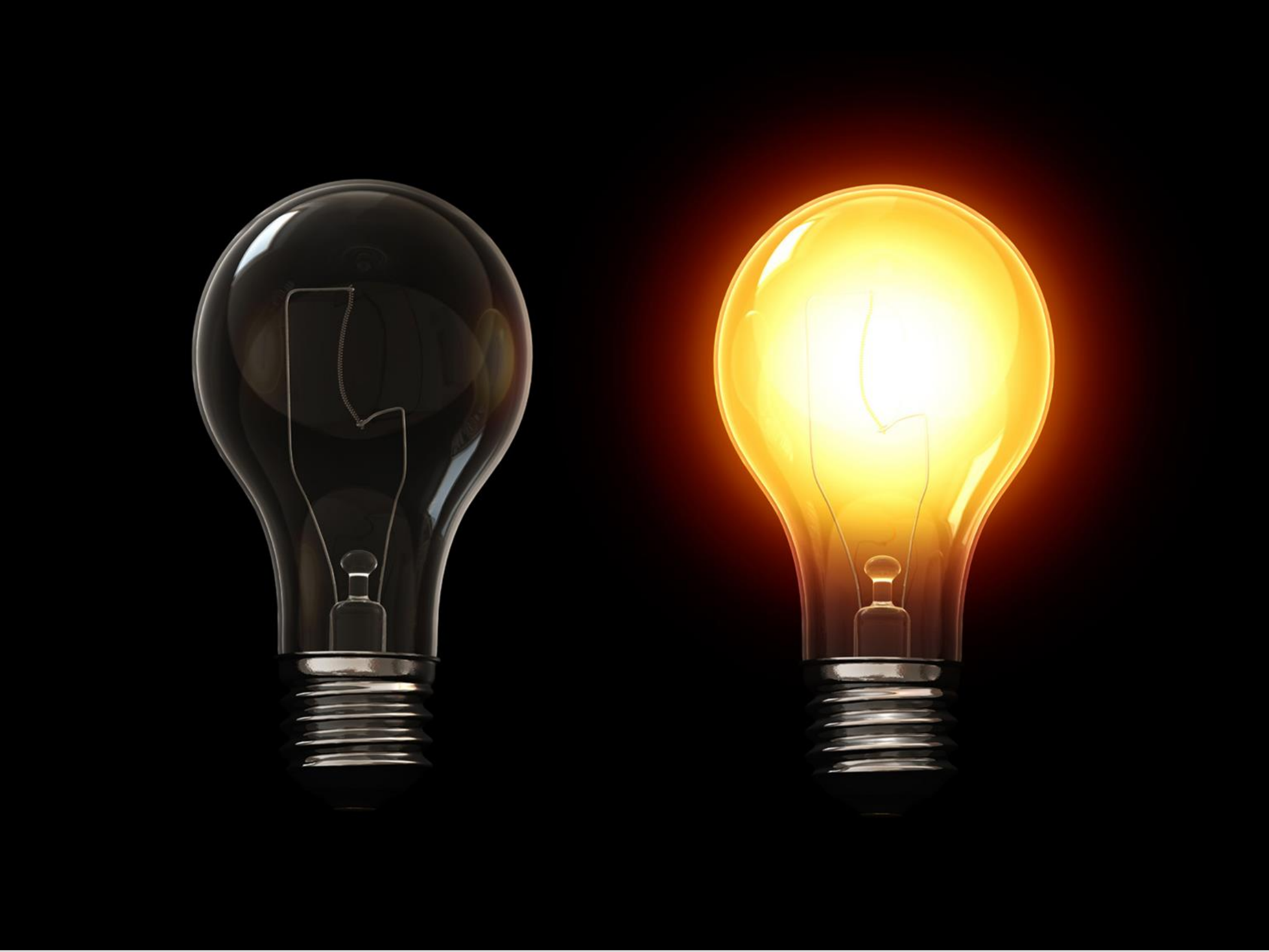
Apšvietimas

T120B167 Žaidimų grafinių specialiųjų efektų kūrimas ir programavimas

Rytis Maskeliūnas
Skype: rytmask
Rytis.maskeliunas@ktu.lt



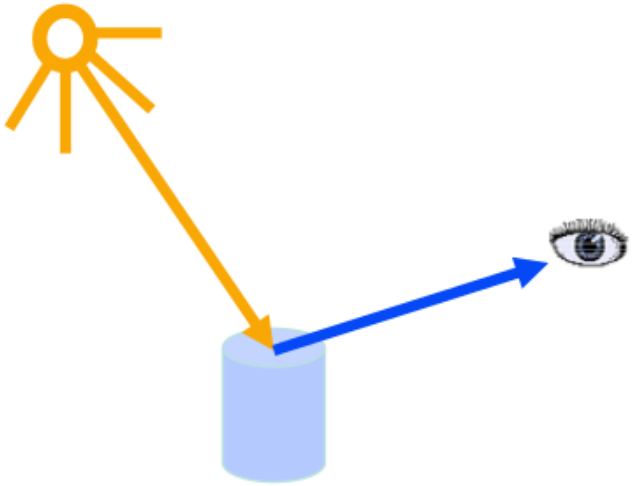
Bakalauro darbo PVZ.

MULTIMEDIJOS INŽINERIJOS KATEDRA

http://www.youtube.com/watch?v=-b7jbqwlW4c

Apšvietimo modelis

- Tikslas – realistinių paveikslų sukūrimui reikalinga modeliuoti įvairius paviršius esant skirtingoms apšvietimo sąlygoms
- Apšvietimo modelis: šaltinis, atspindžio taškas, stebėtojas



Apšvietimo modeliai ir šešėliavimas

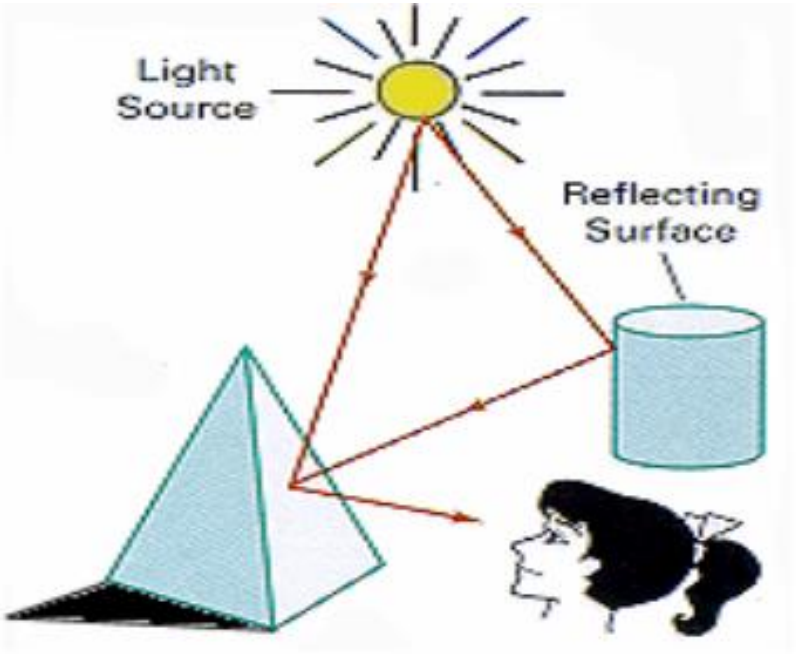
- Šviesos šaltinių modeliai
- Bendrasis (*ambient*) apšvietimas
- Difuzinis (*diffuse*) apšvietimas
- Veidrodinis (*specular*) apšvietimas

Apšvietimo modelio parametrai

- Apšvietimas yra aprašomas modeliais įvertinančiais šviesos šaltinių sąveiką su objektų paviršiais
- Apšvietimą sąlygojantys faktoriai:
  - Šviesos šaltinio parametrai:
    - Pozicija
    - Elektromagnetinis spektras
    - Forma
  - Paviršiaus parametrai:
    - Pozicija
    - Atspindžio savybės
    - Šalia esančių objektų pozicija
  - Stebėtojo (kameros) parametrai:
    - Pozicija
    - Sensoriaus spektrinis jautrumas

Apšvietimo modeliai ir vaizdavimas

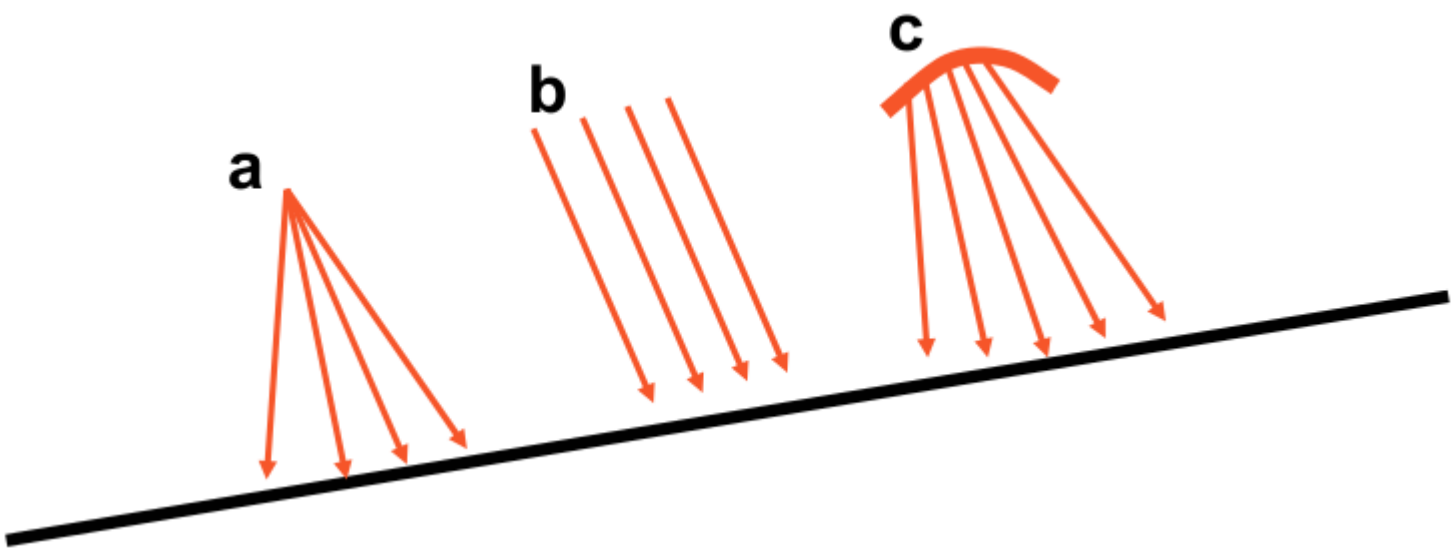
- Apšvietimo modeliai naudojami apskaičiuoti šviesos atspindžio intensyvumą tam tikrame paviršiaus taške
- Vaizdo generavimo metodai naudodami apšvietimo modelio sugeneruotus intensyvumo skaičiavimus nustato šviesos intensyvumą kiekvienam paveikslo pikseliui



Šviesos šaltinių modeliai

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Taškinis šaltinis: visi šviesos spinduliai sklinda iš vieno taško tolygiai gesdami
- Lygiagretus šaltinis: visi šviesos spinduliai yra lygiagretūs
- Paskirstytas šaltinis: šviesos spinduliai sklinda baigtinėje erdvėje



Apšvietimo modeliai

- Supaprastinti, empyriniai ir greiti paviršių apšvietimo intensyvumo skaičiavimo metodai
- Skaičiavimai remiasi optinėmis paviršių ir apšvietimo savybėmis neįvertinant atspindžių ir šešėlių
- Priimama, kad šviesos šaltiniai - taškiniai
- Daugumą scenų tenkinanti aproksimacija

Ambient (fono) apšvietimas

- Priimama, kaip nekryptinis aplinkos apšvietimo šaltinis
- Visiems objektams visomis kryptimis tenka vienodas apšvietimo kiekis.
- Labai paprastas, bet ne realistinis

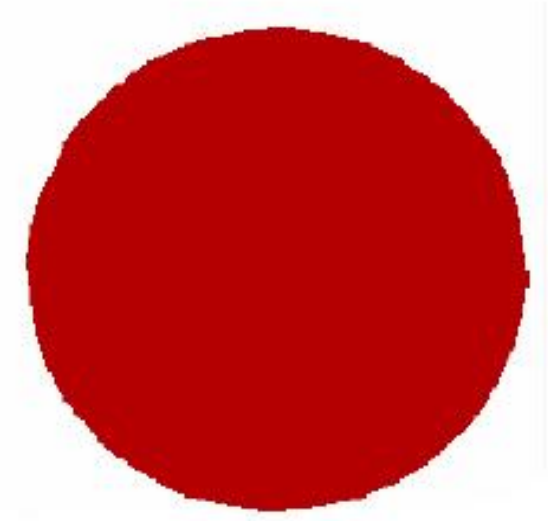
Ambient (fono) apšvietimas

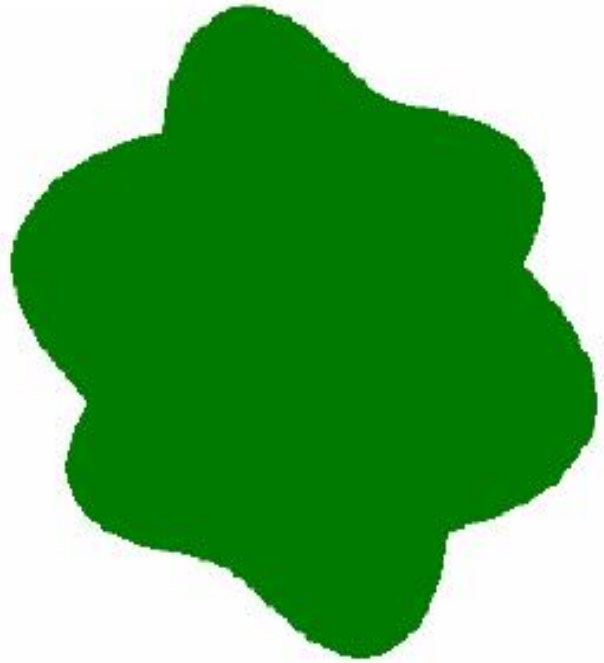
- Atspindėtas intesyvumas I_{amb} bet kuriame paviršiaus taške yra:

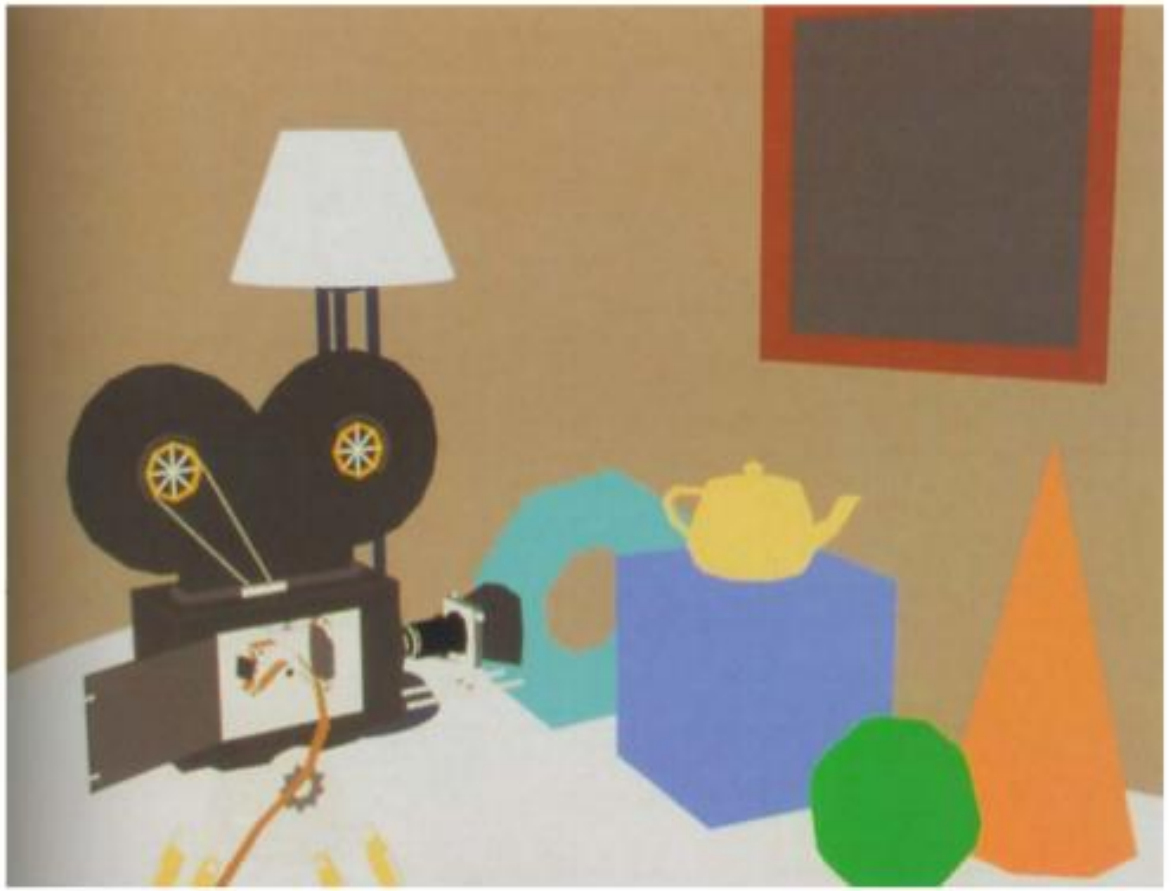
$$I_{amb} = K_a I_a$$

- I_a – ambient šviesos intensyvumas
- $K_a \in [0,1]$ – paviršiaus *ambient* sugeriamumas
- Iš principo I_a ir K_a yra spalvos funkcijos, todėl turime I^R_{amb}, I^G_{amb}, ir I^B_{amb}

Ambient pavyzdys







Unlit / Ambient

- Kodas:

http://pastebin.com/m6n8Ltyz arba moodle

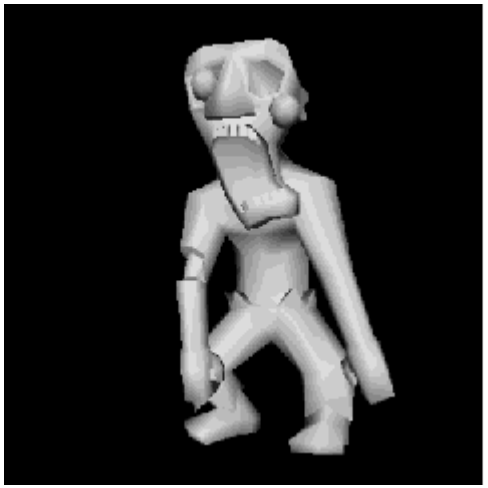
- Ambient spalva pridedame per Unity kintamąjį

```
float4 textureColor = tex2D(_MainTex, input.tex.xy) *
float4(UNITY_LIGHTMODEL_AMBIENT.rgb, 1);
```

Difuzinis apšvietimas

Iš esmės tas pats kas foninis apšvietimas, bet:

šis apšvietimas TURI KRYPTĮ !



Ambient šviesa – 3D modelis atrodo 2D

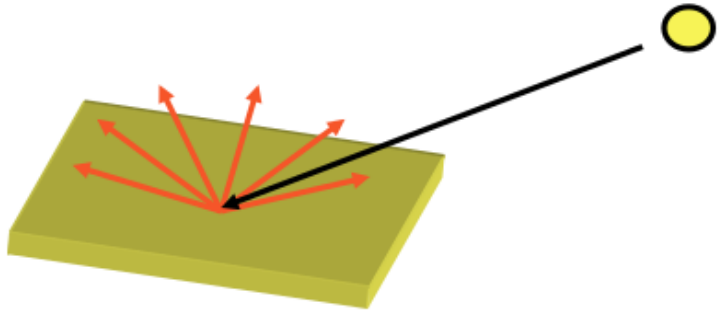
Difuzinė šviesa – 3D modelis atrodo 3D

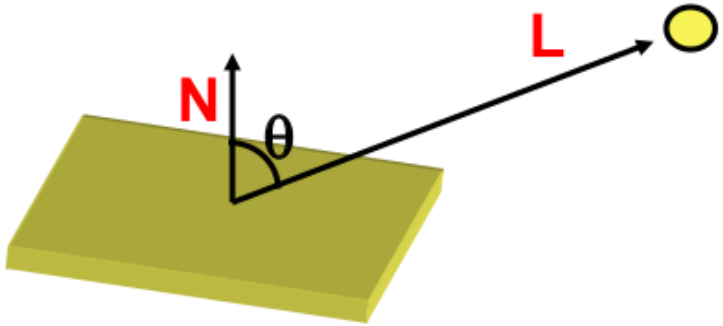
Paveiksliuke: pilka šviesa foninis apšvietimas; balta – difuzinis apšvietimas

Difuzinis apšvietimas

MULTIMEDIJOS INŽINERIJOS KATEDRA

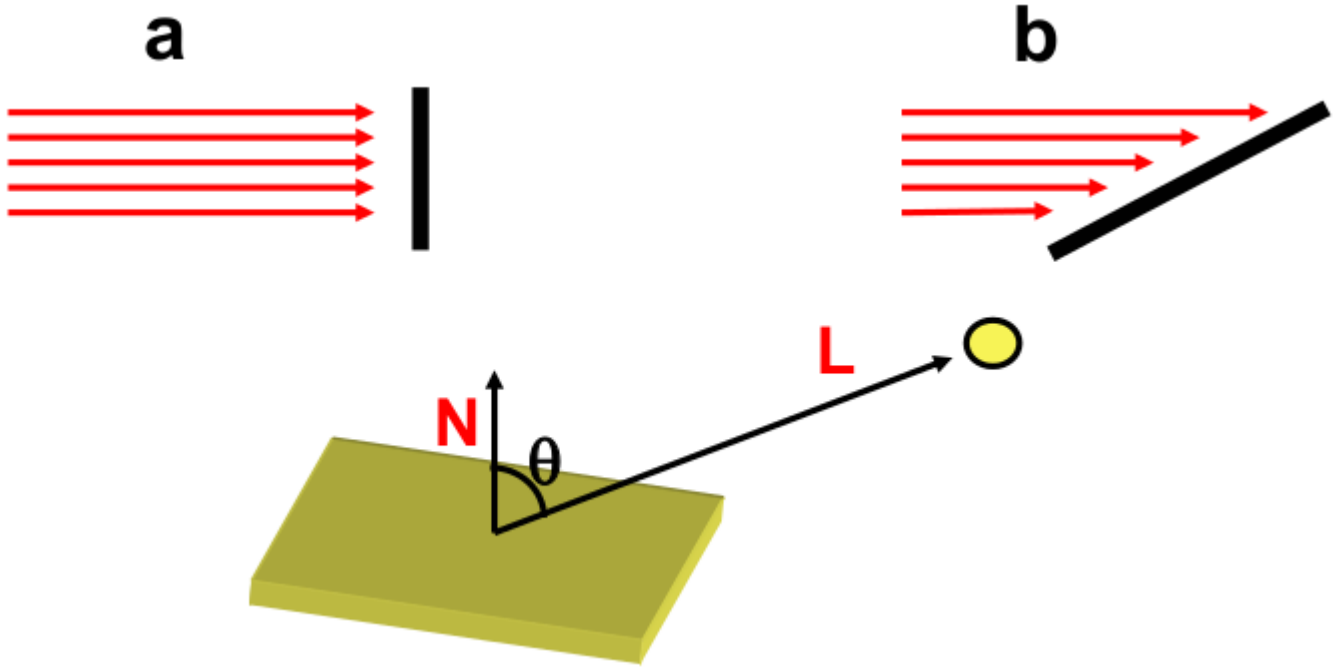
- Difuziniai paviršiai yra šiurkštūs ir netolygūs, pvz: molis, medžiaga
- Paviršius vienodai atspindi šviesą visomis kryptimis
- Šviesos ryškumas kiekviename taške yra proporcingas cos(θ)





Difuzinis apšvietimas

- Šviesos ryškumas yra proporcingas cos(θ): šviesai statmenas paviršius yra labiau apšviestas (a) negu su ja sudarantis tam tikrą kampą (b)



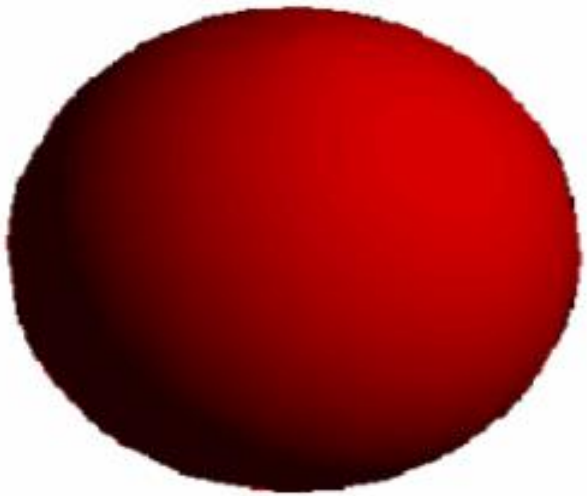
Difuzinis apšvietimas

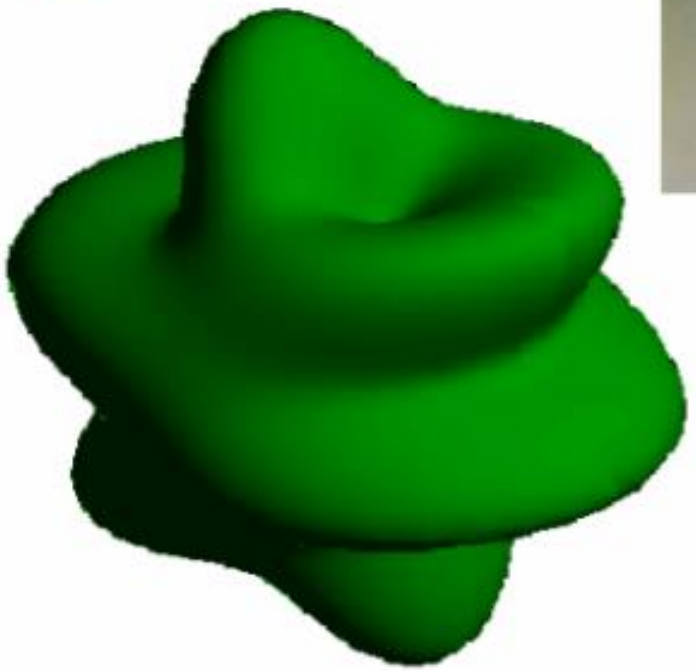
- Atspindėtas intensyvumas I_{diff} paviršiaus taške yra:

$$I_{diff} = K_d I_p \cos(\theta) = K_d I_p (N \cdot L)$$

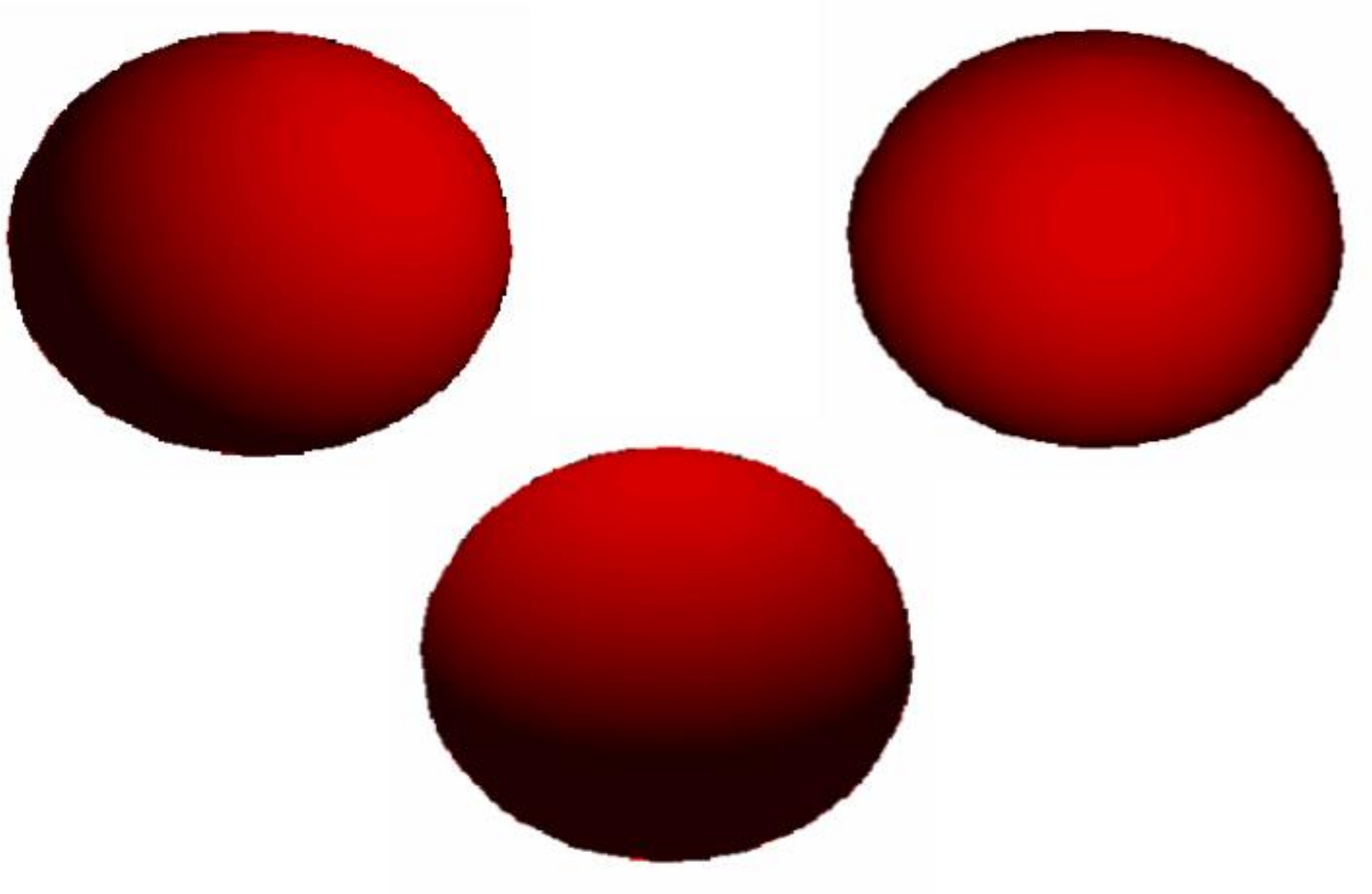
- I_p – taškinio šaltinio intensyvumas.
- $K_d \in [0,1]$ – paviršiaus *diffuse* sugeriamumas
- N – paviršiaus normalės vektorius
- L – šviesos krypties vektorius

Difuzinis apšvietimas





Difuzinis apšvietimas



Unity3D Diffuse per vertex

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Kodas: http://pastebin.com/gikCSgz0 arba moodle
- Pridedame apšvietimo informaciją:

```
Tags { "LightMode" = "ForwardBase" }
 uniform fixed4 _LightColor0; //Unity perduoda pirmą lempą
```

- Vertex input struktūrą papildome normale:

```
float3 normal : NORMAL;
```

- Vertex output papildome apskaičiuota šviesos spalva:

```
float4 color : COLOR0;
```

Unity3D Diffuse per vertex

- Pridedame „N dot L“

```
float3 normalDirection = normalize( mul( float4( input.normal, 1.0 ), _World2Object ).xyz );

float3 lightDirection = normalize( _WorldSpaceLightPos0.xyz );

float ndotl = dot( normalDirection, lightDirection );

float3 diffuse = _LightColor0.xyz * max( 0.0, ndotl );

output.color = half4( diffuse, 1.0 );
```

- Fragment šeideryje pridedame gautą spalvą:

```
float4 textureColor = tex2D(_MainTex, input.tex.xy) * input.color;
```

Difuzinis apšvietimas per HLSL

- Atsiranda 3 nauji kintamieji:
- `float3 DiffuseDirection;`
- `float4 DiffuseColor;`
- `float DiffuseIntensity;`
- Juose saugoma šviesos kryptis (L), spalva ir intensyvumas

- Vertex šeiderio įėjime tik pozicija.

```
struct VertexShaderInput
{
    float4 Position : POSITION0;
};
```

Difuzinis apšvietimas per HLSL

- VertexShaderOutput atsiranda papildomas narys (angl., member) – normalė (angl., Normal).
- Normalė suskaičiuojama vertex šeiderio f-joje, pagal objekto normalė duotuoje viršūnėlėje (vertex‘e).

```
struct VertexShaderOutput
{
    float4 Position : POSITION0;
    float3 Normal : TEXCOORD0;
};
```

Difuzinis apšvietimas per HLSL

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Normalę reikia „paduoti“ Vertex šeiderio įėjime (VertexShaderInput). Tam naudojamas Normal parametras tiek VertexShaderFunction, tiek VertexShaderInput
- Taip:

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput
input,float3 Normal : NORMAL
```

Difuzinis apšvietimas per HLSL

- Vertex šeiderio f-ja vėlgi labai panaši, bet ir čia atsiranda noramalės (angl., Normal), t.y. reikia suskaičiuoti normales turimoje erdvėje (analogiškai kaip tai buvo daryta pozicijų atveju) :

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput input,float3 Normal : NORMAL)
{
    VertexShaderOutput output;

    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);
    float3 normal = normalize(mul(Normal, World));
    output.Normal = normal;

    return output;
}
```

- Iš principo įėjimo normalė (input Normal) yra sudauginama su pasaulio matrica (World matrix), rezultatas normalizuojamas (normalize) ir saugomas normalės kintamajame (variable normal). Išėjime nustatomas minėtas normalės kintamasis (toliau jis bus perduodamas Pixel šeideriui).

Difuzinis apšvietimas per HLSL

- Norint perduodi normalę į pixel šeiderį reikia sukurti naują kintamą VertexShaderOutput struktūroje:
- `float3 Normal : TEXCOORD0`
- Normalės saugomos GPU registre (0,1,2,... yra registro vieta)
- NB. TEXCOORD šiaip naudojamas textūrų koordinatėms saugoti, normalės dedikuoto registro neturi.
- Pixel šeiderio paskirtis yra suskaičiuoti galutinę šviesos matematiką:
- $I = Aintensity \times Acolor + Dintensity \times Dcolor \times N.L$
- $N.L=(|N| \times |L| \times \cos(\Theta))$

Difuzinis apšvietimas per HLSL

- Pirmi keturi yra kintamieji. Kaip gauname N.L?
- Išsaugome normalės (Normal) vektorių, konvertuojam į float4, naudojam gamyklinę HLSL f-ją dot(x,y) (skaičiuojam N.L). L vertė su minusu (-DiffuseDirection) nes L šiuo atveju aprašo švisos kryptį IŠ KUR šviečia, bet ne į kur šviečia.

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    float4 norm = float4(input.Normal, 1.0);
    float4 diffuse = saturate(dot(-DiffuseDirection,norm));

    return AmbientColor*AmbientIntensity+DiffuseIntensity*DiffuseColor*diffuse;
}
```

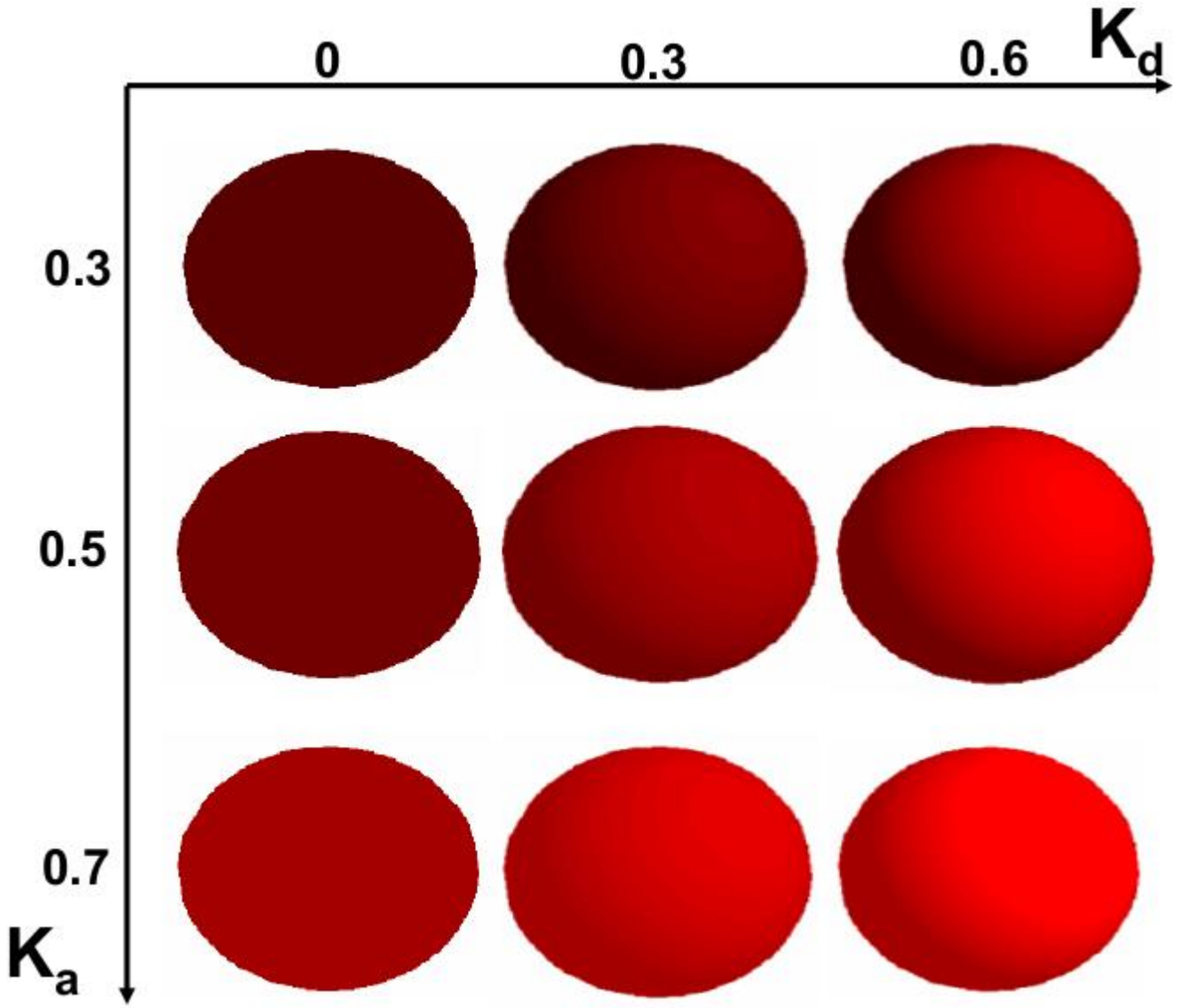
Ambient + Diffuse

- Apjungiamas *ambient* ir *diffuse* apšvietimas:

$$I = I_{diff} + I_{amb} = K_d I_p (N \cdot L) + K_a I_a$$

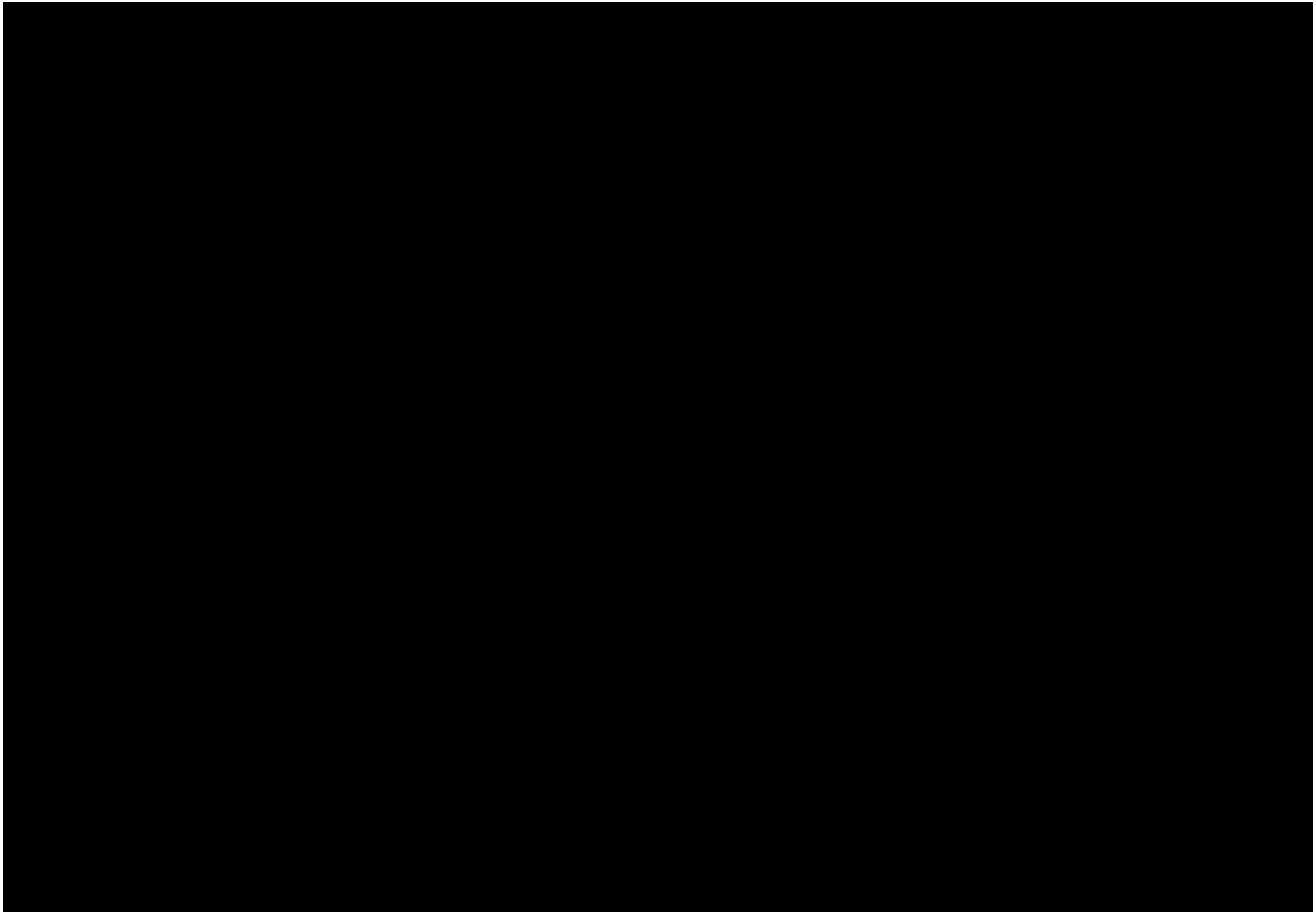
- Skaičiuojant intensyvumą kiekvienam spalvos kanalui turi būti įvertinta kiekviena spalvos komponentė atskirai I^R, I^G ir I^B

Ambient+Diffuse



Gyvenimiška iliustracija

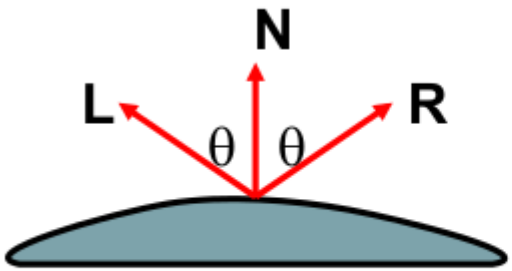
MULTIMEDIJOS INŽINERIJOS KATEDRA



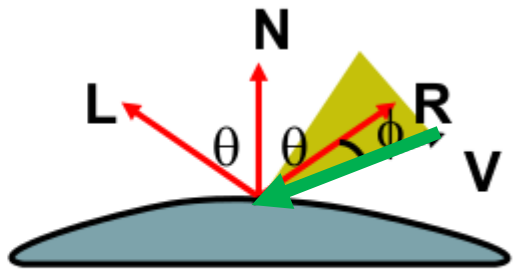
http://www.youtube.com/watch?v=soHYBvYpNRU

Spindintis apšvietimas

- Modeliuoja atspindžius ant blizgių ir lygių paviršių (metalas, plastikas).
- Difuzinė šviesa nesukuria spindėjimo efekto.
- Atspindžio intensyvumas priklauso nuo atspindžio kampo
- Idealus atspindintis paviršius (veidrodis) šviesą atspindi viena kryptimi: **R**
- Matiniai paviršiai skirtingai nei veidrodžiai šviesą skaido tam tikru laipsniu nuo R



Ideal specular surface



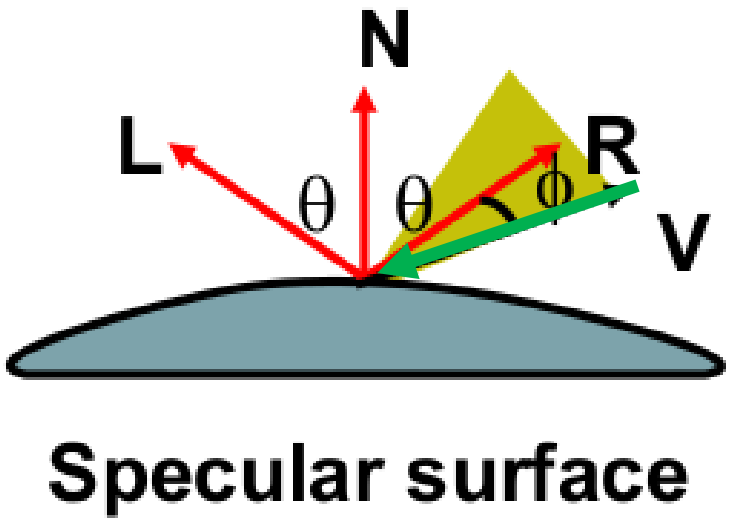
Non-ideal specular surface

Spindintis apšvietimas

- Phong modelis: atspindėtos šviesos intensyvumas mažėja proporcingai cos(θ) laipsniui:

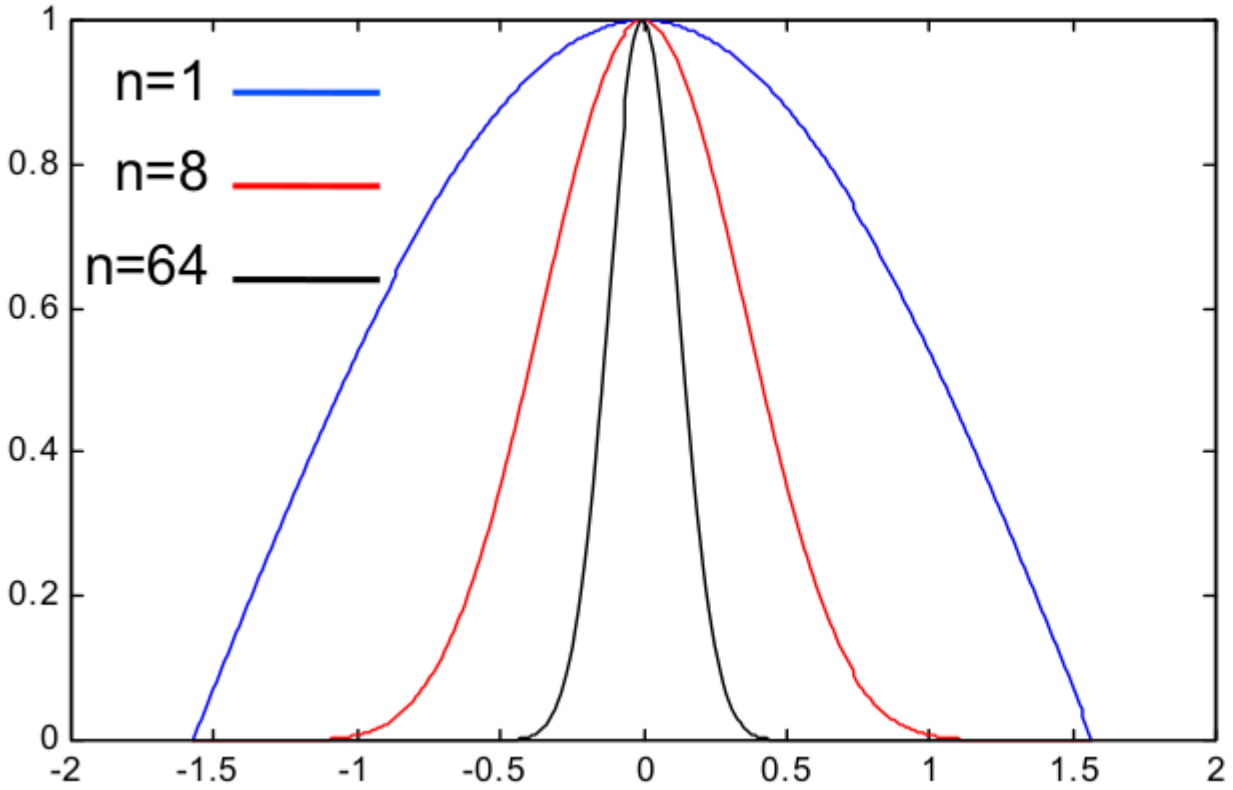
$$I_{spec} = K_s I_p \cos^n(\theta) = K_s I_p (R \cdot V)^n$$

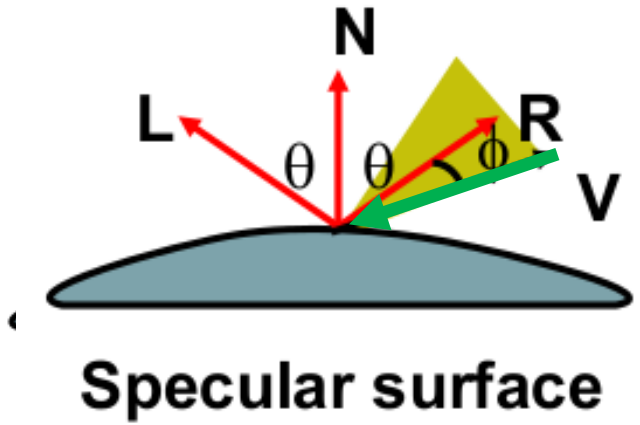
- K_s – paviršiaus veidrodinis atspindimumas
- n - veidrodinio atspindžio parametras nusakantis nuokrypį nuo idealaus veidrodinio paviršiaus (idealus veidrodis n=∞)



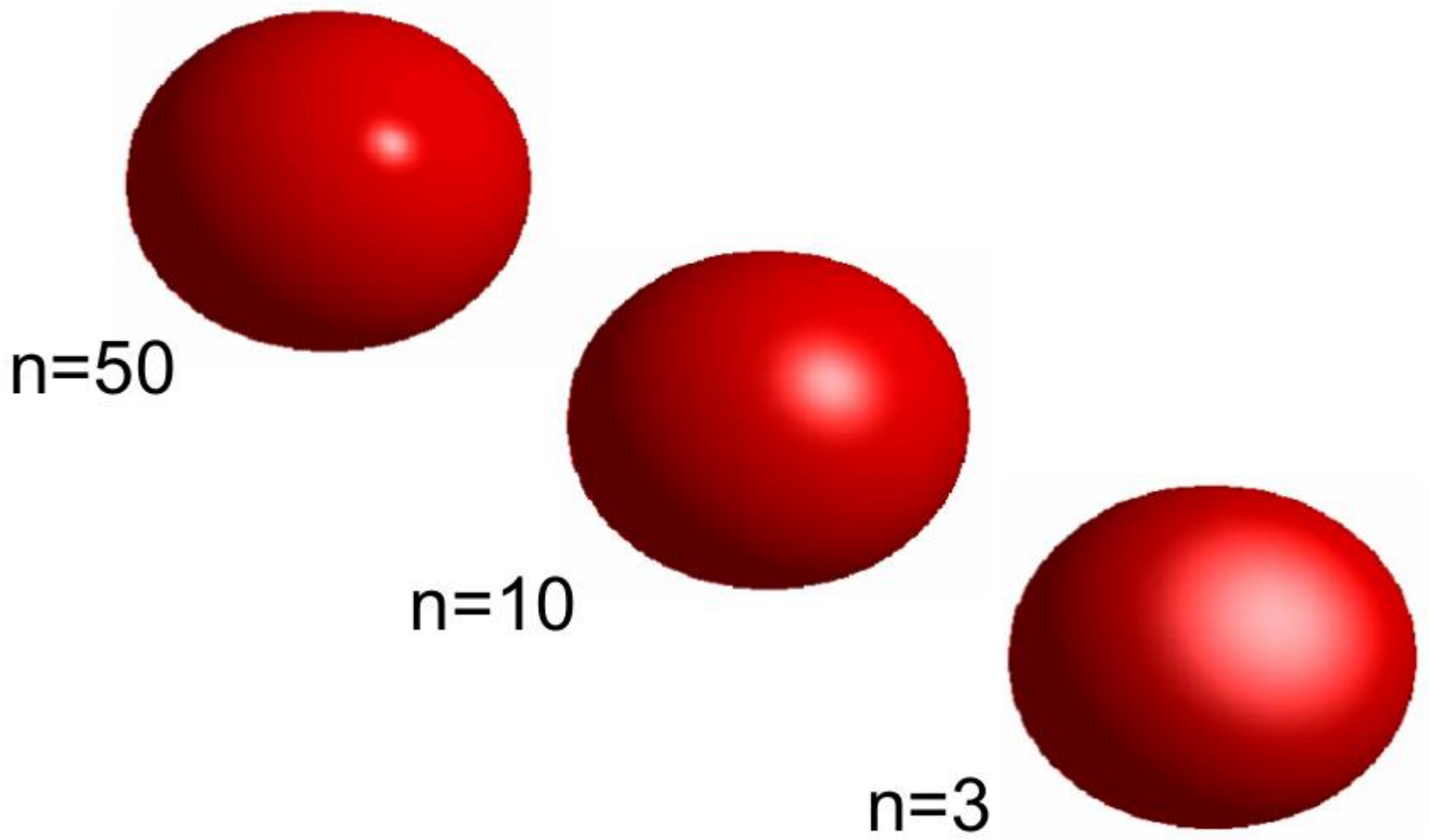
Spindintis apšvietimas

- **Phong modelis**: $\cos^{n}(\theta)$ funkcijos esant trims skirtingoms n parametro reikšmėms
- n- „spindėjimo“ dydis





Spindintis apšvietimas



Unity3D Diffuse specular per vertex

- Kodas: http://pastebin.com/bXLbS1UE
- Spindėjimo spalva ir stiprumas

_SpecColor("Specular Material Color", Color) = (1, 1, 1, 1)

_Shininess("Shininess", Float) = 0.05

```
uniform fixed4 _SpecColor;
uniform float _Shininess;
```

- Unity šeiderių funkcijos:

```
#include "UnityCG.cginc"
```

Unity3D Diffuse specular per vertex

- Spindėjimas iš kart po N dot L:

```
float3 viewDirection = WorldSpaceViewDir( input.vertex );

float3 specularReflection = = float3( 0.0, 0.0, 0.0 );

if( ndotl > 0 )

{

float3 reflection = reflect( -lightDirection, normalDirection );

float4 rdotv = pow( max( 0.0, dot( reflection, viewDirection ) ), _Shininess );

specularReflection = _LightColor0.rgb * _SpecColor.rgb * rdotv;

}

 float3 ambientLighting = UNITY_LIGHTMODEL_AMBIENT.rgb;

output.color = half4( ambientLighting + diffuse + specularReflection, 1.0 );
```

Spindintis apšvietimas per HLSL

- Spindinčiam apšvietime skaičiuojamas dar vienas vektorius kuris imituoja šviesos šaltinio atspindį, į kurį žiūri kameros akis.
- „Akis“ arba apžvalgos vektorius – tai vektorius kuris eina iš kameros pozicijos į tą vietą į kur kamera „žiūri“ (V)
- Arba:

```
viewMatrix = Matrix.CreateLookAt( new Vector3(x, y, z), Vector3.Zero,
Vector3.Up );
```

- „Akies“ pozicija yra pirmas CreateLookAt parametras:

```
new Vector3(x, y, z)
```

- Kuris toliau yra saugomas kintamajame:

```
Vector4 vecEye = new Vector4(x, y, z,0);
```

Spindintis apšvietimas per HLSL

- Norint suskaičiuoti spindintį apšvietimą reikia panaudoti dot f-ją tarp R (atspindys) ir apžvalgos vektoriaus (V, kameros akis), o visa tai valdoma n laipsniu (kiek objektas spindi).
- Vertex šeideryje viso labo viena modifikacija – pridedamas View vektorius prie VertexShaderOutput struktūros, ir skaičiuojama VertexShaderFunction f-joje.

```
struct VertexShaderOutput
{
    float4 Position : POSITION0;
    float3 Normal : TEXCOORD0;
    float3 View : TEXCOORD1;
};
```

Spindintis apšvietimas per HLSL

- Apžvalgos vektoriui (V) skaičiuojama kameros pozicija, kam pridedamas naujas kintamasis:

```
float3 EyePosition;
```

- output structūroje nustatomas apžvalgos (View) vektorius:

```
output.View = normalize(float4(EyePosition,1.0) - worldPosition)
```

Spindintis apšvietimas per HLSL

- Vertex šeideris:

```
struct VertexShaderOutput
{
    float4 Position : POSITION0;
    float3 Normal : TEXCOORD0;
    float3 View : TEXCOORD1;
};
```

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput input,float3 Normal : NORMAL)
{
    VertexShaderOutput output;

    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);
    float3 normal = normalize(mul(Normal, World));
    output.Normal = normal;
    output.View = normalize(float4(EyePosition,1.0) - worldPosition);

    return output;
}
```

Spindintis apšvietimas per HLSL

- Lieka suskaičiuoti pixel šeiderį, kuriame yra float4 su duotojo pikselio spalva I pagal anksčiau minėtą formulę.
- Taipogi reikia suskaičiuoti atspindėjimo vektorių L pagal N: R = 2 * (N.L) * N – L
- Dot L ir N jau skaičiavome:

```
float4 diffuse = saturate(dot(-LightDirection,normal));
```

- Na o atspindžio vektorių galime skaičiuoti taip:

```
float4 reflect = normalize(2*diffuse*normal-float4(LightDirection,1.0));
```

Spindintis apšvietimas per HLSL

- Beliko suskaičiuoti spindintį apšvietimą. Tai atliekama skaičuojat dot produktą tarp R ir V ir pakeliant laipsniu n: (R.V)^n
- Tam galima panaudoti HLSL f-ją pow(a,b) (suskaičiuoja a^b rezultatą)

```
float4 specular = pow(saturate(dot(reflect,input.View)),15);
```

- Viską apjungus galima suskaičiuoti galutinę pikselio spalvą:

```
return
AmbientColor*AmbientIntensity+DiffuseIntensity*DiffuseColo
r*diffuse+SpecularColor*specular;
```

Spindintis apšvietimas per HLSL

- Viską jau girdėjome. Suskaičiuojam foninę ir difuzinę šviesas ir jas sudedam. Tuomet spindinčios šviesos spalvą padauginam su spindėjimo koeficientu ir pridedam prie foninės ir difuzinės šviesos spalvos.
- Rezultate:

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    float4 normal = float4(input.Normal, 1.0);
    float4 diffuse = saturate(dot(-LightDirection,normal));
    float4 reflect = normalize(2*diffuse*normal-float4(LightDirection,1.0));
    float4 specular = pow(saturate(dot(reflect,input.View)),15);

    return AmbientColor*AmbientIntensity+DiffuseIntensity*DiffuseColor*diffuse+SpecularColor*specular;
}
```

Ambient+Diffuse+Specular

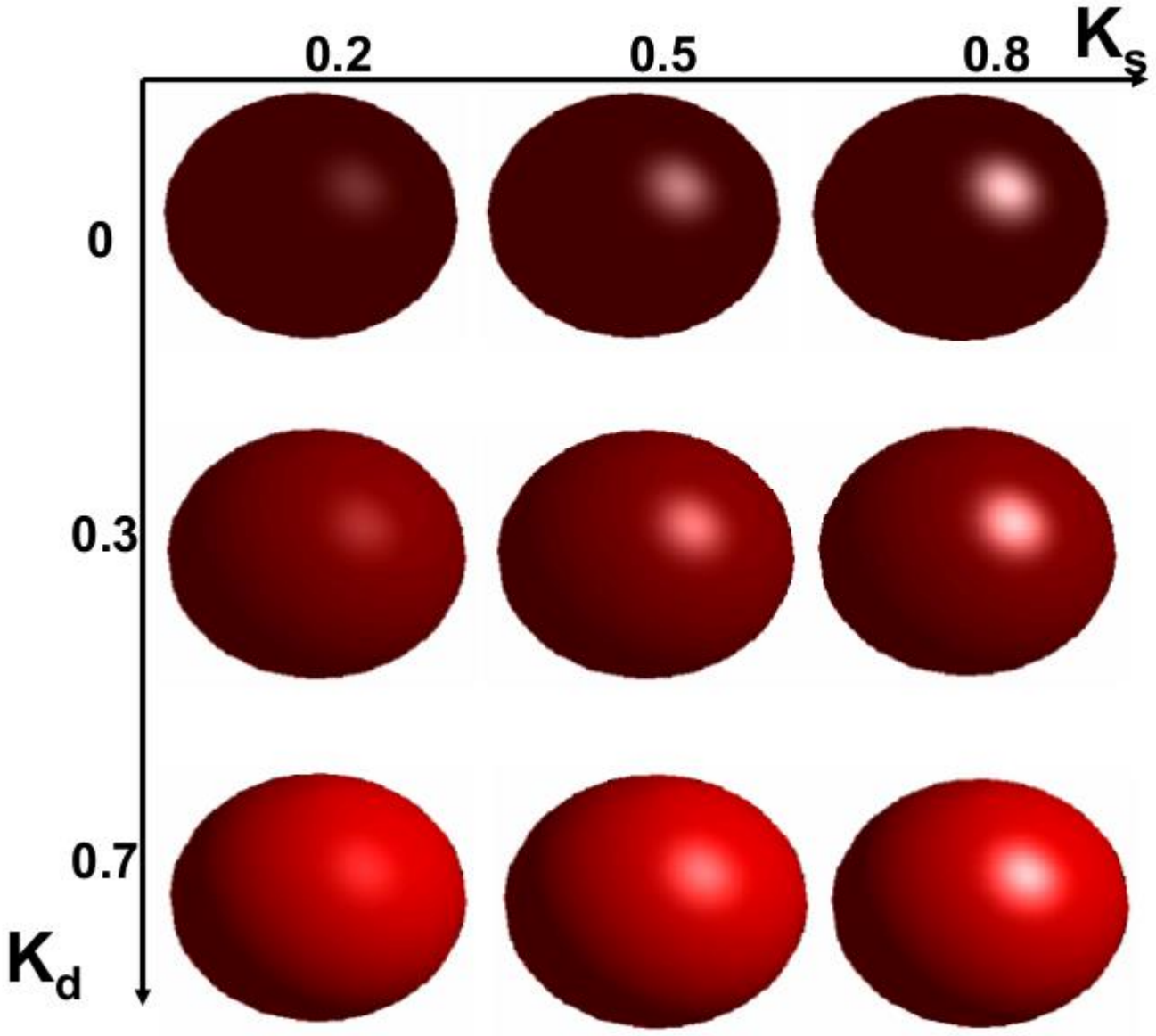
- Sudėjus visas tris komponentes:

$$I = I_{diff} + I_{amb} + I_{spec} =$$
$$K_a I_a + I_p(K_d(N \cdot L) + K_s(R \cdot V)^n)$$

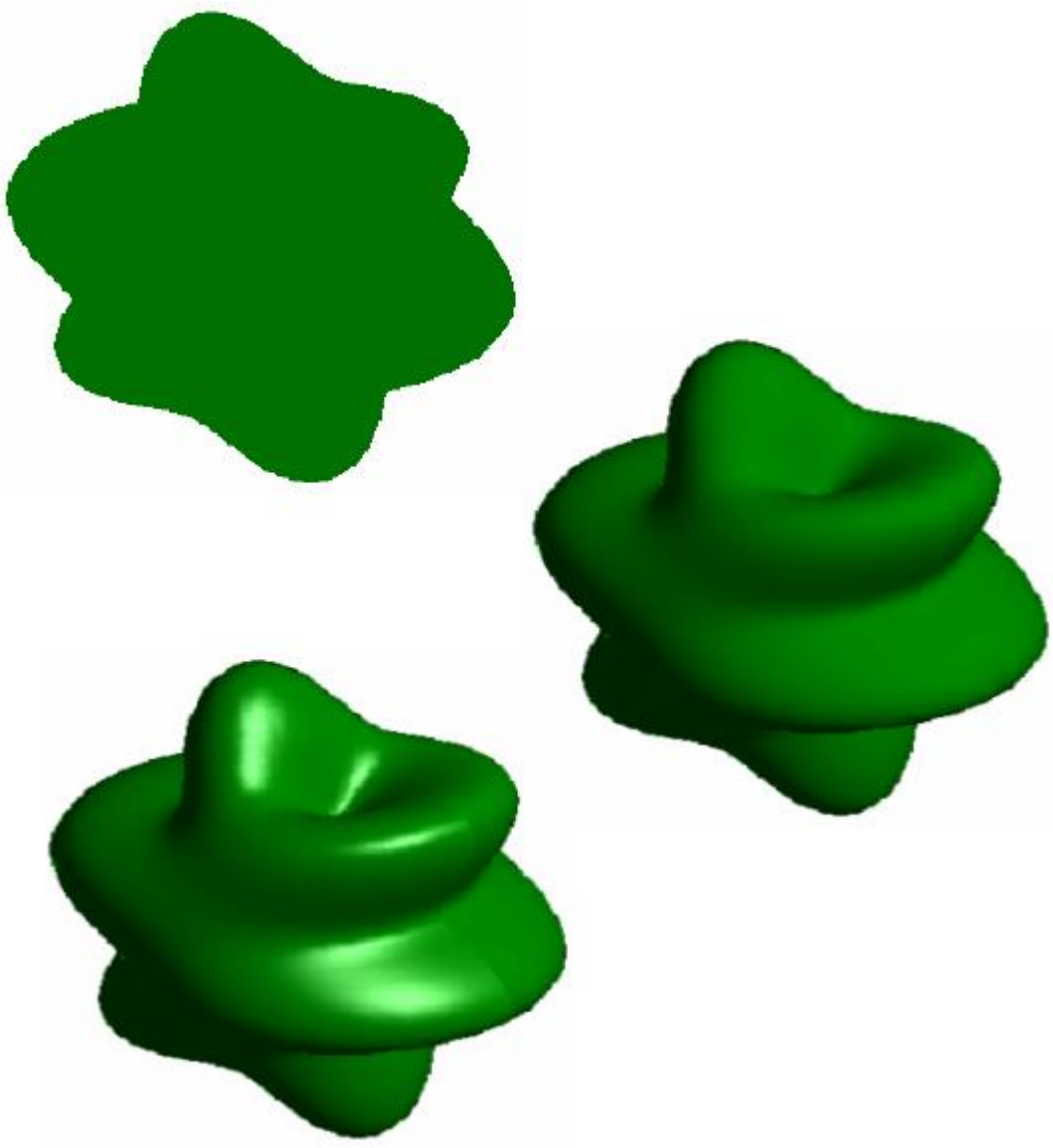
- Jei scenoje **k** šviesos šaltinių:

$$I = I_{amb} + \sum_k \left(I^k_{diff} + I^k_{spec}\right)$$

Diffuse+Specular



Komponenčių sandara



Ambient

Ambient + Diffuse

Ambient + Diffuse + Specular

PABAIGA