レイマーチング0から1

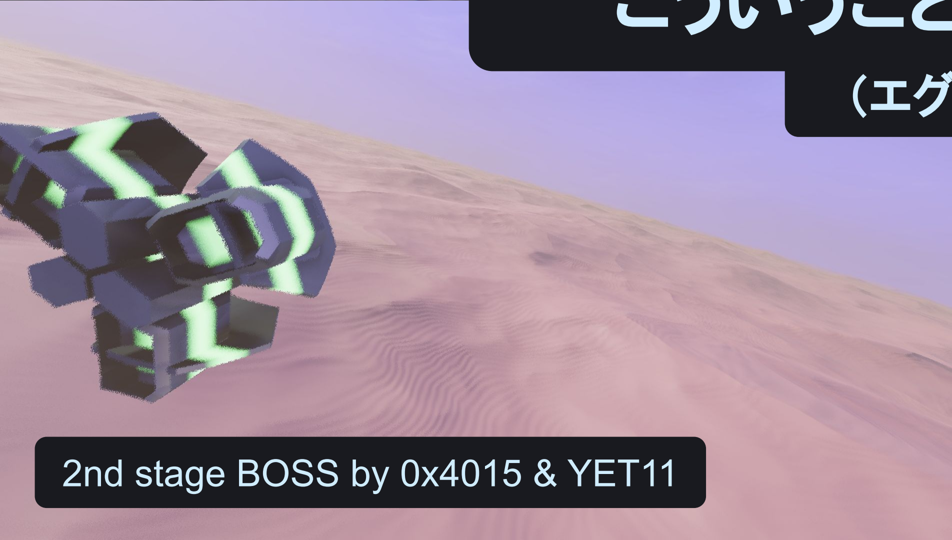
0b5vr

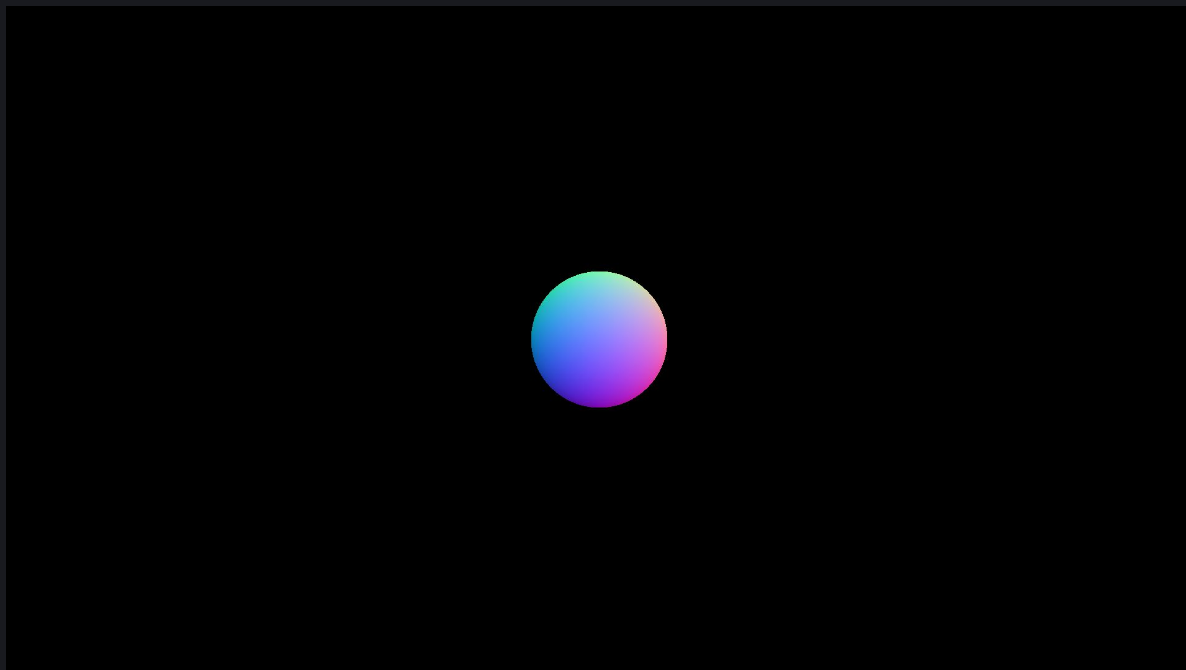
2023-04-29 SESSIONS in C4 LAN 2023 SPRING

レイマーチングってなあに？

こういうことができます

（エグい例）

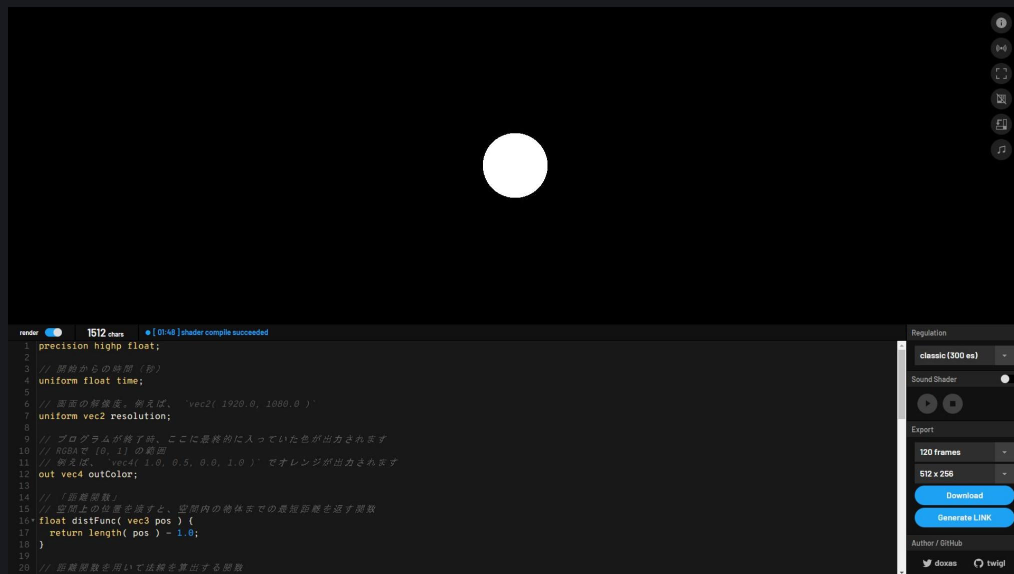




まずはここからはじめましょう

必要なもの

- Webブラウザ
  0b5vrはChromeを使います
- ベクトルとかの知識
  もしくは、ベクトルが出てきても泣かない心
- プログラミング
  もしくは、文章読解能力
- （あれば）GPU
  強ければ強いほど良い



twiglというプラットフォームを使ってコードを書いていきます

https://twigl.app/?ss=-NTymC1niuRzJs2RVbrn

```glsl
precision highp float;

// 開始からの時間(秒)
uniform float time;

// 画面の解像度。例えば、`vec2( 1920.0, 1080.0 )`
uniform vec2 resolution;

// プログラムが終了時、ここに最終的に入っていた色が出力されます
// RGBAで [0, 1] の範囲
// 例えば、`vec4( 1.0, 0.5, 0.0, 1.0 )` でオレンジが出力されます
out vec4 outColor;

//「距離関数」
// 空間上の位置を渡すと、空間内の物体までの最短距離を返す関数
float distFunc( vec3 pos ) {
  return length( pos ) - 1.0;
}

// 距離関数を用いて法線を算出する関数
// いろいろな方法がある中の1手法です
vec3 normalFunc( vec3 pos ) {
  vec2 d = vec2( 0.0, 0.0001 );
  return normalize( vec3(
    distFunc( pos + d.yxx ) - distFunc( pos - d.yxx ),
    distFunc( pos + d.xyx ) - distFunc( pos - d.xyx ),
    distFunc( pos + d.xxy ) - distFunc( pos - d.xxy )
  ) );
}

void main() {
  // 画面上のピクセルの位置を (0, 1) の範囲で格納した2次元ベクトル
  vec2 uv = gl_FragCoord.xy / resolution;

  // 上で定義したuvを、画面中心を原点に・縦横比が1:1の座標系に変換する
  vec2 screenPos = 2.0 * uv - 1.0;
  screenPos.x *= resolution.x / resolution.y;

  // レイ(光線)の始点と向きを定義する
  vec3 rayOri = vec3( 0.0, 0.0, 5.0 );
  vec3 rayDir = normalize( vec3( screenPos, -1.0 ) );

  // レイマーチングを行う
  float t = 0.0; // 現在のレイの始点から探索位置までの距離
  float dist; // 直近の距離関数の結果

  for ( int i = 0; i < 100; i ++ ) {
    // 現在の探索位置を使って距離関数を実行、distに結果を格納する
    dist = distFunc( rayOri + t * rayDir );

    // 距離関数の結果を使って、探索位置を更新する
    t += dist;
  }

  // もし、直近の距離関数の結果が十分にゼロに近かった場合
  // レイが距離関数で表現された物体表面と交差したと判定する
  if ( dist < 0.01 ) {
    // 交差した場合、物体の色として白を描画する
    outColor = vec4( 1.0, 1.0, 1.0, 1.0 );
  } else {
    // 交差しなかった場合、背景色として黒を描画する
    outColor = vec4( 0.0, 0.0, 0.0, 1.0 );
  }
}
```

コメント付きで約70行のコード

「GLSL」というプログラミング言語で書かれています

```
precision highp float;

// 開始からの時間（秒）
uniform float time;

// 画面の解像度。例えば、`vec2( 1920.0, 1080.0 )`
uniform vec2 resolution;

// プログラムが終了時、ここに最終的に入っていた色が出力されます
// RGBAで [0, 1] の範囲
// 例えば、`vec4( 1.0, 0.5, 0.0, 1.0 )` でオレンジが出力されます
out vec4 outColor;
```

```
//「距離関数」
// 空間上の位置を渡すと、空間内の物体までの最短距離を返す関数
float distFunc( vec3 pos ) {
  return length( pos ) - 1.0;
}

// 距離関数を用いて法線を算出する関数
// いろいろな方法がある中の1手法です
vec3 normalFunc( vec3 pos ) {
  vec2 d = vec2( 0.0, 0.0001 );
  return normalize( vec3(
    distFunc( pos + d.yxx ) - distFunc( pos - d.yxx ),
    distFunc( pos + d.xyx ) - distFunc( pos - d.xyx ),
    distFunc( pos + d.xxy ) - distFunc( pos - d.xxy )
  ) );
}
```

```
void main() {
  // 画面上のピクセルの位置を (0, 1) の範囲で格納した2次元ベクトル
  vec2 uv = gl_FragCoord.xy / resolution;

  // 上で定義したuvを、画面中心を原点に・縦横比が1:1の座標系に変換する
  vec2 screenPos = 2.0 * uv - 1.0;
  screenPos.x *= resolution.x / resolution.y;

  // レイ（光線）の始点と向きを定義する
  vec3 rayOri = vec3( 0.0, 0.0, 5.0 );
  vec3 rayDir = normalize( vec3( screenPos, -1.0 ) );

  // レイマーチングを行う
  float t = 0.0; // 現在のレイの始点から探索位置までの距離
  float dist; // 直近の距離関数の結果

  for ( int i = 0; i < 100; i ++ ) {
    // 現在の探索位置を使って距離関数を実行、distに結果を格納する
    dist = distFunc( rayOri + t * rayDir );

    // 距離関数の結果を使って、探索位置を更新する
    t += dist;
  }

  // もし、直近の距離関数の結果が十分にゼロに近かった場合
  // レイが距離関数で表現された物体表面と交差したと判定する
  if ( dist < 0.01 ) {
    // 交差した場合、物体の色として白を描画する
    outColor = vec4( 1.0, 1.0, 1.0, 1.0 );
  } else {
    // 交差しなかった場合、背景色として黒を描画する
    outColor = vec4( 0.0, 0.0, 0.0, 1.0 );
  }
}
```

uniform / out変数の定義

補助関数の定義

main関数の定義

```glsl
precision highp float;

// 開始からの時間（秒）
uniform float time;

// 画面の解像度。例えば、`vec2( 1920.0, 1080.0 )`
uniform vec2 resolution;

// プログラムが終了時、ここに最終的に入っていた色が出力されます
// RGBAで [0, 1] の範囲
// 例えば、`vec4( 1.0, 0.5, 0.0, 1.0 )` でオレンジが出力されます
out vec4 outColor;
```

材料の用意

今回使う材料です

```glsl
//「距離関数」
// 空間上の位置を渡すと、空間内の物体までの最短距離を返す関数
float distFunc( vec3 pos ) {
  return length( pos ) - 1.0;
}

// 距離関数を用いて法線を算出する関数
// いろいろな方法がある中の1手法です
vec3 normalFunc( vec3 pos ) {
  vec2 d = vec2( 0.0, 0.0001 );
  return normalize( vec3(
    distFunc( pos + d.yxx ) - distFunc( pos - d.yxx ),
    distFunc( pos + d.xyx ) - distFunc( pos - d.xyx ),
    distFunc( pos + d.xxy ) - distFunc( pos - d.xxy )
  ) );
}
```

下ごしらえ

本調理に必要なタネを作っていきます

```glsl
void main() {
  // 画面上のピクセルの位置を (0, 1) の範囲で格納した2次元ベクトル
  vec2 uv = gl_FragCoord.xy / resolution;

  // 上で定義したuvを、画面中心を原点に・縦横比が1:1の座標系に変換する
  vec2 screenPos = 2.0 * uv - 1.0;
  screenPos.x *= resolution.x / resolution.y;

  // レイ（光線）の始点と向きを定義する
  vec3 rayOri = vec3( 0.0, 0.0, 5.0 );
  vec3 rayDir = normalize( vec3( screenPos, -1.0 ) );

  // レイマーチングを行う
  float t = 0.0; // 現在のレイの始点から探索位置までの距離
  float dist; // 直近の距離関数の結果

  for ( int i = 0; i < 100; i ++ ) {
    // 現在の探索位置を使って距離関数を実行、distに結果を格納する
    dist = distFunc( rayOri + t * rayDir );

    // 距離関数の結果を使って、探索位置を更新する
    t += dist;
  }

  // もし、直近の距離関数の結果が十分にゼロに近かった場合
  // レイが距離関数で表現された物体表面と交差したと判定する
  if ( dist < 0.01 ) {
    // 交差した場合、物体の色として白を描画する
    outColor = vec4( 1.0, 1.0, 1.0, 1.0 );
  } else {
    // 交差しなかった場合、背景色として黒を描画する
    outColor = vec4( 0.0, 0.0, 0.0, 1.0 );
  }
}
```

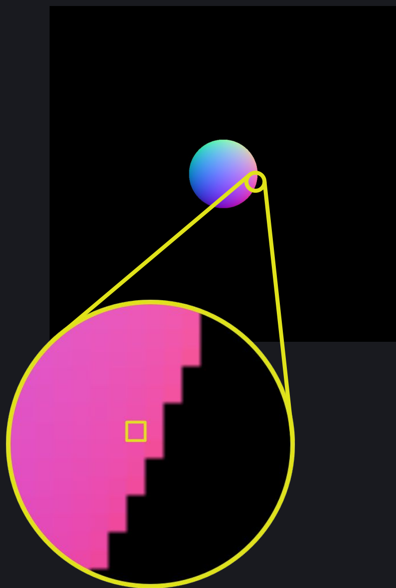
本調理

GLSLでは本調理はmainという関数の中

解決したい課題

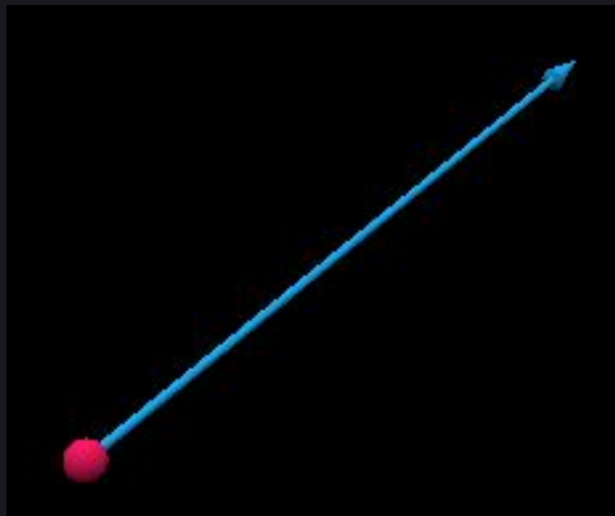
画面上のあるピクセル
から飛ぶレイが
物体と交差しているか？

ピクセル

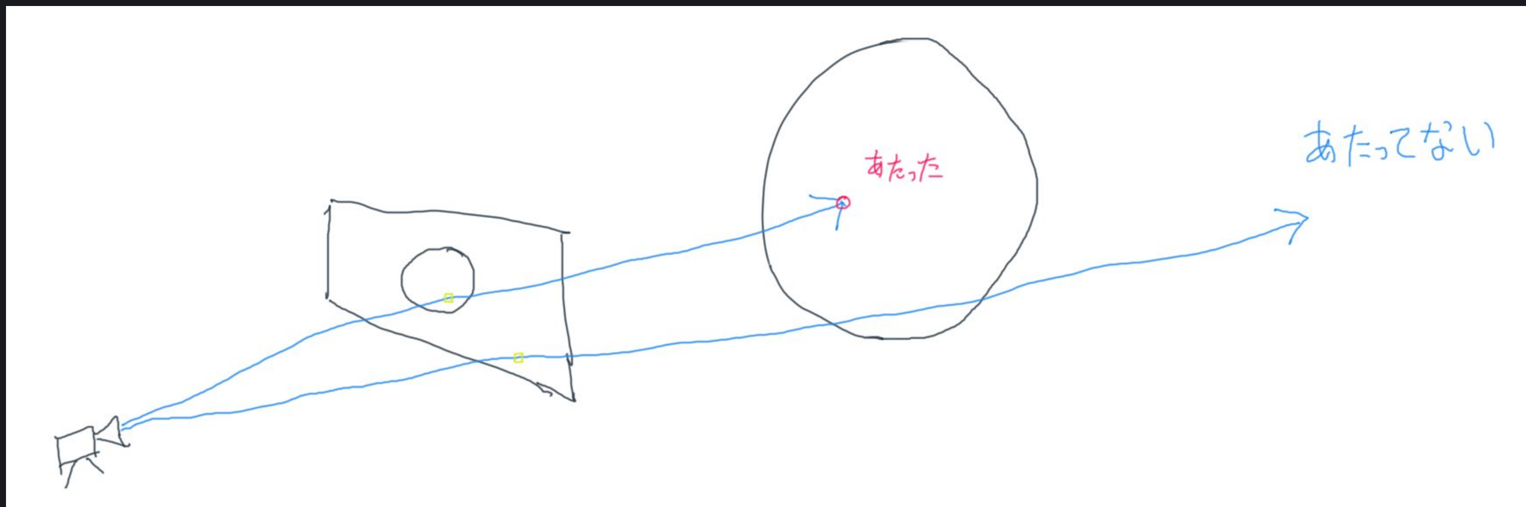


画像上のある一点の色情報

レイ

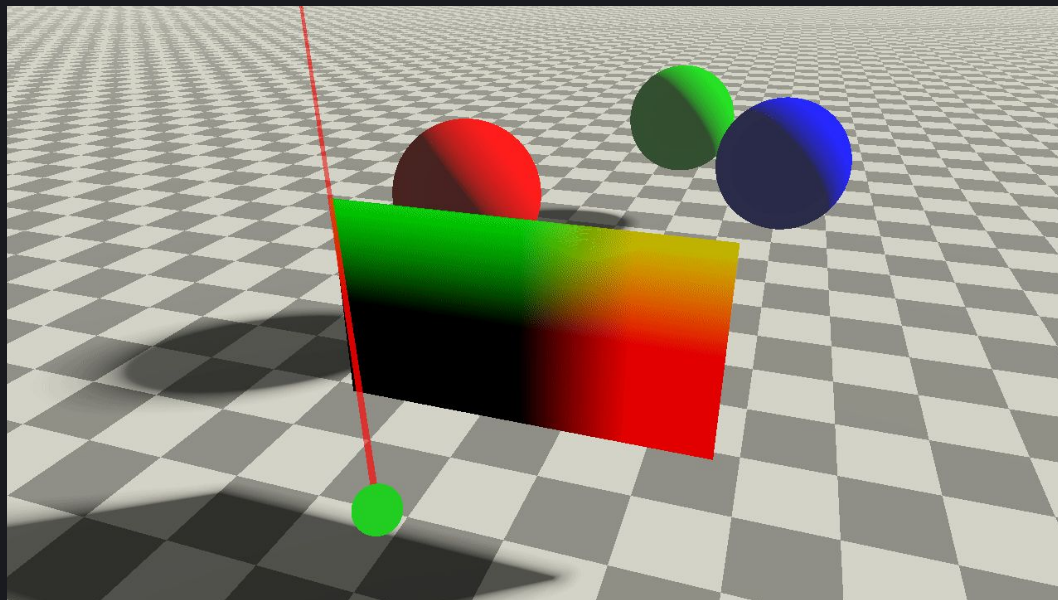


始点と向きを持つ線



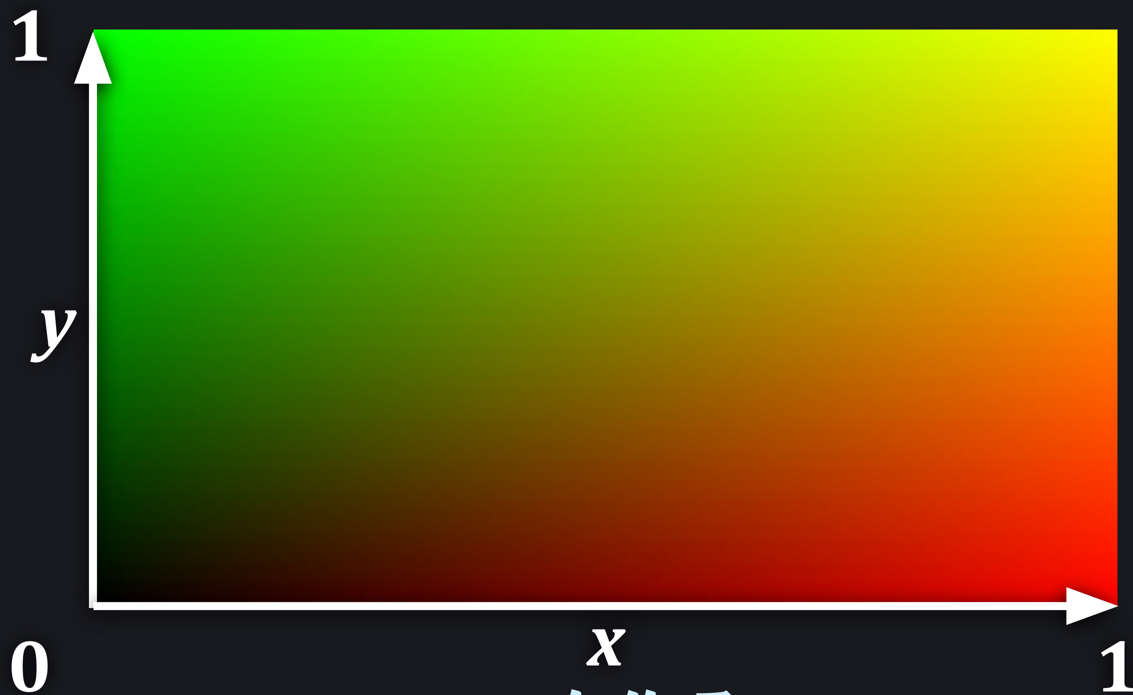
レイが物体と交差している → 物体を描く
レイが物体と交差していない → 背景を描く

ピクセルに対応するレイが飛び物体との交差判定が行われる様子



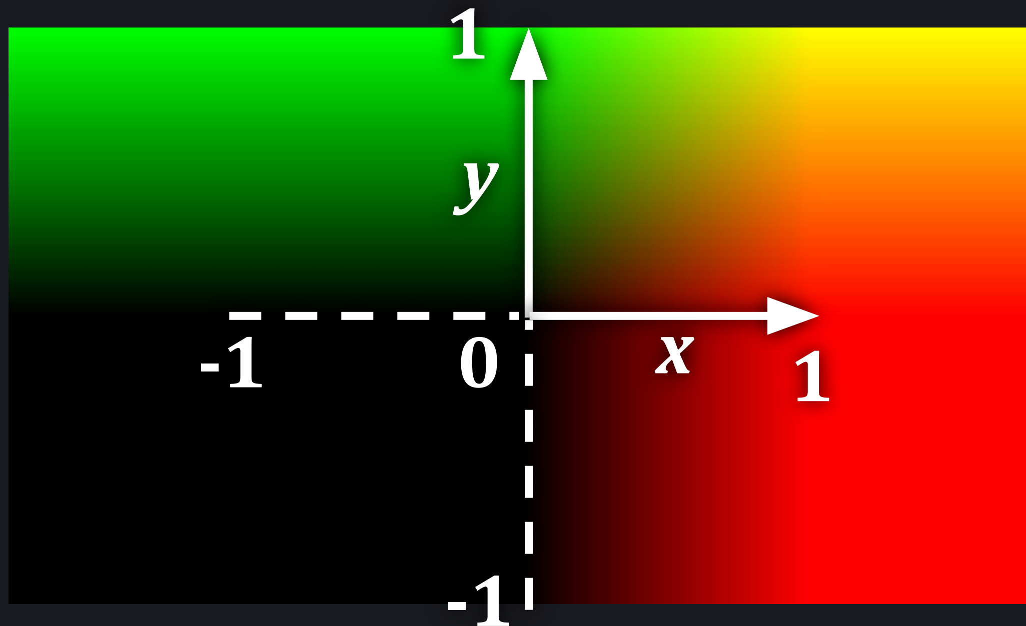
Source: kaneta
https://qiita.com/kaneta1992/items/21149c78159bd27e0860

コードを読んでみよう



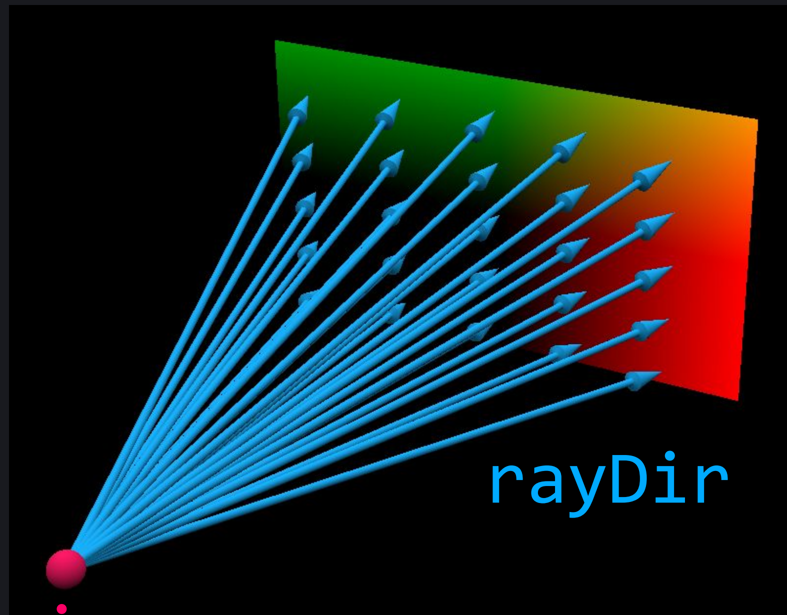
uvを作る

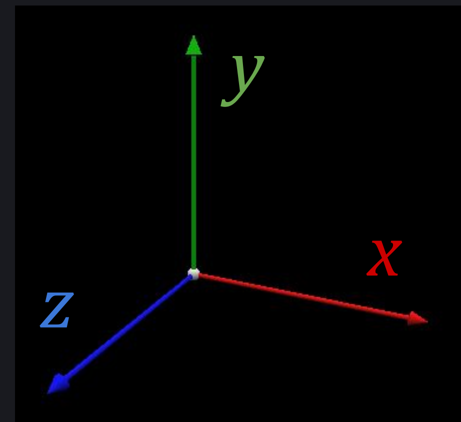
画面のxy座標を[0, 1]の範囲で持つ



uvを加工してscreenPosを作る

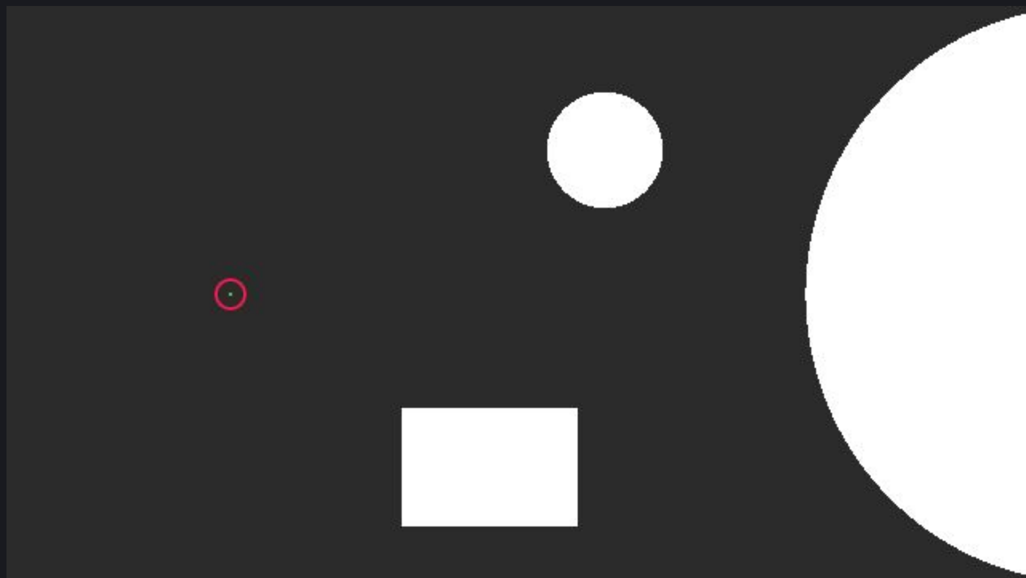
画面の真ん中が原点で
縦横比が1:1の座標系





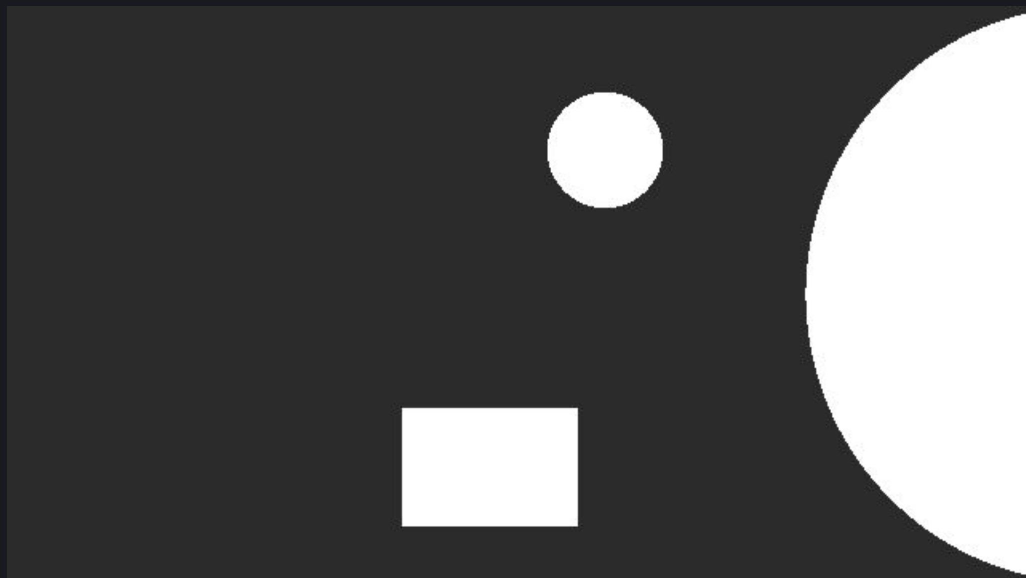
rayOri, rayDirを作る

各ピクセルに対応するレイの始点と方向



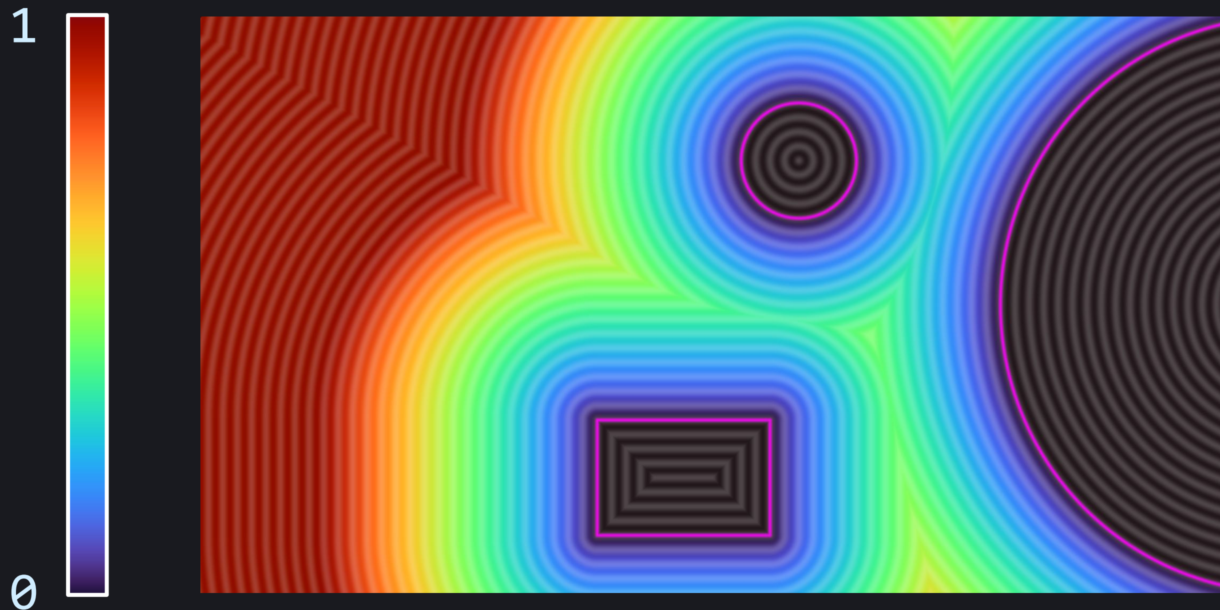
レイマーチング

距離関数を使ってレイを飛ばそう



距離関数(distFunc)

空間位置を渡すと、一番近い表面までの距離を返す関数



距離関数(distFunc)

空間位置を渡すと、一番近い表面までの距離を返す関数

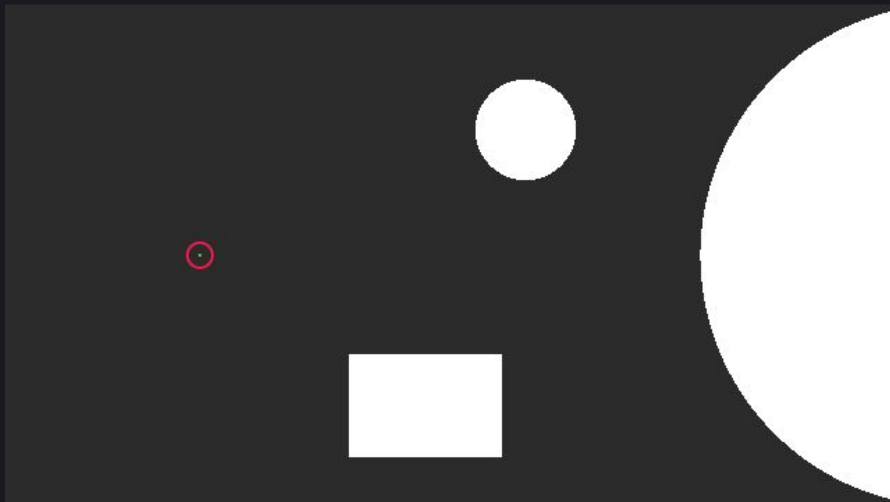
レイの始点に
探索位置を定義

繰り返し

現在の探索位置で
距離関数を実行

距離関数の返り値分
探索位置を前に進ませる

最後の距離関数の結果が
十分に小さかったら
衝突している



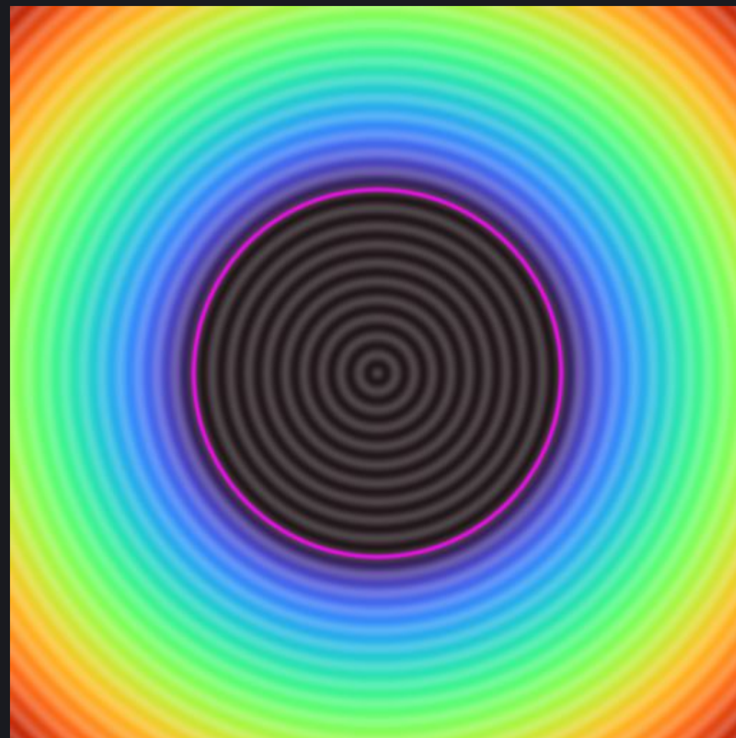
球の距離関数

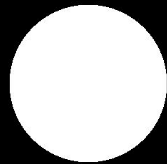
中心からの距離 - 半径

かんたん

```
length( pos ) - 1.0
```

中心からの距離　　半径



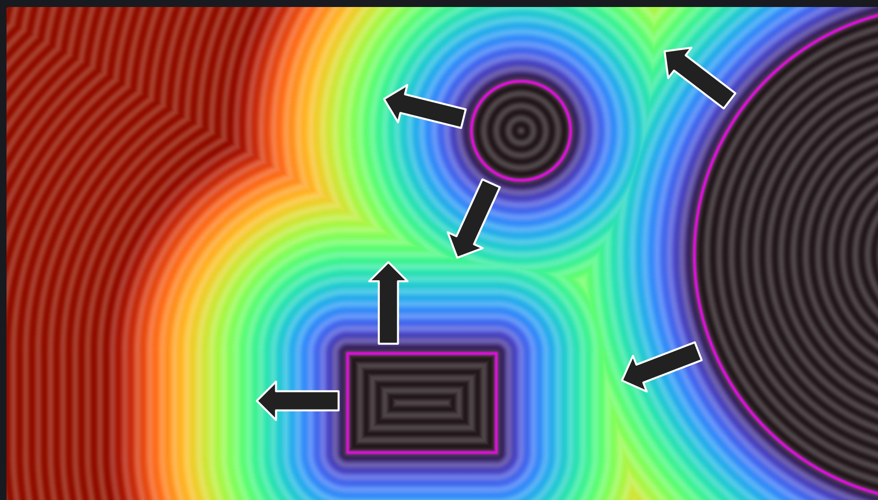


完全理解

白い球だと味気ないので

色を付けましょう

法線



物体の表面がどの向きを向いているか

法線を求める関数 normalFunc が
すでに定義されているので
それを使います

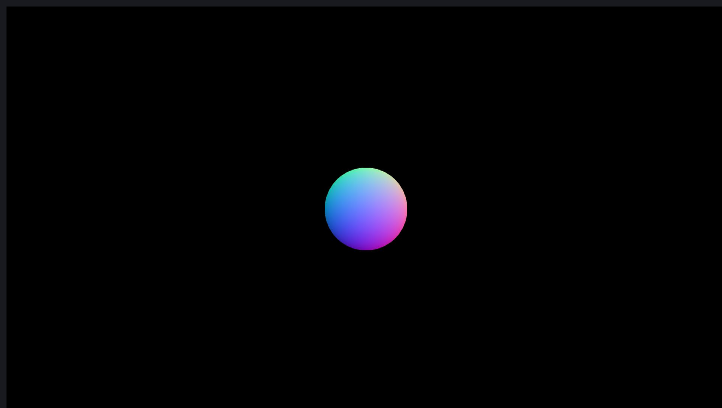
```
// 交差した場合、法線を可視化して描画する
vec3 normal = normalFunc( rayOri + t * rayDir );
outColor = vec4( 0.5 + 0.5 * normal, 1.0 );
```

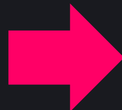
今回は詳しい求め方は端折ります

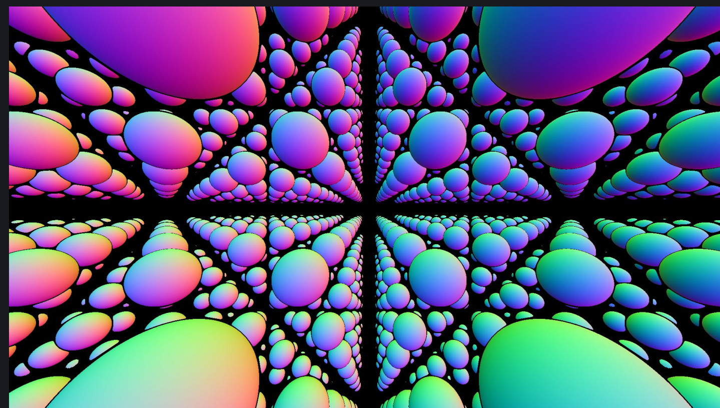
https://iquilezles.org/articles/normalsSDF/

こんな感じの虹色になります

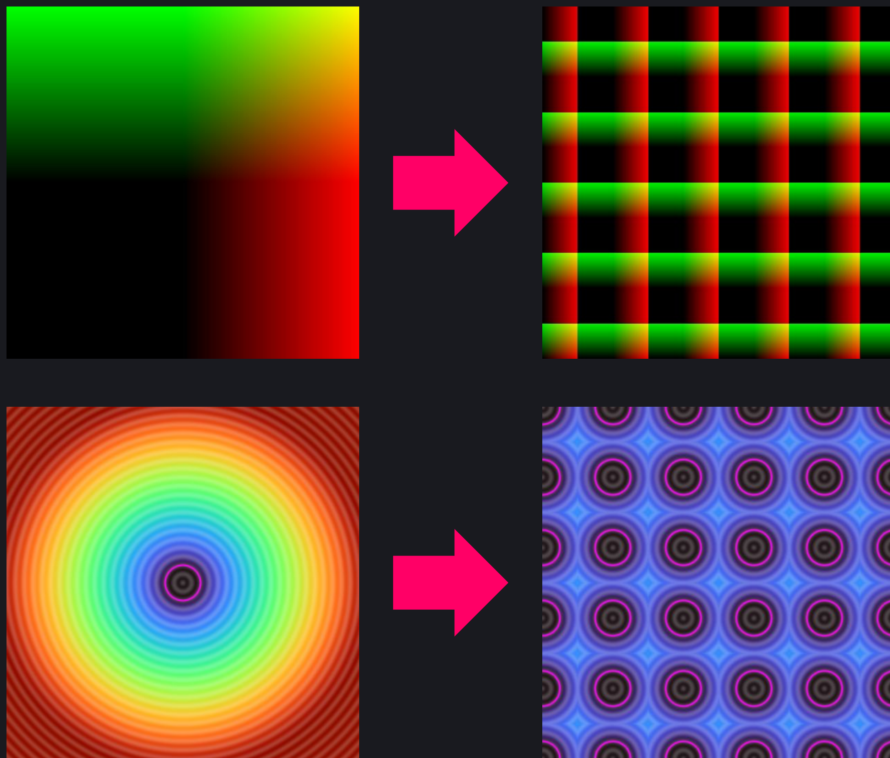
https://twigl.app/?ss=-NTym1RF1fQ11qgF-TdK







球体を増やしてみよう



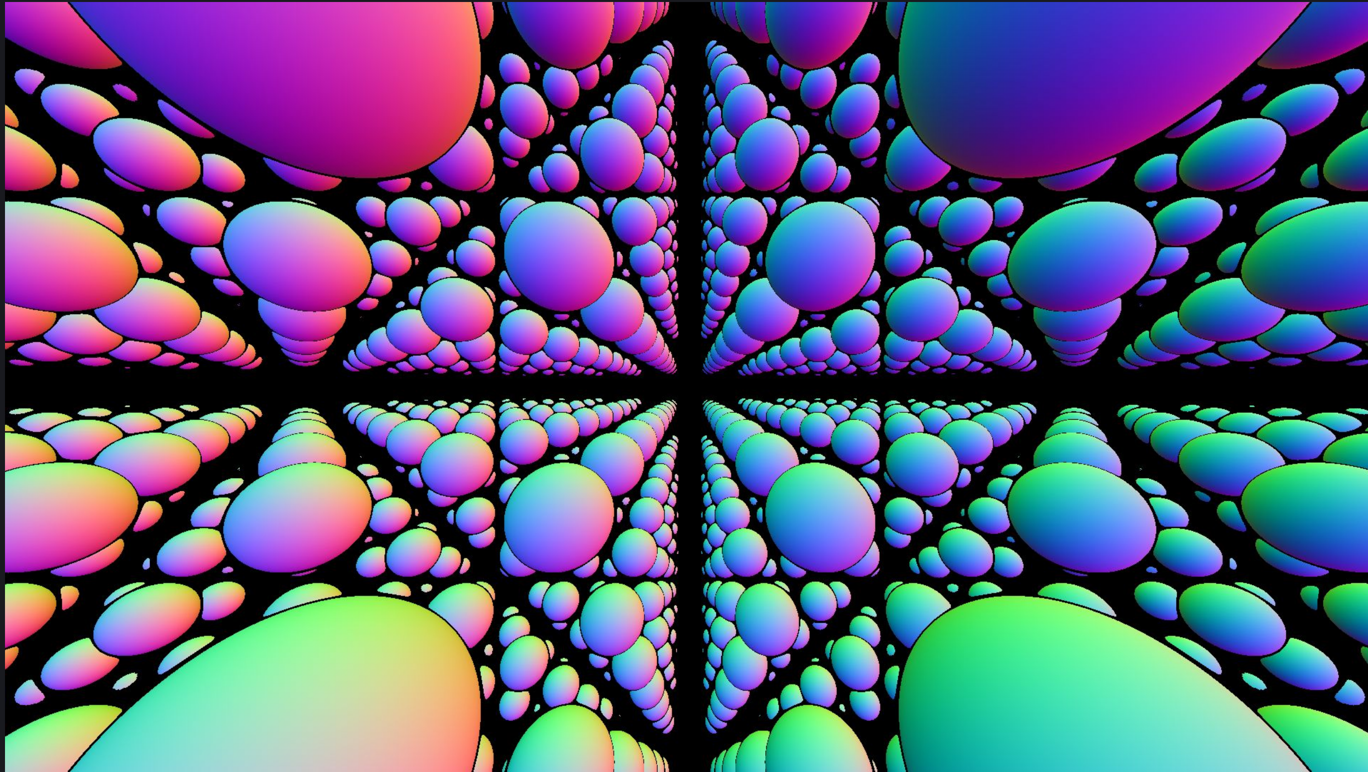
Repetition: 座標系を繰り返して球を増やします

距離関数内で座標系を繰り返す

```
pos = mod( pos, 4.0 ) - 2.0;
```

球体の距離関数を計算する前に
posをいじります

`mod( pos, 4.0 )` ... posの各成分を4で割ったあまりを計算する



増えた

https://twigl.app?ss=-NTymXep4JNbe0nyOKlu

もっといろいろな表現を
してみたい！

→ レイマーチング1から5

https://youtu.be/E0dsqlajCM4

END

[AD]

SESSIONS Shader Jam

セッションズ シェーダー ジャム

April 29
8am UTC
4月29日(土)
5pm JST

Aldroid
anticore
EmmazingGoose
Exca
gam0022
gaz
Ivan Dianov
Kali
Kamoshika
kostik
Musk
Provod
rimina
sp4ghet
totetmatt
visy