

お手軽 フラグメントシェーダー

GLSL スクール3回目 プラスワン講義
佐々木正樹 (Masaki Sasaki)

本プラスワンの目的

フラグメントシェーダーでできることを再確認する

トラブルが発生した時の確認方法

エフェクトを考えてみる、つなげる

フラグメントシェーダーの課題を列挙

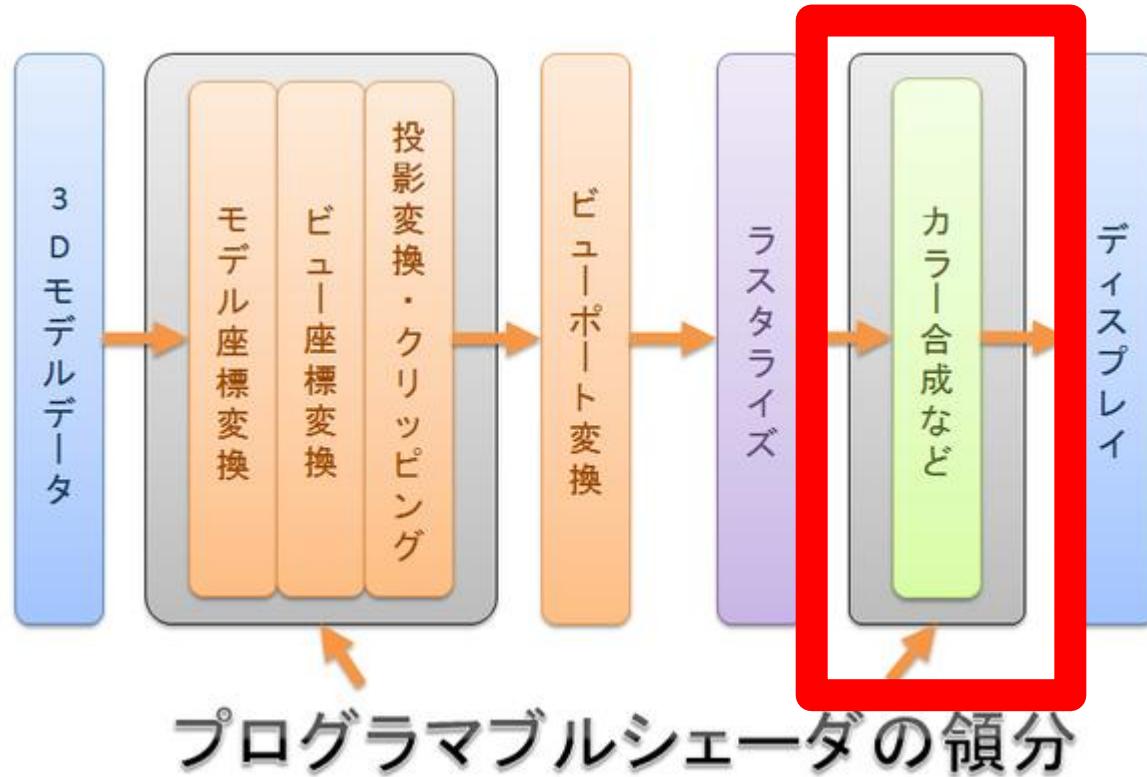
前提

- VertexShader側から入力される情報(varying/in)でフラグメントシェーダーでできることが左右されるのでなるべくプリミティブな解説にします。
- WebGL1.0/2.0などでもなるべく適用できるような解説をします
- OpenGLとDirectX11のGLSL/HLSLの比較は以下がよくまとまっています
(日本語。興味があれば)
<https://docs.microsoft.com/ja-jp/windows/uwp/gaming/glsl-to-hlsl-reference>

フラグメントシェーダーで
できることを再確認する

※ちょっとお堅いですけど再確認

再確認：フラグメントシェーダーは
グラフィックパイプラインの赤いところを担当する！



GLSLのフラグメントシェーダーに 備わっている組み込み変数を ある程度再確認します

OpenGL ES Shading Language 1.0 Quick Reference Card - Page 4

Built-In Inputs, Outputs, and Constants [7]

Shader programs use Special Variables to communicate with fixed-function parts of the pipeline. Output Special Variables may be read back after writing. Input Special Variables are read-only. All Special Variables have global scope.

Vertex Shader Special Variables [7.1]

Outputs:

Variable	Description	Units or coordinate system
highp vec4	gl_Position;	transformed vertex position
mediump float	gl_PointSize;	transformed point size (point rasterization only)

Fragment Shader Special Variables [7.2]

Fragment shaders may write to gl_FragColor or to one or more elements of gl_FragData[], but not both. The size of the gl_FragData array is given by the built-in constant gl_MaxDrawBuffers.

Inputs:

Variable	Description	Units or coordinate system
mediump vec4	gl_FragCoord;	fragment position within frame buffer
bool	gl_FrontFacing;	fragment belongs to a front-facing primitive
mediump vec2	gl_PointCoord;	fragment position within a point (point rasterization only)

Outputs:

Variable	Description	Units or coordinate system
mediump vec4	gl_FragColor;	fragment color
mediump vec4	gl_FragData[n];	fragment color for color attachment <i>n</i>

Built-In Constants With Minimum Values [7.4]

Built-In Constant	Minimum value
const mediump int gl_MaxVertexAttribs	8
const mediump int gl_MaxVertexUniformVectors	128
const mediump int gl_MaxVaryingVectors	8
const mediump int gl_MaxVertexTextureImageUnits	0
const mediump int gl_MaxCombinedTextureImageUnits	8
const mediump int gl_MaxTextureImageUnits	8
const mediump int gl_MaxFragmentUniformVectors	16
const mediump int gl_MaxDrawBuffers	1

Built-In Uniform State [7.5]

Specifies depth range in window coordinates. If an implementation does not support highp precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

```
struct gl_DepthRangeParameters {  
    highp float near; // n  
    highp float far; // f  
    highp float diff; // f - n  
};
```

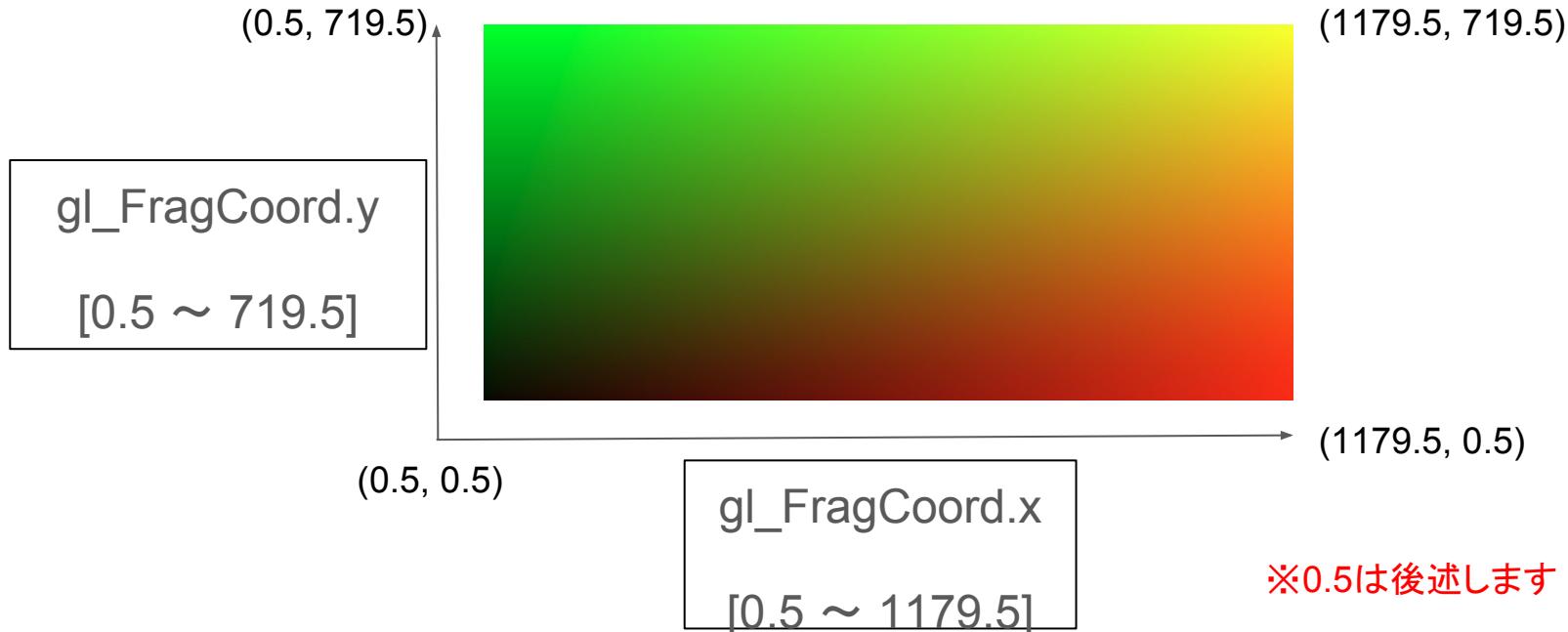
```
uniform gl_DepthRangeParameters gl_DepthRange;
```

gl_FragCoord.xy成分の確認

gl_FragCoord.xy : レンダリングするポリゴンのとあるピクセルの座標が入ってくる

https://www.khronos.org/registry/OpenGL-Refpages/es3.0/html/gl_FragCoord.xhtml

例) 1280x720の解像度に対して全部を「覆うポリゴン」を画面全体に出した場合

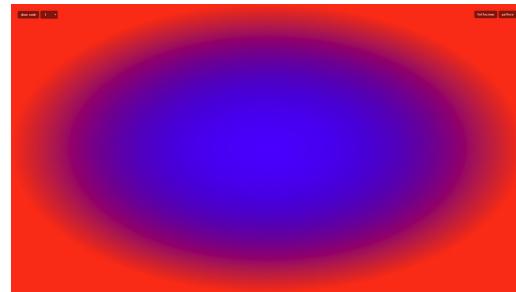


discard命令について確認

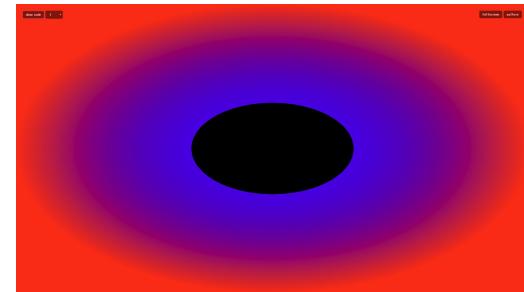
discardは、**実行されたら**、レンダリングしようとしているピクセルのバッファへの結果が**全部**破棄される。

例) 赤成分が0.1未満ならピクセルを破棄

```
void main() {  
    gl_FragColor = vColor;  
    if(gl_FragColor.r < 0.1) {  
        discard;  
    }  
}
```



元画像



discard結果

注意 : discardはちょっと古いモバイルの一部で不思議な動きをするときがありますので
使う場合はdiscard実行したらPixelの操作をしないように組むのを推奨
(具体例 : 固まって動かない。エラーじゃないのに動かないなど)

参考URL : <http://wlog.flatlib.jp/item/1550>

gl_FrontFacingの確認

bool gl_FrontFacing : レンダリングしようとしているポリゴンが「表」なら、**true**, それ以外は**false**

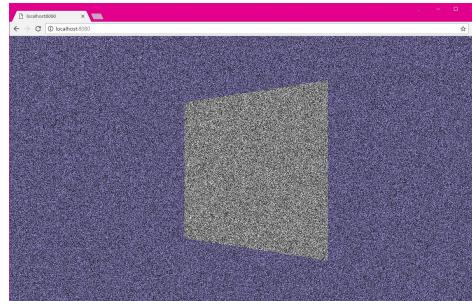
https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

サンプル : 013_Ex001

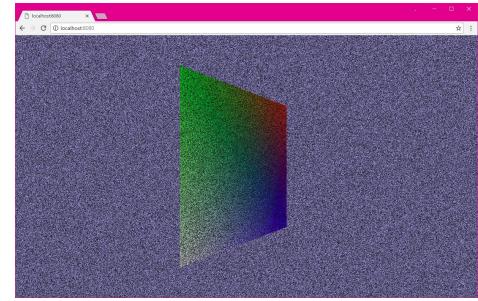
↓

```
precision medium float;  
varying vec4 vColor;  
void main (){  
    gl_FragColor = vColor;  
    if (gl_FrontFacing) { //ポリゴンが表なら  
        gl_FragColor = vec4(1.0); //白くレンダリング  
    }  
}
```

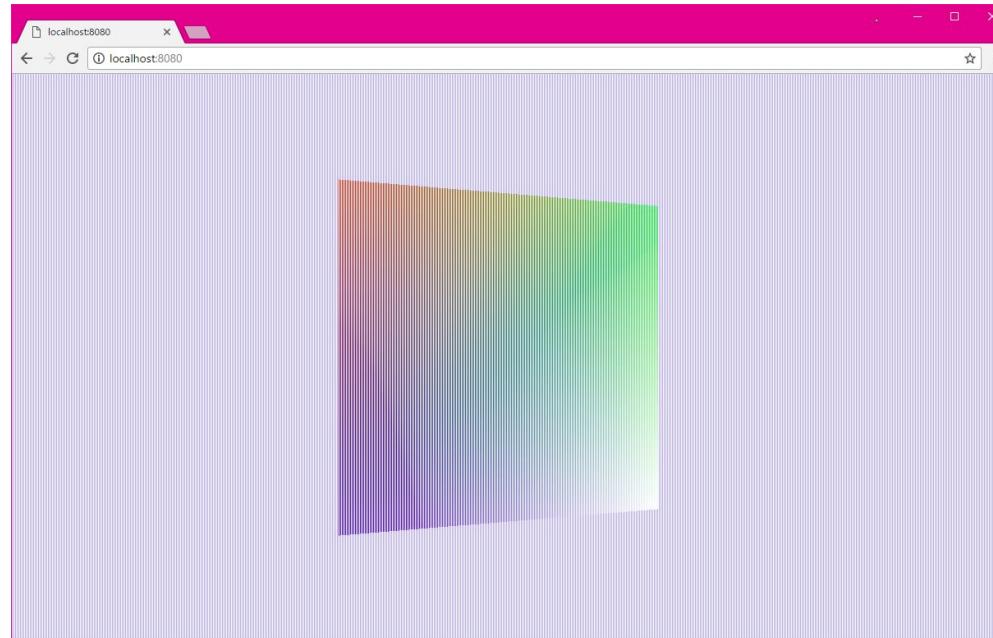
表(gl_FrontFacing == **true**)



裏(gl_FrontFacing == **false**)



エクササイズ：ポストエフェクトで奇数行を白くする
サンプル：013_Ex002



解答例イメージ

エクササイズ：ポストエフェクトで奇数行を白くする

サンプル：013_Ex002

[考え方の例]

- 1) mod (あまりを使う)
- 2) 2で割った余りがほしい
- 3) 0～Nの数字(number)をmod(number, 2.0) -> 0, 1, 0, 1, 0, 1....となる
- 4) 3)の結果 == 1.0を判定すれば、偶数、奇数が判定できるはず
- 5) 4)の結果をもとに、gl_FragColor = vec4(1.0);を入れればOKのはず

エクササイズ：ポストエフェクトで奇数行を白くする

サンプル：013_Ex002

```
void main() {
    // フレームバッファの描画結果をテクスチャから読み出す
    vec4 samplerColor = texture2D(texture, vTexCoord);
    gl_FragColor = samplerColor;
    bool is_odd = mod(gl_FragCoord.x, 2.0) == 1.0; //奇数ならtrue, それ以外ならfalse
    if(is_odd) {
        gl_FragColor = vec4(1.0); //白をレンダリングする
    }
}
```

エクササイズ：ポストエフェクトで奇数行を白くする

サンプル：013_Ex002

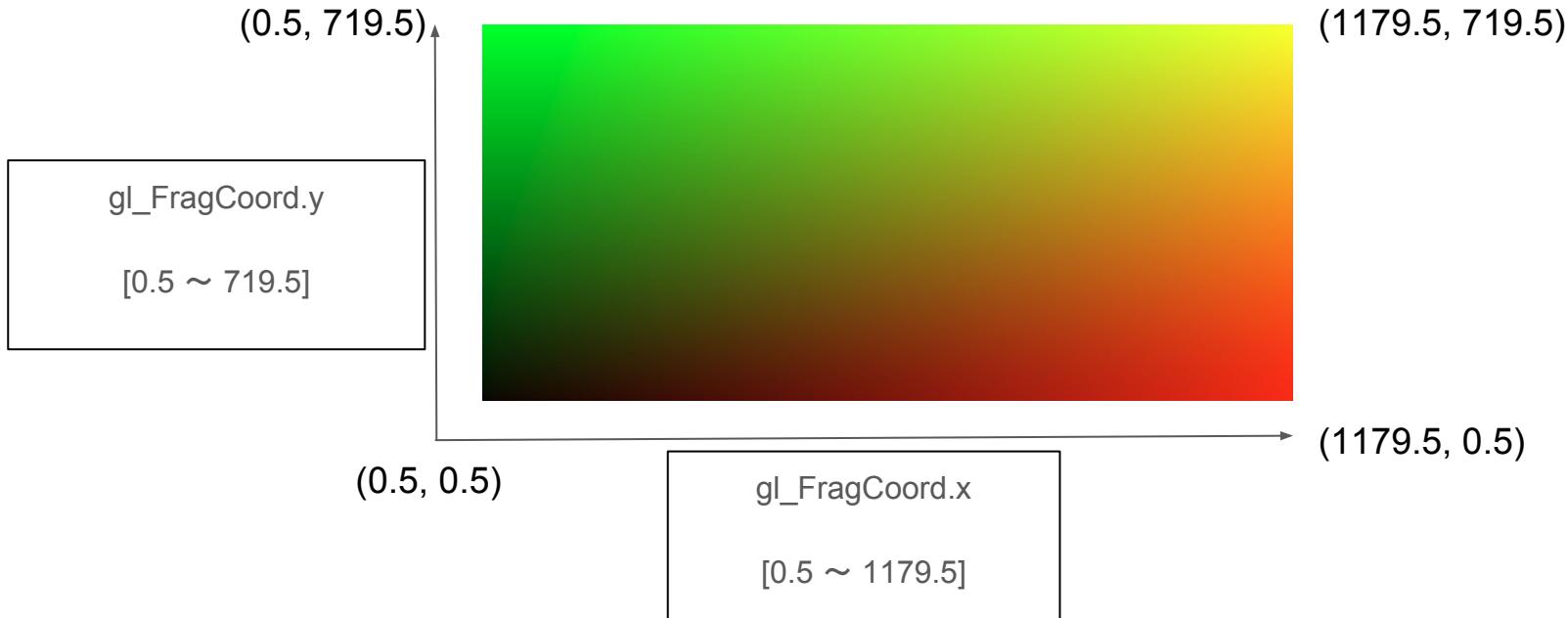
```
void main() {
    // フレームバッファの描画結果をテクスチャから読み出す
    vec4 samplerColor = texture(texture, vTexCoord);
    gl_FragColor = samplerColor;
    bool is_odd = mod(gl_FragCoord.y, 2) == 1; // 奇数ならtrue, それ以外ならfalse
    if(is_odd) {
        gl_FragColor = vec4(1.0);
    }
}
```

再掲：gl_FragCoord.xy成分の確認

gl_FragCoord.xy : レンダリングするポリゴンのとあるピクセルの座標が入ってくる

https://www.khronos.org/registry/OpenGL-Refpages/es3.0/html/gl_FragCoord.xhtml

例) 1280x720の解像度に対して全部を「覆うポリゴン」を画面全体に出した場合



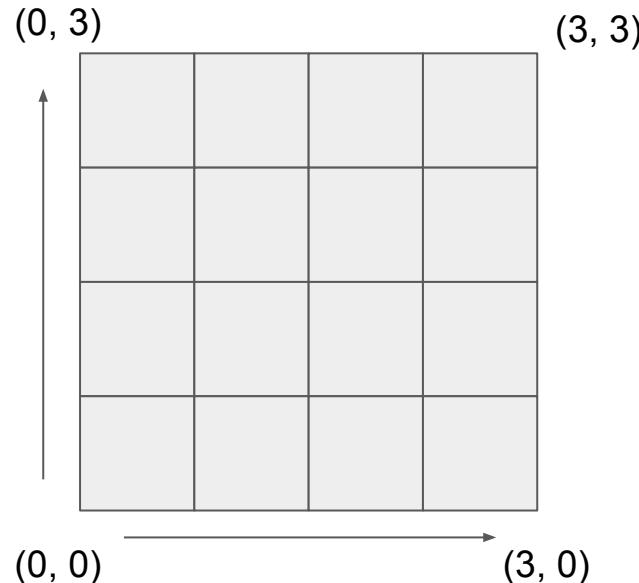
エクササイズ：ポストエフェクトで奇数行を白くする サンプル：013_Ex002

```
void main() {
    // フレームバッファの描画結果をテクスチャから読み出す
    vec4 samplerColor = texture2D(texture, vTexCoord);
    gl_FragColor = samplerColor;
    bool is_odd = mod(gl_FragCoord.x - 0.5, 2.0) == 1.0; //奇数ならtrue, それ以外ならfalse
    if(is_odd) {
        gl_FragColor = vec4(1.0);
    }
}
```

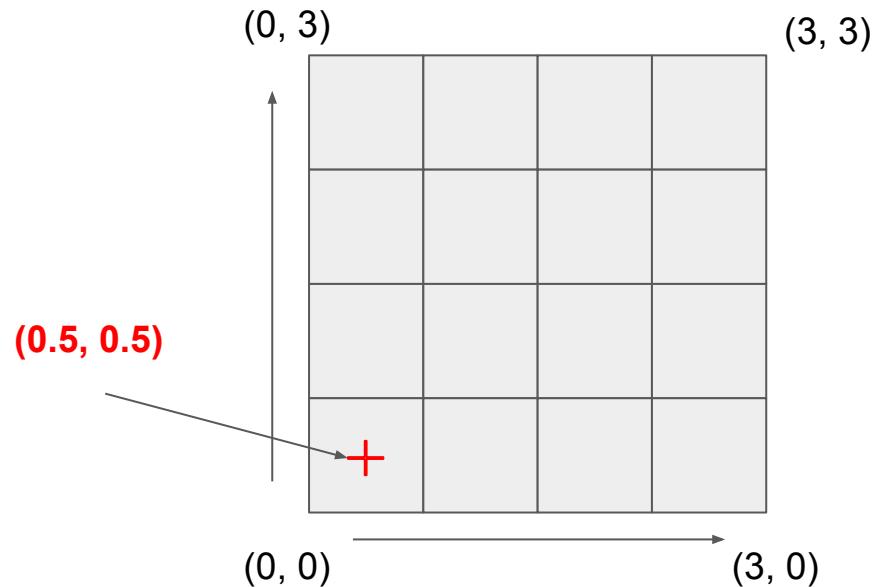
何故？？

エクササイズ：ポストエフェクトで奇数行を白くする
サンプル：013_Ex001

フレグメントシェーダーの世界では、`gl_FragCoord`は「ピクセルの中心」が入る



エクササイズ：ポストエフェクトで奇数行を白くする
サンプル：013_Ex001



エクササイズ：ポストエフェクトで奇数行を白くする サンプル：013_Ex001

[https://www.khronos.org/opengl/wiki/Built-in_Variable_\(GLSL\)](https://www.khronos.org/opengl/wiki/Built-in_Variable_(GLSL))

Fragment Shader Inputの項目を参照すると…

OpenGL window space is defined such that pixel centers are on half-integer boundaries. So the center of the lower-left pixel is (0.5, 0.5). Using `pixel_center_integer` adjust `gl_FragCoord` such that whole integer values represent pixel centers.

[ポイント]

最も左下が $x=0.5, y=0.5$ 。つまりピクセルのど真ん中

WebGLにはlayoutが無いので`pixel_center_integer`は使えない。

もしピクセル単位で`gl_FragCoord`を使って、フレグメントシェーダーを操作したい場合は 0.5を意識すること。

※例えばスクリーン空間いっぱいにドット絵を正確に表示したいなど

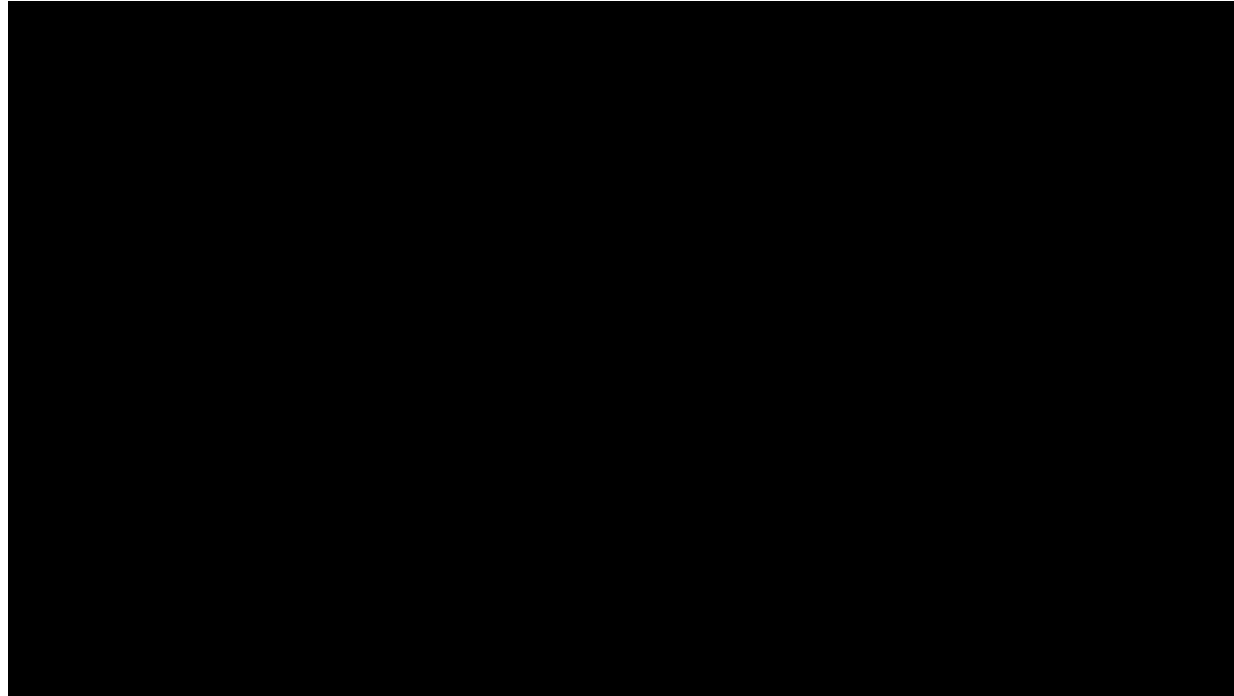
フラグメントシェーダーでできることを再確認する(まとめ)

- ・必須ではないけど、組み込み変数(Built in)について理解を深めるのが大事
(OpenGL4.5だともっといろいろ使えるものがある)
- ・組み込み変数は細かい仕様がある
- ・スクリーン座標で何かするときは細かい仕様が必ずあるので、確認すること
- ・有用なリファレンスはたいてい英語ですけど、翻訳が優秀なので見るとヒントがちゃんと書いてある(よく読む)

トラブルが発生した時の 確認方法

※自由に使えるためには後ろ盾がある状態で

トラブル1：絵が出ない！



破滅

トラブル1：絵が出ない！

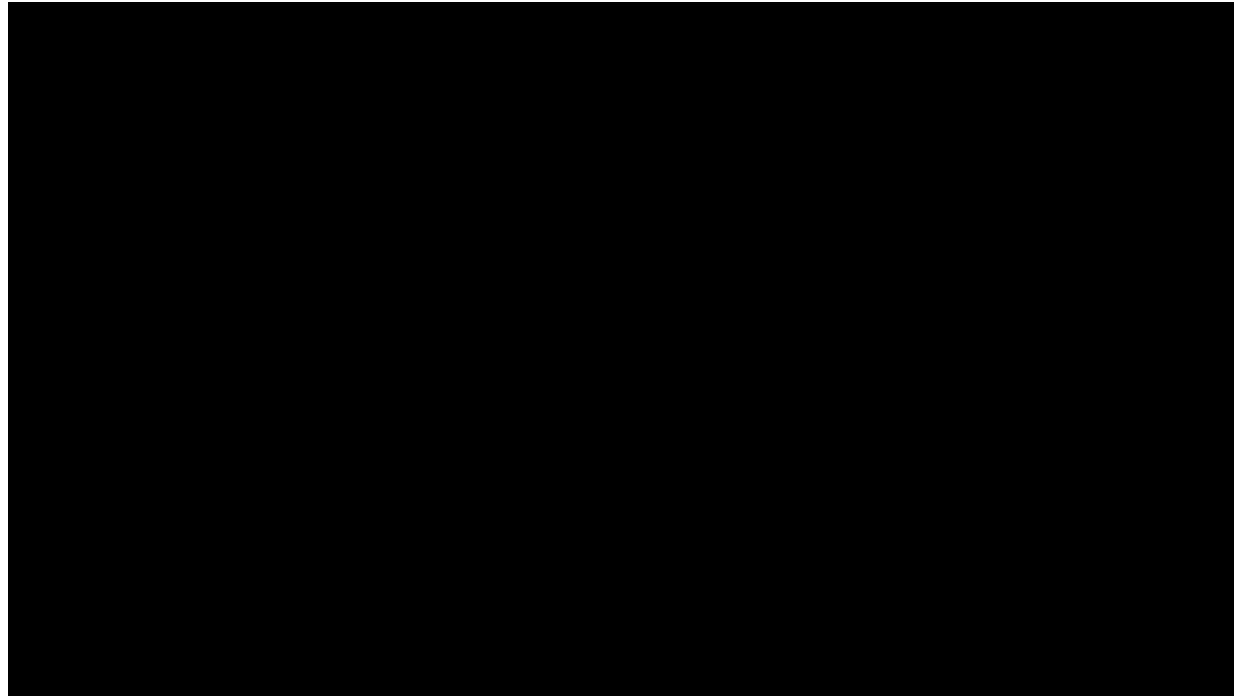
まずはフラグメントシェーダーの mainに
速攻で以下を書いてしまう(背景色がピンクなら、色を変えてみる)

```
void main() {  
    gl_FragColor = vec4(1, 0, 1, 1);  
}
```



- チェック1：画面上のどこかにピンク色が出てたらポリゴンは描画されている。
- チェック2：ピンクが描画される領域が期待通りじゃないなら、VertexShaderからみなおそう
- チェック3：gl.disable(gl.CULL_FACE); を入れてみて見えるならカリングが原因
- チェック4：本当に何も描画できていないなら、シェーダーエラーが出てないかみなおそう

トラブル2：テクスチャを貼ると絵が出ない！



トラブル2：テクスチャを貼ると絵が出ない！

チェック1：まず、**トラブル1**と同じチェックでそもそもポリゴンが出てるかをチェックしよう
※GLSLとHLSL(DirectX)を混同してtex2Dとか書いてしまうこともある

チェック2：ブレンド方法を確かめよう

↓とりあえずブレンドを無効化してみる

`gl.disable(GL.BLEND);` //WebGLならこいつを直前にコールして描画されるならブレンド方法がおかしい

チェック3：テクスチャがバインドされていない、TEXTURE_MIN_FILTER, TEXTURE_MAG_FILTERを設定していない、textureハンドルと、indexを勘違いしている

`gl.activeTexture(gl.TEXTURE0);`

`gl.bindTexture(gl.TEXTURE_2D, hogetexture);` //hogetextureはcreateTextureでちゃんと作成されているものか？

`gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);` //0はTEXTUREユニットのindexであるか？

チェック4：UVがおかしい

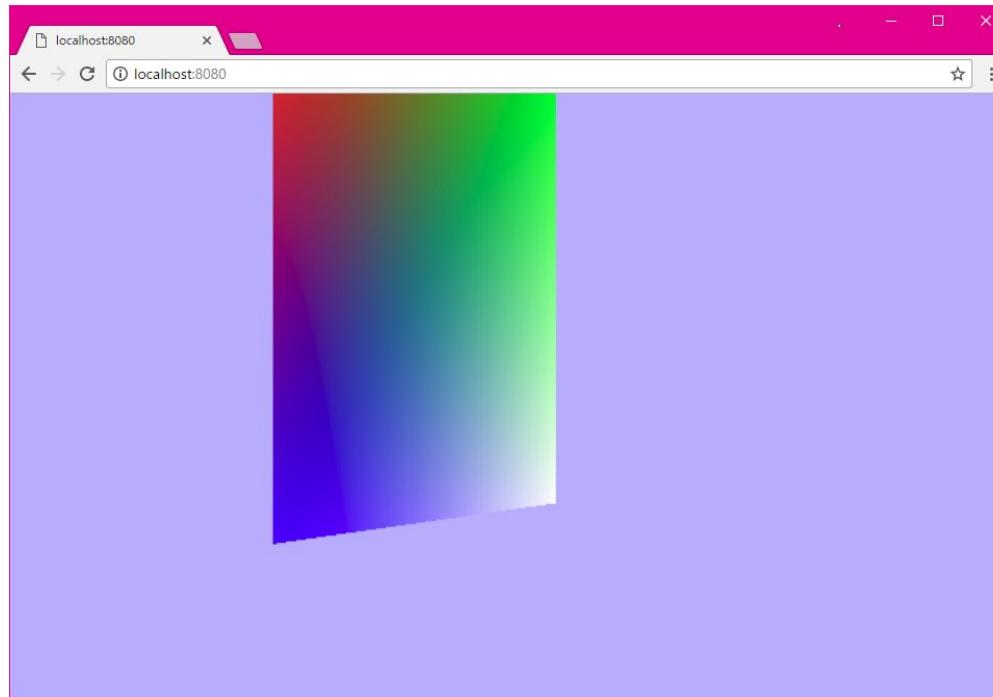
`gl_FragColor = vec4(uv, 0.0, 1.0);`

//表示してみておかしいかを見る。赤一色、黒だと怪しんでください

`vec2 test_uv = gl_FragCoord.xy / vec2(500.0);`

//テスト用のUV。これで何かしら出てたらVertexShader側で渡されてくるUV値を見直す

トラブル3：ポストエフェクト：UVがおかしい



のびてる

トラブル3：ポストエフェクト：UVがおかしい

チェック1：uniformで解像度を渡すとき間違った値を渡してませんか？

例)

```
glUniform4f(location, screenWidth, screenWidth, 0.0, 0.0);  
screenHeight = width; //ってなってる時も
```

チェック2：VertexShaderの時点でUVが間違ってませんか？

```
gl_FragColor = vec4(vertex_uv, 0.0, 1.0); //これで速攻で確認する
```

経験上、UVおかしい系はケアレスミスが多い印象

(モデルデータ自体のUVが異常、モデルデータ -> bufferDataに入れるときにUVデータを取り損ねてるなど)

トラブル4：よくわからないシェーダーの絵が出ない ※シェーダー例

```
void main() {  
    vec4 color = vec3(0.0);  
    color += get_albedo(uv);  
    color += get_specular(uv);  
    color *= get_ao(uv);  
    gl_FragColor = vec4(color, 1.0);  
}
```

トラブル4：よくわからないシェーダーの絵が出ない チェック1

```
void main() {
    vec4 color = vec3(0.0);
    gl_FragColor = vec4(1.0, 0.0, 1.0, 1.0); //トラブル1と同じように基本的なチェック
    return; //ここで終了させる
    color += get_albedo(uv);
    color += get_specular(uv);
    color *= get_ao(uv);
    gl_FragColor = vec4(color, 1.0);
}
```

トラブル4：よくわからないシェーダーの絵が出ない チェック2

```
void main() {
    vec4 color = vec3(0.0);
    color += get_albedo(uv);
    gl_FragColor = vec4(color, 1.0); //get_albedoの結果を確認する
    return;
    color += get_specular(uv);
    color *= get_ao(uv);
    gl_FragColor = vec4(color, 1.0);
}
```

トラブル4：よくわからないシェーダーの絵が出ない チェック3

```
void main() {
    vec4 color = vec3(0.0);
    color += get_albedo(uv);
    color += get_specular(uv);
    gl_FragColor = vec4(color, 1.0); //同様に
    return;
    color *= get_ao(uv);
    gl_FragColor = vec4(color, 1.0);
}
```

トラブル4：よくわからないシェーダーの絵が出ない チェック4

```
void main() {
    vec4 color = vec3(0.0);
    color += get_albedo(uv);
    color += get_specular(uv);
    color *= get_ao(uv);
    gl_FragColor = vec4(color, 1.0); //事前にget_aoがcolorとカケザンしているのに注意
    return;
    gl_FragColor = vec4(color, 1.0);
}
```

トラブル4のまとめ：returnを使って効果的にデバッグしましょう
(DirectXの場合はレンダリングした値を途中で return value;として返してデバッグ！)

トラブル5：0除算で絵がまっしろ(ないし真っ黒になる)

```
uniform float misc_value;  
void main() {  
    vec4 color = vec3(0.0);  
    color = get_color(uv) / vec3(misc_value); //misc_valueが0.0になってる可能性が  
    gl_FragColor = vec4(color, 1.0);  
}
```

すこし上級シェーダーだと割り算が使われるので、一つずつチェックすること
※エラーが出ないので

0divが一部に発生すると、異様に重くなる時もある

トラブルが発生した時の確認方法(まとめ)

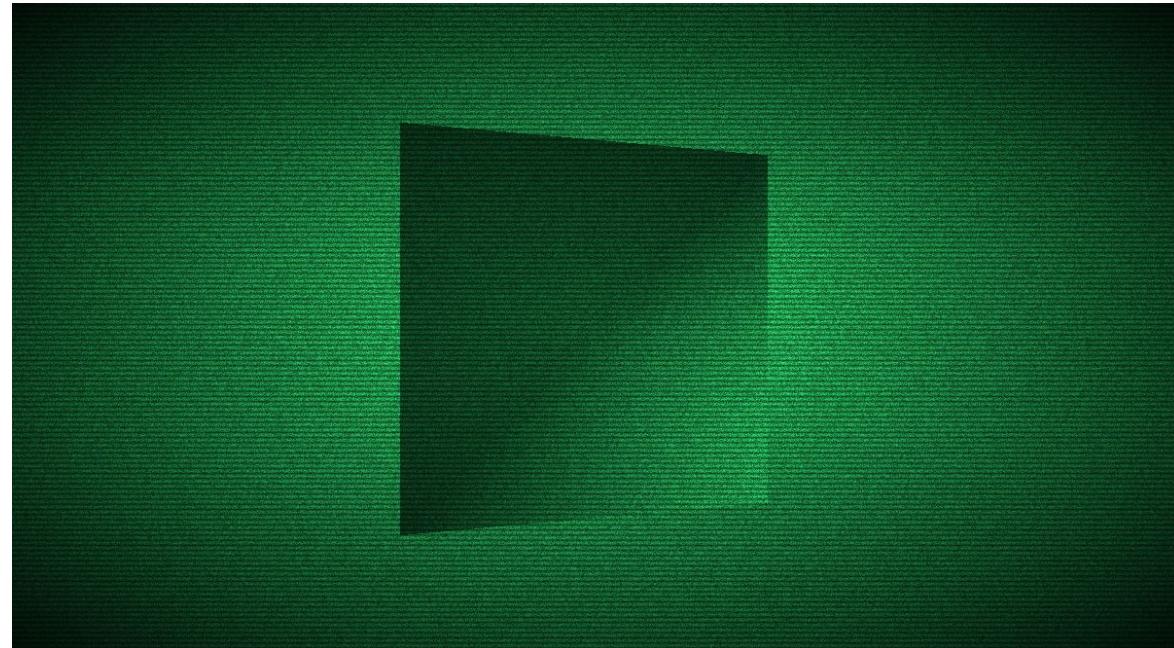
- ・絵が出ない → まず落ちついて、`gl_FragColor = チェックしたい色; return;` を書く。
- ・ケアレスミスが結構多いので、本当に間違ってるかどうか **単純なシェーダー** で確認
- ・VertexShaderもチェックしましよう → そもそも頂点ちゃんと出でますか？
- ・モデルデータを読み込ませて UV使ってトラブル → **モデルデータもうたがってみよう**
- ・色の計算結果を途中で出力するのも効果的 → `return;`を使う。`gl_FragColor.a = 1.0;`も！
- ・高度なシェーダーは誰かが作った関数がブラックボックスになってる場合もあるので、問題が発生したら`gl_FragColor`のからちょっとずつ解析(地味だけど強力です)

エフェクトを考える

つなげる

その1：改造編

ナイトスコープシェーダーを改造しよう :: 014_No01



その1：改造編

ナイトスコープシェーダーを改造しよう :: 014_No01

改造例：双眼鏡みたいなので覗いたような 2つ丸のレイアウトを導入したい



<https://outdoorempire.com/night-vision-gear-guide/>

その1：改造編

ナイトスコープシェーダーを改造しよう :: 014_No01

改造例：双眼鏡みたいなので覗いたような 2つ丸のレイアウトを導入したい



こういう丸と、ナイトスコープを乗算すればよい → 問題を単純化する

その1：改造編

ナイトスコープシェーダーを改造しよう :: 014_No01

双眼鏡みたいなので覗いたような2つ丸だけを出力するマスク関数を作る

```
//p = x,y -> -1.0～1.0が返却されること
float createMask(vec2 p, vec2 offset) {
    //アスペクト比を意識してUVを新円にする
    p.y /= resolution.x / resolution.y;

    //右
    float vignette0 = clamp(1.0 - length(p + offset), 0.0, 1.0);

    //左
    float vignette1 = clamp(1.0 - length(p - offset), 0.0, 1.0);

    //smoothstepでの0～1に変化させる開始と終了
    float startD = 0.50;
    float endD = 0.55;

    //マスクを作成
    float value = 0.0;
    value += smoothstep(startD, endD, vignette0);
    value += smoothstep(startD, endD, vignette1);

    //値域を制限して返却
    return clamp(value, 0.0, 1.0);
}
```

```
void main(){
    vec2 p = (gl_FragCoord.st / resolution) * 2.0 - 1.0;
    float mask_value = createMask(p, vec2(0.3, 0.0));
    gl_FragColor = vec4(vec3(mask_value), 1.0);
}
```



出力結果

その1：改造編

ナイトスコープシェーダーを改造しよう :: 014_No01

元のシェーダーに乗算する

```
void main(){
    vec2 p = (gl_FragCoord.st / resolution) * 2.0 - 1.0;

    vec4 samplerColor = texture2D(texture, vTexCoord);

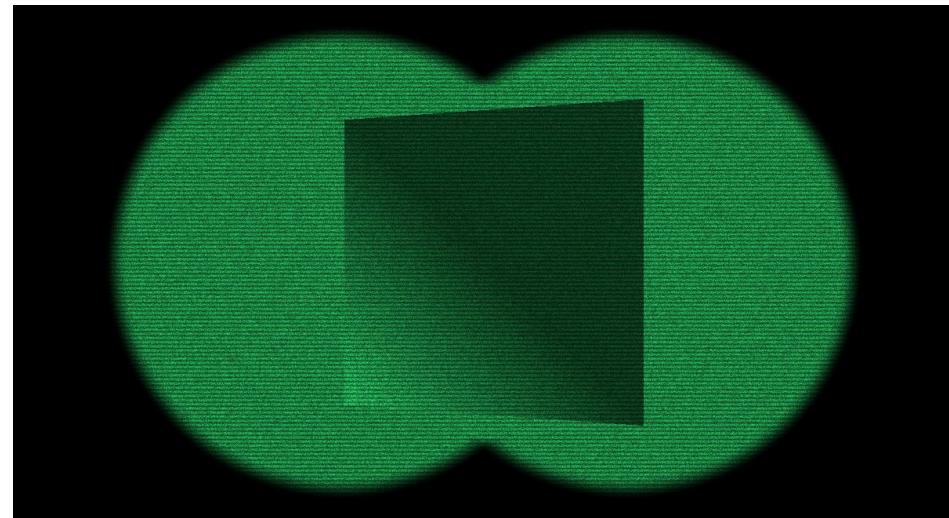
    float dest = (samplerColor.r + samplerColor.g + samplerColor.b) / 3.0;

    float noise = rnd(gl_FragCoord.st + mod(time, 10.0));
    dest *= noise * 0.5 + 0.5;

    float scanLine = abs(sin(p.y * 200.0 + time * 5.0)) * 0.5 + 0.5;
    dest *= scanLine;

    dest *= createMask(p, vec2(0.3, 0.0));

    gl_FragColor = greenColor * vec4(vec3(dest), 1.0);
}
```



出力結果

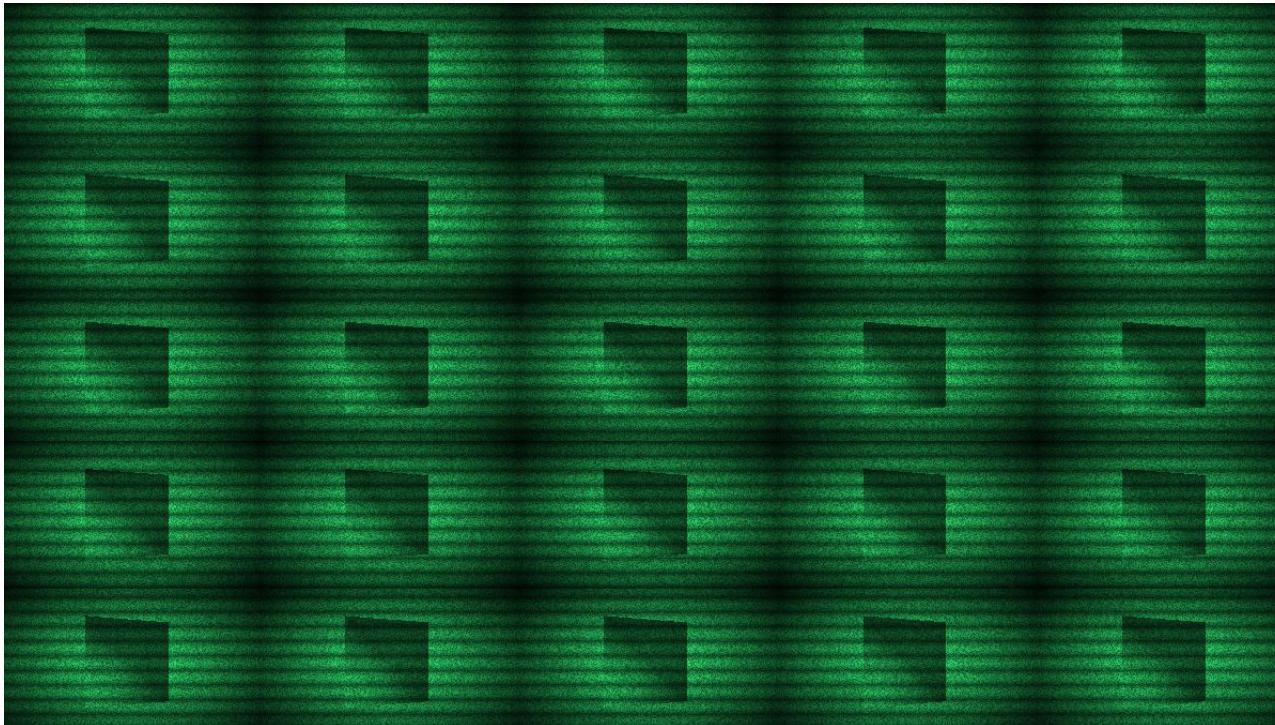
その1：改造編(まとめ)

ナイトスコープシェーダーを改造しよう :: 014_No01

- ・問題を単純化する(丸いマスクを2つ出力する関数を作成する)
- ・最初のうちは、まず関数化する(クラス化、部品化、レイヤーみたいに整理整頓)
- ・元になるエフェクトを **改造**する場合、ドラスティックに改造しない場合はできるだけすぐ分離できるようにした方が何かと後で楽



その2：テクスチャループをしよう
ナイトスコープシェーダーを改造しよう :: 014_No02



その2：テクスチャループをしよう :: 014_No02

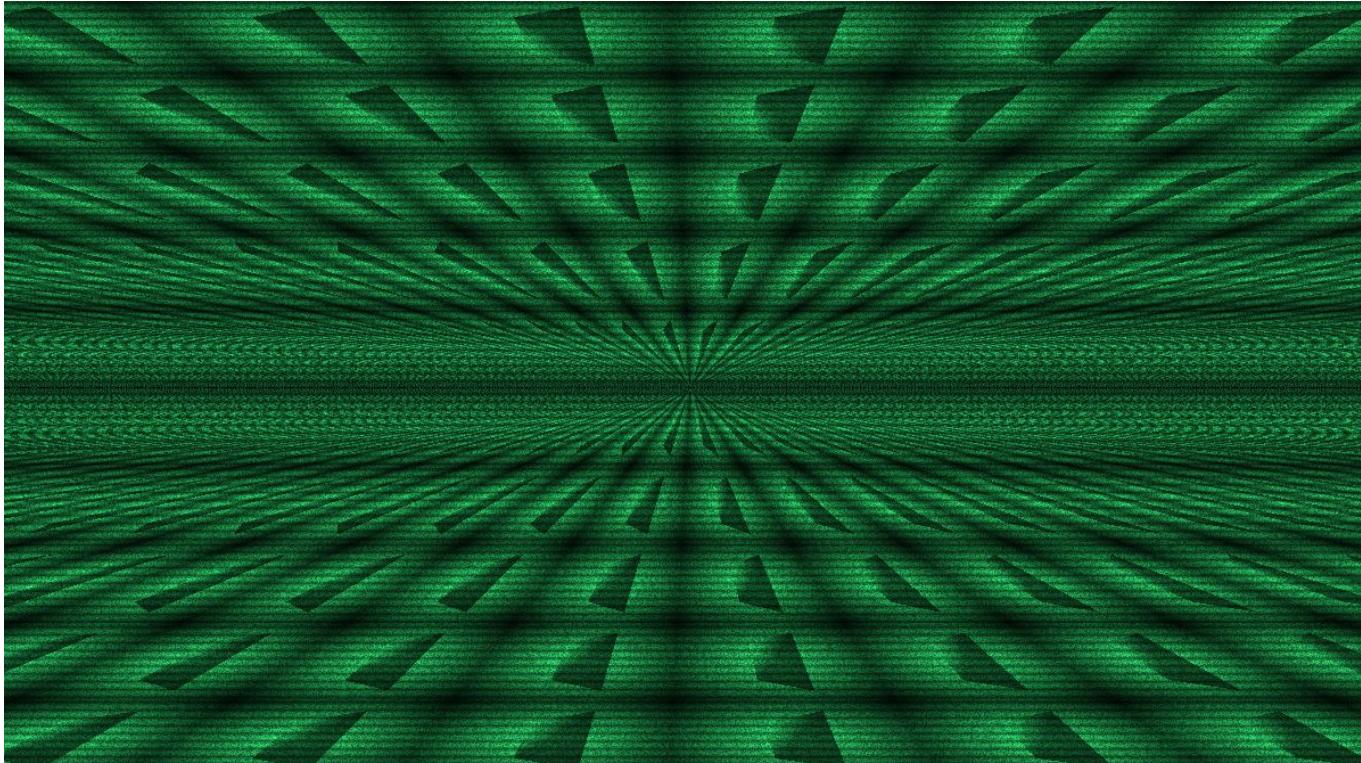
```
vec4 nightScope(sampler2D tex, vec2 uv){  
    vec2 p = uv * 2.0 - 1.0;  
    vec4 samplerColor = texture2D(tex, uv);  
    float dest = (samplerColor.r + samplerColor.g + samplerColor.b) / 3.0;  
    float vignette = 1.5 - length(p);  
    dest *= vignette;  
    float noise = rnd(gl_FragCoord.st + mod(time, 10.0));  
    dest *= noise * 0.5 + 0.5;  
    float scanLine = abs(sin(p.y * 200.0 + time * 5.0)) * 0.5 + 0.5;  
    dest *= scanLine;  
    return greenColor * vec4(vec3(dest), 1.0);  
}  
  
void main() {  
    const float loopNum = 5.0;  
    gl_FragColor = nightScope(texture, mod(vTexCoord * loopNum , 1.0)) //modと、渡すテクスチャをx5の座標にして、0-1でループ  
}
```

その2：テクスチャループをしよう :: 014_No02

```
vec4 nightScope(sampler2D tex, vec2 uv){  
    vec2 p = uv * 2.0 - 1.0;  
    vec4 samplerColor = texture2D(tex, uv);  
    float dest = (samplerColor.r + samplerColor.g + samplerColor.b) / 3.0;  
    float vignette = 1.5 - length(p);  
    dest *= vignette;  
    float noise = rnd(gl_FragCoord.st + mod(time, 10.0));  
    dest *= noise * 0.5 + 0.5;  
    float scanLine = abs(sin(p.y * 200.0 + time * 5.0)) * 0.5 + 0.5;  
    dest *= scanLine;  
    return greenColor * vec4(vec3(dest), 1.0);  
}  
  
void main() {  
    const float loopNum = 5.0;  
    gl_FragColor = nightScope(texture, mod(vTexCoord * loopNum, 1.0)) //modと、渡すテクスチャを5の座標にして、0-1でループ  
}
```

← 014のシェーダのmainを
nightScopeに名前変更して
uvいれて、returnするようにしただけ。
楽にテクスチャとして使用できるようにする
※おすすめ

その3：インチキパスをつけてみよう ナイトスコープシェーダーを改造しよう :: 014_No03



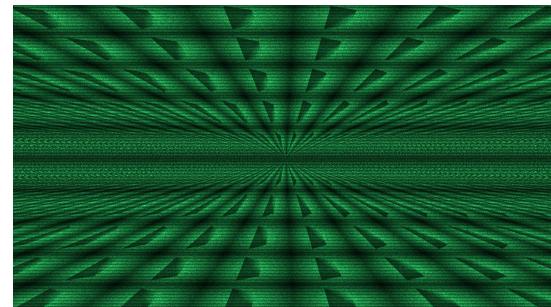
その3：インチキパースをつけてみよう ナイトスコープシェーダーを改造しよう :: 014_No03

```
void main() {
    vec2 uv = vTexCoord;      //ポリゴンのUV座標を取得
    uv = uv * 2.0 - 1.0;     //-1～1にする
    uv.x *= 1.0 / abs(uv.y * 2.0); //高さで横幅を割ってインチキパースUVを作る(uv.yは真ん中に行けば行くほど小さくなる)
    uv.y = abs(uv.y);         //画面上半分とした半分で同じYの値を使いたいので、絶対値を取る
    uv.x += time;             //X方向にスクロール(無くてもいい)
    uv = mod(uv * 5.0, 1.0); //出来上がったUVをテクスチャループ

    //ナイトスコープにしてレンダリング(もちろん岩とか、地面とかのテクスチャでもいい)
    gl_FragColor = nightScope(texture, uv);
}
```

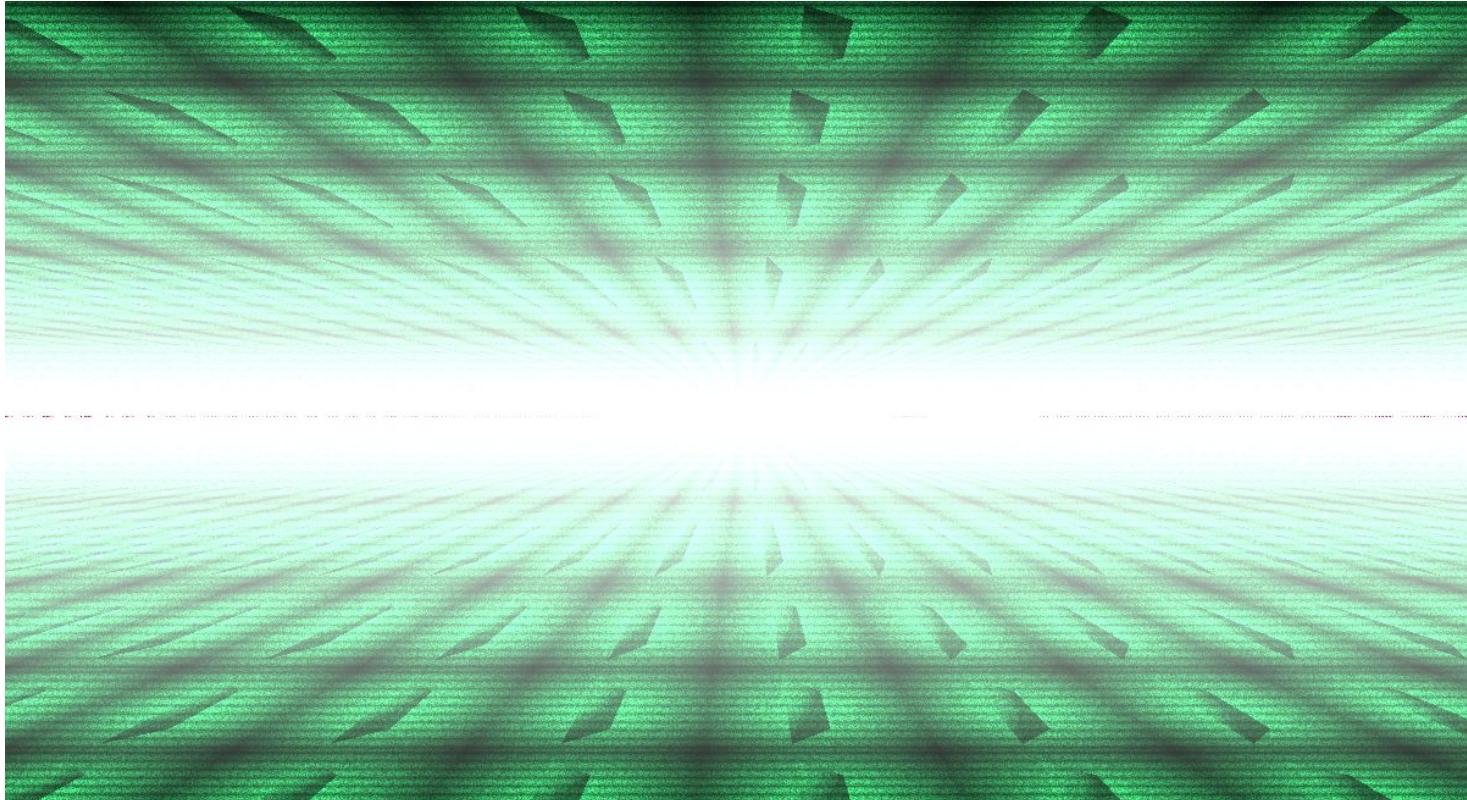
[遠近法]：同じ大きさの物でも、視点から遠いほど小さく描く

<https://ja.wikipedia.org/wiki/%E9%81%A0%E8%BF%91%E6%B3%95>



UVの操作だけで劇的に変わる

その4：フェイクフォグをつけてみよう::014_No04



その4：フェイクフォグをつけてみよう::014_No04

```
void main() {  
    vec2 uv = vTexCoord;      //ポリゴンのUV座標を取得  
    uv = uv * 2.0 - 1.0;    // -1~1にする  
    uv.x /= abs(uv.y * 2.0); //高さで横幅を割ってインチキバースUVを作る  
    uv.y = abs(uv.y);        //画面上半分とした半分で同じYの値を使いたいので、絶対値を取る  
  
    //フェイクフォグを作る(画面タテ中央がuv.y == 0。なので、1.0 - yで徐々にグラデーション  
    float fog = 1.0 - uv.y;  
  
    uv.x += time;           //X方向にスクロール(無くてもいい)  
    uv = mod(uv * 5.0, 1.0); //出来上がったUVをテクスチャループ  
  
    //ナイトスコープにしてレンダリング(もちろん岩とか、地面とかのテクスチャでもいい)  
    gl_FragColor = nightScope(texture, uv);  
  
    //フェイクフォグを適用  
    gl_FragColor.xyz += vec3(fog);  
}
```



UVの操作だけで劇的に変わる(その2)

その4：フェイクフォグをつけてみよう::014_No04

```
void main() {
    vec2 uv = vTexCoord;          //ポリゴンのUV座標を取得
    uv = uv * 2.0 - 1.0;         //-1～1にする
    uv.x /= abs(uv.y * 2.0);     //高さで横幅を割ってインチキバースUVを作る
    uv.y = abs(uv.y);            //画面上半分とした半分で同じYの値を使いたいので、絶対値を取る

    //フェイクフォグを作る(画面タテ中央がuv.y == 0。なので、1.0 - yで徐々にグラデーション
    float fog = 1.0 - uv.y;

    uv.x += time;                //X方向にスクロール(無くてもいい)
    uv = mod(uv * 5.0, 1.0);      //出来上がったUVをテクスチャループ

    //ナイトスコープにしてレンダリング(もちろん岩とか、地面とかのテクスチャでもいい)
    gl_FragColor = nightScope(texture, uv);

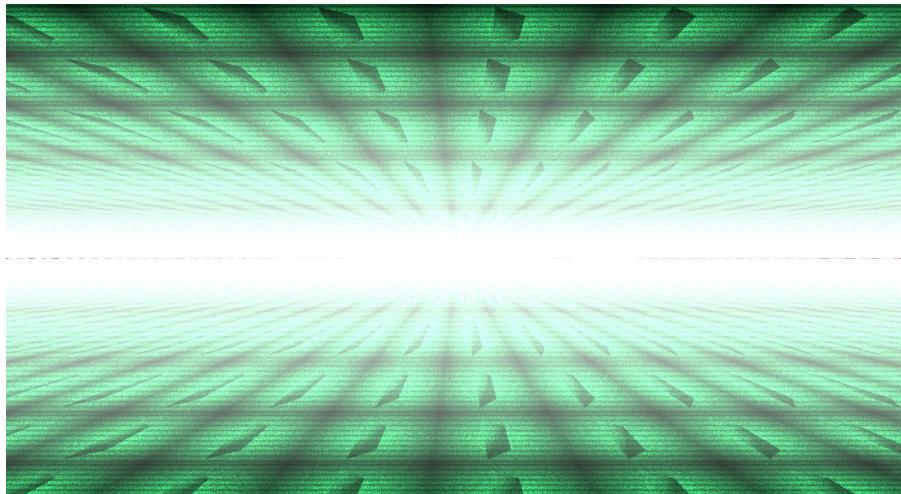
    //フェイクフォグを適用
    gl_FragColor.xyz += vec3(fog * fog); //これで完全になる
}
```



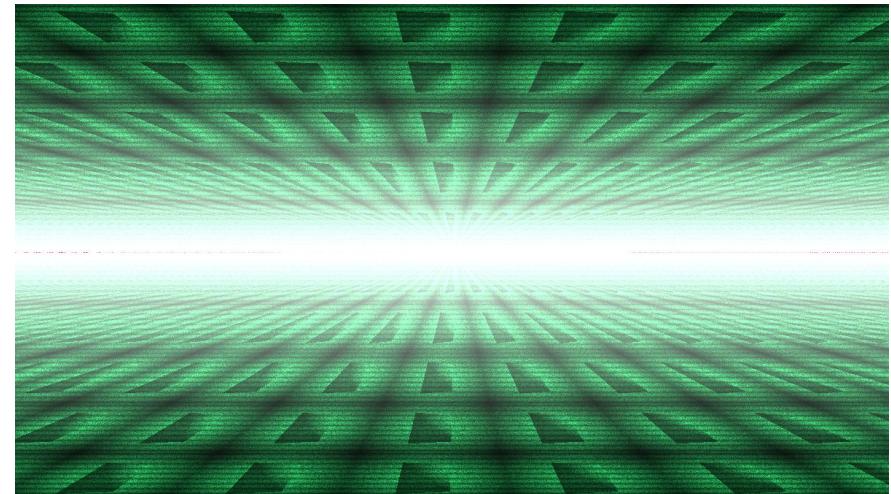
補足：逆2乗の法則：

<https://ja.wikipedia.org/wiki/%E9%80%862%E4%B9%97%E3%81%AE%E6%B3%95%E5%89%87>

その4：フェイクフォグをつけてみよう(結果)::014_No04



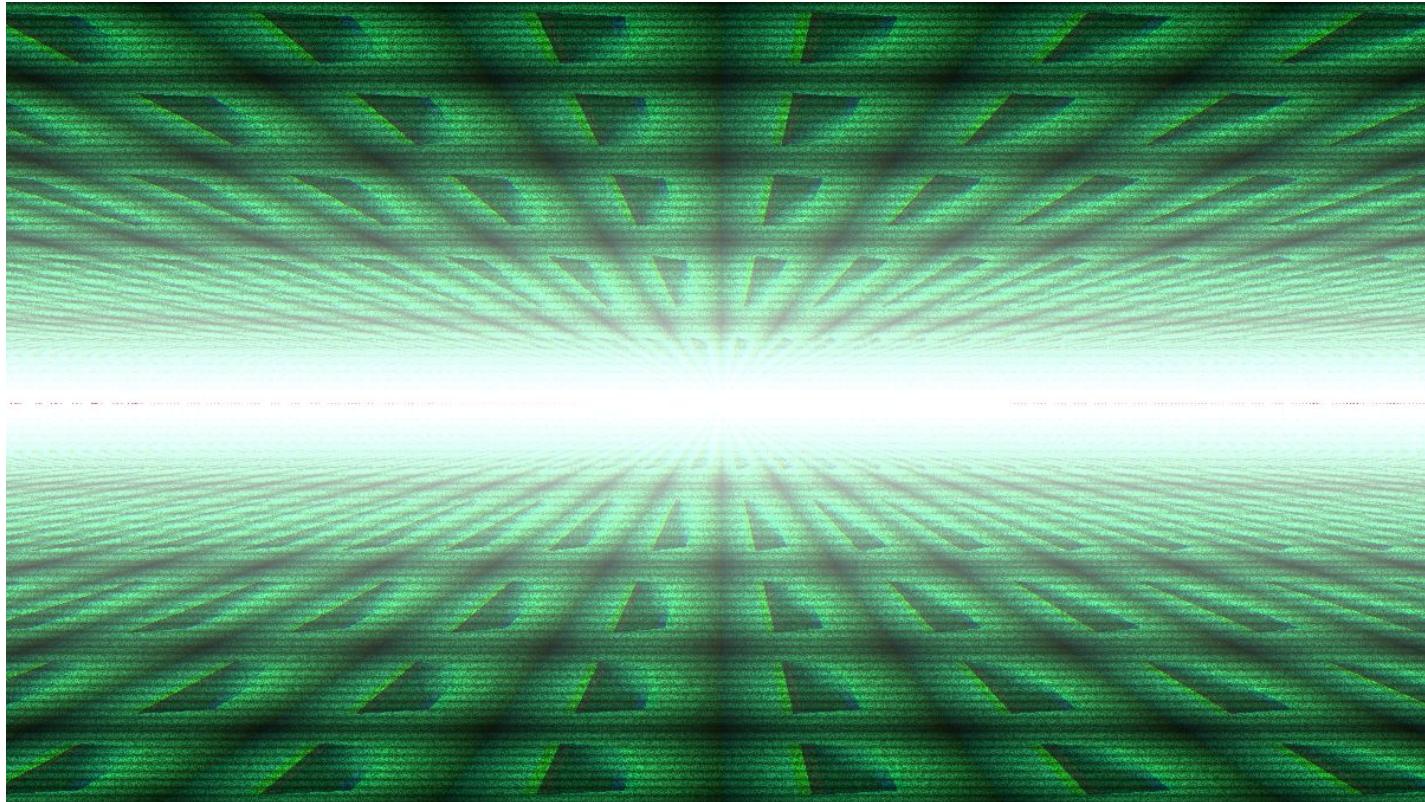
```
gl_FragColor += vec3(fog);
```



```
gl_FragColor += vec3(fog * fog);
```

どちらがいいかは場合によりけり

その5：色収差をつけてみる::014_No05



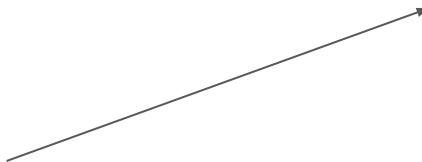
その5：色収差をつけてみる::014_No05

[色収差]

<https://ja.wikipedia.org/wiki/%E8%89%B2%E5%8F%8E%E5%B7%AE>

考え方：よく見ると
Rが左(uv.xがマイナス方向)にずれてる
Bが右(uv.xがマイナス方向)にずれてる

※この考え方で近似する(フェイク)



その5：色収差をつけてみる::014_No05

```
void main() {
    vec2 uv = vTexCoord;      //ポリゴンのUV座標を取得
    uv = uv * 2.0 - 1.0;     //‐1～1にする
    uv.x /= abs(uv.y * 2.0); //高さで横幅を割ってインチキパースUVを作る
    uv.y = abs(uv.y);        //画面上半分とした半分で同じYの値を使いたいので、絶対値を取る

    //フェイクフォグを作る(画面タテ中央がuv.y == 0。なので、1.0 - yで徐々にグラデーション
    float fog = 1.0 - uv.y;

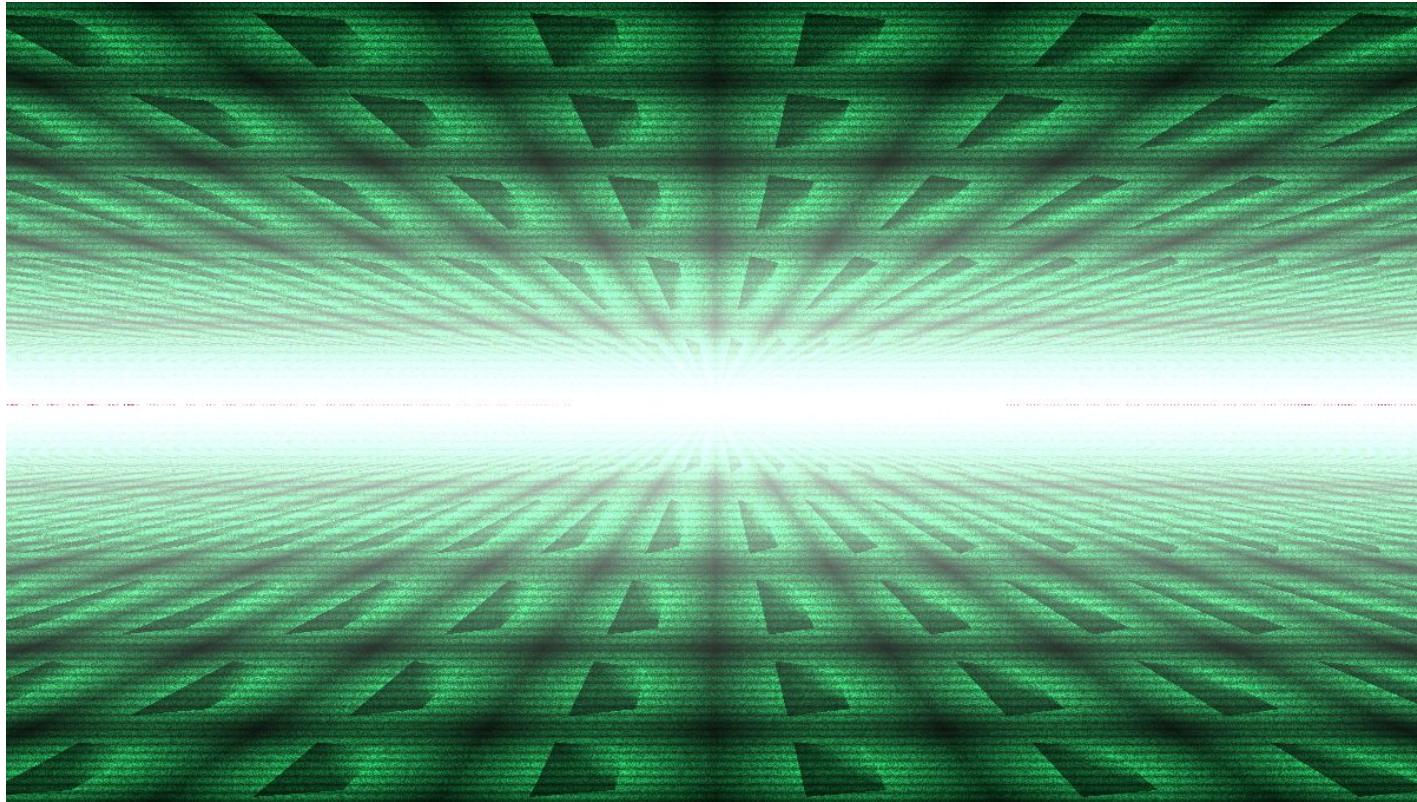
    uv.x += time;           //X方向にスクロール(無くてもいい)
    uv = mod(uv * 5.0, 1.0); //出来上がったUVをテクスチャループ

    //ナイトスコープにしてレンダリング(もちろん岩とか、地面とかのテクスチャでもいい)
    gl_FragColor = nightScope(texture, uv);

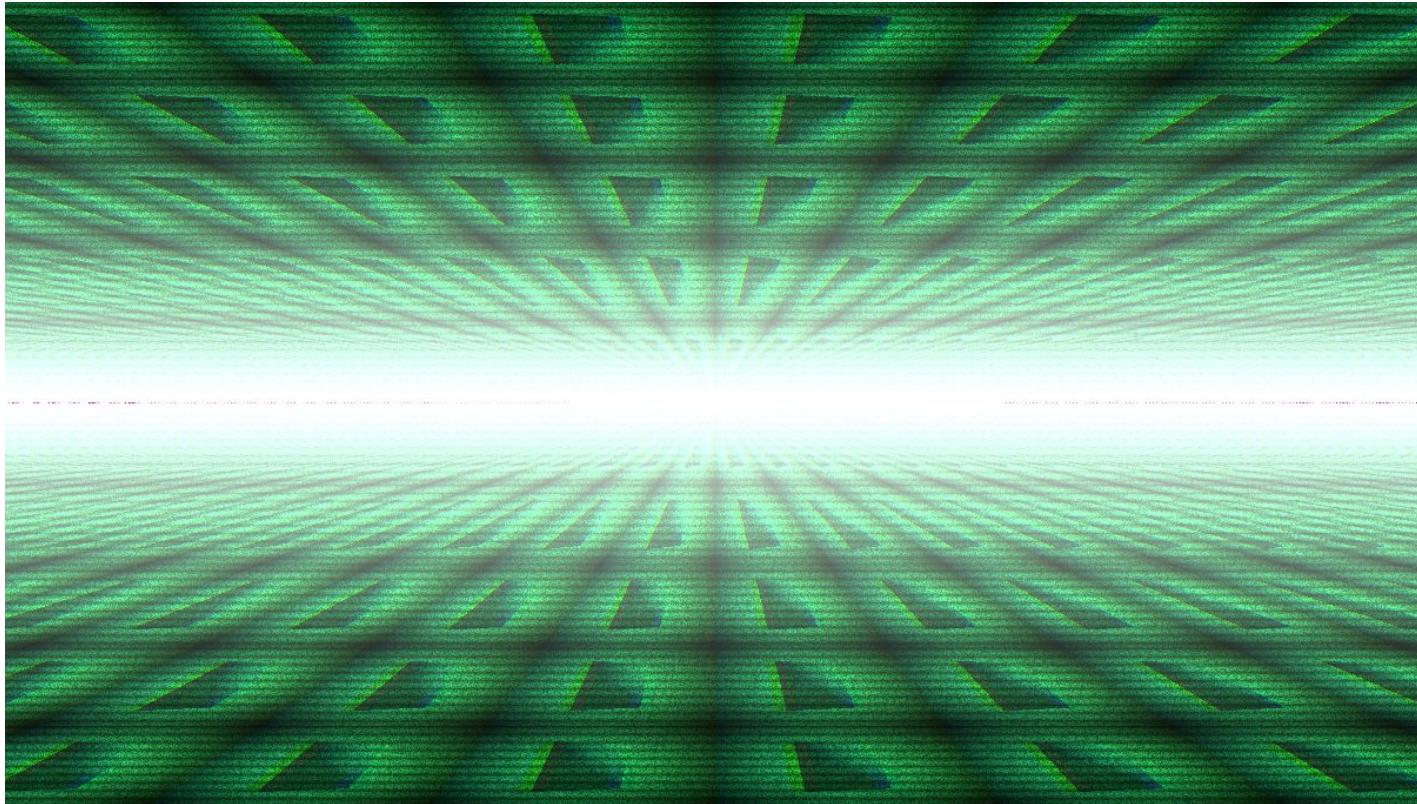
    //色収差をつけてみる(https://ja.wikipedia.org/wiki/%E8%89%B2%E5%8F%8E%E5%B7%AE)
    const vec2 offset = vec2(0.05, 0.0);
    gl_FragColor.r = nightScope(texture, uv - offset).r; //UVを左にずらしてナイトスコープのR成分を取得
    gl_FragColor.b = nightScope(texture, uv + offset).b; //UVを右にずらしてナイトスコープのB成分を取得

    //フェイクフォグを適用
    gl_FragColor.xyz += vec3(fog * fog);
}
```

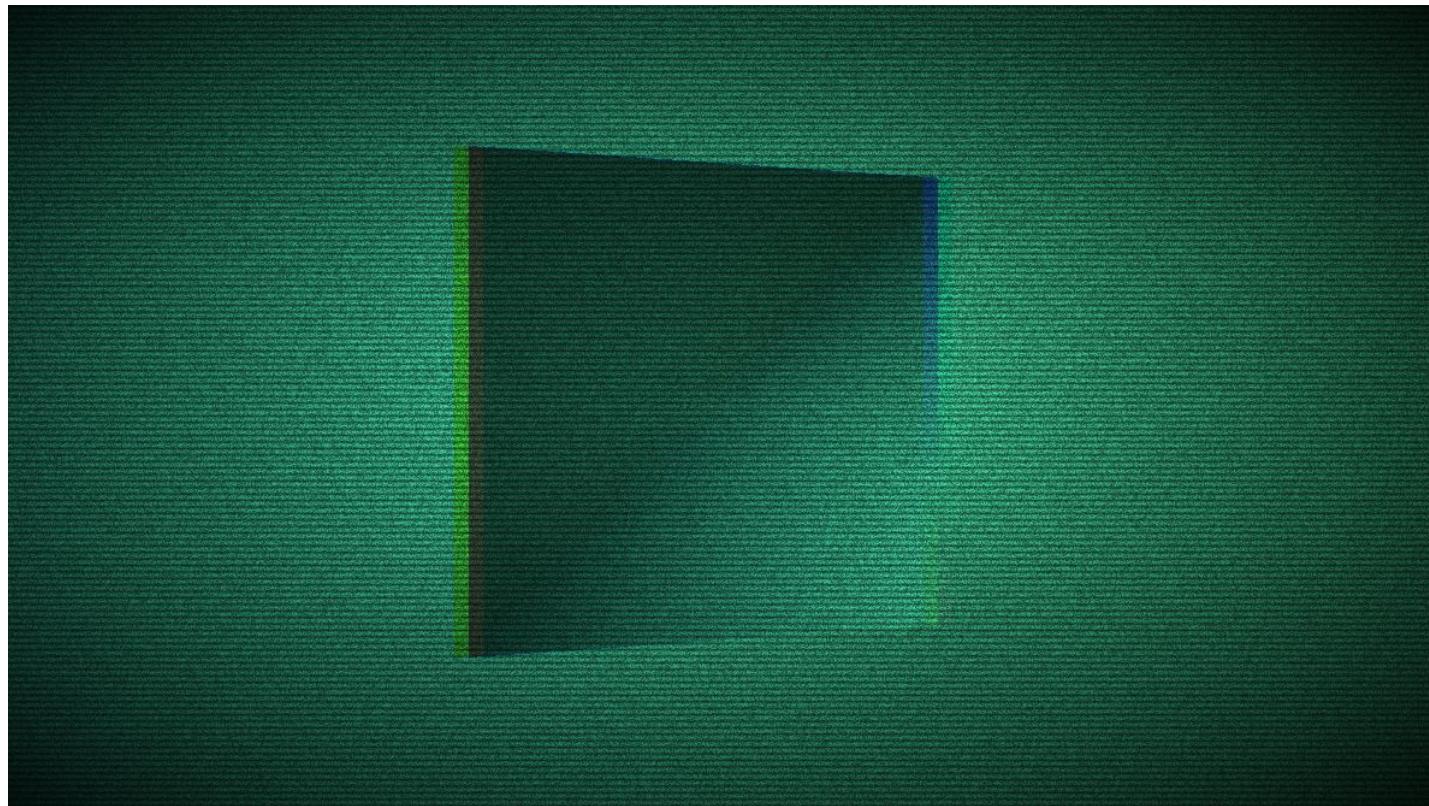
その5：色収差をつけてみる(適用前)::014_No05



その5：色収差をつけてみる(適用後)::014_No05



その5：補足 色収差だけをナイトスコープに::014_No06



エフェクトを考える、つなげる(まとめ)

- ・まずは、最初はサンプルを改造しましょう。手を動かしてみてください
- ・シェーダーを育てる(と表現する。細かい調整の積み重ねで劇的に変わる)
- ・色をtexture, uvで取得するものはマスクなど分離できるものは関数にして整理
- ・フェイクで近似する場合、対象物をよく見てどんな見た目があるのか並べて、それを書き下すだけで大まかに再現できるものもあります
- ・スウィズル(color.xyz, color.yzx, color.yz)を活用する
- ・実際に狙った効果を作るのが難しいです
- ・偶然を楽しんでください

補足：フラグメントシェーダーの課題

OpenGL ESの場合、VertexShaderからのUVの精度に注意

```
precision mediump float;
```

```
precision highp float;
```

参考：<http://wate.jp/wp/?p=170>

フラグメントシェーダーだけで何でもかんでもしようとする、結構時間がかかる

→例マスク抜き：マスクをテクスチャから読み込ませて作った方がいいのか

シェーダーだけで試行錯誤した方がいいのか、時間と労力など天秤にかけて

何でもかんでもフラグメントシェーダーでやる

→簡単だけどワナにはまつたり、解像度変わったときに破綻して使えなくなる時が結構痛い

実際に狙った効果を作るのが難しい

→なれるしかない。自分なりの手筋をまとめてどこかに作り置きしてするのが吉

OpenGL ESの場合、UVの精度を落とすと速度が向上する場合がある

→最後の手段として。精度が落ちる == レンダリングの品質も劇的に落ちる

参考になる文献

[ガウシアンブラーのサンプル]

<https://github.com/Jam3/glsl-fast-gaussian-blur>

[dFdxとdFdyでわずか4行のお手軽エッジ検出！]

<https://qiita.com/gam0022/items/1342a91d0a6b16a3a9ba>

[Raymarching Distance Fields]

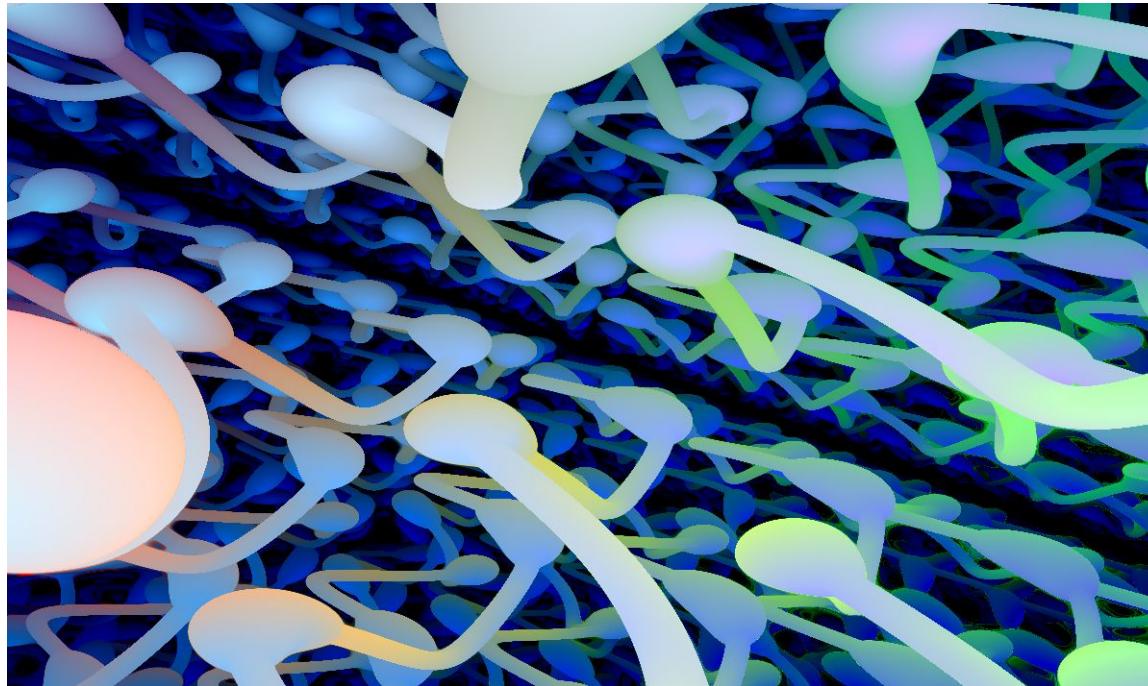
http://9bitscience.blogspot.jp/2013/07/raymarching-distance-fields_14.html

[書籍：フルスクラッチによるグラフィックスプログラミング入門]

<https://www.amazon.co.jp/dp/479800958X>

ご清聴ありがとうございました

(Enjoy FragmentShader!)



014_No007