



**GAME CREATORS
CONFERENCE**

コンシューマ畑の人間が本気を出してスマホに取り組んでみた

(グラフィック編)

株式会社ロジカルビート 堂前 嘉樹

自己紹介

- 株式会社ロジカルビートの代表取締役・プログラマー。
 - 2016年5月に法人設立。新横浜にあります。
 - ゲームプログラミング系の会社です。
 - スタッフ数6名。（2017/2）
- 個人としてはグラフィック系のプログラムが主戦場。
 - 関連作は「いけにえと雪のセツナ」（スクウェア・エニックス）など。
 - 書籍に「ゲームを動かす技術と発想」「ゲームを動かす数学・物理」がある。



関西！！

自己紹介（関西編）

- 1998年にコナミコンピュータエンタテインメント神戸に入社。
 - ポートアイランド（神戸）にオフィスがありました。
 - ゲームボーイやゲームボーイアドバンスのゲームを作っていました。
- 2003年にナムコに入社。（大阪開発）
 - 知らない人が多いですけど、大阪にも開発室がありました。
 - そこではPSPなど向けのゲームを作っていました。
- 2006年に関西を離れて、2008年に関東へ。

經歷

いけにえと 雪のセツナ

「いけにえと雪のセツナ」

©2016 Tokyo RPG Factory Co., Ltd. All rights reserved.



GB
GBA
PS2
PSP
PS3
XBOX360
WiiU
3DS
PS Vita
PS4

GAME CREATORS CONFERENCE'17

コンシューマ主体



GAME CREATORS CONFERENCE'17

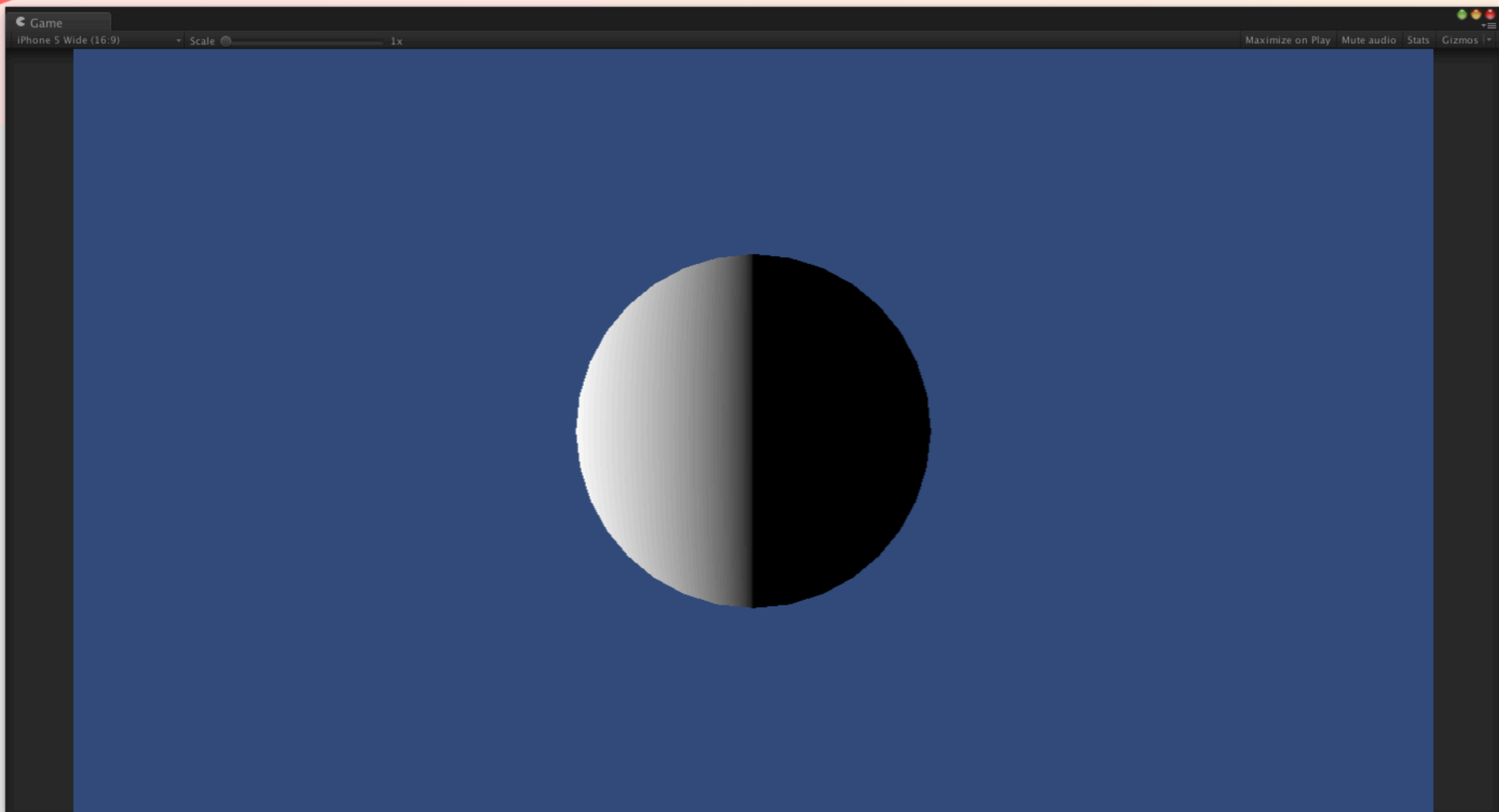
スマホにようやく
関わる

本日のテーマ

コンシューマー→スマホで戸惑ったこと

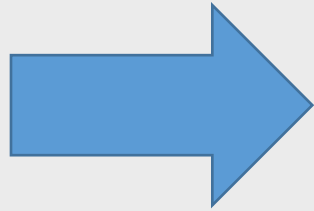
描画設定について

Unity（のスマホ）でのリニアワークフロー処理

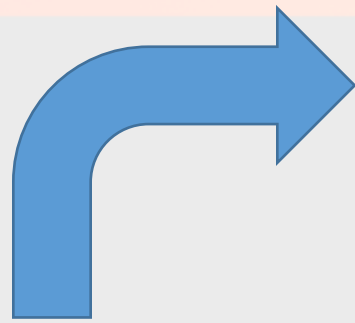


GAME CREATORS CONFERENCE'17

コンシューマー→スマホで 戸惑ったこと



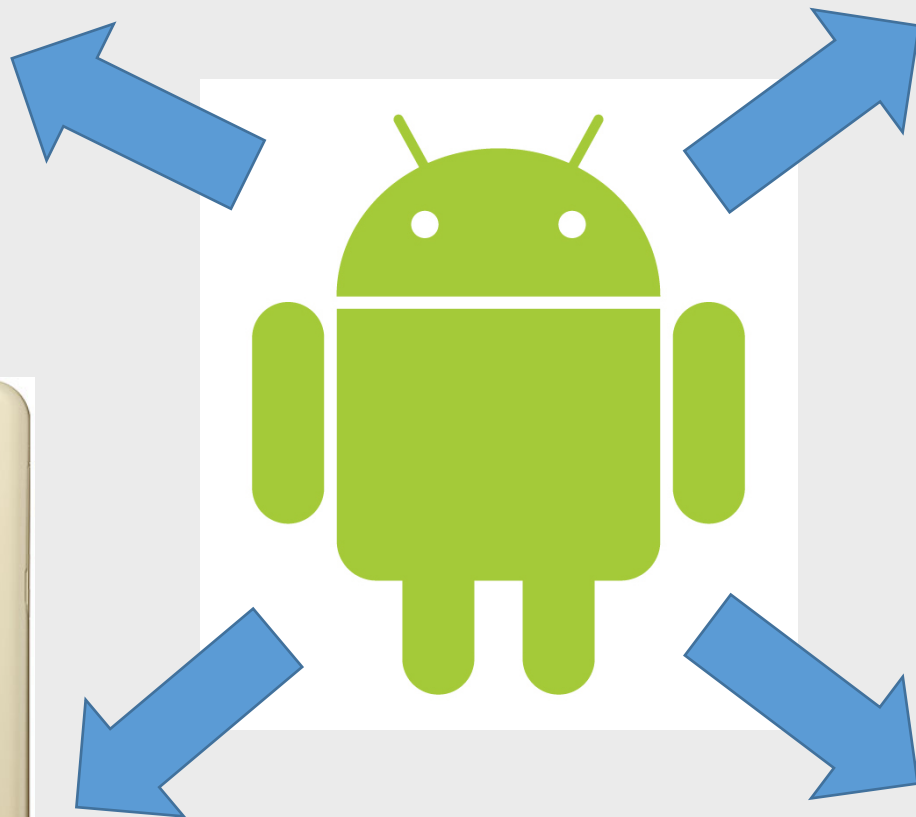
GAME CREATORS CONFERENCE'17



PS4™



GAME CREATORS CONFERENCE '17



GAME CREATORS CONFERENCE'17

コンシューマ機とスマホ

- コンシューマ機はプラットフォームに対して端末の性能の種類は（原則として）1つのみ。
 - 例えばPS3なら薄型などはあるものの、性能としては一種類。
 - 例外としてGBに対するGBC、PS4に対するPS4 Proといった同一プラットフォームで性能の違う端末があることがあるが、それでも二種類程度？
- スマホの場合はプラットフォームに対して端末数が多い！
 - iPhone、iPad（iOS系）は年々新端末が出て、性能向上されている。
 - Androidは本当に多すぎる・・・。

端末が違って
異なってくること
(一例)

CPUとGPU
搭載メモリ
画面のアスペクト比
画面解像度
対応グラフィックスAPIのバージョン

-
-
-

コンシューマの場合

- コンシューマ機はそのハードウェアで実行できればOK。
 - プログラムやシェーダを組んで実行して、動けばあまり気にしない。
 - 確認が非常に楽！
- 処理負荷やメモリ。
 - 固定ハードなので、限界まで挑戦（？）できる。
 - 逆に言えば、限界まで挑戦させられる、とも言える。

対応がそんなに苦ではないと思った部分

- メモリ。
 - 機種によって差がありすぎるが、そもそもそんなに使わないようにする。
(エンジン側がどう確保するかわからないというのもある。)
- 画面解像度とアスペクト比。
 - 端末によっては無茶苦茶な解像度のものもあるらしい。
が、別にRenderTextureを持って、内部解像度を固定するのが良い。
 - 端末の物理的面積から、720pでも問題ないと思う。
 - アスペクト比は、Unityの画角が垂直であることを考慮。
 - 水平画角にしたい場合、変換計算を挟むと良い。

大変だったこと！



GAME CREATORS CONFERENCE'17

Mali™-T658

Inter-core Task Management

Shader

Cores

Shader

Cores

Memory Management Unit

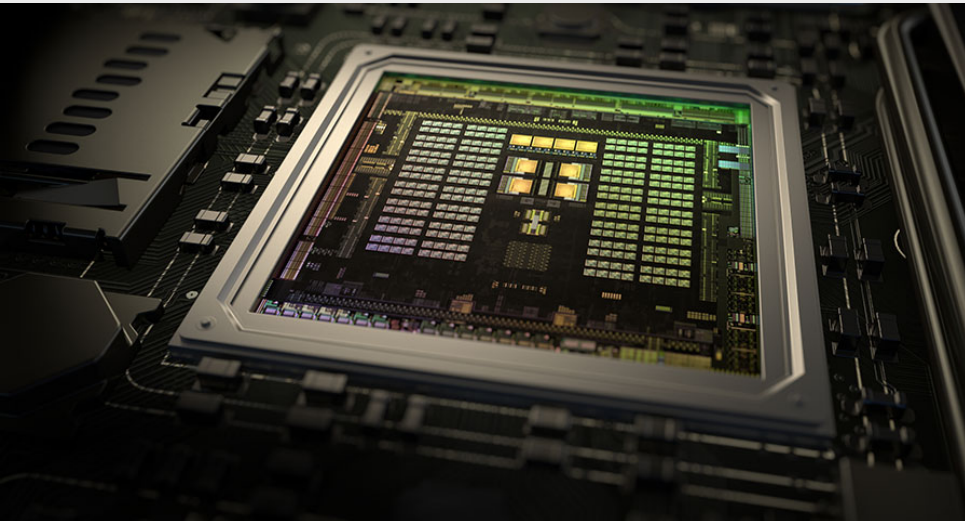
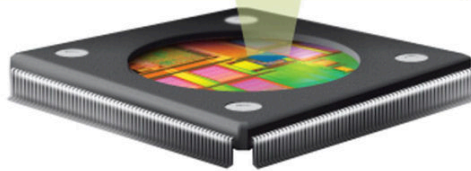
Memory Management Unit

Level 2 Cache / SCU

Level 2 Cache / SCU

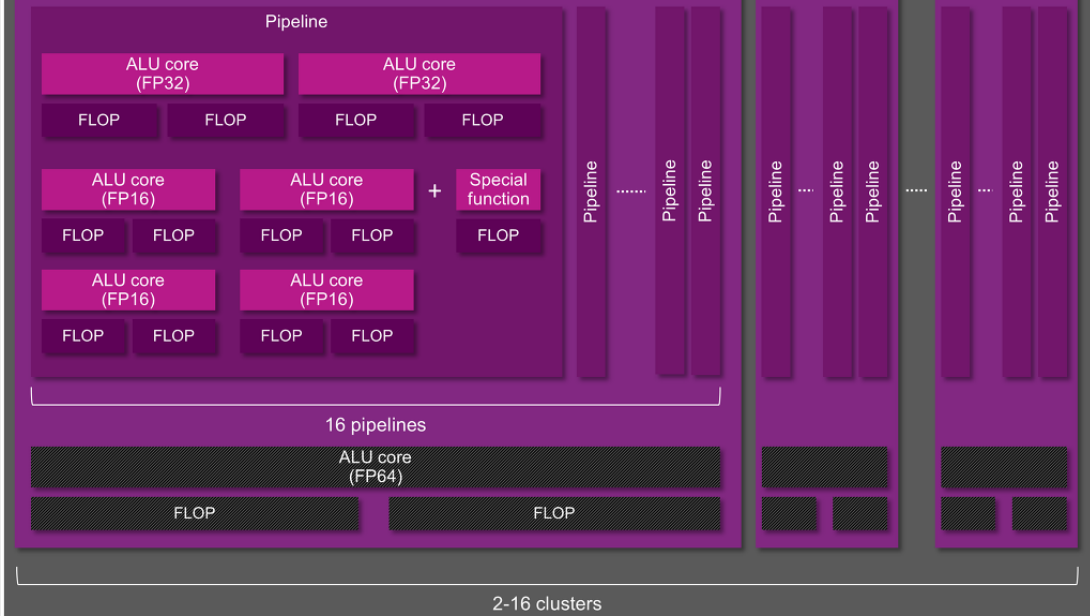
AMBA® 4 ACE-Lite™
Interface

AMBA® 4 ACE-Lite™
Interface



PowerVR Series7XT Unified Shading Cluster Array

PowerVR Series7XT USC



Optional

GAME CREATORS CONFERENCE'17

グラフィックスAPIと GPUの違い

OpenGL ES 2.0

OpenGL ES 3.0

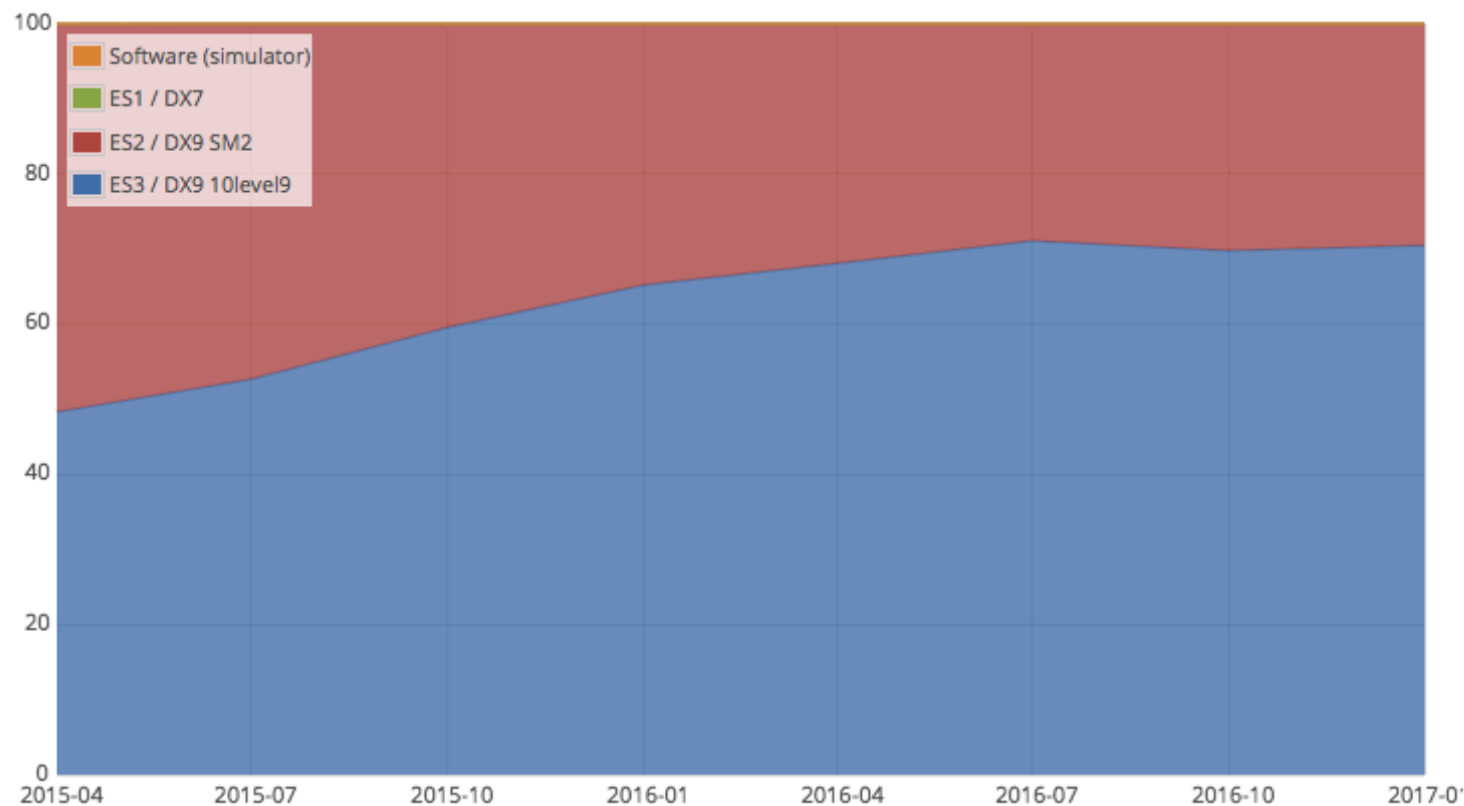
現在の状況

SHADER GENERATIONS

Shows shader technology supported by the GPU. This does not mean it can actually be used though (e.g. DX10+ shaders can not be used on Windows XP with D3D9).

Top on 2017-01:

- ES3 / DX9 10level9: **70.5%**
- ES2 / DX9 SM2: **29.5%**
- [Others \(click to show\)](#): 0.0%



※ 「Unity Hardware Statistics」 より

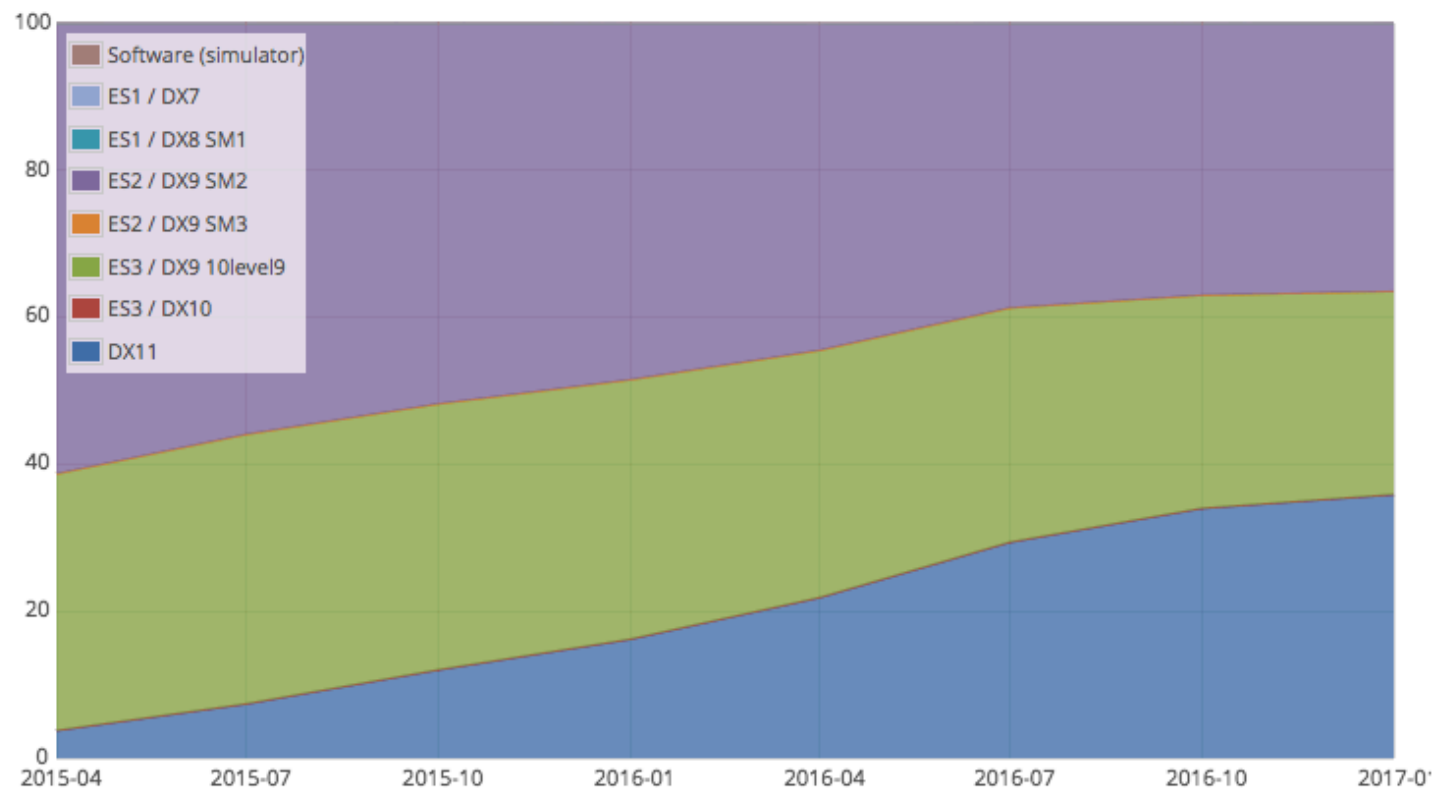
<http://hwstats.unity3d.com/mobile/gpu-ios.html>

SHADER GENERATIONS

Shows shader technology supported by the GPU. This does not mean it can actually be used though (e.g. DX10+ shaders can not be used on Windows XP with D3D9).

Top on 2017-01:

- ES2 / DX9 SM2: **36.4%**
- DX11: **35.9%**
- ES3 / DX9 10level9: **27.6%**
- [Others \(click to show\)](#): 0.1%



※ 「Unity Hardware Statistics」 より

<http://hwstats.unity3d.com/mobile/gpu-android.html>

OpenGL ES Hardware Processors

All currently shipping iPhone, iPad, and iPod Touch devices use the PowerVR SGX graphics processors (535, 543, and 554). These processors support [1.1](#) by efficiently implementing the fixed-function pipeline using shaders.

Table 1-1 describes each processor-device combination.

Table 1-1 iOS graphics hardware compatibility

Device compatibility	PowerVR hardware platform	OpenGL ES versions supported
iPod Touch (3rd and 4th generations)	SGX 535	1.1, 2.0
iPod Touch (5th generation)	SGX 543	1.1, 2.0
iPhone 3GS, iPhone 3GS (China), and iPhone 4	SGX 535	1.1, 2.0
iPhone 4S and iPhone 5	SGX 543	1.1, 2.0
iPad Wi-Fi and iPad Wi-Fi + 3G	SGX 535	1.1, 2.0
iPad 2 Wi-Fi, iPad 2 Wi-Fi + 3G, iPad 3 Wi-Fi, and iPad 3 Wi-Fi + 3G	SGX 543	1.1, 2.0
iPad 4 Wi-Fi and iPad 4 Wi-Fi + 3G	SGX 554	1.1, 2.0
iPad Mini Wi-Fi and iPad Mini Wi-Fi + 3G	SGX 543	1.1, 2.0

※https://developer.apple.com/library/content/documentation/OpenGL/Conceptual/OpenGLHardwarePlatformGuide_iOS/OpenGLPlatforms/OpenGLPlatforms.html

Develop > API Guides > Animation and Graphics

OpenGL ES

Android includes support for high performance 2D and 3D graphics with the Open Graphics Library (OpenGL®), specifically, the OpenGL ES API. OpenGL is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware. OpenGL ES is a flavor of the OpenGL specification intended for embedded devices. Android supports several versions of the OpenGL ES API:

- OpenGL ES 1.0 and 1.1 - This API specification is supported by Android 1.0 and higher.
- OpenGL ES 2.0 - This API specification is supported by Android 2.2 (API level 8) and higher.
- OpenGL ES 3.0 - This API specification is supported by Android 4.3 (API level 18) and higher.
- OpenGL ES 3.1 - This API specification is supported by Android 5.0 (API level 21) and higher.

✂<https://developer.android.com/guide/topics/graphics/opengl.html>

OpenGL ES 2.0と3.0のシェア

- 現時点でのスマホのグラフィックスAPIの境界線がES2.0と3.0になっていると言えそう。
- シェア的にはiOS、Androidとも、ES2.0は30%程度強。
- ES2.0と3.0の境界線となる端末。
 - iOS : iPhone5 (2012年・ES2.0) とiPhone5s (2013年・ES3.0)
 - Android : Android4.3以上でES3.0が使える。
 - 2011、2012年代の端末も意外と対応している。

OpenGL ES 2.0と3.0の性能差

- ES2.0は、シェーダモデル2.0相当？
 - PS3は3.0相当なので、ちょっと物足りない印象。
 - 実際、MRTやVTF（Vertex Texture Fetch）などは利用出来ない模様。
- ES3.0は、シェーダモデル3.0相当？
 - 理論的にはPS3くらいのことにはできるはず。
 - 実際、扱ってて不便に感じた事はない。

GPUの違い

Mali™-T658

Inter-core Task Management

Shader

Cores

Shader

Cores

Memory Management Unit

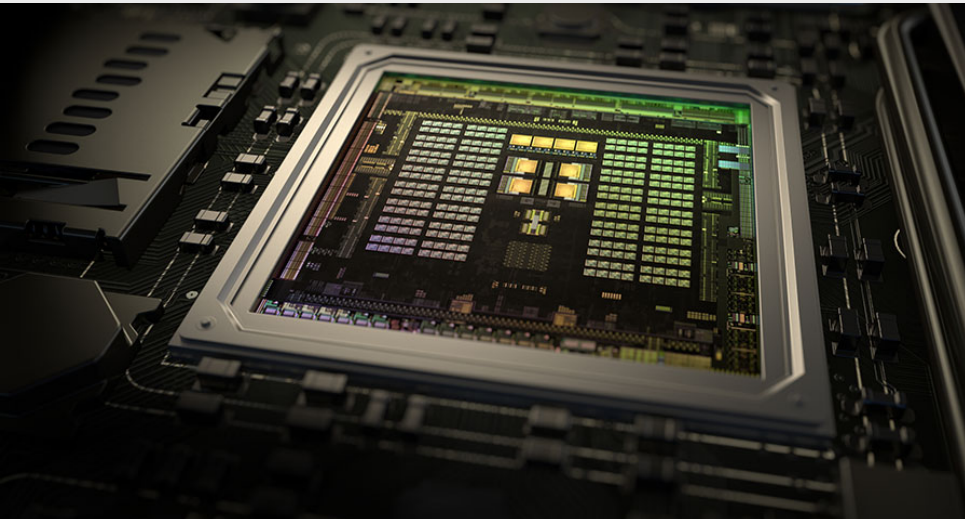
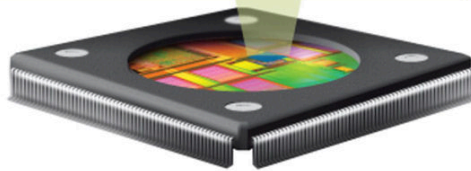
Memory Management Unit

Level 2 Cache / SCU

Level 2 Cache / SCU

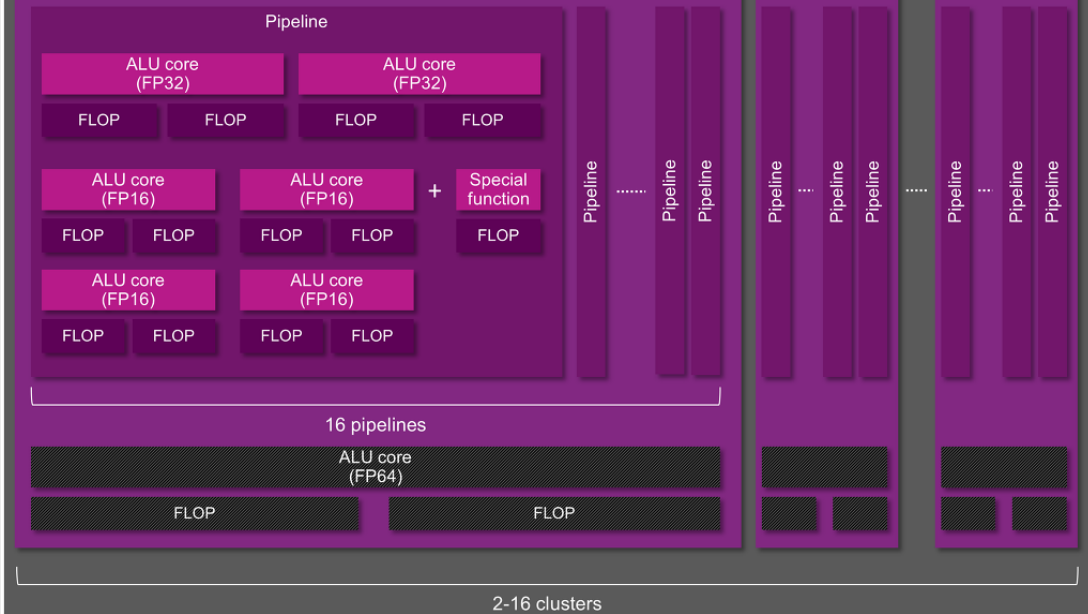
AMBA® 4 ACE-Lite™
Interface

AMBA® 4 ACE-Lite™
Interface



PowerVR Series7XT Unified Shading Cluster Array

PowerVR Series7XT USC



GAME CREATORS CONFERENCE'17

一番困るのは
テクスチャ

GPUとテクスチャフォーマット

- フルカラー形式（RGBA8888）は、GPU関わらずまず間違いなく利用できるはず。
 - だが、積極的に使う機会は少ないので、圧縮フォーマットがメイン。
- 圧縮フォーマットは、PCやPS3などのコンシューマ機ではDXTフォーマットが一般的によく使われる。
 - 「DXT1もしくはDXT5で～」と話を進めれば良かったので楽。
 - 画質も読みやすい。

テクスチャは
GPUに依存

iOSのテクスチャ

PVRTC

iosのテクスチャ

- iOS系はPowerVR系のGPUを搭載している。
そしてPowerVRで処理しやすい「PVRTC」というのがメイン。
 - ちなみにPowerVRを搭載しているAndroidでも利用可。
- アルファチャンネルも扱える。
- メリットとデメリット。
 - メリット：高速である、圧縮率が高いのがある。（2bpp）
 - デメリット：正方形である必要がある、品質がよろしくない。。

Androidのテクスチャ

(主に)
ETC、ETC2

Androidのテクスチャ

- Androidは端末によってGPUがバラバラ！！
それにより扱えるフォーマットもマチマチ！！
 - DXT、ATC、ASTC、PVRTCなど。
 - GPUによって使える使えないがある。
 - 扱いづらいのは想像に難くない。。。
- そんな中、安定して利用出来るのが「ETC」系。
 - ETC、ETC2というフォーマット。
 - ETCは不透明形式（アルファ無し）、ETC2はアルファも扱える。
 - DXT1とDXT5のようなノリ。

実はテクスチャは
グラフィックスAPIにも
依存する！

OpenGL ESバージョン	利用できるETCフォーマット
2.0	ETC
3.0	ETC ETC2

OpenGL ESとETC系テクスチャ

- OpenGL ES2.0ではETC2が利用出来ない！！
 - 使えるのはETCのみ。
- ETCのみだと何が困るかということ、半透明の処理。
 - ETC自体は不透明のみサポート。

OpenGL ESとETC系テクスチャ

- 半透明描画をしたい時は以下のいずれかの措置を取る。
- テクスチャをフルカラー形式にする。
 - 重いのでありえない・・・。
- シェーダ側で工夫する。
 - カラーとアルファ部分でテクスチャを分ける。
(アルファはグレースケールにして対応。)
 - この方法しかなさそう。

ETCで
半透明描画！



カラーのみ



アルファのみ
(グレースケール)



一旦
まとめます！！

プラットフォーム		メイン圧縮フォーマット
iOS (iPhone、 iPad)		PVRTC
Android	OpenGL ES 2.0	ETC
	OpenGL ES 3.0	ETC2

要点を
見極める

プラットフォーム		メイン圧縮フォーマット
iOS (iPhone、 iPad)		PVRTC
Android	OpenGL ES 2.0	ETC
	OpenGL ES 3.0	ETC2

様々な要因を考慮

- テクスチャは正方形にしておく！
 - PVRTCのため。
- PVRTCの品質がよろしくないのを知っておく。
 - 実機で見れるなら見ておく。
 - どうしても我慢できない時は16ビットディザ化なども考慮。
- AndroidのOpenGL ES2.0を考慮し、カラーとアルファを別テクスチャで持ち、シェーダも自作する。

これで解決！



カラーのみ



アルファのみ
(グレースケール)

シェーダ



これってかなり
めんどくさそう . . .

カラーとアルファを分離

- ETCが不透明のみということを考慮したら、「カラー＋アルファ」という構成が必須になる。
そしてそれはシェーダで対応することになる。
- 半透明を扱うシェーダは、全てこれを考慮しなければならない??
それって大変じゃないの??
 - ゲームタイトルによって変わるが、シェーダの種類は非常に多くなるはず。

課題を
残したまま・・・

描画設定について

プラットフォーム		メイン圧縮フォーマット
iOS (iPhone、 iPad)		PVRTC
Android	OpenGL ES 2.0	ETC
	OpenGL ES 3.0	ETC2

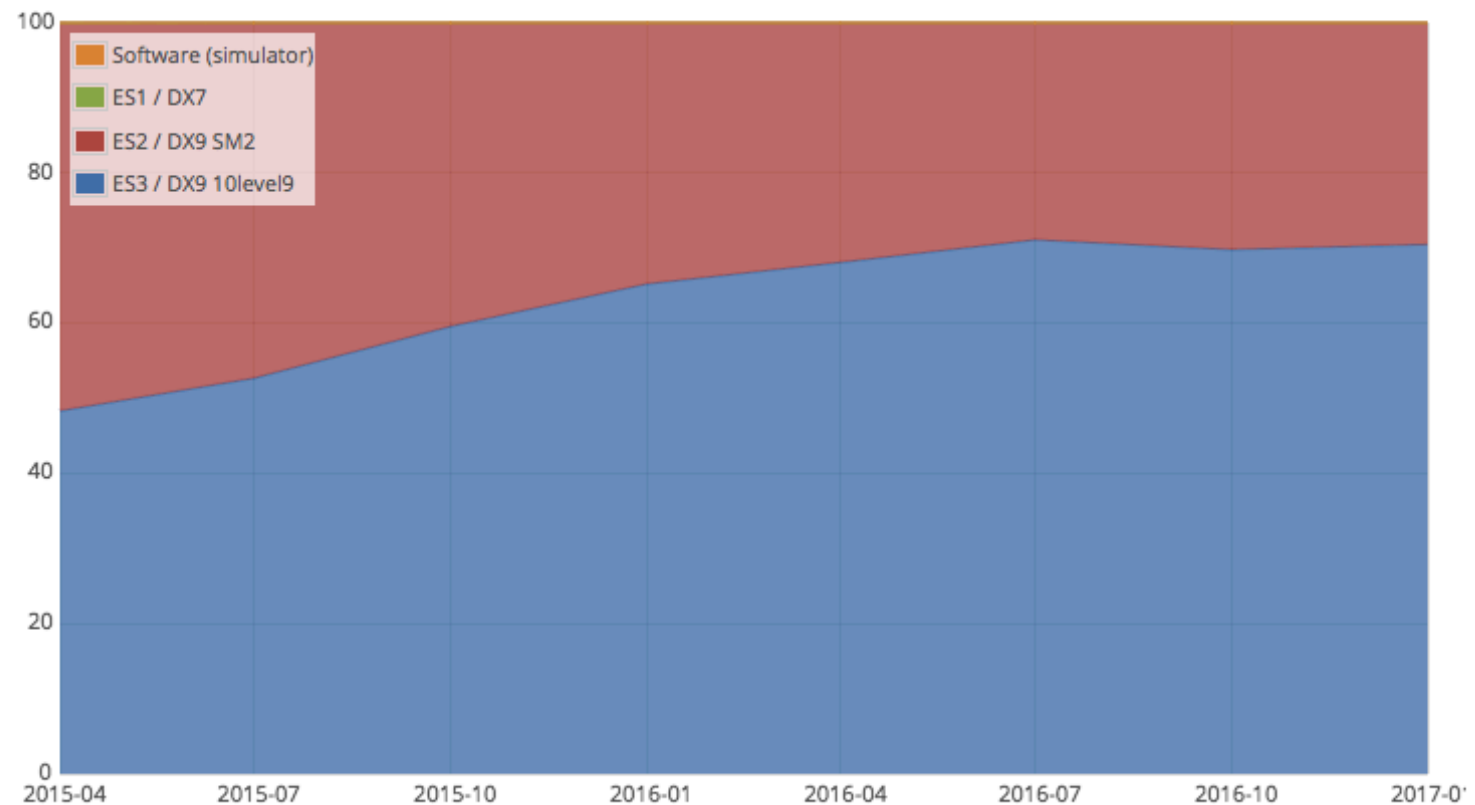
「OpenGL ES2.0」
というのをどう捉える？

SHADER GENERATIONS

Shows shader technology supported by the GPU. This does not mean it can actually be used though (e.g. DX10+ shaders can not be used on Windows XP with D3D9).

Top on 2017-01:

- ES3 / DX9 10level9: **70.5%**
- ES2 / DX9 SM2: **29.5%**
- [Others \(click to show\)](#): 0.0%



※ 「Unity Hardware Statistics」 より

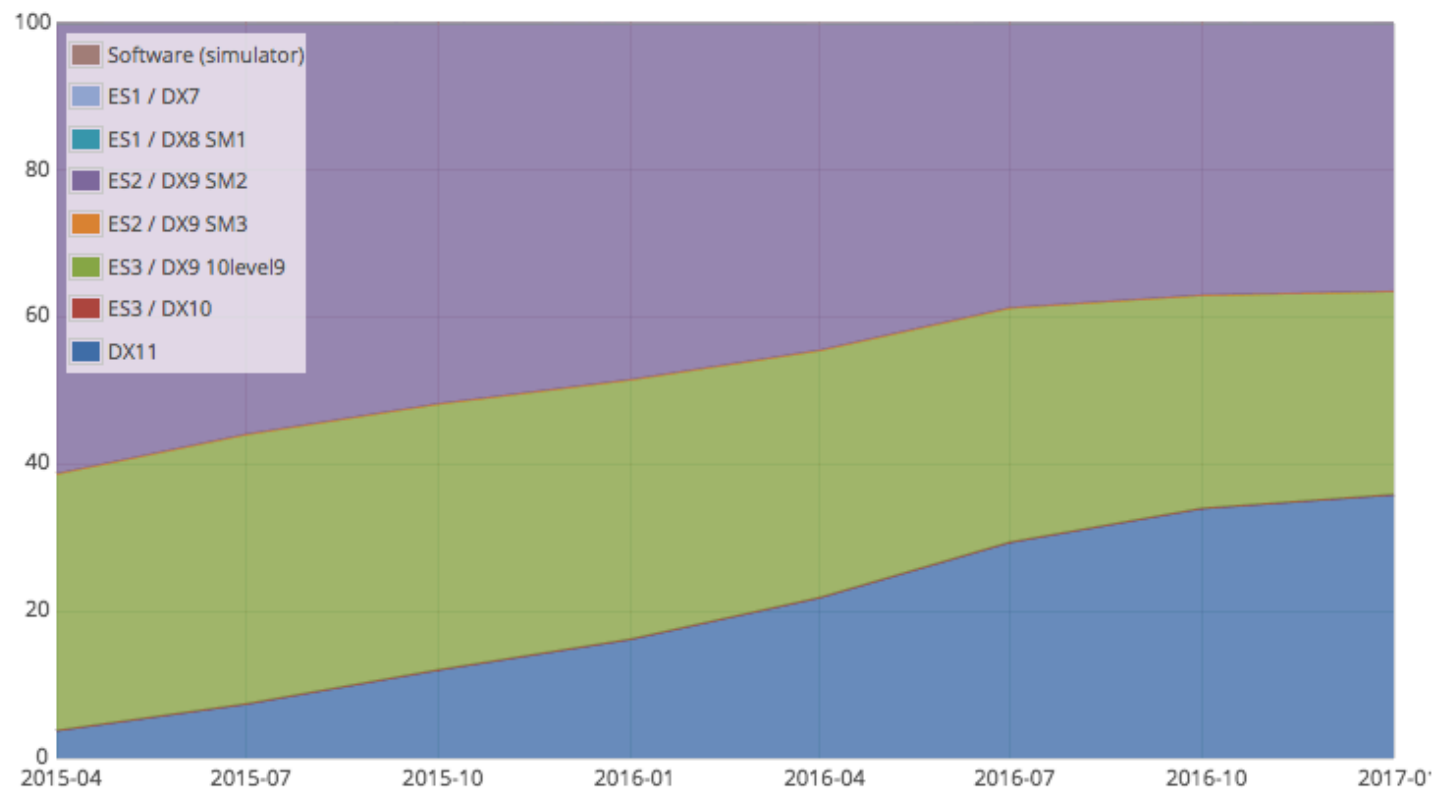
<http://hwstats.unity3d.com/mobile/gpu-ios.html>

SHADER GENERATIONS

Shows shader technology supported by the GPU. This does not mean it can actually be used though (e.g. DX10+ shaders can not be used on Windows XP with D3D9).

Top on 2017-01:

- ES2 / DX9 SM2: **36.4%**
- DX11: **35.9%**
- ES3 / DX9 10level9: **27.6%**
- [Others \(click to show\)](#): 0.1%



※ 「Unity Hardware Statistics」 より

<http://hwstats.unity3d.com/mobile/gpu-android.html>

OpenGL ES Hardware Processors

All currently shipping iPhone, iPad, and iPod Touch devices use the PowerVR SGX graphics processors (535, 543, and 554). These processors support [1.1](#) by efficiently implementing the fixed-function pipeline using shaders.

Table 1-1 describes each processor-device combination.

Table 1-1 iOS graphics hardware compatibility

Device compatibility	PowerVR hardware platform	OpenGL ES versions supported
iPod Touch (3rd and 4th generations)	SGX 535	1.1, 2.0
iPod Touch (5th generation)	SGX 543	1.1, 2.0
iPhone 3GS, iPhone 3GS (China), and iPhone 4	SGX 535	1.1, 2.0
iPhone 4S and iPhone 5	SGX 543	1.1, 2.0
iPad Wi-Fi and iPad Wi-Fi + 3G	SGX 535	1.1, 2.0
iPad 2 Wi-Fi, iPad 2 Wi-Fi + 3G, iPad 3 Wi-Fi, and iPad 3 Wi-Fi + 3G	SGX 543	1.1, 2.0
iPad 4 Wi-Fi and iPad 4 Wi-Fi + 3G	SGX 554	1.1, 2.0
iPad Mini Wi-Fi and iPad Mini Wi-Fi + 3G	SGX 543	1.1, 2.0

※https://developer.apple.com/library/content/documentation/OpenGLES/Conceptual/OpenGLSHardwarePlatformGuide_iOS/OpenGLESPlatforms/OpenGLESPlatforms.html

Develop > API Guides > Animation and Graphics

OpenGL ES

Android includes support for high performance 2D and 3D graphics with the Open Graphics Library (OpenGL®), specifically, the OpenGL ES API. OpenGL is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware. OpenGL ES is a flavor of the OpenGL specification intended for embedded devices. Android supports several versions of the OpenGL ES API:

- OpenGL ES 1.0 and 1.1 - This API specification is supported by Android 1.0 and higher.
- OpenGL ES 2.0 - This API specification is supported by Android 2.2 (API level 8) and higher.
- OpenGL ES 3.0 - This API specification is supported by Android 4.3 (API level 18) and higher.
- OpenGL ES 3.1 - This API specification is supported by Android 5.0 (API level 21) and higher.

✂<https://developer.android.com/guide/topics/graphics/opengl.html>

シェアなどから
総合的に判断

今回の アプリ動作基準 (仮)

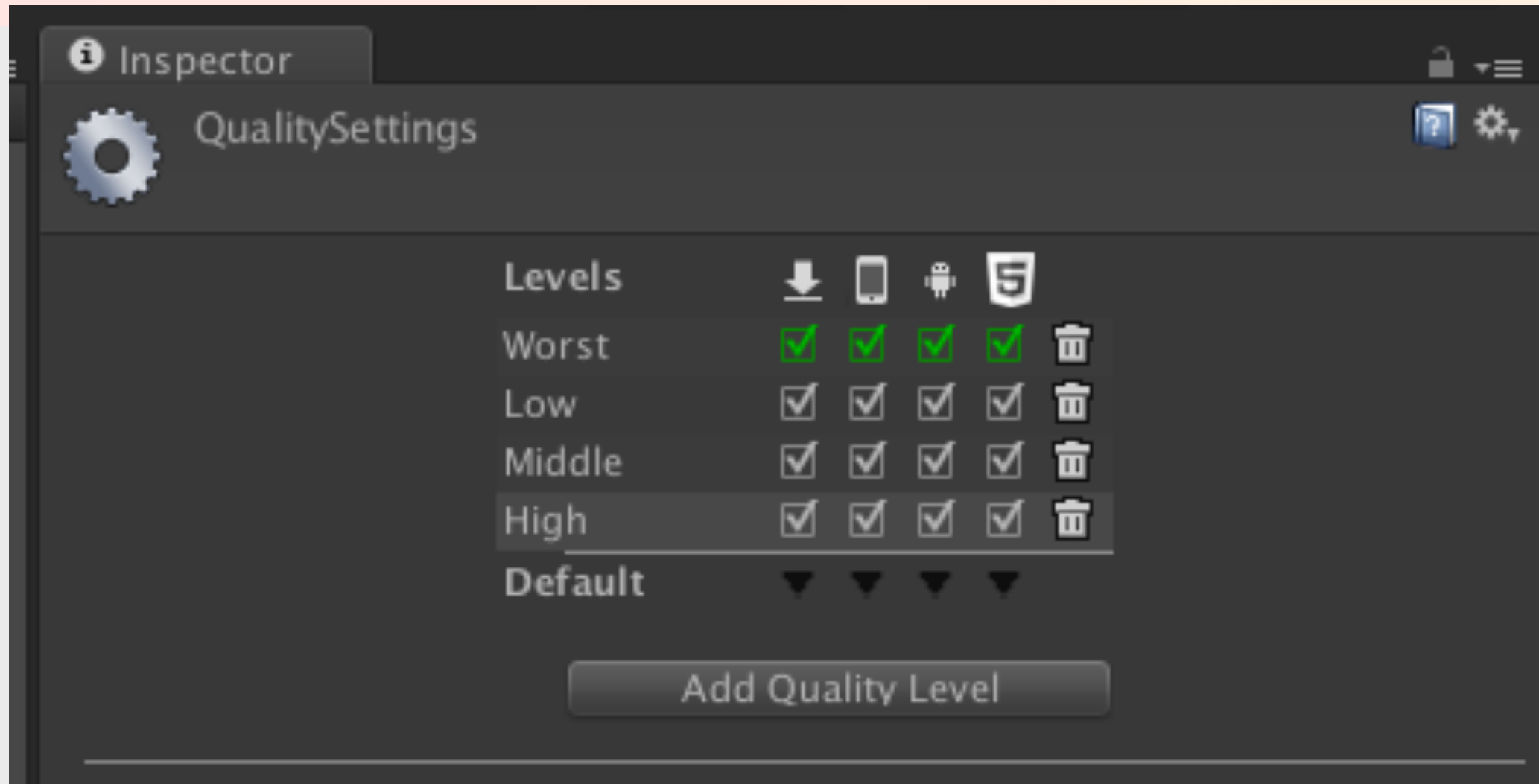
OpenGL ES 2.0でも動作はする

ES2.0では過敏に動けなくても仕方ない

ハイスペックマシンだと綺麗にサクサク動く

性能差を考慮して

アプリ内での 描画設定



GAME CREATORS CONFERENCE'17

描画設定を設ける

- アプリ内で描画設定を設ける。
 - 端末性能により、適したものを選んでもらう or アプリが提案する。
 - コンシューマ機ではこういうのが無いので新鮮。
 - とはいえスマホではよくある手法？
- 大雑把に4段階くらいに分ける。
 - Unityの場合、Quality Settingsがあるので連動しやすい。

どういう
設定にする？



GAME CREATORS CONFERENCE'17

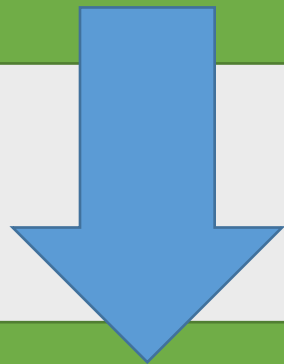
設定	内容	想定端末
Worst	2Dモード (3Dもちょっとだけ表示)	OpenGL ES2.0しか動かない端末
Low	3Dモード (低)	OpenGL ES3.0は動くが、低めのスペックの端末
Middle	3Dモード (中)	OpenGL ES3.0が動く現行機種
High	3Dモード (高)	将来のハイスペックマシン

設定	内容	想定端末
Worst	2Dモード (3Dもちょっとだけ表示)	OpenGL ES2.0しか動かない端末
Low	3Dモード (低)	OpenGL ES3.0は動くが、低めのスペックの端末
Middle	3Dモード (中)	OpenGL ES3.0が動く現行機種
High	3Dモード (高)	将来のハイスペックマシン

描画設定の一例

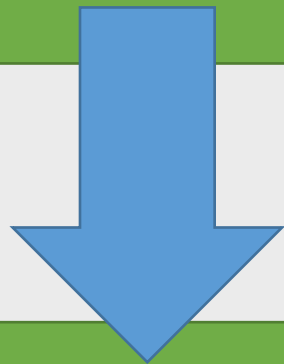
- 2Dモードと3Dモードを設ける。
 - 2Dモードは低スペックマシンを意図。
- 最上位バージョンを設けている。
 - 近い将来現れるであろう、高性能機向けに。
 - ここは夢を持ちたい。
- OpenGL ES2.0の場合。
 - 2Dモードへ強く誘導。
 - 3Dモードには入れるが、処理はだいぶ重い。
(3Dモードは非推奨という形。)

3Dオブジェクト描画



UI描画

3Dオブジェクト描画

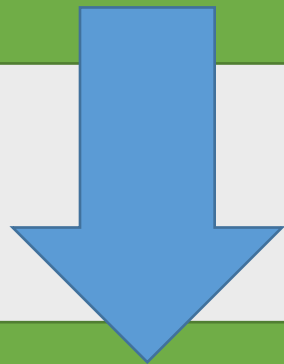


UI描画

- シェーダの種類：多い

- シェーダの種類：少ない

3Dオブジェクト描画

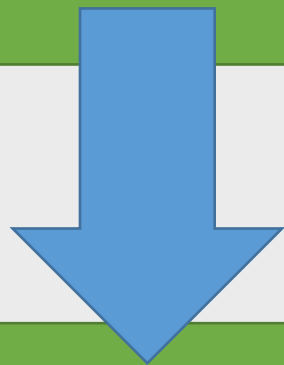


UI描画

- シェーダの種類：多い
- 2Dモードではほぼ利用しない

- シェーダの種類：少ない
- どのモードでも動作

3Dオブジェクト描画



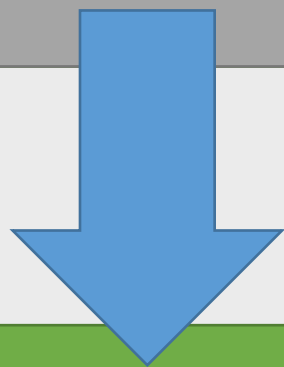
UI描画

- シェーダの種類：多い
 - 2Dモードではほぼ利用しない
- ↓
- 通常のテクスチャを利用！

- シェーダの種類：少ない
 - どのモードでも動作
- ↓
- 「カラー＋アルファ」テクスチャ

OpenGL ES 2.0では？

3Dオブジェクト描画



UI描画

- シェーダの種類：多い
 - 2Dモードではほぼ利用しない
- ↓
- 通常のテクスチャを利用！

↑フルカラー等をあえて利用。

- シェーダの種類：少ない
 - どのモードでも動作
- ↓
- 「カラー＋アルファ」テクスチャ

描画設定の狙い

- UIはETCを考慮し、「カラー＋アルファ」のテクスチャで作る。
 - シェーダも用意しなければならないが、絶対数は少ないはず。
 - 半透明テクスチャ（ETC2）が扱えたとしてもそうする。（管理の面で）
- 3Dに関しては、「カラー＋アルファ」の構成にはしない！
- OpenGL ES2.0では困るが、AssetBundleでフルカラーなどを用意＆ダウンロードしてそれを使ってしまう。
 - 「カラー＋アルファ」だとシェーダ対応が大変なので！
 - DXT等が使えるGPUならそれにする。

Unity（のスマホ）での リニアワークフロー処理



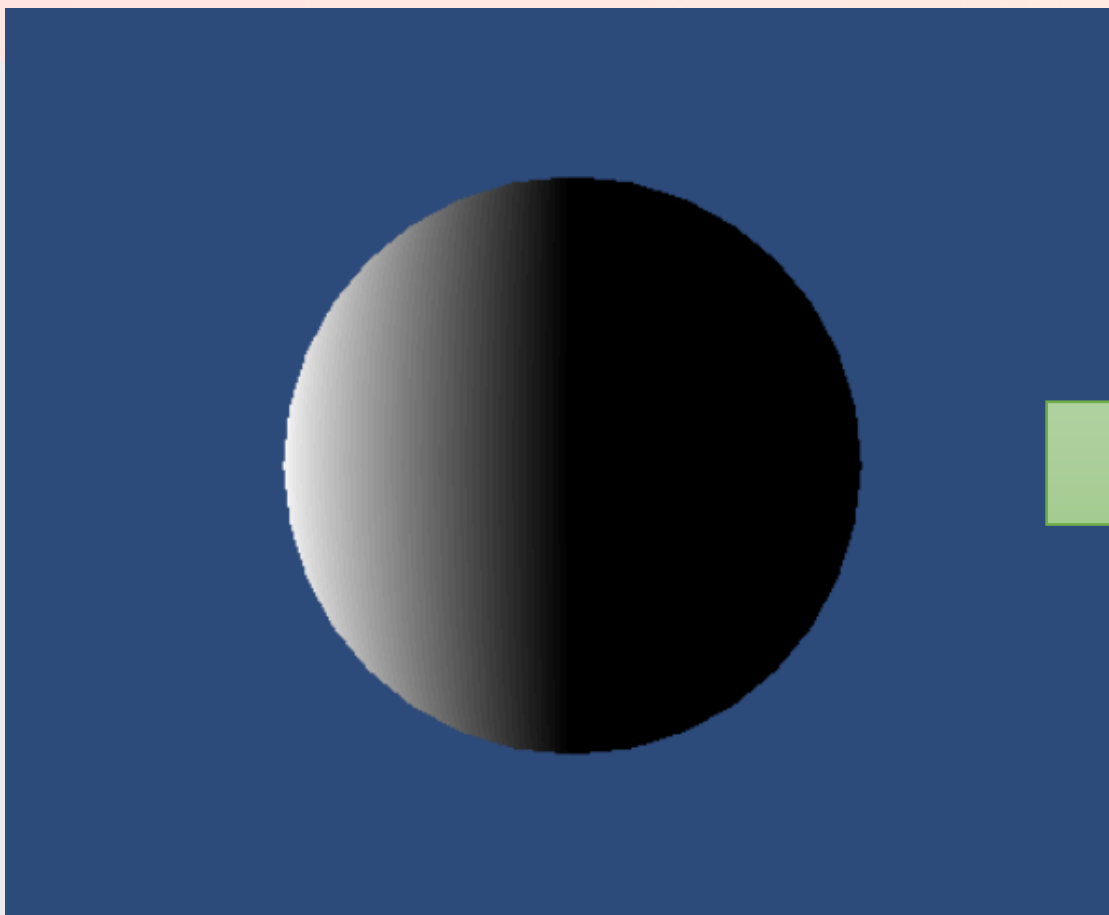
GAME CREATORS CONFERENCE '17



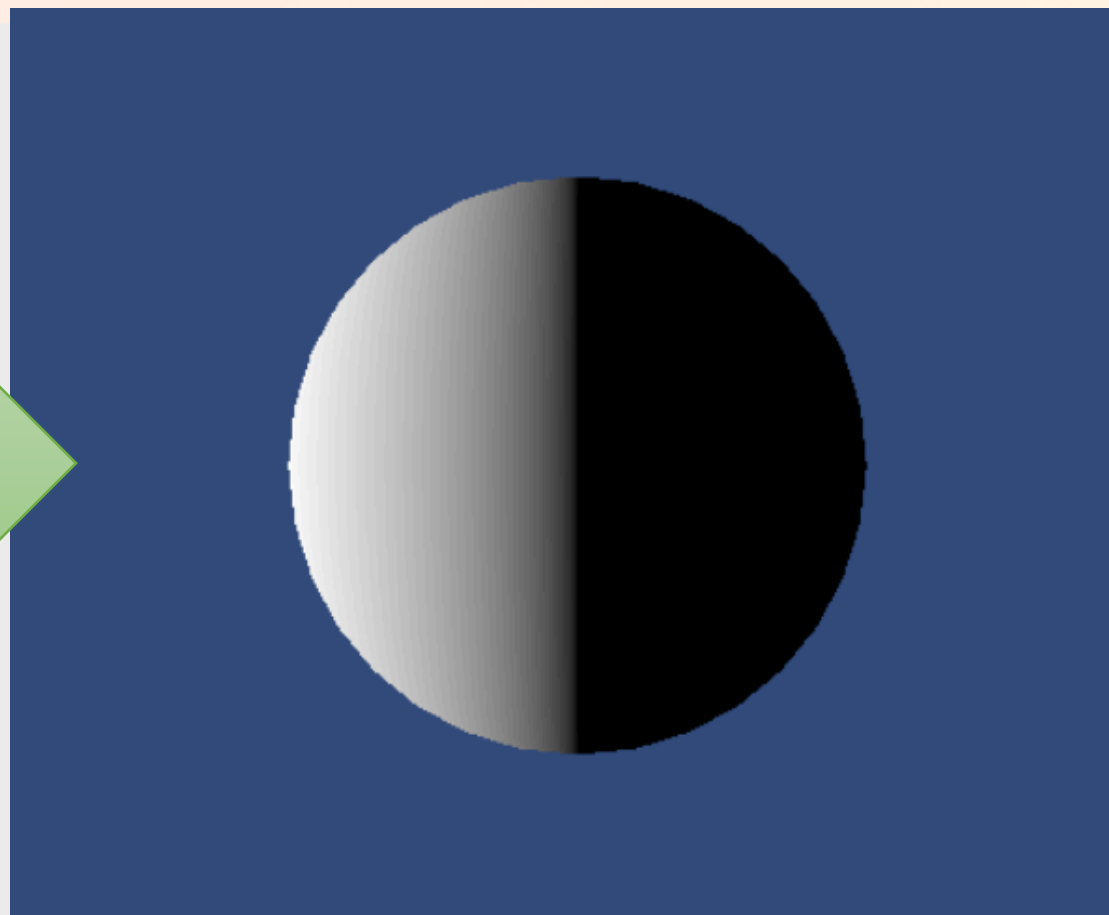
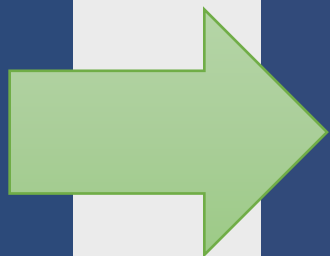
GAME CREATORS CONFERENCE'17

ゲームエンジンは
下に揃える傾向にある

今回の 技術チャレンジ！



※リニア対策なし



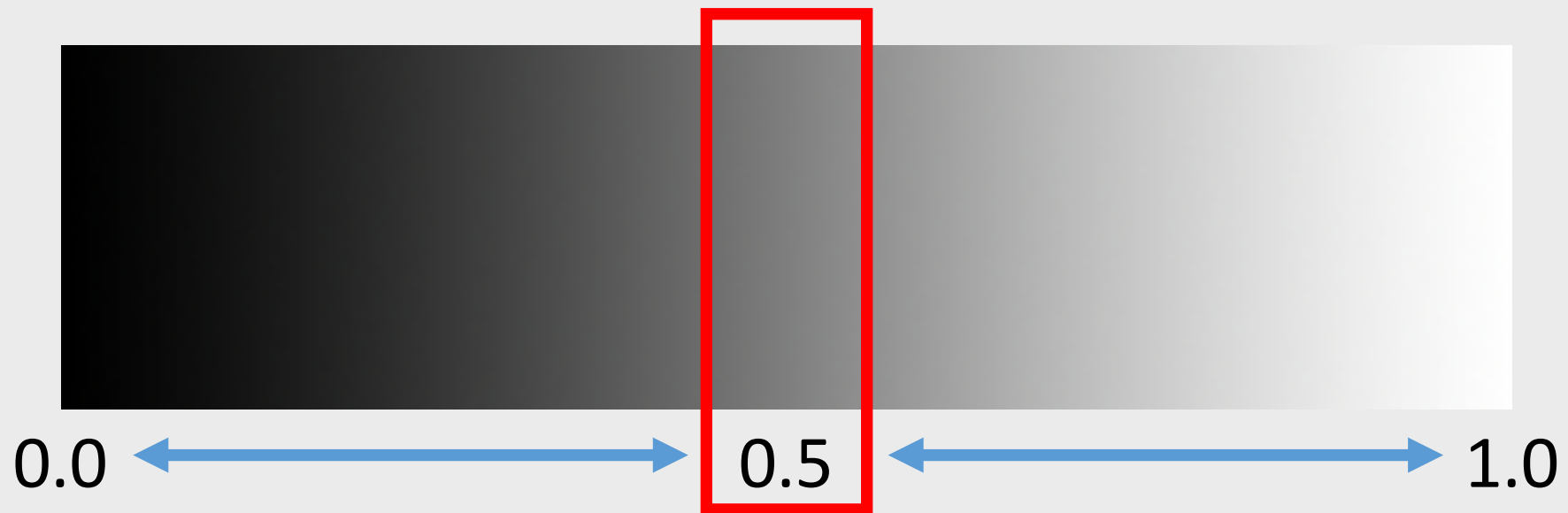
※リニア対策あり

リニアワークフローに 挑戦

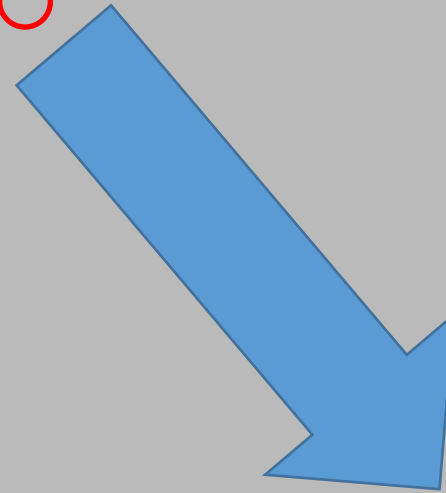
リニアワークフローに挑戦

- 色空間をリニアで計算することを考慮し、シェーディングを行う。
 - テクスチャなどがsRGB空間で作られているのをリニア空間に変換&考慮しながら処理を作る。
- 今回は3Dの部分のみ適応。
 - というより、自ずとそうなるが・・・。

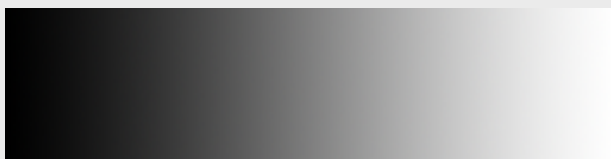
簡単な解説



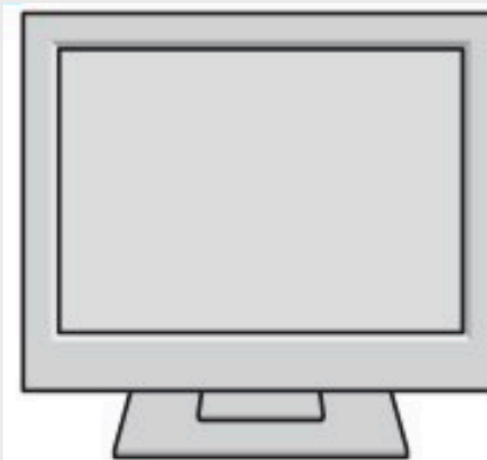
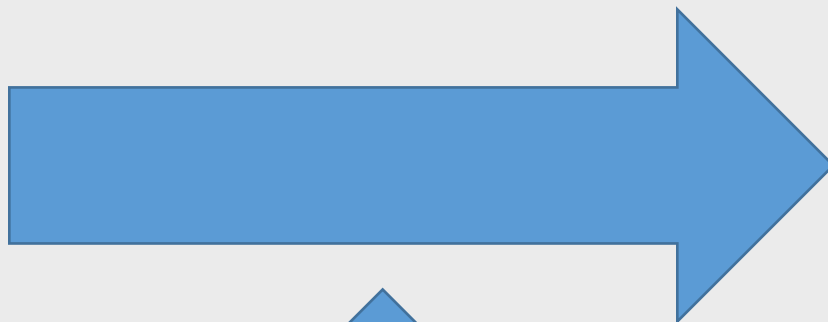
RGBデータが0.5のカラー



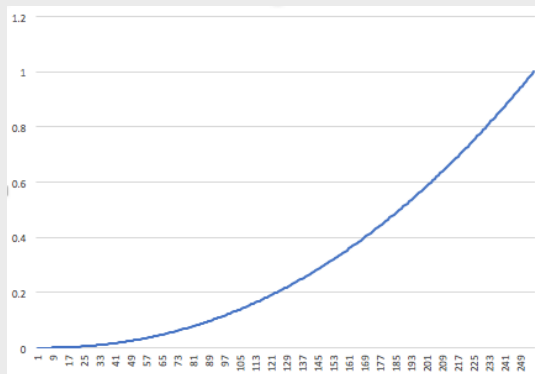
市松模様での0.5再現



カラー情報



ディスプレイ





$$\text{出力} = \text{入力}^{2.2}$$

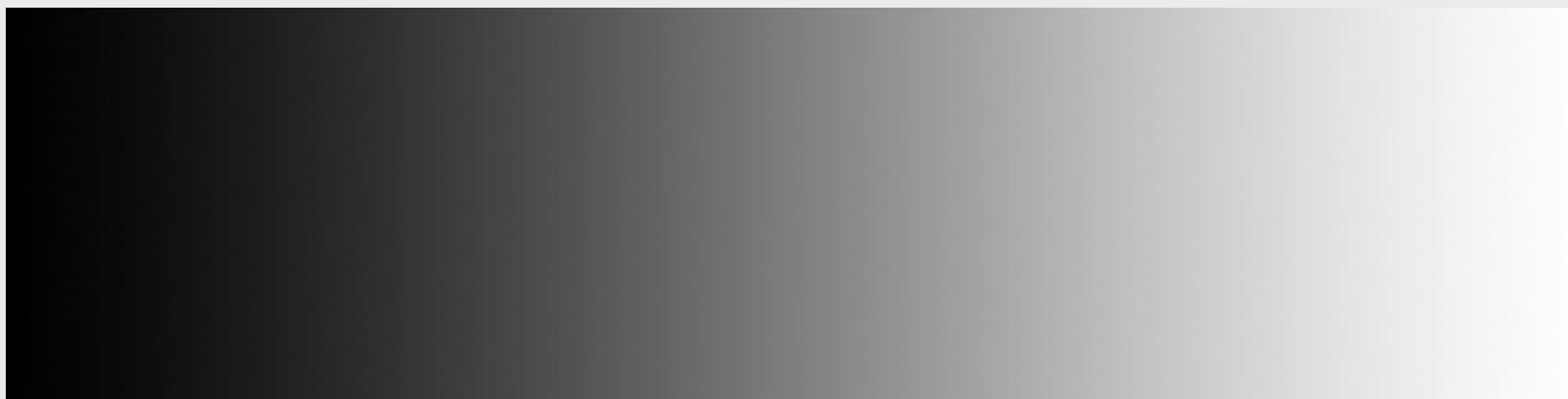
GAME CREATORS CONFERENCE'17

呼び方	コンピュータでの 数値	光学的カラー数値
sRGB空間	0.5	0.21763...
リニア空間	0.5	0.5

リニア



sRGB



Intensity 0.5

Intensity 1

Intensity 1.5

Linear Space



Gamma Space



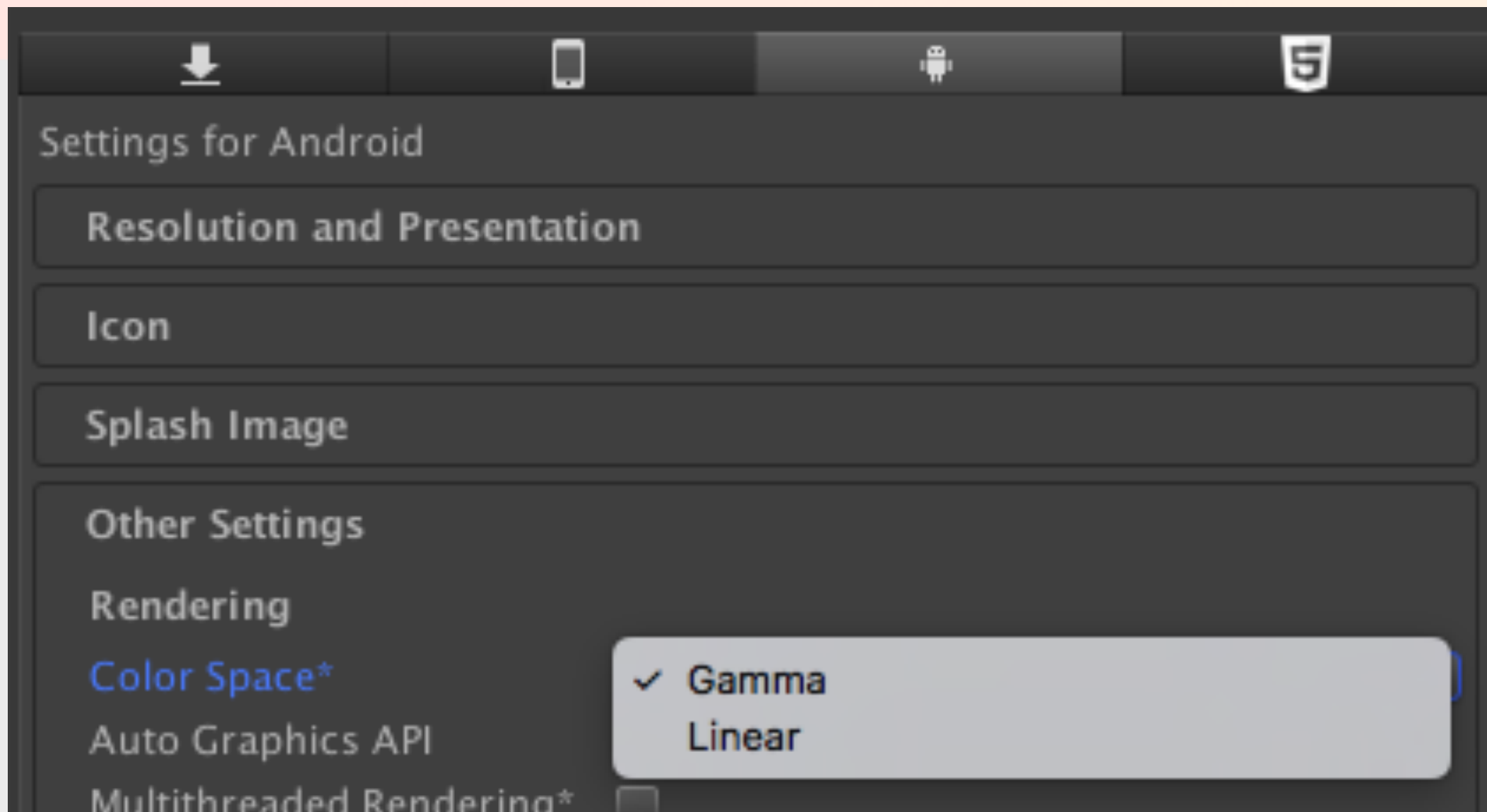
※Unityの「リニアレンダリング」より

<https://docs.unity3d.com/ja/current/Manual/LinearLighting.html>

リニアワークフローとは？

- 前提として、カラーというのはRGB毎に0～1で表現される。
- コンピュータ上で「中間の色」として0.5を入れたとしても、光学的に「中間の色」で表示されず、若干暗く表示されてしまう。
 - カラー数値に対し2.2乗されるので。
(0.5の場合、 $0.5^{2.2} = 0.21763\dots$)
 - ディスプレイの特性からこうなっている。
- こういった特性を加味しながら処理を行うことを指す。

Unityではどうなる？



GAME CREATORS CONFERENCE'17

Rendering

Color Space*

Linear



On Android, linear colorspace requires OpenGL ES 3.0 only, uncheck "Automatic Graphics API" to remove OpenGL ES 2 API and "Minimum API Level" must be at least Android 4.3

Auto Graphics API



Graphics APIs

= OpenGL ES 3

= OpenGL ES 2



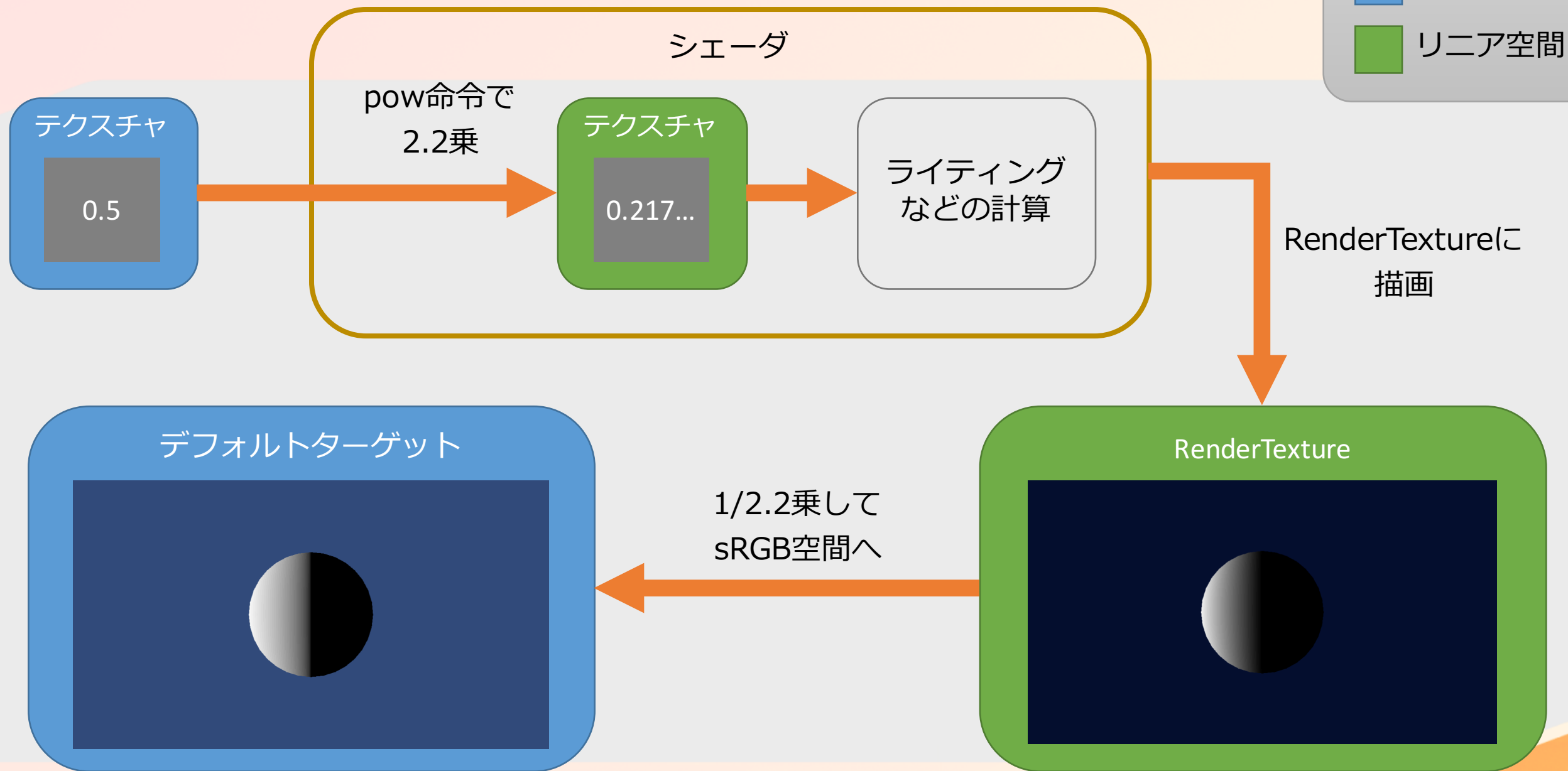
Multithreaded Rendering*



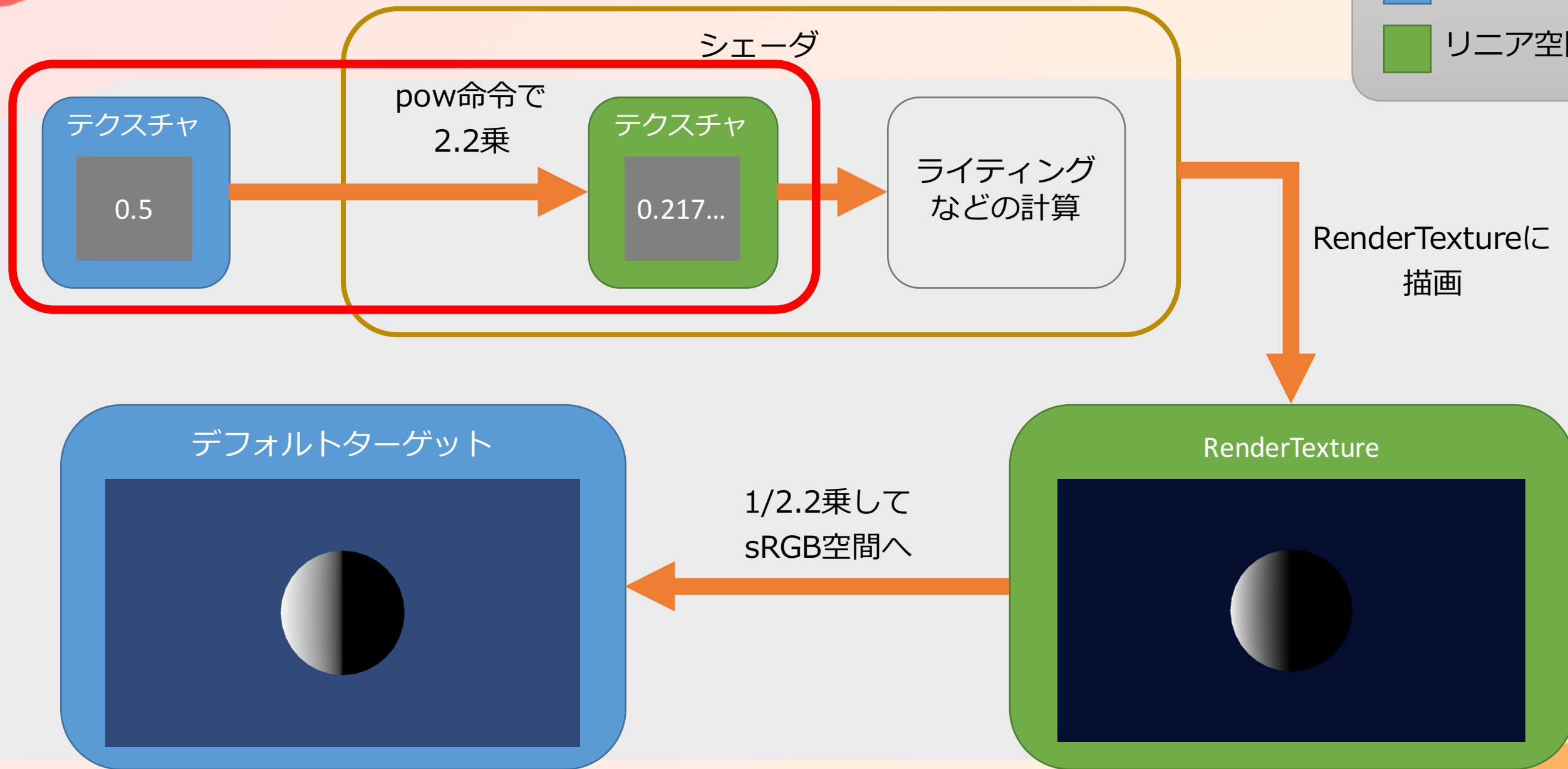
Unityでのリニアワークフロー

- Unityでも、PlayerSettingsで設定変更ができるようになっている。
 - スマホでは、Unity5.5より搭載された。
- **しかしOpenGL ES2.0を対象に含めっているとビルドできない！！**
 - 弊社ブログを参照ください。
(<http://logicalbeat.jp/blog/662/>)
 - 気持ちはわかるが起動ができないのは超困る・・・。

独自で
対応してしまおう！



- SRGB空間
- リニア空間



テクスチャの リニア空間変換



高負荷！

テクスチャをsRGB空間からリニア空間に変換（1）

- sRGB空間で作られたテクスチャをリニア空間に変換する必要がある。
 - sRGB→リニアは2.2乗することで得られる。
- 2.2乗するにはシェーダでpow命令を使う。
- しかしpow命令は**高負荷！！**
できれば使いたくない！！



インポーター (Unity)

テクスチャ

0.5

予め2.2乗
しておく

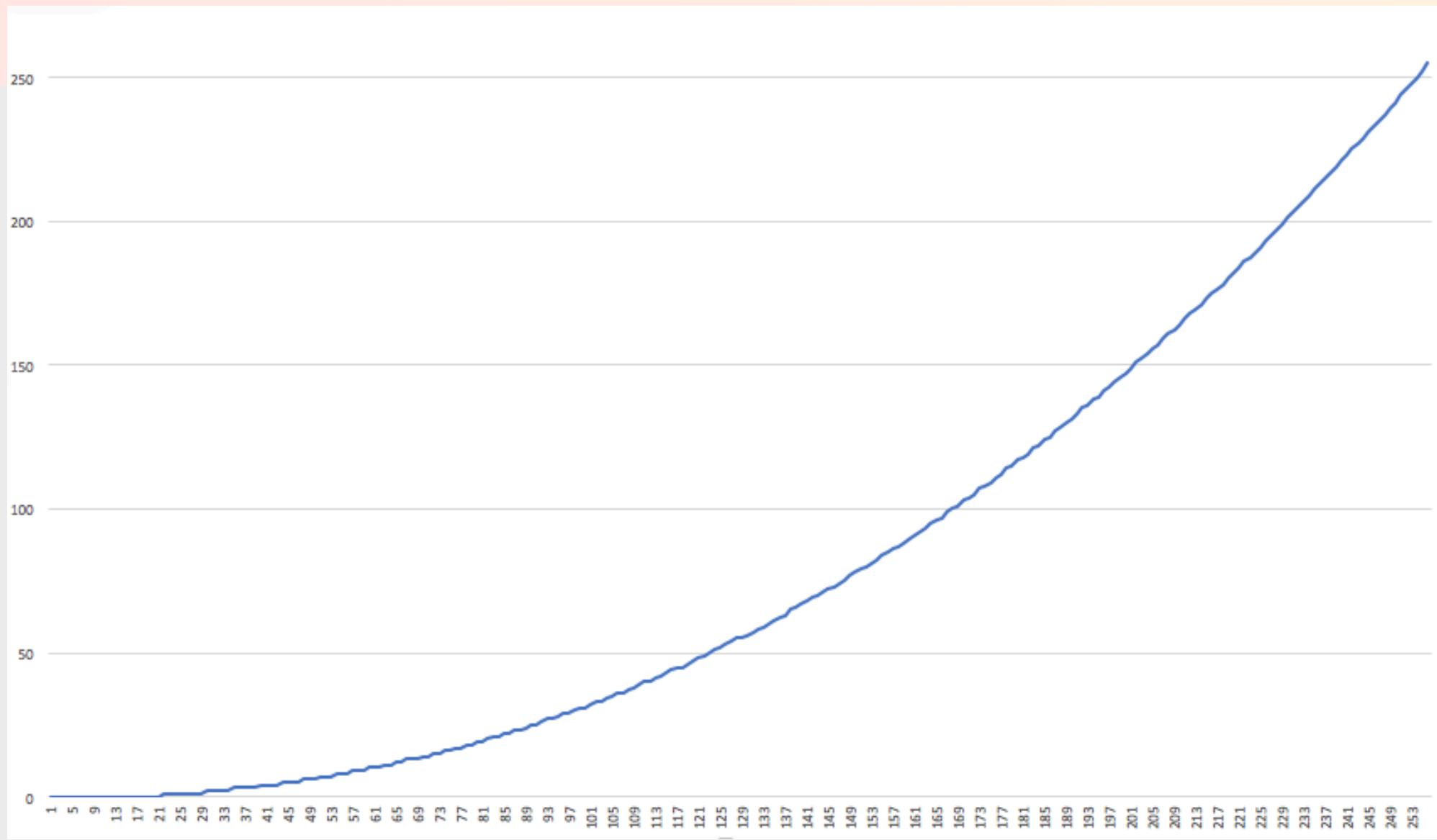
テクスチャ

0.217...

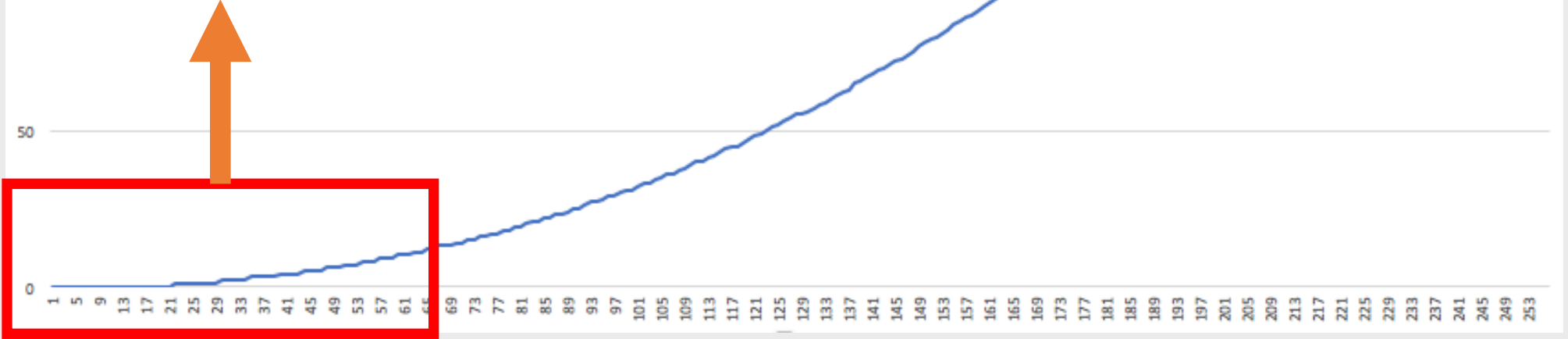
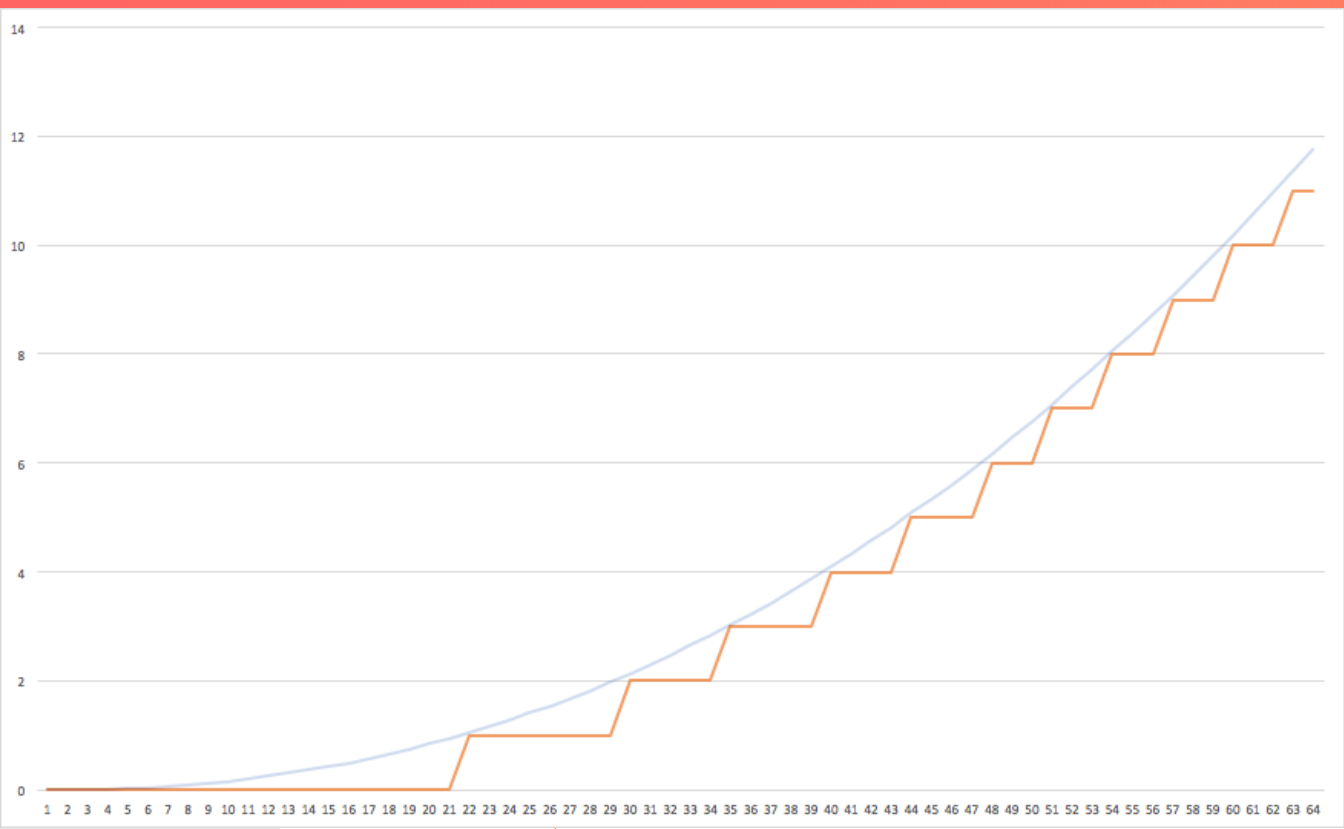
そのまま利用

テクスチャ

0.217...



GAME CREATORS CONFERENCE '17



GAME CREATORS CONFERENCE'17

テクスチャをsRGB空間からリニア空間に変換（２）

- テクスチャを予めリニア空間（2.2乗）しておけば良い？
 - 幸いUnityではテクスチャインポート時に画素を編集できる。
(AssetPostprocessor、TextureImporter)
- しかし変換後の精度劣化が激しい！！
 - カラーは256段階整数（0～255）が基本なので。
 - 「20」でも「0」として格納される・・・。
暗い方の精度が非常に心配。

ここで数式

C^{2.2}

$$C^{2.2}$$
$$= C^{1.1 \times 2}$$

$$\begin{aligned} C^{2.2} \\ &= C^{1.1 \times 2} \\ &= C^{1.1} \times C^{1.1} \end{aligned}$$

インポーター (Unity)



0.5

C

予め1.1乗
しておく



0.467...

C^{1.1}

シェーダ

2乗計算

$C^{1.1} \times C^{1.1}$



0.217...

C^{2.2}

テクスチャをsRGB空間からリニア空間に変換（3）

- テクスチャを予め1.1乗し、シェーダ内で更に2乗するのは？
 - これもテクスチャインポート時に画素を編集。
- 大きなメリットが生まれる！
 - 1.1乗なら256段階だとしても劣化が目立ちづらい。
 - シェーダ内で2乗する必要があるが、それはpowを使わなくて良い。
（自分自身で掛け算すればOK！）
- これで進めそう！

※インポートで
加工済み



2乗計算

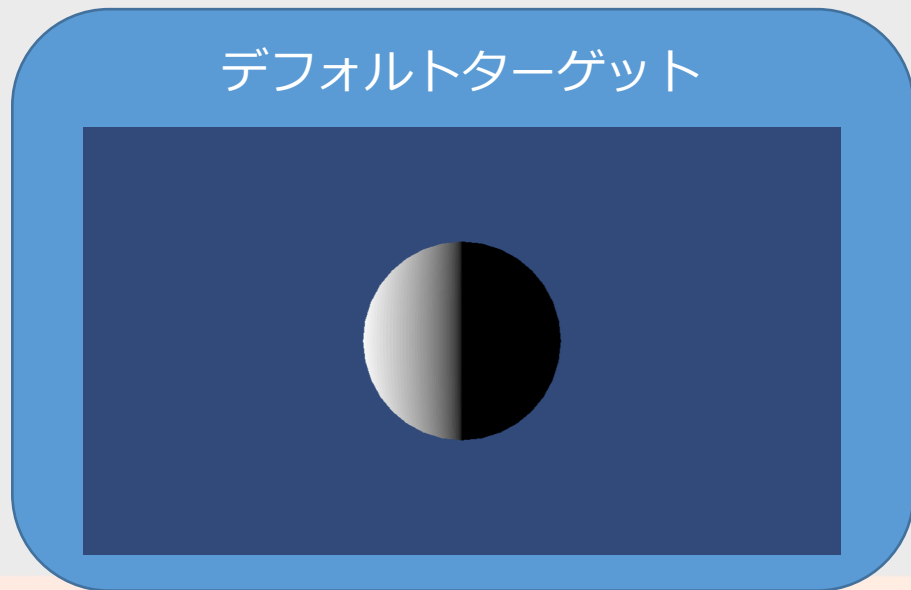
シェーダ



ライティング
などの計算

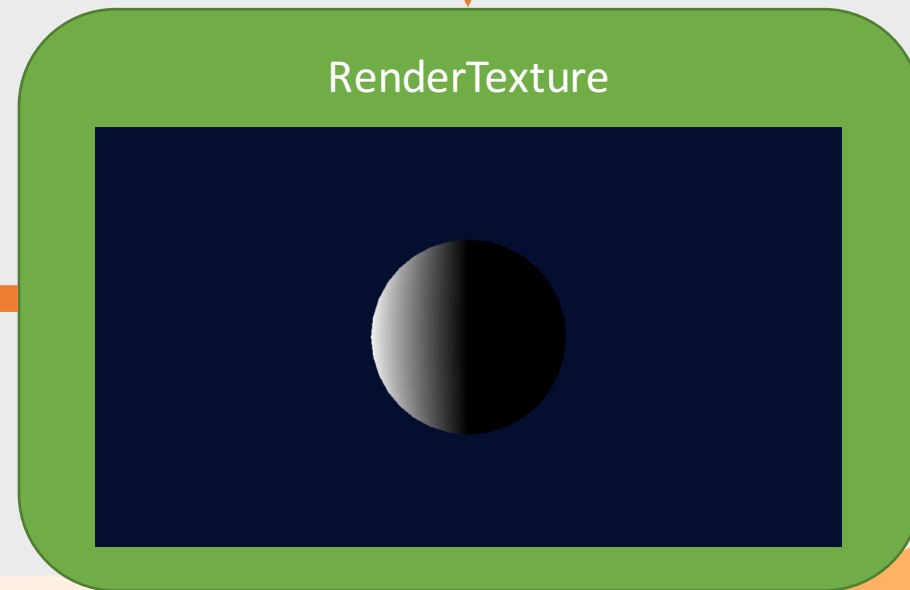
RenderTargetに
描画

デフォルトターゲット



1/2.2乗して
sRGB空間へ

RenderTarget

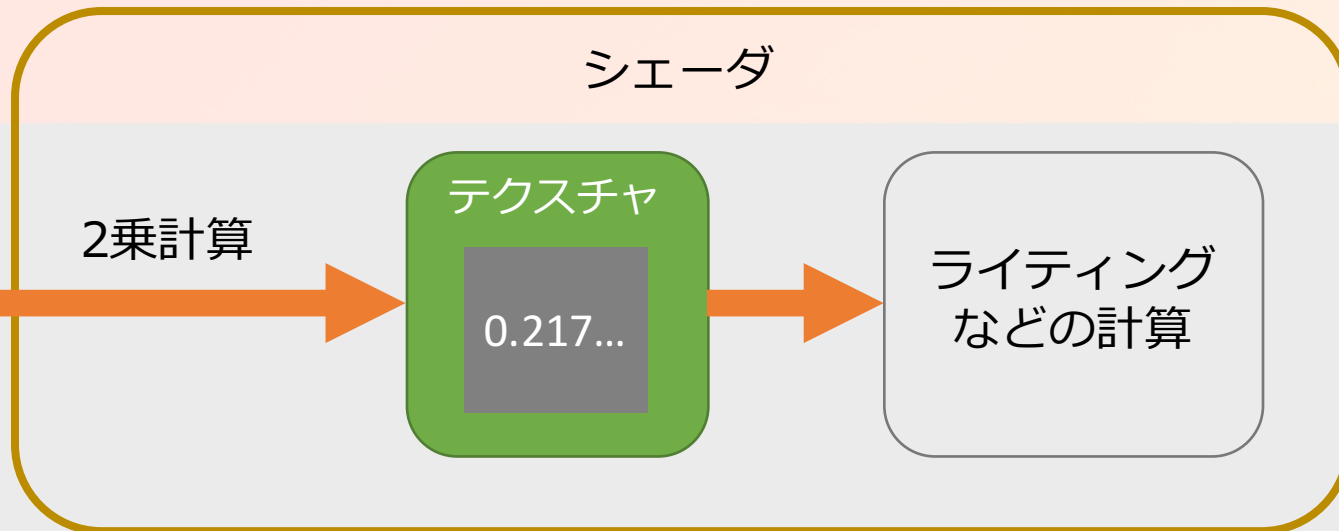


SRGB空間
リニア空間

※インポートで
加工済み

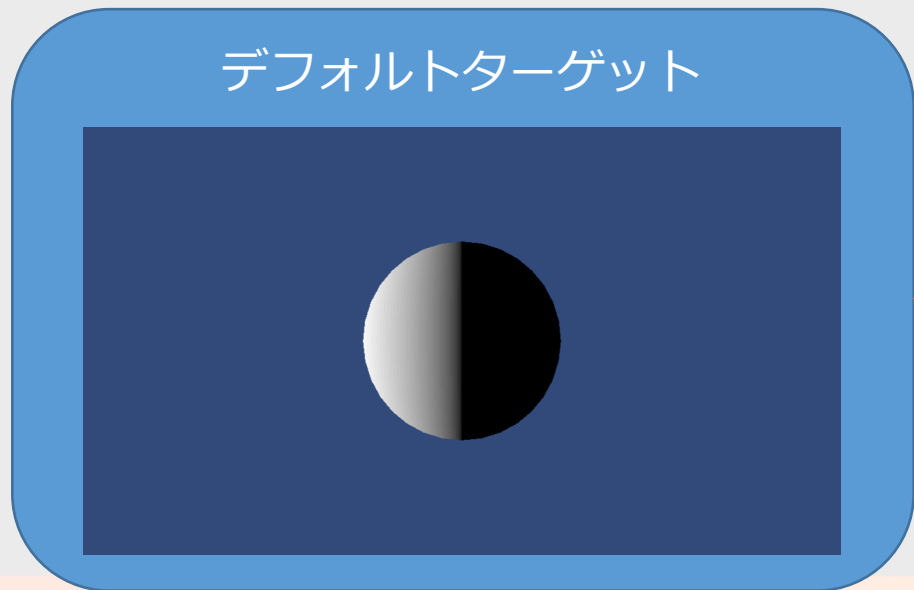


2乗計算

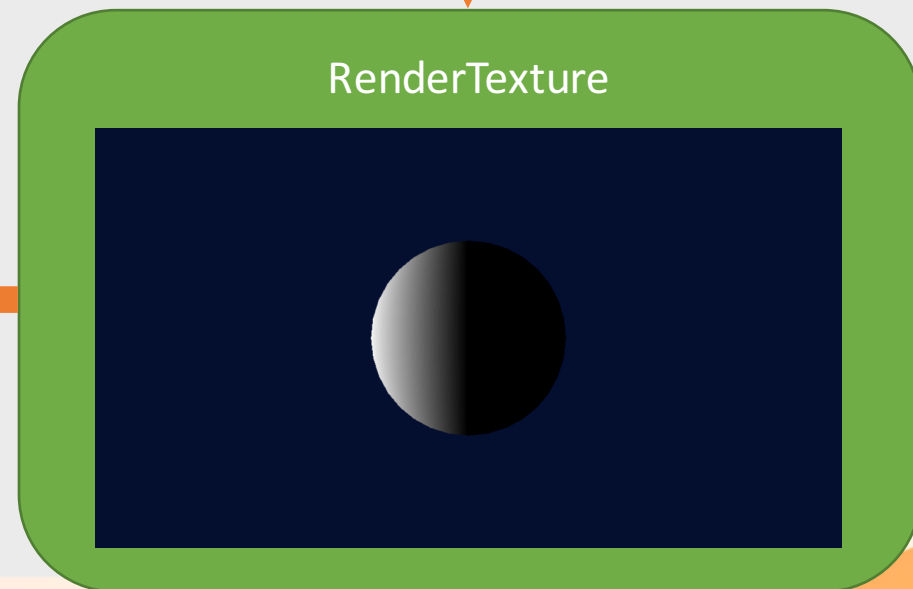


RenderTextureに
描画

デフォルトターゲット



1/2.2乗して
sRGB空間へ

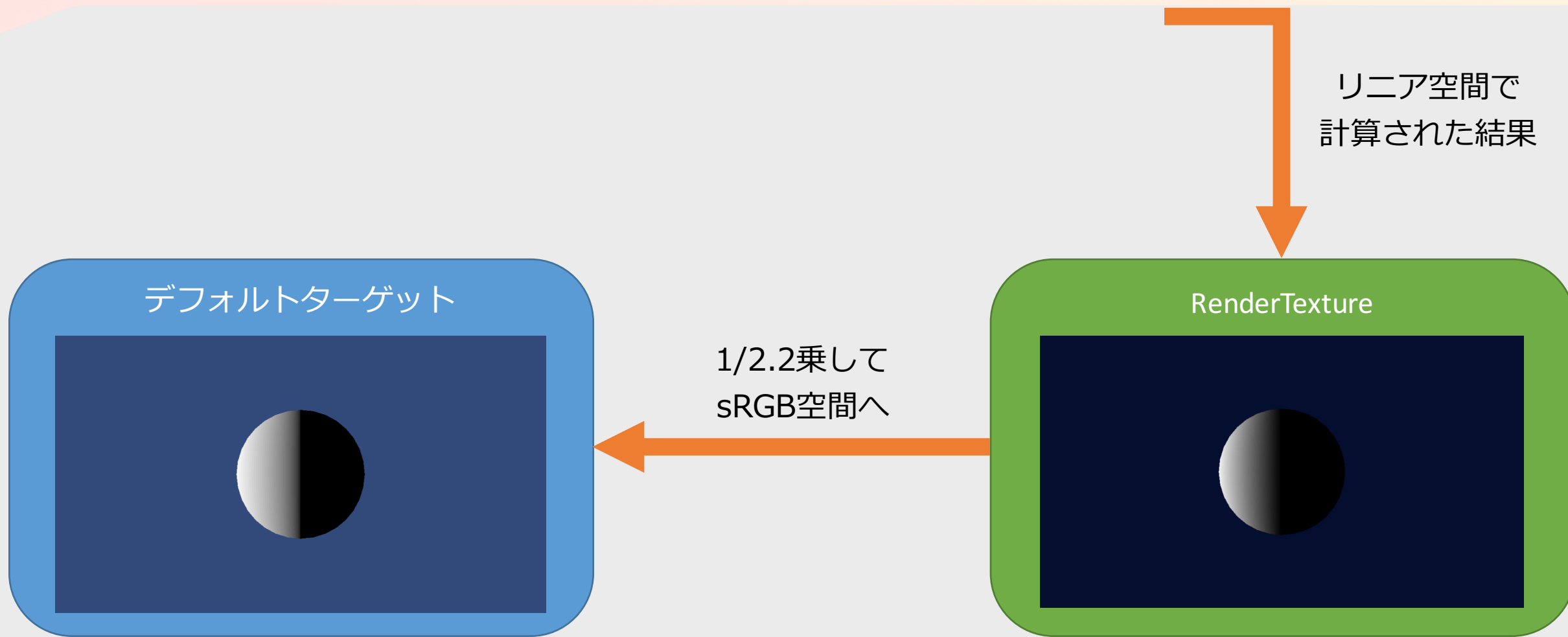


SRGB空間

リニア空間

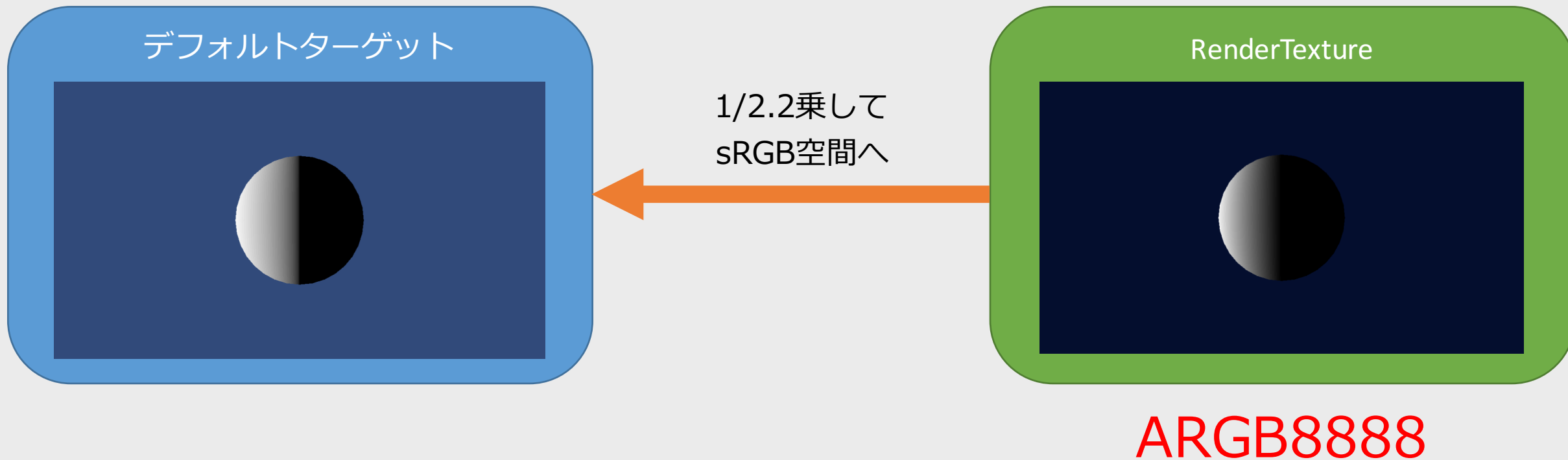
A legend in the top right corner. A blue square is next to the text 'SRGB空間' (SRGB space), and a green square is next to the text 'リニア空間' (Linear space).

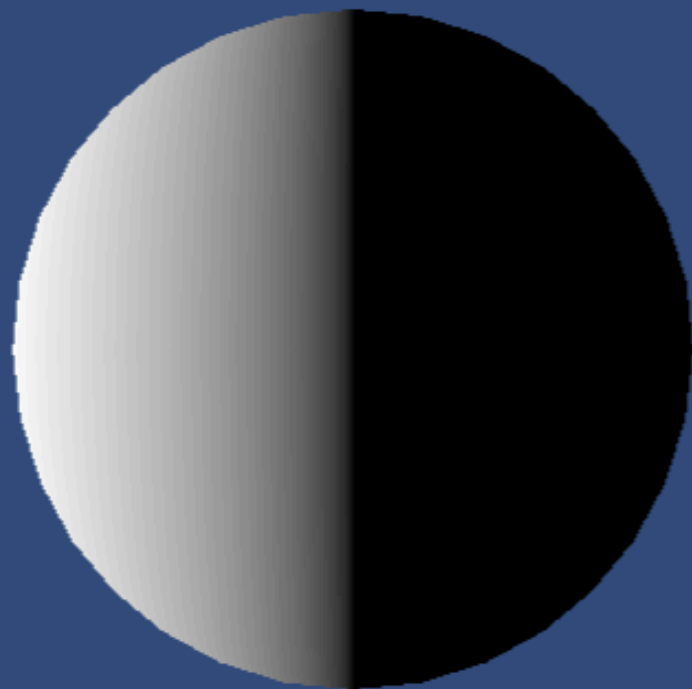
書き込み レンダーターゲット



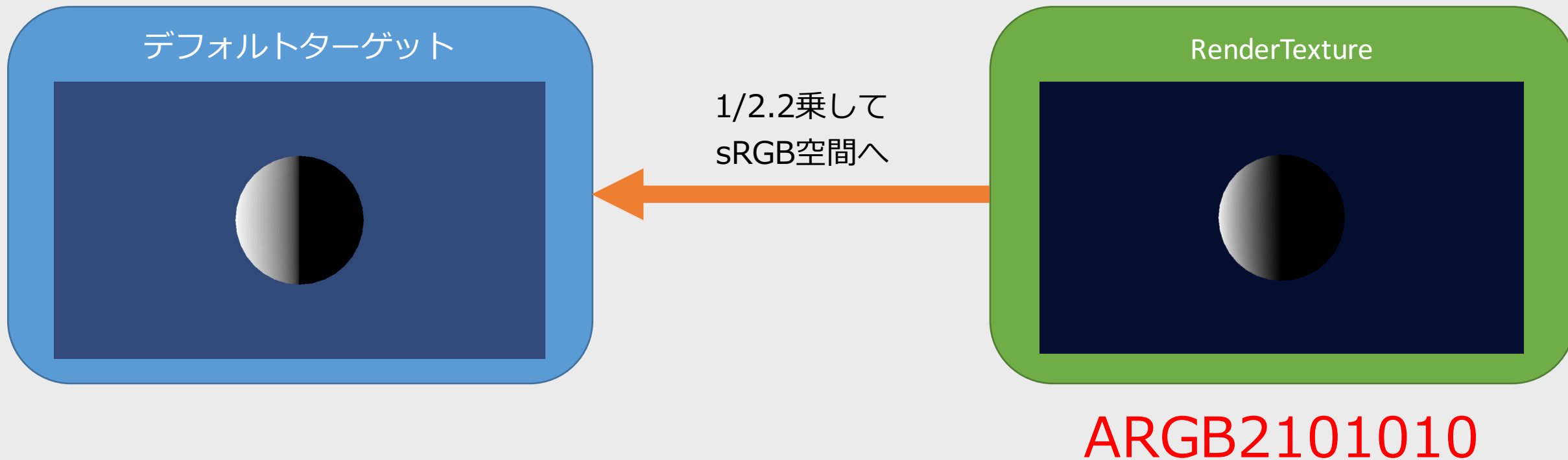
リニア空間の絵を画面に出力（1）

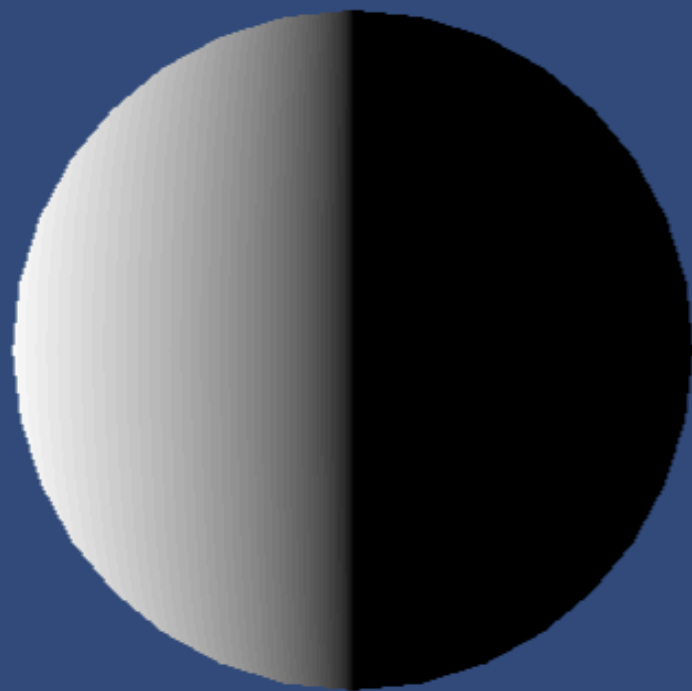
- 作業時はリニア空間で作業するが、最終的にはsRGB空間に戻す必要がある。
 - 「リニア空間の色合いを加味しながら」がポイント。
 - ただ単に $1/2.2$ 乗するだけ。
- 処理的にはRenderTexture（リニア空間）に描画し、最後にそれをデフォルトターゲット（sRGB空間）にコピー（ $1/2.2$ 乗）。
 - ついでに各種ポストエフェクトもここに入れる。



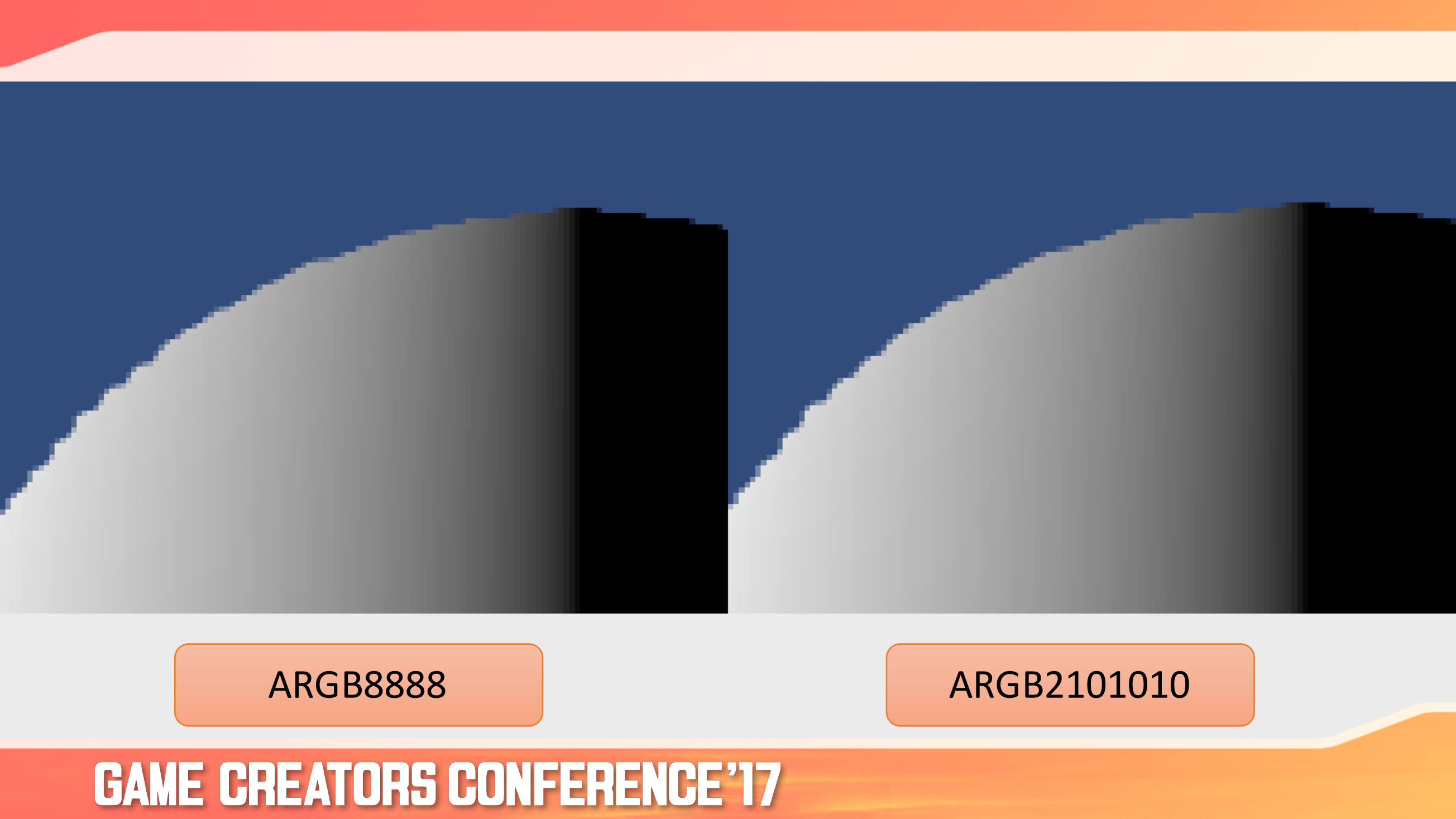


ARGB8888を介した場合





ARGB2101010を介した場合



ARGB8888

ARGB2101010

GAME CREATORS CONFERENCE'17

リニア空間の絵を画面に出力（２）

- 絵を描いているRenderTexture（リニア空間）自体の精度も重要！
 - カラーを1/2.2乗した時にも情報損失が発生。
（整数化を考慮）
 - 誤差程度ですが・・・。
- RenderTexture自体のフォーマット精度を高くしたい。
ので、ARGB8888（RGBA32）ではなくARGB2101010を採用。
 - halfやfloatバッファは使わない。（重いので）
 - RGB毎に8ビット→10ビットにし、精度を上げておく。
 - Aの2ビットは、ポストフィルタで活用。（割愛）

ARGB2101010

OpenGL ES2.0では
使えないかも！

リニア空間の絵を画面に出力（3）

- ARGB2101010を使ったが、どうもこれはOpenGL ES3.0から使えるものらしい。
 - OpenGL ES2.0では未確認・・・。
 - OpenGL ES3.0では標準で使えるらしい？
- これも「OpenGL ES2.0」をどう捉えるか？の話。
- OpenGL ES2.0では2Dモードを推奨している。
 - このリニアワークフロー部分は3D部分なので・・・。
 - ARGB8888で精度が落ちてても仕方ないというスタンス。

※インポートで
加工済み



2乗計算

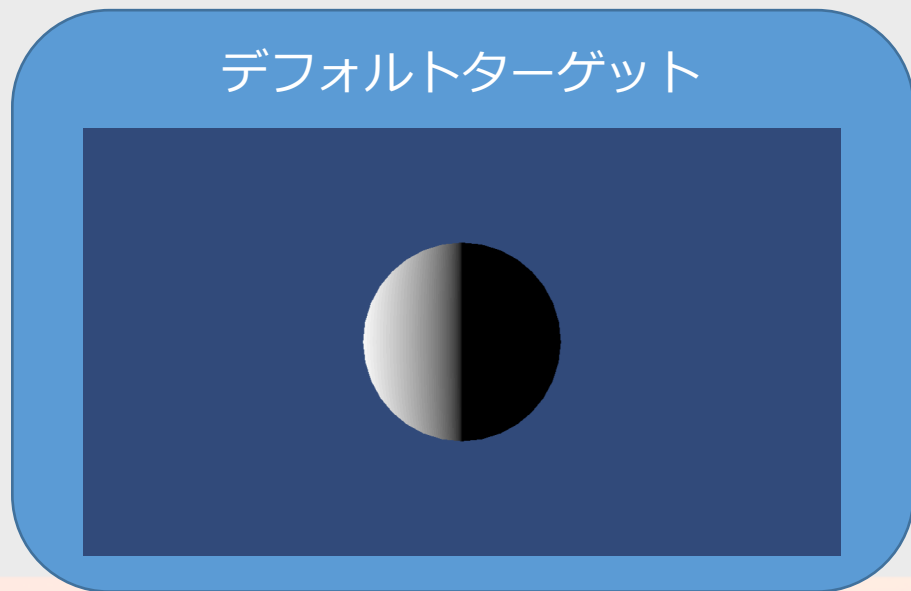
シェーダ



ライティング
などの計算

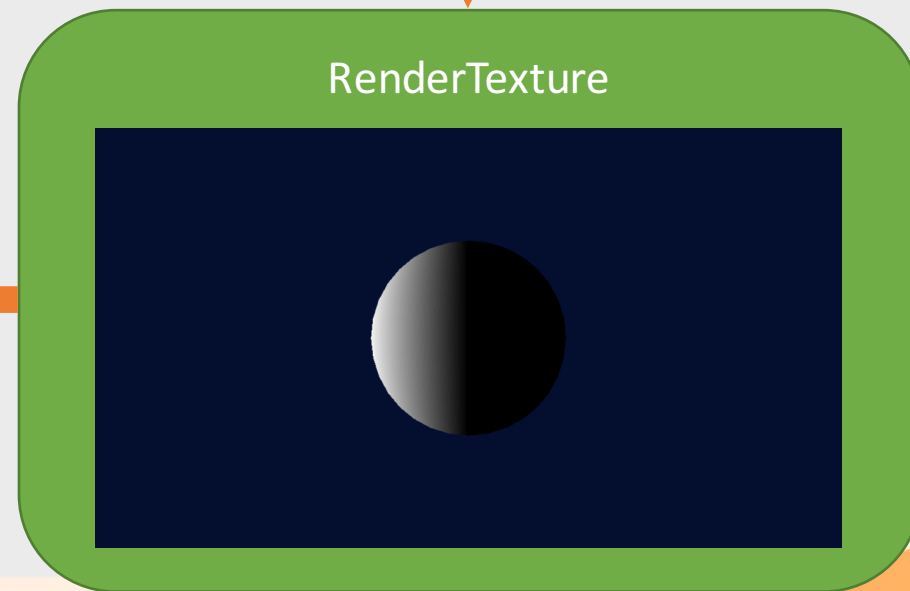
RenderTextureに
描画

デフォルトターゲット

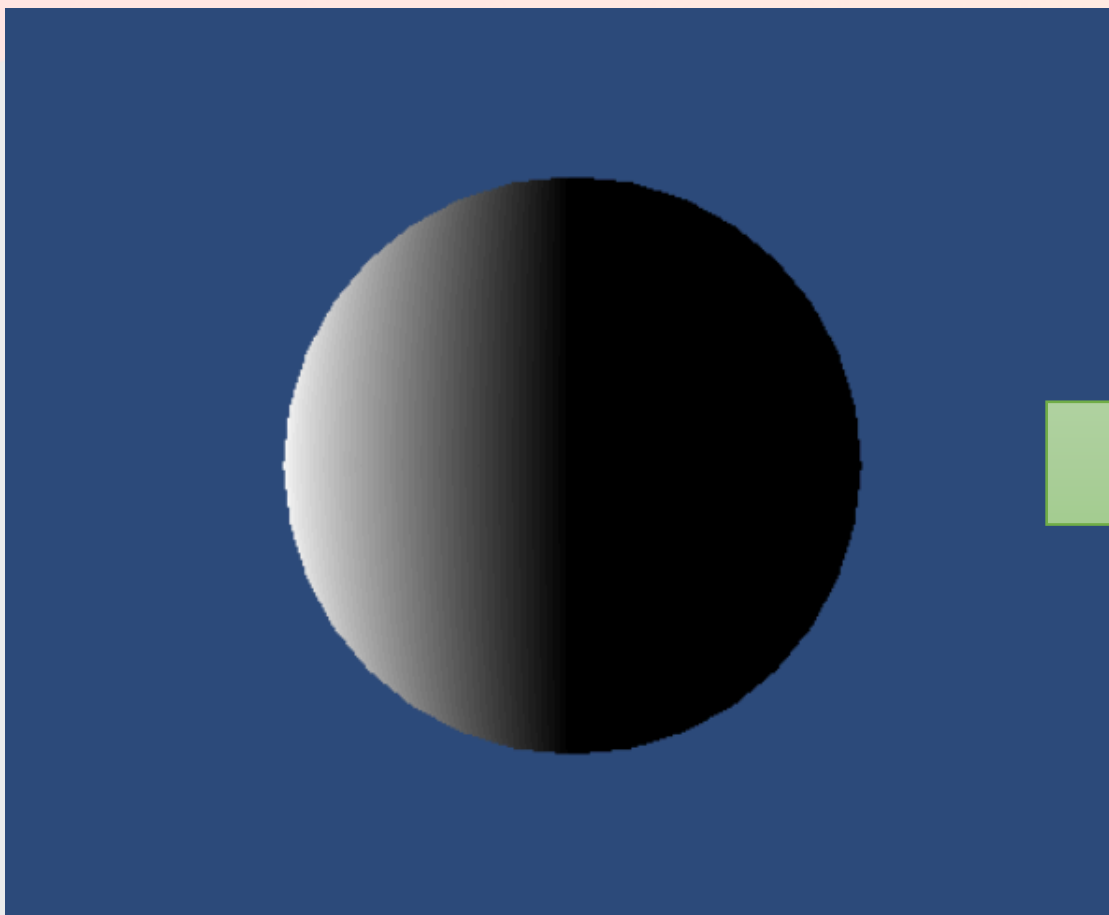


1/2.2乗して
sRGB空間へ

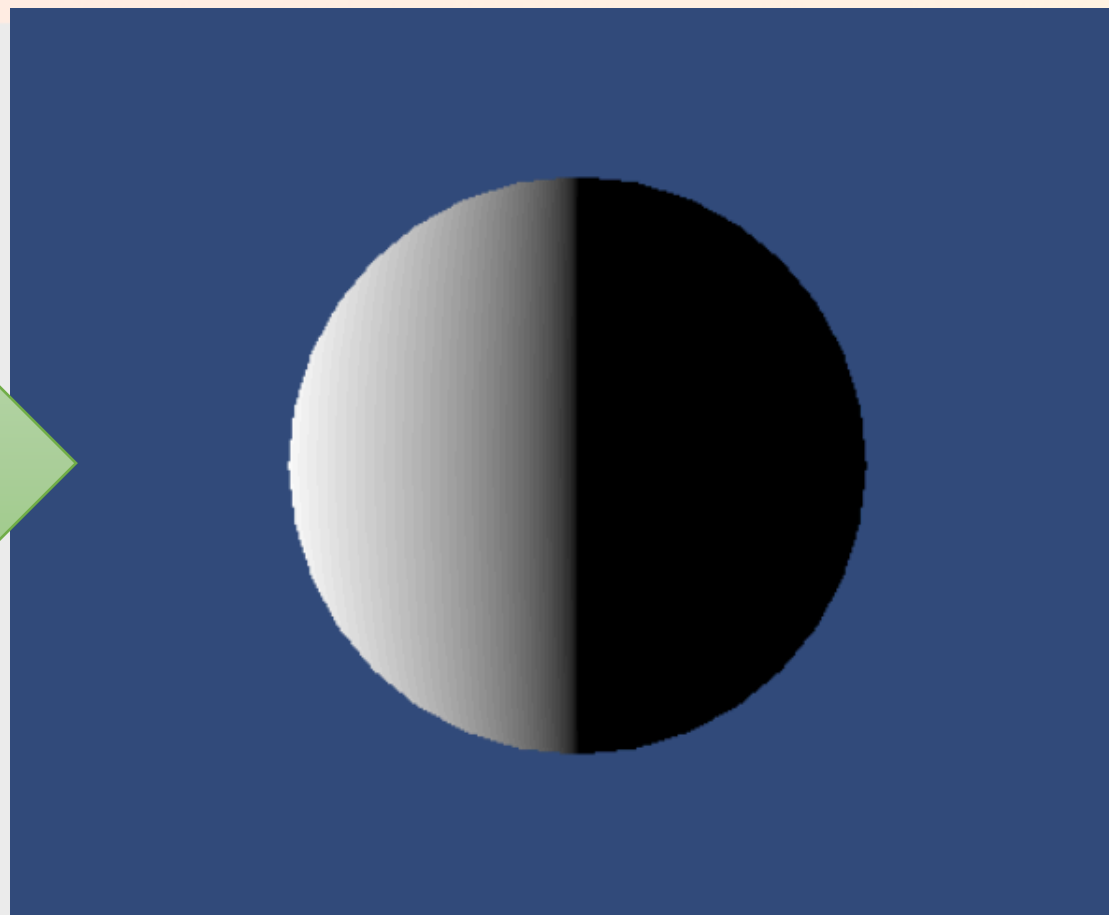
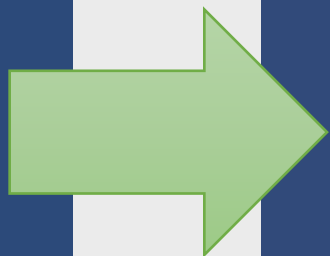
RenderTexture



SRGB空間
リニア空間



※リニア対策なし



※リニア対策あり

リニアワークフロー 完成！

(条件付きですが)

リニアワークフローまとめ

- OpenGL ES2.0でも動作、それなりの描画負荷で動作するリニアワークフローが出来上がった。
- ゲームエンジンでサポートされていないということで早々に諦めないのが大事。
 - 特にスマホでは下のバージョンに機能を揃えてしまう傾向があるが、それはアイデアでカバー。
 - ここはコンシューマ機での「機能を吸い尽くす」精神が働いてる気がする。

まとめ

まとめ

- コンシューマ機とスマホでは、制作スタンスを変えざるを得ないところが出てくる！
 - 規格に対し端末の種類が多く、性能差などが生じるため。
 - 考え方の違い！脳みそを切り替える。
- グラフィックスAPIで出来ること、出来ないことを見極める。
 - 現在はOpenGL ES2.0の存在がいろんな意味で大きい。
 - 頑張って調べられるだけ調べてみる。
が、考えすぎても仕方ないので作ってしまうのも大事。

まとめ

- 同時に、グラフィックスAPIのサポートや未来について考えることも重要になってくる。
 - 「描画設定」を設けておくと良い。
 - 下限をどこにおくか？そしてどうサポートするか？
 - 将来出る高性能機向けのモードがあっても良い。
ここはスマホならではの楽しんで作る。
- リニアワークフローの話。
 - ゲームエンジンで出来ないからと早々に諦めない！
(逆に言えば差別化にもなる。)

ご静聴
ありがとうございました