DRECOM
with entertainment

Metal と Vulkan を用いた 水彩画レンダリング技法の紹介

次世代のモバイルグラフィックスは Non-Photorealistic Rendering へと移行する

株式会社ドリコム　増野 健人
川上 知成

Copyright Drecom Co., Ltd. All Rights Reserved.

DREECOM with entertainment

本日の発表について

講演者紹介

川上 知成　-　進行

株式会社ドリコム　プログラムマネジメント室
サーバーサイドエンジニア、PMを担当し、現在は複数のプロジェクトに関与
CEDEC2014, 2015登壇

増野 健人　-　メインスピーカー

株式会社ドリコム　ゲームプロダクト部
コンシューマゲーム会社にてグラフィックスシステムやシェーダー開発に携わる
ドリコム入社後は3D・シェーダー技術のスキルボトムアップの活動を行う

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM
with entertainment

Copyrights.

Vulkan and the Vulkan logo are trademarks of the Khronos Group Inc.
OpenGL is a registered trademark and the OpenGL ES logo is a trademark of Silicon Graphics Inc. used by permission by Khronos.
SPIR and the SPIR logo are trademarks of the Khronos Group Inc.
Apple, Mac, iMac, MacBook, iPhone, Xcode, macOS, Metal and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.
iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.
Android™.
DirectX®.
FlowMap Painter (c)2012 Teck Lee Tan www.teckArtist.com

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

はじめに

ゲームエンジンと要素技術/基礎技術

要素技術/基礎技術を見る理由

- 対象事象の原理原則を押さえる

→比較対象や比較すべきポイントを選定出来、評価が可能

- 時代に乗る

→市場、発表、研究、発見などの各段階で見極め必要な技術を取捨選択

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

セッションの内容

- 近年のモバイルグラフィックス
- グラフィックスAPI のパラダイムシフト
- Low-Overhead, Low-Level プログラミング
- Non-Photorealistic Rendering における水彩画表現
- まとめ、質疑応答

Copyright Drecom Co., Ltd. All Rights Reserved.

近年のモバイルグラフィックス

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

モバイルデバイスとゲーム

そもそもスマートフォンは多岐に渡る利用用途がある。

- 電話、メール、メッセージアプリ
- SNS (コミュニケーション)
- 動画視聴
- ゲーム

ゲームはあくまでコンテンツの一つ。
ユーザーはバッテリーリソースを消費しながらこれらコンテンツを利用している。

Copyright Drecom Co., Ltd. All Rights Reserved.

ゲームの更新とレンダリング

現在、スマートフォン端末はフルHD（1920x1080px) の画面解像度を持ったデバイスが一般的に普及している。総ピクセル数で言えば約207万px あり、これを1秒間に60回書き換えを行なっている。

この膨大なピクセルの更新を CPU を使用して行うことも可能だが、リアルタイム性を求めるゲームではレンダリング専用のプロセッサ(GPU)を通して画面更新の高速化を図っている。

→ スマホゲーム通常は OpenGL ES を通して GPU で描画が行われている。

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

OpenGL ES

OpenGL for Embedded System

Khronos Group という標準化団体によって策定されている組み込み機器向けのグラフィックスAPI

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

モバイルゲームを支えるAPI

OpenGL ES は現在の iOS / Android 開発を支える共通のグラフィックスAPI。
描画部分においては最も基本であり、最も重要なAPIの一つ。

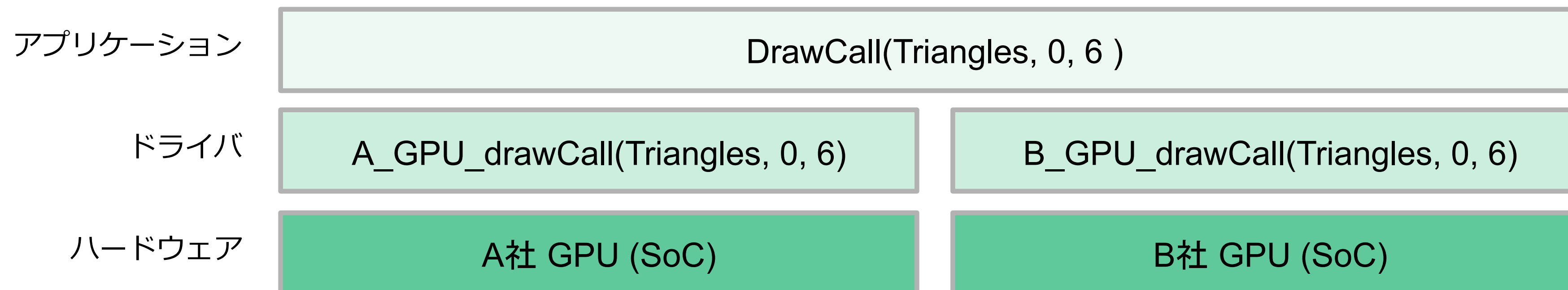
現在のスマホネイティブアプリは、iOS/Android の両対応がほぼ一般的になっているが、プラットフォームが異なるこの2つのデバイスにおいて、OpenGL ES でグラフィックスエンジンを記述すれば、両プラットフォームの対応を行うことができる。

Copyright Drecom Co., Ltd. All Rights Reserved.

OpenGL ES

グラフィックスAPI の呼び出しにより CPU で動作しているアプリケーションから GPU に対してタスクの発行が行われる。(GPUドライバが制御する)

A社 の GPU と B社 の GPU でアーキテクチャが異なっても、この共通の API によりその違いを吸収している。



Copyright Drecom Co., Ltd. All Rights Reserved.

OpenGL ES

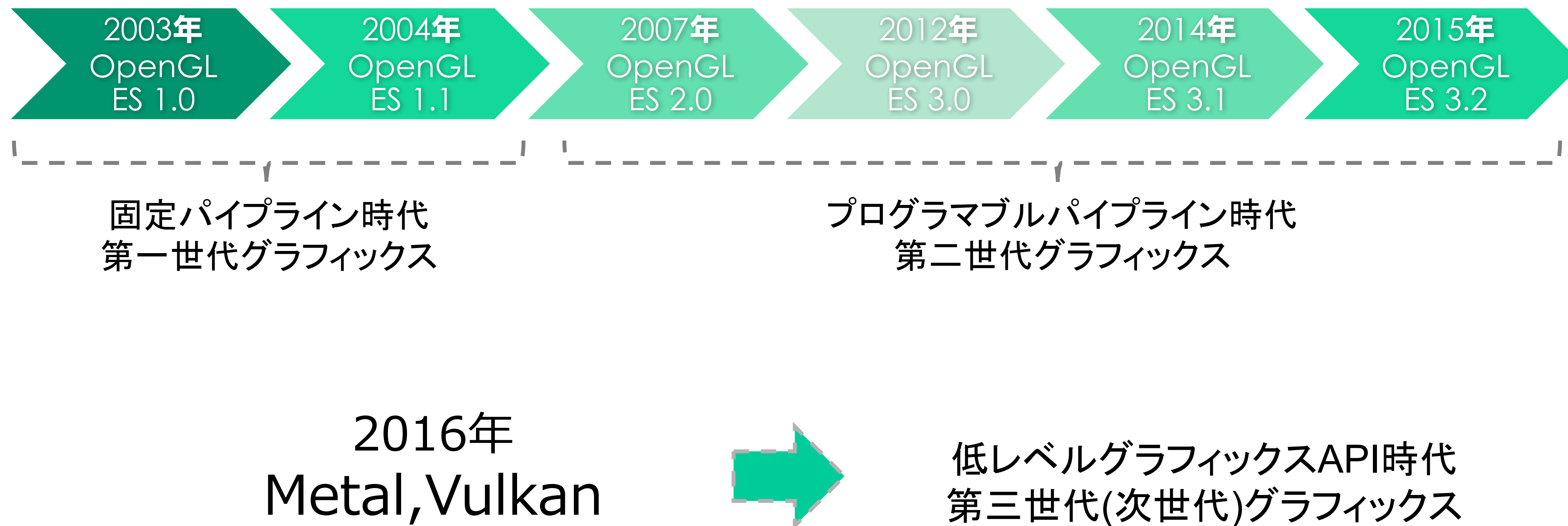
Khronos による仕様の策定

- 2015年8月には、最新バージョンとなる ES 3.2 がリリース
- 現在のモバイルグラフィックスの多くは ES 2.0 をベースにしている

後述する、Metal, Vulkan も、OpenGL ES と同じグラフィックスAPIの一つ。

Copyright Drecom Co., Ltd. All Rights Reserved.

モバイルグラフィックスの推移



Copyright Drecom Co., Ltd. All Rights Reserved.

グラフィックスAPI

グラフィックスAPIを選ぶ基準としては、OSサポートは勿論、端末に搭載される GPU (ドライバ) の対応が必須となる。

後述する、Metal, Vulkan も端末によってサポートが異なるため留意

OpenGL ES	Android	iOS
2.0	2.2	iPhone3GS
3.0	4.3	iPhone5S
3.1+AEP	5.0	-
3.2	6.0	-

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

モバイルグラフィクスの今

最新のモバイルグラフィックスは何が出来るか？

→ ES 3.2 は DirectX11.0 世代と同等の機能を有している

- 機能面だけで言えば PC, PS4, XboxOne の世代に追いついている
- しかし性能面では大きな開きがあるため同等のグラフィックス表現は難しい

最新の OpenGL ES 3.2 では、コンピュートシェーダーは勿論、テッセレーションとジオメトリシェーダーまでサポートされている。

Copyright Drecom Co., Ltd. All Rights Reserved.

OpenGL ES 3.0

2012年8月リリース。ES2.0 からは実に5年ぶりのアップデート。
主な機能

・レンダリングパイプラインの強化
  - Occlusion Query
  - Transform Feedback
  - Instanced Rendering
  - Multiple Render Targets
・ETC2 テクスチャ圧縮形式のサポート
・and more...

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM
with entertainment

OpenGL ES 3.1

2014年3月にリリースされた。
主な機能として、コンピュートシェーダ がサポートされた。

- Vertex, Fragment オブジェクトの独立
- Indirect draw Commands
- and more...

ほぼ同時期に策定された Android Extension Pack というGL拡張も、
大きな注目を集めた

Copyright Drecom Co., Ltd. All Rights Reserved.

Android Extension Pack (AEP)

AEP とは 下記の GL 拡張のことを指します

GL_ANDROID_extension_pack_es31a

AEP はメタ拡張と呼ばれ、開発者はこの GL 拡張が定義されている場合、後述する20以上の拡張がサポートされていると見なす。

この拡張を GPU がサポートすることにより、DirectX11 世代のゲームを Android への移植を容易にした。

Copyright Drecom Co., Ltd. All Rights Reserved.

Android Extension Pack (AEP)

KHR_debug
KHR_texture_compression_astc_ldr
KHR_blend_equation_advanced
OES_sample_shading
OES_sample_variables
OES_shader_image_atomic
OES_shader_multisample_interpolation
OES_texture_stencil8
OES_texture_storage_multisample_2d_array
EXT_copy_image

EXT_draw_buffers_indexed
EXT_geometry_shader
EXT_gpu_shader5
EXT_primitive_bounding_box
EXT_shader_io_blocks
EXT_tessellation_shader
EXT_texture_border_clamp
EXT_texture_buffer
EXT_texture_cube_map_array
EXT_texture_sRGB_decode

Copyright Drecom Co., Ltd. All Rights Reserved.

OpenGL ES 3.2

OpenGL ES 最新の策定。2015年8月に仕様がリリース。

- Tessellation, Geometry Shader
- ASTC テクスチャ圧縮
- Floating point render targets
- and more...

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

まとめ

- モバイルグラフィックスAPI進化
  - 現在、主流となっているのは ES2.0。最新は ES3.2
  - ES3.2 は DirectX11.0 世代と同等の機能を有している

Copyright Drecom Co., Ltd. All Rights Reserved.

グラフィックスAPI のパラダイムシフト

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

GPUの描画

従来のグラフィックスAPI と GPU の関係について。
まずは、アプリケーションから Draw 関数を呼び出すと何が起こるかを見る。

DrawCall とは？

グラフィックスAPI の描画処理に関する関数呼び出しのこと。
ドライバは Draw 関数を一つの基準として処理する。

glDrawElements()
D3D11DeviceContext::DrawIndexed()

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

GPUの描画

１：アプリケーションからグラフィックスAPI の呼び出しを行うと、
ドライバがメインメモリ上のコマンドバッファに内容を蓄積する

２：DrawCall の呼び出しで コマンドバッファが一杯になる、あるいは Flush, Present が呼び出されるとドライバは GPU へのフラッシュを準備する

3：ドライバが描画コマンドを GPU が解釈できる命令に変換し、同時に描画ステートのバリデーションを行う

４：GPU にコマンドが転送され処理される

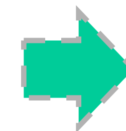
※ ハードウェアにより実際の処理は異なるので留意

Copyright Drecom Co., Ltd. All Rights Reserved.

GPUの描画

基本的にドライバは Draw 関数を一つの基準として処理し、
描画に関するステートをパッケージ化してコマンドバッファに積み、GPU 転送時にバリデーションしていた。これらの処理はドライバが暗黙のうちに行う。

```
glClearColor(0.65f, 0.65f, 0.65f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glBindVertexArrayOES(_vertexArray);
glUseProgram(_program);
glDrawArrays(GL_TRIANGLES, 0, 36);

glClearColor(0.65f, 0.65f, 0.65f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glBindVertexArrayOES(_vertexArray);
glUseProgram(_program);
glDrawArrays(GL_TRIANGLES, 0, 36);
```



Draw Command
clearColor = 0.65f, 0.65f, …
clearFlag = color | depth …
vertexNum = 36 …

Draw Command
clearColor = 0.65f, 0.65f, …
clearFlag = color | depth …
vertexNum = 36 …

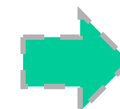
Copyright Drecom Co., Ltd. All Rights Reserved.

グラフィックスAPIのオーバーヘッド

OpenGL ES では一部ベンダーの拡張を除いて、標準機能としては
シェーダーのオフラインコンパイルをサポートしていない。

シェーダーの字句解析、構文解析、最適化、命令コードの生成の全ては
ドライバがランタイム上で実行している。

```
void main()
{
    vec3 eyeNormal = normalize(normalMatrix * normal);
    vec3 lightPosition = vec3(0.0, 0.0, 1.0);
    vec4 diffuseColor = vec4(0.4, 0.4, 1.0, 1.0);
    float nDotVP = max(0.0, dot(eyeNormal, normalize(light)));
    colorVarying = diffuseColor * nDotVP;
    gl_Position = modelViewProjectionMatrix * position;
}
```



実行時に
GPU のマシンコードへ

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

グラフィックスAPIのオーバーヘッド

余計なオーバーヘッドはCPU負荷を生み、CPU負荷は発熱とバッテリー消費を招く
これが、スマートフォン上では大きな問題となりえていた。

静的に解決できるものは静的に解決し、
効率化できるものは効率化し、
ランタイムでの CPU 負荷を削減することはできないか？

低レベルグラフィックスAPIが生まれた

Copyright Drecom Co., Ltd. All Rights Reserved.

低レベルグラフィックスAPI

2013年以降 グラフィックスAPI に大きな変革が起きている。

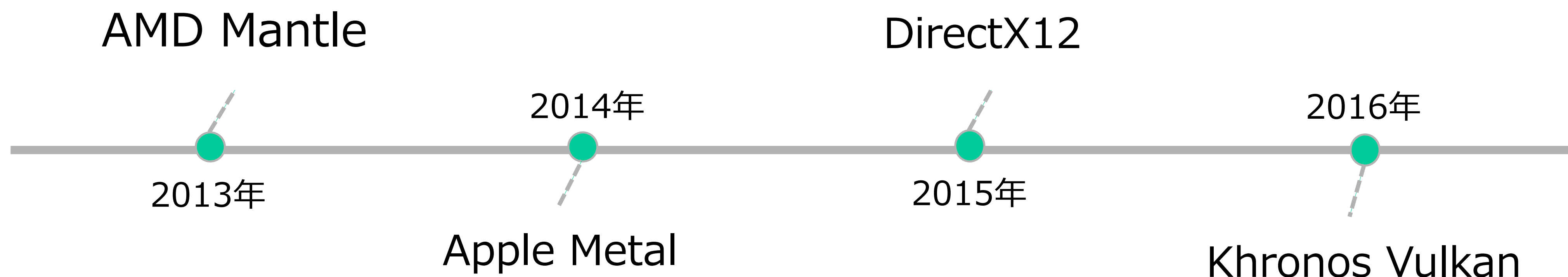
AMD Mantle, Apple Metal, DirectX12, Khronos Vulkan など、
新しい API が登場した。

コンシューマゲーム機では一般的であった、低レベルAPI の思想が一般化

より効率的に、より明示的に CPU・GPU を活用する
→ CPU のボトルネックを削減する

Copyright Drecom Co., Ltd. All Rights Reserved.

低レベルグラフィックスAPI



Copyright Drecom Co., Ltd. All Rights Reserved.

低レベルグラフィックスAPI

Metal, Vulkan は最新のモバイルレンダリングアーキテクチャを最大限に
活用する機能も含まれている。例えば一般的な下記3つの GPU に対応する。

- Immediate Mode Rendering (IMR)
- Tile Based Rendering (TBR) - Mali, Adreno
- Tile Based Deffered Rendering (TBDR) – PowerVR

詳細は後述。

Copyright Drecom Co., Ltd. All Rights Reserved.

Apple Metal

Apple が策定 2014年9月にリリース（iOS8～、OSX El Capitan～)
iOS の対応が先行して行われ、翌年に mac OSX のサポートも行われた。

2016年6月の WWDC2016 でも、Metal のアップデートが発表された。
大きな更新としては、iOS で初めて テッセレーション(※) がサポートされる。

※ mac OSX sierra / iOS10 の iPhone6s, iPhone6s Plus, iPhone SE 以降のデバイスが必須
A9プロセッサー(PowerVR 7XT) でないと、ハードウェアがテッセレータを搭載していないため

Copyright Drecom Co., Ltd. All Rights Reserved.

Khronos Vulkan

2016年2月に Vulkan 1.0 がリリース
モバイルからデスクトップまで幅広いプラットフォームをサポートしている

当初、glNext と呼ばれていた。
LunarG 社より Windows/Linux の Vulkan SDK が提供されている
Google からも Vulkan ビルドに対応した、NDK がリリースされています。

Copyright Drecom Co., Ltd. All Rights Reserved.

Modern mobile architecture

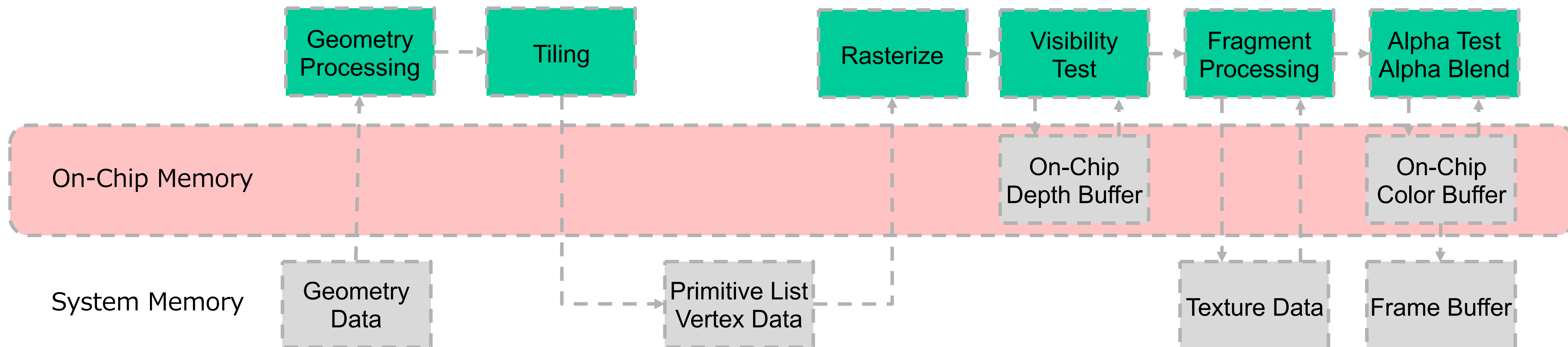
描画時はフレームバッファ(Color, Depth, Stencil)へのアクセスに注意。
フレームバッファのサイズが大きい場合 Depth / Stencil Buffer など、
システムメモリアクセスに伴う、フレームバッファの転送の負荷が大きい。

この負荷を下げるために、最新のモバイル GPU の特徴として
Tile Based Rendering アーキテクチャを採用している。

Copyright Drecom Co., Ltd. All Rights Reserved.

Tile Based Rendering

レンダリング時は可能な限りシステムメモリへのアクセスを避ける。
高速な On-Chip 上のメモリに置かれたタイルバッファへの読み書きを行う。

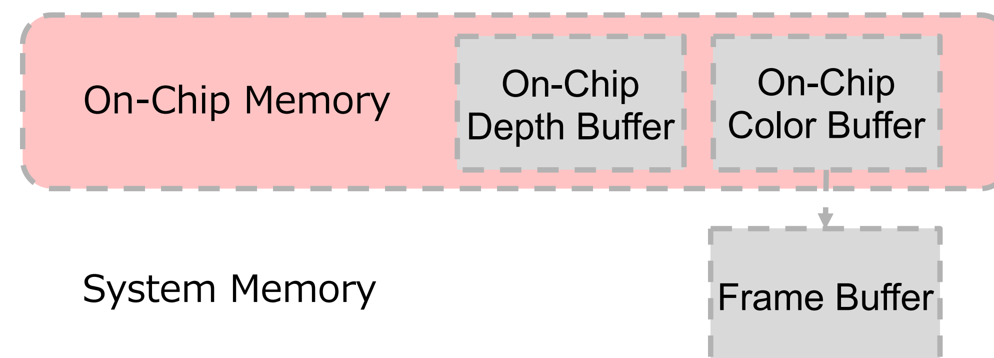


Copyright Drecom Co., Ltd. All Rights Reserved.

Tile Based Rendering

Metal, Vulkan はこの TBR アーキテクチャに最適化されている。
描画時に高速な On-Chip 上のバッファだけでレンダリングを完結させることが出来る。

そのため負荷の高いシステムメモリへのアクセスを避けることが可能で、
これにより消費電力が削減できる。



Copyright Drecom Co., Ltd. All Rights Reserved.

Tile Based Rendering

レンダリング時におけるフレームバッファのアクセスは

Metal : `MTLRenderPassDescriptor`
Vulkan : `VkAttachmentDescription` 構造体に指定する。

それぞれ loadAction(op), storeAction(op) というメンバ変数を持っており、
レンダリングパス開始時、終了時のフレームバッファの取り扱いを設定する。

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Tile Based Rendering

loadAction(op)

- Clear … フレームバッファの内容を指定の色でクリア
- Load … フレームバッファの内容を読み込む (On-chip へのコピー発生)
- DontCare … 全てのピクセルが更新される際に指定

storeAction(op)

- Store … フレームバッファへ書き出す
- DontCare … Depth/Stensil などフレームバッファへ書き出す必要がない場合
  (プラットフォームごとに最適なパフォーマンス結果となる)

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

静的に解決する

低レベルグラフィックスの思想として、静的に決定できる物は静的に解決する
→ ラインタイム時にかかる CPU パワーを排除

- パイプラインステート
- シェーダーコンパイル
- 描画のバリデーション

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

パイプラインステート

Pipeline State Object とは描画に関する情報のこと。

OpenGL ES 世代では個々に指定されていたが、一まとまりで定義されるようになった。ステートは生成時に検証されるため、実行時のオーバーヘッドがない。

GPU はこの Pipeline State オブジェクトを差し替えるだけで、次のレンダリングに移ることができる。

Metal : MTLRenderPipelineState

Vulkan : VkPipeline　オブジェクトに保持される。

Copyright Drecom Co., Ltd. All Rights Reserved.

シェーダーコンパイル

Metal, Vulkan ともにシェーダーのオフラインコンパイルをサポート

- シェーダーはオフラインでコンパイルする
  - ランタイム時の生成コストを排除

Copyright Drecom Co., Ltd. All Rights Reserved.

バリデーション

OpenGL API 呼び出しのエラーチェックはドライバで実装されており、

- 最後に発生したエラーを glGetError()
- Debug, Release 問わず エラーチェック

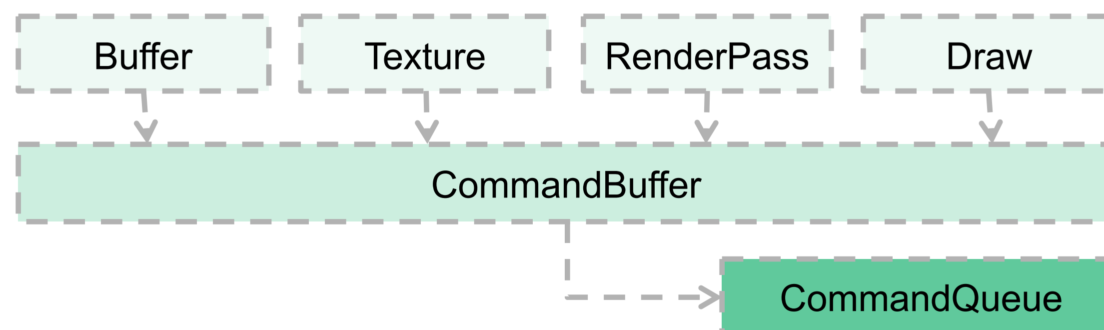
Metal, Vulkan 世代では、明示的に Validation Layer を有効にすることで、
API 呼び出しのエラーチェックを実施する。

通常は開発時に有効にして、製品版では無効にし CPU のオーバーヘッドを削減する。

Copyright Drecom Co., Ltd. All Rights Reserved.

コマンドバッファ構築

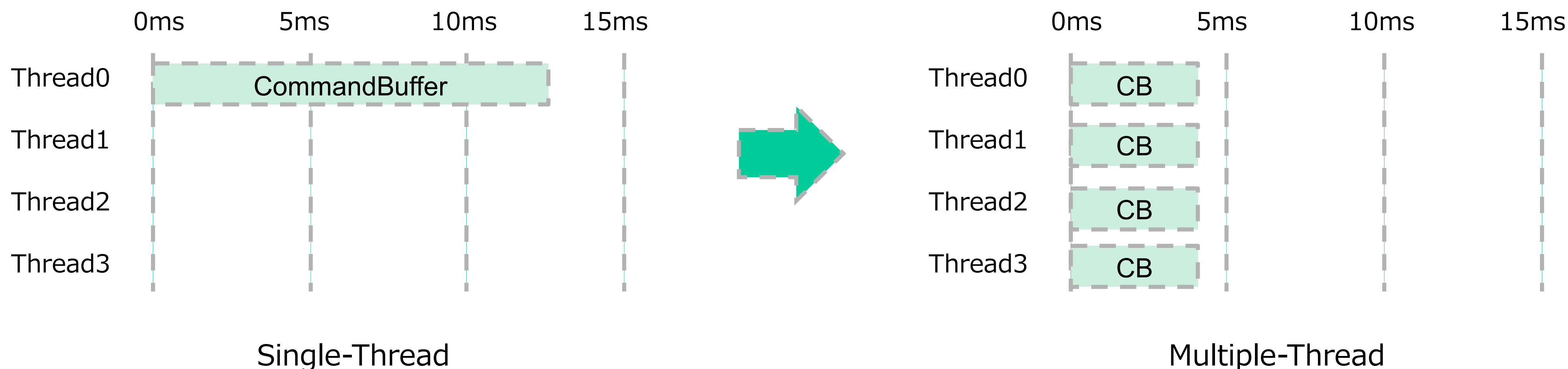
- 従来の グラフィックスAPI ではコマンドバッファ構築はドライバの仕事だった
- Metal, Vulkan 世代ではアプリケーションマターへと変わった
- CommandBuffer に対してコマンドを追加
- CommandBuffer を CommandQueue にサブミットする



Copyright Drecom Co., Ltd. All Rights Reserved.

コマンドバッファ構築

CommandBuffer はマルチスレッドで構築できる
空いた時間で更に多くの Draw を発行するか、またはCPUを休めれる



Copyright Drecom Co., Ltd. All Rights Reserved.

コマンドバッファ構築

Metal と Vulkan において Queue と Buffer では使い方が異なるので注意

Metal は基本的に一つの Queue 上に Command Buffer を追加していく
Vulkan は graphics と compute で Queue が分かれて個別にサブミットする
(VKSemaphore で Queue の同期を取る）

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Adaptive Scalable Texture Compression (ASTC)

ブロック圧縮アルゴリズムの一種。
iOS であれば PVRTC、 Android であれば ETC が一般的に使われてきた。

Metal, Vulkan 世代では ASTC という統一的な圧縮フォーマットが使用できる。
→ 今までのように、プラットフォームごとに圧縮を分ける必要はない

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Low-Overhead, Low-Level プログラミング

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Metal の環境

iOS, macOS の Apple プラットフォームのみサポート
開発環境も macOS 上の Xcode が必須となる。

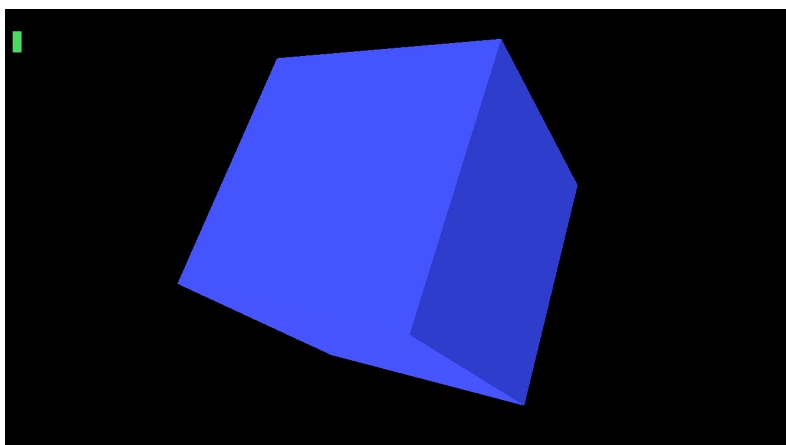
開発環境：OSX El Capitan / Xcode7　以降
実行環境：iOS 8 / iPhoen5s (Apple A7) / 以降
2012年以降に発売された MacBook, iMac など

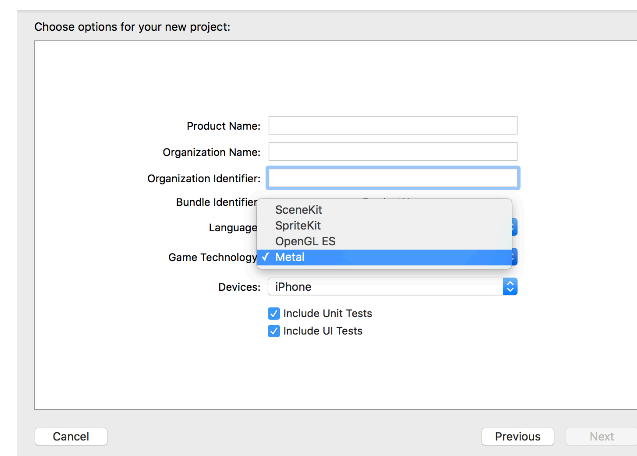
Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Metal の開発環境

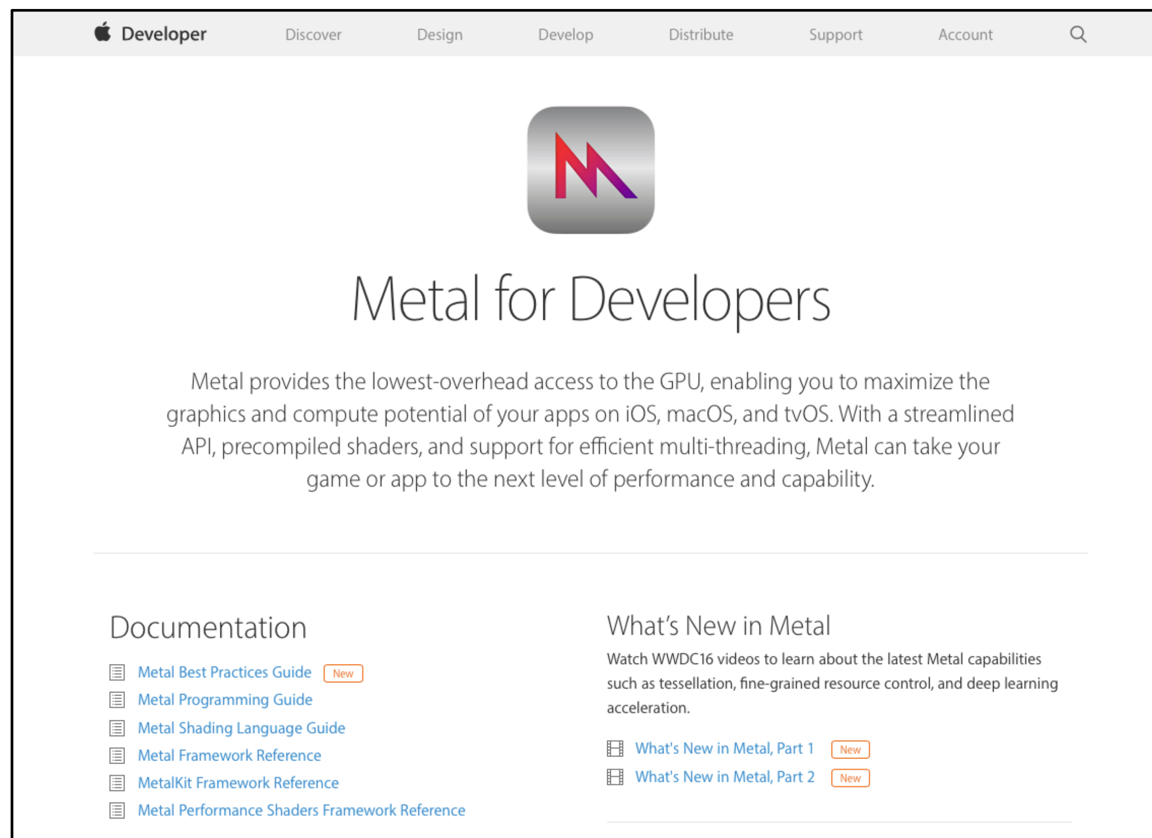
最新の macOS と Xcode と iOS デバイスがあれば開発出来る。
Xcode の新規プロジェクト作成より Game -> Metal を選べば、
テンプレートプロジェクトが作成される。





Copyright Drecom Co., Ltd. All Rights Reserved.

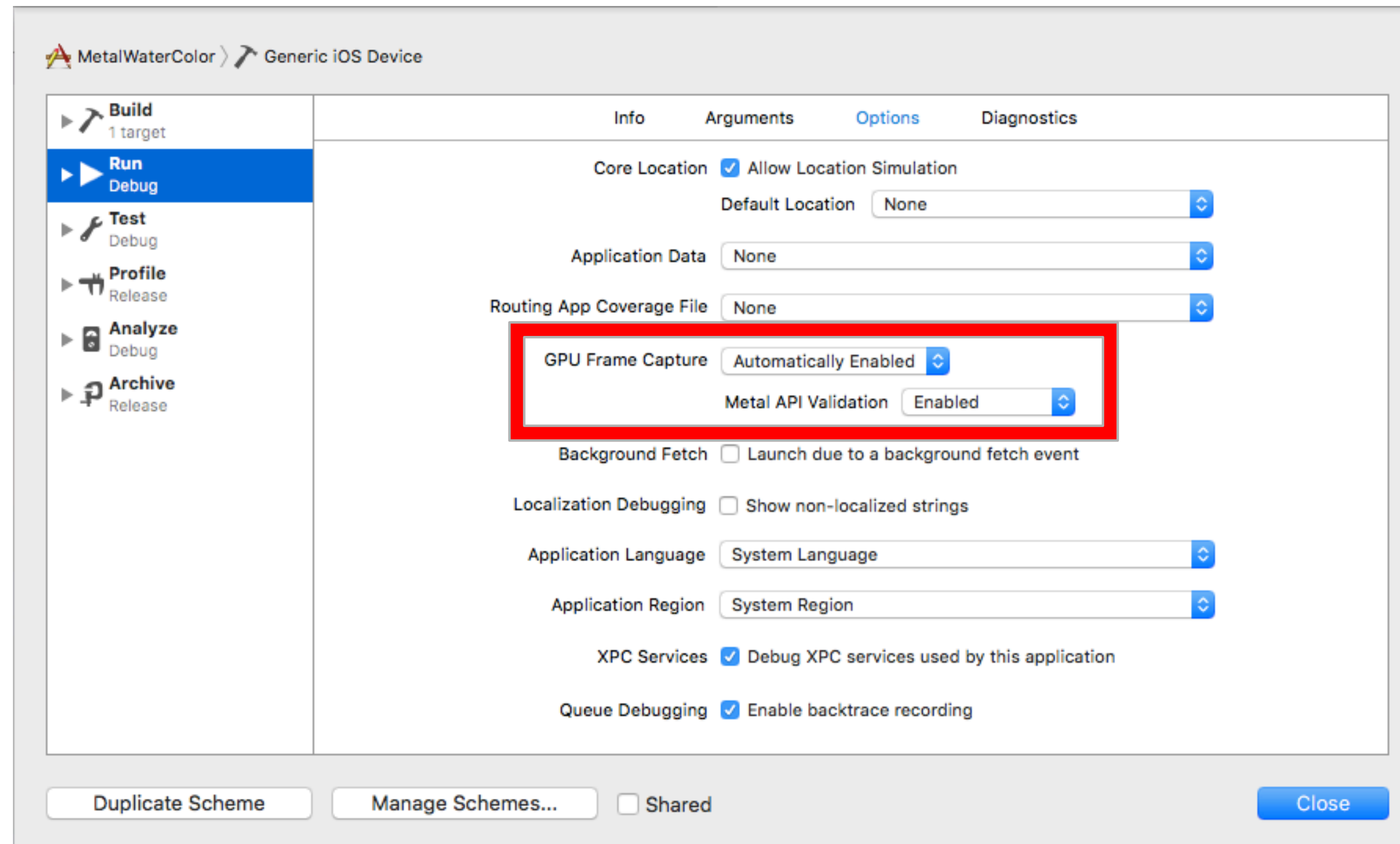
Metal の開発環境



https://developer.apple.com/metal/

Copyright Drecom Co., Ltd. All Rights Reserved.

Validation Layer



Copyright Drecom Co., Ltd. All Rights Reserved.

Metal Programming

- MTLDevice
- MTLCommandQueue
- MTLLibrary
- MTLFunction(Shader)
- MTLBuffer
- MTLRenderPipelineState
- MTLDepthStencilState
- MTLRenderPassDescriptor
  - MTLRenderPassColorAttachmentDescriptor
  - MTLRenderPassDepthAttachmentDescriptor
- MTLCommandBuffer
- MTLRenderCommandEncoder

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Metal Programming

以降のスライドで実際の Metal のコードを記述していますが、
説明のため重要コードのみピックアップして記載している点に注意して下さい！

完全なソースは Apple Developer のサンプルを参考にして下さい。

Copyright Drecom Co., Ltd. All Rights Reserved.

Metal Programming

ベースとなる初期化部分。
デバイスの取得、コマンドキュー、ライブラリは下記のコードで取得。
ライブラリはシェーダーのアーカイブ。

```
id<MTLDevice>       device          = MTLCreateSystemDefaultDevice();

id<MTLCommandQueue> commandQueue    = [device newCommandQueue];
id<MTLLibrary>      defaultLibrary  = [device newDefaultLibrary];
```

Copyright Drecom Co., Ltd. All Rights Reserved.

Metal Programming

シェーダー作成。

```
[defaultLibrary newFunctionWithName:@"lighting_fragment"];
[defaultLibrary newFunctionWithName:@"lighting_vertex"];
```

Copyright Drecom Co., Ltd. All Rights Reserved.

Metal Programming

パイプラインステートを設定

```
MTLRenderPipelineDescriptor *pipelineDescriptor      = [MTLRenderPipelineDescriptor new];
pipelineDescriptor.vertexDescriptor                  = vertexDescriptor; // 頂点レイアウト
pipelineDescriptor.vertexFunction                    = vertexFunction;   // Vertexシェーダー
pipelineDescriptor.fragmentFunction                  = fragmentFunction; // Fragmentシェーダー
pipelineDescriptor.colorAttachments[0].pixelFormat   = MTLPixelFormatBGRA8Unorm;
pipelineDescriptor.depthAttachmentPixelFormat        = MTLPixelFormatDepth32Float;
```

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Metal Programming

パイプラインパスを設定

```
id<MTLRenderPassDescriptor> _renderPassDescriptor = [MTLRenderPassDescriptor renderPassDescriptor];

MTLRenderPassColorAttachmentDescriptor *colorAttachment = _renderPassDescriptor.colorAttachments[0];
colorAttachment.loadAction       = MTLLoadActionClear;
colorAttachment.storeAction      = MTLStoreActionStore;
colorAttachment.clearColor       = MTLClearColorMake(0.65f, 0.65f, 0.65f, 1.0f);

MTLRenderPassDepthAttachmentDescriptor *depthAttachment = _renderPassDescriptor.depthAttachment;
depthAttachment.loadAction       = MTLLoadActionClear;
depthAttachment.storeAction      = MTLStoreActionDontCare;
depthAttachment.clearDepth       = 1.0;
```

Copyright Drecom Co., Ltd. All Rights Reserved.

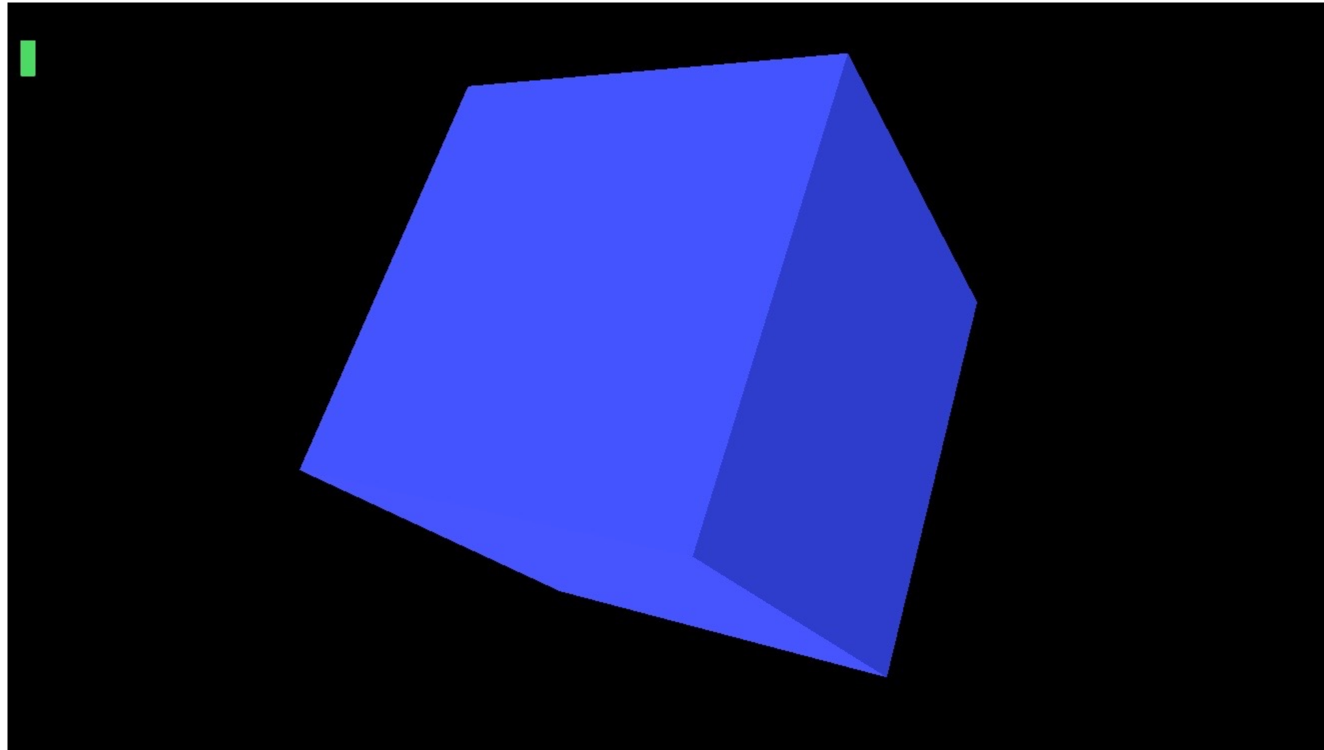
Metal Programming

コマンドバッファの構築
Metal は CommandEncoder という仕組みで、コマンドバッファに追加

```
id <MTLCommandBuffer> commandBuffer = [_commandQueue commandBuffer];
id <MTLRenderCommandEncoder> renderEncoder =
                          [commandBuffer renderCommandEncoderWithDescriptor:desc];

    [renderEncoder setDepthStencilState:_depthState];
    [renderEncoder setRenderPipelineState:_pipelineState];
    [renderEncoder setVertexBuffer:_vertexBuffer offset:0 atIndex:0 ];
    [renderEncoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0 vertexCount:36];
    [renderEncoder endEncoding];

[commandBuffer presentDrawable:view.currentDrawable];
[commandBuffer commit];
```

Copyright Drecom Co., Ltd. All Rights Reserved.

Metal Programming



Copyright Drecom Co., Ltd. All Rights Reserved.

Vulkan の環境

Vulkan はクロスプラットフォームに対応する。
2016年8月現在、Windows, Linux, Android, Tizen と言った、
各種プラットフォームをサポートしている。

Windows / Linux の SDK が LunarG 社より提供されている。

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Vulkan の実行環境

Android 7.0 (Nougat) 以降でサポート

- 2016年3月より開発者向け Preview と NDK がリリース
- 2016年8月23日に正式版リリース
- 現時点の対応端末は Nexus 6, Nexus 5X, Nexsus 6P, Nexsus 9.

- Android Studio にインポートできるサンプルが提供されている
  Vulkan-basic-samples
  https://github.com/googlesamples/vulkan-basic-samples

Copyright Drecom Co., Ltd. All Rights Reserved.

SPIR-V

Vulkan は SPIR-V と呼ばれる、
中間バイナリーデータをサポートしている。
→ Graphics Shader, Compute Kernel などは SPIR-V でロードする

GLSL/HLSL でシェーダーを記述し、ビルド時に SPIR-V を生成する

Copyright Drecom Co., Ltd. All Rights Reserved.

Vulkan Programming

Vulkan は vkInstance を生成することから始める。
プラットフォームごとの描画サーフェスや、実行に必要なレイヤーなどシステムへのアクセスは vkInstance を通して行われる

```
VkInstance inst;
vkCreateInstance(&inst_info, NULL, &inst);
```

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Vulkan Programming

vkInstance からシステムの物理デバイスを取得する。
物理デバイスからアプリで使う GPU インターフェース VKDevice が作成できる。

```
VkPhysicalDevice physical_devices;
VkDevice device;
vkEnumeratePhysicalDevices(inst, &gpu_count, &physical_devices);
vkCreateDevice(physical_devices, &deviceInfo, NULL, &device);
```

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Vulkan Programming

VKDevice を通して下記オブジェクト、リソースを生成する。
長くなるので Draw の解説のみに割愛。

vkCreateRenderPass
vkCreateShaderModule
vkCreateGraphicsPipelines
vkAllocateDescriptorSets
vkUpdateDescriptorSets
vkCreateCommandPool
vkAllocateCommandBuffers
vkCreateImage
vkCreateImageView

vkCreateBuffer
vkBindBufferMemory
vkCreateDescriptorSetLayout
vkCreatePipelineLayout
vkCreateDescriptorPool
vkCreateFramebuffer
vkAllocateMemory
vkBindImageMemory

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Vulkan Programming

Vulkan のコマンドバッファは VkCommandBuffer
これは vkAllocateCommandBuffers を通してコマンドプールより確保される。

vkBeginCommandBuffer，vkEndCommandBuffer でコマンドを構築し、
構築したコマンドを vkQueueSubmit でキューに追加する。

コマンドキューは Graphics, Compute, Transfer 用途ごとに分ける必要がある

Copyright Drecom Co., Ltd. All Rights Reserved.

Vulkan Programming

vkBeginCommandBuffer
- vkCmdPipelineBarrier
- vkCmdBeginRenderPass
- vkCmdBindPipeline
- vkCmdBindDescriptorSets
- vkCmdSetViewport
- vkCmdBindVertexBuffers
- vkCmdDraw
- vkCmdEndRenderPass

vkEndCommandBuffer

一度作成したVulkan のコマンドバッファは
次フレームでも再利用が可能

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Metal, Vulkan まとめ

Metal, Vulkna はアプローチが似ており、共通項目が多い

- 片方を理解すれば、もう片方を覚えるのもそこまで大変ではない
- 言語を覚えるというよりも、低レベルAPIの思想を知った方が理解が早い

Copyright Drecom Co., Ltd. All Rights Reserved.

Non-Photorealistic Rendering における水彩画表現

Copyright Drecom Co., Ltd. All Rights Reserved.

Non-Photorealistic Rendering

Non-Photorealistic Rendering (NPR) は、コンシューマゲームに代表される
フォトリアルな絵作りとは対を成すレンダリング技法。

トゥーンシェーダなどのアニメ調の表現も NPR の技法に区分される

Copyright Drecom Co., Ltd. All Rights Reserved.

なぜNPRなのか？

フォトリアルな絵作りに必要な要素を考えてみる

- HDRレンダリング
- 高解像度テクスチャ/ハイレゾメッシュ
- Physically Based Rendering
- Global Illumination
- 高品質なアンチエイリアス
- カラーグレーディング
- Depth of Fields
- 多数の動的ライト

これらの処理をモバイルで実現？

Copyright Drecom Co., Ltd. All Rights Reserved.

モバイルのジレンマ

モバイルが抱える問題にはバッテリー消費/発熱の問題がある。
演算一回、メモリアクセス一回にも僅かながらバッテリーを消費しており、時間とともに某大なバッテリー消費へと発展する。

CPU/GPUに余裕があっても、追加のグラフィックス用途を組み込む、
ということがやり辛い環境にある。

現実的なバッテリー消費と絵作りの品質について、そのトレードオフを考える

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM
with entertainment

NPRへのモチベーション

現実的なタスク量で実現可能な絵作りとは。
コンシューマはハードウェアと技術の進歩により現実的なタスク量での
フォトリアルレンダリングを実現する手段を得た。

モバイルもフォトリアルな絵作りのために、ハードウェアの進化を待つか？
→ GPU の性能向上は、画面の高解像度化に相殺される可能性もある

それよりプラットフォームが異なり、提供できるゲームの形が異なるのだから
絵作りの方向性もまた、コンシューマのリアルとは別路線に振り切ってみる

Copyright Drecom Co., Ltd. All Rights Reserved.

水彩画を表現

※ 本物の水彩画です

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

NPRの利点

- 多彩なアプローチ
  → PBR 実装のような、物理現象への依存が少ないため実装が容易
  → 視覚的効果が高い実装の追求が可能

- オブジェクト描画時に、統一的なマテリアルになることが期待できる
  → バッチ描画できる可能性が増える
  → 質感はポストプロセスで掛ける。基本は1マテリアルで描画できる

Copyright Drecom Co., Ltd. All Rights Reserved.

NPRの利点

- 質感をポストポロセスへ
  → 描画マテリアルの簡素化
  → 描画オブジェクト数に依存せずに、NPR に関する負荷を固定で見積もる

- 繊細な表現ではなく、視認性の高さを評価し採用
  → 例えば、日本アニメとの親和性の高さなど

アニメ表現はほぼ一般的に行われているため、
今回は同じ題材を選ぶよりは思考を変えて水彩画に挑戦する。

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

水彩画の特徴

- 水彩画に使われる水彩紙の表現

紙肌の粗目による凹凸感、

- 絵の具のにじみ、ぼかし、かすれ表現

輪郭部分は他の色と混ざり合い、グラデーションがかかる

- デジタル特有のポスト加工処理の追加

水彩表現をベースとした上で、更にゲームの世界観の向上へ

Copyright Drecom Co., Ltd. All Rights Reserved.

水彩紙の表現

水彩紙とは、透明水彩や不透明水彩などの水彩絵具を用いた描画に適した専用紙の総称。

紙質によっては高い吸水性を保持しており、水分量を調整することで、にじみやぼかし効果に大きな影響を与えることができる。

http://zokeifile.musabi.ac.jp/%E6%B0%B4%E5%BD%A9%E7%B4%99/

Copyright Drecom Co., Ltd. All Rights Reserved.

絵の具のにじみぼかし

水彩は絵の具に含まれる水分と、水彩紙特有の吸収性により、にじみやぼかしが効果が起こる。

塗り方によりエッジ部分は他の色と薄く混ざり合う

DRECOM with entertainment

水彩表現のアプローチ

Final result

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM
with entertainment

Final result

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

水彩表現のアプローチ

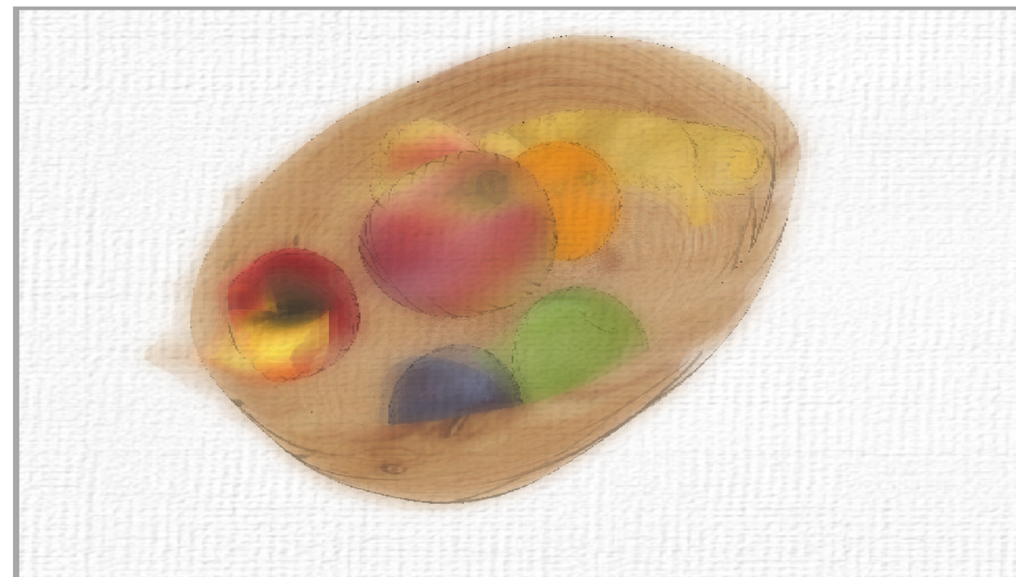
Texture only

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

水彩画表現のアプローチ

Texture only



Final result

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

水彩画表現のアプローチ

- Multiple Render Target で G-Buffer にシェーディング情報の書き出し
  - G-Buffer は2枚使用
  - DiffuseMap と ハッチングを含むアウトライン線
  - 絵の具のにじみ方向を書き込んだ VectorMap

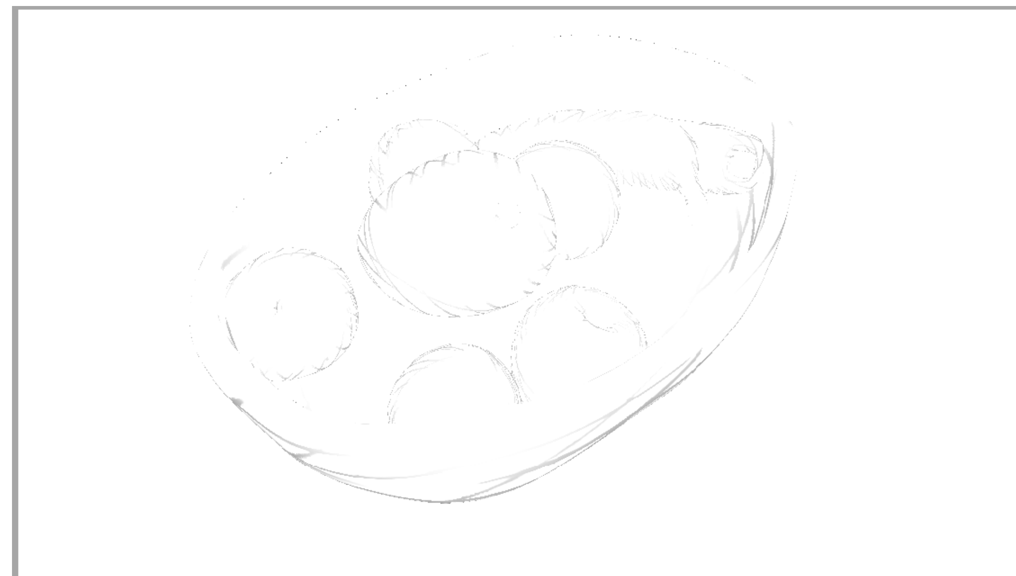
- Compute Kernel で水彩表現を処理
- コンポジット

Copyright Drecom Co., Ltd. All Rights Reserved.

G-Buffer への書き出し

G-Buffer1

DiffuseMap

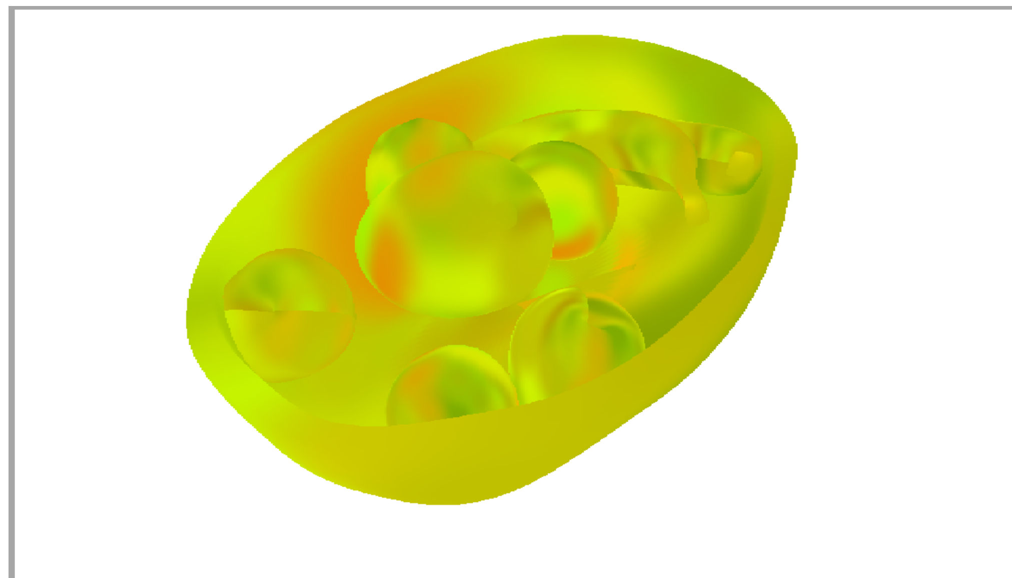


Hatching + Outline

Copyright Drecom Co., Ltd. All Rights Reserved.

G-Buffer への書き出し

G-Buffer2

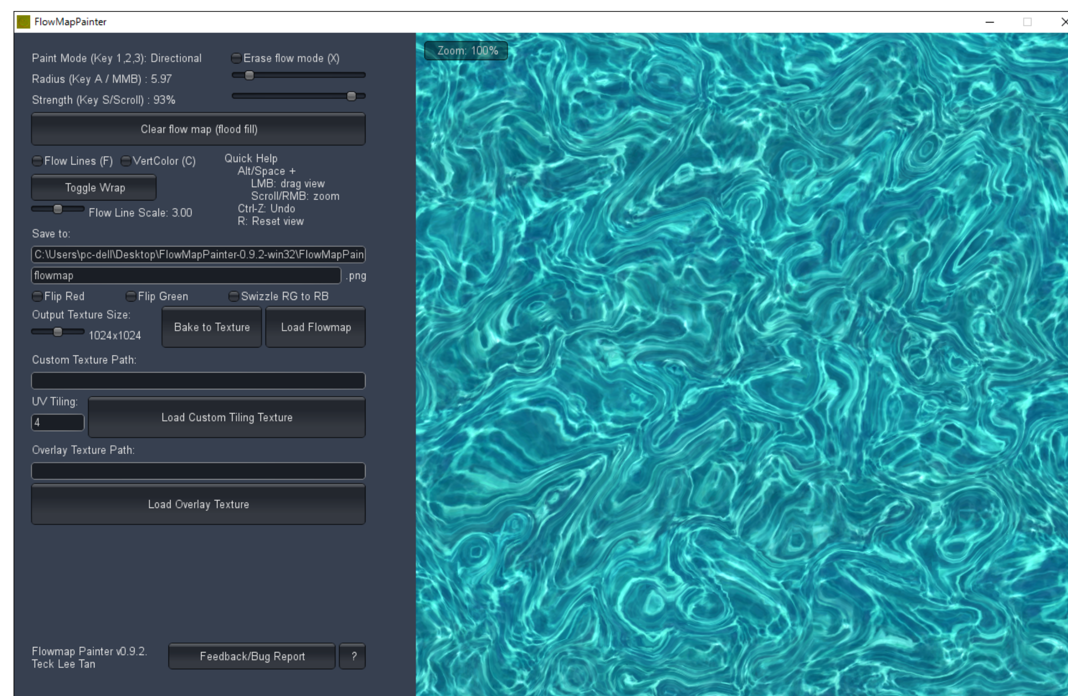


VectorMap

Copyright Drecom Co., Ltd. All Rights Reserved.

G-Buffer への書き出し

モデルに適用する VectorMap は FlowMapPainter を使用して作成



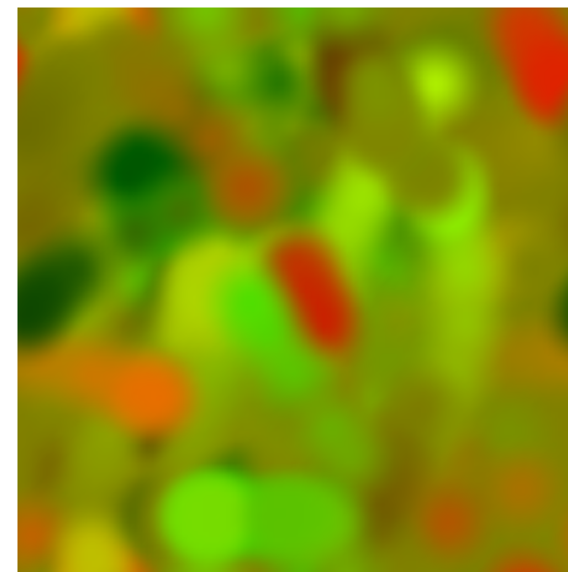
http://teckartist.com/?page_id=107

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

G-Buffer への書き出し

FlowMap の XY には流れの方向が記述されている。
この 流れ を、水彩のストロークとして使用
このベクトルの移動量をにじみの方向に



Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

G-Buffer への書き出し

```
float3  diffuseColor    = diffuseTexture.sample(samplr, vert.texCoord).rgb;
float3  normal          = normalize(vert.normal);
float3  eyeDirection    = normalize(vert.eyePosition);
// NdotE
half    NdotE       = dot( normal, eyeDirection );
// hatching
half    hatching    = hatchingTexture.sample(samplr, vert.texCoord).x;
hatching            = mix(hatching, half(1.0f), half(0.15f ));
// Outline
half4   edgeColor   = half4( half3( 0.6f ), 1.0f );
half    edgePower   = 5.0f;
half4   outline     = saturate( half4( NdotE * edgePower * hatching  ) + edgeColor );
// Flowmap
float4  flowmap     = flowmapTexture.sample(samplr, vert.texCoord);
flowmap.xy          = (flowmap.xy * float2(0.5f) ) + float2(0.5f);

MX::FragOutput output;
output.albedo       = float4( diffuseColor, outline.x );
output.normal       = float4( flowmap.xyz, 1.0f );
return output;
```

Fragment shader

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

G-Buffer で書き出した2枚のテクスチャを Compute Kernel へ入力する。
書き出した VectorMap の情報をもとに DiffuseMap のテクスチャ位置をずらして
サンプリングする。

```
half4   inColor   = inTexture.read(gid);
half4   inColor2  = inTexture2.read(gid);

half    flowPower = 80.0f;
uint2   coord = uint2( gid.x + (int)( inColor2.x * flowPower ),
                       gid.y + (int)( inColor2.y * flowPower ));

half4   flow = inTexture.read( coord );
```

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Compute Kernel

VectorMap 適用した結果

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Compute Kernel

思ってた結果にならない…
もっとぐにゃっと曲がってほしい。

殆どのモデルの VectorMap サンプリング位置が均一になっている
　→ VectorMap の方向ベクトルとモデルの UV 位置が問題

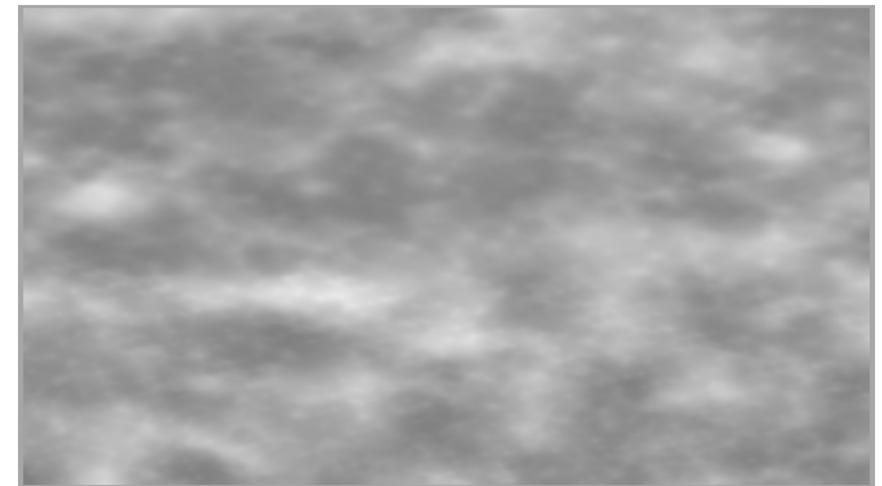
ただ、モデルごとに個別に VectorMap を用意するのも、
UV を修正するのもやりたくない。

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

回避策

- VectorMap にノイズで重み付けを行う
- サンプリングした VectorMap に揺らぎをつける



Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Compute Kernel

ランタイム上の Compute Kernel で Perlin Noise を生成する

→　起動時に生成して G-Buffer と一緒に Kernel へ入力する

```
half rand( half2 n ) {
    return fract( sin( dot( n, half2(12.9898f, 4.1414f ) ) ) * 43758.5453f );
}

half perlinNoise( half2 p ){
    half2 ip    = floor(p);
    half2 u     = fract(p);
    u = u * u * (3.0f - 2.0f * u );
    half res = mix( mix( rand( ip ),
                         rand( ip + half2( 1.0f, 0.0f ) ), u.x ),
                    mix( rand( ip + half2( 0.0f, 1.0f ) ),
                         rand( ip + half2( 1.0f, 1.0f ) ), u.x ), u.y);
    return res * res;
}
```

https://gist.github.com/patriciogonzalezvivo/670c22f3966e662d2f83

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

VectorMap only

VectorMap + Perlin Noise

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

元の DiffuseColor と、ずらした DiffuseColor を合成する
合成時は Perlin Noise の重み付けをもとに、色を mix する

```
// パーリンノイズの強度をブレンドの強度にする
half     flowBlend = noise;
half4    originMix = mix( inColor, flow, flowBlend );
```

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Compute Kernel

Diffuse only

ブレンド結果

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

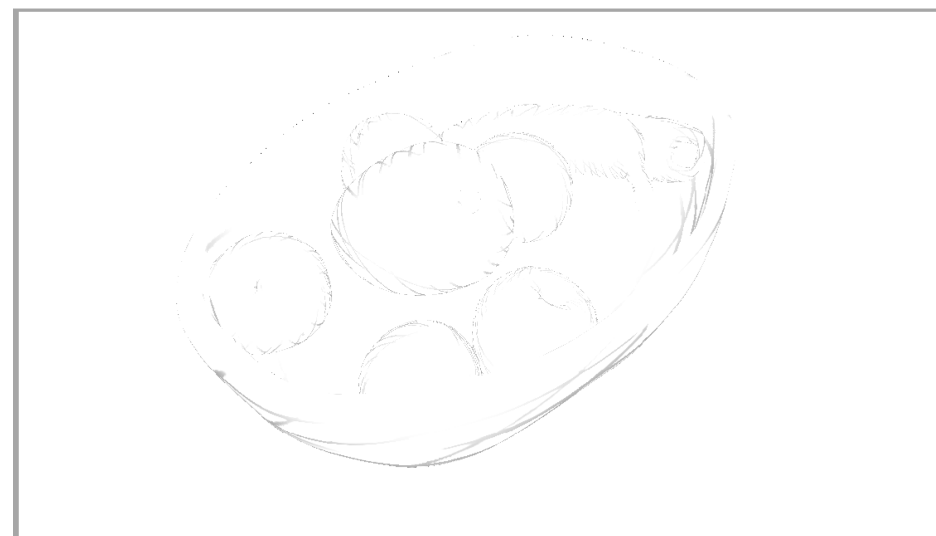
Compute Kernel

```
half4   inColor   = inTexture.read(gid);
half4   inColor2  = inTexture2.read(gid);

half flowPower = 80.0f;
uint2   coord = uint2( gid.x + (int)( inColor2.x * flowPower * noise ),
                       gid.y + (int)( inColor2.y * flowPower * noise ));

half4   flow = inTexture.read( coord );

// 輝度が高すぎるので、輝度を落とす
flow    = mix( flow, half4( 1.0f ), (half)0.3f );
inColor = mix( inColor, half4( 1.0f ), (half)0.1f );

// パーリンノイズの強度をブレンドの強度にする
half    flowBlend = noise;
half4   originMix = mix( inColor, flow, flowBlend );

// ハッチングとアウトラインを書き出す
originMix.w = inColor.w;
outTexture.write( half4( originMix ), gid );
```

Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

Hatching + Outline の線は、次のぼかしパスがあるため、
ComputeKernel からはテクスチャの w 要素に入れて書き出しておく。
後のコンポジットパスで合成する。

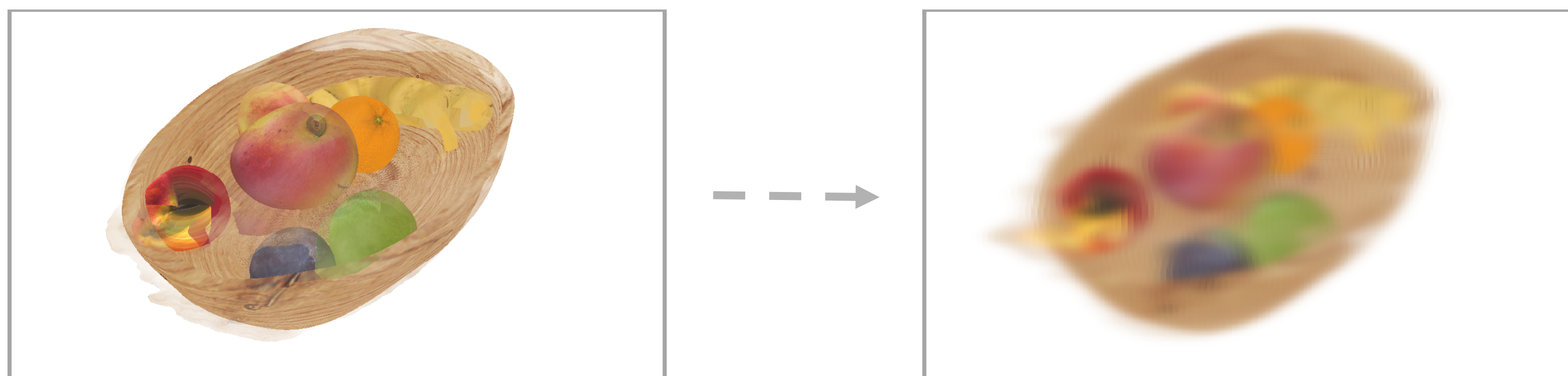


Copyright Drecom Co., Ltd. All Rights Reserved.

Compute Kernel

結果をさらに Compute Kernel で加工する

- 1/2 にダウンサンプリング
- ガウスぼかし



このぼかした結果を最終的なコンポジットパスで合成する

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

コンポジット

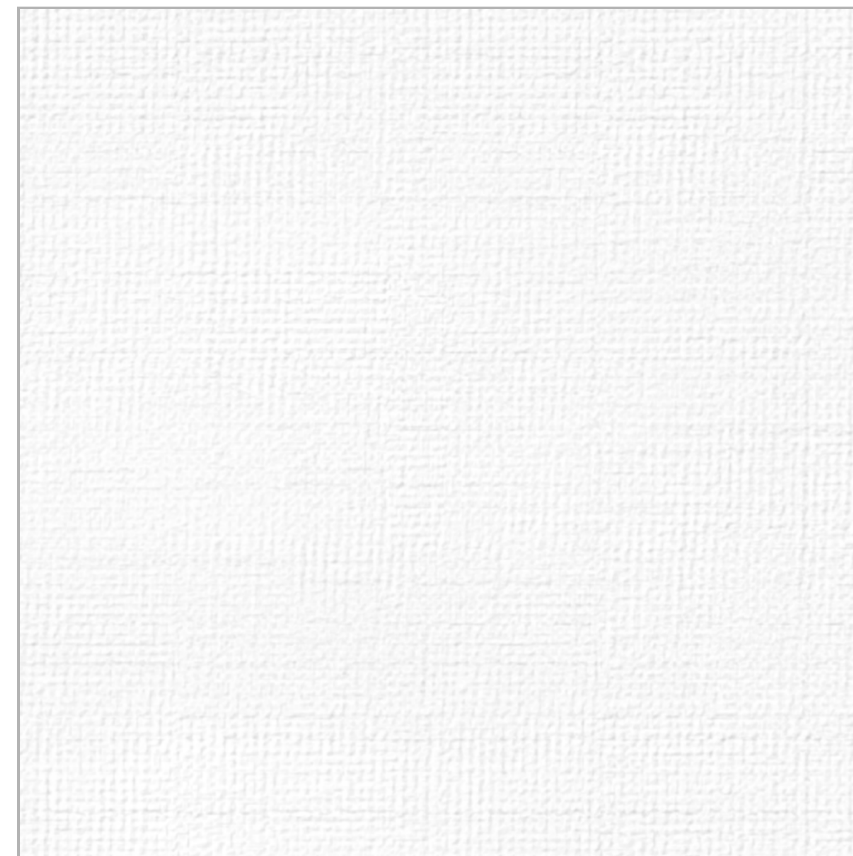
コンポジットの段階では3枚のテクスチャを利用する

- にじみテクスチャ
- ぼかしテクスチャ
- 水彩紙テクスチャ

Copyright Drecom Co., Ltd. All Rights Reserved.

水彩紙テクスチャ

事前に Photoshop のフィルターで作成。
水彩紙の凹凸を表現したテクスチャ
コンポジットで合成する



Copyright Drecom Co., Ltd. All Rights Reserved.

コンポジット

にじみテクスチャとぼかしテクスチャをブレンドする

```
half blendRate = 0.51f;
return mix( color0, color1, blendRate );
```

Copyright Drecom Co., Ltd. All Rights Reserved.

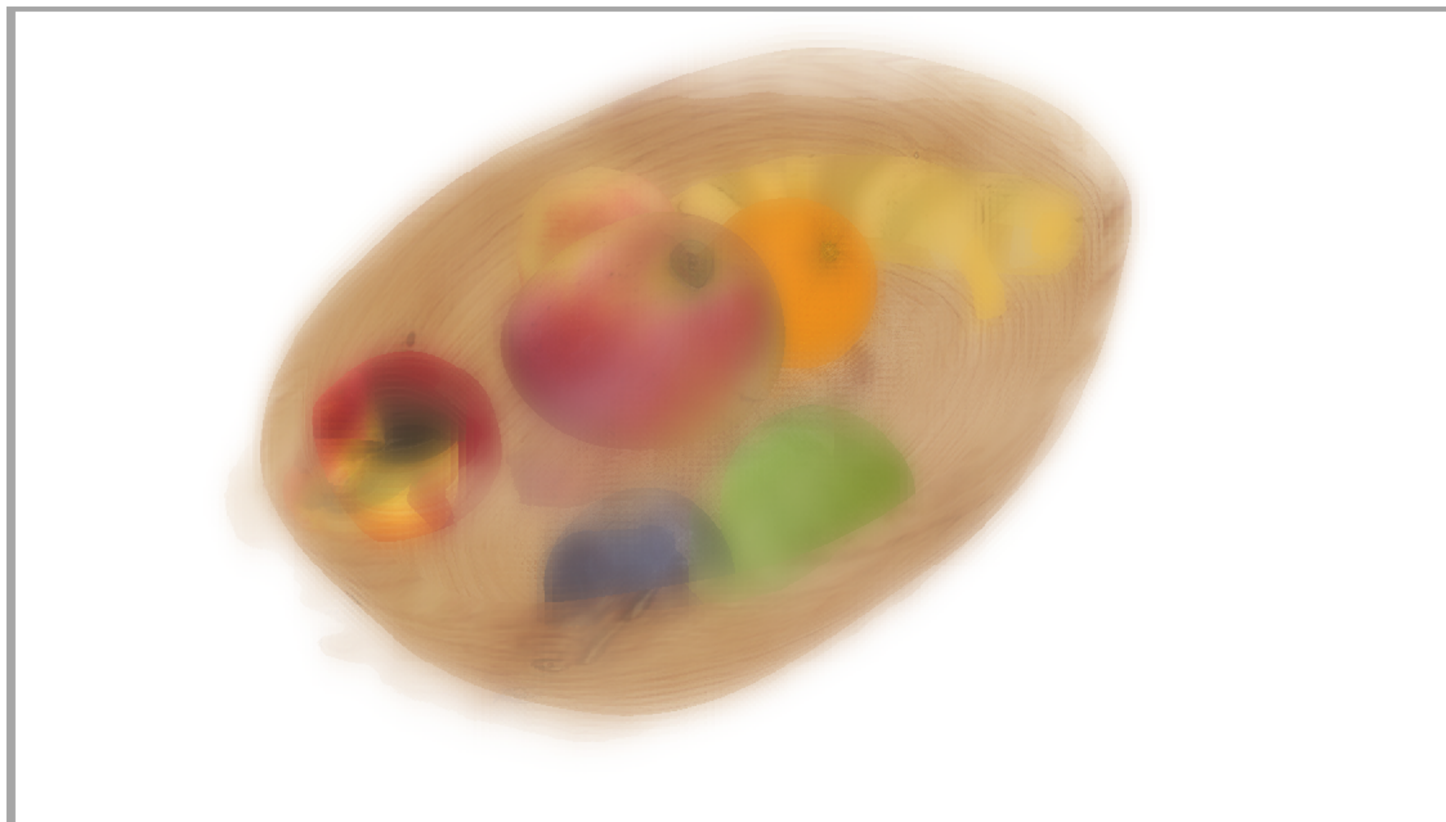
コンポジット

ソフトフォーカスのような効果が得られる

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

コンポジット

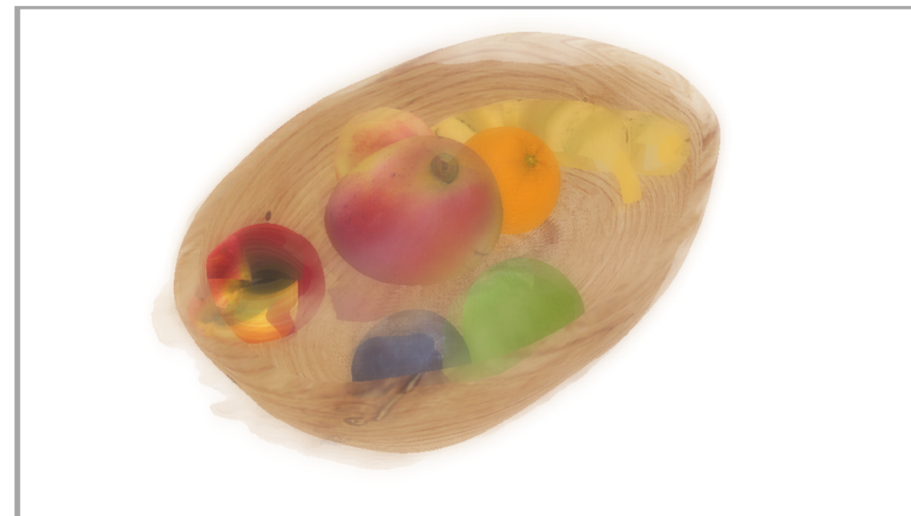


この時ブレンドする BlendRate が高すぎるとぼやけた絵になるので注意

Copyright Drecom Co., Ltd. All Rights Reserved.

コンポジット

ブレンド前



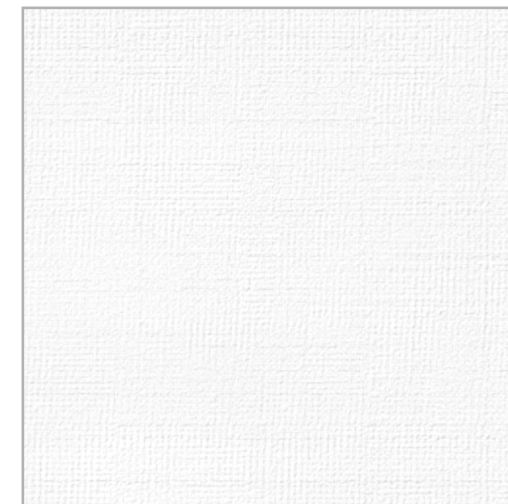
ブレンド後

Copyright Drecom Co., Ltd. All Rights Reserved.

コンポジット

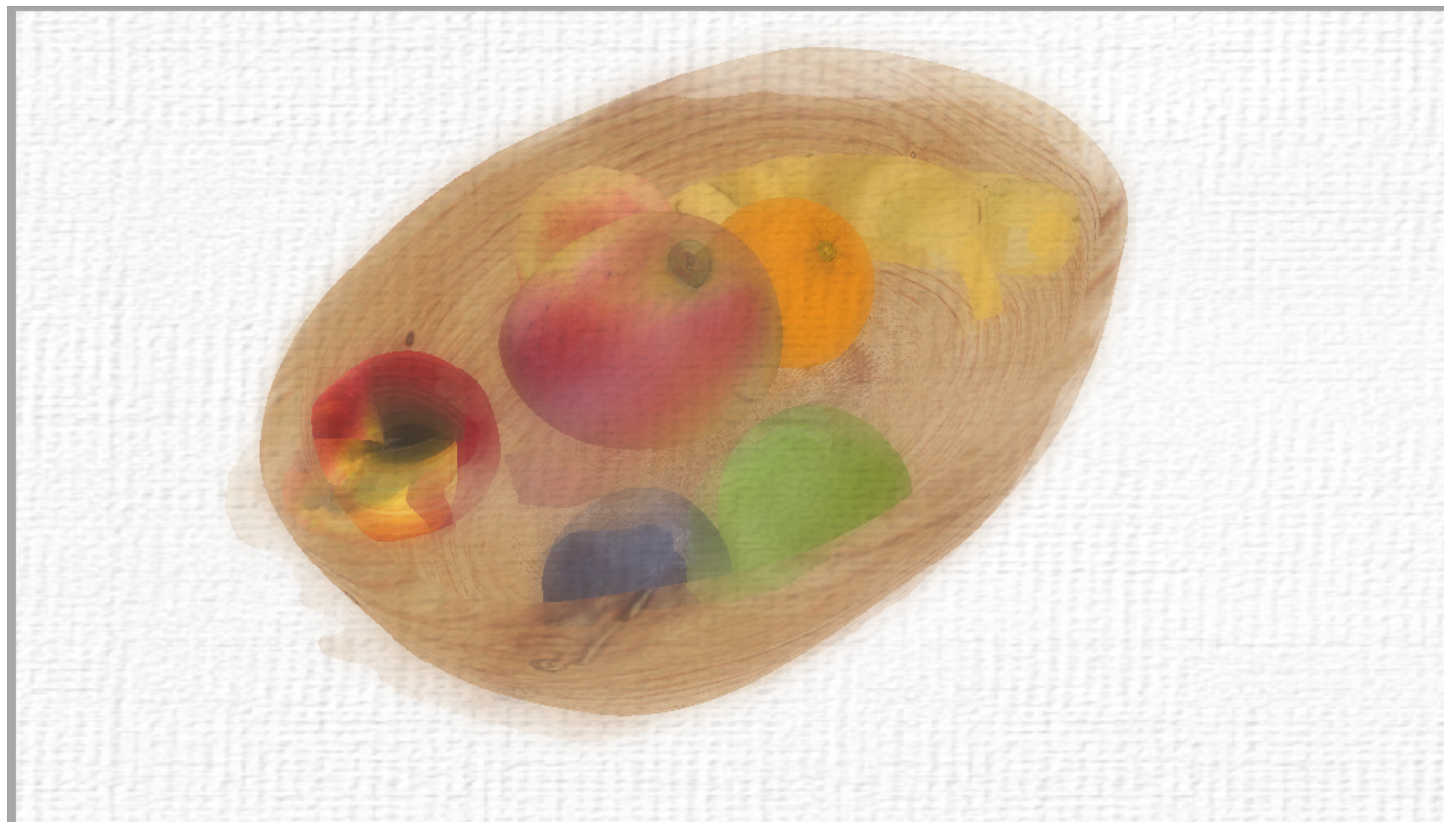
この結果に対して、スクリーン全体に水彩紙テクスチャを追加する。

```
float2  screenUV  = (inFrag.position.xy / uniforms.screenSize);
half4   paperTex  = paperTex2D.sample(quadSampler, screenUV);

half4   color0    = tex2D.sample(quadSampler, inFrag.texCoord);
Half4   color1    = blurTex2D.sample(quadSampler, inFrag.texCoord);

half    blendRate = 0.51f;
return mix( color0, color1, blendRate )  * paperTex;
```



Copyright Drecom Co., Ltd. All Rights Reserved.

コンポジット



Copyright Drecom Co., Ltd. All Rights Reserved.

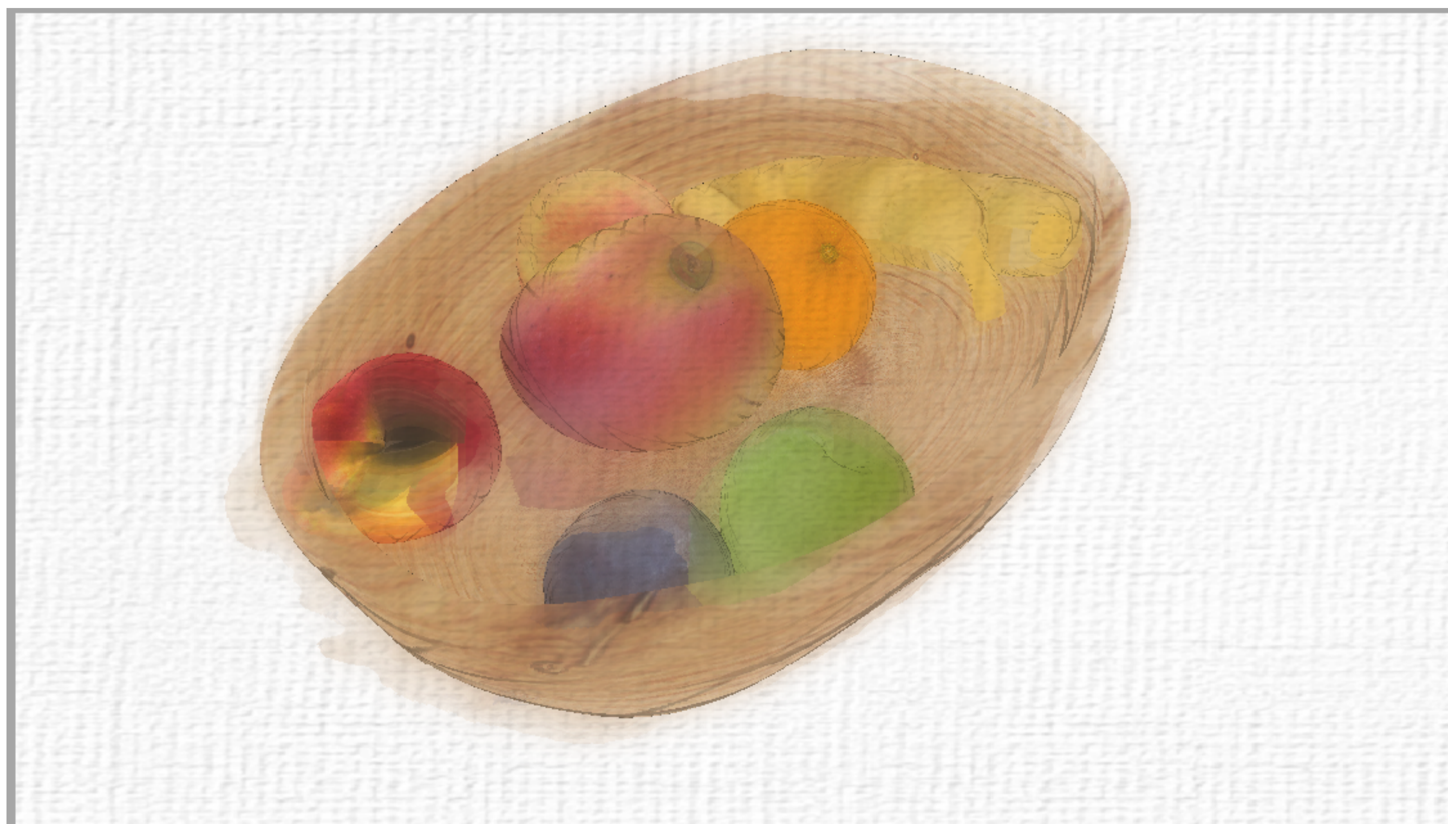
コンポジット

更にハッチングとアウトライン線を追加

```
float2  screenUV  = (inFrag.position.xy / uniforms.screenSize);
half4   paperTex  = paperTex2D.sample(quadSampler, screenUV);

half4   color0    = tex2D.sample(quadSampler, inFrag.texCoord);
Half4   color1    = blurTex2D.sample(quadSampler, inFrag.texCoord);

half    blendRate = 0.51f;
return mix( color0, color1, blendRate ) * color0.w * paperTex;
```

Copyright Drecom Co., Ltd. All Rights Reserved.

コンポジット



Final result

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

Final result

Copyright Drecom Co., Ltd. All Rights Reserved.

まとめ

- モバイルでも NPR は現実的な負荷で処理できる
  - アイディア次第で全く新しいグラフィックス表現が実現できる可能性
- ベースとなる GPU の機能強化と性能向上
  - Metal, Vulkan に対応する GPU そのもののベース性能が向上している
  - 各 GPU 機能の均一化が進み、拡張に依存しない統一的な実装環境
- Shader が書きやすい
  - GLSLでは #include, #define が使えないため、複雑なシェーダーが書き辛い
  - MSL, SPIR-V(変換前のシェーダー) ではコンパイル時にプリプロセッサが働く

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM with entertainment

まとめ

- モバイルでは、処理負荷の1msecを削減することに意味がある
  - 低レベルグラフィックスAPIはモバイルにこそ価値がある
  - バッテリー消費と発熱の問題を考える上で、僅かなオーバーヘッドの削減は、アプリ全体の継続率にも影響を与える

- 今後の低レベルグラフィックスAPI
  - 登場時期の速さから、今は Metal が先行して普及している
  - Vulkan も対応プラットフォーム数の多さから、いずれ Metal に追いつく
  - 2018年頃には モバイルでの低レベルグラフィックスAPI が一般化している

Copyright Drecom Co., Ltd. All Rights Reserved.

DRECOM
with entertainment

ご清聴ありがとうございました。

Copyright Drecom Co., Ltd. All Rights Reserved.