Stage3D & AGAL

Stage3D勉強会 #1

2012/5/25

Profile

高輪 知明

http://nutsu.com

Twitter : @nutsu

Facebook : nutsu

プログラムで絵を描くのが好きです。

テーマ

「ビルトインクラスでのStage3Dプログラムに必要なこと。」

・Stage3D関連APIの利用
・AGALの記述

テーマ

Stage3Dプログラムに必要なことは大きく以下の点になる。

- **Stage3Dの処理イメージ**
- **AGALの概要**
  レジスタ・オペコード

その基礎として…

- **3Dプログラムの知識**

Stage3Dの処理イメージ

まずは、簡単なプログラムを。

プログラムの概要(1)

```
public function Sample1(){
  var stage3d:Stage3D = stage.stage3Ds[0];
  stage3d.addEventListener(Event.CONTEXT3D_CREATE, context3DCreateHandler);
  stage3d.requestContext3D();
}

private function context3DCreateHandler(e:Event):void {
  var ctx:Context3D = Stage3D(e.target).context3D;
  ctx.configureBackBuffer(800, 600, 2);
  //ここから描画処理
}
```

まずは、コンテンツを表示するStage3Dを準備

・Stageから、Stage3Dのインスタンスを取得する。→3D表示用のレイヤー

・Context3Dなるものをリクエストする

・表示サイズなどの設定を行う

プログラムの概要(2)

```
var vertices:Vector.<Number> = new <Number>[
    -0.5,-0.5,0.0, 0.0,0.5,0.0, 0.5,-0.5, 0.0
];
var vb:VertexBuffer3D = ctx.createVertexBuffer(3, 3);
vb.uploadFromVector(vertices, 0, 3);
ctx.setVertexBufferAt(0, vb, 0, Context3DVertexBufferFormat.FLOAT_3);

var indices:Vector.<uint> = new <uint>[0, 1, 2];
var indexBuffer:IndexBuffer3D = ctx.createIndexBuffer(3);
indexBuffer.uploadFromVector(indices, 0, 3);

ctx.setProgramConstantsFromVector(Context3DProgramType.FRAGMENT, 0,
                                  new <Number>[1,0,0,1]);
```

描画するデータの準備

・このプログラムでは、上から頂点座標、インデックス、描画色

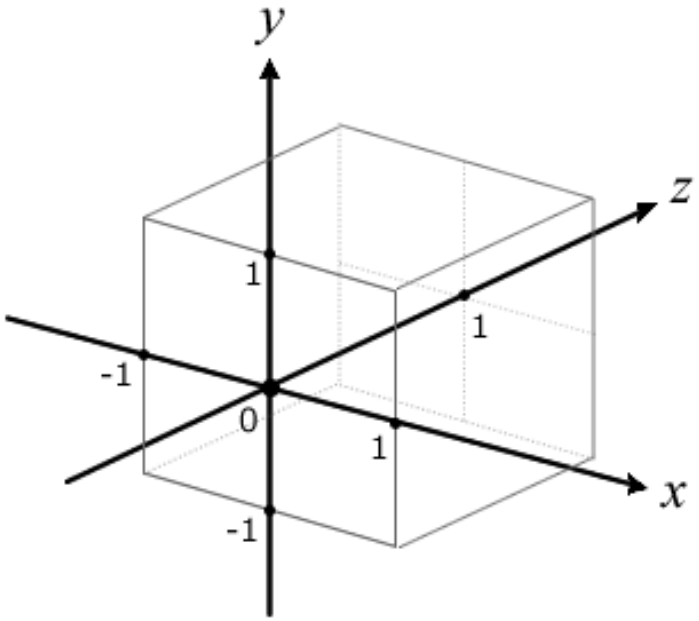
※Stage3Dは頂点座標と頂点インデックスでグラフィック形状を指定する

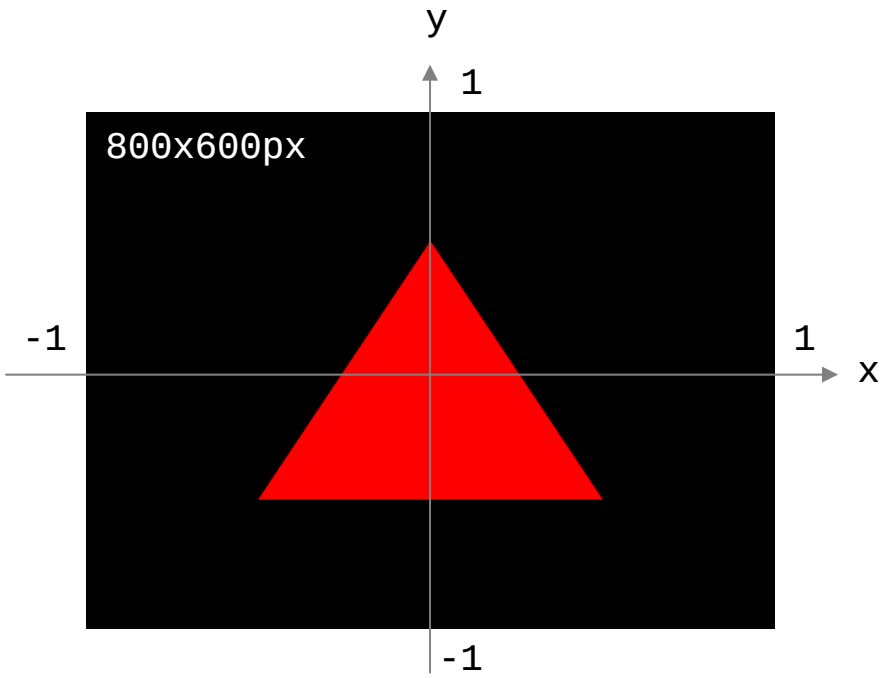
→グラフィックはポリゴンで構成する

Stage3Dの座標系

・Zが奥向きの左手座標系

・描画されるのは、-1≦x≦1, -1≦y≦1, 0≦z≦1 の範囲





プログラムの概要(3)

```
var vertexPrg:String   = "mov op, va0";
var fragmentPrg:String = "mov oc, fc0";
var program:Program3D = ctx.createProgram();
var assembler:AGALMiniAssembler = new AGALMiniAssembler();
program.upload(
    assembler.assemble(Context3DProgramType.VERTEX, vertexPrg),
    assembler.assemble(Context3DProgramType.FRAGMENT, fragmentPrg)
);
ctx.setProgram(program);
```

シェーダプログラム(AGAL)の準備

プログラムの概要(4)

```
ctx.clear();
ctx.drawTriangles(indexBuffer);
ctx.present();
```

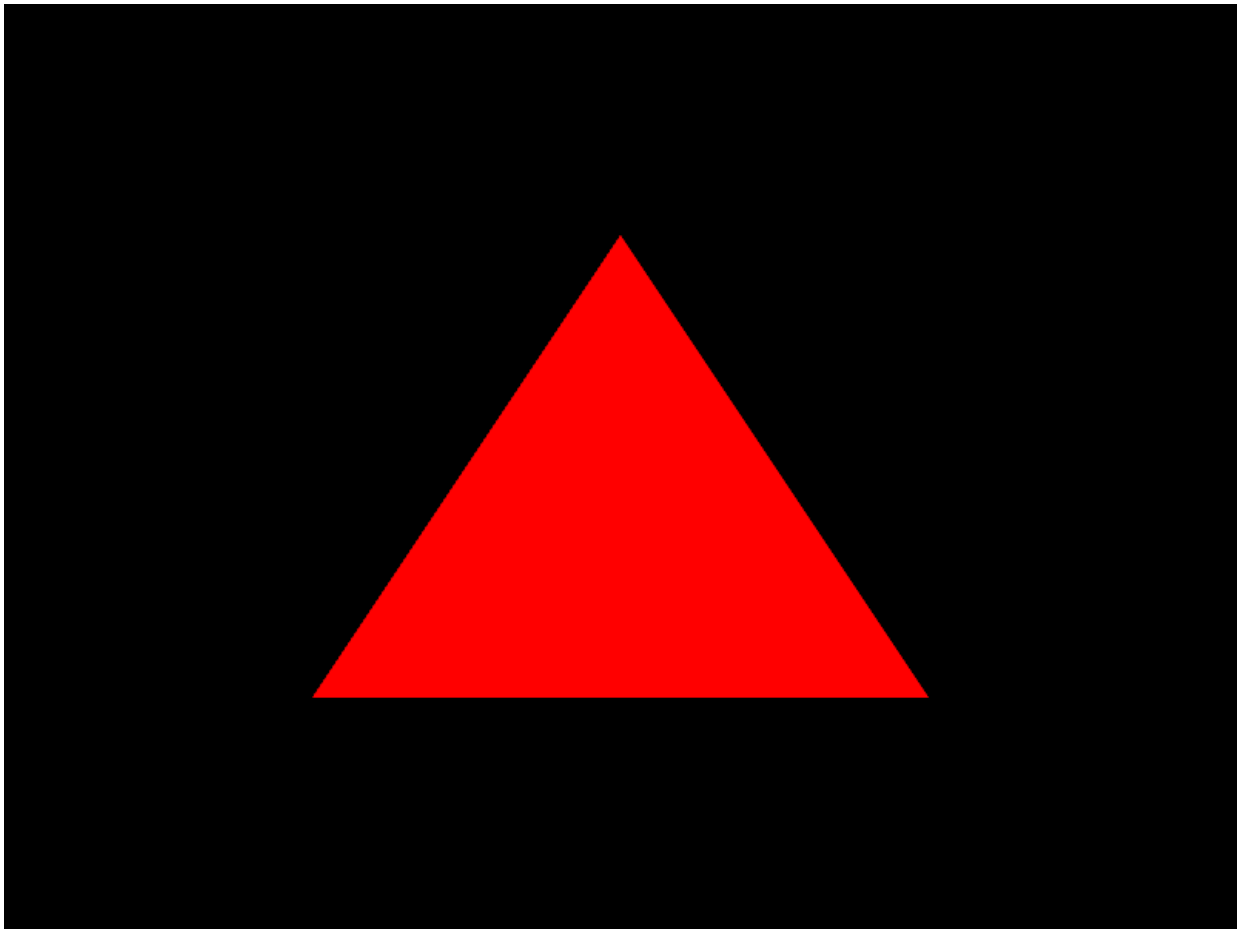
やっと描画

・クリア（clear）して

・バッファに描画して（drawTriangles)

・表示（present)

実行すると赤い三角形が表示される…



Graphicsクラスの場合：例1

```
graphics.beginFill(0xff0000);
graphics.moveTo(200, 450);
graphics.lineTo(400, 150);
graphics.lineTo(600, 450);
graphics.endFill();
```

これだけ…

Graphicsクラスの場合：例2

```
var vertices:Vector.<Number> = new <Number>[
  200, 450,  400, 150,  600, 450
];
var indices:Vector.<int> = new <int>[0,1,2];

graphics.beginFill(0xff0000);
graphics.drawTriangles(vertices, indices);
graphics.endFill();
```

drawTrianglesによる描画はStage3Dのプログラムと相似点がある

・頂点座標、インデックスを準備する

・drawTriangleメソッドで描画

Stage3DとGPU

Stage3Dの描画プログラムは、どうして冗長になるのか？

・Stage3DはGPUを利用するため
・GPUを制御するための手続きが必要になる

また、

・ライブラリ、ゲームエンジン(UnityやらUnrealやら）での利用を想定
・APIが汎用的（低レベル）に設計されとる

＊ *GPU（Graphics Processing Unit）*

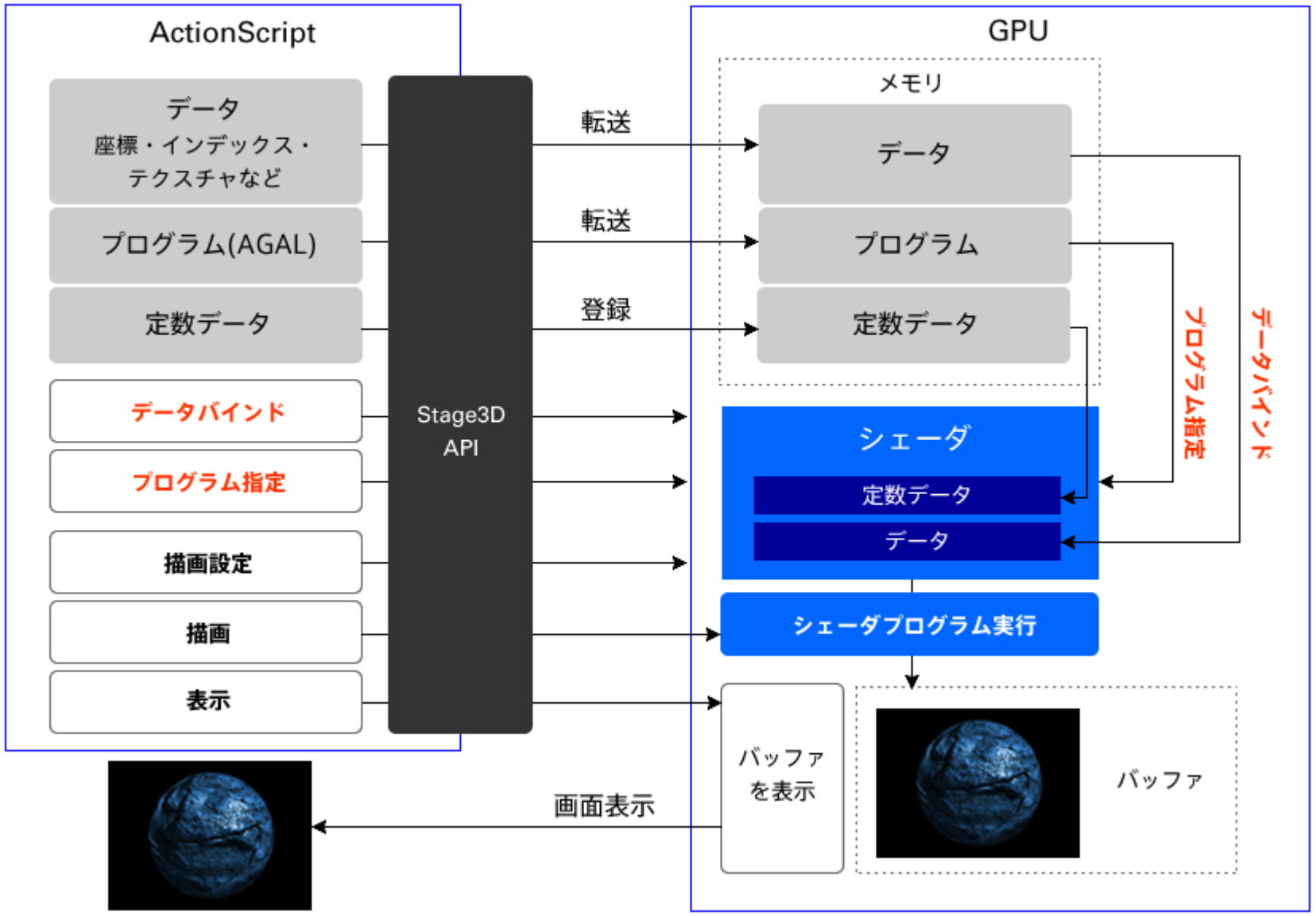
画像データ処理に特化したハードウェア。グラフィックをポリゴン処理やピクセル処理に定式化して高速に演算する。この用途に特化したもの。高速アクセスする為の独自のメモリも持っている。

Stage3D処理の概要

・GPUのメモリにデータをつくる
・GPUにシェーダプログラム(AGAL)をつくる
・GPUの描画設定をする
・GPUの描画に使うプログラムとデータを指定する
・GPUに描画させる→シェーダプログラム(AGAL)
・GPUがこしらえたグラフィックを表示する

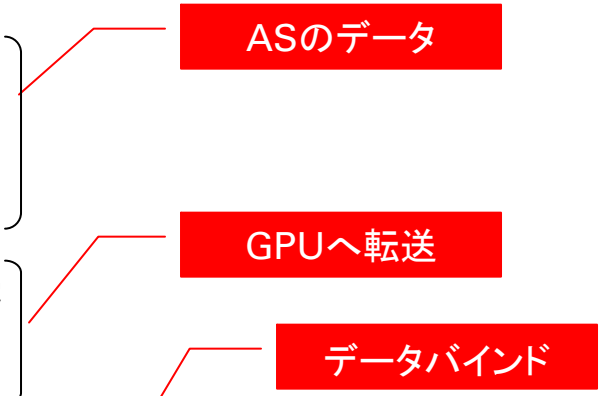
シェーダプログラムとは？

・GPUのデータを処理してグラフィックを生成するプログラム
・Stage3Dのシェーダプログラム言語はAGAL


ActionScript
データ
座標・インデックス・
テクスチャなど
プログラム(AGAL)
定数データ
データバインド
プログラム指定
描画設定
描画
表示
Stage3D
API
転送
転送
登録
GPU
メモリ
データ
プログラム
定数データ
プログラム指定
データバインド
シェーダ
定数データ
データ
シェーダプログラム実行
バッファ
を表示
バッファ
画面表示

頂点座標のデータ

```
var vertices:Vector.<Number> = new <Number>[
    -0.5,-0.5,0.0, 0.0,0.5,0.0, 0.5,-0.5,0.0
];
var vb:VertexBuffer3D = ctx.createVertexBuffer(3, 3);
vb.uploadFromVector(vertices, 0, 3);
ctx.setVertexBufferAt(0, vb, 0, Context3DVertexBufferFormat.FLOAT_3);
```



・頂点座標(ASのデータ)をVector.<Number>で用意

・GPUのデータ領域を作成(VertexBuffer3D)し、データを転送(アップロード)

・シェーダプログラムにデータを割り当てる

※GPUにデータを転送しただけでは、単にGPUのメモリにデータがあるだけ
　データバインドして、シェーダプログラムで使えるように指定する必要がある。

インデックスデータ

```
var indices:Vector.<uint> = new <uint>[0, 1, 2];
var indexBuffer:IndexBuffer3D = ctx.createIndexBuffer(3);
indexBuffer.uploadFromVector(indices, 0, 3);
```

ASのデータ

GPUへ転送

・インデックス(ASのデータ)をVector.<uint>で用意

・GPUのデータ領域を作成(IndexBuffer3D)し、データを転送(アップロード)

※IndexBuffer3DはdrawTriangles()で描画対象を指定する引数になる

定数データ

GPUの定数データ登録

```
ctx.setProgramConstantsFromVector(Context3DProgramType.FRAGMENT, 0,
                                  new <Number>[1,0,0,1]);
```

ASのデータ

・定数をGPUに登録する

※頂点のような大きなデータ(可変量のデータ)は、メモリへの転送とデータ指定が別になるが、定数は登録できる数が決まっているデータで、登録とデータ指定を同時にする

シェーダプログラム

```
var vertexPrg:String   = "mov op, va0";
var fragmentPrg:String = "mov oc, fc0";
var program:Program3D = ctx.createProgram();
var assembler:AGALMiniAssembler = new AGALMiniAssembler();
program.upload(
    assembler.assemble(Context3DProgramType.VERTEX, vertexPrg),
    assembler.assemble(Context3DProgramType.FRAGMENT, fragmentPrg)
);
ctx.setProgram(program);
```

ASのデータ(AGAL)

GPUへ転送

プログラムの指定

シェーダプログラムも他のデータと同様に、ASでデータ(AGAL)を準備し、GPUへ転送。

実行するプログラムの指定も行う。

※シェーダプログラムはGPUで動作する、GPUのデータを処理してグラフィックを生成するプログラム。

描画

```
ctx.clear();
ctx.drawTriangles(indexBuffer);
ctx.present();
```

GPUバッファのクリア

GPUに転送したインデックスで描画

GPUのバッファを表示

drawTrianglesでシェーダプログラムが動作する

シェーダプログラムはどのような処理をするのか？

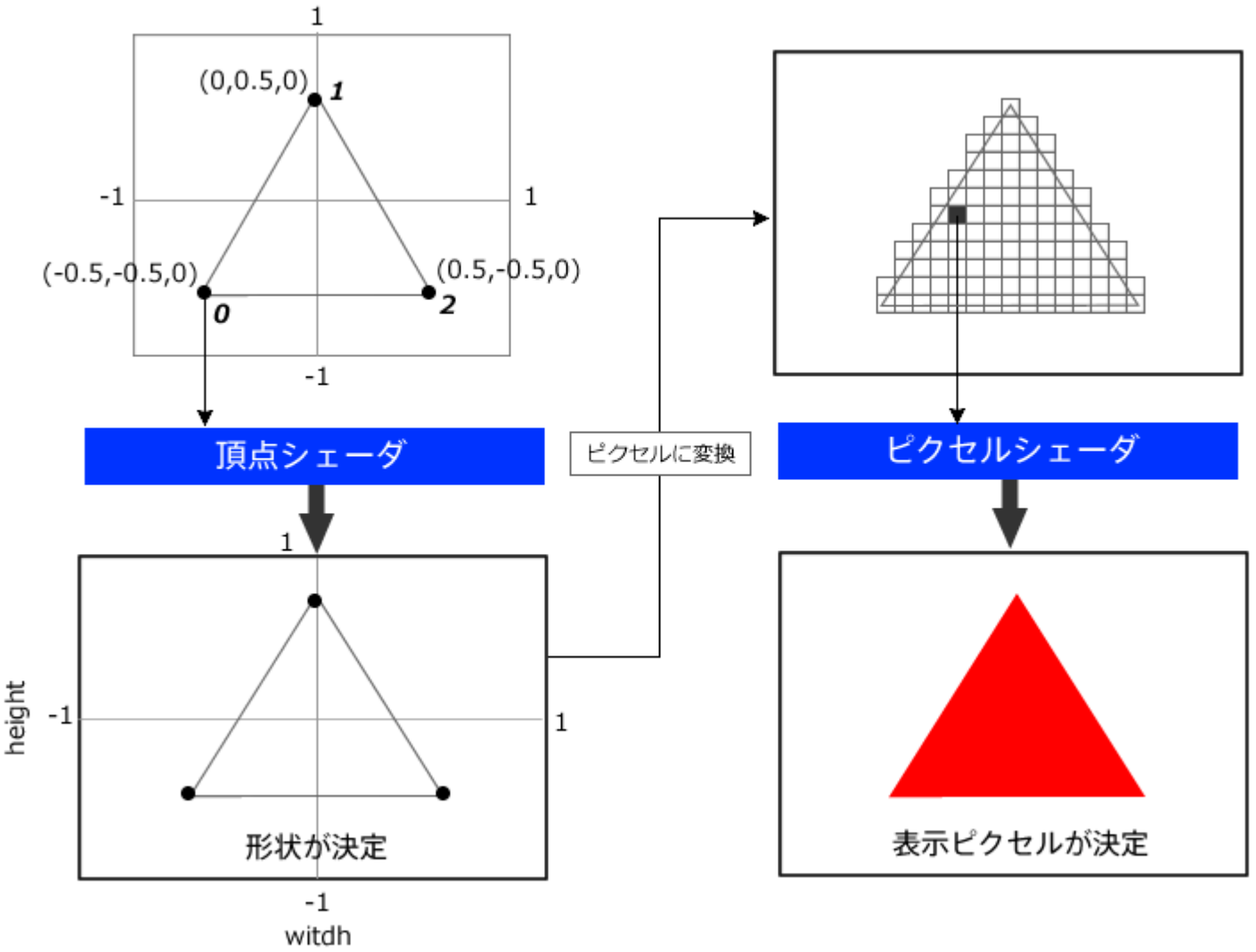
2つシェーダプログラムで処理を行う。

頂点シェーダ(Vertex Shader)

各頂点を処理。描画する位置を決める

ピクセルシェーダ(Fragment Shader)

ポリゴン内部の各ピクセルを処理。描画色を決める。


1
(0,0.5,0)
1
-1
1
(-0.5,-0.5,0)
(0.5,-0.5,0)
0
2
-1
頂点シェーダ
ピクセルに変換
ピクセルシェーダ
1
height
-1
1
形状が決定
表示ピクセルが決定
-1
witdh

Stage3D関連クラス

flash.display

Stage3D

flash.display3D, flash.display3D.textures

メイン

Staqge3Dの核となるAPI

Context3D

データ系クラス

GPUメモリのデータを表す
upload系、dispose のメソッドがある

Program3D

VertexBuffer3D

IndexBuffer3D

Texture

CubeTexture

定数クラス

単に静的プロパティを持つクラス

Context3DRenderMode

Context3DVertexBufferFormat

Context3DTextureFormat

Context3DProgramType

Context3DClearMask

Context3DTriangleFace

Context3DCompareMode

Context3DStencilAction

Context3DBlendFactor

Stage3Dの注意点

・Stage3DクラスはDisplayObjectではない

→addChild()するものではない
→IBitmapDrawableでもない

・Event.CONTEXT3D_CREATEイベントの発行タイミング

⇒demo

AGALの概要

AGALって何？

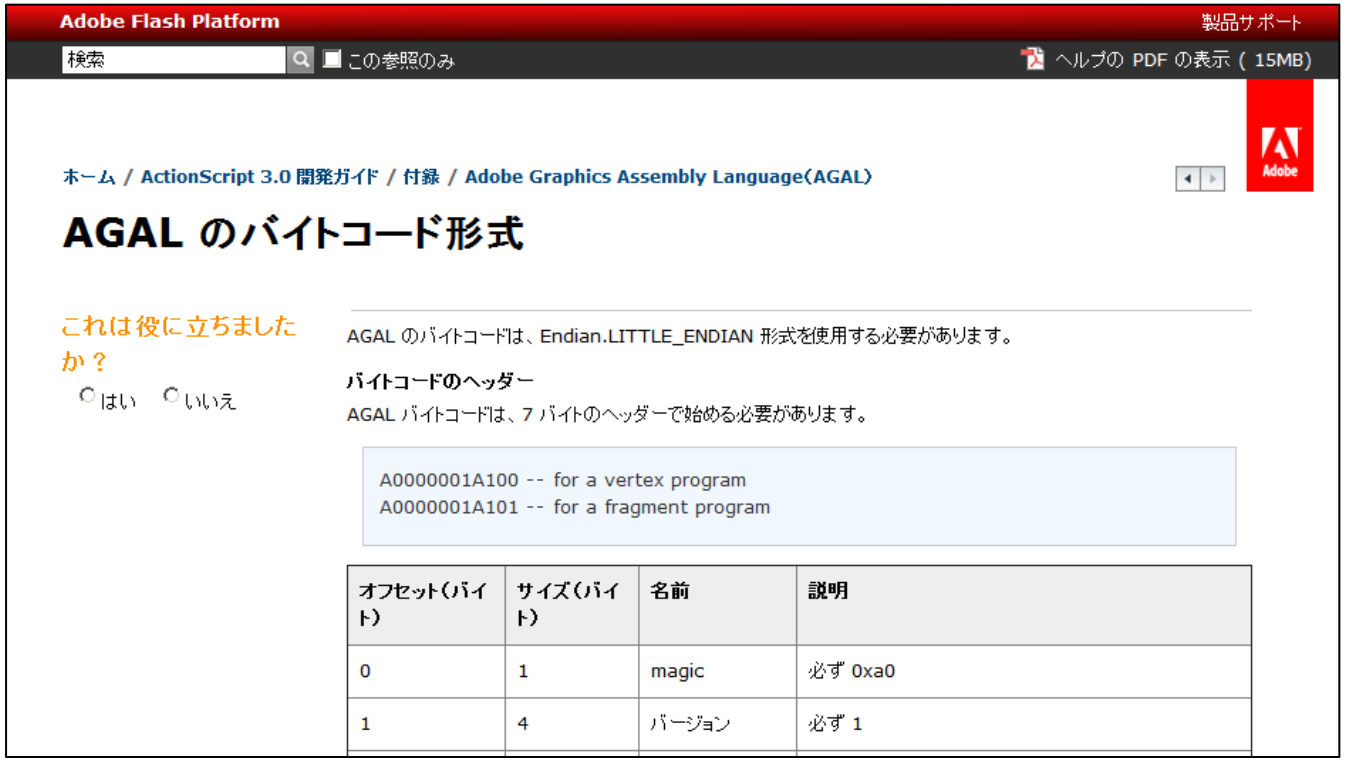
・Stage3Dのシェーダプログラム言語
・アセンブリ言語（Adobe Graphics Assembly Language）

＊ *アセンブリ言語*
アセンブリ言語というのは機械語を単に、単語に置き換えたもの

公式情報

・AGAL のバイトコード形式（adobeのドキュメント）

http://help.adobe.com/ja_JP/as3/dev/WSd6a006f2eb1dc31e-310b95831324724ec56-8000.html



このページにあるのはAGALのバイトコードの仕様に関するもの。

ビルトインクラスにこれを実装したものはない

AGALMiniAssembler

・GPUに転送する(upload)できるのはAGALをバイトコードにしたもの

・AGALプログラム文字列をバイトコード仕様に変換するものが必要

→com.adobe.utils.AGALMiniAssemblerクラス

どのライブラリにも含まれている

```
var vertexPrg:String   = "mov op, va0";
var fragmentPrg:String = "mov oc, fc0";
var program:Program3D = ctx.createProgram();
var assembler:AGALMiniAssembler = new AGALMiniAssembler();
program.upload(
    assembler.assemble(Context3DProgramType.VERTEX, vertexPrg),
    assembler.assemble(Context3DProgramType.FRAGMENT, fragmentPrg)
);
ctx.setProgram(program);
```

AGALについて前知識

・AGALは1行に1つの処理(命令)を記述

加算なども１つの処理。加算、減算などを１行にまとめて書けない

・制御文はない

ifやforなどはない。ひたすら処理をリニアに行う

・リテラルはない

数値などを直接書くことはできない

AGALの書式

AGALはオペコードとレジスタで記述する

```
add vt0, va1, vc1
```

オペコード ディスティネーションレジスタ, ソースレジスタ, ソースレジスタ

・オペコード：処理内容

・ディスティネーション：処理結果を格納する

・ソース：処理対象となるデータ

※レジスタの数はオペコードで異なる

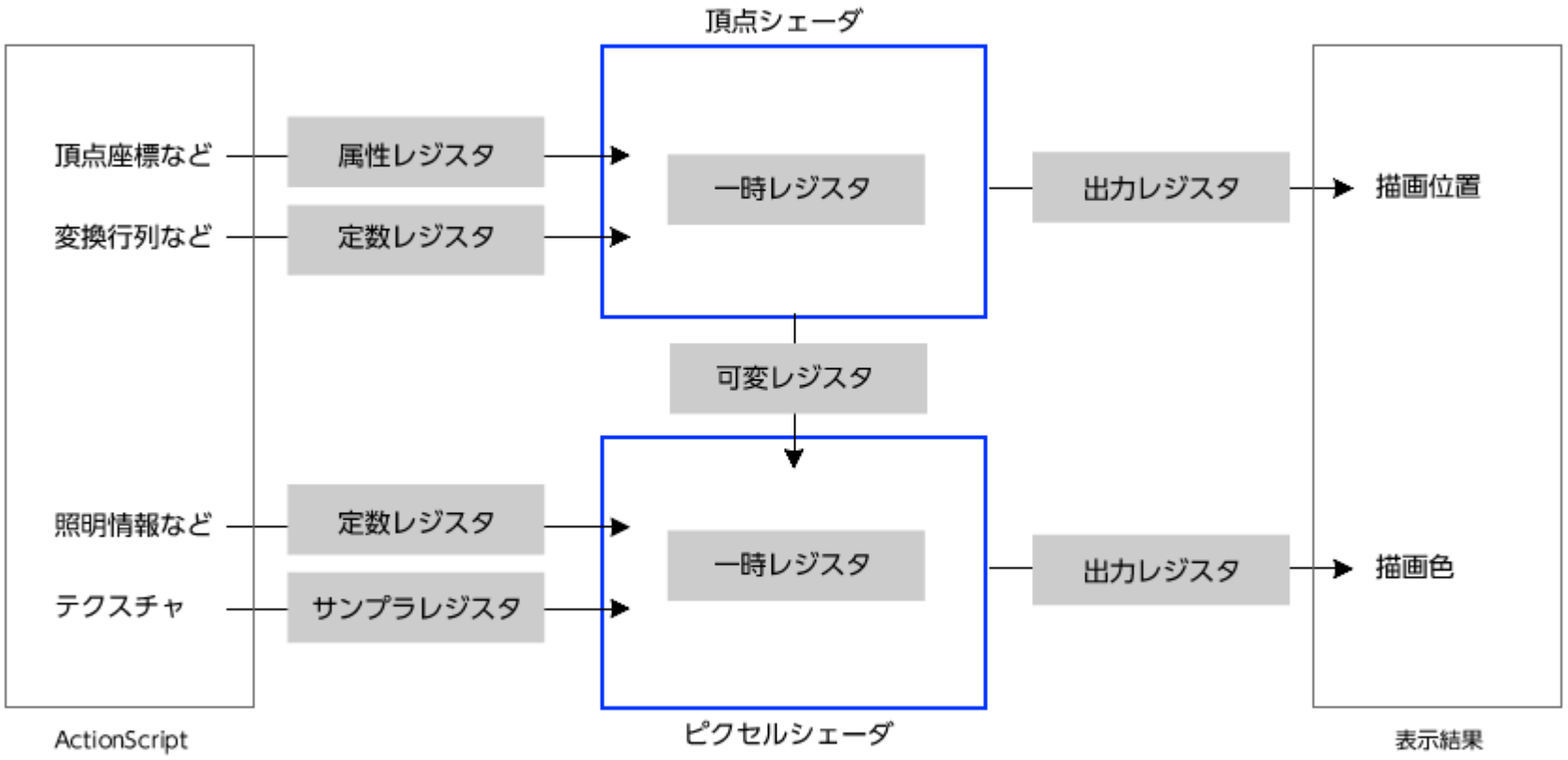
レジスタ

レジスタ

レジスタはAGALの変数的なもので6種類ある。

・属性レジスタ(Attribute)
・定数レジスタ(Constant)
・サンプラレジスタ(Sampler)
・一時レジスタ(Temporary)
・可変レジスタ(Varying)
・出力レジスタ(Output)

レジスタ

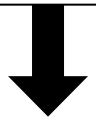

頂点シェーダ
頂点座標など
変換行列など
属性レジスタ
定数レジスタ
一時レジスタ
出力レジスタ
描画位置
可変レジスタ
照明情報など
テクスチャ
定数レジスタ
サンプラレジスタ
一時レジスタ
出力レジスタ
描画色
ActionScript
ピクセルシェーダ
表示結果

レジスタの成分

レジスタは4つの数値成分(x, y, z, w)がある。

基本的に、各成分で処理される。

```
add vt0, va1, vc1
```



```
vt0.x = va1.x + vc1.x
vt0.y = va1.y + vc1.y
vt0.z = va1.z + vc1.z
vt0.w = va1.w + vc1.w
```

レジスタの注意点

・レジスタの記述

変数のように自由に名前を定義できない。

レジスタの種類で決まった接頭詞と識別番号で記述。

・レジスタの数

プログラムで利用できるレジスタの数には上限がある。

・読み取り／書き込み

種類によっての読み取り・書き込みが可能が異なる。

属性レジスタ(Attribute)

シェーダ	数	接頭詞	記述	読取／書込
頂点シェーダ	8	va	va0 ～ va7	○／×

・頂点情報を表すレジスタ（座標、UV、法線ベクトルなど）

⇒3Dオブジェクトのデータ

・IndexBuffer3Dでデータを作成

・setVertexBufferAt()でシェーダにデータを割り当てる

⇒demo

属性レジスタの作成と割り当て

```
var vertices:Vector.<Number> = new <Number>[
 -0.5, -0.5, 0.0,  1.0, 0.0, 0.0, //座標(x,y,z) 色(r,g,b)
  0.0,  0.5, 0.0,  0.0, 1.0, 0.0,
  0.5, -0.5, 0.0,  0.0, 0.0, 1.0
];
//頂点数と、1頂点あたりの数値数でデータ領域作成
var vb:VertexBuffer3D = ctx.createVertexBuffer(3, 6);
//アップロード
vb.uploadFromVector(vertices, 0, 3);

//データの割り当て
ctx.setVertexBufferAt(0, vb, 0, Context3DVertexBufferFormat.FLOAT_3);//va0
ctx.setVertexBufferAt(1, vb, 3, Context3DVertexBufferFormat.FLOAT_3);//va1
```

頂点数

頂点の数値の数

属性の識別番号

属性の開始位置

属性のフォーマット

・Vectorの他、ByteArrayのデータをアップロードするAPIもある

・属性を解除する場合はnullを指定。

```
setVertexBufferAt( 0, null );
```

※シェーダで使わない属性を指定したままだと動かない

定数レジスタ(Constant)

シェーダ	数	接頭詞	記述	読取／書込
頂点シェーダ	128	vc	vc0 ～ vc127	○／×
ピクセルシェーダ	28	fc	fc0 ～ fc27	○／×

・シェーダにあらかじめ登録できる数値のレジスタ

・属性以外の数値は定数レジスタで指定

・通常リテラルで記述するような数値はこれで

定数レジスタの作成

```
//データの準備
Var c:Vector.<Number> = new <Number>[1,0,0,1];
//シェーダ、識別番号を指定してアップロード
ctx.setProgramConstantsFromVector(Context3DProgramType.FRAGMENT, 0, c);//fc0
```

・数値は４つセットで登録する
・Vectorの他、Matrix3D、ByteArrayのデータを登録するAPIもある
・Matrix3Dをアップロードした場合、数値が16個なので４つのレジスタになる
　※頂点シェーダに「０」で登録した場合、vc0～vc3が行列のレジスタ
・AGALで定数同士の計算はエラーになるので注意

サンプラレジスタ(Sampler)

シェーダ	数	接頭詞	記述	読取／書込
ピクセルシェーダ	8	fs	fs0 ～ fs8	○／×

・テクスチャサンプリング用のレジスタ

・Texure、CubeTexureでデータを作成

・setTexureAt()でシェーダにデータを割り当てる

サンプラレジスタの作成と割り当て

```
//サイズを指定してデータ領域作成
var texture:Texture = ctx.createTexture(512, 512,
                                        Context3DTextureFormat.BGRA, false);
//アップロード
texture.uploadFromBitmapData(bitmapdata);
//データの割り当て
ctx.setTextureAt(0, texture);//fs0
```

識別番号

・レジスタを解除する場合はnullを指定。

```
setTextureAt( 0, null );
```

一時レジスタ(Temporary)

シェーダ	数	接頭詞	記述	読取／書込
頂点シェーダ	8	vt	vt0 ～ vt7	○／○
ピクセルシェーダ	8	ft	ft0 ～ ft7	○／○

・各シェーダのローカル変数のようなもの

・数が８つでやりくりをがんばる

・唯一、読み込み／書き込み両方が可能

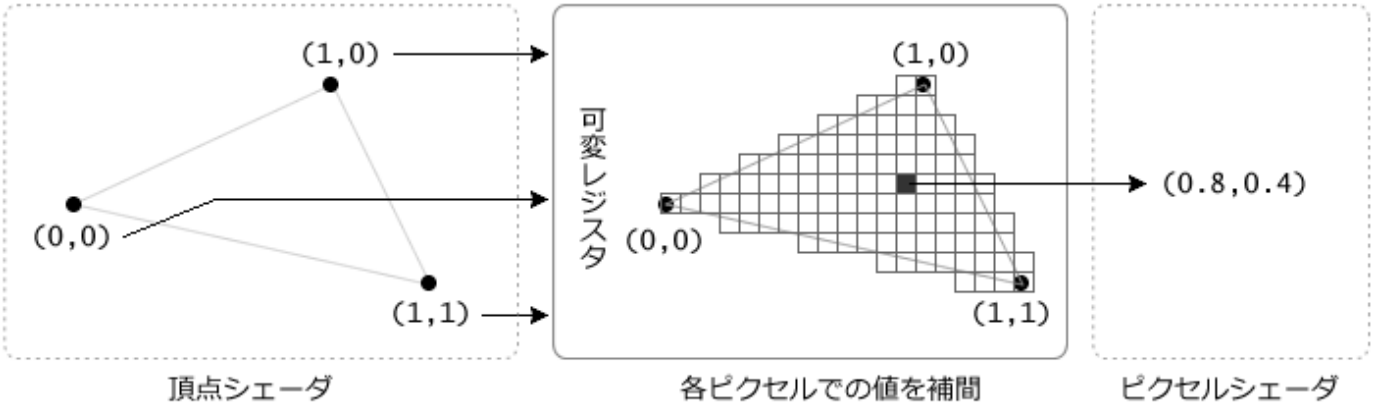
可変レジスタ(Varying)

シェーダ	数	接頭詞	記述	読取／書込
頂点シェーダ	8	v	v0 ～ v7	×／○
ピクセルシェーダ				○／×

・頂点シェーダからピクセルシェーダへデータを送るためのレジスタ

・ピクセルシェーダに送られる値は、各ピクセルで補間された値

⇒demo



出力(Output)レジスタ

シェーダ	数	記述		読取／書込
頂点シェーダ	1	op	*output position*	×／○
ピクセルシェーダ	1	oc	*output color*	×／○

・各シェーダの処理結果を格納するレジスタ

・頂点シェーダ⇒描画位置、ピクセルシェーダ⇒ピクセル描画色

頂点シェーダの出力について

頂点シェーダの出力レジスタ(op)に格納した値はw値で除算される

・opに書き込んだ値　(x, y, z, w)
・出力座標　(x/w, y/w, z/w)

⇒*demo*

一般的にwにz値を入れておいてプロジェクションの処理に使う

スウィズル演算子

・レジスタの各成分は「.」で指定できる

・複数指定、順番の入れ替えが可能

・ディスティネーションに記述した場合、処理する成分の指定となる

```
add     vt0 ,    va0.xxxx, vc0.wyxz
```

↓

```
        vt0.x  =  va0.x  +  vc0.w
        vt0.y  =  va0.x  +  vc0.y
        vt0.z  =  va0.x  +  vc0.x
        vt0.w  =  va0.x  +  vc0.z
```

```
add     vt0.xyz, va0 ,    vc0
```

↓

```
        vt0.x  =  va0.x  +  vc0.x
        vt0.y  =  va0.y  +  vc0.y
        vt0.z  =  va0.z  +  vc0.z
```

オペコード

オペコード

・プログラムの処理内容を示すもの

・関数的なもの

オペコードの種類

mov　代入

算術

add　加算
sub　減算
mul　乗算
div　除算

rcp　逆数
frc　少数
neg　符号反転

min　最小
max　最大
sqt　平方根
rsq　逆数平方根
pow　累乗
log　自然対数

exp　指数
abs　絶対値
sat　飽和

sin　正弦波
cos　余弦波

ベクトル

nrm　正規化
crs　外積
dp3　内積
dp4　内積

行列

m44
m33
m34

テクスチャサンプリング

tex

破棄

kil

条件

sge　以上
slt　未満

行列計算

行列変換の処理では第２ソースレジスタ(vc0)が連続した複数のレジスタとなる

```
m44 vt0, va1, vc4
```

$$\begin{bmatrix} vt0.x \\ vt0.y \\ vt0.z \\ vt0.w \end{bmatrix} = \begin{bmatrix} vc4.x & vc4.y & vc4.z & vc4.w \\ vc5.x & vc5.y & vc5.z & vc5.w \\ vc6.x & vc6.y & vc6.z & vc6.w \\ vc7.x & vc7.y & vc7.z & vc7.w \end{bmatrix} \begin{bmatrix} va1.x \\ va1.y \\ va1.z \\ va1.w \end{bmatrix}$$

テクスチャサンプリング（ピクセルシェーダ用）

サンプラ(fs0:テクスチャ)のテクスチャ座標(v0:UVなど)にある色値を取得

```
tex ft0, v0, fs0<...オプション>
```

サンプリングオプション(赤字は規定値)

テクスチャタイプ設定
2d（Textureクラス）または、cube（CubeTextureクラス）

繰り返し設定
テクスチャをタイル状に繰り返すかどうか。clamp, wrap(repeat)

フィルタ
色値を取得するときの方法。nearest, linear

ミップマップ
ミップマップから色値を取得するときの方法。nomip, mipnearest, miplinear

破棄（ピクセルシェーダ用）

ソースが０未満のとき、ピクセル描画をキャンセル

```
kil ft0.x
```

⇒demo

条件

条件処理(if文)の代用

```
sge vt0, va0, vc0        //vt0 = (va0 >= vc0 )? 1 : 0;
```

```
slt vt0, va0, vc0        //vt0 = (va0 < vc0  )? 1 : 0;
```

左のif文をAGALで記述すると右のようになる

```
if( va0.x >= vc0.x ){
  vt2.x = va0.x;
}else{
  vt2.x = vc0.x;
}
```

```
sge vt0.x, va0.x, vc0.x
slt vt1.x, va0.x, vc0.x
mul vt0.x, vt0.x, va0.x
mul vt1.x, vt1.x, vc0.x
add vt2.x, vt0.x, vt1.x
```

デモなど

Stage3DとAGALで3Dプログラム

APIやAGAL以前に、3Dプログラムの知識が必要

・座標変換（オブジェクトの操作・カメラ・プロジェクションなど）

・ライティング

・テクスチャマッピング

・モデルデータ

※デモでどのようなプログラムになるか紹介します。

2D

Stage3Dで2Dプログラム

Stage3Dは、2Dのほうが3Dより利用しやすい。

・3Dプログラムのような知識は必要ない

・モジュール化しやすい（四角形・テクスチャ描画程度）

・後はBitmapData.drawと同じような感覚で

・GPUの恩恵がある

2Dライブラリ Starling

Starlingで使われているAGALは３種類しかない

・starling.display.QuadBatch#registerPrograms の部分

・属性レジスタのIDには独自ルールがある

```
va0 -> position
va1 -> color
va2 -> texCoords
vc0 -> alpha
vc1 -> Matrix
fs0 -> texture
```

※違ってたらすいません

StarlingのAGAL

Quad(四角形)の描画

VertexShader

```
m44 op, va0, vc1  // 4x4 matrix transform to output clipspace
mov oc, v0        // multiply alpha (vc0) with color (va1)
```

FragmentShader

```
mov oc, v0        // output color
```

StarlingのAGAL

Image(テクスチャ)の描画

VertexShader

```
m44 op, va0, vc1  // 4x4 matrix transform to output clipspace
mov v1, va2       // pass texture coordinates to fragment program
```

FragmentShader

```
tex  oc,  v1, fs0 <???> // sample texture 0
```

※<???>にはサンプリングオプションが一通りのパターン用意される

StarlingのAGAL

Image(テクスチャ)の描画：Tintあり

VertexShader

```
m44 op, va0, vc1 // 4x4 matrix transform to output clipspace
mul v0, va1, vc0 // multiply alpha (vc0) with color (va1) : tint
mov v1, va2      // pass texture coordinates to fragment program
```

FragmentShader

```
tex ft1,  v1, fs0 <???> // sample texture 0
mul  oc, ft1,  v0       // multiply color with texel color
```

※<???>にはサンプリングオプションが一通りのパターン用意される

おまけ

おまけ

・AGALで線を描く

・AGALでグリッチ

まとめ

AGALを調べる

・公式の「AGALバイトコード」を読む
⇒AGALの基本がわかる

・AGALMiniAssemblerのプログラムを読む
⇒AGALで何が書けるかわかる

・オンラインヘルプProgram3Dのエラー一覧
⇒公式にないルール、何ができないかがわかる。

AGALを書く心構え

・レジスタはIDで管理しないとだめ

記述から何のデータなのか分かりにくい。コメント必須

データ指定を変えるとAGAL書き直し。

ライブラリはある程度決めうちしてある。０番が座標など

・レジスタの数

一時レジスタが、とにかくやりくり。処理の順番に気をつかう

・試行錯誤はしにくい

あらかじめ処理内容は決まっていて、AGALに書き直すぐらいが調度いい

・中間言語が欲しくなる

・2Dがお手軽

ご清聴ありがとうございました。