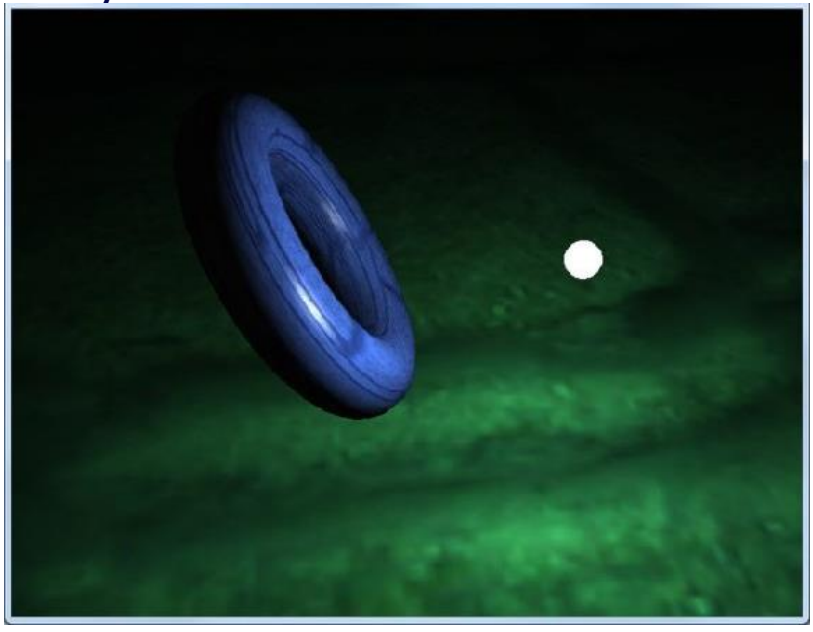
Шейдеры: расчёт освещения

Компьютерная графика

Содержание

- Освещение по модели Фонга. Точечный источник
- Направленный и прожекторный источники
- Другие модели освещения
  - Естественные модели освещения
  - Анизотропные модели освещения
  - Искусственные модели освещения

Что хотим получить?



Освещение по модели Фонга

Расчёт освещения

Повершинный (per-vertex lighting)

Попиксельный (per-pixel lighting)

Вершинный шейдер

- перевод координат вершины в мировую систему координат
- вычисление направления от вершины на источник освещения в мировой системе координат
- перевод координат вершины в однородные
- передача параметров во фрагментный шейдер

Параметры вершин из вершинного шейдера во фрагментный

- Нормаль к поверхности объекта в вершине (normal) в мировой системе координат
- Направление падающего света, от вершины к источнику освещения (light direction)
- Направление взгляда, от вершины к наблюдателю (view direction) в мировой системе координат
- Расстояние от точечного источника освещения до вершины (distance)

Вершинный шейдер

```
#version 330 core

#define VERT_POSITION 0
#define VERT_TEXCOORD 1
#define VERT_NORMAL   2

    layout(location = VERT_POSITION)   in vec3    position;
    layout(location = VERT_TEXCOORD)   in vec2    texcoord;
    layout(location = VERT_NORMAL)     in vec3    normal;
```

Вершинный шейдер

```
// параметры преобразований
uniform struct Transform {
 mat4   model;
 mat4   viewProjection;
 mat3   normal;
 vec3   viewPosition;
} transform;
// параметры точеченого источника освещения
uniform struct PointLight{
 vec4   position;
 vec4   ambient;
 vec4   diffuse;
 vec4   specular;
 vec3   attenuation;
} light;
```

Вершинный шейдер

```
// параметры для фрагментного шейдера
out Vertex {
    vec2    texcoord;
    vec3    normal;
    vec3    lightDir;
    vec3    viewDir;
    float   distance;
} Vert;
```

Вершинный шейдер

```
void main(void){
    //переведём координаты вершины в мировую систему координат
    vec4 vertex=transform.model*vec4(position,1.0);
    //направление от вершины на источник освещения
    //в мировой системе координат
    vec4 lightDir=light.position-vertex;
    //переводим координаты вершины в однородные
    gl_Position=transform.viewProjection*vertex;
    ...
}
```

Вершинный шейдер

```
void main(void){
    ...
    //передадим во фрагментный шейдер некоторые параметры
    //передаём текстурные координаты
    Vert.texcoord=texcoord;
    //передаём нормаль в мировой системе координат
    Vert.normal=transform.normal*normal;
    //передаём направление на источник освещения
    Vert.lightDir=vec3(lightDir);
    //передаём направление от вершины к наблюдателю
    //в мировой системе координат
    Vert.viewDir=transform.viewPosition-vec3(vertex);
    // передаём рассятоние от вершины до источника освещения
    Vert.distance=length(lightDir);
}
```

Фрагментный шейдер

```
#version 330 core

#define FRAG_OUTPUT0 0

layout(location = FRAG_OUTPUT0) out vec4 color;
```

Фрагментный шейдер

```
uniform struct PointLight{
    vec4    position;
    vec4    ambient;
    vec4    diffuse;
    vec4    specular;
    vec3    attenuation;
} light;
uniform struct Material {// параметры материала
sampler2D texture;
    vec4    ambient;
    vec4    diffuse;
    vec4    specular;
    vec4    emission;
    float   shininess;
} material;
```

Фрагментный шейдер

```
// параметры полученные из вершинного шейдера
in Vertex {
    vec2    texcoord;
    vec3    normal;
    vec3    lightDir;
    vec3    viewDir;
    float   distance;
} Vert;
```

Фрагментный шейдер

```
void main(void){
//нормализация полученных данных для коррекции интерполяции
    vec3 normal=normalize(Vert.normal);
    vec3 lightDir=normalize(Vert.lightDir);
    vec3 viewDir=normalize(Vert.viewDir);
...
}
```

Фрагментный шейдер

```
void main(void){
...
  //вычисление коэффициента затухания
  float attenuation=1.0/( light.attenuation[0]+light.attenuation[1]*
          Vert.distance+light.attenuation[2]*Vert.distance*Vert.distance);
...
}
```

Фрагментный шейдер

```
void main(void){
...
  //добавление собственного свечения материала
  color= material.emission;
  //добавление фонового освещения
  color+=material.ambient*light.ambient*attenuation;
  //добавление рассеянного света
  float Ndot=max(dot(normal,lightDir),0.0);
  color+=material.diffuse*light.diffuse*Ndot*attenuation;
...
}
```

Фрагментный шейдер - продолжение

```
void main(void){
 ...
 //добавление отраженного света
 float RdotVpow=max(pow(
 dot(reflect(-lightDir,normal),viewDir),material.shininess),0.0);
 color+=material.specular*light.specular*RdotVpow*attenuation
 ...
}
```

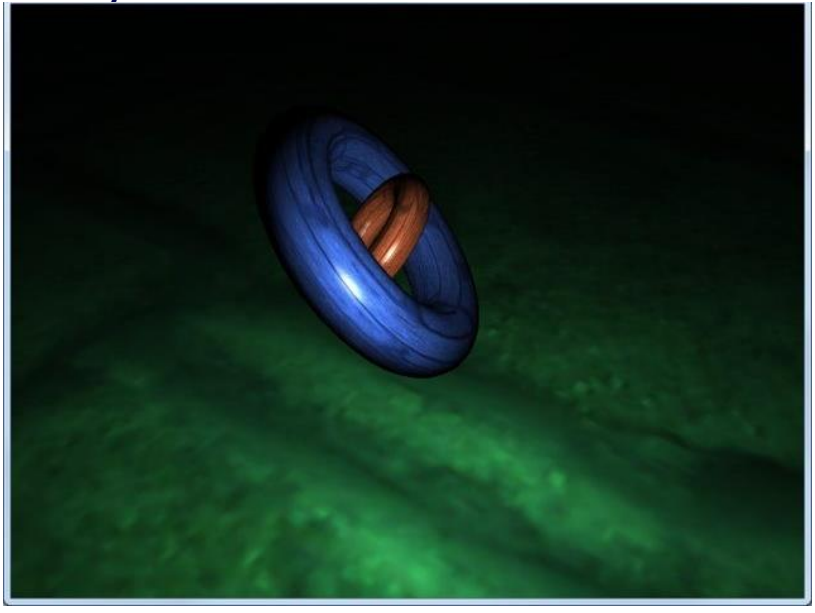
Фрагментный шейдер - продолжение

```
void main(void){
 ...
//или вычисление итогового цвета пикселя с учётом собственного цвета
color*=color_obj;
//или вычислим итоговый цвет пикселя на экране с учётом текстуры
color *=texture(material.texture,Vert.texcoord);
 ...
}
```

Содержание

- Освещение по модели Фонга. Точечный источник
- Направленный и прожекторный источники
- Другие модели освещения
  - Естественные модели освещения
  - Анизотропные модели освещения
  - Искусственные модели освещения

Что хотим получить?



Направленный источник освещения (directional light)

Направление в пространстве (direction) - с нулевой четвертой компонентой (w = 0)

Мощность фонового освещения (ambient)

Мощность рассеянного освещения (diffuse)

Мощность отраженного освещения (specular)

Изменения в вершинном шейдере

```
//передаём направление на источник освещения
Vert.lightDir=vec3(light.position);
//исключается расчёт параметра Vert.distance
//убирается расчёт параметра attenuation
```

Прожектор

Положение в пространстве (position)

Мощность фонового освещения (ambient)

Мощность рассеянного освещения (diffuse)

Мощность отраженного освещения (specular)

Коэффициенты затухания (attenuation)

Направление (spot direction)

Угол влияния (spot cutoff)

Коэффициент влияния (spot exponent)

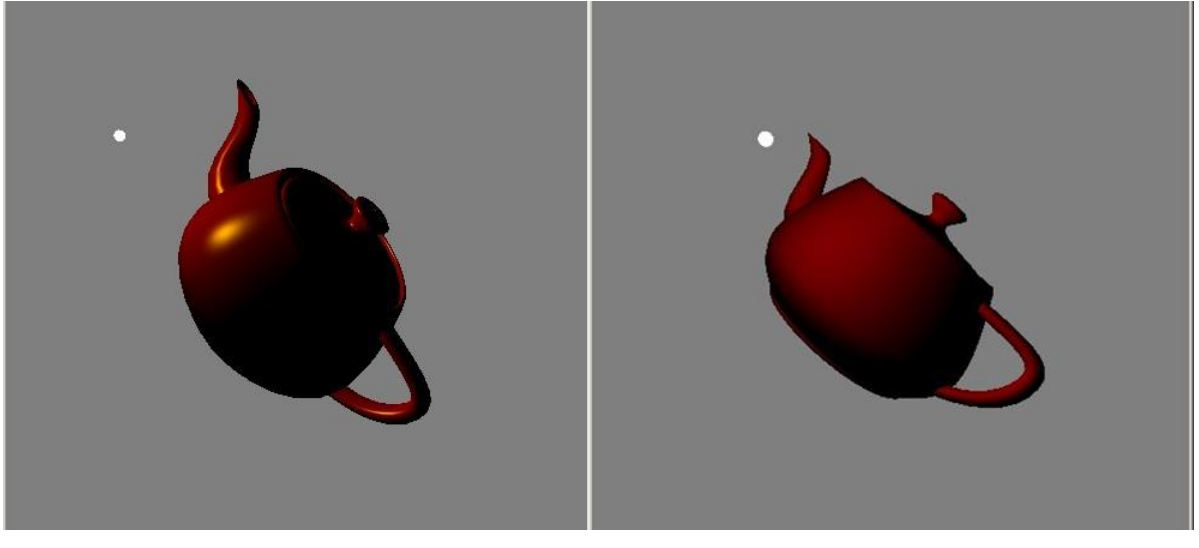
Изменения во фрагментном шейдере

```
//расчёт угла отклонения от направления прожектора до текущей точки
float spotEffect=dot(normalize(light.spotDirection),–lightDir);
//ограничим зону влияние прожектора
float spot=float(spotEffect>light.spotCosCutoff);
//экспоненциальное затухание к краям зоны влияния
spotEffect=max( pow (spotEffect,light. spot Exponent ),0.0)
//коэффициент затухания прожектора
float attenuation=spot*spotEffect/(light.attenuation[0]
   +light.attenuation[1]*Vert.distance+
      light.attenuation[2]*Vert.distance*Vert.distance);
```

Содержание

- Освещение по модели Фонга. Точечный источник
- Направленный и прожекторный источники
- Другие модели освещения
  - Естественные модели освещения
  - Анизотропные модели освещения
  - Искусственные модели освещения

Модель Фонга и модель Ламберта



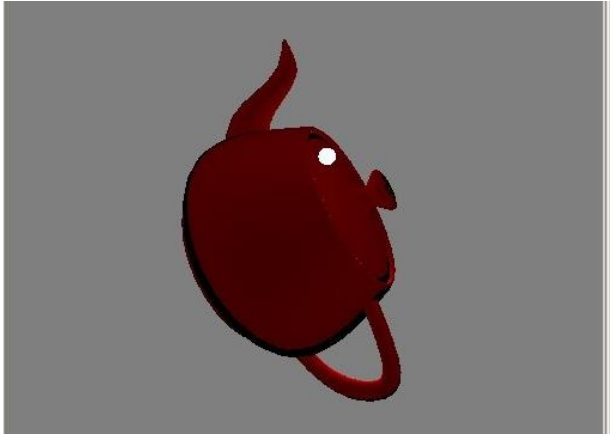
Lambert vertex shader

```
varying   vec3  l;
varying   vec3  n;
uniform   vec4  lightPos;
uniform   vec4  eyePos;

void main(void){
   // transformed point to world space
   vec3  p=vec3 ( gl_ModelViewMatrix * gl_Vertex ) ;
   // vector to light source
   l=normalize (vec3( lightPos )-p ) ;
   // transformed n
   n= normalize( gl_NormalMatrix  *  gl_Normal ) ;
   gl_Position=gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Lambert fragment shader

```
varying vec3 l;
varying vec3 n;

void main (void) {
   const vec4 diffColor=vec4(0.5,0.0,0.0,1.0);

   vec3 n2=normalize(n);
   vec3 l2=normalize(l);
   vec4 diff=diffColor*max(dot(n2,2), 0.0);

   gl_FragColor=diff;
}
```

Модель Орен-Найара - цвет шершавых поверхностей



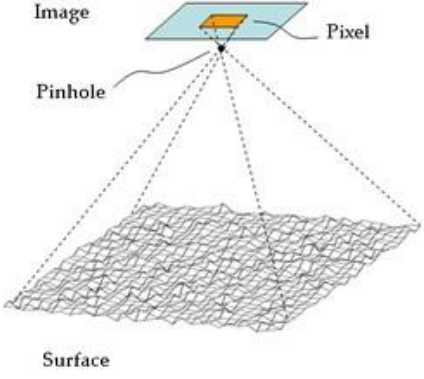
Модель Орен-Найара

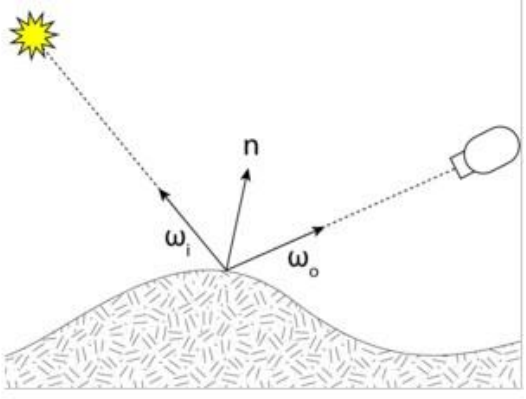
Поверхность состоит из множества микрограней (для каждой - модель Ламберта)

Угол между нормалью к микрограни и нормалью ко всей поверхности является случайной величиной

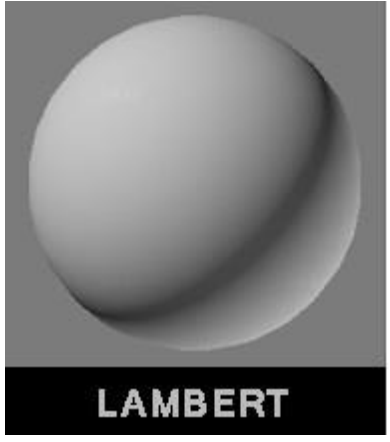
Модель учитывает взаимное закрывание и затенение микрограней и также учитывает взаимное отражение света между микрогранями.

Модель Орен-Найара

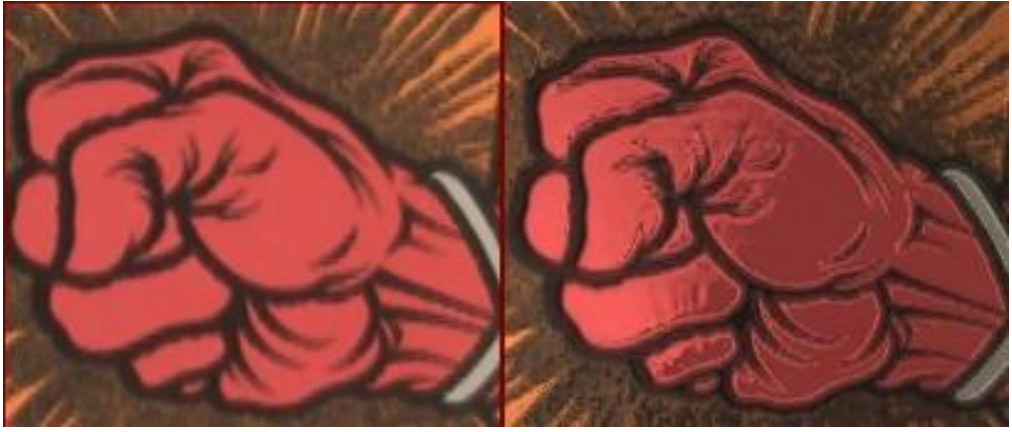




Модель Ламберта – Модель Орен-Найара



Модель Ламберта – Модель Орен-Найара



Модель Орен-Найара

Формула расчета диффузной модели освещения Орена – Наяра:

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \max[0, \cos(\phi_i - \phi_r)] \cdot \sin\alpha \cdot \tan\beta)) \cdot E_0$$

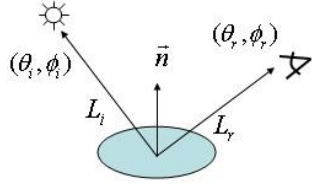
where

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.57},$$

$$B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09},$$

$$\alpha = \max(\theta_i, \theta_r),$$

$$\beta = \min(\theta_i, \theta_r),$$



and ρ is the albedo of the surface, and σ is the roughness of the surface. In the case of $\sigma = 0$ (i.e., all facets in the same plane), we have $A = 1$, and $B = 0$, and thus the Oren-Nayar model simplifies to the Lambertian model:

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot E_0$$

Альбедо — характеристика диффузной отражательной способности поверхности.

Модель Minnaert

Изначально модель создавалась для моделирования Луны (поэтому иногда можно встретить название «лунный шейдер») и планет

Подходит для других поверхностей, имеющих корпускулярную или губчатую поверхность

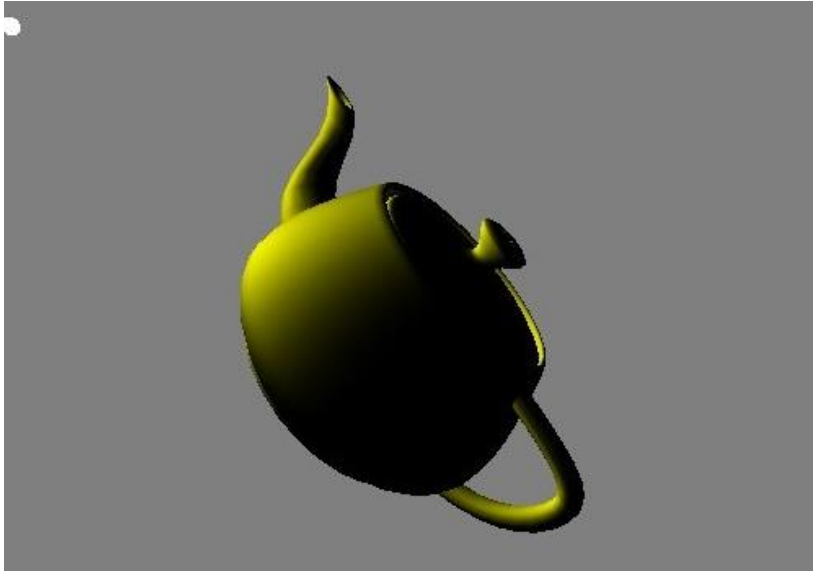
Хорошо подходит для моделирования видов ткани типа вельвета

$$I = (n,l)^{1+k}(1-(n,v))^{1-k}$$

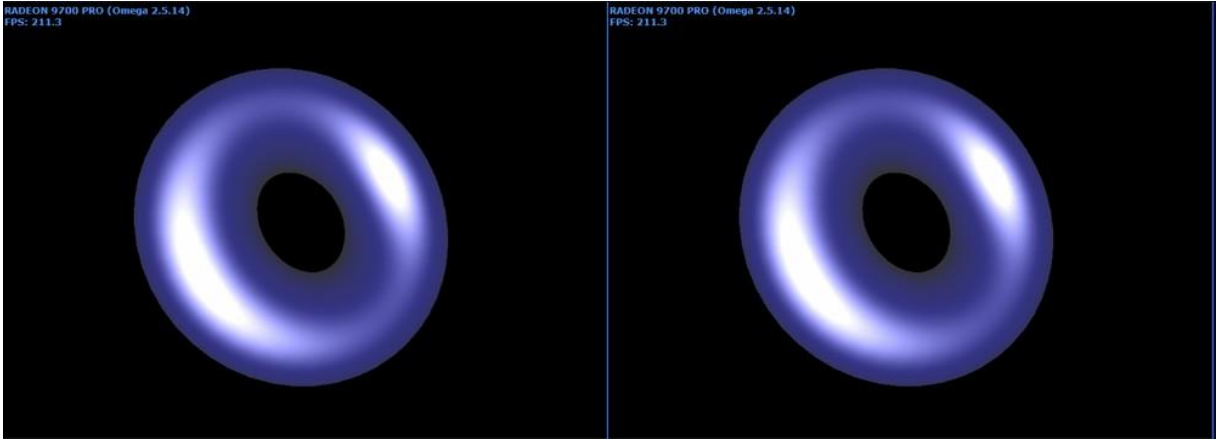
Minnaert fragment shader

```
const vec4 diffColor=vec4(1.0,1.0,0.0,1.0);
const float k = 0.8;


vec3  n2 = normalize (n);
vec3  l2 = normalize (l);
vec3  v2 = normalize (v);
float d1 = pow (max(dot(n2,l2),0.0),1.0+k);
float d2 = pow ( 1.0 - dot ( n2, v2 ), 1.0 - k );

gl_FragColor = diffColor * d1 * d2;
```

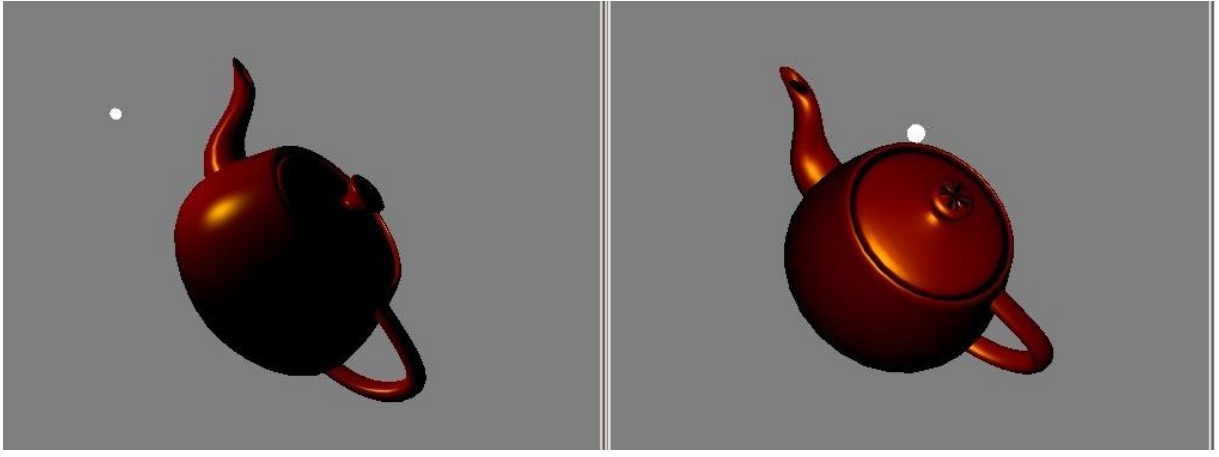
Модель Minnaert



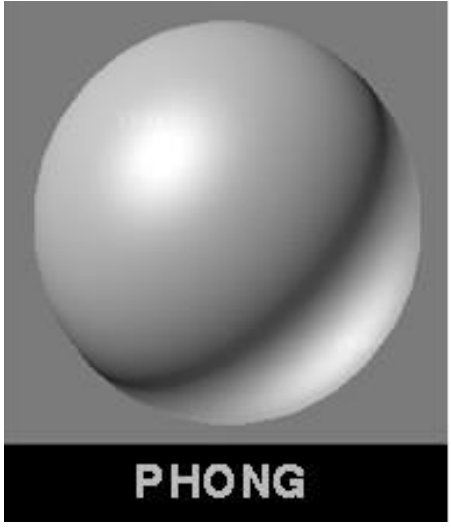
Фонг и Блинн

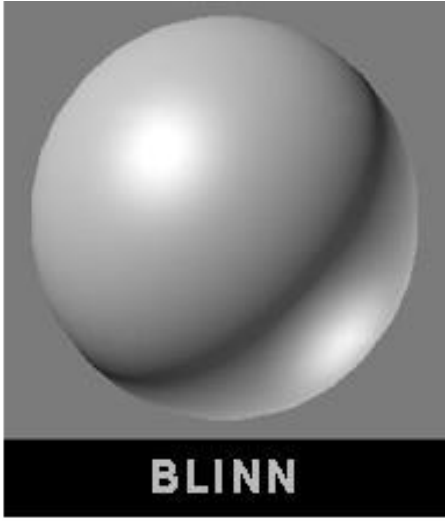

RADEON 9700 PRO (Omega 2.5.14)
FPS: 211.3
RADEON 9700 PRO (Omega 2.5.14)
FPS: 211.3

Фонг и Блинн



Фонг и Блинн





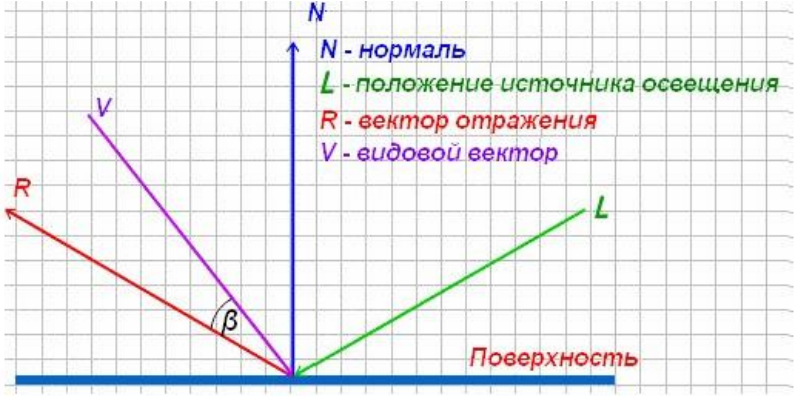
Модель Блинна-Фонга

Blinn (световой блик с малыми искажениями если свет падает под скользящим углом)

Blinn – это улучшенная версия Фонга. Блики по модели Блинна в сравнении с моделью Фонга лучше держат форму при разных углах падения света

Модель Фонга генерирует более растянутые блики под скользящими углами падения света, в то время как по модели Блинна форма бликов не меняется.

Фонг и Блинн



$$I_{specular}=k_s \times I_s \times (V \bullet R)^n \qquad I_{blin\_specular}=k_{b\_s} \times I_{b\_s} \times (N \bullet H)^n$$

Модель Блинна-Фонга

Упрощенный расчет зеркальной компоненты освещенности Джим Блинн ввел промежуточный вектор, который является средним значением между видовым вектором и вектором позиции источника освещения: $H=(L+V)/(|L+V|)$

Фрагментный шейдер

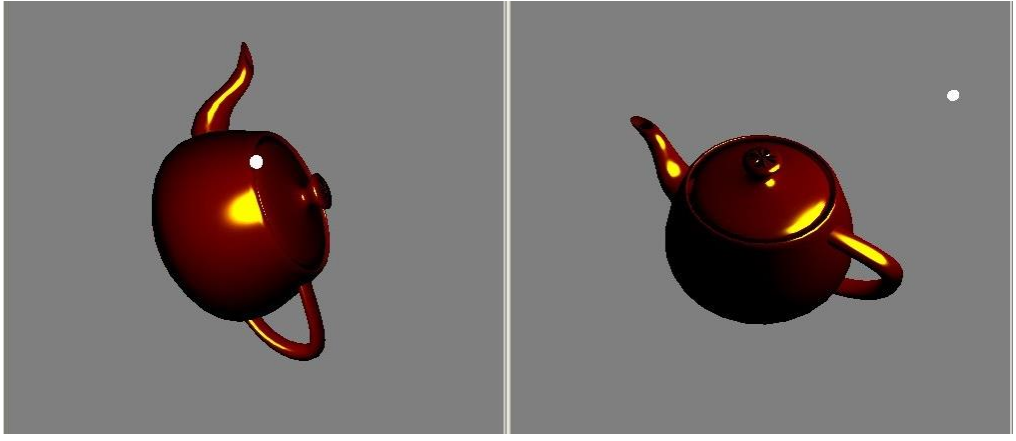
```
float3 H=normalize(light+view);
float n = 16;
return specular_color*specular_intensity*pow(dot(normal,H),n);
```

Ускорение вычисления яркости свечения

Шлик предложил замену степени n

```
float3 H = normalize(In.light+In.view);
float n = 16;
float D = dot(In.normal,H);
return specular_color*specular_intensity*D/((n-D)*(n+D))
```

Модель Кука-Торранса



Модель Кука-Торранса

Одна из наиболее согласованных с физикой

Поверхность состоит из множества микрограней, каждая из которых является идеальным зеркалом

Модель учитывает коэффициент Френеля и взаимозатенение микрограней

Модель Кука-Торранса

Количество отраженного света зависит от трех факторов:

Коэффициент Френеля(F)

Геометрическая составляющая, учитывающая самозатенение (G)

Компонент, учитывающий шероховатость поверхности (D)

Общая формула для вычисления отраженного света

$$K = \frac{F \cdot G \cdot D}{(\vec{V} \bullet \vec{N}) \cdot (\vec{L} \bullet \vec{N})}$$

Геометрическая составляющая

$$G = \min\left(1, \frac{2(\vec{H} \bullet \vec{N})(\vec{V} \bullet \vec{N})}{(\vec{V} \bullet \vec{H})}, \frac{2(\vec{H} \bullet \vec{N})(\vec{L} \bullet \vec{N})}{(\vec{V} \bullet \vec{H})}\right)$$

Компонент, учитывающий шероховатость поверхности

– это распределение микрограней поверхности. Обычно, для вычисления этого компонента используют распределение Бекмана:

$$D = \frac{1}{4m^2(\vec{H} \bullet \vec{N})^4} \cdot e^{\left(\frac{(\vec{H} \bullet \vec{N})^2 - 1}{m^2(\vec{H} \bullet \vec{N})^2}\right)}$$

где параметр m (от 0 до 1) определяет шероховатость поверхности. Чем он больше, тем поверхность шероховатее, следовательно, отражает свет даже под широкими углами.

Коэффициент Френеля

$$\text{fresnel\_}(\cos\Theta) := \left[\frac{2\cdot\eta^2\cdot\cos\Theta^2 - 2\cdot\left(1-\eta^2+\eta^2\cdot\cos\Theta^2\right)^{\frac{1}{2}}\cdot\cos\Theta\cdot\eta + 1 - \eta^2}{\eta^2 - 1} \right]^2$$

$$F1(\cos\Theta) := \frac{1}{(1+\cos\Theta)^{\lambda}}$$

$$F2(\cos\Theta) := (1-\cos\Theta)^{\lambda}\cdot(1-R_0) + R_0$$

Аппроксимация Шлика

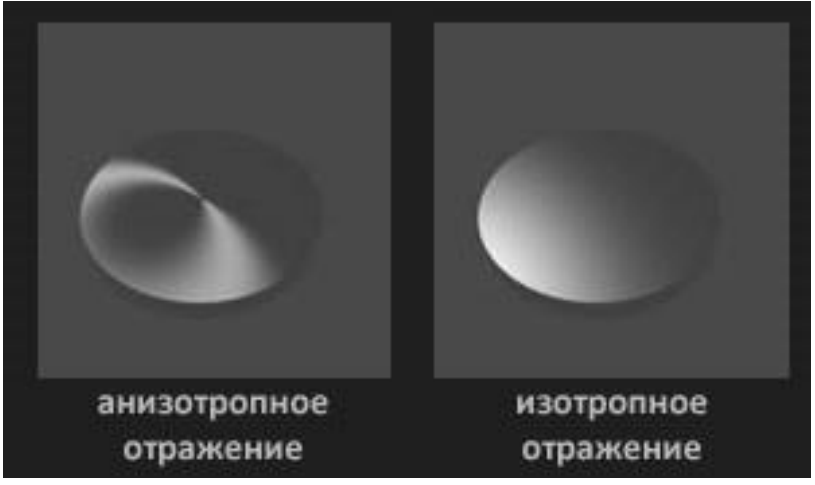
Металлический заяц - Модель Кука-Торранса

Зеркальные отражения могут быть:

Изотропными (isotropic)

Анизотропными (anisotropic)

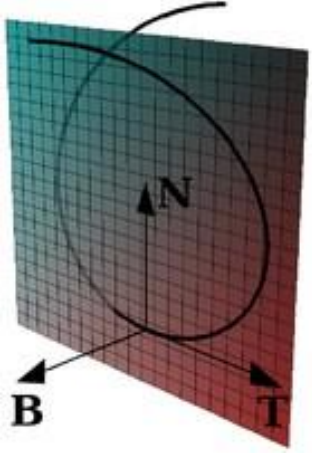
Анизотропные модели



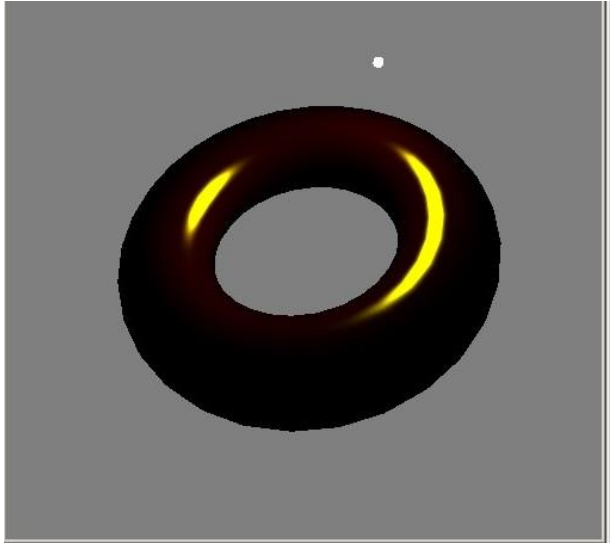
Анизотропные модели

Касательный базис

нормаль N
касательная T
бинормаль B



Модель Ashikhmin-Shirley



Модель Ashikhmin-Shirley

$$I_{diff} = \frac{28R_d}{23\pi}(1-R_s)\left(1-\left(1-\frac{(n,l)}{2}\right)^5\right)\left(1-\left(1-\frac{(n,v)}{2}\right)^5\right)$$

$$I_{spec} = \frac{\sqrt{(n_x+1)(n_y+1)}}{8\pi(h,l)max((n,l),(n,v))}(n,h)^{n_u\cos 2\phi + n_v sin 2\phi}F((h,l))$$

Ashikhmin-Shirley fragment shader

```
const  vec4    diffColor = vec4   (1.0, 0.0,  0.0, 1.0);
const  vec4    specColor = vec4   (0.7, 0.7,  0.0, 1.0);
const  float   PI= 3.1415926;
const  float   specPower = 30.0;

vec3   n2    =  normalize( n );
vec3   t2    =  normalize( t );
vec3   b2    =  normalize( b );
vec3   l2    =  normalize( l );
vec3   h2    =  normalize( h );
vec3   v2    =  normalize( v );
```

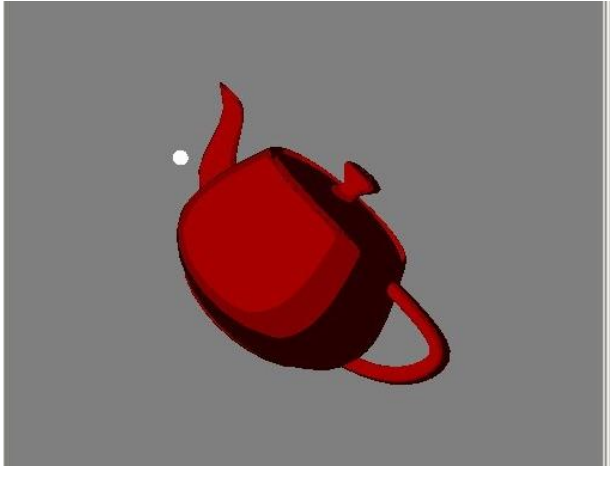
Ashikhmin-Shirley fragment shader

```
float nv  = max ( 0.0, dot ( n2, v2 ) );
float nl  = max ( 0.0, dot ( n2, l2 ) );
float nh  = max ( 0.0, dot ( n2, h2 ) );
float hl  = max ( 0.0, dot ( h2, l2 ) );
float t1h = dot ( b2, h );
float t2h = dot ( t2, h );
```

Ashikhmin-Shirley fragment shader

```
// calculate diffuse
float rd = (28.0/(23.0 ∗ PI)) ∗ (1.0 − pow(1.0 − 0.5 ∗ nv , 5.0))∗ (1.0 − pow(1.0 − 0.5 ∗ nl, 5.0));


 // calculate specular
 float B =  pow(nh, (mx ∗ t1h ∗ t1h + my ∗ t2h ∗ t2h )/( 1.0 − nh ∗ nh));

  float F =  (r0 + (1.0 − r0) ∗ pow(1.0 − hl, 5.0)) /(hl∗max (nv,nl));

  float rs = A∗B∗F;
  gl_FragColor = nl ∗ (diffColor ∗ kd∗ (1.0 − ks) ∗ rd + specColor ∗ ks ∗ rs) ;
```

Модель Toon shading



Фрагментный шейдер

```
void main( void ) {
...
vec3   n2 = normalize(n);
vec3   l2 = normalize(l);
float  diff = 0.2 + max(dot(n2,l2),0.0);

 if (diff < 0.4)
    clr = diffColor∗0.3;
 else if (diff < 0.7)
    clr = diffColor;
 else
   clr = diffColor∗1.3;
 gl_FragColor = clr;
}
```

Модель Ами Гуч

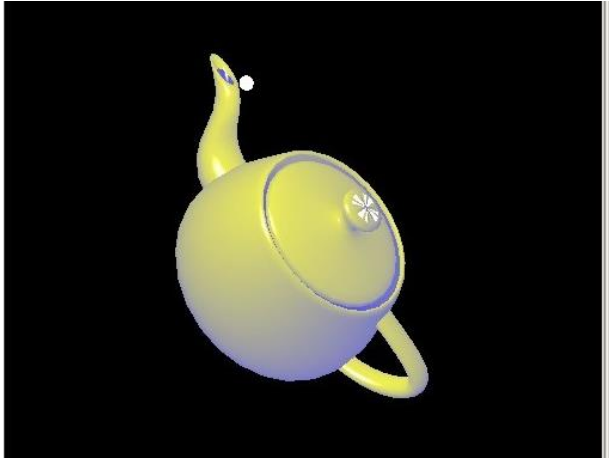
Модель Ами Гуч

Диффузная освещенность из модели Ламберта используется для смешивания двух оттенков – "теплого"и "холодного"

Модель Гуч позволяет более четко отображать контуры объекта.

$$I = \left(\frac{1 + \hat{\mathbf{l}} \cdot \hat{\mathbf{n}}}{2}\right) k_{cool} + \left(1 - \frac{1 + \hat{\mathbf{l}} \cdot \hat{\mathbf{n}}}{2}\right) k_{warm}$$

Модель Ами Гуч



Фрагментный шейдер

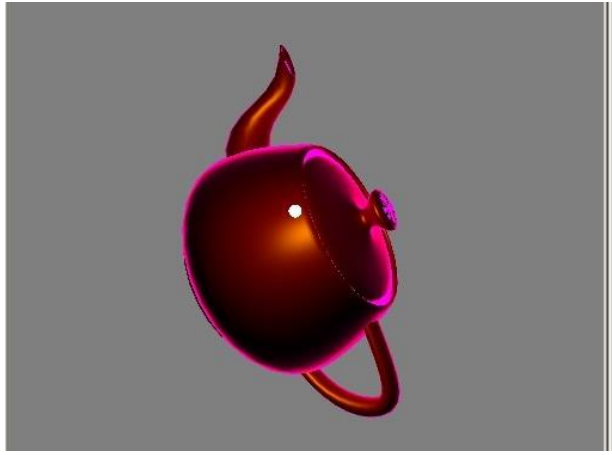
```
void main (void){
    ...
    vec3 kCool=min(coolColor + diffuseCool * surfaceColor,1.0);
    vec3 kWarm=min(warmColor + diffuseWarm * surfaceColor,1.0);
    vec3 kFinal = mix(kCool, kWarm, NdotL);
    vec3 r = normalize (reflectVec);
    vec3 v = normalize(viewVec);
    float spec = pow(max(dot(r,v),0.0),32.0);
    gl_FragColor = vec4(min(kFinal+spec, 1.0),1.0) ;
}
```

Модель Rim

Стандартная модель освещения с добавлением подсветки краев, где вектор нормали перпендикулярен вектору на наблюдателя

$$I = (1 + b - \max(0, (n, v)))^p$$

Модель Rim



Фрагментный шейдер

```
void   main  ( void ) {
     . . .
     vec4 diff = diffColor ∗ max(dot(n2,l2),0.0);
     vec4 spec = specColor ∗ pow(max(dot(n2,h2),0.0), specPower);
     float rim = pow(1 .0 + bias − max(dot(n2,v2),0.0), rimPower);
     gl_FragColor = diff + rim ∗ vec4(0.5,0.0,0.2,1.0) + spec ∗ specColor;
}
```

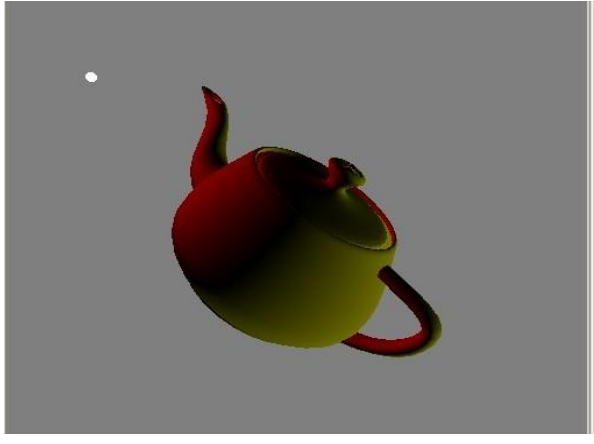
Модель Bidirectional Lighting

Использует интерполяцию между двумя цветами

Ее можно рассматривать как освещение сразу с двух противоположных сторон сторон источниками с разными цветами

$$I = C_0 \max(0,(n,l)) + C_2 \max(0,(-n,l))$$

Модель Bidirectional Lighting



Фрагментный шейдер

```
void main (void){
  const vec4 color0 = vec4(0.5, 0.0, 0.0, 1.0);
  const vec4 color2 = vec4(0.5, 0.5, 0.0,1.0);
  vec3 n2 = normalize(n) ;
  vec3 l2 = normalize(l) ;
   vec4 diff = color0*max(dot(n2,l2),0.0) + color2*max(dot(n2,-l2),0.0);
  gl_FragColor=diff;
}
```

Ссылки

http://steps3d.narod.ru/tutorials/lighting-tutorial.html