リスト 9.4 | diffraction.frag

```
varying vec3 P;
varying vec3 N;
varying vec3 T;
uniform float roughX ;//粗さ係数
uniform float roughY ;//粗さ係数
uniform float blendFactor;//混合係数
uniform float GratingSpace ;//nano meter
const float ultraviolet = 380.0;
const float infrared = 780.0;

vec3 getRGB(float lambda)
{
  //正規化波長
  float x = (lambda - ultraviolet) / (infrared - ultraviolet);
  const float wR = 0.3, wB = 0.2, wV = 0.08;
  vec3 y = vec3(x) - vec3(0.7, 0.4, 0.2);
  vec3 col ;
  col.r = max((1.0 - y.r * y.r / (wR*wR)), 0.0);
  if(x < 0.4) col.g = max((1.0 - y.g * y.g / (wB*wB)), 0.0);
  else       col.g = max((1.0 - y.g * y.g / (wR*wR)), 0.0);
  col.b = max((1.0 - y.b * y.b / (wB*wB)), 0.0);
  //violetを加味
  col.r += max((1.0 - (x-0.08) * (x-0.08) / (wV*wV)), 0.0) * 0.7;
  return col;
}

void main(void)
{
  vec3 L = normalize(gl_LightSource[0].position.xyz - P);
  N = normalize(N);
  T = normalize(T);

  vec4 ambient = gl_FrontLightProduct[0].ambient;
  float dotNL = dot(N, L);
  vec3 V = normalize(-P);
  float dotNV = dot(N, V);
  vec3 H = normalize(L + V);
  float dotNH = max(dot(N, H), 0.0);
  vec4 diffuse = gl_FrontLightProduct[0].diffuse * max(0.0, dotNL);
  vec4 specular = vec4(0.0);
  if(dotNL > 0.0)
  {
    float d = dotNH * dotNH;
    float b = d * d;
    vec3 Q = normalize(H - dotNH * N);
    float dotTQ2 = dot(T, Q) * dot(T, Q);
    float c = ((d - 1.0) / d) * (dotTQ2 / (roughX*roughX) + (1.0 - dotTQ2) / (roughY
* roughY));
    //分布関数
    float D = min(1.0, exp(c) /(4.0 * 3.14 * roughX * roughY * b));
    //フレネル係数(垂直反射係数)
    vec3 F0 = gl_FrontLightProduct[0].specular.rgb;
    //Schlickの近似式
    vec3 F = F0 + (vec3(1.0) - F0) * pow(1.0 - dot(L, H), 5.0);
    //幾何減衰係数
    float dotVH = dot(V, H);
    float Gout = 2.0 * dotNH * dotNV / dotVH;
    float Gin = 2.0 * dotNH * dotNL / dotVH;
    float G = min(1.0, min(Gout, Gin));
    specular.rgb = D * F * G / dotNV;
  }

  //回折干渉による色の計算
```

```
  vec3 Q = normalize(H - dotNH * N);
  float dotTQ2 = dot(T, Q) * dot(T, Q);
  float a = sqrt(1.0- dotTQ2 * dotTQ2);//sinΦ
  float b = sqrt(1.0 - dotNL * dotNL);
  float c = sqrt(1.0 - dotNV * dotNV);
  float u = abs(b - c) ;
  vec4 colorInter = vec4(0.0);//干涉色
  for(int i = 1; i <= 5; i++)
  {
    if(a == 0.0) continue;
    float lambda = (GratingSpace / a) * u / float(i);
    if(lambda < ultraviolet || lambda > infrared) continue;
    colorInter.rgb += getRGB(lambda);
  }

  //統合
  if( colorInter.rgb == vec3(0.0) )
    gl_FragColor = ambient + diffuse + specular;
  else
    gl_FragColor = ambient + diffuse * mix(vec4(1.0), colorInter, blendFactor) +
specular;
}
```