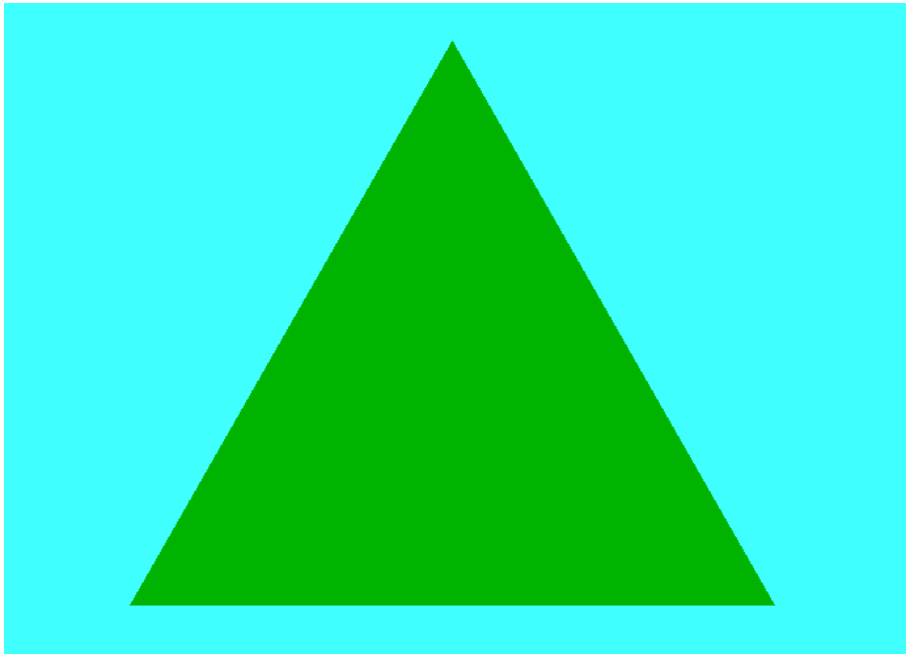
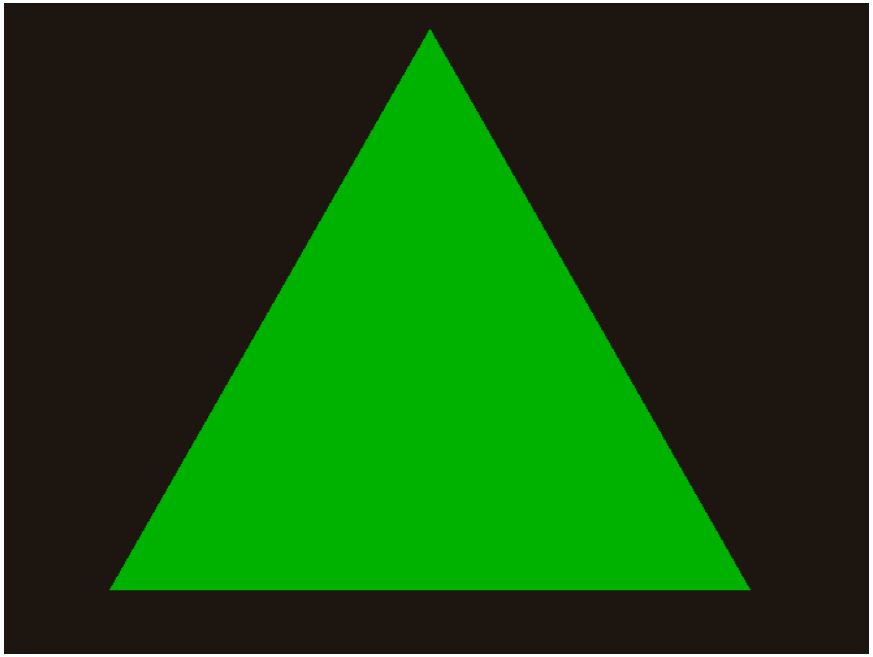
Компьютерная Графика

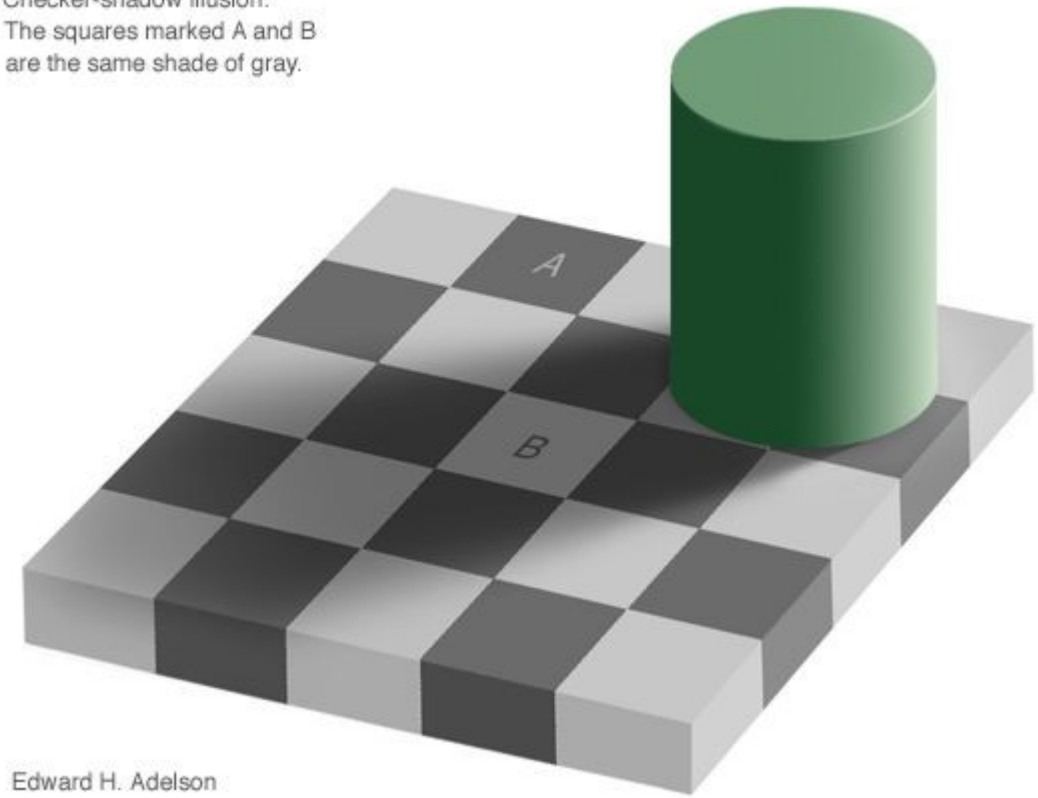
Как мы видим

Чувствительность глаза



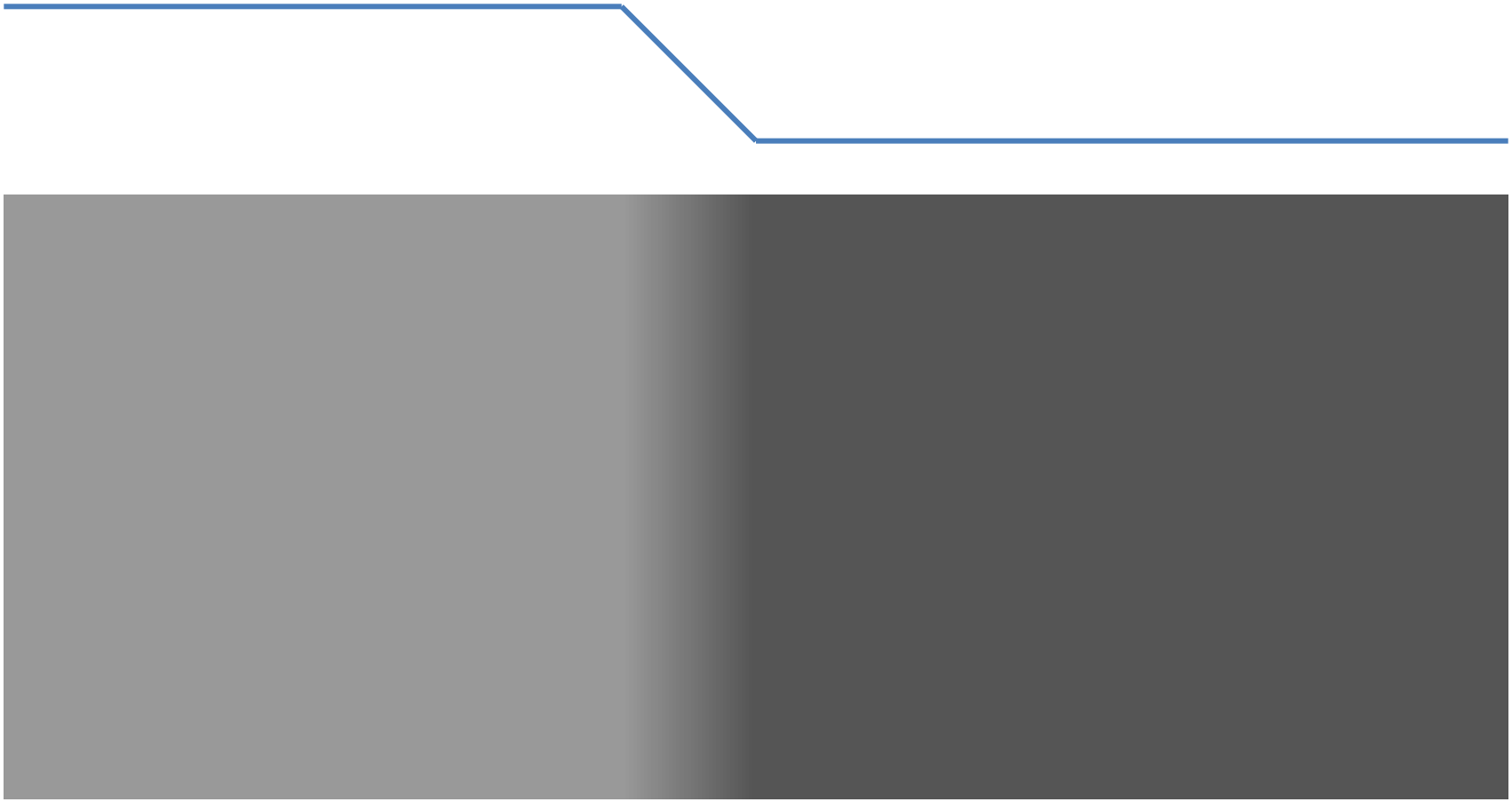


Checker-shadow illusion:
The squares marked A and B
are the same shade of gray.

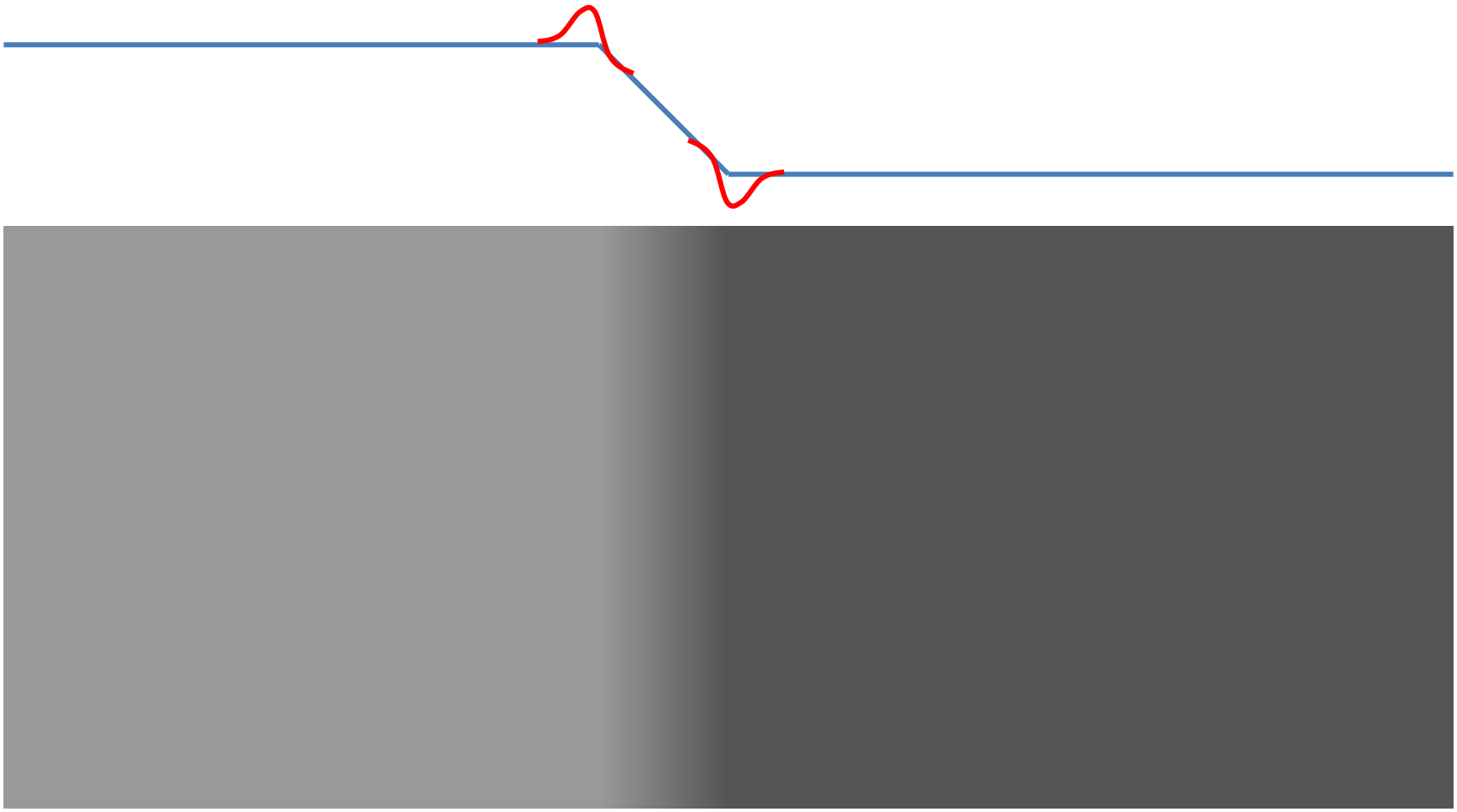


Edward H. Adelson

Полосы Маха



Полосы Маха

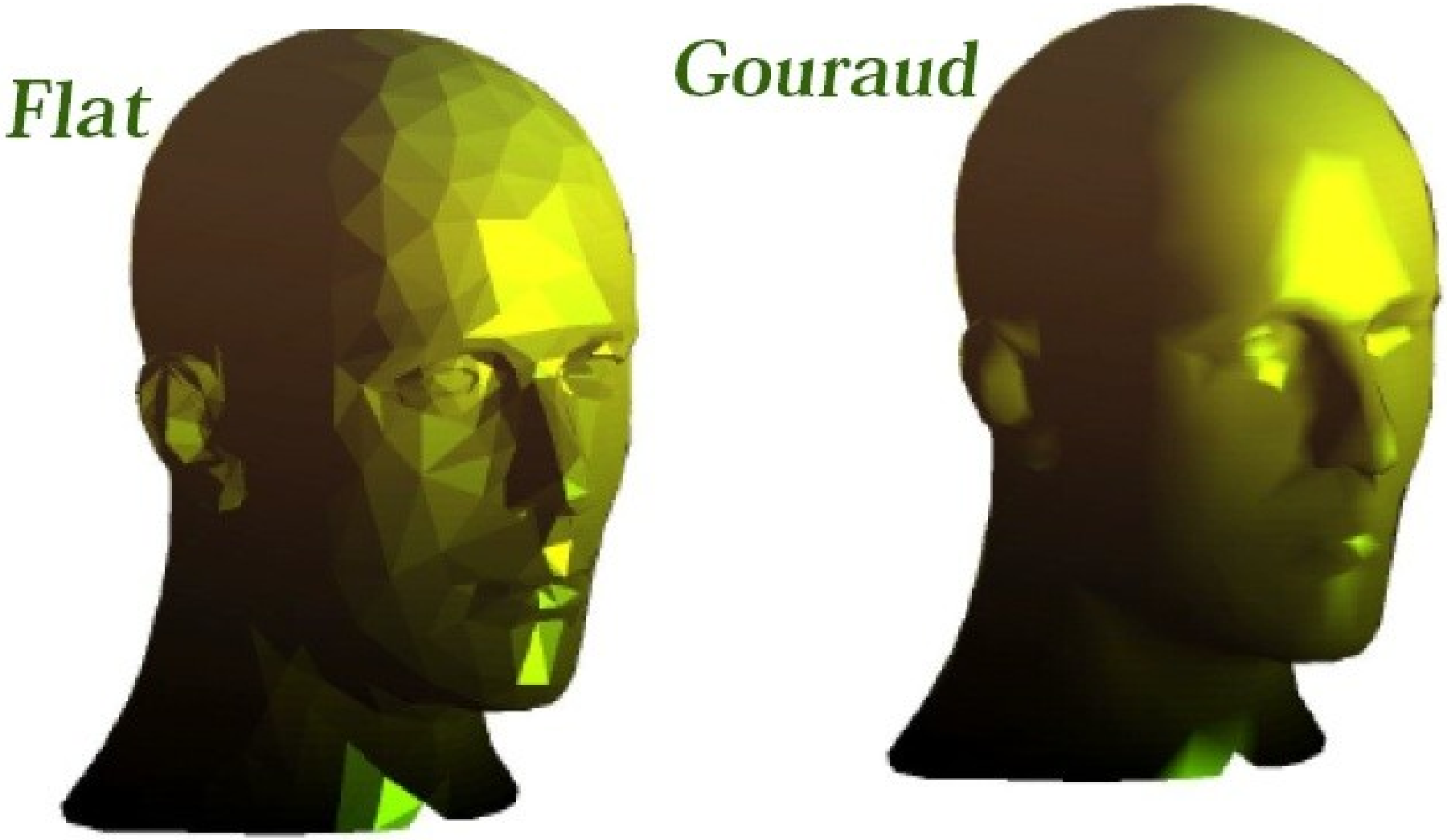


Глаз чувствует на границах перепады *интенсивности и ее производной*

Полосы Маха

Итак, это интересный визуальный артефакт, известный как Mach band effect — на границе скачкообразного перехода интенсивности человек видит темные цвета более темными, а светлые цвета более светлыми, чем они есть на самом деле, что в случае полигональных геометрических моделей еще более усиливает эффект «гранёности» поверхности

Полосы Маха

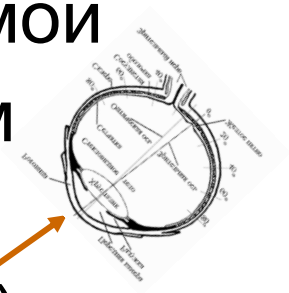


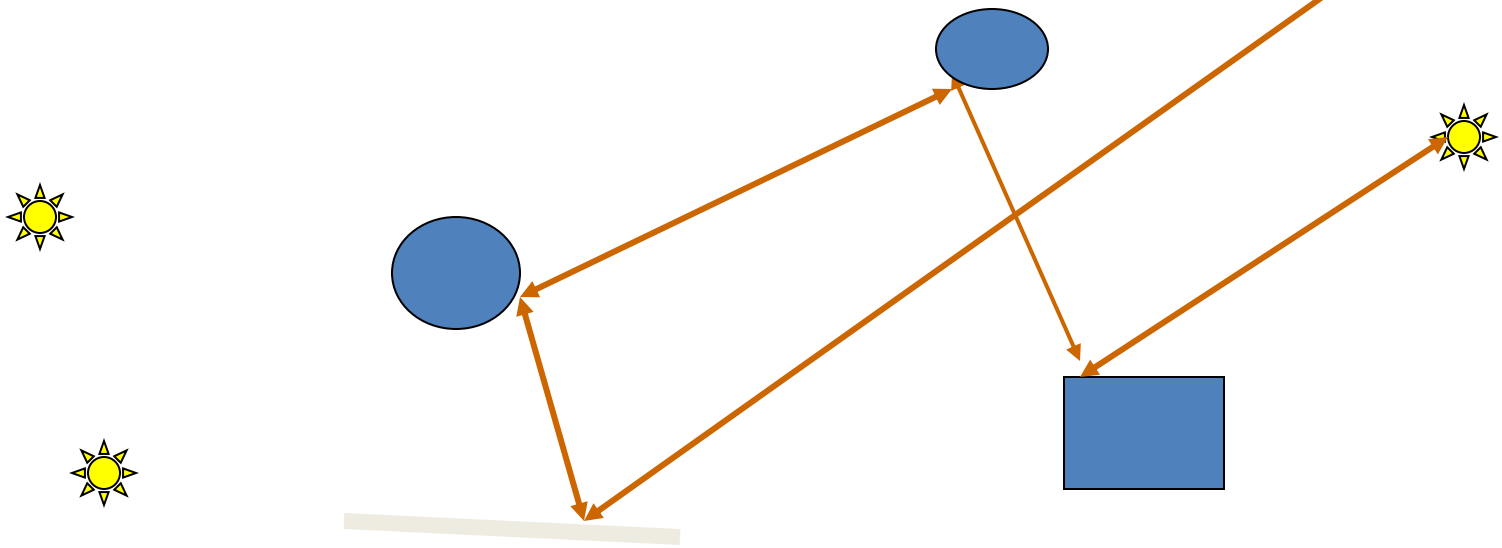
Компьютерная Графика

Освещение

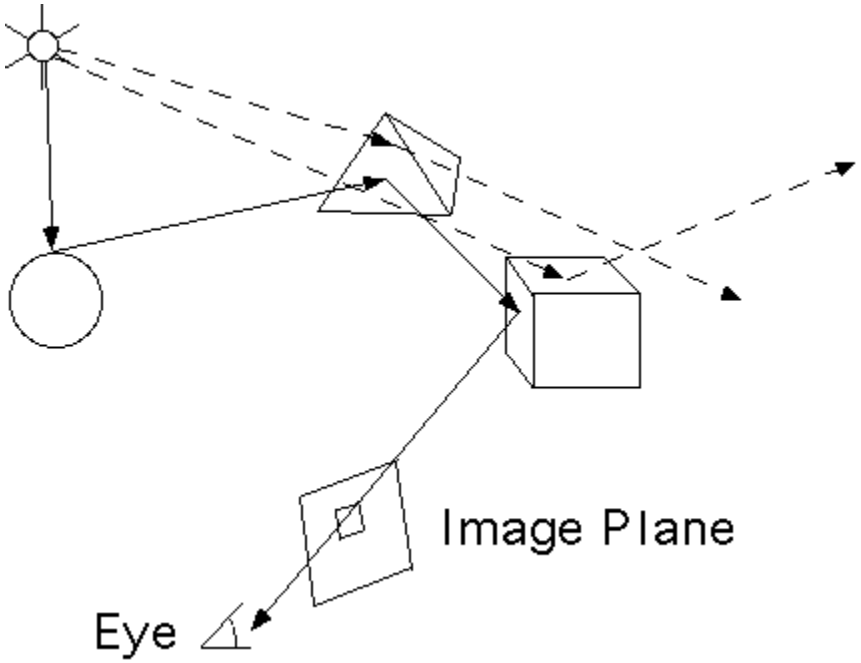
Геометрическая оптика и среда

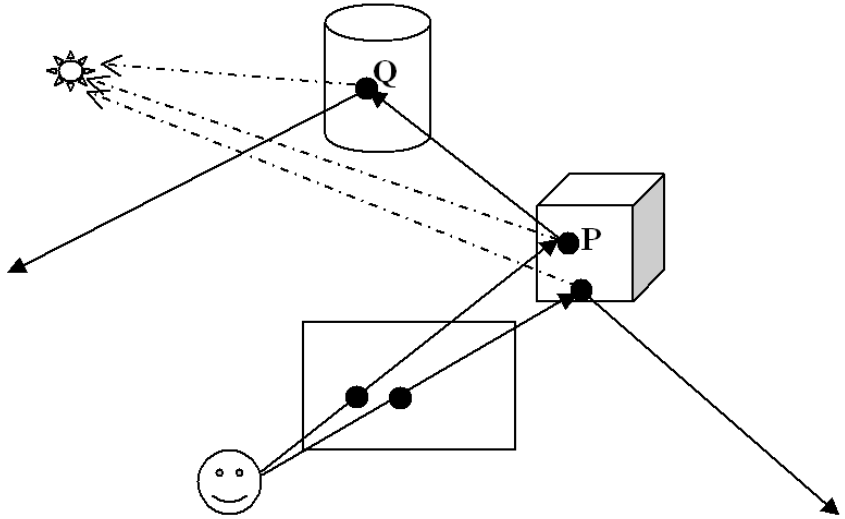
- Луч света (ЛС или фотон) всегда по прямой
- Два ЛС не взаимодействуют друг с другом
- ЛС не взаимодействует со средой
- Закон обращения (принцип Гельмгольца)





Прямая и обратная трассировки

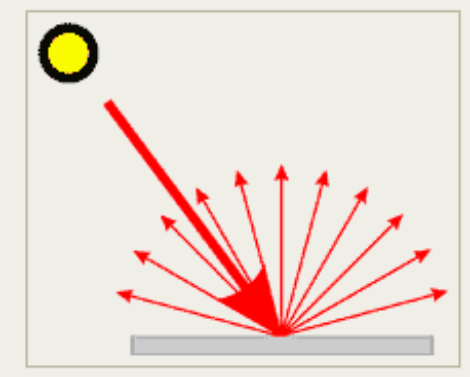




Отражение от поверхности

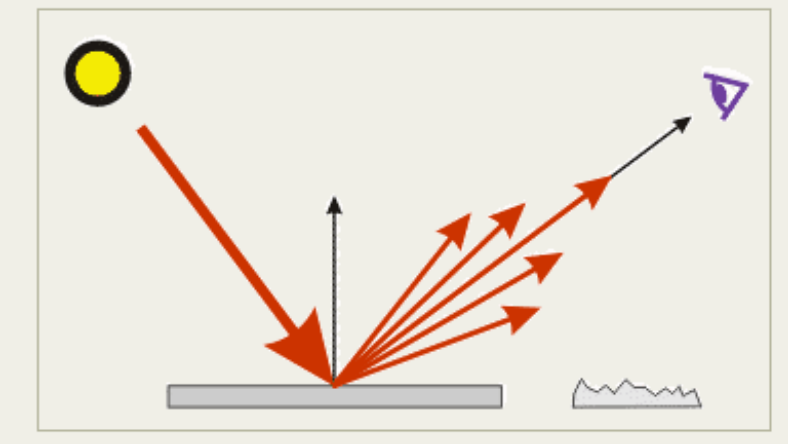
Диффузное

- Поглощение и эмиссия
- По всем направлениям равномерно

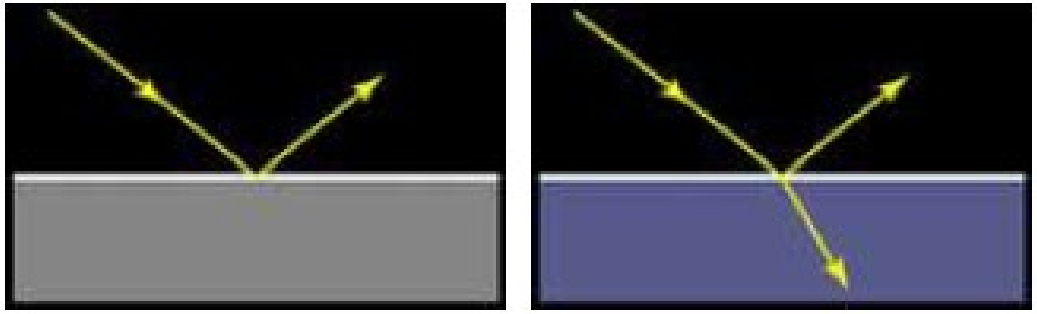


Зеркальное

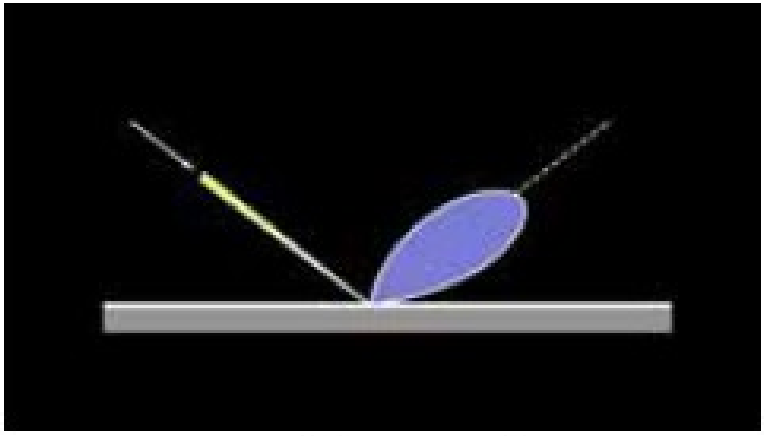
- От внешней поверхности
- Зависит от положения источников и камеры
- Гладкая – узкий блик; шероховатая – широкое пятно



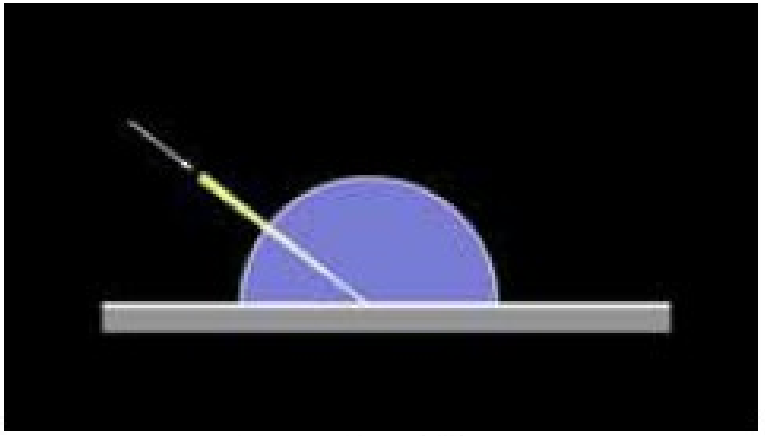
Допущения модели



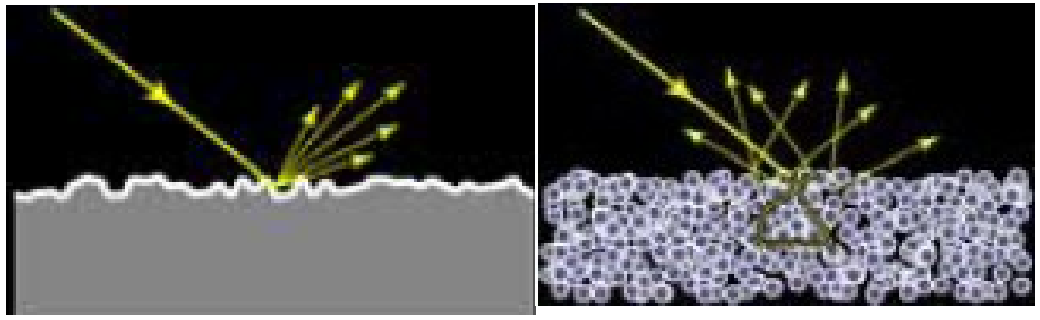
ideal specular (Fresnel)



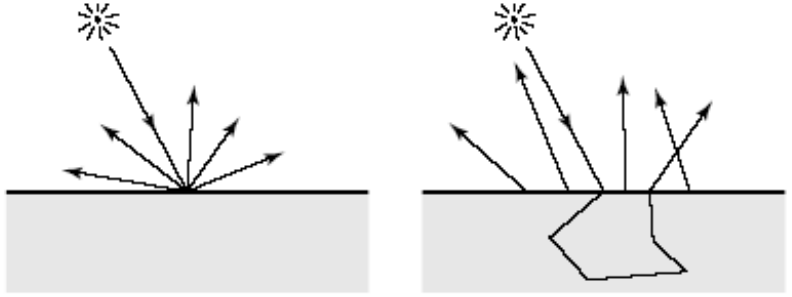
rough specular



Lambertian



Subsurface Light Transport



Локальная модель освещенности

- Каждый источник освещения – отдельно
- Линейная комбинация для нескольких источников:
  - I(a+b) = I(a) + I(b)
  - I(k*a) = k*I(a)
- Нет влияния объектов друг на друга
- Без теней, без отражений
- Каждый пиксель обрабатывается независимо от других пикселей

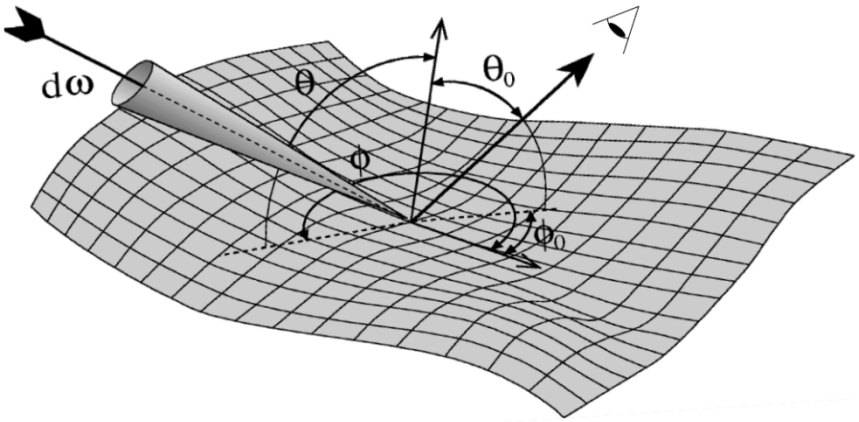
Локальная модель освещенности

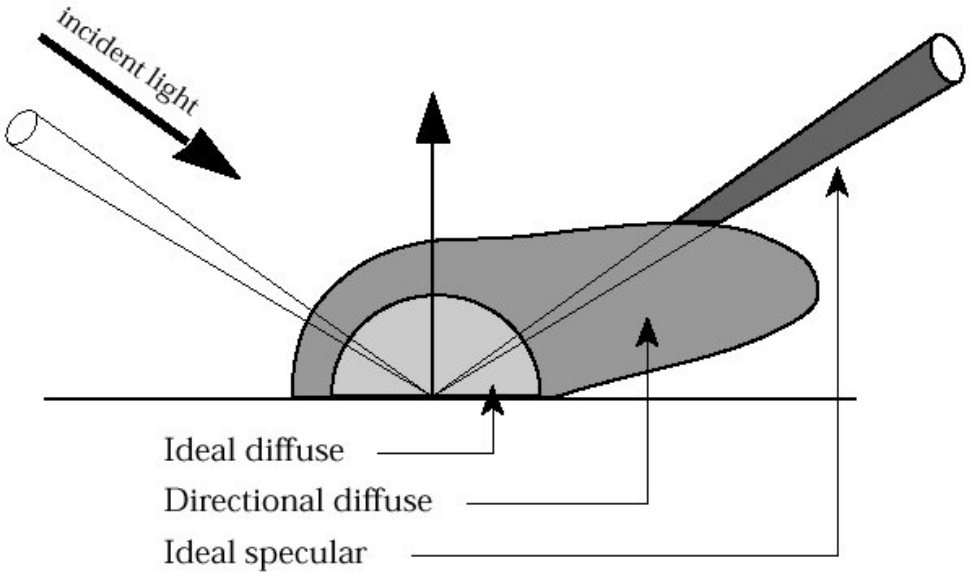
BRDF

Bidirectional reflectance distribution function

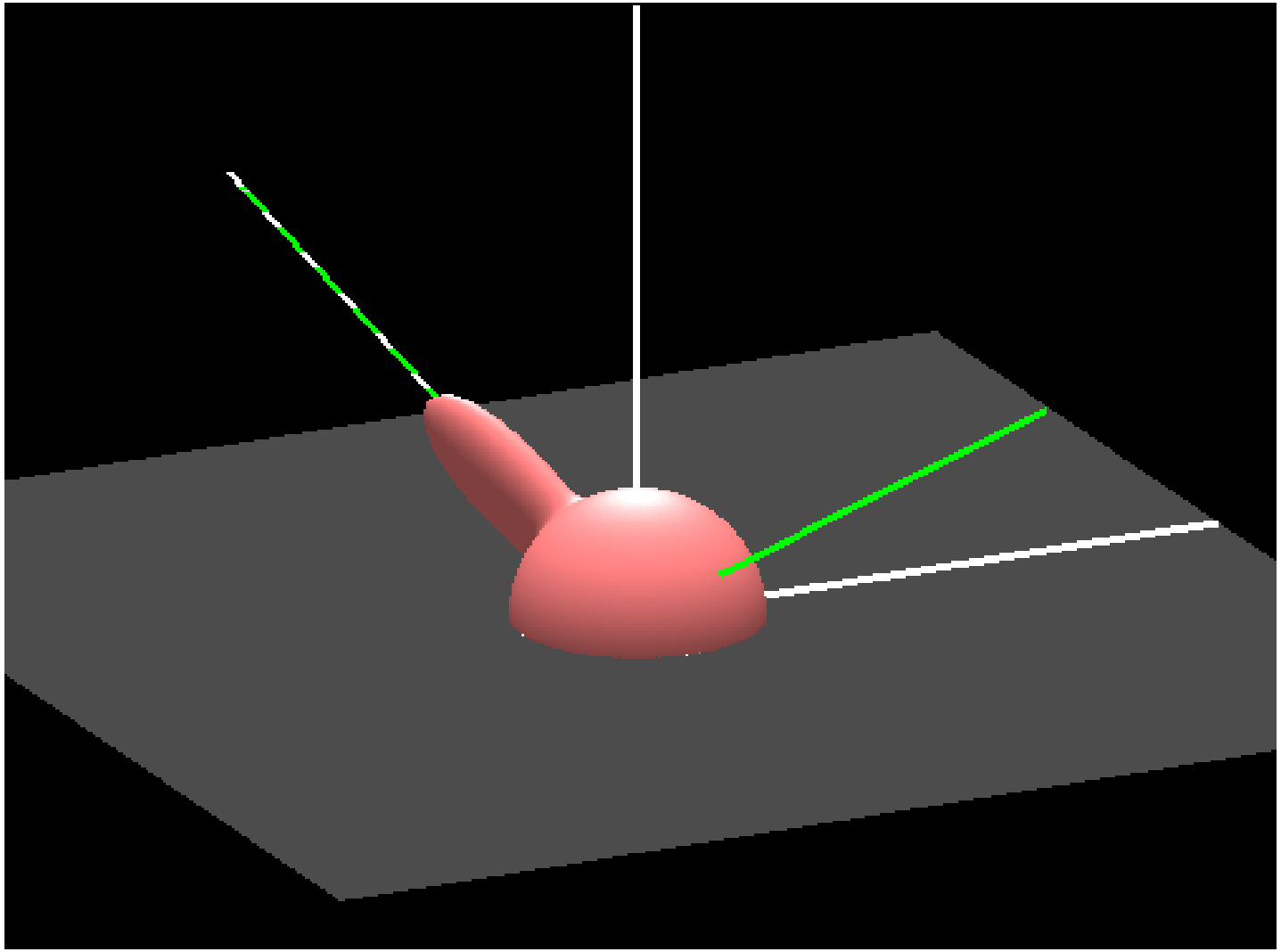
Двухлучевая функция отражательной способности

Функция $I(\varphi_{out}, \theta_{out}) = brdf(\varphi_{in}, \theta_{in}, \varphi_{out}, \theta_{out}) \cdot I(\varphi_{in}, \theta_{in})$



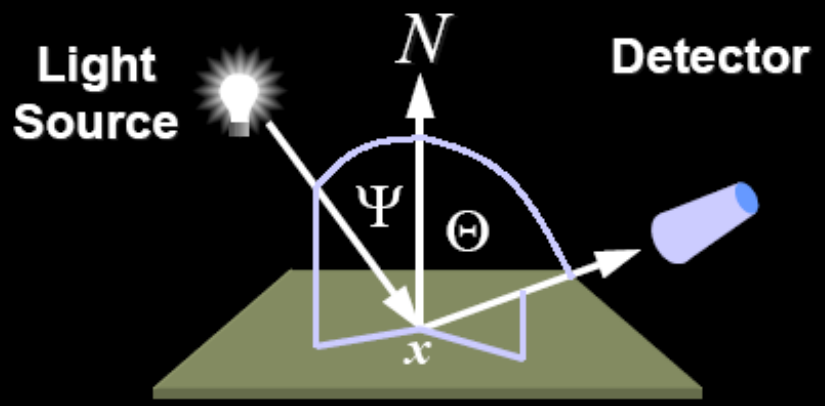


BRDF



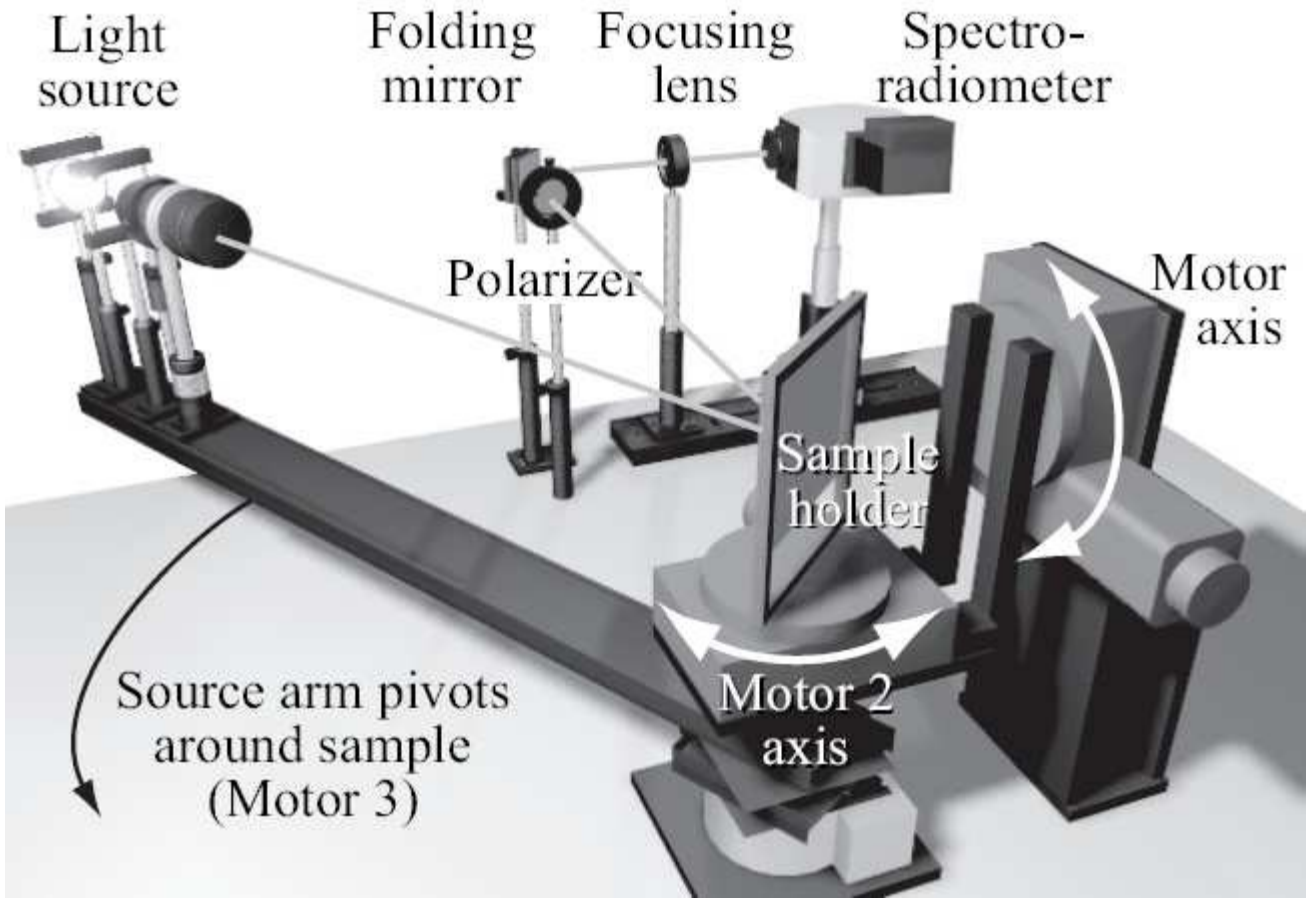
измерения BRDF

- Bidirectional Reflectance Distribution Function



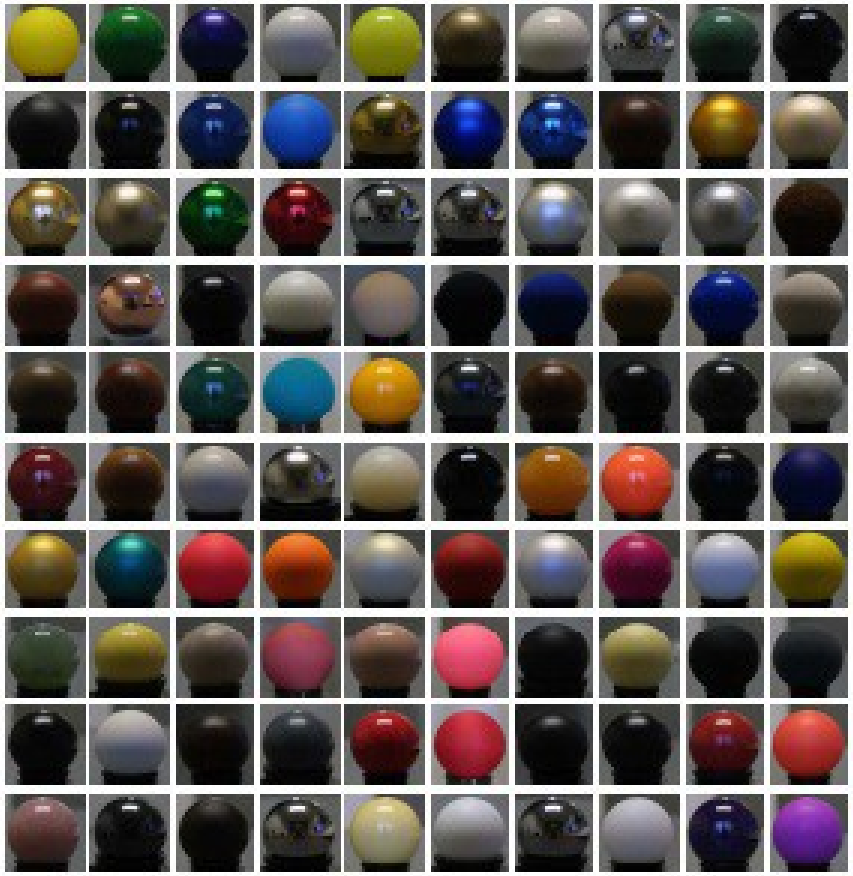
$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} = \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi}$$

Измерения BRDF



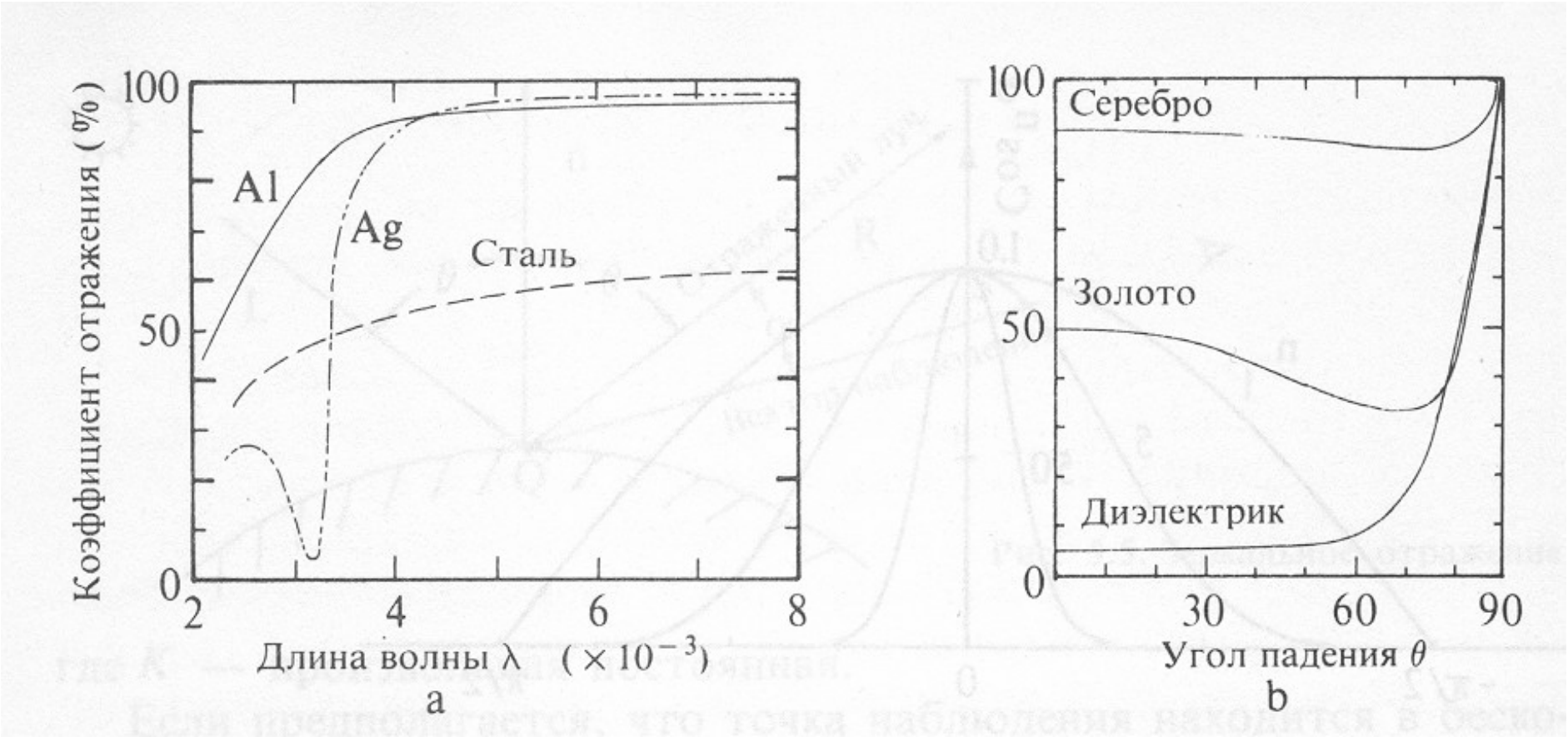
База данных MERL BRDF

MITSUBISHI ELECTRIC RESEARCH LABORATORIES



100
измеренных
материалов

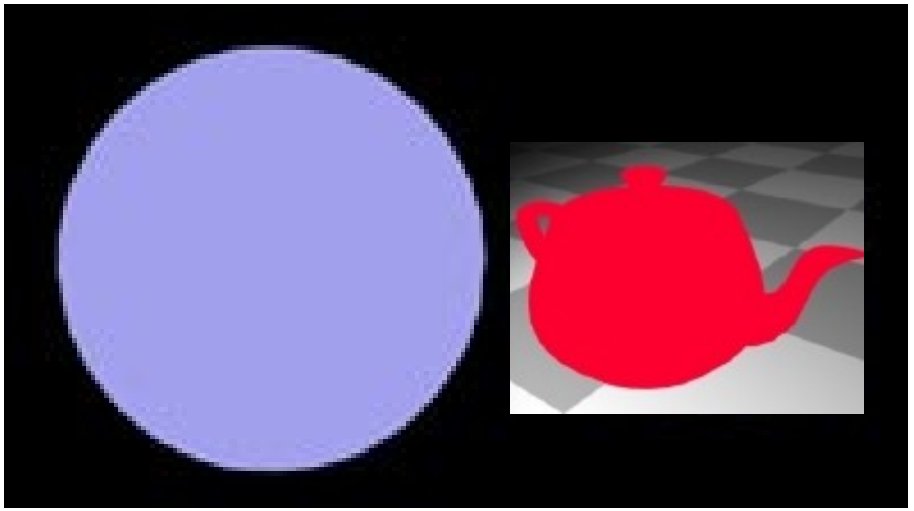
Зеркальное отражение



Рассеянный свет и диффузное отражение

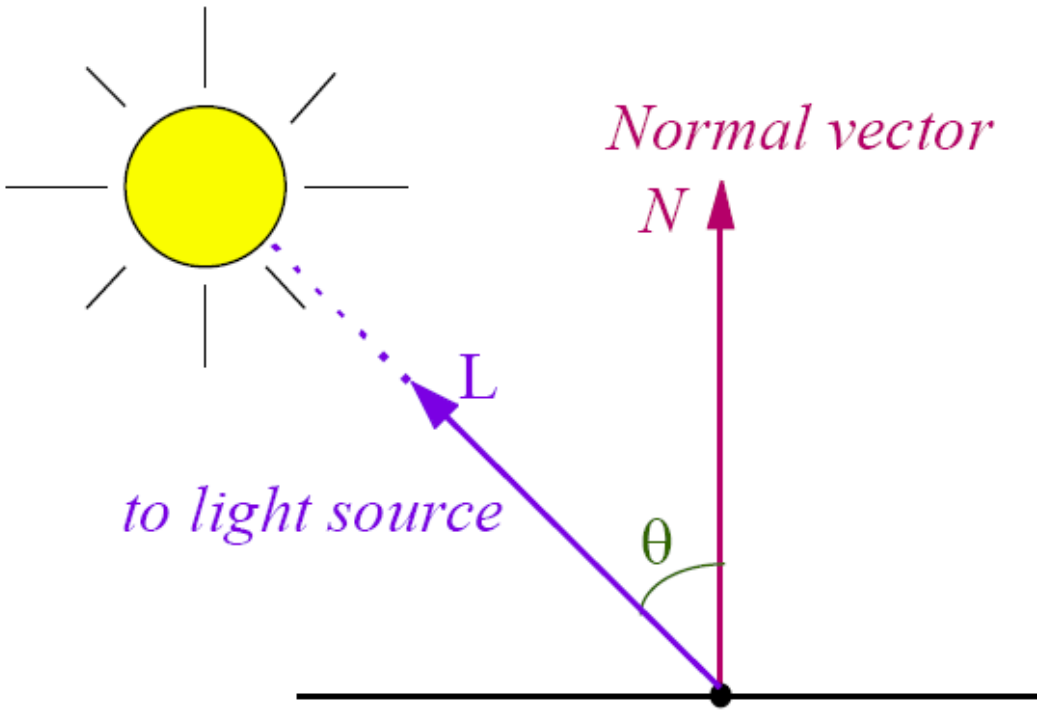
$$I = k_a \cdot I_A$$

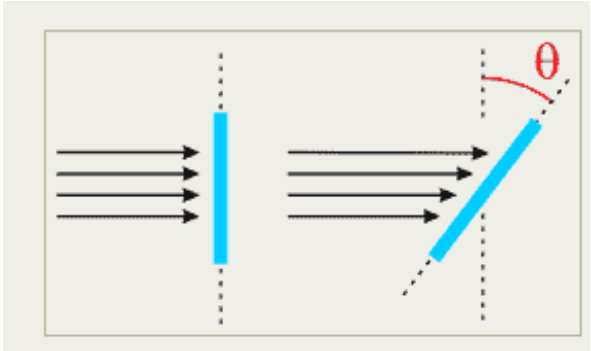
ambient light

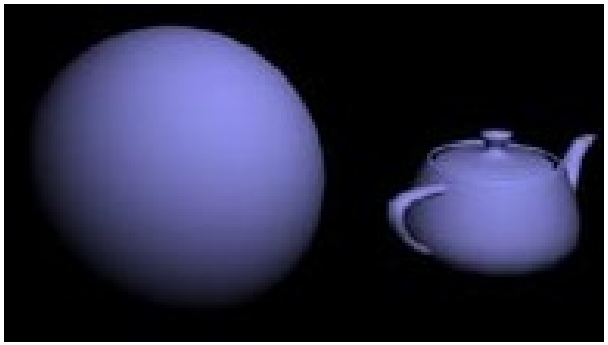


Увеличение k_a

Рассеянный свет и диффузное отражение







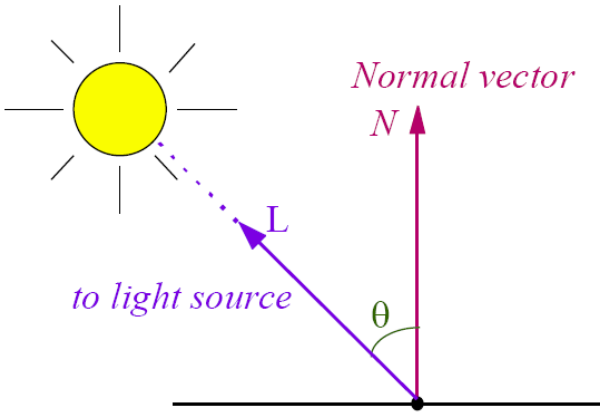
$$I = k_d \cdot \cos\theta \cdot I_L$$

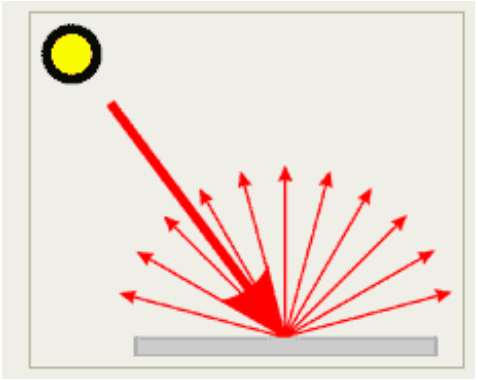
diffuse light

Рассеянный свет и диффузное отражение

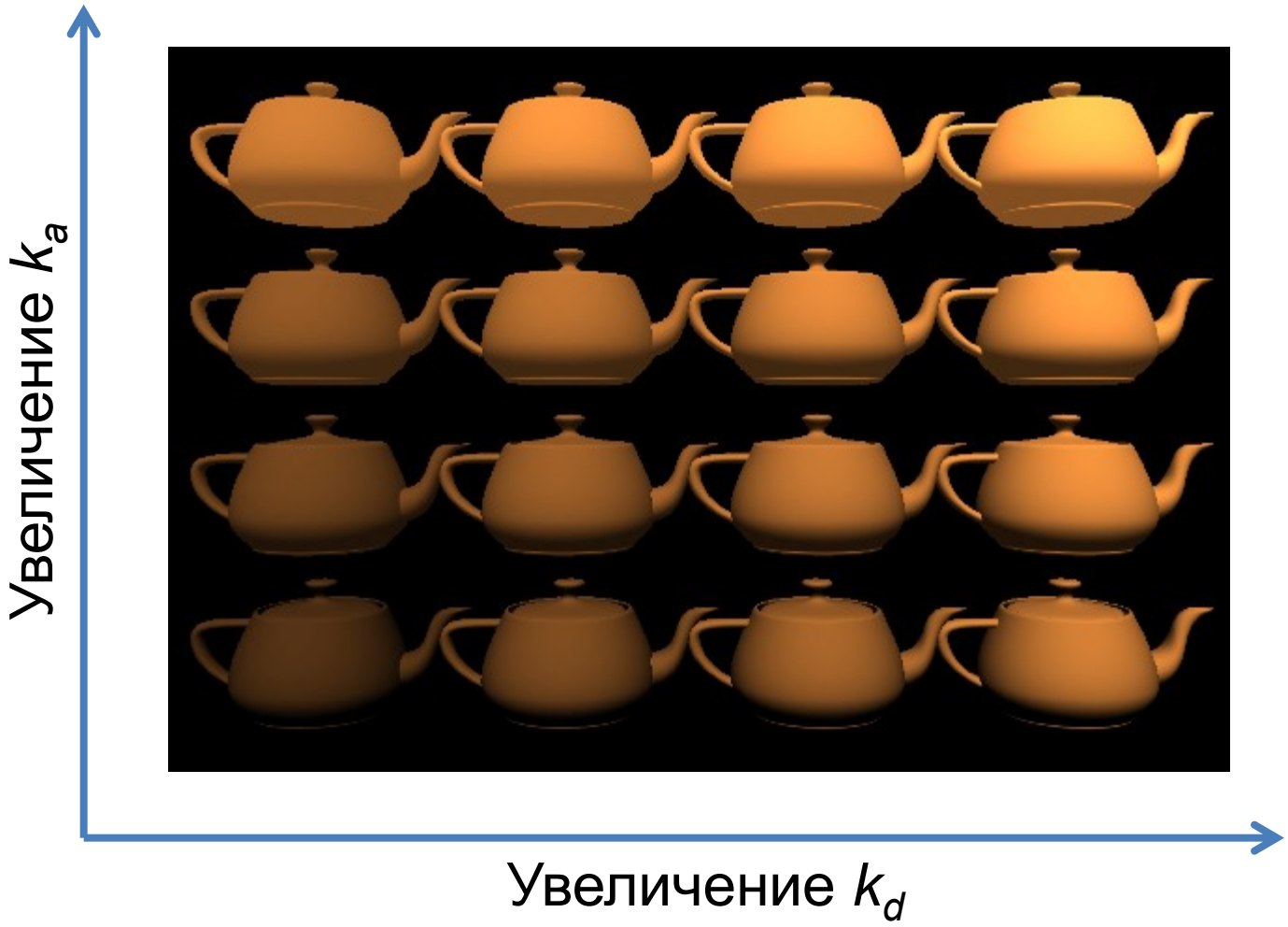
$$I = k_d \cdot \cos\theta \cdot I_L + k_a \cdot I_A$$

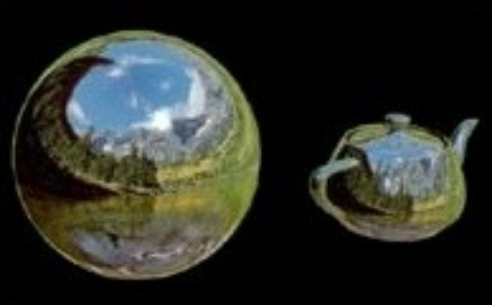
diffuse light ambient light



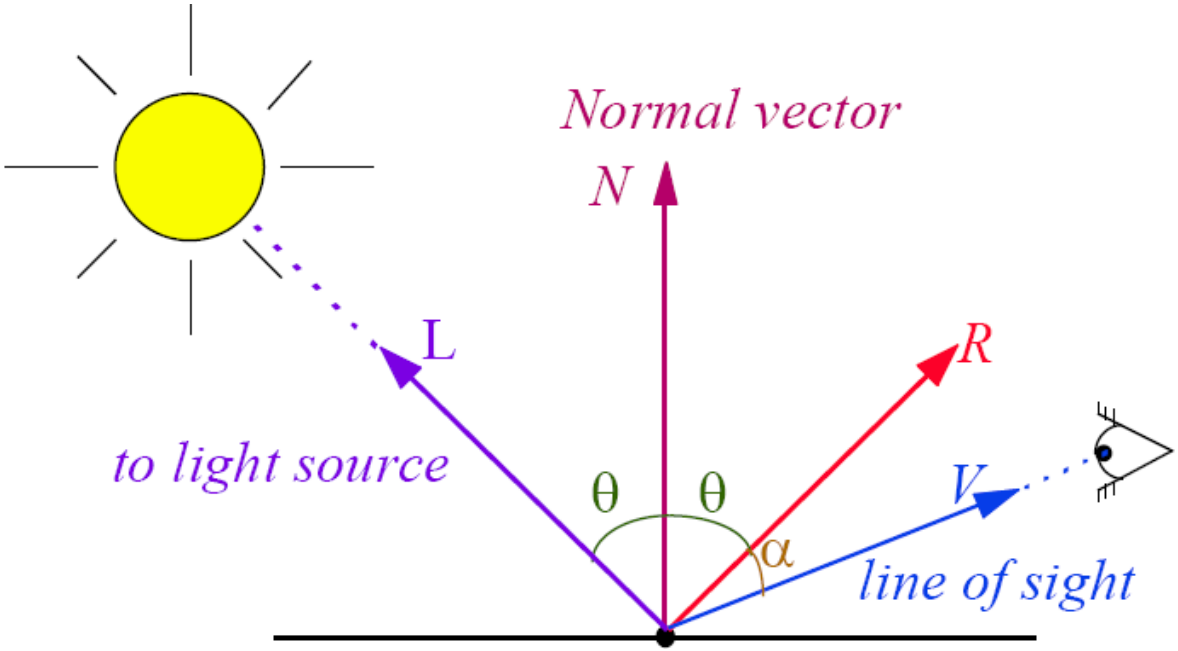


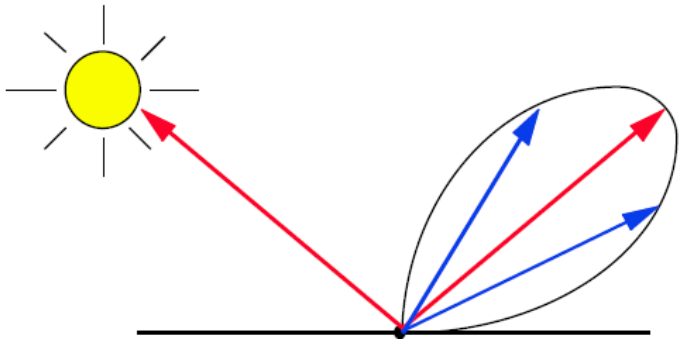
Рассеянный свет и диффузное отражение

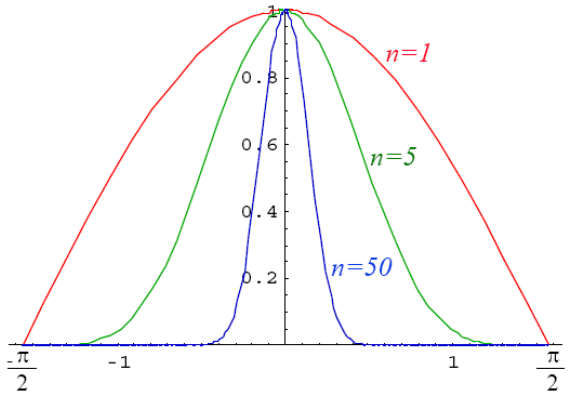




Зеркальное отражение



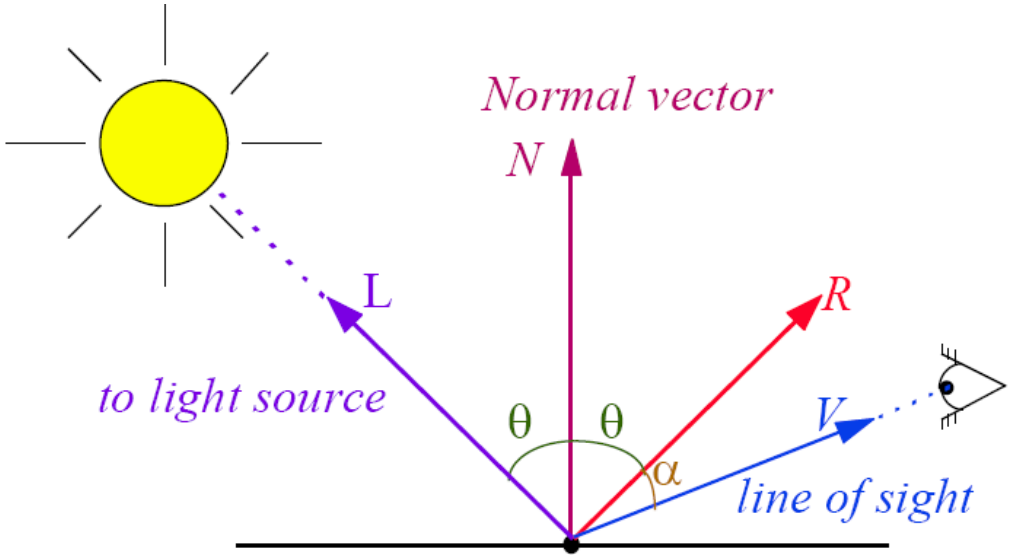




$$I = k_s \cdot \cos^{n} \alpha \cdot I_L$$

specular light

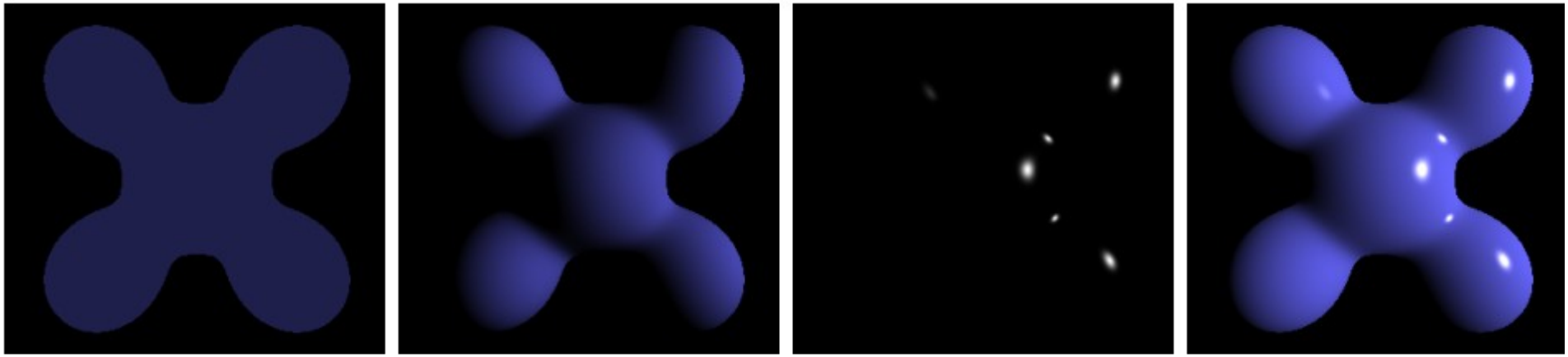
Модель полного отражения по Фонгу (Phong reflection model)

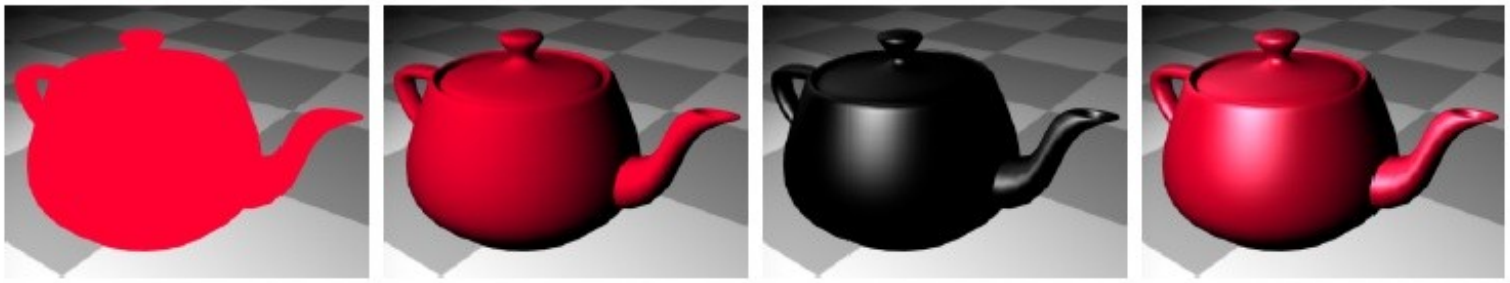


$$I = I_A \cdot k_a + I_L \cdot (k_d \cdot \cos\theta + k_s \cdot \cos^{\mathrm{n}}\alpha)$$

$$I = I_A \cdot k_a + I_L \cdot (k_d \cdot (N \cdot L) + k_s \cdot (R \cdot V)^n)$$

Модель полного отражения по Фонгу (Phong reflection model)

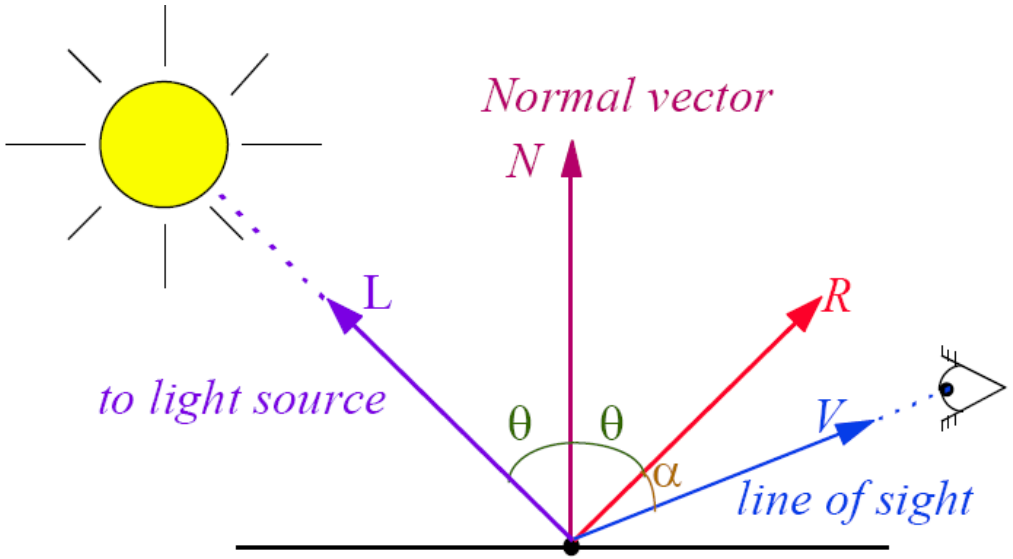




(a) ambient (b) diffuse (c) specular (d) diff.+spec.

Модель полного отражения по Фонгу (Phong reflection model)

$$I = \frac{I_{source}}{4\,\pi\,d^2}$$

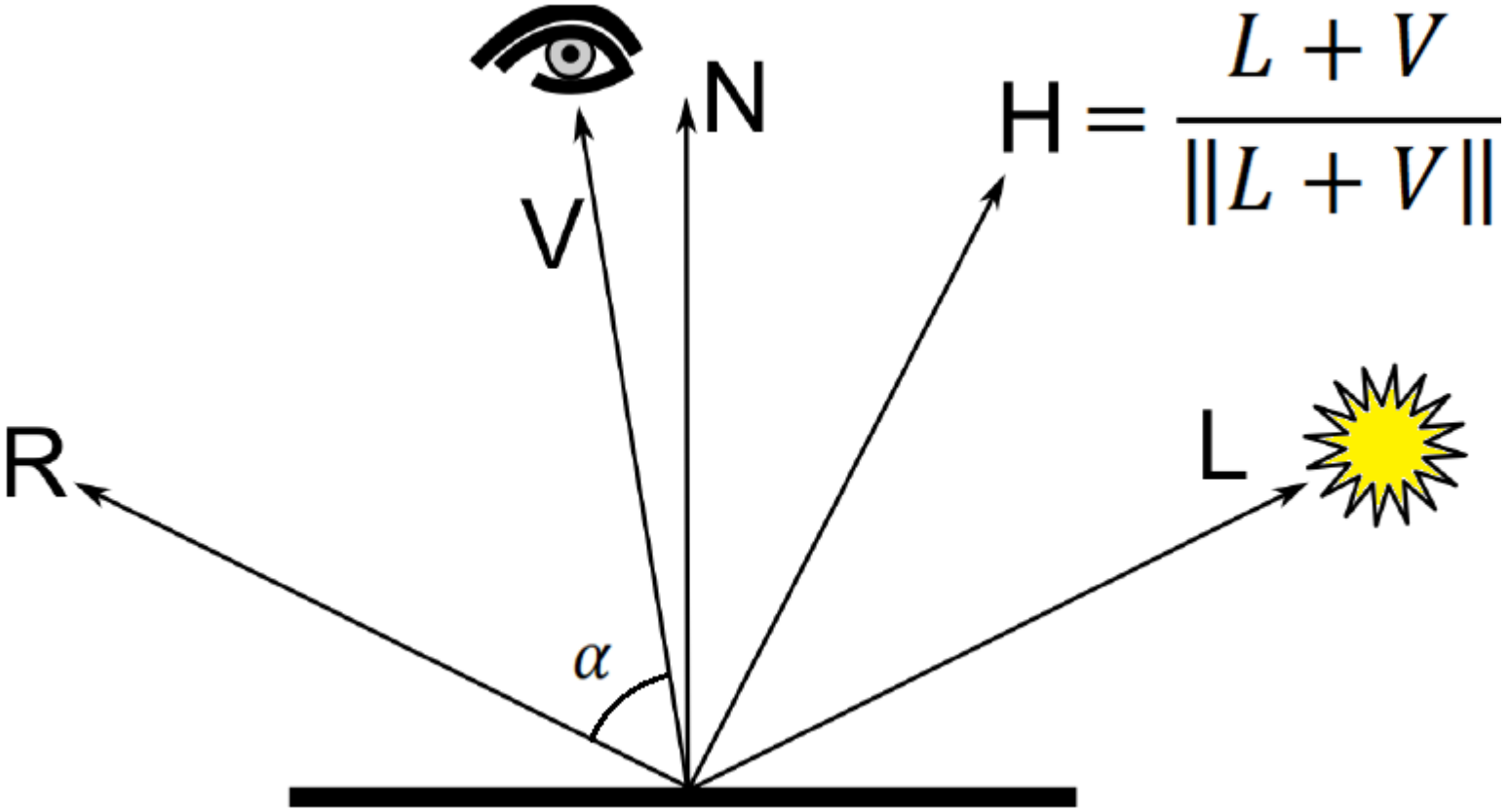


$$I = K_\alpha I_\alpha + \frac{I_l}{d^p + K}\left(K_d \cos(\theta) + K_s \cos^n(\alpha)\right)$$

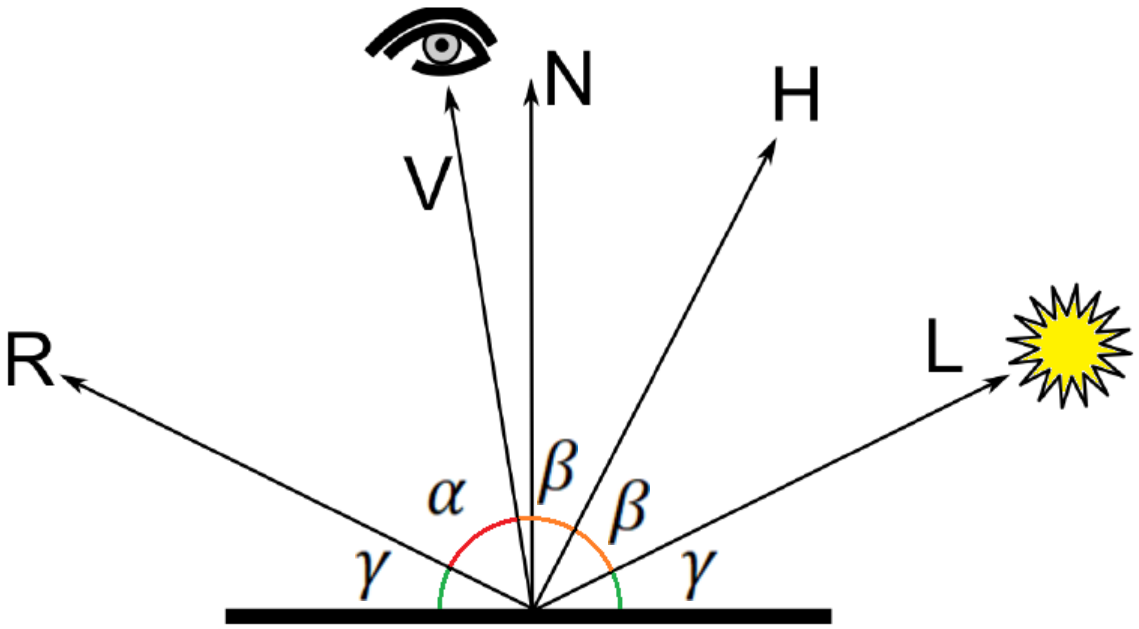
$$I = Ambient + attenuation\,(\,diffuse + specular\,)$$

$$I = K_\alpha I_\alpha + \frac{I_l}{d^p + K}\left(K_d\ (N.L) + K_s\ (R.V)^n\right)$$

Blinn-Phong reflection model



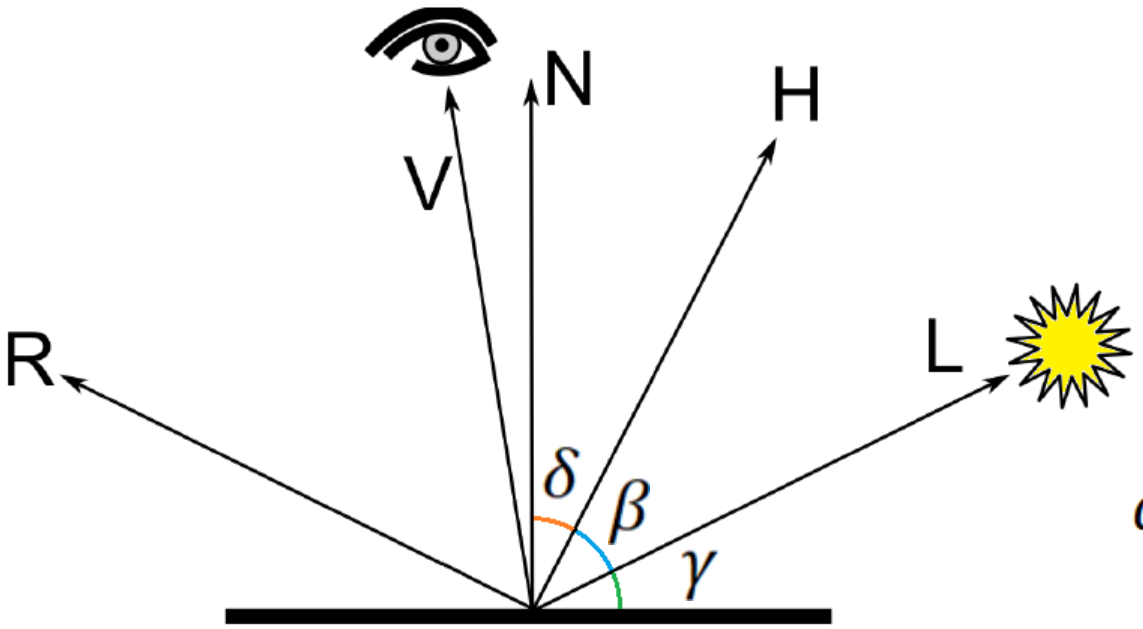
Blinn-Phong reflection model



$$H = \frac{L + V}{\|L + V\|}$$

$$\alpha + 2\beta + 2\gamma = 180°$$

Blinn-Phong reflection model



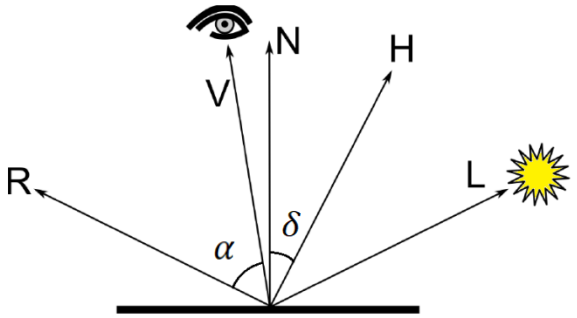
$$H = \frac{L + V}{\|L + V\|}$$

$$\alpha + 2\beta + 2\gamma = 180°$$

$$\delta + \beta + \gamma = 90°$$

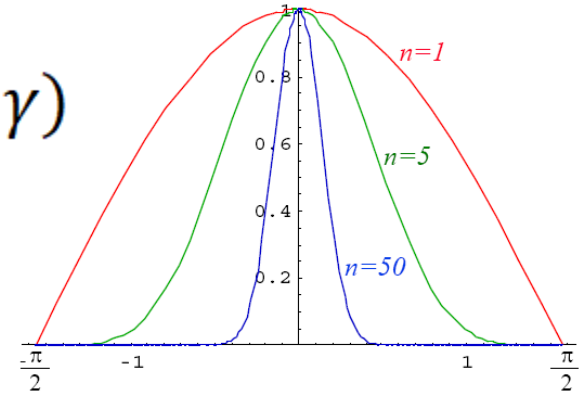
Blinn-Phong reflection model

$$\alpha + 2\beta + 2\gamma = 180°$$
$$\delta + \beta + \gamma = 90°$$



$$\delta = 90° - \beta - \gamma$$
$$\alpha = 180° - 2\beta + 2\gamma = 2 \cdot (90° - \beta - \gamma)$$

$$\alpha = 2\delta$$

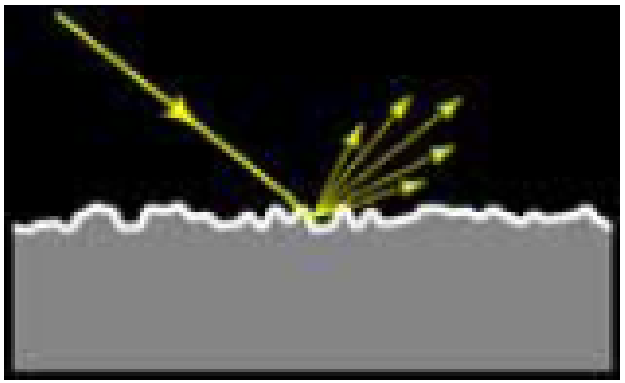


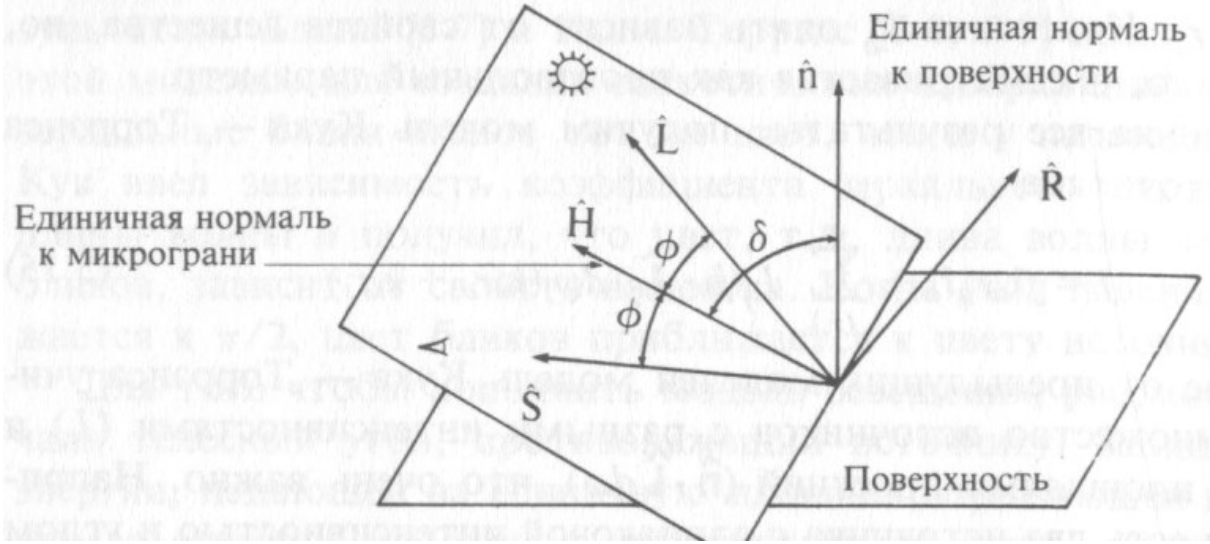
$$I = I_A \cdot k_a + I_L \cdot (k_d \cdot (N \cdot L) + k_s \cdot (N \cdot H)^n)$$
$$I = I_A \cdot k_a + I_L \cdot (k_d \cdot (N \cdot L) + k_s \cdot (R \cdot V)^n)$$

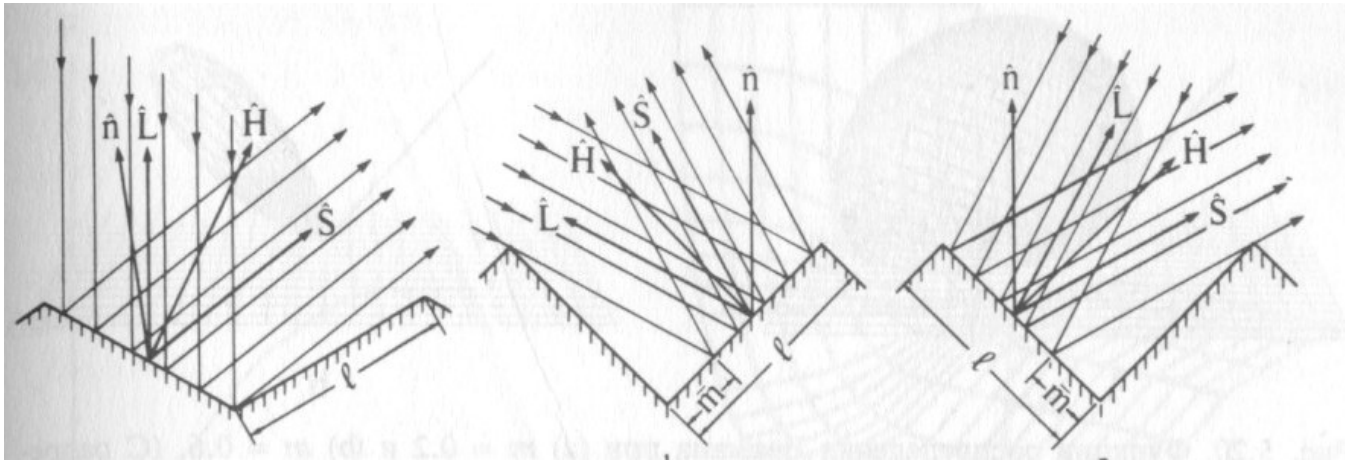
Торрэнс-Спэрроу
(распр.Гаусса, Бэкмана)

$$D = c_1 e^{-(\delta/m)^2},$$

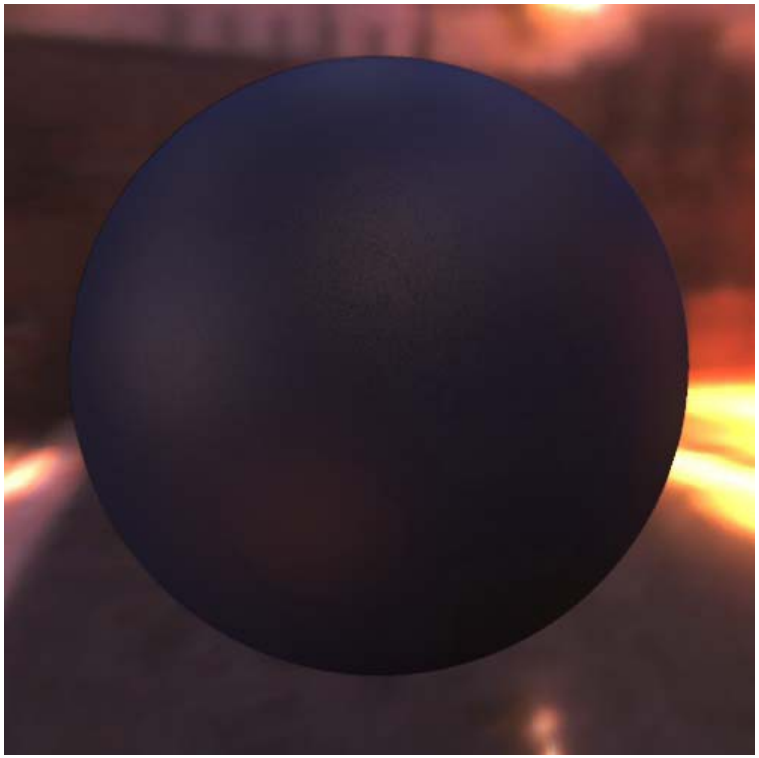
$$D = \frac{1}{m^2 \cos^4 \delta} e^{-(\mathrm{tg}\delta/m)^2},$$



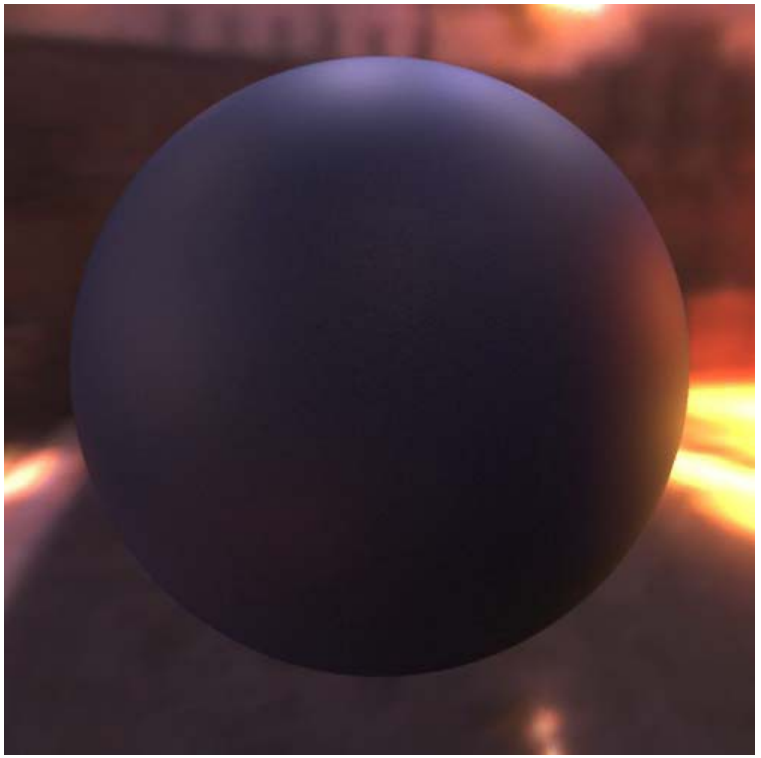




Торрэнс-Спэрроу

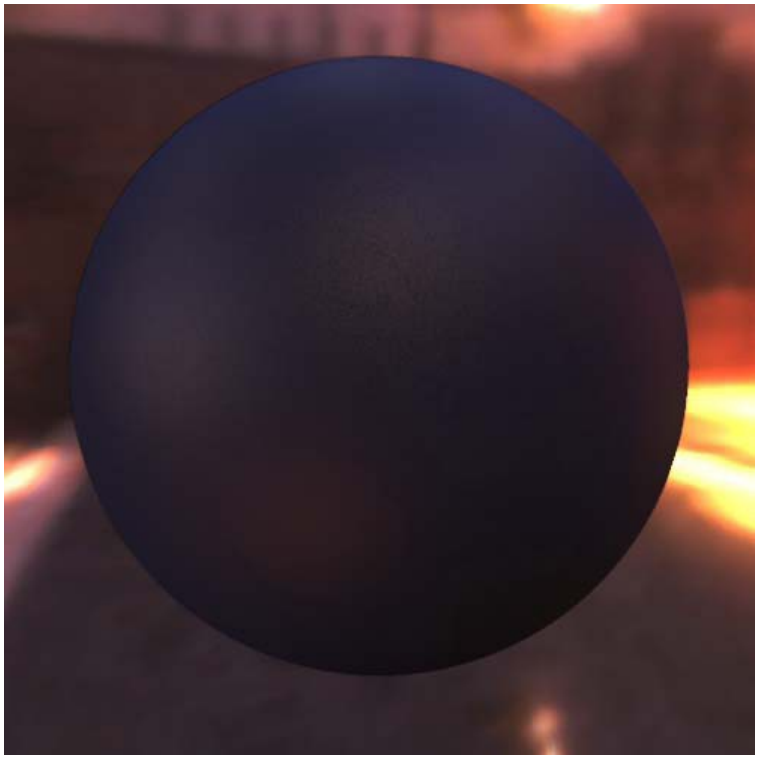


реальное изображение

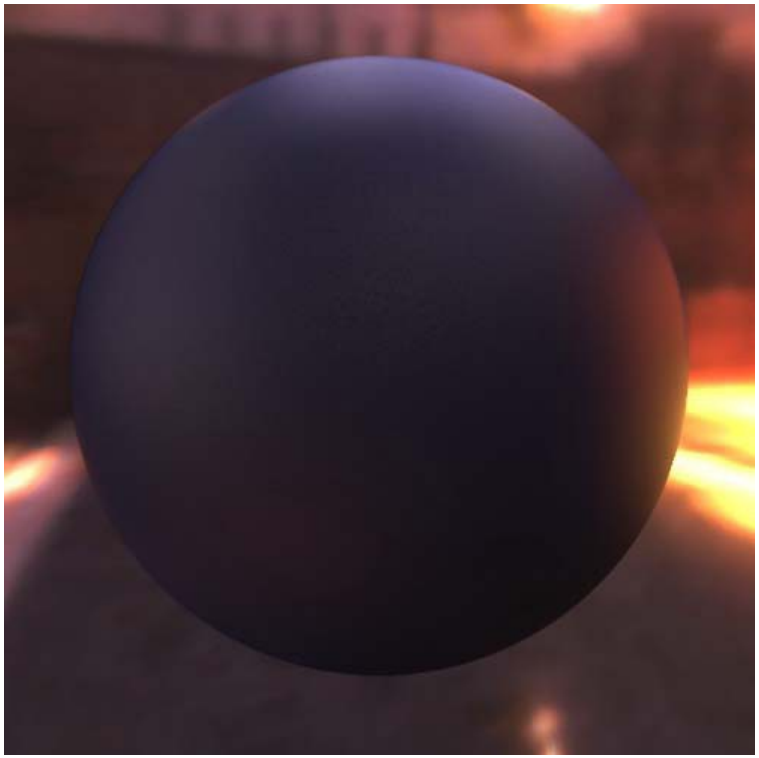


модель Фонга

Торрэнс-Спэрроу



реальное изображение



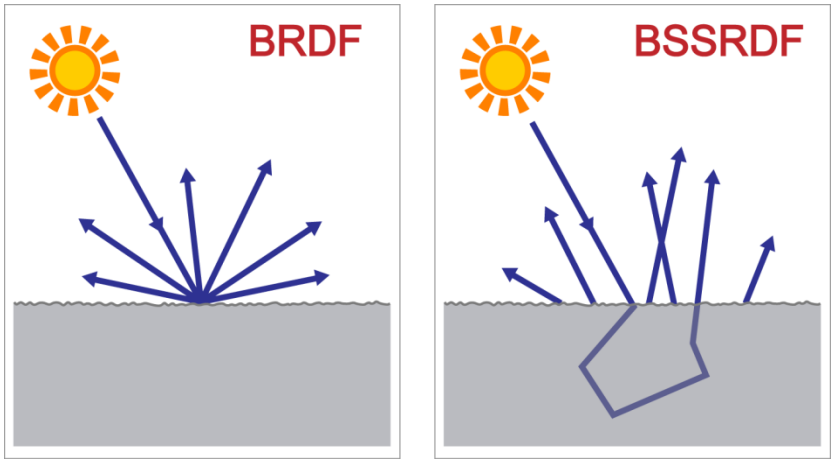
модель Торрэнса

Текстурирование BRDF

Одинаковая BRDF

BRDF в каждой точке

BSSRDF : Bidirectional surface scattering reflectance distribution function



- 8D function
- Subsurface Scattering

Ravi Ramamoorthi

BSSRDF : Bidirectional surface scattering reflectance distribution function

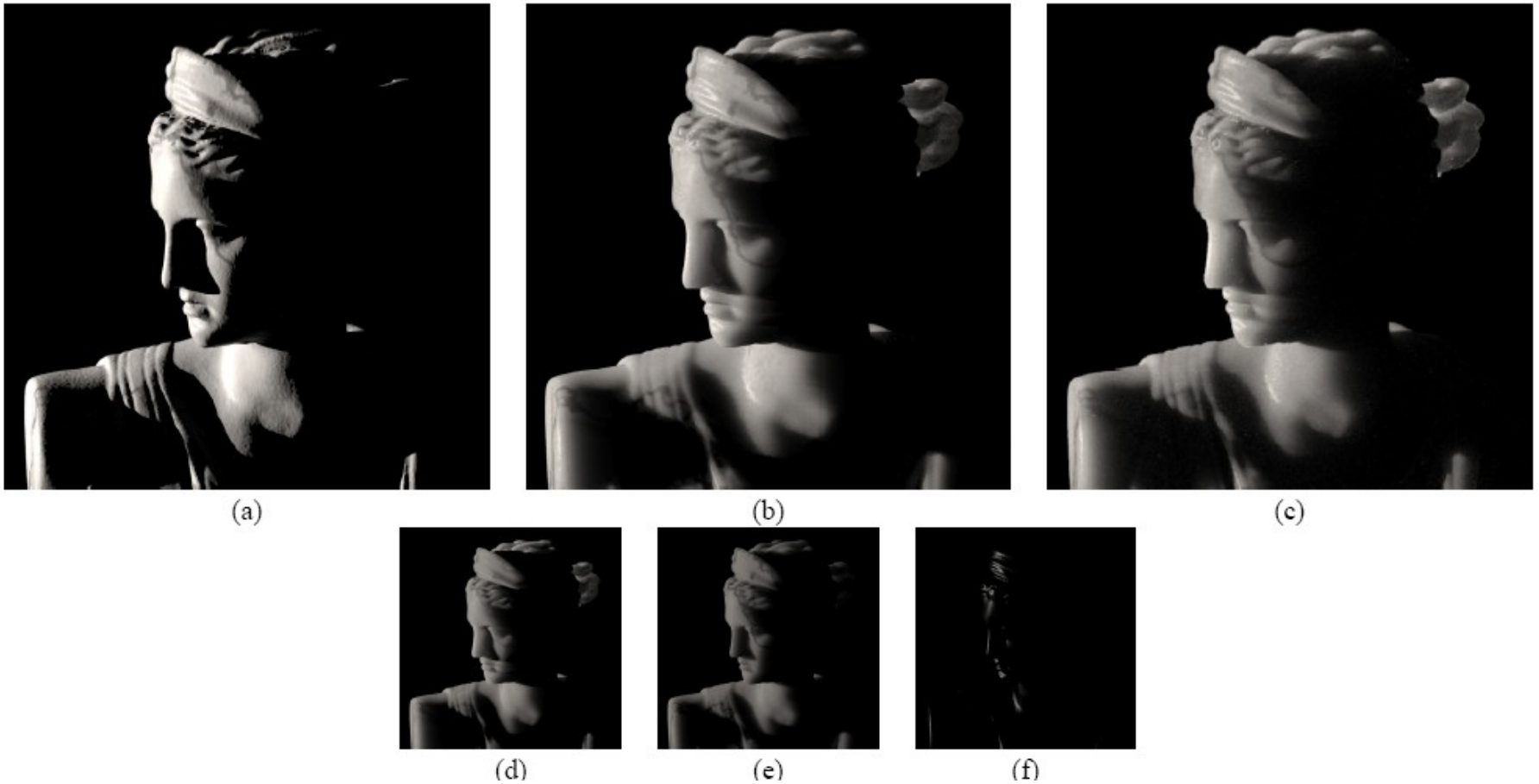
BRDF

BSSRDF

BSSRDF : Bidirectional surface scattering reflectance distribution function

BRDF

BSSRDF



A simulation of subsurface scattering in a marble bust. illuminated from behind and rendered using: (a) BRDF approximation (2 m), (b) BSSRDF approximation (5 m), and (c) a full Monte Carlo simulation (in 1250 minutes).
Notice how the BSSRDF model matches the appearance of the Monte Carlo simulation, yet is significantly faster. The images in (d–f) show
the different components of the BSSRDF: (d) single scattering term, (e) diffusion term, and (f) Fresnel term.

Компьютерная Графика

Освещение

Lights and Materials

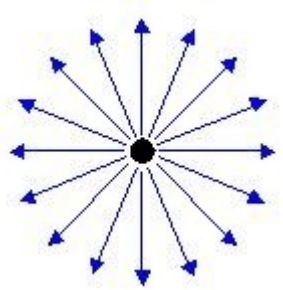
Источники света (**Lights**) освещают объекты сцены

DirectX подчитывает цвет каждой вершины объекта, исходя из:

- цвет вершины - Materials
- цвет источника света - Lights

Lights and Materials

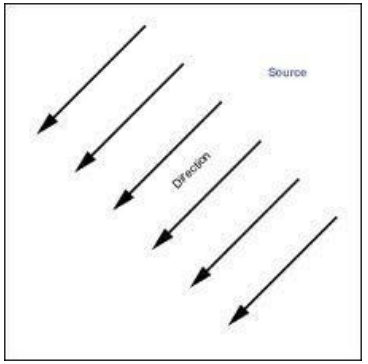
Point Light - аналог лампы накаливания



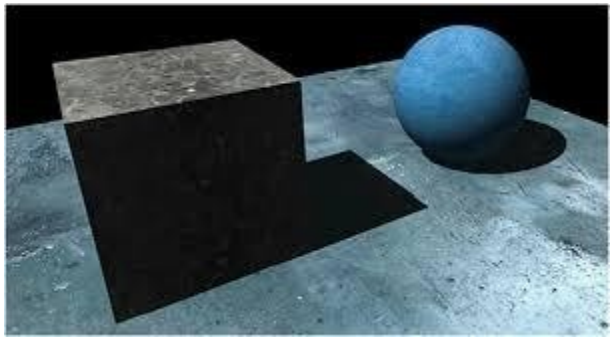
- позиция
- цвет
- направление - все стороны
- затухает на расстоянии

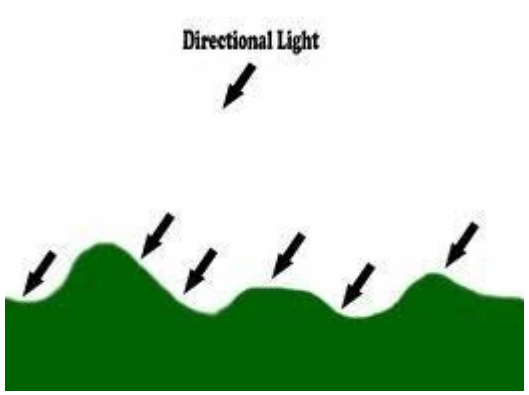
Lights and Materials

Directional Light - аналог солнца днём (на Земле)



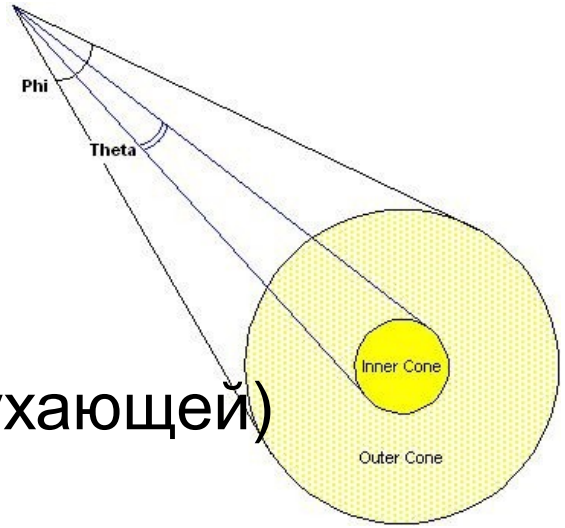
- позиции нет
- цвет
- направление - лучи параллельны
- не затухает на расстоянии



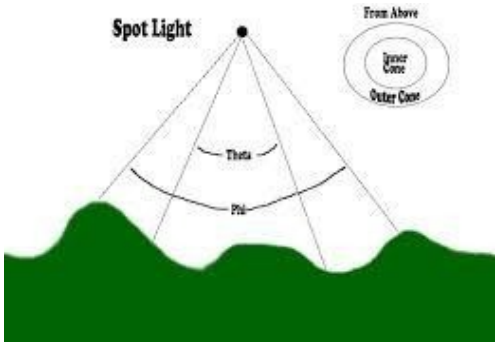


Lights and Materials

Spot Light - аналог фонарика или прожектора



- позиция
- цвет
- направление - два конуса (постоянной интенсивности и затухающей)
- затухает на расстоянии



Lights and Materials

Materials

Материалы задают то, как объекты отражают свет

Цвет каждого пиксела на экране состоит из:

- ambient
- diffuse
- specular
- emissive
- reflection
- ...

Lights and Materials

Составляющая Ambient

это постоянная составляющая цвета

Обычно используется эта формула:

Ambient Lighting = Ga

Ga - Global ambient color

Например, когда небо пасмурное днём, все объекты всё равно освещены на улице

Lights and Materials

Составляющая Ambient

более общая формула:

Ambient Lighting = Ca*[Ga + sum(Atteni*Spoti*Lai)]

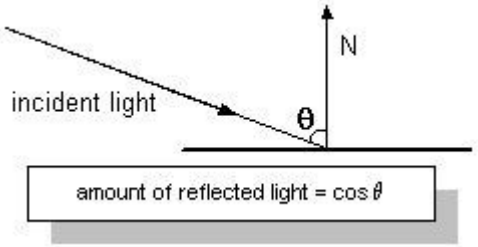
Parameter	Default value	Type	Description
C_a	(0,0,0,0)	D3DCOLORVALUE	Material ambient color
G_a	(0,0,0,0)	D3DCOLORVALUE	Global ambient color
$Atten_i$	(0,0,0,0)	D3DCOLORVALUE	Light attenuation of the ith light. See **Attenuation and Spotlight Factor (Direct3D 9)**.
$Spot_i$	(0,0,0,0)	D3DVECTOR	Spotlight factor of the ith light. See **Attenuation and Spotlight Factor (Direct3D 9)**.
sum	N/A	N/A	Sum of the ambient light
L_{ai}	(0,0,0,0)	D3DVECTOR	Light ambient color of the ith light

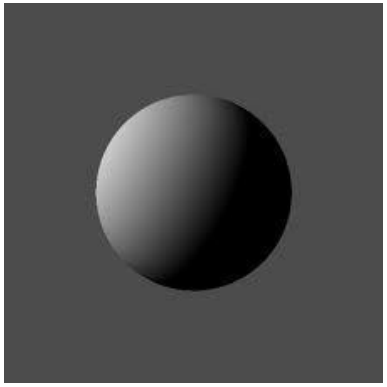
Lights and Materials

Составляющая Diffuse

это составляющая цвета зависит от нормали

Diffuse Lighting = (N*Ldir)





N - Vertex normal

Ldir - Direction vector from object vertex to the light

diffuse

\+

ambient

=

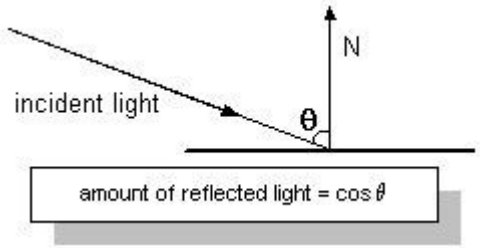
diffuse + ambient

Lights and Materials

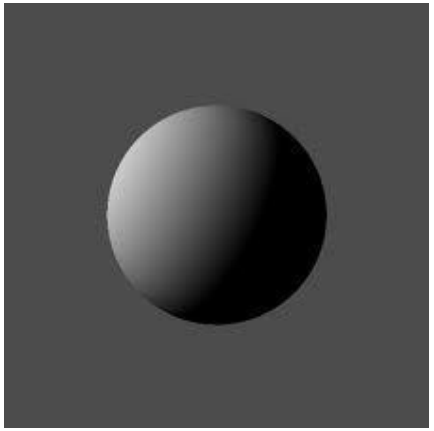
Составляющая Diffuse

более общая формула:

Diffuse Lighting = sum[Cd*Ld*(N.Ldir)*Atten*Spot]



Parameter	Default value	Type	Description
sum	N/A	N/A	Summation of each light's diffuse component.
C_d	(0,0,0,0)	D3DCOLORVALUE	Diffuse color.
L_d	(0,0,0,0)	D3DCOLORVALUE	Light diffuse color.
N	N/A	D3DVECTOR	Vertex normal
L_{dir}	N/A	D3DVECTOR	Direction vector from object vertex to the light.
Atten	N/A	FLOAT	Light attenuation. See **Attenuation and Spotlight Factor (Direct3D 9)**.
Spot	N/A	FLOAT	Spotlight factor. See **Attenuation and Spotlight Factor (Direct3D 9)**.

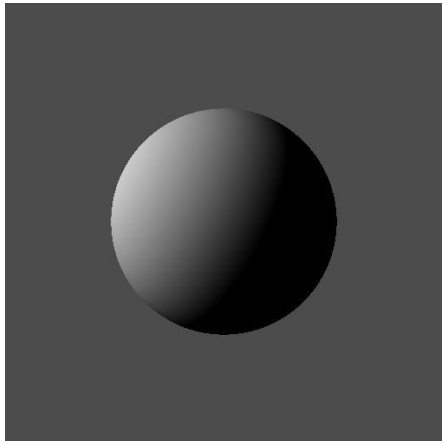


Lights and Materials

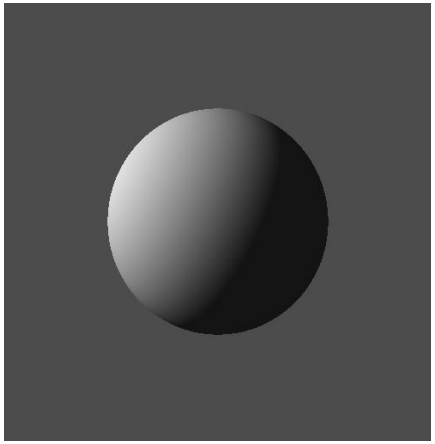
Составляющая Specular

это просто блик

Specular Lighting = $(N \cdot H)^P$



diffuse



diffuse + ambient

N - Vertex normal

H - Half way vector

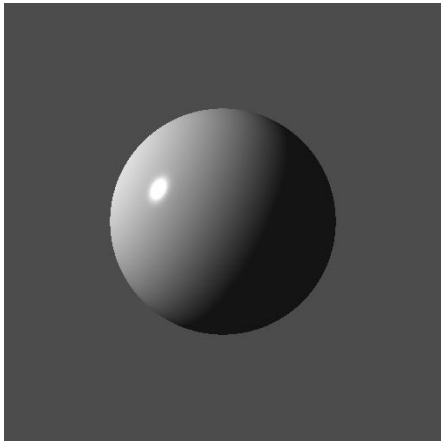
P - Specular reflection power. Range is 0 to +infinity

H = norm(norm(Cp - Vp) + Ldir)

Cp - Camera position

Vp - Vertex position

Ldir - Direction vector from vertex position to the light position



diffuse + ambient + specular

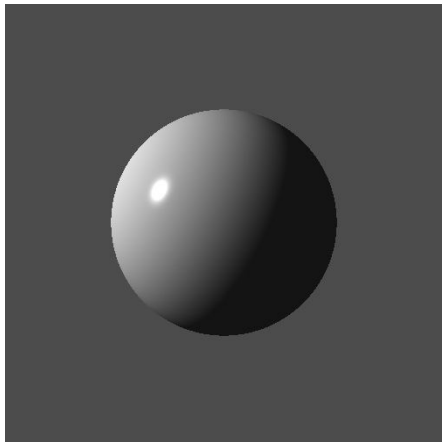
Lights and Materials

Составляющая Specular

это просто блик. Общая формула:

Specular Lighting = Cs * sum[Ls * $(N \cdot H)^P$ * Atten * Spot]

Parameter	Default value	Type	Description
C_s	(0,0,0,0)	D3DCOLORVALUE	Specular color.
sum	N/A	N/A	Summation of each light's specular component.
N	N/A	D3DVECTOR	Vertex normal.
H	N/A	D3DVECTOR	Half way vector. See the section on the halfway vector.
P	0.0	FLOAT	Specular reflection power. Range is 0 to +infinity
L_s	(0,0,0,0)	D3DCOLORVALUE	Light specular color.
Atten	N/A	FLOAT	Light attenuation value. See **Attenuation and Spotlight Factor (Direct3D 9)**.
Spot	N/A	FLOAT	Spotlight factor. See **Attenuation and Spotlight Factor (Direct3D 9)**.



diffuse + ambient + specular

Lights and Materials

Attenuation (коэффициент затухания)

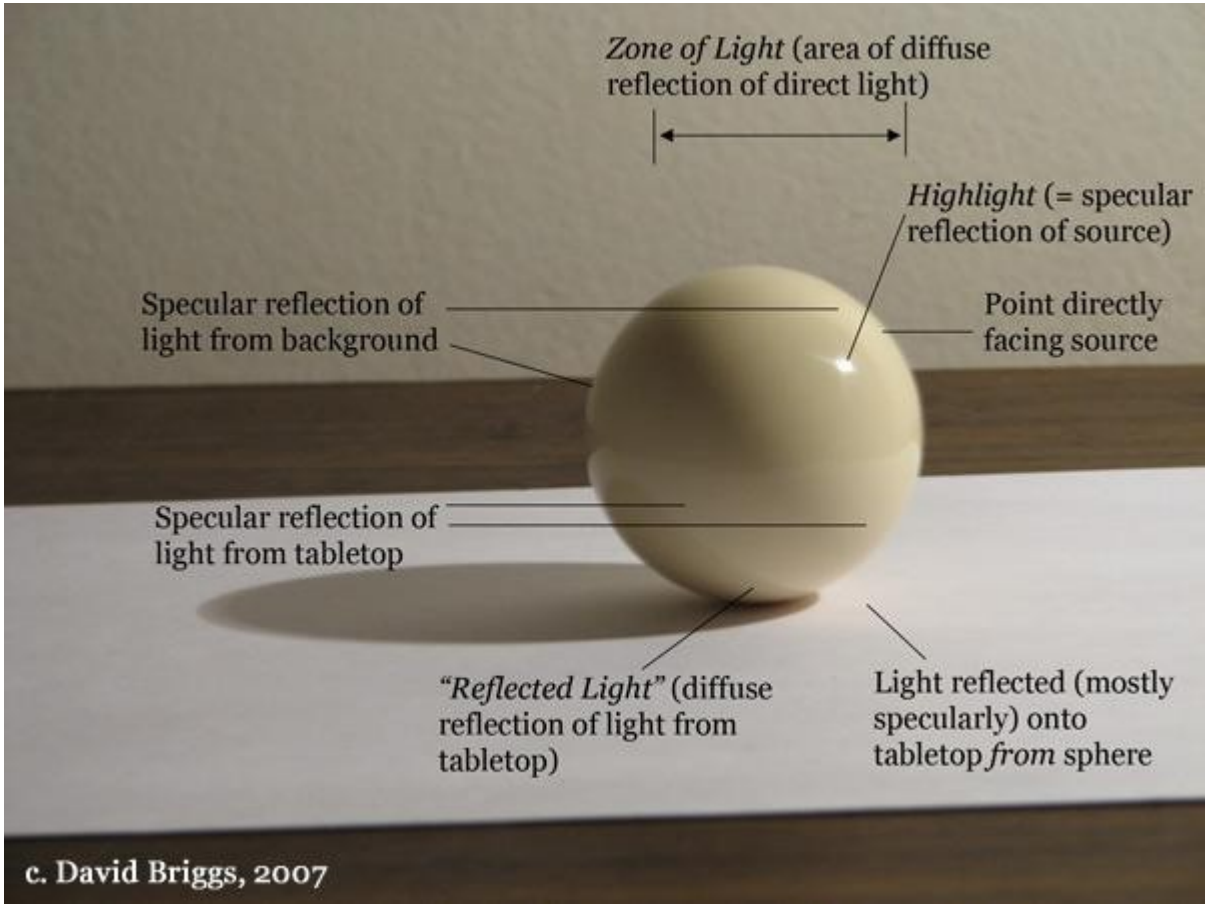
Atten = 1/(att0i + att1i * d + att2i * d2)

Parameter	Default value	Type	Description	Range
$att0_i$	0.0	FLOAT	Constant attenuation factor	0 to +infinity
$att1_i$	0.0	FLOAT	Linear attenuation factor	0 to +infinity
$att2_i$	0.0	FLOAT	Quadratic attenuation factor	0 to +infinity
d	N/A	FLOAT	Distance from vertex position to light position	N/A

- Atten = 1, if the light is a directional light.
- Atten = 0, if the distance between the light and the vertex exceeds the light's range.

Lights and Materials

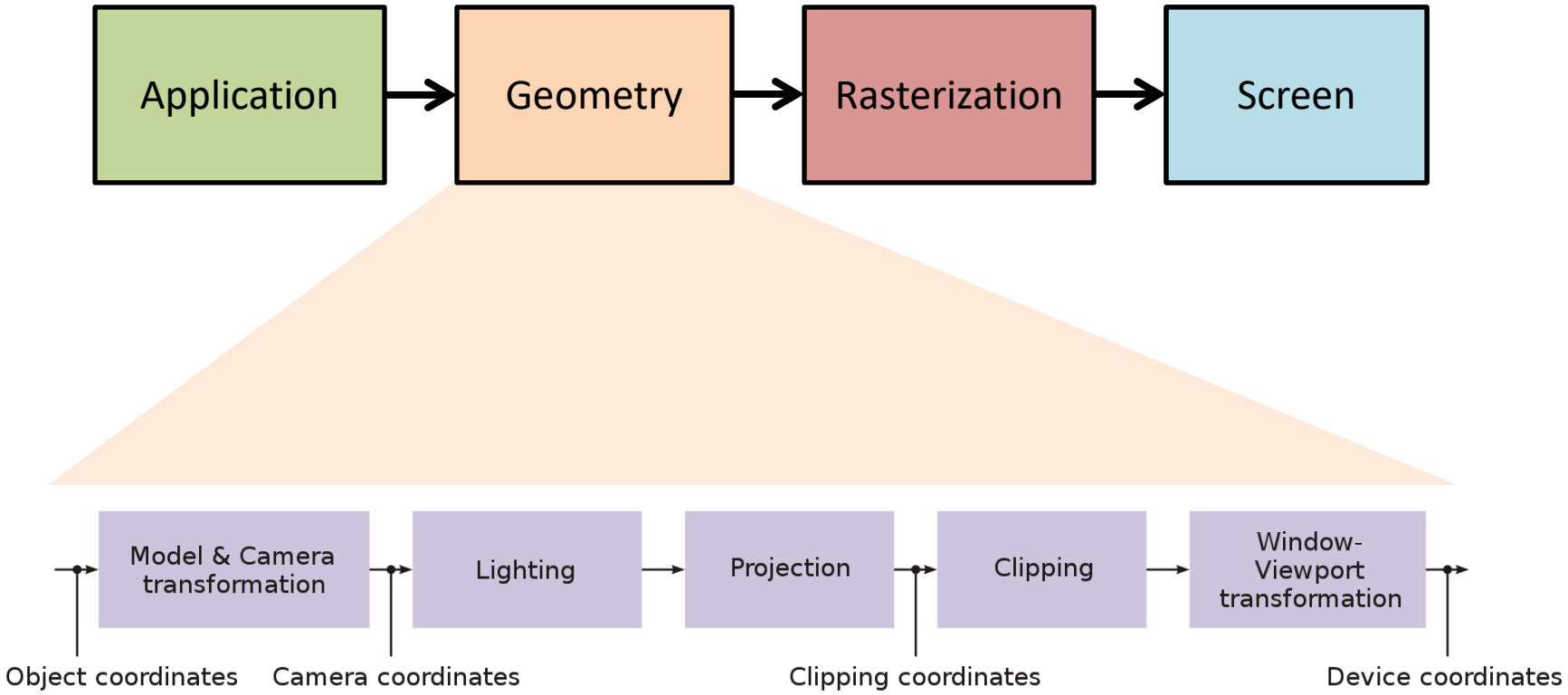
Классический пример для художников



Компьютерная Графика

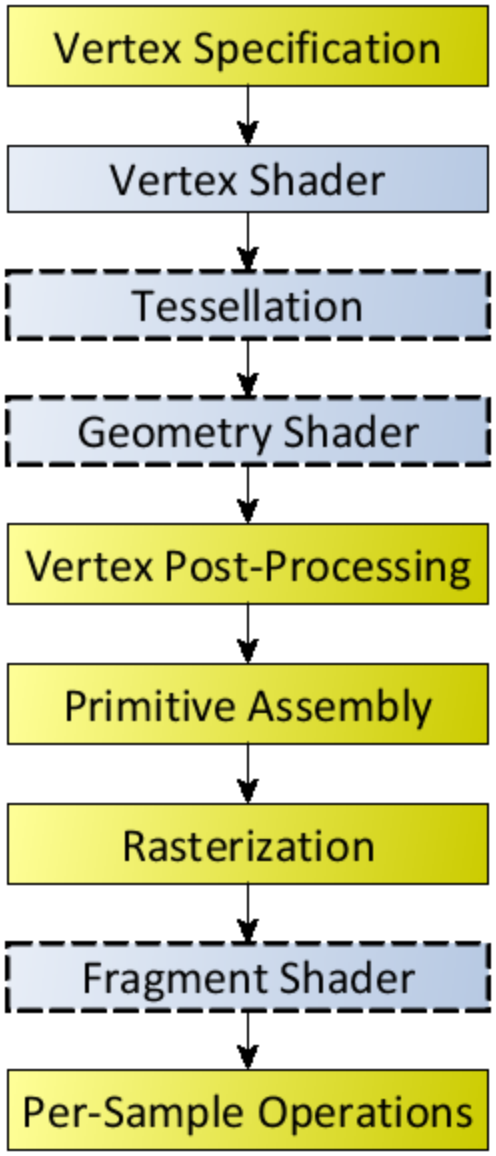
Vertex Shaders

Конвейер OpenGL



Конвейер OpenGL

1) Vertex Specification – задание геометрии объектов.
2) Vertex shader – обработка заданных вершин (в мировом пространстве).
3) - (опционально) -
4) - (опционально) -
5) Vertex post-processing – сохранение примитивов.
6) Primitive assembly – клипирование (отбрасывание ненужных частей), видовое и проективное преобразования (в пространство экрана).
7) Rasterization – разбиение на горизонтальные фрагменты, видимые на экране.
8) Fragment Shader – обработка фрагментов.
9) Per-Sample Operations – итоговое заполнение пикселей на экране.



Конвейер OpenGL

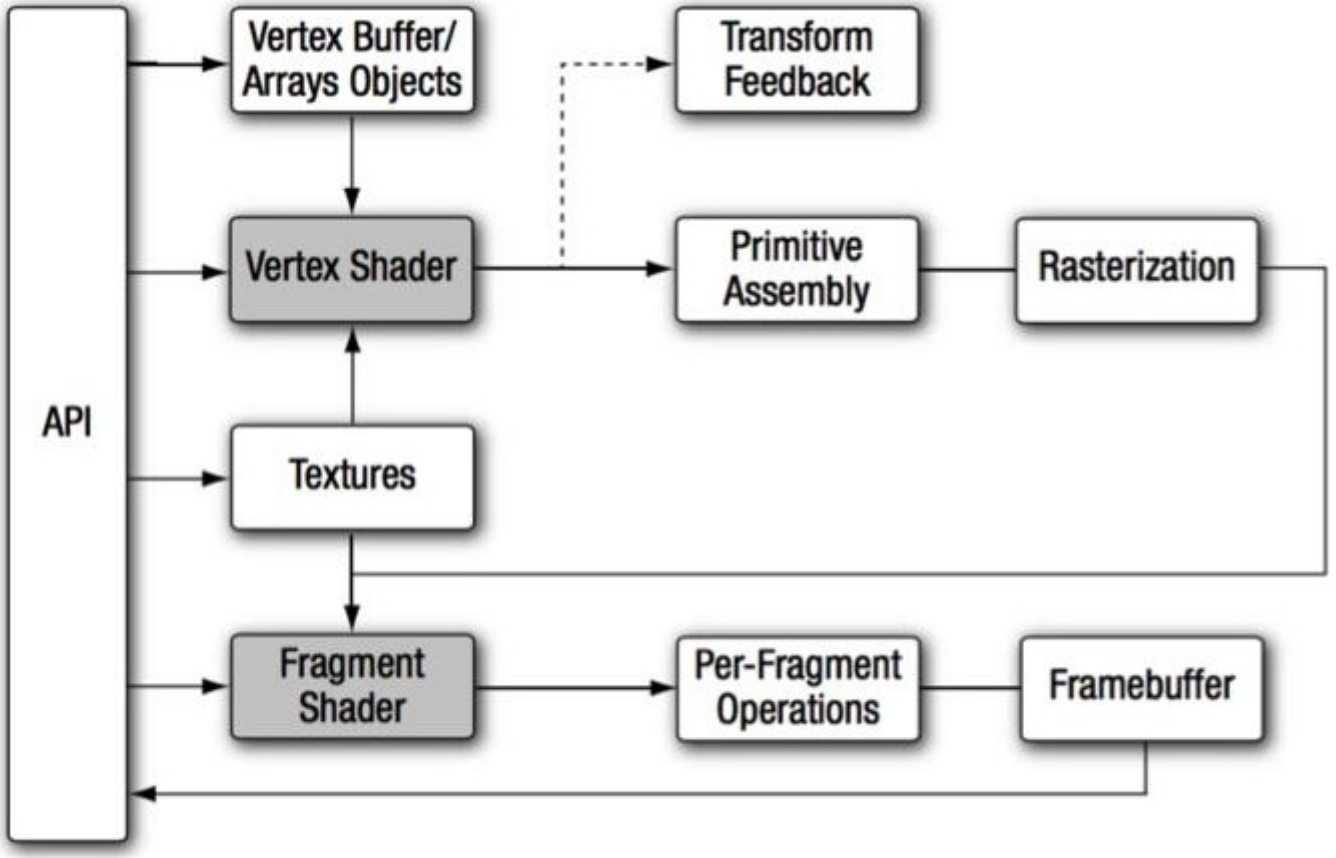
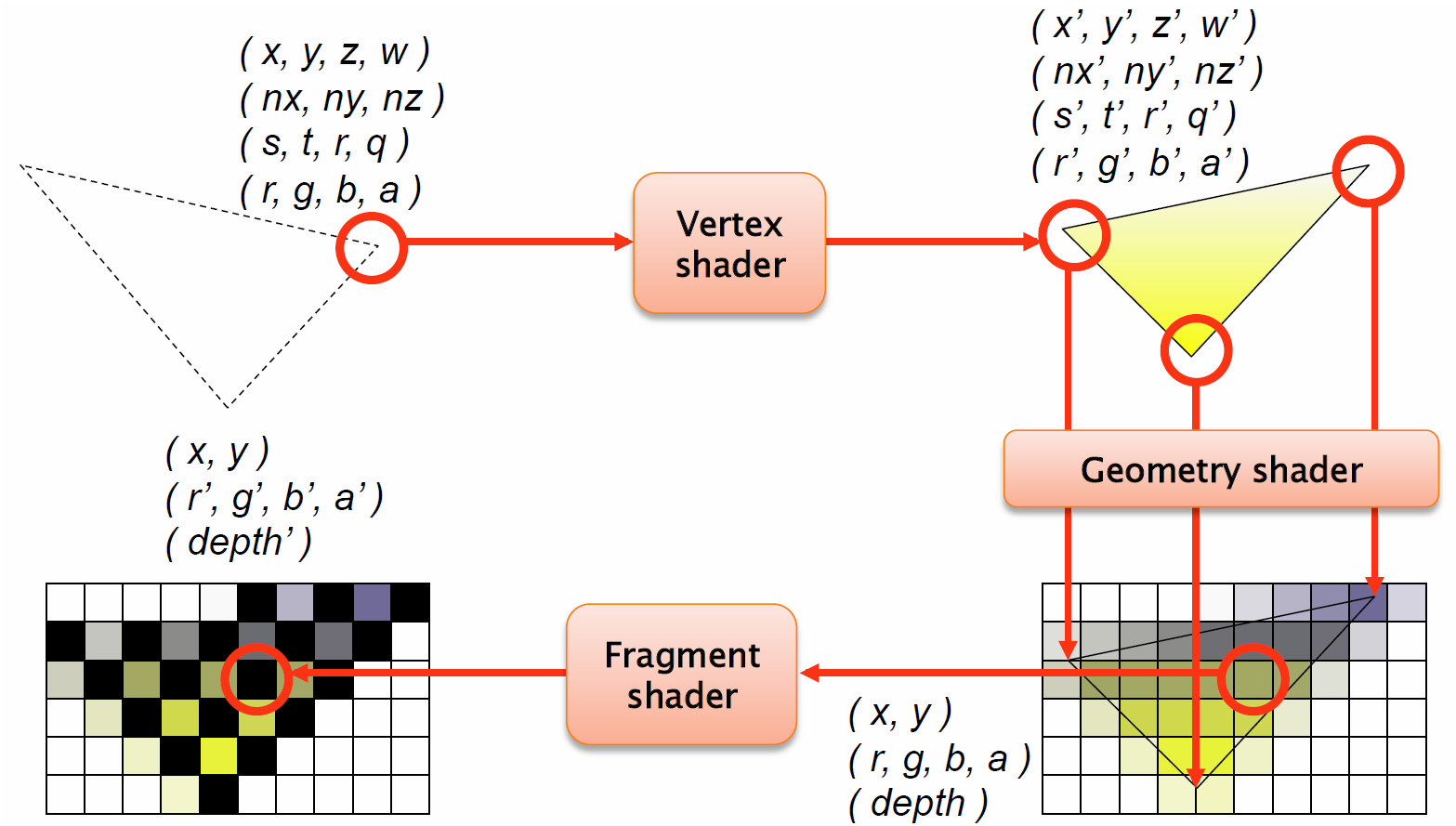


Figure 1-1 OpenGL ES 3.0 Graphics Pipeline

Шейдеры


(x, y, z, w)
(nx, ny, nz)
(s, t, r, q)
(r, g, b, a)
Vertex shader
(x', y', z', w')
(nx', ny', nz')
(s', t', r', q')
(r', g', b', a')
Geometry shader
(x, y)
(r', g', b', a')
(depth')
Fragment shader
(x, y)
(r, g, b, a)
(depth)

Vertex Shader

Vertex shader is a small program[1] that is executed on GPU[2] for each vertex[3] in mesh

small program[1] = is a small assembly-language program (compiled code)

executed on GPU[2] = i.e. 1 000 000 vertices per frame, GPU has 1 000 cores

each vertex[3] = single vertex at a time, doesn't generate new vertices

Shader от слова **shading**, по-русски - **освещение** (прямой перевод - **затенение**)

Vertex Shader

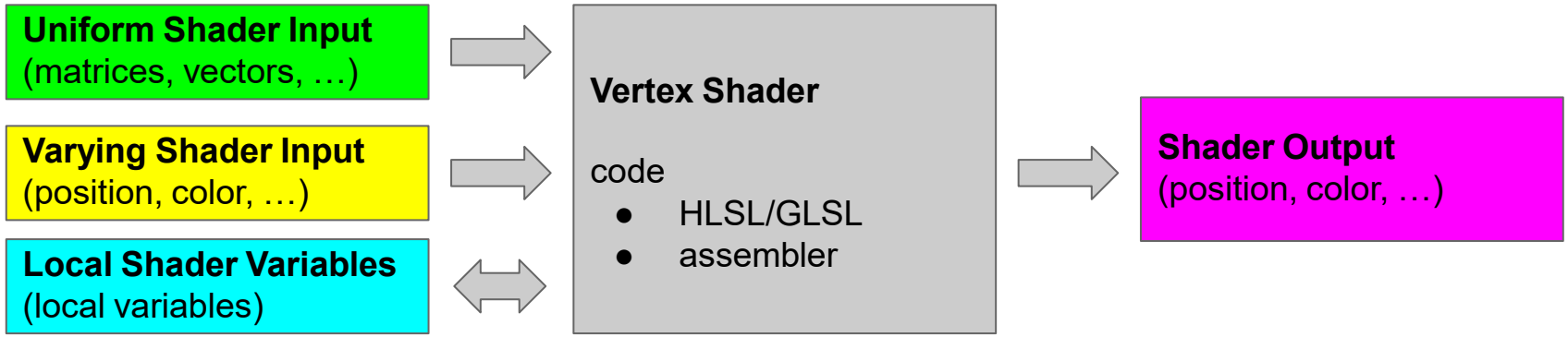
Can do:

- **transform** each vertex's **3D position** in virtual space **to** the **2D coordinate** at which it appears on the screen
- **manipulate** properties such as **position**, **color** and **texture coordinate**
- can enable **powerful control** over the details **of position**, **movement**, **lighting**, and **color** in any scene

Can’t do:

- polygon based operations (back face culling, occlusion culling)
- write to other vertices
- create new vertices

Example. OpenGL 2.0.



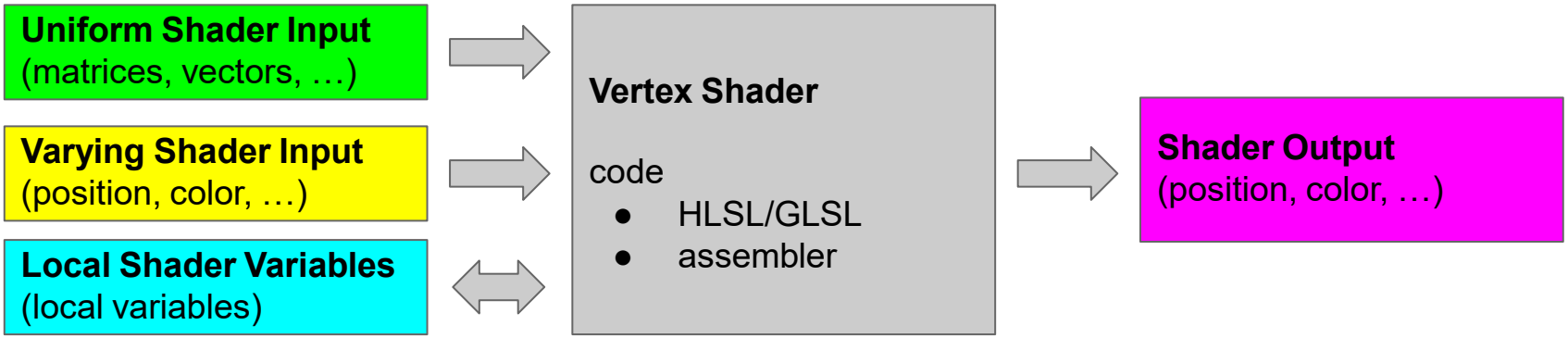

```
#version 110

uniform vec4 constColor;
uniform mat4 matrixWorldViewProjT;

attribute vec3 position;
attribute vec4 color;
```

```
void main()
{
        gl_Position   = vec4(position, 1.0) * matrixWorldViewProjT;
        gl_FrontColor = color * constColor;
}
```

Example. DirectX 9.



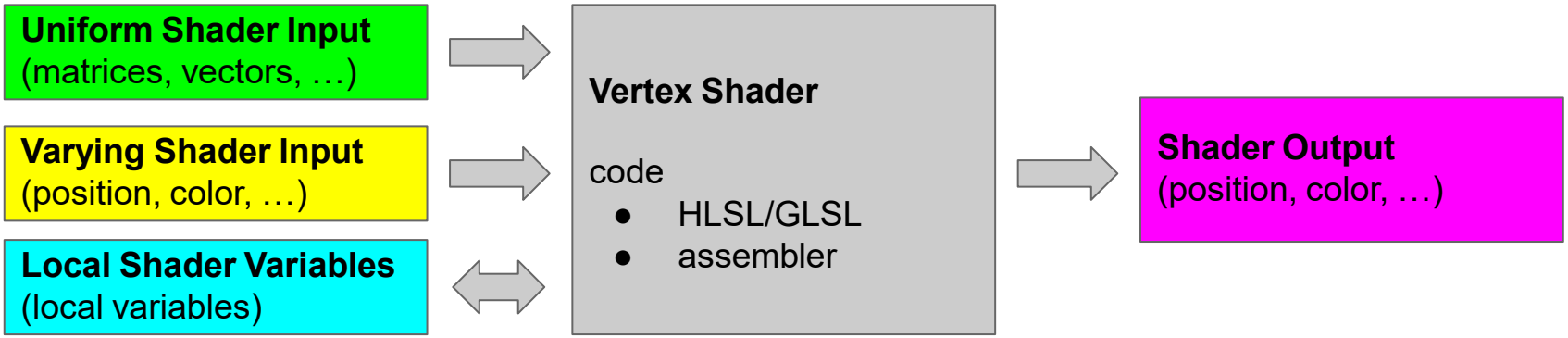

```
float4x4        matrixWorldViewProjT;
float4          materialColor;
struct VS_INPUT
{
        float4  Position        : POSITION;
        float4  Color           : COLOR;
};
struct VS_OUTPUT
{
        float4  Position        : POSITION;
        float4  Color           : COLOR0;
};
```

```
VS_OUTPUT main( VS_INPUT input )
{
        VS_OUTPUT output;

        output.Position = mul(input.Position, matrixWorldViewProjT);
        output.Color    = input.Color * materialColor;

        return output;
};
```

Example. DirectX 11.



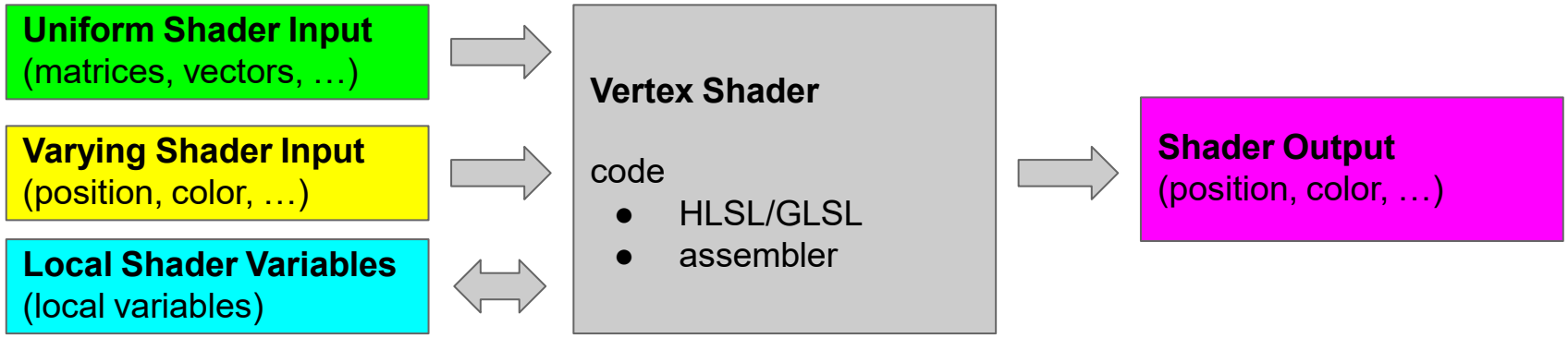

```
cbuffer ConstantBuffer : register( b0 )
{
        float4x4  matrixWorldViewProjT;
        float4    materialColor;
}
struct VS_INPUT
{
        float4    Position : POSITION;
        float4    Color    : COLOR;
};
struct PS_INPUT
{
        float4    Position : SV_POSITION;
        float4    Color    : COLOR;
};
```

```
PS_INPUT VS( VS_INPUT input )
{
        PS_INPUT output = (PS_INPUT)0;

        output.Position = mul( input.Position, matrixWorldViewProjT );
        output.Color    = input.Color * materialColor;

        return output;
}
float4 PS( PS_INPUT input) : SV_Target
{
        return input.Color;
}
```

HLSL/GLSL. Versions.



DirectX version	8.0	9.0	9.0a	9.0c	10, 10.1, 11
Vertex shader version	1.1[13]	2.0[6][13][14]	2.0a[6][13][14]	3.0[9][13]	4.0[10] 4.1[15] 5.0
# of instruction slots	128	256	256	≥ 512	4096
max # of instructions executed	128	1024	65536	65536	65536
# constant registers (Uniform Shader Input)	≥ 96	≥ 256	256	≥ 256	16×4096
# temp registers (Local Shader Variables)	12	12	16	32	4096

HLSL/GLSL. Versions.

Сегодня шейдеры пишут на:

- DirectX **HLSL** (High Level Shader Language)
- OpenGL **GLSL** (OpenGL Shader Language)

C-based languages that are compiled to assembler and executed on GPU

Every GPU has list of supported shader versions

DirectX and OpenGL has list of corresponding versions:

DirectX	2.0	2.1	3.0	3.1	3.2	4.0	5.0
OpenGL	1.10	1.20	1.20	1.40	1.50	3.30	4.30

HLSL/GLSL. Types and Functions.

Data Types	DirectX	OpenGL
scalars	bool, int, float	bool, int, float
vectors	float2, float3, float4	vec2, vec3, vec4
matrix	float4x4, float4x3, ...	mat4, mat3, mat2

Built-in Functions	DirectX	OpenGL
trigonometric	sin, cos, tan, asin, acos, atan, atan2	sin, cos, tan, asin, acos, atan
exponential	pow, exp, log, sqrt	pow, exp, log, sqrt
common	abs, sign, floor, ceil, fmod, min, max, clamp, saturate	abs, sign, floor, ceil, mod, min, max, clamp
geometric	length, distance, dot, cross, normalize	length, distance, dot, cross, normalize

GLSL. Features.

Синонимы:

x,y,z,w = r,g,b,a = s,t,p,q

Подстановки:

v.xyxy == vec4(v.x, v.y, v.x, v.y);

Жесткая типизация:

~~float v = 1;~~

Vertex shader:

```
highp vec4 gl_Position;          // should be written to
mediump float gl_PointSize;      // may be written to
```