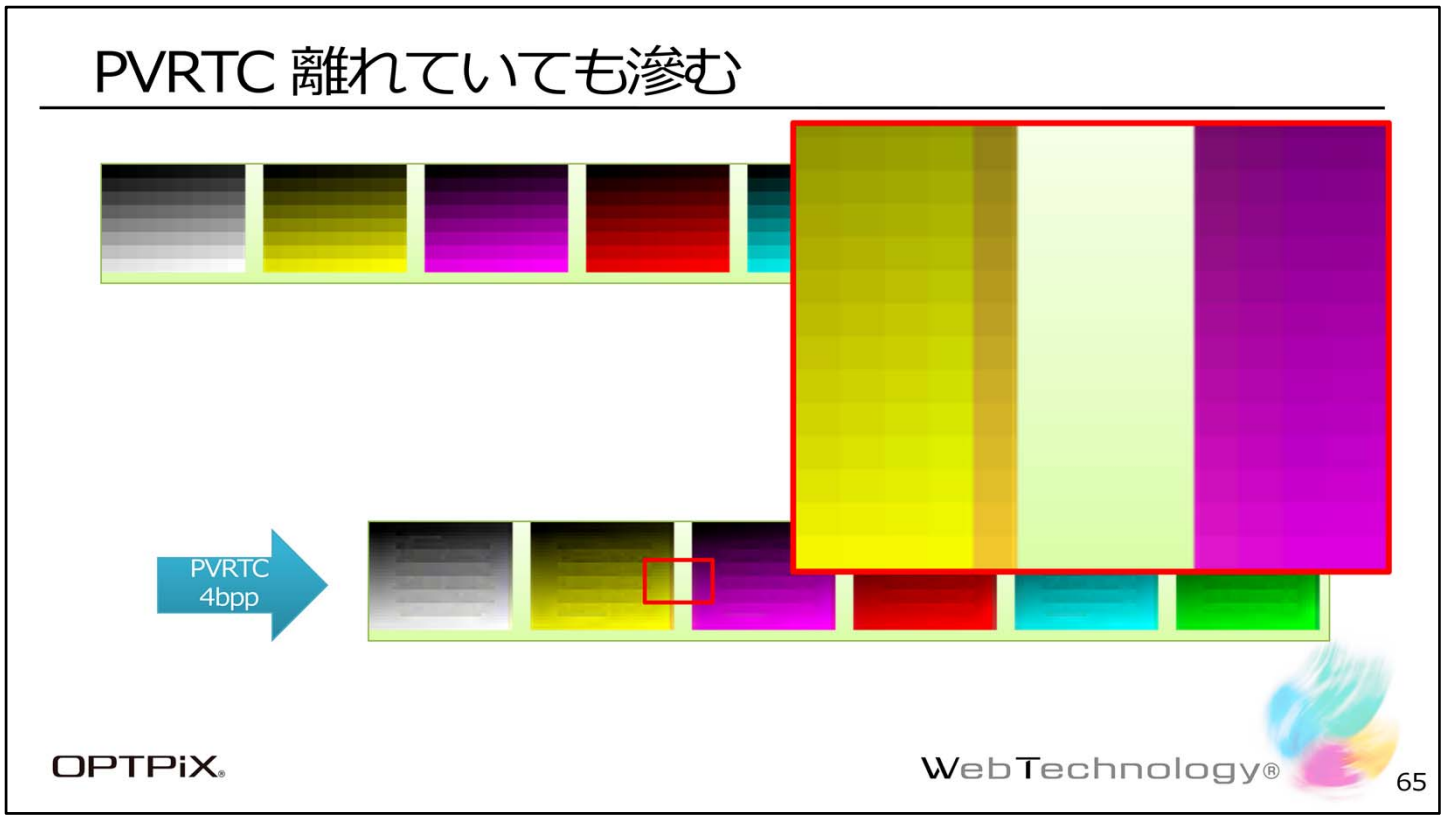
この右側のマトリックスがそれです。

この値が0のピクセルはColor Aの色を採用します。

　　値が3のピクセルはColor Bの色を採用します。

値が1や2のピクセルは、この式の通りにAとBをブレンドした色を採用するわけです。

このような工夫がされているのですが、やはり色の元になるColor A、Color Bがもともとバイリニア補間で作られた画像なので、どうしても輪郭線のまわりは再現性が低くなってしまうというわけです。

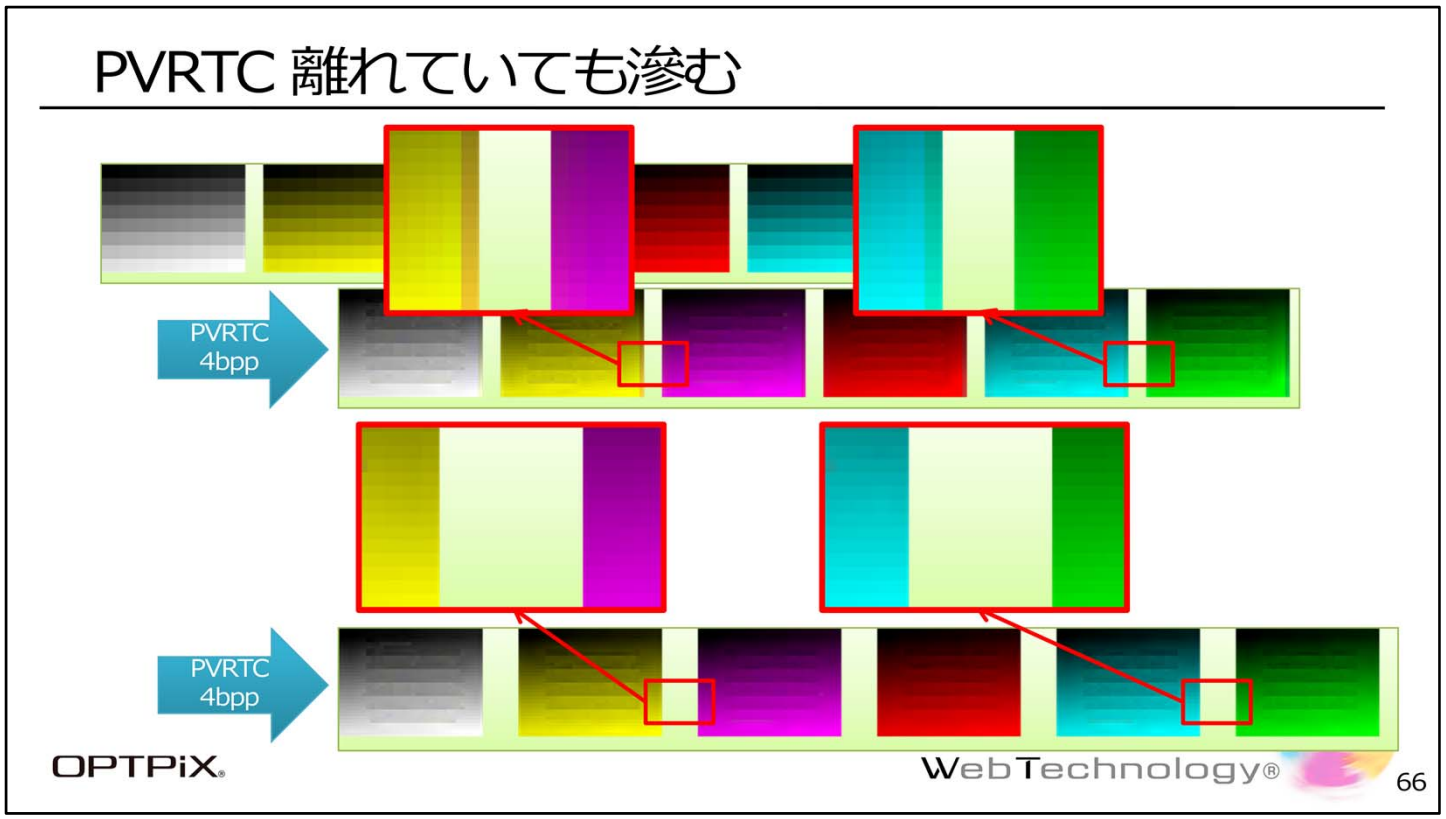


次に、圧縮のサンプルとして、こういうカラーバーの画像を圧縮した場合を見てみましょう。

エッジのところが滲んでいるのがわかると思います。

アトラスでのマージンが狭いと、PVRTCでは隣同士のパーツが干渉してしまうことがあるので、

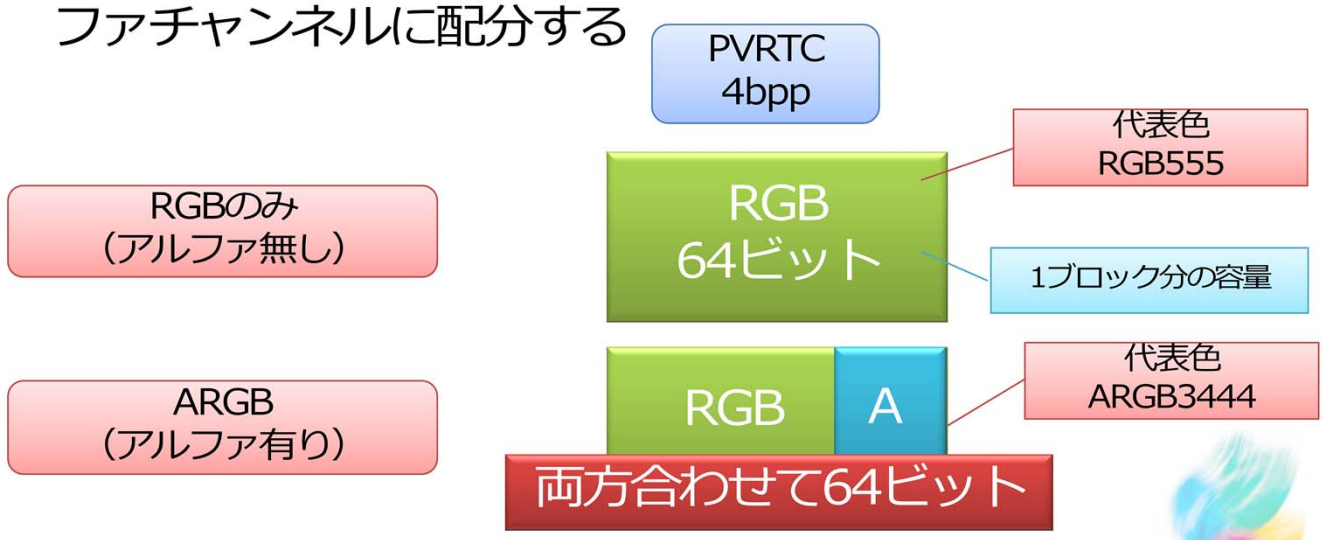
その場合はマージンを広げると解決します。



上の画像はは先ほどと同じマージン、下の画像はマージンを倍にした場合です。
下の画像ではにじみがなくなっています。
ただ、マージンを取り過ぎると、結果的に圧縮率が下がったのと同じになるので、必要以上に隙間を空けないように気をつけた方が良いです。

PVRTCのアルファチャンネル

- PVRTCはRGBチャンネルに使うビット数を削ってアルファチャンネルに配分する

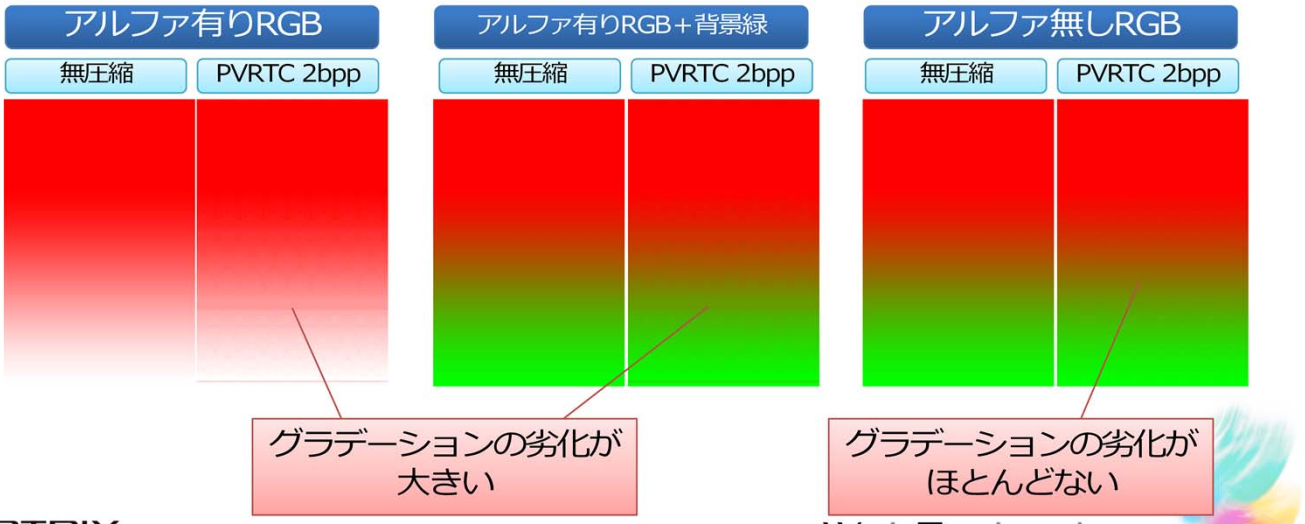


OPTPiX WebTechnology® 67

次に、PVRTCのアルファチャンネルについてです。
PVRTCでは、アルファなしでもアルファありでも、どちらも4bit per pixelです。
アルファなしのときには代表色はRGB555なのですが、
アルファありではARGB3444と色の情報量が少なくなってしまいます。

PVRTCのアルファチャンネル

- PVRTCではグラデーションをアルファで表現するよりも、RGBで表現する方が綺麗に表示できる



OPTPiX WebTechnology® 68

もともとPVRTCは、このようなグラデーション画像の圧縮は得意なはずなのですが、アルファありではこのようにノイズが乗っているのがわかると思います。

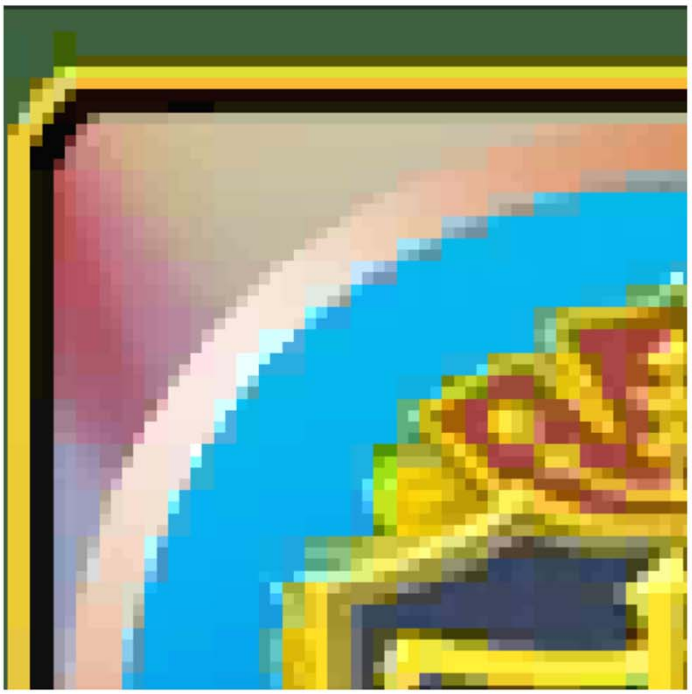
なお、これはわかりやすくするために圧縮率の高い2bppモードの結果をお見せしています。

同じ2bppでも、アルファなしであれば、非常に綺麗にグラデーションが再現できています。

得意なタイプの画像では、16分の1に圧縮してもここまで綺麗にグラデーションが再現できるのは凄いです。

ETCのブロックノイズ

OPTPiX



69

ETCは、Ericsson Texture Compressionの略で、Androidでは全ての機種で共通して使える圧縮テクスチャ形式です。

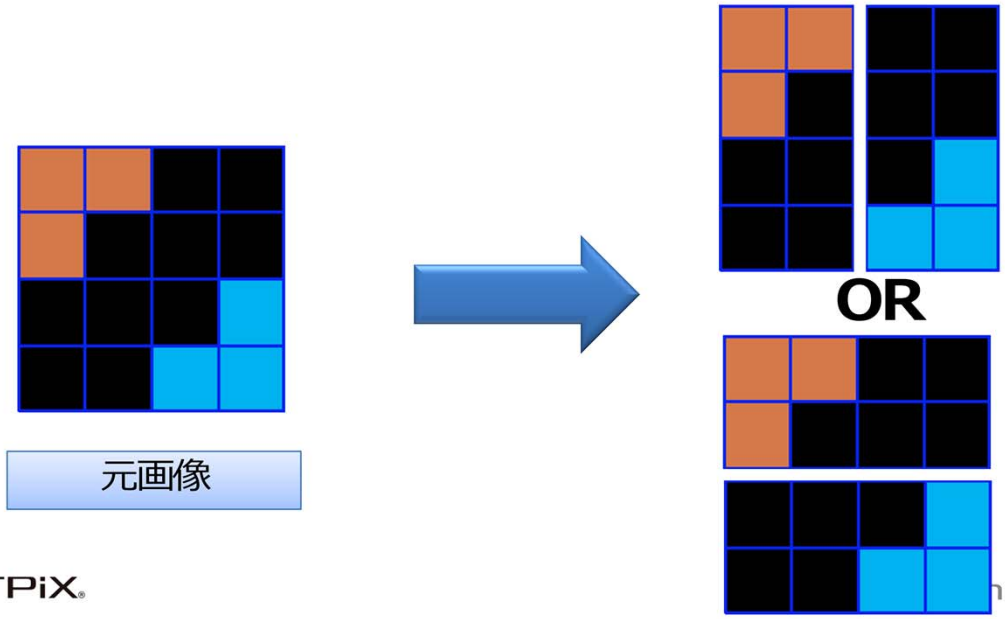
比較的劣化の少ないフォーマットですが、やはりこういうクッキリした輪郭の部分では、ブロックノイズが目立つことがあります。

ゲームアプリで使う場合の最大の問題は、アルファが使えないことです。抜き色も使えないので、ゲームでは使える場面が限定されてしまいます。

ETC2という新バージョンではアルファも使えるのですが、まだ普及していないので利用できるのはこれからになります。

ETCの圧縮方法

- ４×４のブロックを２×４のサブブロックに分割する



OPTPiX

nology®

70

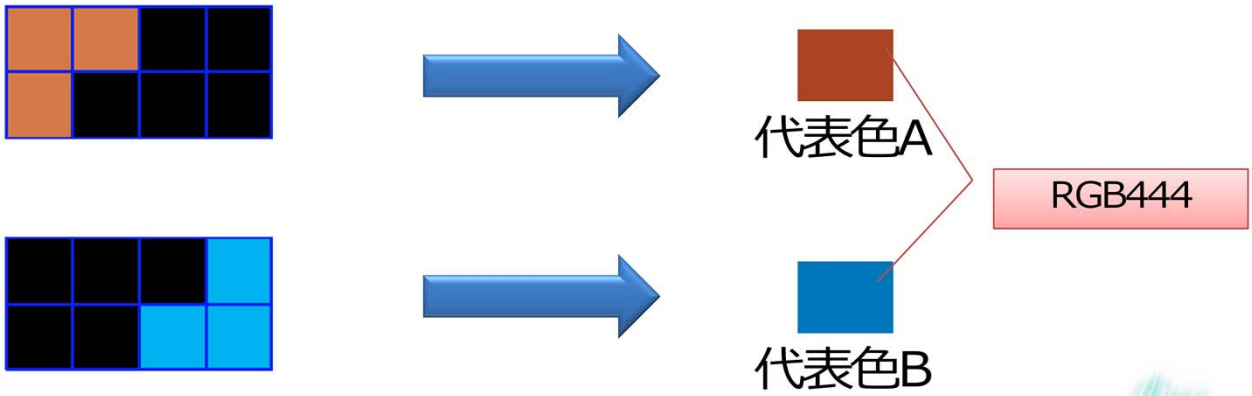
ETCの圧縮方法についてです。

ETCでは、PVRTCと同じく4x4ピクセルを1ブロックとして圧縮していくのですが、これをさらに二つのサブブロックに分割します。

縦に分割することも、横に分割することもできます。

ETCの圧縮方法

- サブブロック毎に代表色を決める



OPTPiX

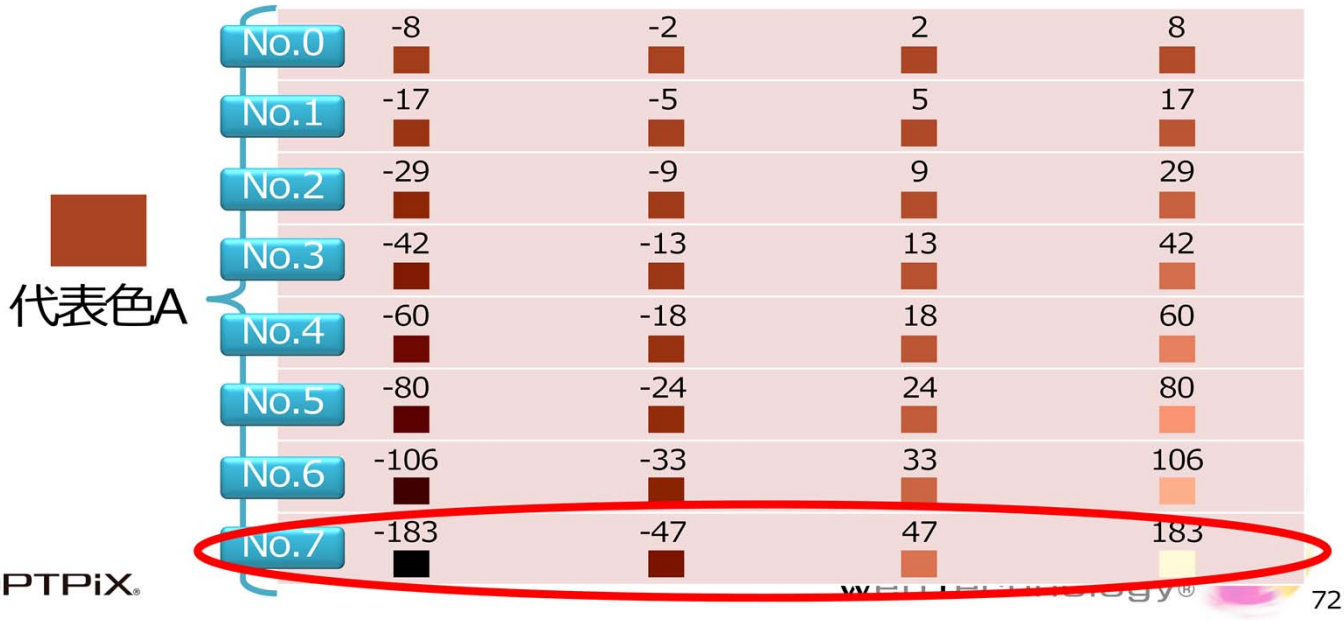
WebTechnology®

71

サブブロックに分割したら、例によってそれぞれについて代表色を決めます。
サブブロック1つについて代表色は1つだけです。

ETCの理論 (3) 代表色から4色を作る

- 代表色に対する輝度の変換テーブルを選ぶ



OPTPiX

72

代表色が決まったら、補間色を4色作りだします。

補間の方法は決まっていて、このような輝度の変換テーブルNo.0～No.7のなかから一つ選ぶという方式になっています。

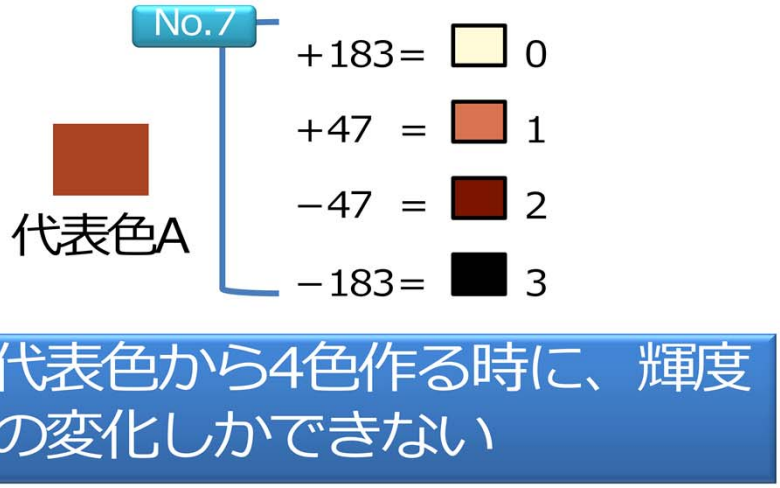
このテーブルは完全に固定で、数値を変更したりすることはできません。

それぞれの数値をRGB値全てに足したり引いたりして補間色を作ります。

この時、もし0以下や256以上の数値になったら0と255にします。

今回はNo.7を使うことにします。

ETCの圧縮方法



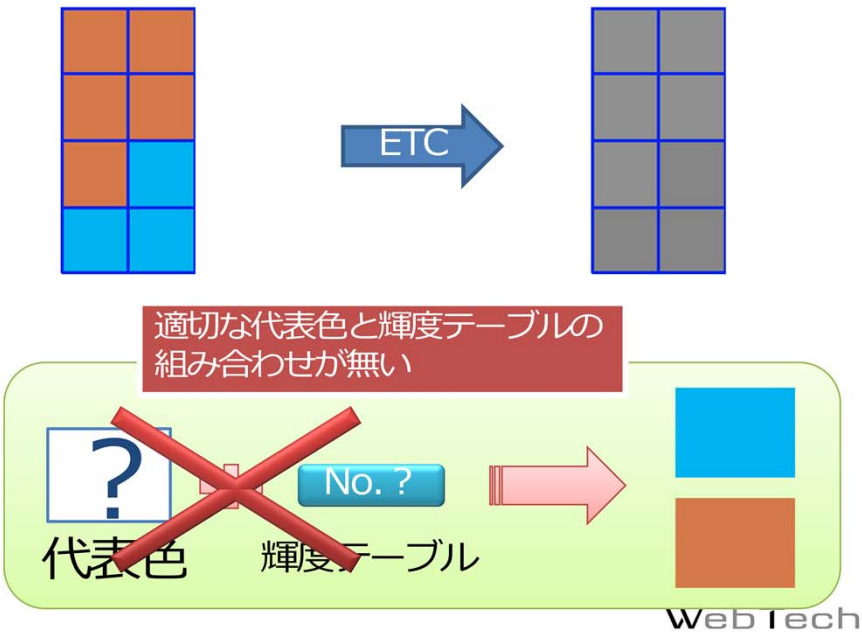
OPTPiX WebTechnology® 73

ETCのポイントとしては、代表色を決めると、輝度の変化しかできない、ということです。

つまり、代表色を茶色と決めたら、同じサブブロック内で青色や緑色を使うことはできません。

逆に、この例で使った7番の変換テーブルを選べば、黒や白は概ね作り出せるということになります。

ETCの圧縮方法



OPTPiX

Web Technology®

74

このようなサブブロックをETC圧縮する場合、青色と茶色を使いたい訳ですが、
この2色を出せるような代表色と輝度テーブルの組み合わせはありません。
その結果、ETCのアルゴリズムでは右の画像のように全部灰色になる、といった
おかしな結果になります。
これが最初の画像のエンブレム部分でブロックノイズが出ていた原因です。

輪郭を入れた例

黒 RGB(0,0,0)や白 RGB(255,255,255)はサブブロック中で
1色としてカウントされないのでジョーカー的に使うことができる。
そのため絵柄がOKであればノイズ低減に利用できる。

Technology®

75

ただし、黒や白はサブブロック内で作りだすことが容易なので、比較的自由に使うことができます。

そのため、絵柄的に問題なければ、色の境界に黒や白で輪郭を入れることでノイズ低減に利用できます。

圧縮テクスチャにおける傾向と対策 ダイジェスト

詳しくは、こちらをご覧ください
http://cedil.cesa.or.jp/session/detail/1024

OPTPiX WebTechnology® 76

詳しくは、こちらをご覧ください
http://cedil.cesa.or.jp/session/detail/1024

Tips - #04 .pvrファイルをSpriteとして使う方法(Unity)

- Unityエディタ上で .pvrファイルを追加すると、Spriteの作成ができない (Inspectorの設定項目が無効化されている)
- スクリプトでロードすると、Spriteのテクスチャに.pvrファイルを使用できる

```
var tex = Resources.Load("<FilePath>") as Texture2D;
var texRect = new Rect(0, 0, tex.width, tex.height);
var sprite = Sprite.Create(tex, texRect, new Vector2(0.5f, 0.5f),
100, 1, SpriteMeshType.FullRect)
```

Tips - #05　どの範囲の機種をサポートするべきか

- Cocos2d-x は、iOS 5.0以降、Android 2.3以降をサポート
- Unity は、iOS 4.0以降、Android 2.3.1以降をサポート
- サポート範囲を広げると
  - 低性能機対応のための開発工数増大
  - 低性能機にあわせたゲーム設計

- ゲームアプリでは、iOS 7以降、Android 4.0以降という割り切りも

Tips - #05　どの範囲の機種をサポートするべきか

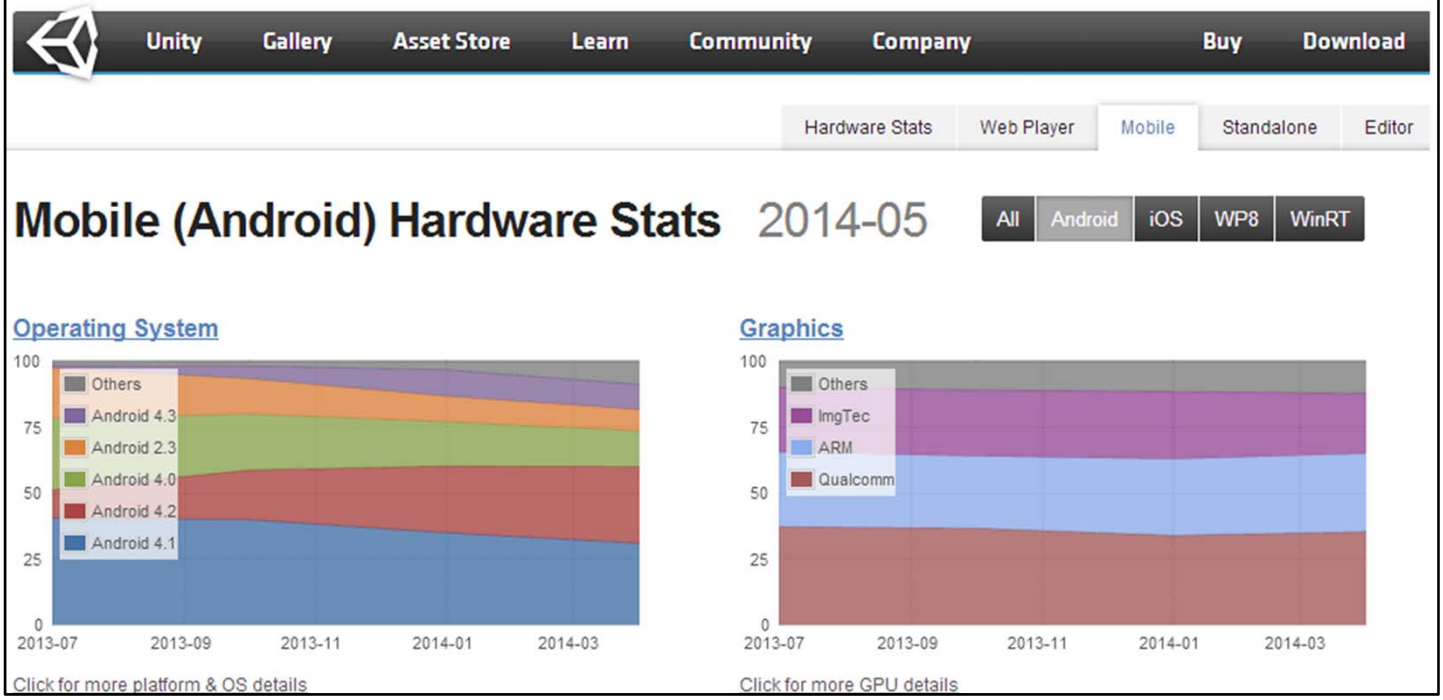
「Unity Hardware Statistics」が役に立つ

http://stats.unity3d.com/mobile/

「Unity Hardware Statistics」が役に立つ

http://stats.unity3d.com/mobile/

Tips - #05　どの範囲の機種をサポートするべきか



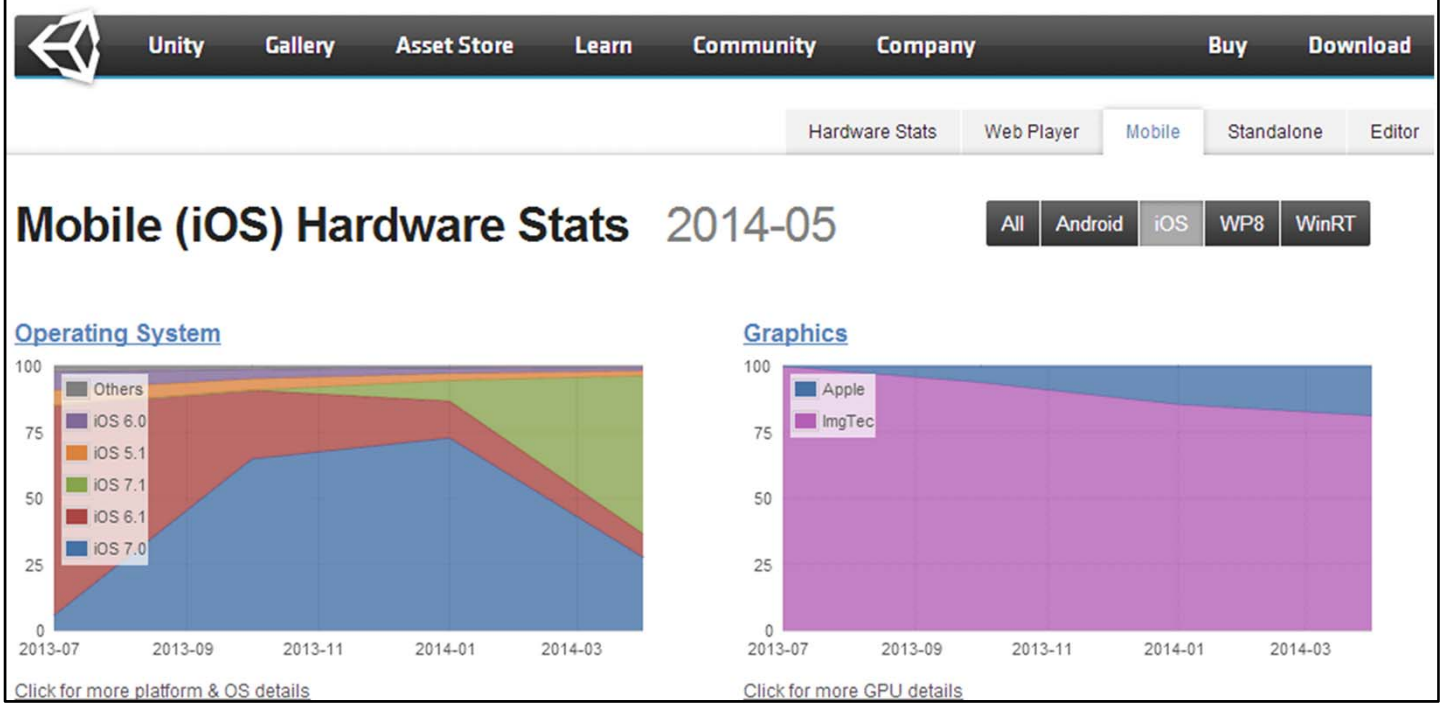
http://stats.unity3d.com/mobile/index-android.html

Androidのスペック統計で、1年前からの傾向がわかるようになっています。

このように、OSバージョン、GPUの種類、CPUのコア数、メーカー名、メモリ容量、ディスプレイのアスペクト比がわかります。

OSバージョンについて言えば、今年の5月の段階で、Android 4.0以上が9割以上です。

Tips - #05　どの範囲の機種をサポートするべきか



http://stats.unity3d.com/mobile/index-ios.html
iOSについての情報です。
なぜかここにもメーカー名があって、当然ですが100% Appleになってます。
面白いのは、Grphicsのところで、Imagination TechnologiesとAppleの2つにわかれていて、1年前からAppleが増えてきています。
統計の詳細を見ると、iPhone 5Sからは、Apple A7というSoCの名前になっているようです。といっても実際の中身はPowerVRのままです。
iOSのほうも、9割方はiOS 7になっていますね。

Tips - #05　どの範囲の機種をサポートするべきか

- AppStore のiOSシェアhttps://developer.apple.com/support/appstore/ の結果とほぼ同じ
- Android (Google Play)のOSバージョンシェア
  http://developer.android.com/intl/ja/about/dashboards/ の結果でも、85%がAndroid 4以上
- 上記は全世界の情報なので、日本のユーザーの動向そのものではないことに注意
- Google Playにアプリを登録すると、そのカテゴリ(ゲーム、ツールなど)毎のAndroidバージョンのシェア情報が得られるようになる。日本をターゲット市場にしたアプリをリリースしていれば、そこから情報が得られる
- Android 3.xの情報が出ていないが、1.0%以下のため。3.xはもともと少なかった上に、バージョンアップで4.xになった機種があるため。3.xは無視して良い。

Tips - #06 Androidの画面解像度

- Android の画面解像度は多種多様
- Androidの断片化問題のひとつ
- 画面解像度にどのように対応するか

Tips - #06 Androidの画面解像度

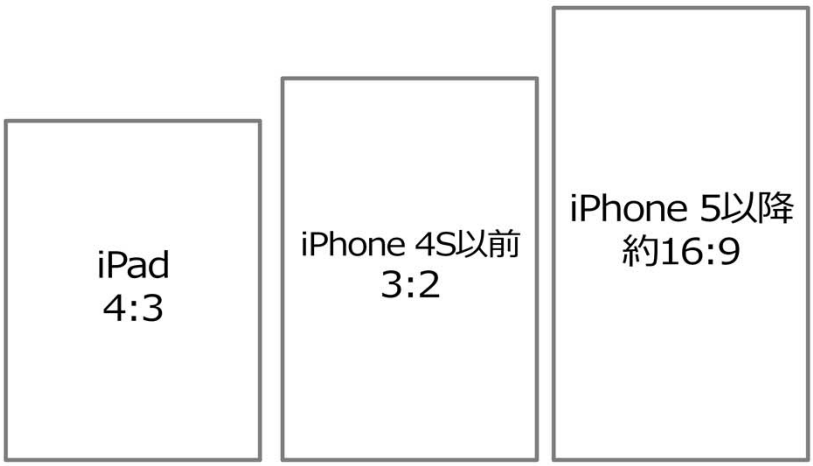
解像度	アスペクト比	Androidの主な解像度一覧(国内発売製品)
320x480	2:3	採用機種少ない
480x800	3:5	Nexus One, Nexus Sで採用。2010～2011には非常に多かった。9:16と10:16の中間
480x854	約9:16	2010～2011には非常に多かった。
540x960	9:16	2011～2012には非常に多かった。
640x960	2:3	採用機種少ない
600x1024		採用機種少ない(一部タブレット)
768x1024	3:4	意外なことに採用機種少ない(iPadの解像度)
720x1280	9:16	Galaxy Nexusで採用。2011末頃～採用多い
768x1280	3:5	Nexus 4で採用。他の搭載機は非常に少ない。9:16と10:16の中間
800x1280	10:16	Nexus 7(2012)で採用。2011末頃～タブレットそれなりに多い
1080x1920	9:16	Nexus 5で採用。2013～現在まで、非常に多い
1200x1920	10:16	Nexus 7(2013)で採用。2012～タブレットで多い
1440x2560	9:16	採用機種少ない
1600x2560	10:16	Nexus 10で採用。2013～タブレットに採用されはじめた

16:10	1.6
5:3	1.666…
1024x600	1.706…
16:9	1.777…

国内で発売された主なAndroid製品の解像度をまとめてみました。
この表をよくよく見てみると、あることに気がつきます。
16:9と16:10の間のアスペクト比の画面が大半なのです。
3:5の機種がそれなりにありますが、これは16:9と16:10の間です。
採用機種がレアになりますが、1024x600も、やはり16:9と16:10の間です。
一番外れているのが、3:2と4:3ですが、採用機種が僅かなので、これは無視して良いでしょう。

Tips - #06 Androidの画面解像度

- 実はiOSのほうがアスペクト比問題は深刻



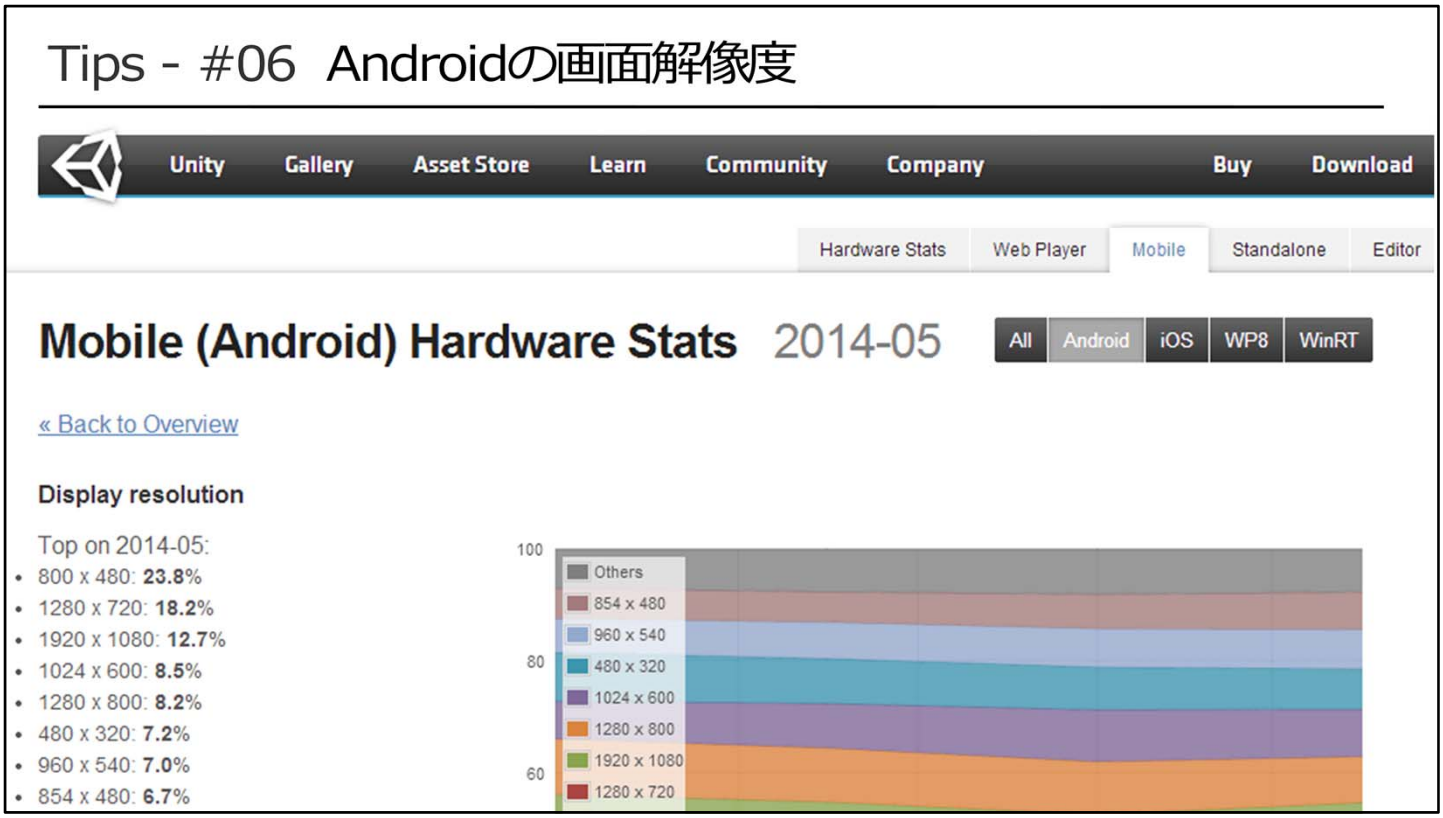
4:3	1.333…
3:2	1.5
16:10	1.6
5:3	1.666…
1024x 600	1.706…
16:9	1.777…

iOS
Android

Androidの画面解像度というタイトルでお話をしていますが、実はiOSのほうがアスペクト比の問題は深刻です。
iPadの場合は、まったく画面レイアウトが変わってくるので、ここで同列に論じる必要はないかも知れませんが、
iPhone 4S以前のサポートをやめれば、Androidと同じアスペクト比のなかに収まります。

Unity Hardware Statistics を調べてみると、iPhone 4と4Sあわせて、2014/5時点でまだ23%あるようです。
現時点ではこれはなかなか切り捨てづらいですが、もうすぐiPhone 6も発売されますし、1年経てば状況はかなり変わってきそうです。

Tips - #06 Androidの画面解像度



http://stats.unity3d.com/mobile/display-android.html
Androidの画面解像度の話に戻ります。
「Unity Hardware Statistics」で画面解像度を調べてみると、トップは800x480で23.8%です。
2番目がHD解像度、3番目がフルHDになっています。

Tips - #06 Androidの画面解像度

- Unity Hardware Statistics では、800x480のシェアが23.8%でトップだが、国内ではここまで多くないのではないか
- おそらく、2～4位の1280x720(HD), 1920x1080(FullHD), 1136x640(iPhone5)あたりが主なターゲットのはず
- Android端末一覧 http://ja.wikipedia.org/wiki/Android端末一覧 と併用すると良い。正確性の保証はないが、概ね正しいはず

前の統計結果では、800x480のシェアが23.8%でトップになっていますが、これは日本国内の実感とだいぶ違います。
日本では、おそらく、2～4位のHD解像度からフルHD解像度あたりが現在の主なターゲットになりそうです。
WikipediaにAndroid端末一覧というページがあるので、これと併用するのが良いと思います。
http://ja.wikipedia.org/wiki/Android端末一覧
Wikipediaなので正確性の保証はありませんが、概ね正しいはずなので傾向を掴むのに役立つはずです。

Tips - #07 カタログスペックと実測値の乖離

Xperia arc (SO-01C)

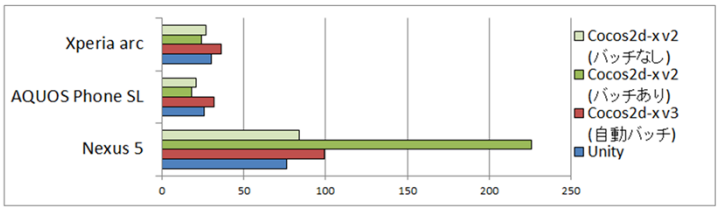
Snapdragon MSM8255 1GHz(1 Core) 480×854 (2011春発売)

AQUOS PHONE SL (IS15SH)

Snapdragon S2 MSM8655T 1.4GHz(1 Core) 540×960 (2012夏発売)

原因は不明

- 省電力モードを完全にOFFにできないため？
- プリインストールアプリの問題？
- メーカーのチューニングの差？



	Cocos2d-x v2 (バッチなし)	Cocos2d-x v2 (バッチあり)	Cocos2d-x v3 (自動バッチ)	Unity
Xperia arc	27個	24個	36個	30個
AQUOS PHONE SL	21個	18個	32個	26個
Nexus 5	84個	226個	99個	76個

カタログスペックと実測値の乖離についてお話しします。

第２部のベンチマークテストでは、Androidの旧世代機の代表として、Xperia arcの値を使いましたが、実はほかにも測定を行っていました。

AQUOS PHONE SLという機種です。Xperia arcより1年以上新しくて、スペックも明らかに上です。画面解像度は少し高い程度です。

ところが、ベンチマーク結果はご覧のように、Xperia arcとどっこいどっこいか、少なくともこのテスト内容では若干悪いくらいです。

原因は追及できていないのですが、省電力モードを完全にOFFにはできないようなので、そのあたりが原因なのかも知れません。

実際の性能とカタログスペックは必ずしも比例しないという実例として、ご紹介しました。

Tips - #08 プロファイラ

- Instruments
- Android Debug Monitor
- OpenGL ES Trace
- Trepn Profiler

OPTPiX WebTechnology® 89

SpriteStudioのサポートをしていると良く頂くのが、描画性能が出ないというお問い合わせです。

いろいろとやりとりしていてわかるのは、意外とプロファイラを使っていない方が多いということです。

プロファイラを使えば、どこにボトルネックがあるのかがわかり、対策が立てやすくなります。

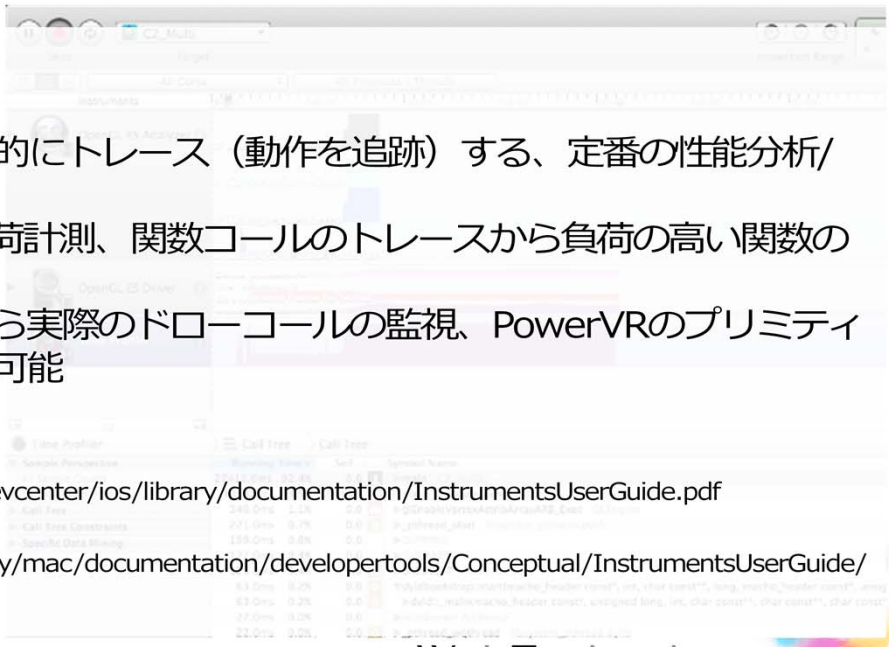
ぜひ、プロファイラを活用しましょう。

ここでは、４種類のプロファイラをご紹介します。

今回はご紹介しませんが、Unity Proは標準でプロファイラを持っていますので、それを活用してください。

Tips - #08 プロファイラ

Instruments



- OS XやiOSのコードを動的にトレース（動作を追跡）する、定番の性能分析/テストツール
- TimeProfilerからCPU負荷計測、関数コールのトレースから負荷の高い関数の特定が短時間で可能
- OpenGL ES Analyzerから実際のドローコールの監視、PowerVRのプリミティブのバッチングの監視も可能

- 日本語情報(少し古い)
  https://developer.apple.com/jp/devcenter/ios/library/documentation/InstrumentsUserGuide.pdf
- 英語(最新)
  https://developer.apple.com/library/mac/documentation/developertools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

OPTPiX WebTechnology® 90

Instrumentsは、iOSの定番プロファイラです。

OS標準のものですが、非常に高機能で、ボトルネックの特定と解消に有用なツールです。

日本語情報(少し古い)
https://developer.apple.com/jp/devcenter/ios/library/documentation/InstrumentsUserGuide.pdf

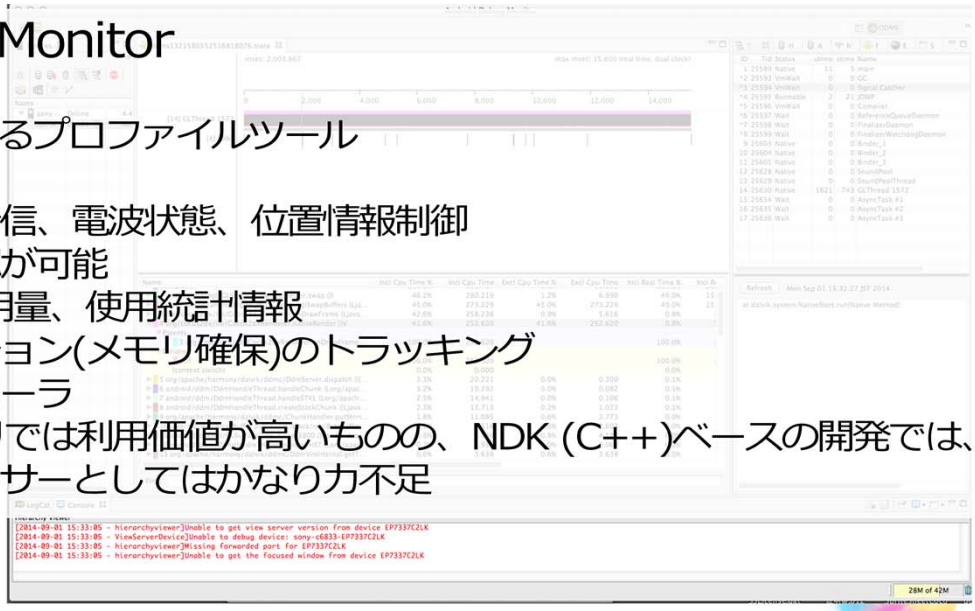
英語(最新)
https://developer.apple.com/library/mac/documentation/developertools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

Tips - #08 プロファイラ

Android Debug Monitor

Android SDKに含まれるプロファイルツール

- エミュレータへの着信、電波状態、位置情報制御
- スレッドの状態確認が可能
- Heap(メモリ)の使用量、使用統計情報
- メモリ・アロケーション(メモリ確保)のトラッキング
- ファイルエクスプローラ
- Javaベースのアプリでは利用価値が高いものの、NDK (C++)ベースの開発では、関数コールのトレーサーとしてはかなり力不足



OPTPiX WebTechnology® 91

Android Debug Monitorは、Android SDKに含まれているので、一応定番プロファイラですが、ほかのツールと組み合わせて使うのが効果的です。

特に、NDKでC++ベースの開発には非力なようです。

NDKベースの開発経験のある弊社のプログラマに聞いたところ、ほとんど役に立たないのでPCでデバッグして組み込んだ、とのことでした。

Tips - #08 プロファイラ

Android用のさまざまなプロファイラ

- OpenGL ES Trace
- チップメーカー毎に提供されているツール
  - Qualcomm Snapdragon Performance Visualizer
  - Qualcomm Trepn Profiler
  - PerfHUD
  - Mali Graphics Debugger
  - Power VR Performance Analysis Utilities

OPTPiX WebTechnology® 92

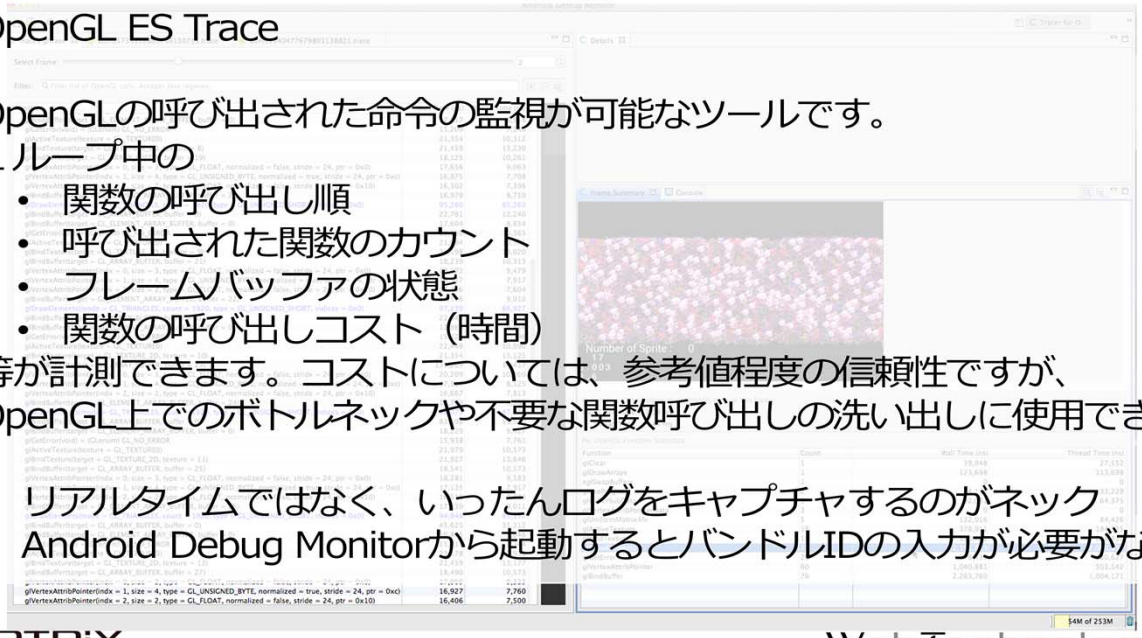
Androidでは、Android Debug Monitor のほかに、このようなさまざまなプロファイラが存在します。

OpenGL ESのプロファイラのほか、各チップセットごとにプロファイラが提供されています。

このなかから、OpenGL ES TraceとQualcomm Trepn Profiler についてこのあと簡単に触れます。

Tips - #08 プロファイラ

OpenGL ES Trace



OpenGLの呼び出された命令の監視が可能なツールです。
１ループ中の
- 関数の呼び出し順
- 呼び出された関数のカウント
- フレームバッファの状態
- 関数の呼び出しコスト（時間）

等が計測できます。コストについては、参考値程度の信頼性ですが、
OpenGL上でのボトルネックや不要な関数呼び出しの洗い出しに使用できます。

- リアルタイムではなく、いったんログをキャプチャするのがネック
- Android Debug Monitorから起動するとバンドルIDの入力が必要がなく便利

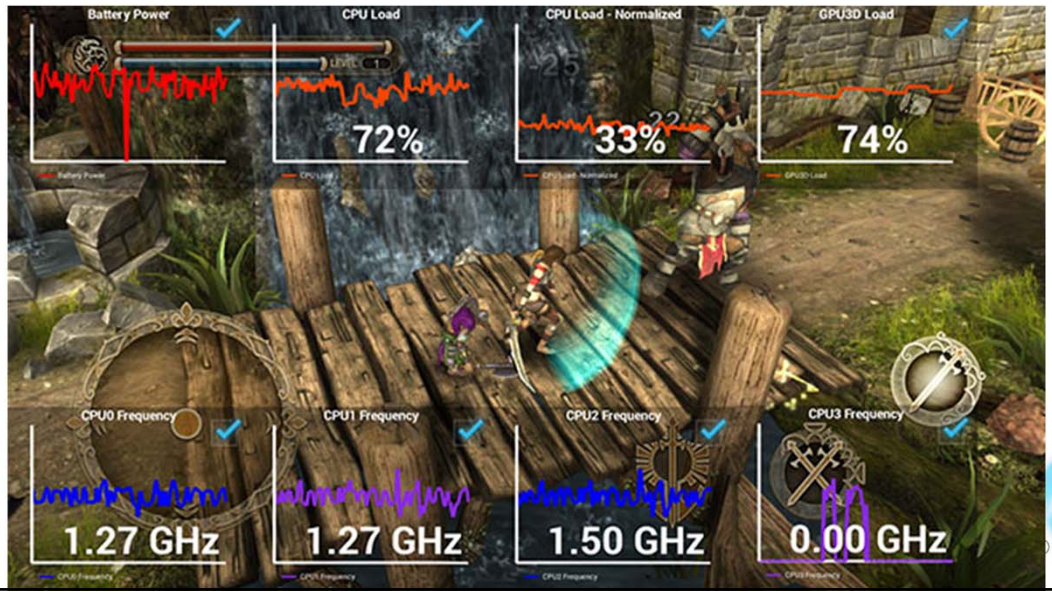
OPTPiX WebTechnology® 93

AndroidでOpenGLのプロファイリングを行う場合の定番ツールです。

OpenGL ES Traceは、Android Debug Monitorから起動すると便利なのですが、たまにこのことをご存知ない方が苦労しているようなので、ご参考まで。

Tips - #08 プロファイラ

Qualcomm Trepn Profiler



OPTPiX

94

Android端末に常駐させるタイプのプロファイラーで、こんなふうに画面に表示されます。

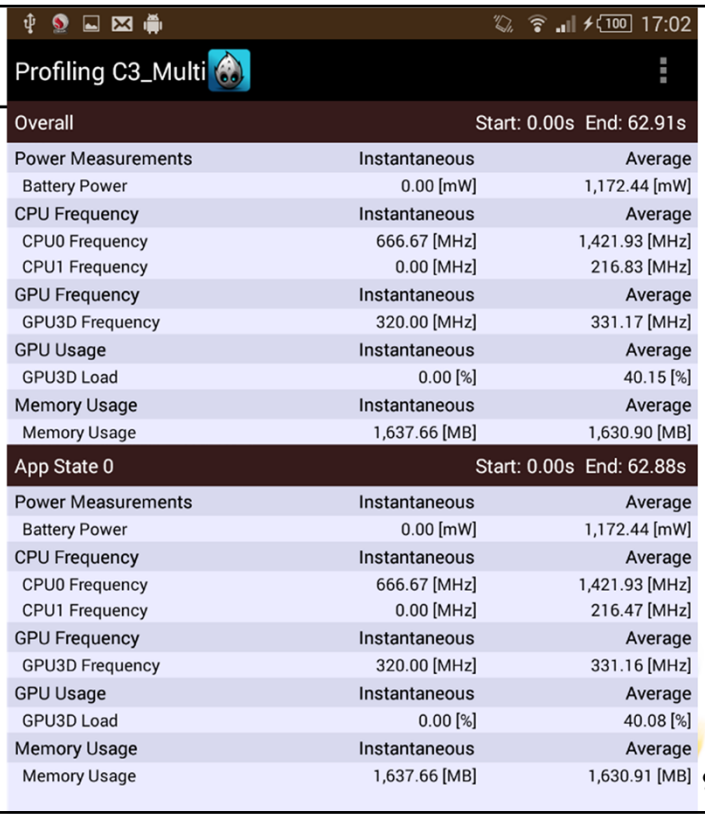
当然ですが、Snapdragon端末のみで動作します。

バッテリー使用量、CPUのコア毎の周波数と負荷率、GPUの負荷率、メモリー使用量、その他ジャイロ等のハードウェアの情報がリアルタイムで計測できます。

ただし、プログラムコードのプロファイルはできません。実行時のパフォーマンス測定に使用するものです

Tips - #08 プロファイラ

Qualcomm Trepn Profiler



OPTPiX

95

詳細は、こういう画面でも確認することができます。

最後に

手戻りを起こさないためのポイント

- スマートフォンは、機種による性能差が非常に大きい
- エンジンによる性能差は、それに比較すれば小さいほうなので、目的にあったものを選択すべし
- ベンチマークは重要。ただし、ピーク値に注意
- 適切な画像リソースを使おう
- プロファイラを使おう

ご清聴ありがとうございました。