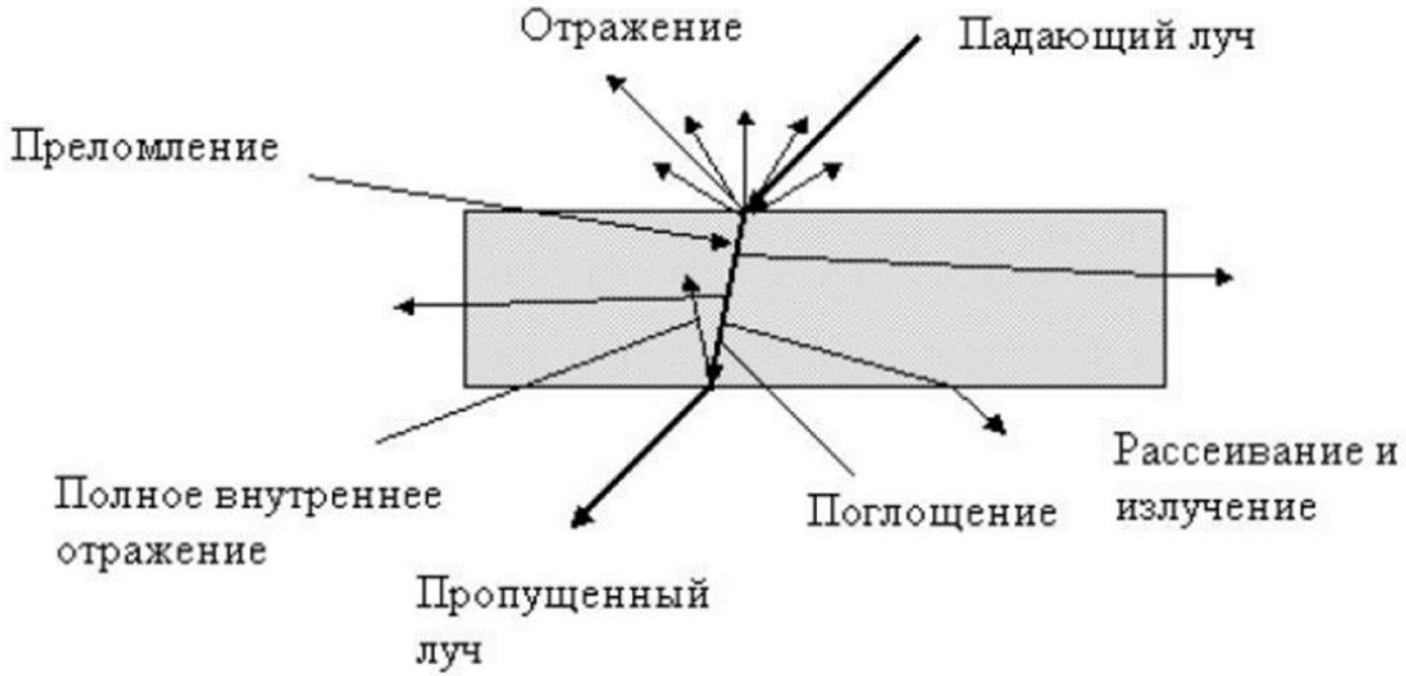
Компьютерная графика
WebGL

Лекция 3

Демяненко Я.М. ЮФУ 2024

Взаимодействие света с поверхностью

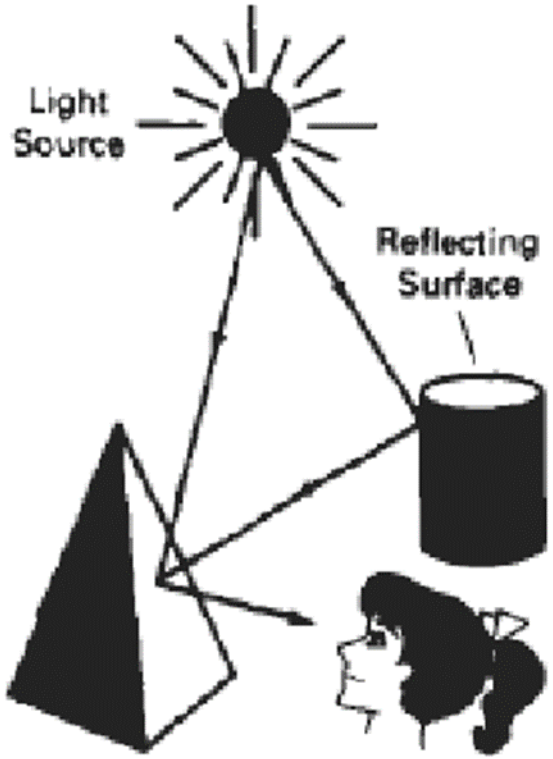


Источники света: первичные и вторичные

- Две категории:
  - светоиспускающие источники = первичные источники
  - светотражающие источники = вторичные источники

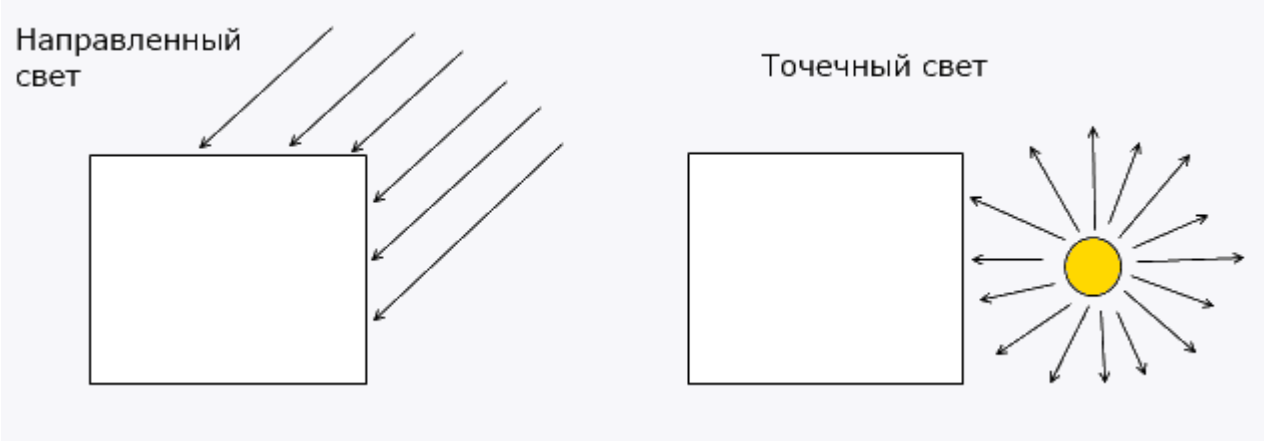
- Поверхность, не освещаемая первичным источником света, всё равно может быть видима!

- Полный отраженный свет = сумма вклада от источников света и других поверхностей сцены



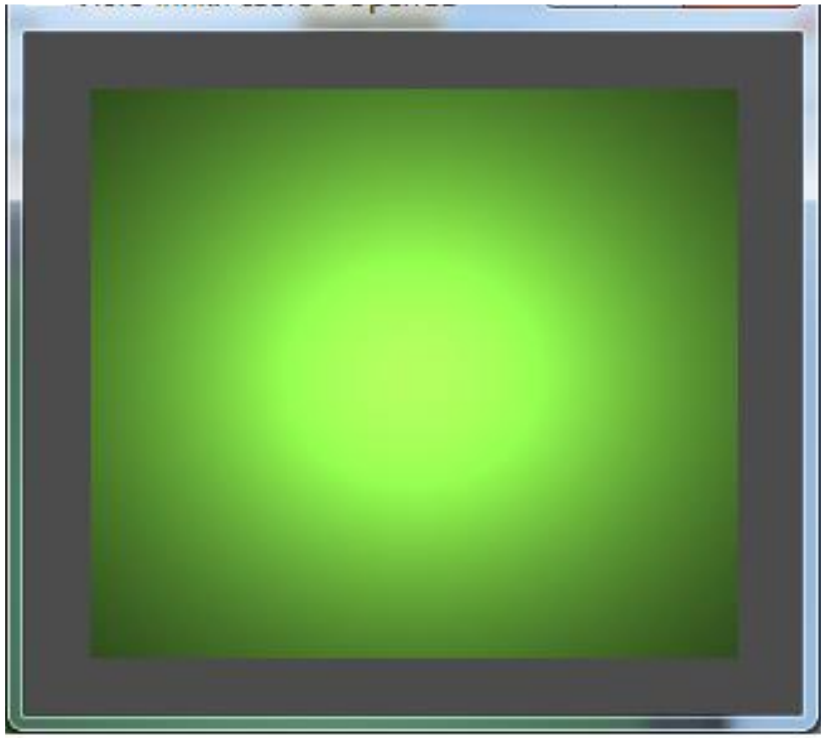
Типы освещения

- **Ambient light**: окружающее естественное освещение, рассеянный свет
- **Directional light**: направленный свет, например, солнечный свет
- **Point light**: точечный свет, например, свет лампы
- **Прожектор**



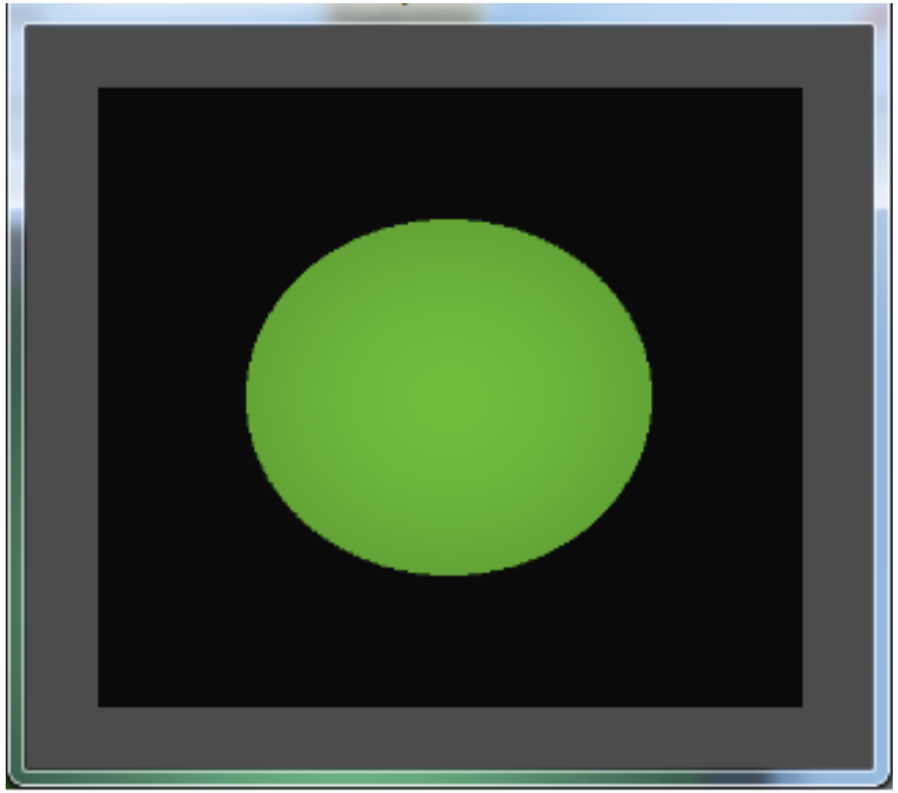
Точечный источник

выключено убывание интенсивности

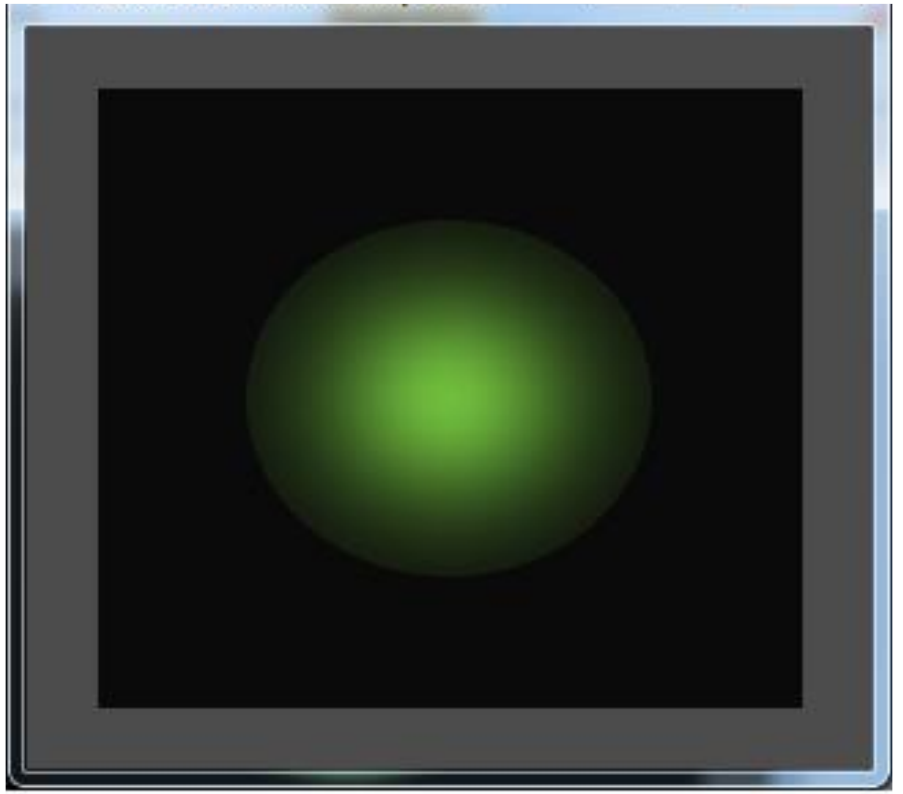


включено убывание интенсивности

Прожектор

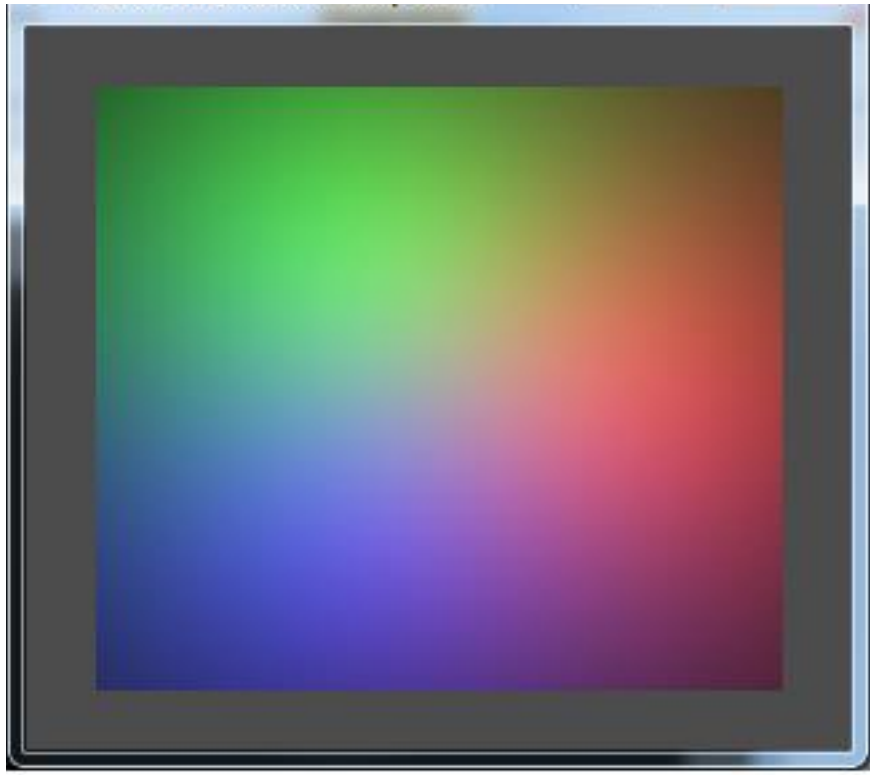


выключено убывание интенсивности



включено убывание интенсивности

Несколько источников света

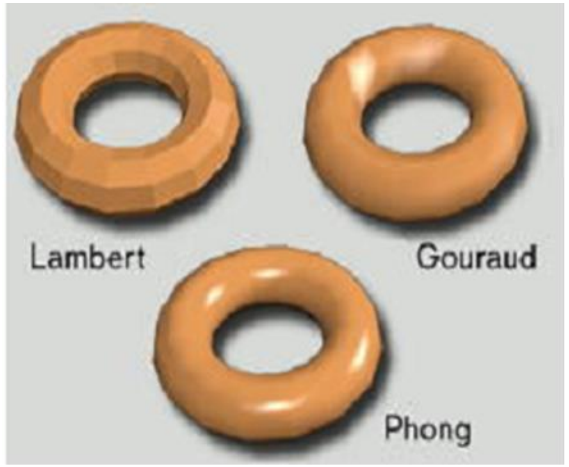


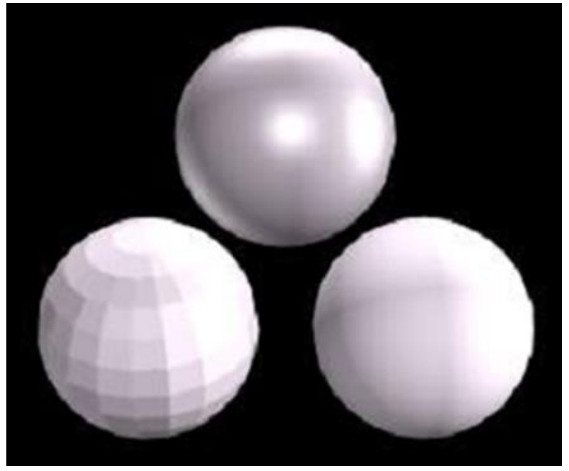
Разное количество глобального фонового света

Модели затенения (shading model)

Модель затенения представляет определенный тип интерполяции, позволяющей получить конечное значение цвета объекта в зависимости от освещения.

1. Плоское закрашивание (по Ламберту) — Вычисление цвета для примитива (flat shading)
2. Закрашивание по Гуро — Вычисление на вершинах с последующей интерполяцией (vertex shading)
3. Закрашивание по Фонгу — Вычисление цвета по модели освещения для каждого пикселя (per-pixel shading)

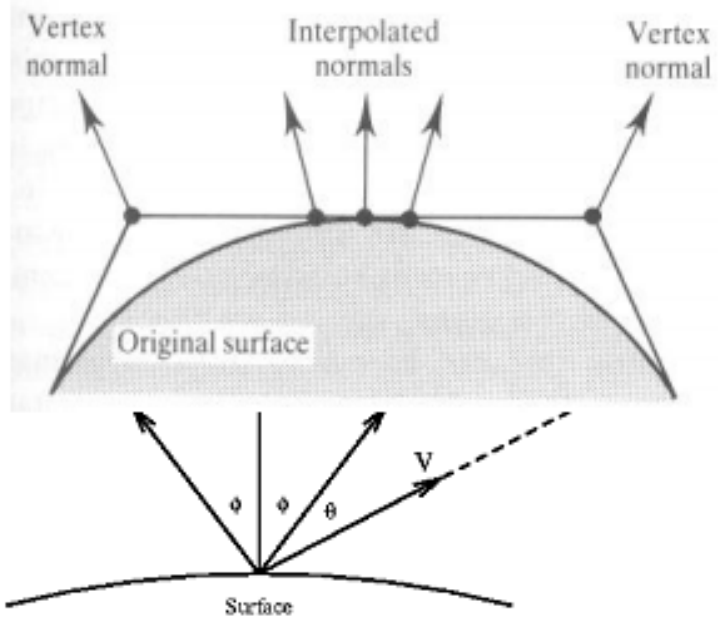




Генерация цвета пикселя: нужна интерполяция, или цвета, или нормалей

Закраска Фонга (пописельное)

- Расчет нормалей в вершинах
- Линейная интерполяция нормалей по треугольнику
- Расчет цвета по формуле Фонга (или другой)

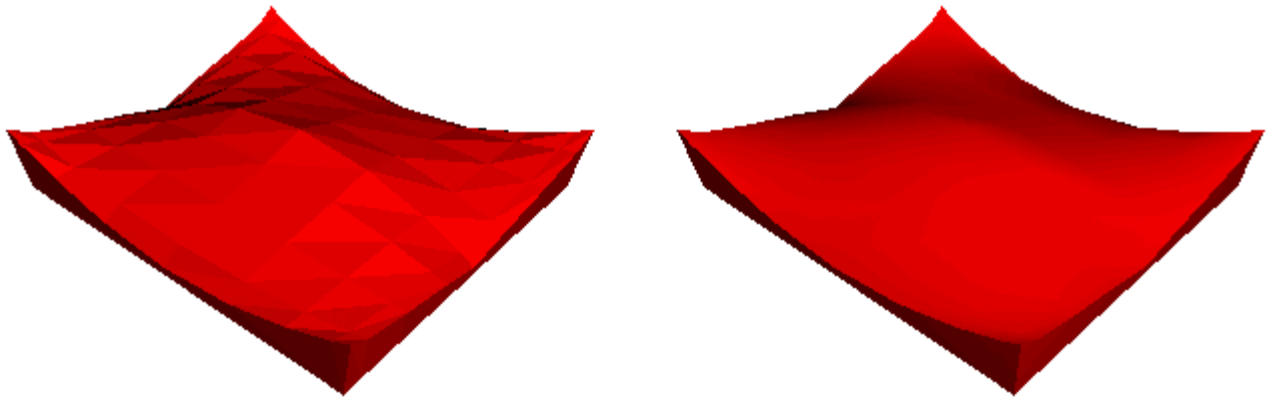


$$I_r = I_{pr} k_s (\cos\theta)^n \qquad I_g = I_{pg} k_s (\cos\theta)^n \qquad I_b = I_{pb} k_s (\cos\theta)^n$$

Закраска Гуро (повершинное)

- Расчет цвета в вершинах
- Линейная интерполяция цвета

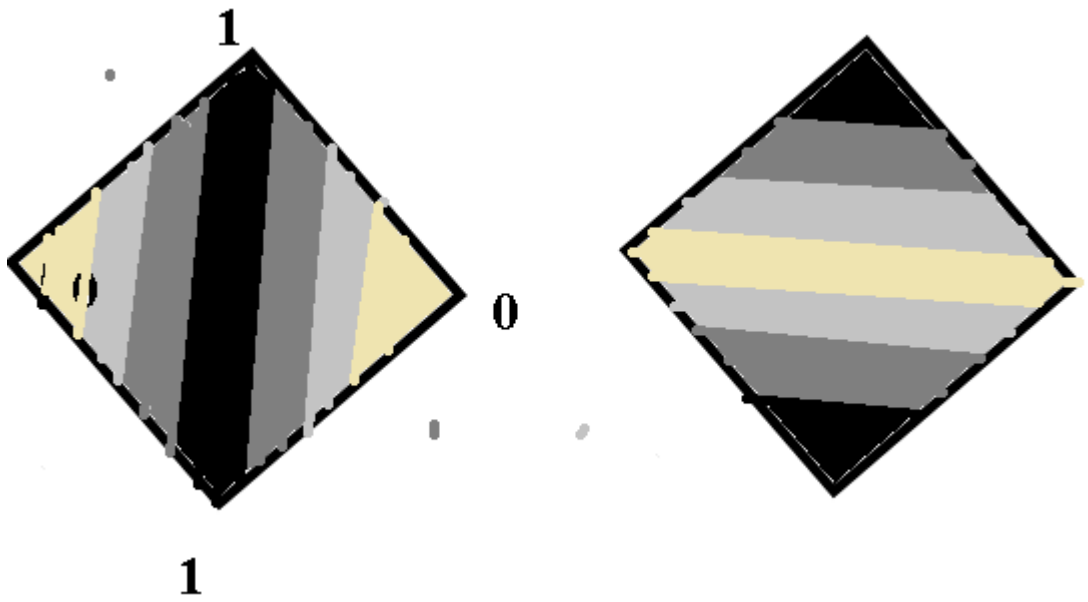
Затенение по Ламберту и по Гуро (Gourad Shading) 1971



Вопрос

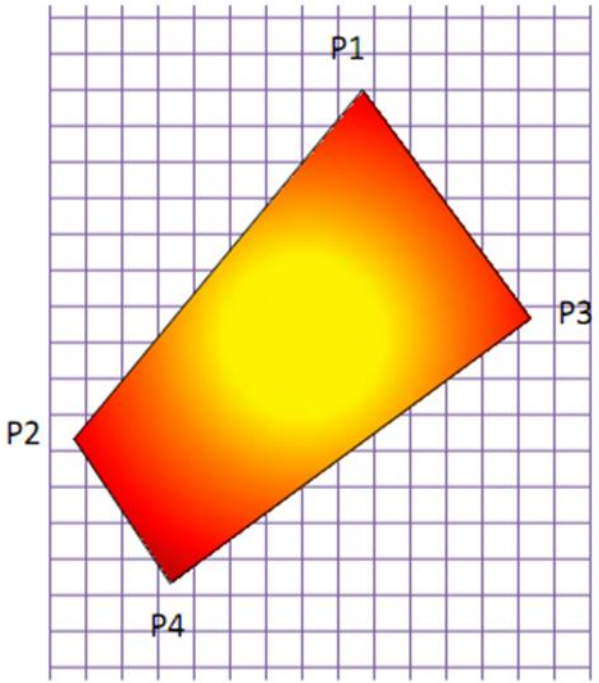
- Представьте себе — вы имеете большой полигон, освещенный единственным источником света, расположенным в центре полигона.
- Освещенность, создаваемая источником света в вершинах полигона, будет очень малой ввиду значительной удаленности от источника света.
- Что мы увидим?

Проблемы закраски Гуро

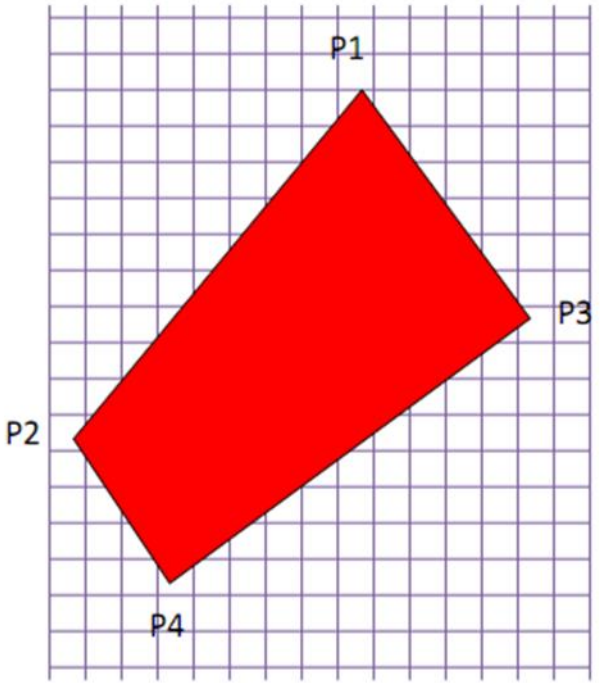


Результат зависит от направления интерполяции

можно пропустить блики



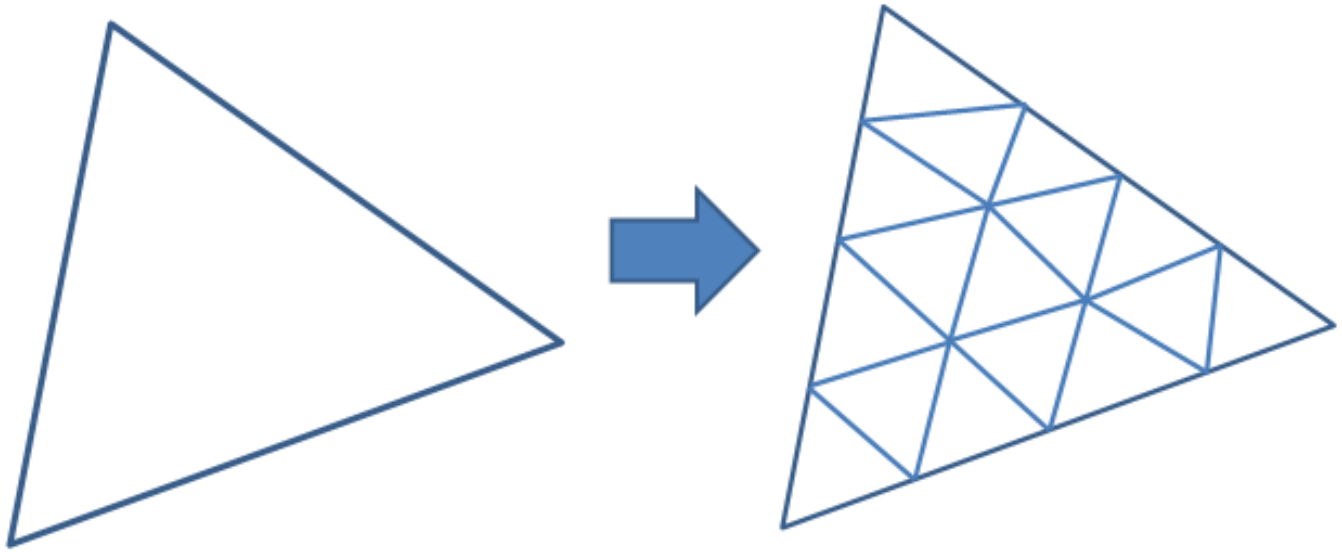
Фонг



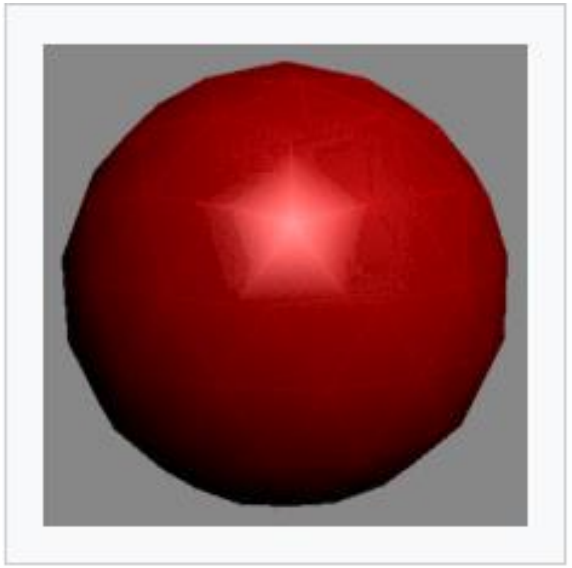
Гуро

Решение проблемы закраски Гуро: подразбиение полигонов

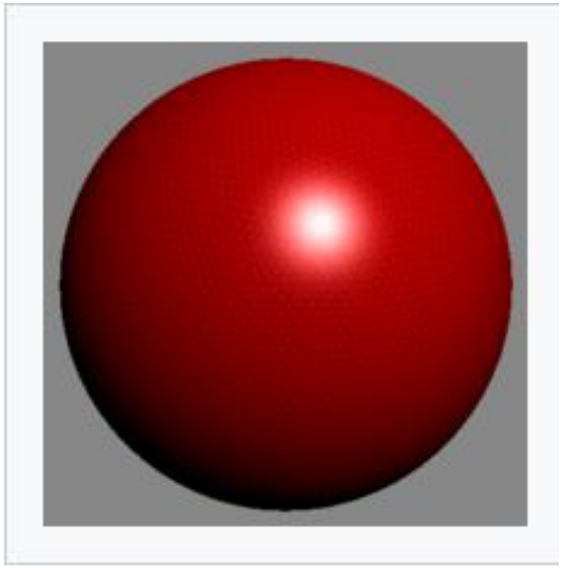
Увеличивает сложность геометрии



Метод тонирования Гуро

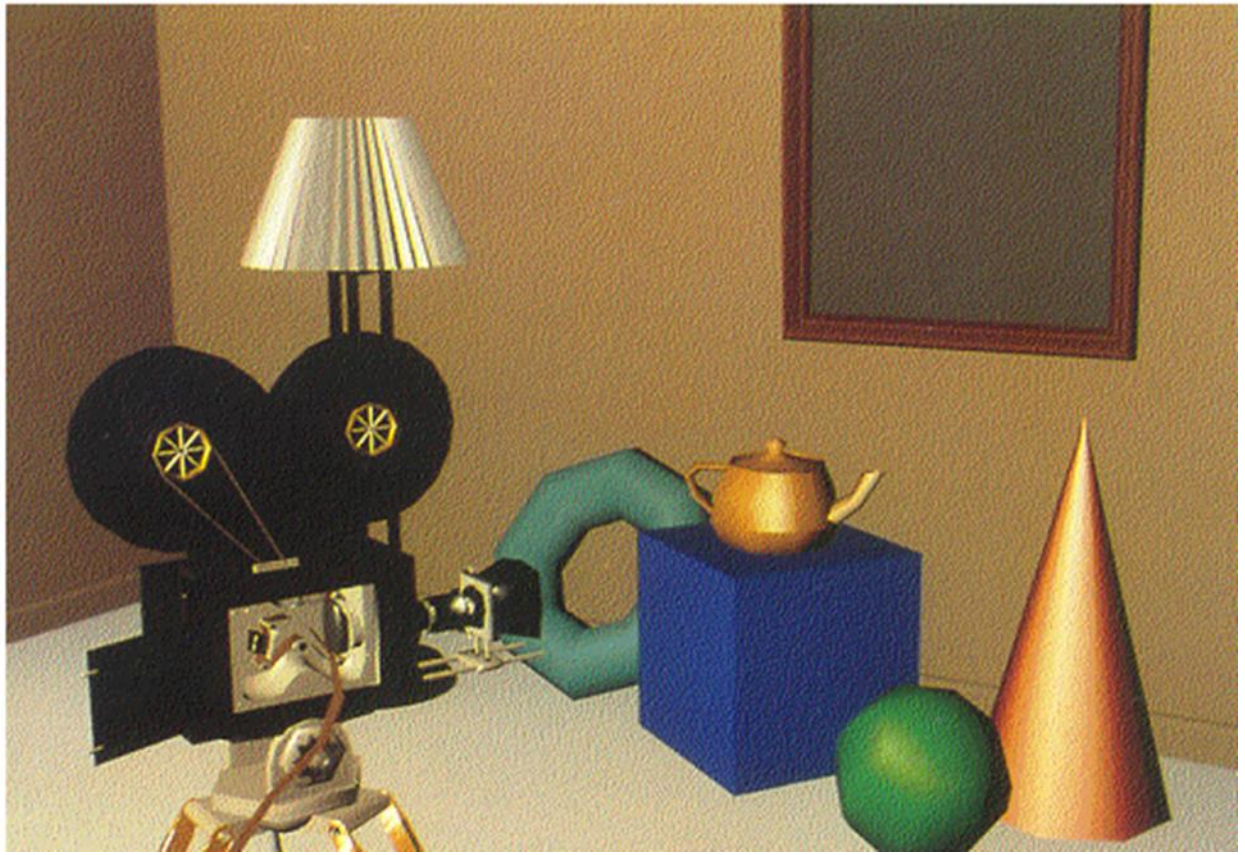


Сфера, тонированная по Гуро (малое количество граней)



Она же, но со значительно большим количеством граней.

Закраска Гуро (Gouraud) – зеркальное (specular) отражение



Закраска Фонга (Phong) – зеркальное отражение

Модели освещения (lighting model)

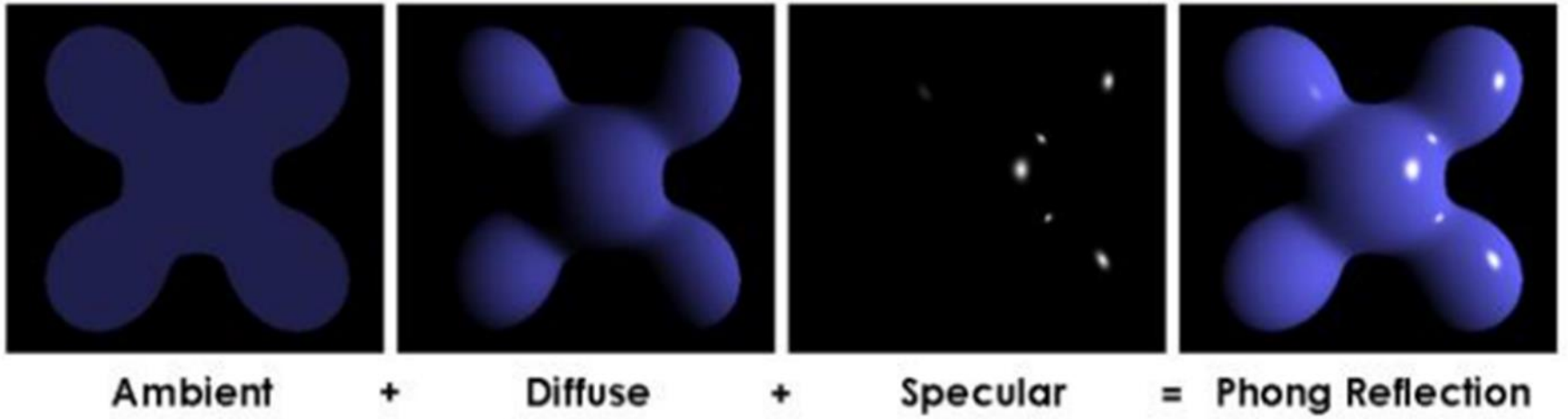
Модель освещения определяет способ взаимодействия материалов и света для получения финального значения цвета объекта.

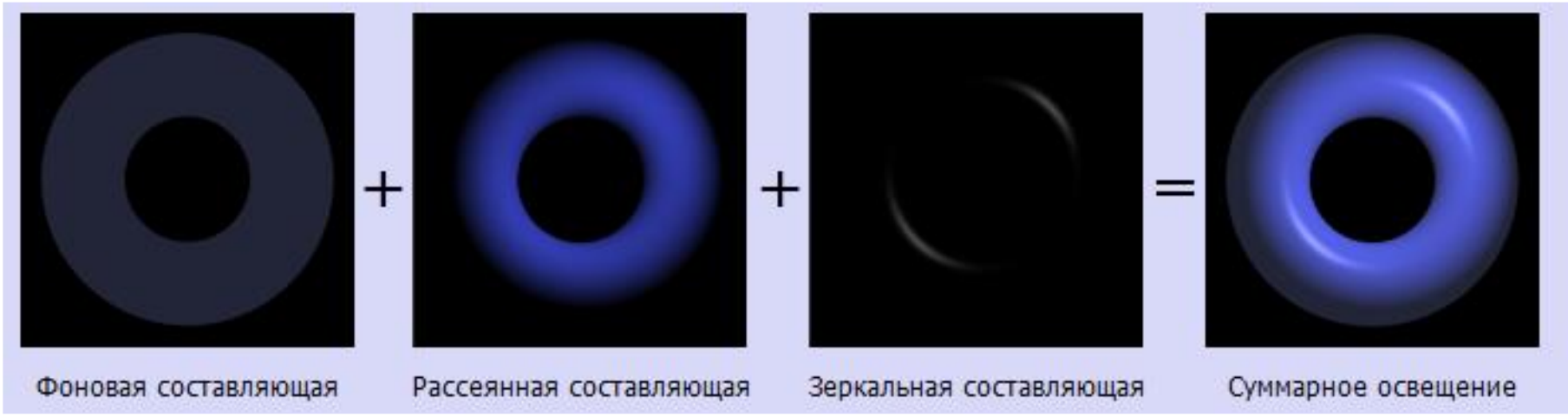
- Ламберт
- Фонг
- Блинн
- Тун-шейдер
- Орен-Найар
- Минаерт
- Кук-Торранс

Физический смысл модели Фонга

В модели отражения Фонга отраженный свет представлен как **сумма фонового, диффузного и зеркального** отражений.

В вершинном шейдере мы находим эту сумму и затем передаем переменную во фрагментный шейдер для окончательной установки цвета фрагмента (в случае модели затенения Гуро).

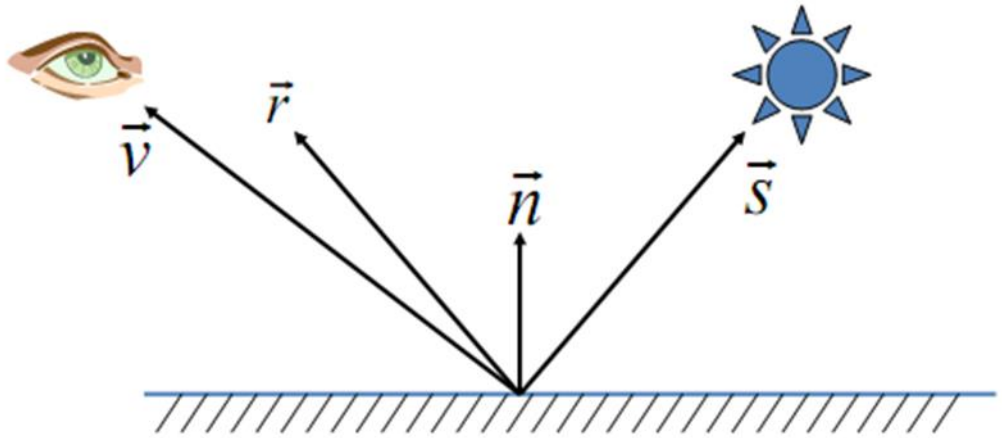




Модель Фонга

$$I = \frac{1}{k_0 + k_1 d + k_2 d^2}(k_d L_d \cdot \cos\theta + k_s L_s \cdot \cos^p \alpha) + k_a L_a$$

Учёт ослабления



Какие данные нужно передавать?

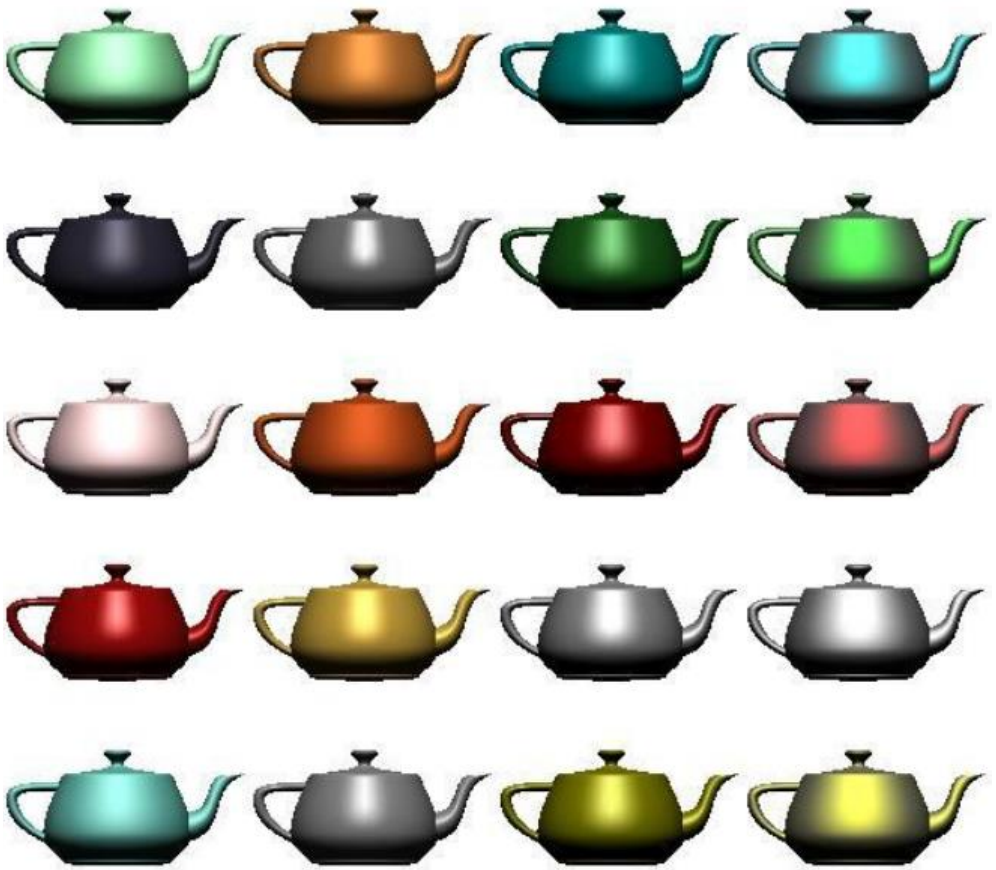
Материалы

Для источника света

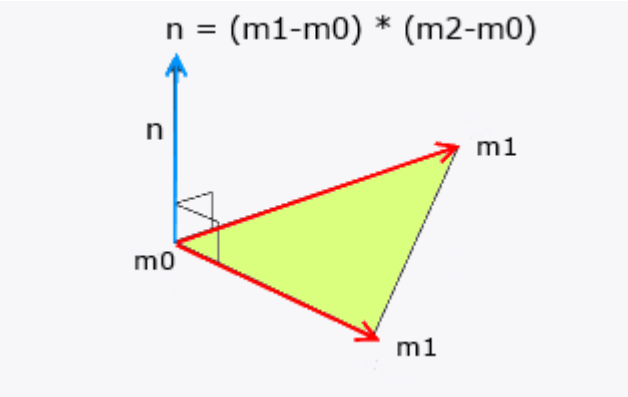
uAmbientLightColor;
uDiffuseLightColor;
uSpecularLightColor;

Для материалов объектов

AmbientLightColor;
DiffuseLightColor;
SpecularLightColor;



Нормали

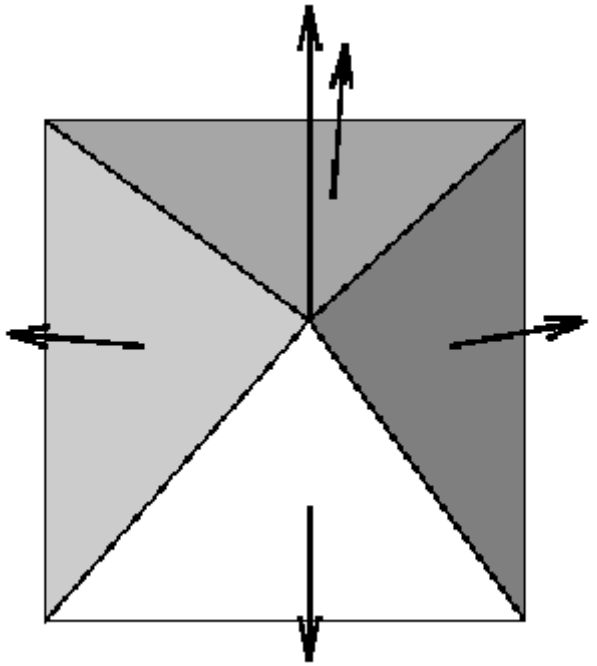


Использование библиотеки glMatrix в WebGL облегчает расчеты, так как там есть встроенная функция **vec3.cross(vector1, vector2, normal);**
где **normal** - это вектор нормали, получающийся в результате произведения векторов **vector1** и **vector2**.

Перед использованием вектор нормали надо нормализовать, то есть привести к единичному виду. Для этого в **языке шейдеров WebGL** определена специальная функция **normalize**.

Вычисление нормалей в вершинах

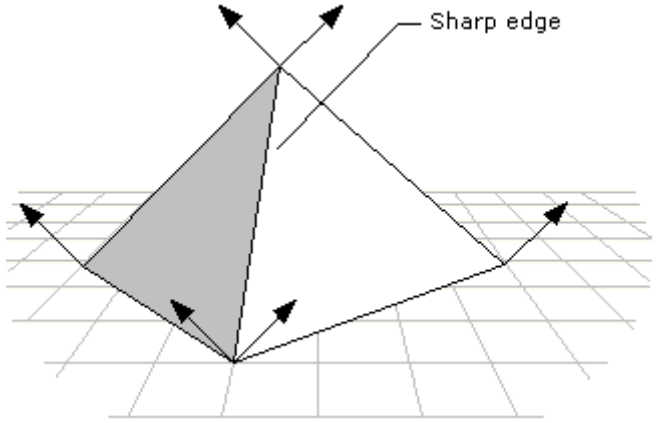
Нормали в вершинах вычисляются усреднением нормалей смежных граней

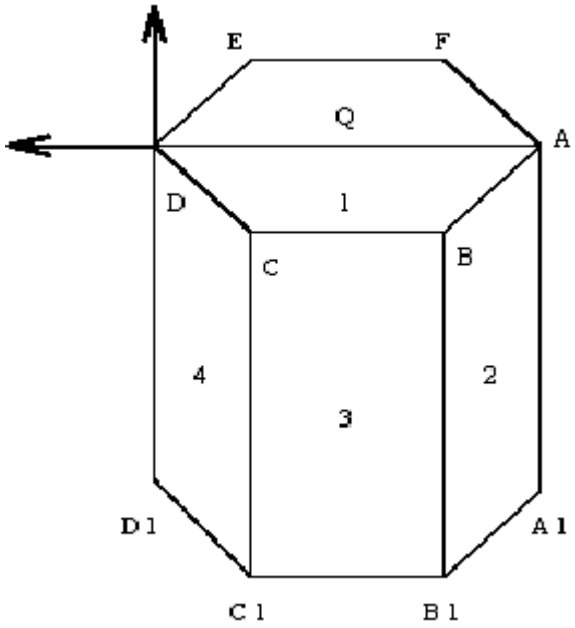


Для корректной закраски на стыках поверхности необходимо «клонирование» вершин

- В вершине D нужно иметь две нормали:
  - Одна нормаль для гладкой закраски боковой поверхности
  - Другая нормаль для закраски торца

- На острых ребрах нормали дублируются





На стороне javascript

В стандартном коде требуется только передать ряд характеристик объектов в шейдеры (дополнительно к выводу без освещения)

- Матрицу нормалей
- Нормали вершины
- Параметры источников освещения
- Цвета освещения

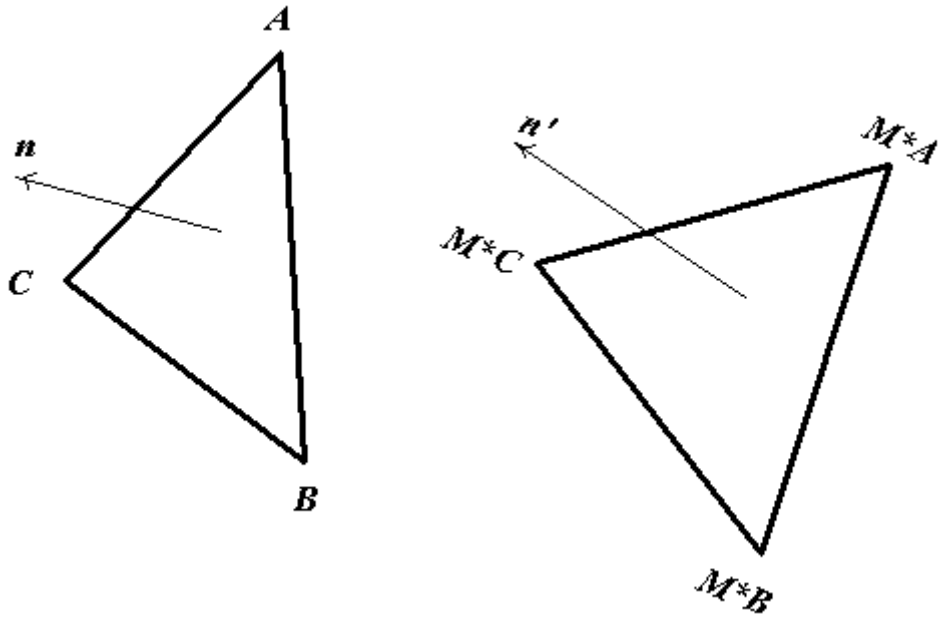
Настройка цветов освещения

```
function setupLights() {
  //позиция источника света
  gl.uniform3fv(shaderProgram.uniformLightPosition, [0.0, 10.0, 5.0]);
  //составвляющие цвета
  gl.uniform3fv(shaderProgram.uniformAmbientLightColor, [0.1, 0.1, 0.1]);
  gl.uniform3fv(shaderProgram.uniformDiffuseLightColor, [0.7, 0.7, 0.7]);
  gl.uniform3fv(shaderProgram.uniformSpecularLightColor, [1.0, 1.0, 1.0]);
}
```

Добавилась матрица нормалей

```
var mvMatrix = mat4.create(); // матрица вида модели
var pMatrix = mat4.create();  // матрица проекции
var nMatrix = mat3.create();  // матрица нормалей

function setupWebGL() {
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT || gl.DEPTH_BUFFER_BIT);

    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    mat4.perspective(pMatrix, 1.04, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix,mvMatrix,[0, 0, zTranslation]);
    mat4.rotateX(mvMatrix,mvMatrix, xAngle);
    mat4.rotateY(mvMatrix,mvMatrix, yAngle);

    mat3.normalFromMat4(nMatrix, mvMatrix);
}
```

Преобразование нормалей



$$(n,B-A)=0$$
$$(n,C-A)=0$$

$$(n',M\cdot(B-A))=0$$
$$(n',M\cdot(C-A))=0$$

$$0=(n',M\cdot(B-A))=(M^T\cdot n',B-A)$$
$$0=(n',M\cdot(C-A))=(M^T\cdot n',C-A)$$

$$M^T\cdot n'=n$$

$$n'=(M^T)^{-1}n$$

mat3.normalFromMat4(nMatrix, mvMatrix);

Вершинный шейдер

```
attribute vec3 aVertexPosition;
attribute vec3 aVertexNormal;
//attribute vec2 aVertexTextureCoords;

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
uniform mat3 uNMatrix;

uniform vec3 uLightPosition;
uniform vec3 uAmbientLightColor;
uniform vec3 uDiffuseLightColor;
uniform vec3 uSpecularLightColor;

//out vec2 vTextureCoords;
out vec3 vLightWeighting;
out lowp vec4 vColor;

const float shininess = 16.0;
```

Последовательность действий

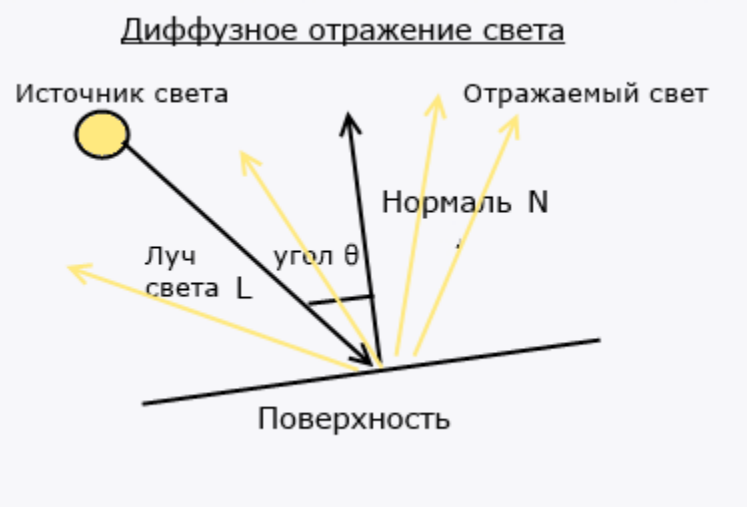
Вначале необходимо модифицировать координаты вершины и ее нормаль, чтобы правильно провести вычисления по нахождению отражения света, поскольку на входе в шейдер и вершина, и нормаль пока не учитывают преобразования с матрицами — вращения, перемещения и т.д.:

```
void main() {
  // установка позиции наблюдателя сцены
  vec4 vertexPositionEye4 = uMVMatrix * vec4(aVertexPosition, 1.0);
  vec3 vertexPositionEye3 = vertexPositionEye4.xyz / vertexPositionEye4.w;

  // получаем вектор направления света
  vec3 lightDirection = normalize(uLightPosition - vertexPositionEye3);

  // получаем нормаль
  vec3 normal = normalize(uNMatrix * aVertexNormal);
```

Фоновое и Диффузное отражение света

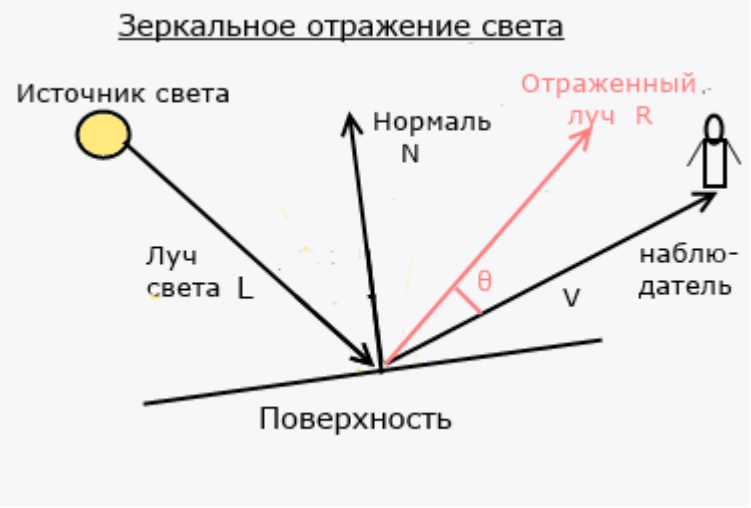
Фоновое отражение (ambient light): **I = Ka*Ia**,
где **Ka** – цветовые параметры материала, а **Ia** – это цвет фонового света.

Диффузное отражение (diffuse light): **I = Kd *Id *max(cos θ, 0)**
Kd – диффузный материал и **Id** – цвет освещения.
max(cos θ, 0) – учитывает направление луча, направленного на поверхность.
θ – угол между нормалью поверхности и вектором направления луча света.



//скалярное произведение векторов нормали и направления света
float diffuseLightDot = max(dot(normal, lightDirection), 0.0);

Зеркальное отражение света



Зеркальное отражение (specular light): $\mathbf{I = Ks * Is \max(\cos \theta, 0)^a}$

Ks – материал, а **Is** – цвет зеркального отражения

θ – угол между вектором, направленным от точки к наблюдателю, и вектором отражаемого луча

a – указывает на степень блеска материала

Вектор луча отражения R: **R=2(L * N)*N-L**

Однако язык шейдеров имеет встроенную функцию **reflect**, которая позволяет найти вектор отраженного луча по нормали и направлению падающего света.

```
// получаем вектор отраженного луча и нормализуем его
vec3 reflectionVector = normalize(reflect(-lightDirection, normal));

// установка вектора камеры
vec3 viewVectorEye = -normalize(vertexPositionEye3);

float specularLightDot = max(dot(reflectionVector, viewVectorEye), 0.0);

float specularLightParam = pow(specularLightDot, shininess);
```

Вершинный шейдер

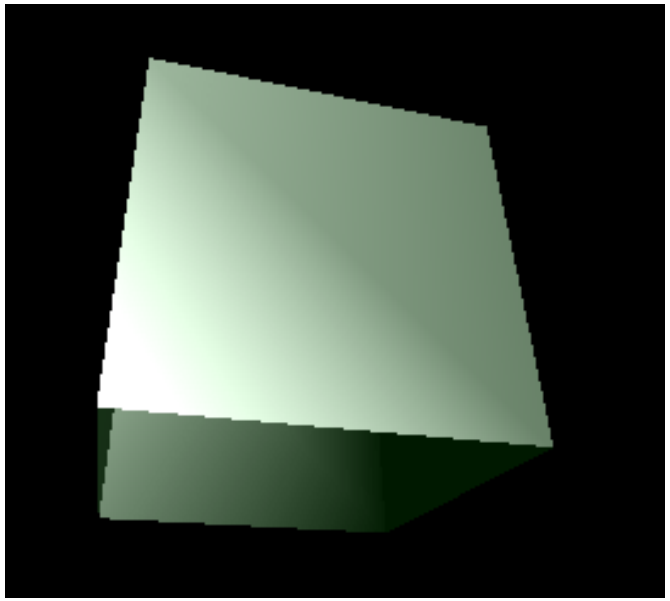
И в конце мы собираем все световые значения и получаем общее значение отраженного света, которое затем передаем во фрагментный шейдер.

```
// отраженный свет равен сумме фонового, диффузного и зеркального отражений света
vLightWeighting = uAmbientLightColor + uDiffuseLightColor * diffuseLightDot + uSpecularLightColor *
                  specularLightParam;

// или
vLightWeighting = uAmbientMaterialColor * uAmbientLightColor +
                   uDiffuseMaterialColor * uDiffuseLightColor * diffuseLightDot +
                  uSpecularMaterialColor * uSpecularLightColor * specularLightParam;

// Finally transform the geometry
gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
//vTextureCoords = aVertexTextureCoords;
 }
```

Фрагментный шейдер

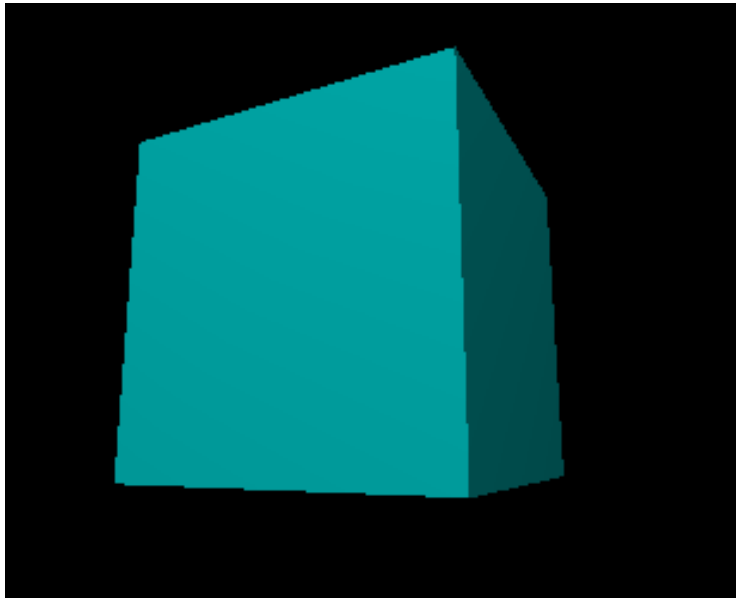
```
precision mediump float;

 //in vec2 vTextureCoords;
 in vec3 vLightWeighting;
 //uniform sampler2D uSampler;
 in lowp vec4 vColor;

 void main() {
   //vec4 texelColor = texture2D(uSampler, vTextureCoords);
   //gl_FragColor = vec4(vLightWeighting.rgb * texelColor.rgb, texelColor.a);
   gl_FragColor = vec4(vLightWeighting.rgb * vColor.rgb, vColor.a);
 }
```

В итоге как-то так))



Освещение Ламберта

Модель Ламберта намного проще модели Фонга, так как учитывает только одну составляющую - диффузное отражение.



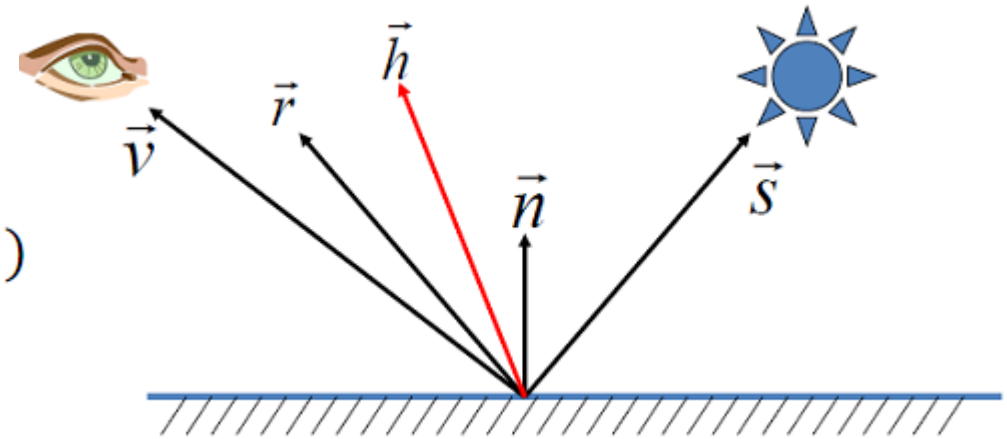
Модель Блинна-Фонга

- Это унификация модели Фонга, принципиальной разницы между двумя этими моделями нет.

$$L_O = L_i(k_d(s,n) + k_s(r,v)^{k_l})$$

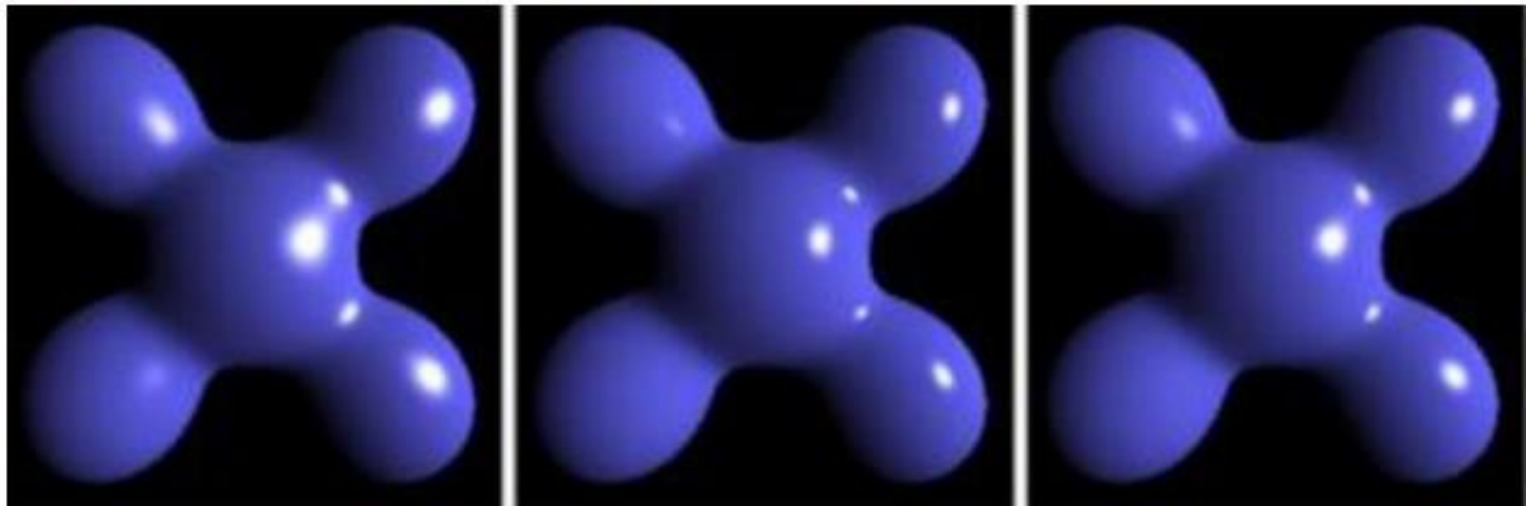
$$L_O = L_i(k_d(s,n) + k_s(h,n)^{k_l})$$

$$h = \frac{(v+s)}{2}$$



Данная аппроксимация позволяет в ряде случае упростить вычисления.

Применение модели освещения Блинна-Фонга вместо модели Фонга **непринципиально** изменяет результат

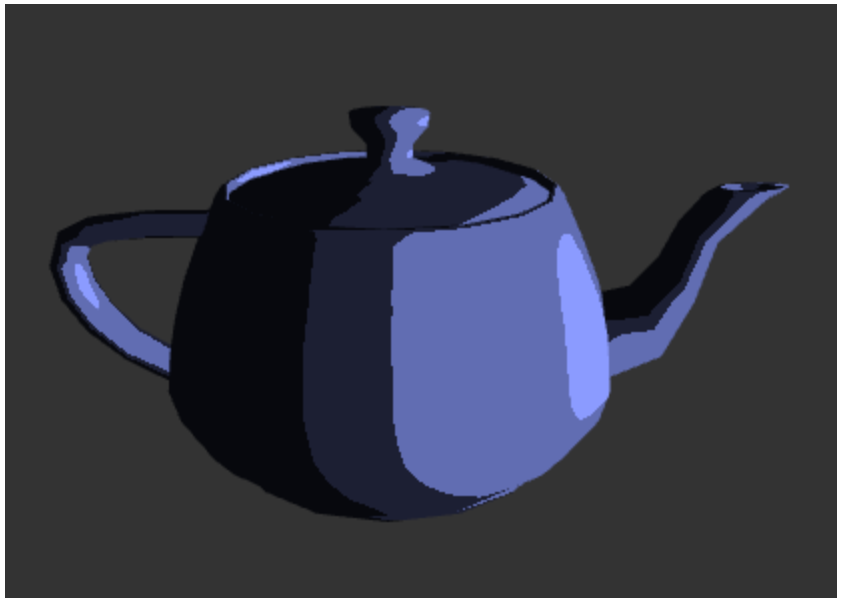


Слева – результат применения модели Блинна-Фонга,
посередине – результат применения модели Фонга,
справа – модель Блинна-Фонга с меньшей, чем у левого изображения, степенью.

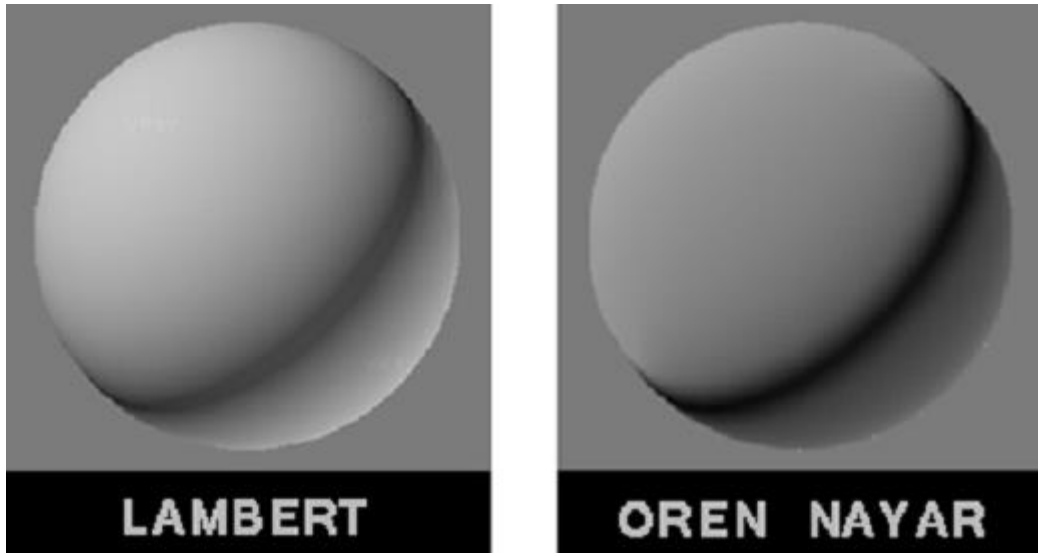
Модели Фонга и Блинна

Сэл-шейдерная или тун-шейдерная модель

Эта модель дает изображение, имитирующее рисование вручную. Использование этой модели можно встретить в таких мультфильмах как Человек-паук, Футурама и других.
Идея шейдера довольно простая: мы берем за основу модель Ламберта, а затем вводим резкие переходы в освещении от порога к порогу.
Например, при значениях коэффициента Ламберта от 0.5 до 0.95 цвет умножается на коэффициент 0.7 — таким образом мы убираем плавное сглаживание в освещении.



Модель Ламберта – Модель Орен-Найара

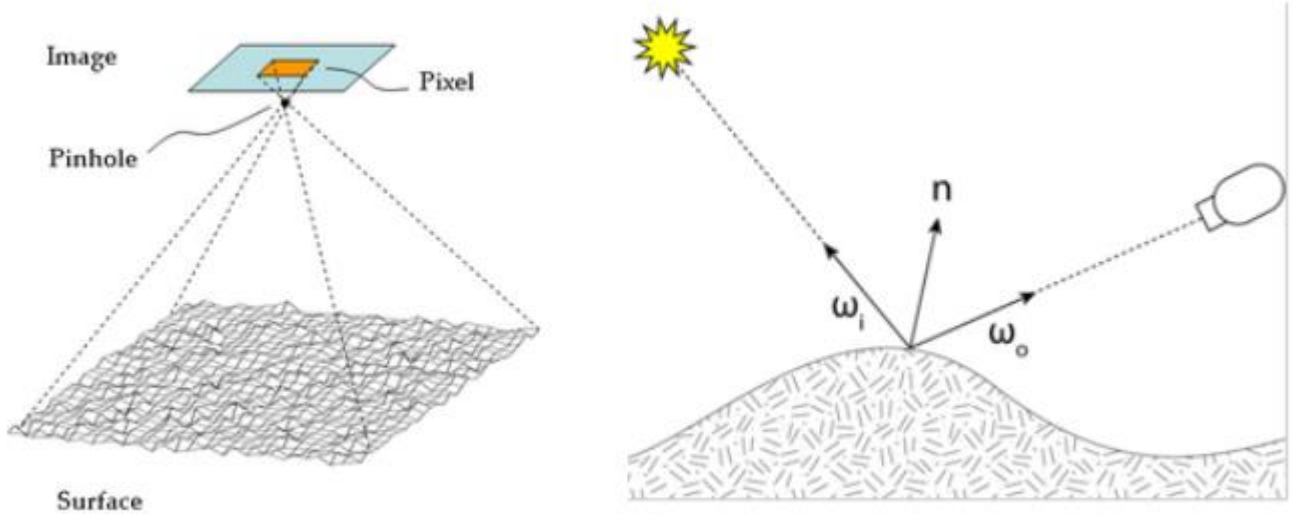


– цвет шершавых поверхностей – для подповерхностного отражения.
Модель имеет параметр для контроля шершавости поверхности. Он определяет, сколько света отразится назад в направлении источника света, что является характеристикой "шершавой" (запыленной) поверхности.

Чем более шершавая поверхность, тем менее отчетливым является диффузное отражение. Шершавая поверхность рассеивает свет во всех направлениях, но абсолютно неравномерно.

Поэтому Орен-Найар – это упрощенное представление реальности.

Модель Орен-Найара

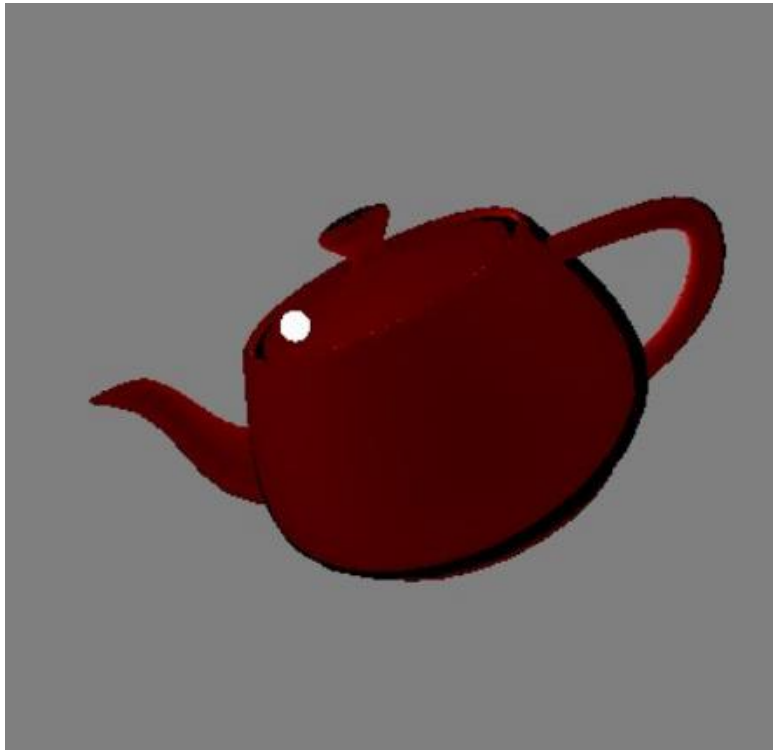


Шершавость не значит реально шершавый (как наждачная бумага, к примеру), речь скорее идет о малюсеньких выступах (микроструктуре) на поверхности

Модель Орен-Найара

Шершавой считается поверхность кожи и бархата из-за наличия очень мелких деталей, таких как поры кожи и волокна бархата.

К примеру, поверхность пластика не такая шершавая, а резина, камень, ржавчина значительно грубее (шершавее) кожи или бархата.



Модель Минаерта

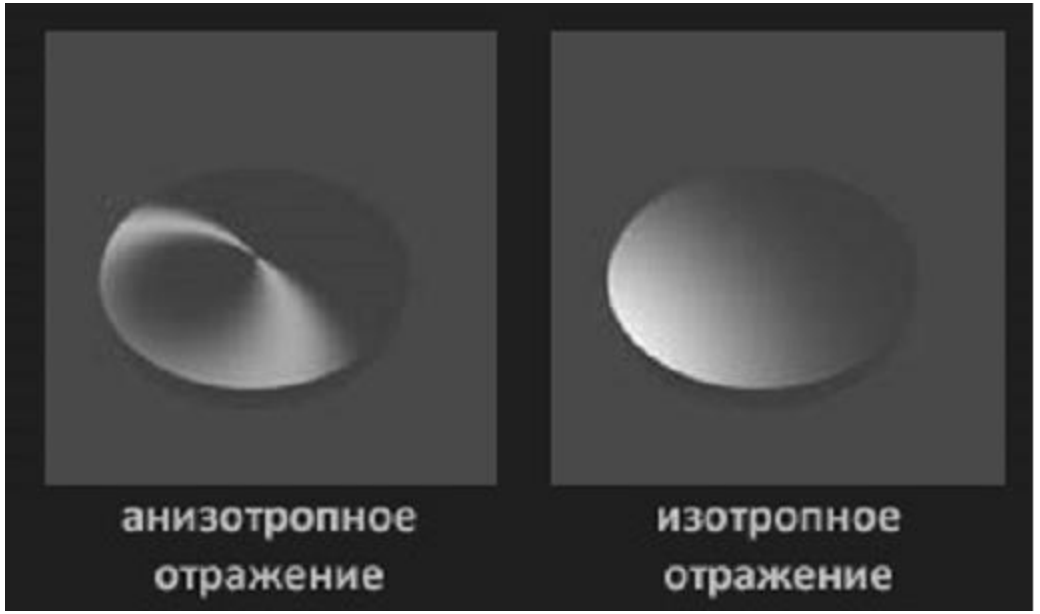
Изначально модель создавалась для моделирования Луны (поэтому иногда можно встретить название «лунный шейдер»), но также подходит для других поверхностей, имеющих корпускулярную или губчатую поверхность.

Модель Кука-Торранса (Cook-Torrance)

- Модель для зеркальных бликов на поверхностях с микрогранями.

Металлический заяц

Анизотропные модели



Ward-anisotropic (или просто Ward – анизотропные световые блики)