アニメーションと照明

2015 年度 情報可視化論

陰山 聡

2015.07.07

アニメーション

二つの方法

1. JavaScript の setInterval() を使う
2. requestAnimationFrame() を使う

requestAnimationFrame() は HTML DOM の window オブジェクトのメソッド。

setInterval

setInterval(codeToCall, timeoutInMilliSeconds)

使い方：

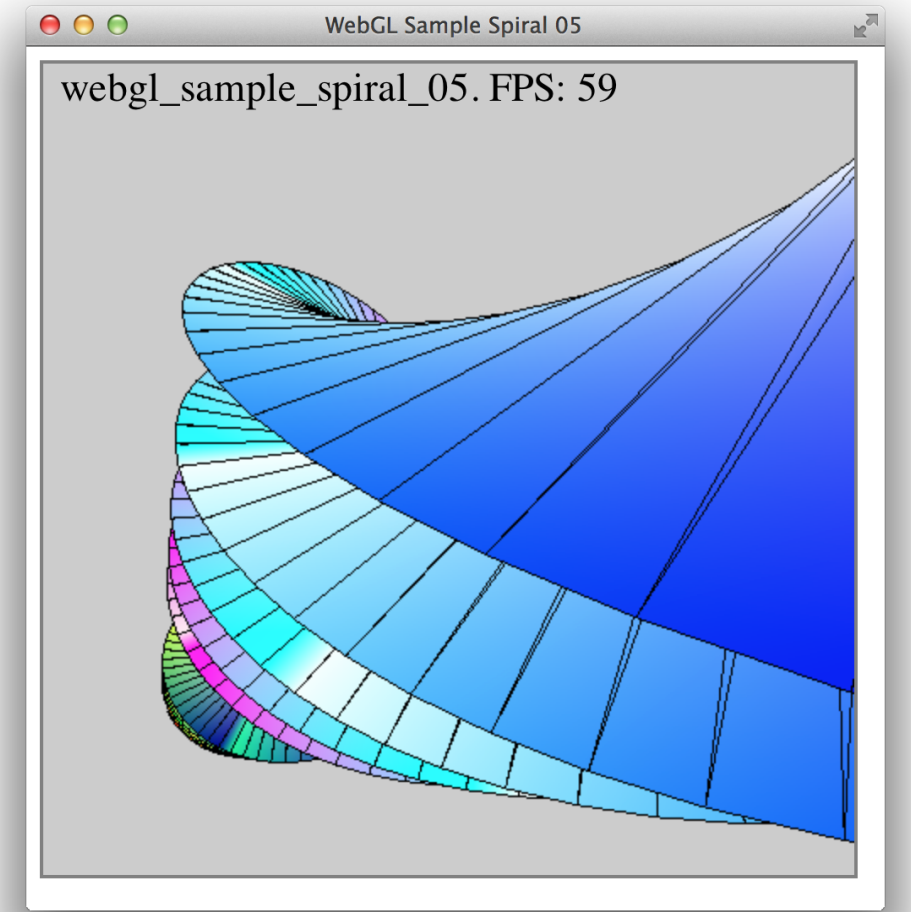
```
function draw() {
    . . .
}

function start() {
    . . .
    setInterval(draw, 16.7);
}
```

16.7 ミリ秒毎に draw() を呼ぶ。

$1/60 \sim 0.0167$

setInterval を使ったサンプルコード

webgl_sample_spiral_05.html

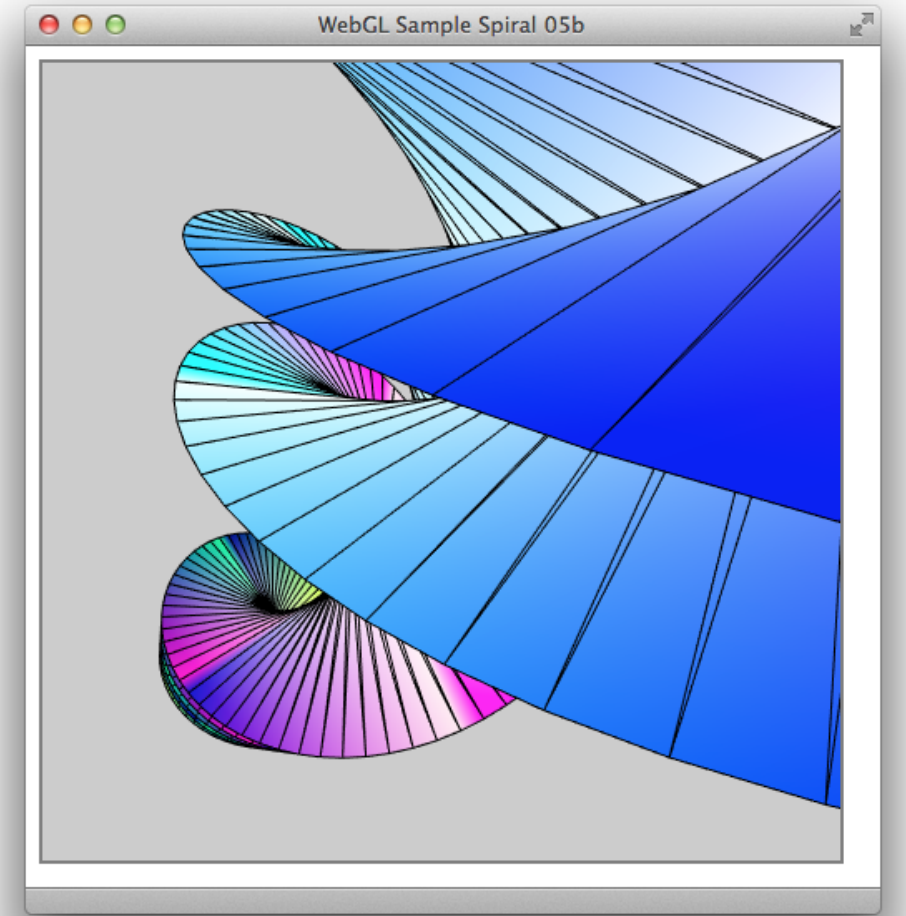


requestAnimationFrame

- シーンの更新タイミングをブラウザに任せる。
- こちらの方がよい。

requestAnimationFrame を使ったサンプルコード

webgl_sample_spiral_05b2.html



requestAnimationFrame の使い方

引数にブラウザから呼び出すコールバック関数を渡す。

requestAnimationFrame(draw);

指定したコールバック draw 関数が呼び出されるときには、引数として現在時刻（currentTime）が自動的に渡される。

つまり、draw() 関数は currentTime を引数として受け取るように定義する。

ただし、startup 関数から最初に draw() を呼び出すときには引数なしでコールする。（JavaScript では文法エラーにならない。）

requestAnimationFrame の使い方

```
function draw(currentTime) {
    requestAnimationFrame(draw);
    . . .
}

function startup() {
    . . .
    draw();
}
```

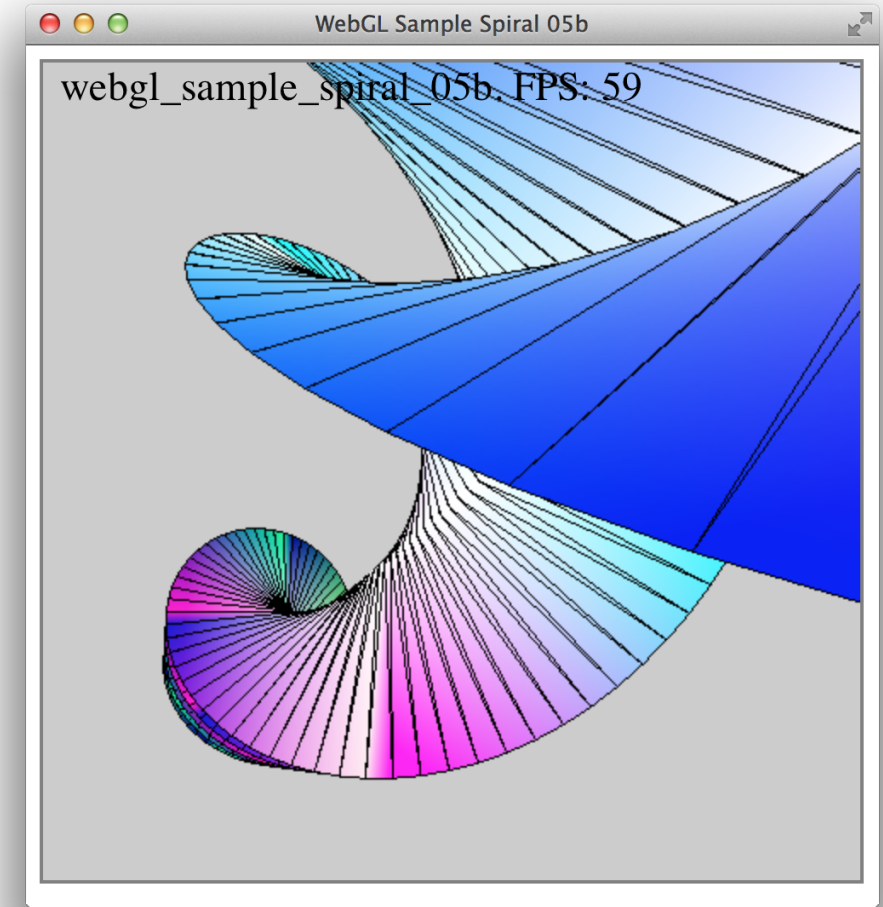
```
function draw(currentTime) {
  // 1. 現在のフレーム描画を開始する前に次のフレームを
  //    描画する新たな呼び出しを要求

  requestAnimationFrame(draw);

  // 2. シーンの中で動くオブジェクトの位置を更新
  ...
  // 3. シーンを描画
```

演習

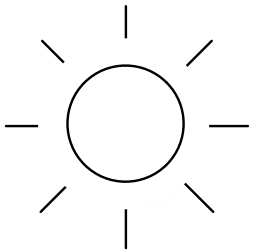
- webgl_sample_spiral_05b2.html を自由に変更し、アニメーションで遊んでみよう。

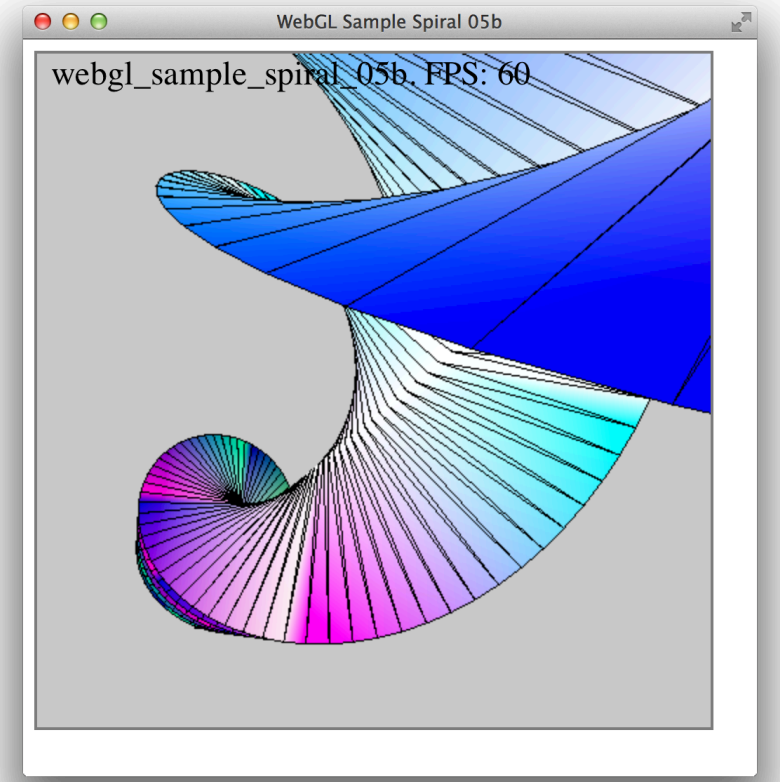
照明

- これまでは照明を全く考えていなかった
- 各頂点がそれぞれ指定された色で「光る」

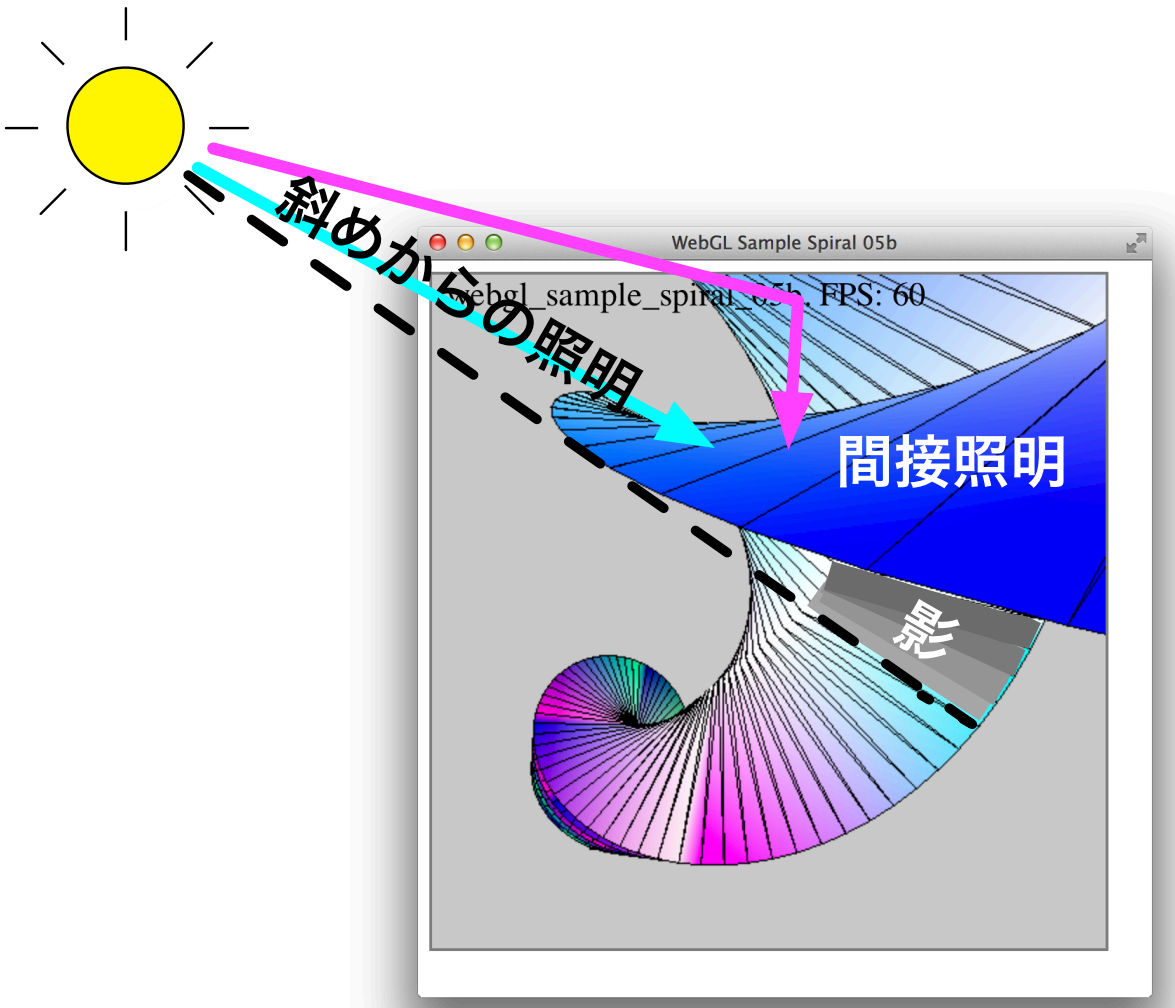


光源を置く





結構大変



二つの照明モデル

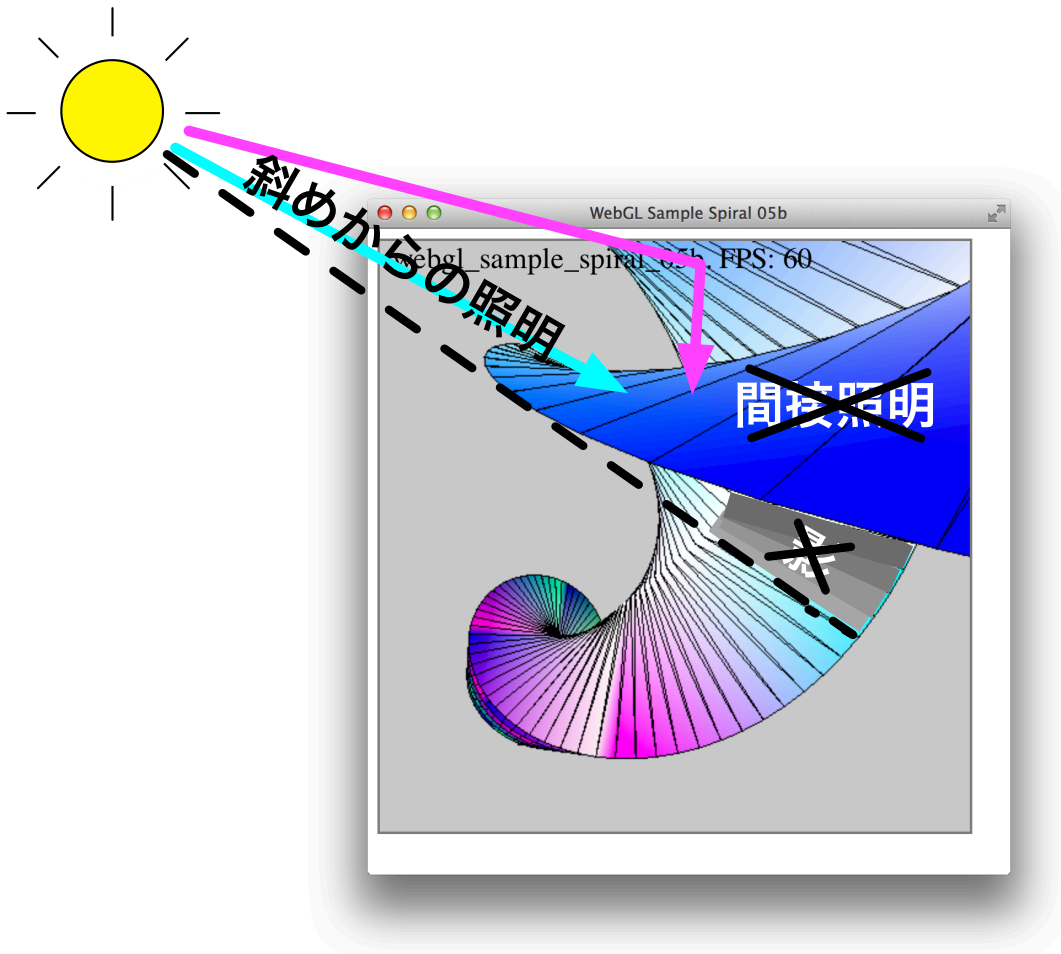
ある頂点の照明状態を計算するとき、

1. 大域照明モデル： 他の物体の存在を計算に入れる。
2. 局所照明モデル： 他の物体の存在を無視する 。

局所照明モデル

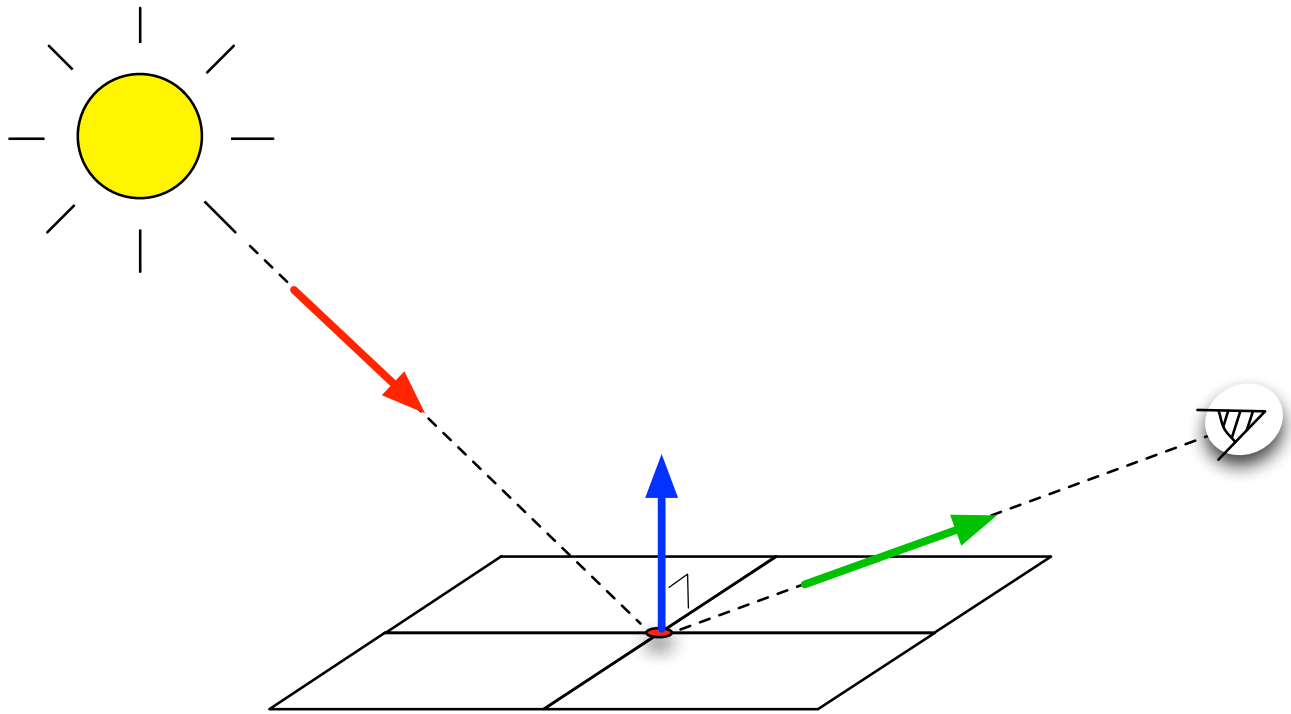
間接照明や影はできない。

間接照明は後述する環境光によって擬似的にその効果を入れる。



局所照明モデル

ある頂点での色（反射）は、3つの単位ベクトルの関数。



フォン（Phong）の反射モデル

Bui Tuong Phong (1942–1975)

- 局所モデル
- 3 種類の光を考える。
  - 環境光（Ambient Light）
  - 拡散光（Diffuse Light）
  - 鏡面光（Specular Light）
- それぞれが独立に物体を照らす。
- それぞれの光は独立の色（RGB）を持つ。

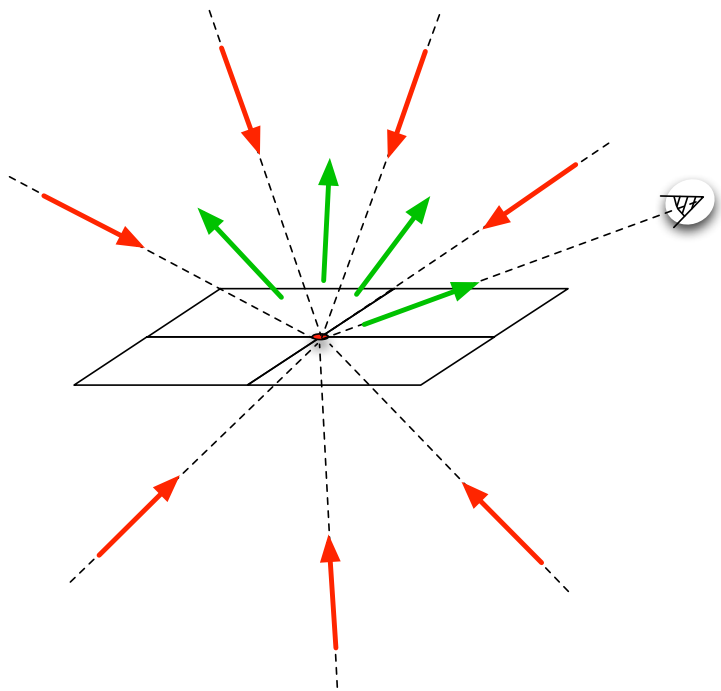
フォン（Phong）の反射モデル

物体の見え方（色や質感）の表現

- 3 つの色のそれぞれに対して頂点が独立に反射する。
  - 環境光反射（Ambient Reflection）
  - 拡散反射（Diffuse Reflection）
  - 鏡面反射（Specular Reflection）

環境光

- あらゆる方向からくる光による照明
- 面の裏側からも光が来る。
- 間接照明の模擬
- 弱めに入れる。例： $(0.2, 0.2, 0.2, 1.0)$



環境光のシェーダコード

バーテックスシェーダ

```
uniform vec3 uAmbientLightColor;
varying vec3 vLightWeighting;

vLightWeighting = uAmbientLightColor;
```

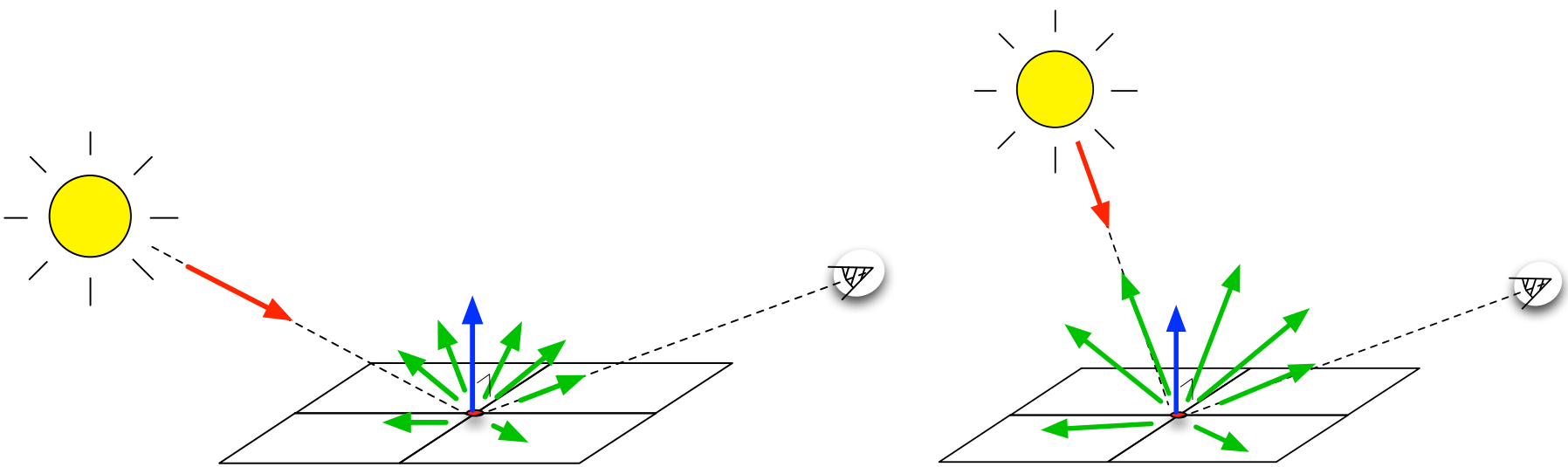
環境光のシェーダコード

フラグメントシェーダ

```
varying vec3 vLightWeighting;
vec4 texelColor = texture2D(uSampler, vTextureCoordinates);
gl_FragColor = vec4(vLightWeighting.rgb * texelColor.rgb,
texelColor.a);
```

拡散光

- 光の入射角に依存。
- 頂点の法線ベクトル $\boldsymbol{n}$ と、光の方向ベクトル $\boldsymbol{\ell}$ のなす角度で決まる。
- $\boldsymbol{n} \cdot \boldsymbol{\ell}$ の関数。
- カメラの方向 $\boldsymbol{e}$ には依存しない。



拡散光のシェーダコード

バーテックスシェーダ

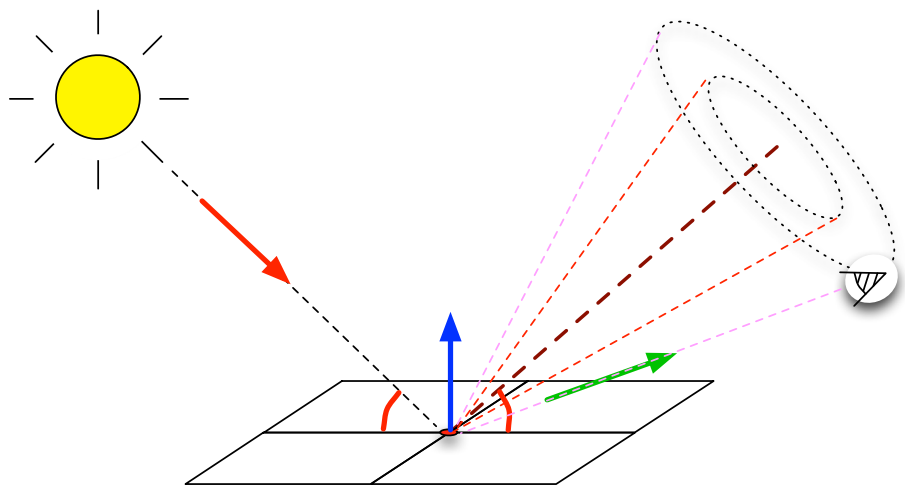
```
float diffuseLightWeightning = max(dot(normalEye,
     vectorToLightSource), 0.0);
vLightWeighting = uAmbientLightColor +
     uDiffuseLightColor * diffuseLightWeightning;
```

拡散光のシェーダコード

フラグメントシェーダ【変更なし】

鏡面光

- 金属のような光沢
- 頂点位置に鏡があるとして、入射する光が反射する方向にカメラがあるときに最も明るくなる。
- 頂点の法線ベクトル $\boldsymbol{n}$ と、光の方向ベクトル $\boldsymbol{\ell}$ と、カメラの方向 $\boldsymbol{e}$ の関数。
- 光沢度（鏡への近さ）を決めるパラメータを使う。反射する光がどれほど広がるか。



鏡面光のシェーダコード

バーテックスシェーダ

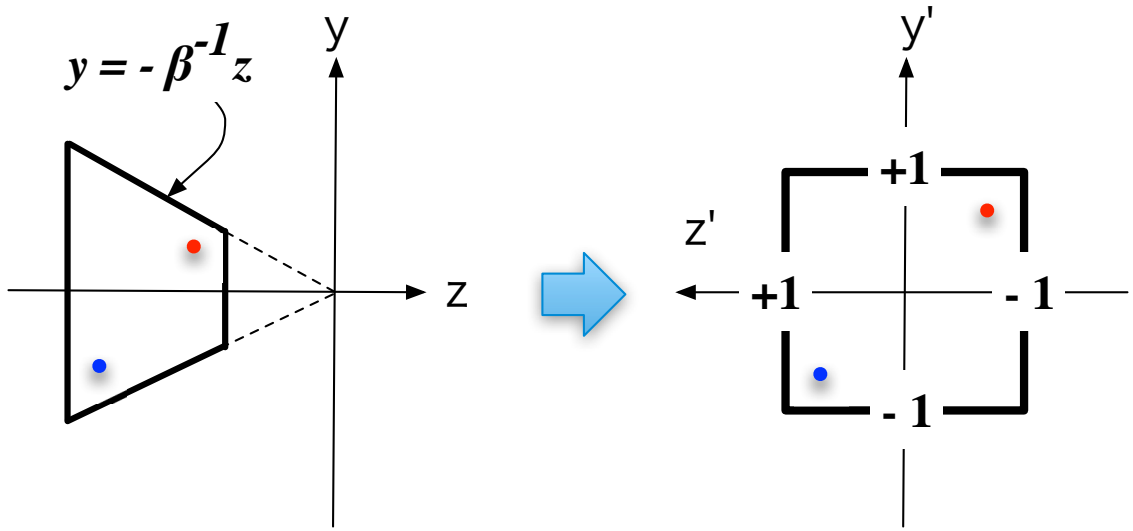
```
const float shininess = 32.0;
float rdotv = max(dot(reflectionVector, viewVectorEye), 0.0);
float specularLightWeightning = pow(rdotv, shininess);
vLightWeighting = uAmbientLightColor +
       uDiffuseLightColor * diffuseLightWeightning +
       uSpecularLightColor * specularLightWeightning;
```

鏡面光のシェーダコード

フラグメントシェーダ【変更なし】

復習

透視射影（以前の講義資料より）



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \\ c_1\, z + c_2 \end{pmatrix}$$

$\beta > 0, c_1, c_2$ は定数。x と y に関しては $-\beta/z$ 倍のスケール変換。

このスケール係数は、たとえば、視錐台の上面 $y = -\beta^{-1}\, z$ が $y' = +1$ に変換されることで容易に確認できる。

透視射影変換の二つのステップ（以前の講義資料より）

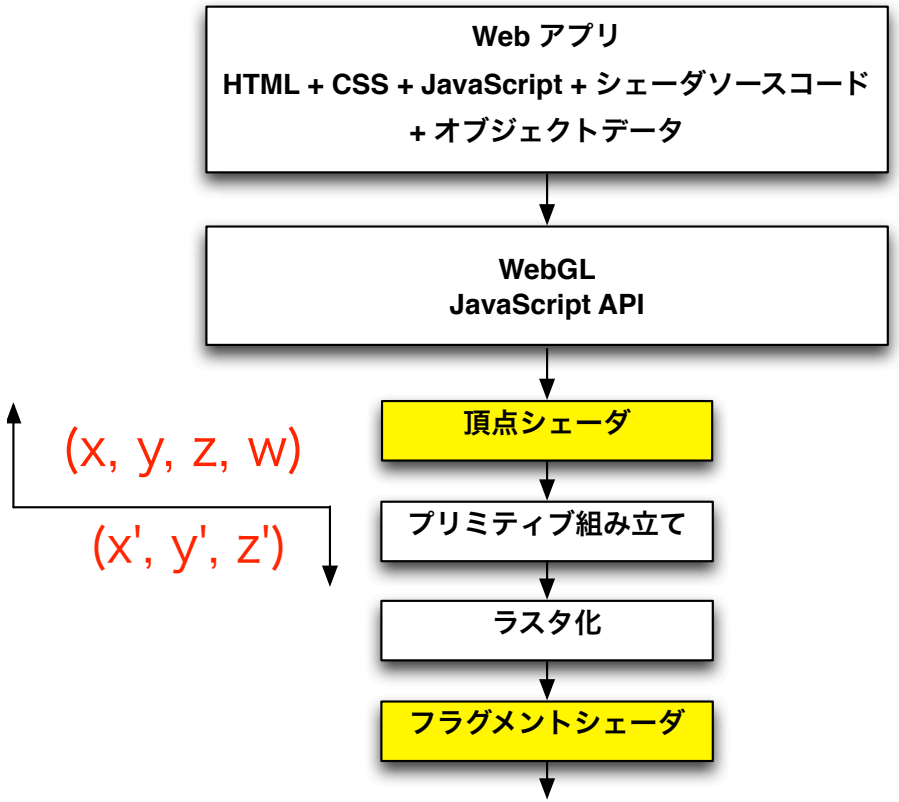
(1) 同次座標の行列演算（この行列は一意には決まらない。ここで挙げるのは一つの例。）

$$\begin{pmatrix} x^\dagger \\ y^\dagger \\ z^\dagger \\ w^\dagger \end{pmatrix} = \begin{pmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \alpha & \gamma \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(2) w 座標での割り算。これを透視除算という。プリミティブ組み立て時に実行される。

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x^\dagger/w^\dagger \\ y^\dagger/w^\dagger \\ z^\dagger/w^\dagger \end{pmatrix}$$

パイプラインでの座標の処理（以前の講義資料より）



- $(x', y', z') = (x/w, y/w, z/w)$
- GPU が自動的に実行

サンプルコード

サンプルコードを見る前に

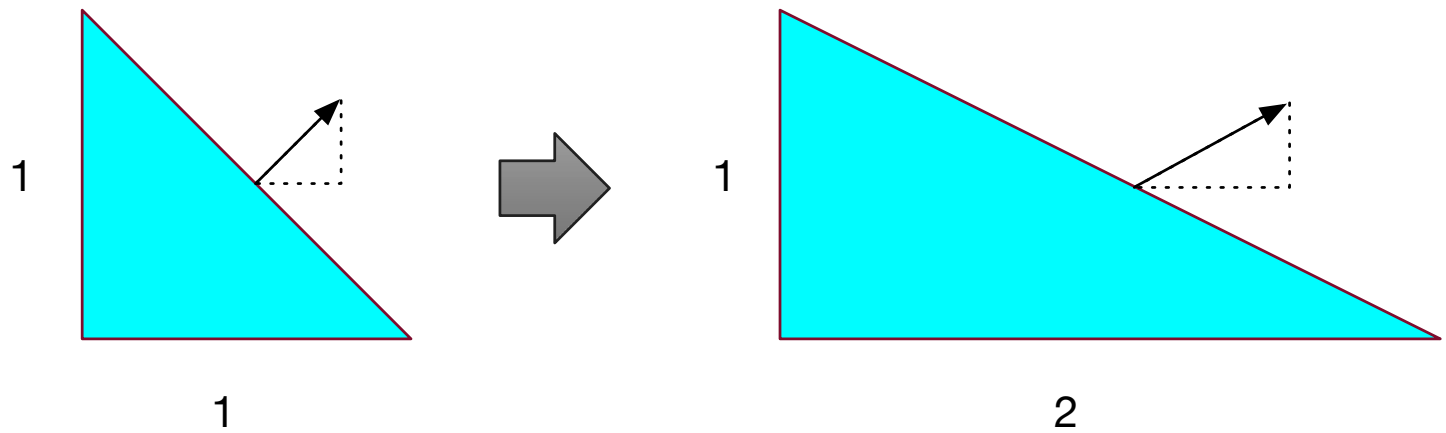
様々な方向ベクトルのシェーダでの計算方法

頂点から光源へ向かう単位ベクトル

バーテックスシェーダ

```
// Get the vertex position in eye coordinates
vec4 vertexPositionEye4 = uMVMatrix * vec4(aVertexPosition, 1.0);
vec3 vertexPositionEye3 = vertexPositionEye4.xyz /
vertexPositionEye4.w;

// Calculate the vector (l) to the light source
vec3 vectorToLightSource = normalize(uLightPosition -
vertexPositionEye3);
```

法線ベクトルの変換



法線ベクトルの変換

法線ベクトルの変換行列は、モデルビュー行列（の 3 行 3 列部分）の転置の逆である（証明は次のページ）。

```
// Transform the normal (n) to eye coordinates
vec3 normalEye = normalize(uNMatrix * aVertexNormal);
```

法線ベクトルの変換行列

ある点での面の法線ベクトルを $\boldsymbol{n}$, その点で面に接する任意のベクトルを $\boldsymbol{a}$ とする。

$$\boldsymbol{n} \cdot \boldsymbol{a} = \boldsymbol{n}^t \ \boldsymbol{a} = 0 \tag{1}$$

オブジェクトの変換行列を M、法線ベクトルの変換行列を N とすると、

$$\boldsymbol{a} \to \boldsymbol{a}' = M\boldsymbol{a}, \quad \boldsymbol{n} \to \boldsymbol{n}' = N\boldsymbol{n}$$

$\boldsymbol{n}'$ と $\boldsymbol{a}'$ が直交するから

$$0 = \boldsymbol{n}' \cdot \boldsymbol{a}' = (\boldsymbol{n}')^t \boldsymbol{a}' = (N\boldsymbol{n})^t \, (M\boldsymbol{a}) = \boldsymbol{n}^t N^t M \boldsymbol{a}$$

式 (1) より

$$N^t M = I$$

$$\therefore \quad N = (M^t)^{-1}$$

M が直交行列なら $N = M$.

反射方向ベクトル

バーテックスシェーダに reflect という関数が組み込まれている。

```
// Calculate the reflection vector (r) that is needed for
specular light
vec3 reflectionVector = normalize(reflect(-vectorToLightSource,
normalEye));
```

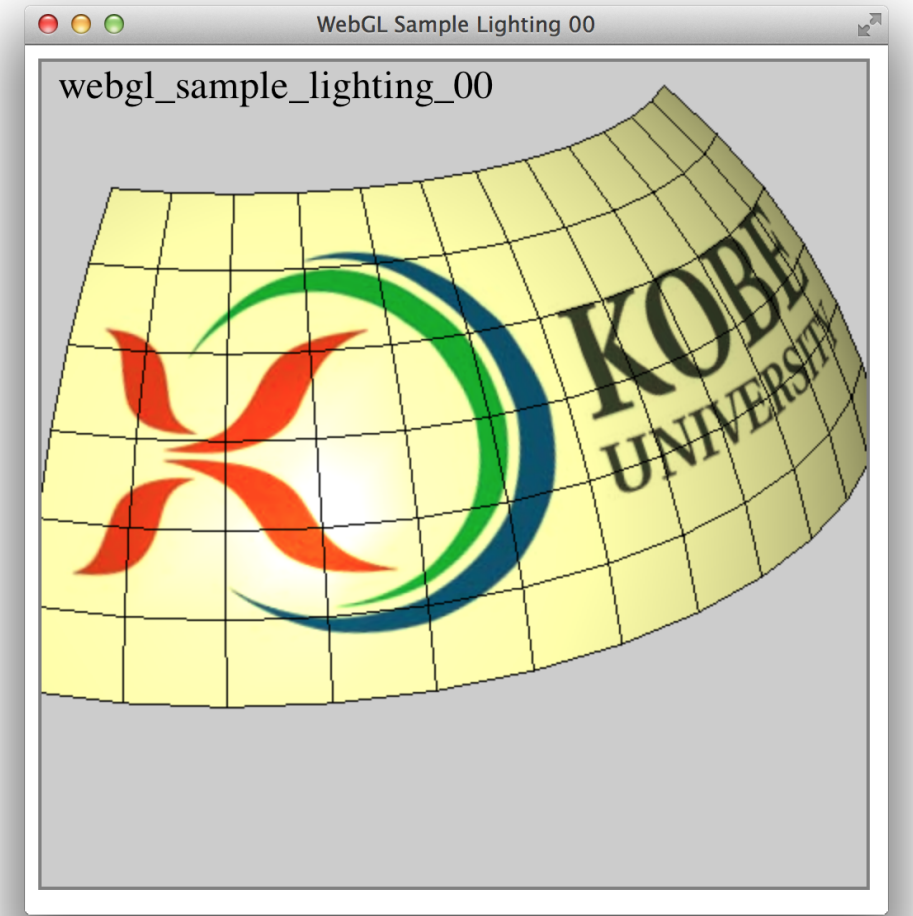
頂点からカメラ方向へのベクトル

原点（カメラ）からみた、頂点の位置ベクトルの逆。

```
vec3 viewVectorEye = -normalize(vertexPositionEye3);
```

サンプルコード 01

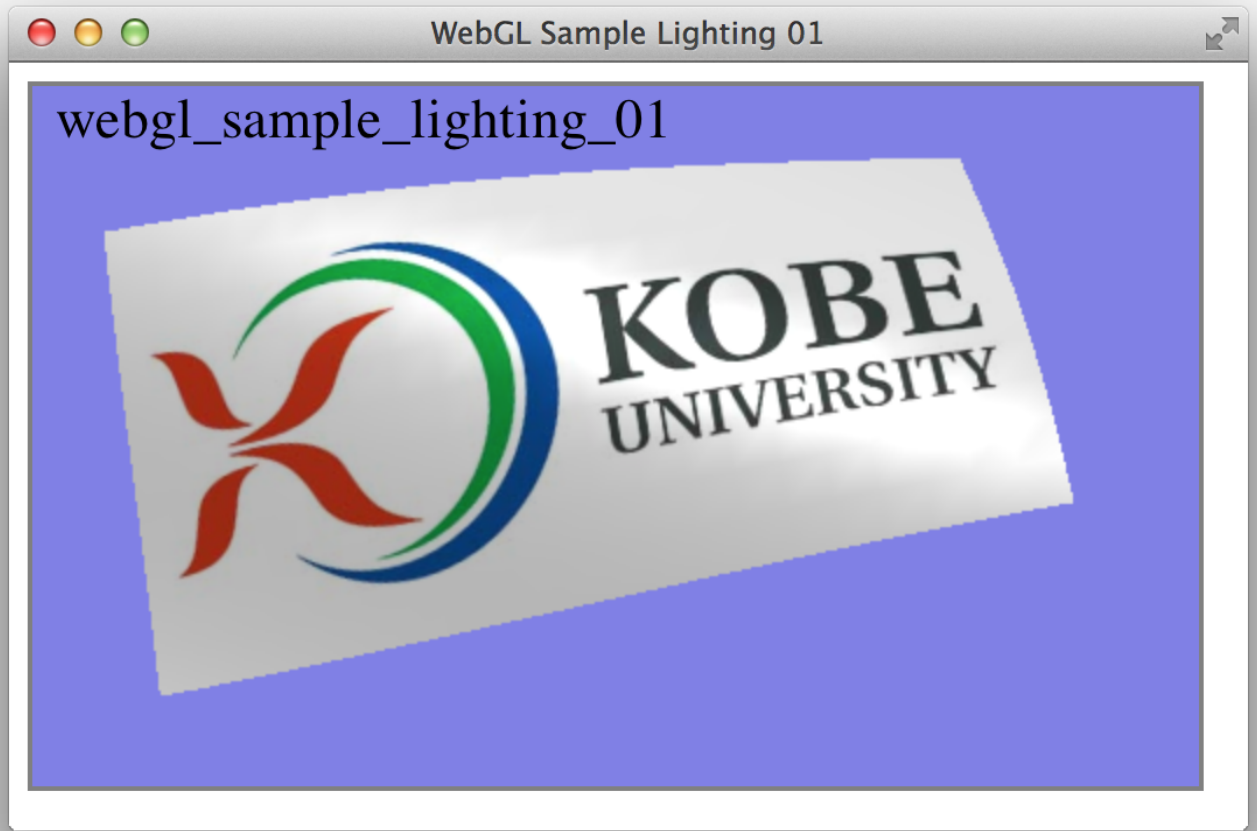
webgl_sample_lighting_00.html



サンプルコード 02

webgl_sample_lighting_01.html

頂点と法線の変形をバーテックスシェーダで計算している。



サンプルコード 03

webgl_spiral_kobe.html