

Základy počítačové grafiky

Přednáška 5

Martin Němec

VŠB-TU Ostrava

2024

Aktuálně:

- Vykreslování objektů (VAO+VBO).
- Modelové transformace.
- Pohledová transformace.
- Projekční transformace.

Je něco, co byste chtěli zopakovat?

Co si pamatujete z návrhu kódu?

- Jasně definujte povinnosti/zodpovědnost.
- Každá třída by měla mít pouze jednu odpovědnost. To zjednodušuje údržbu a rozšiřování kódu.
- Pozor na "božské" třídy, které všechno ví a všechno umí.
- Třídy by měly být navrženy tak, aby bylo snadné přidat nové funkce (rozšiřovat je) bez nutnosti měnit stávající kód.
- Důležitá je abstrakce kódu.

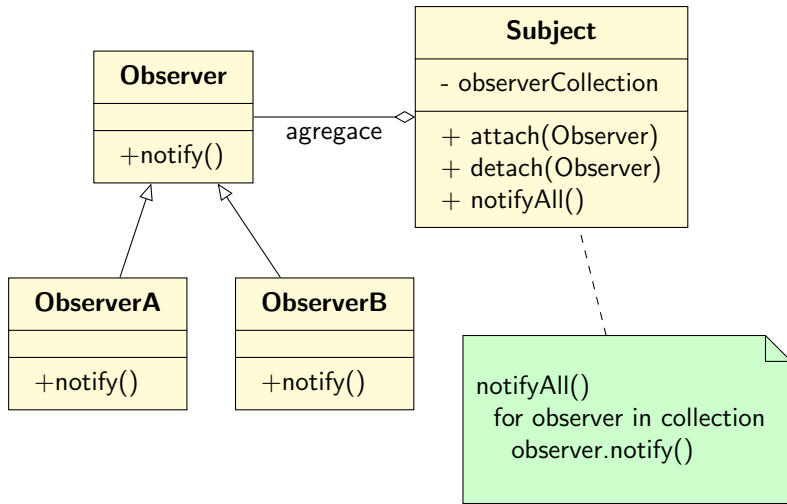
Návrh tříd není jednorázový proces. Během vývoje může být potřeba některé třídy přidat, upravit, nebo sloučit.

Refaktoring je klíčová součást vývoje OOP, používáme nástroje jako UML diagramy.



Dominik Regec - Principal Level Architect Hangar 13

Observer



Kulová plocha

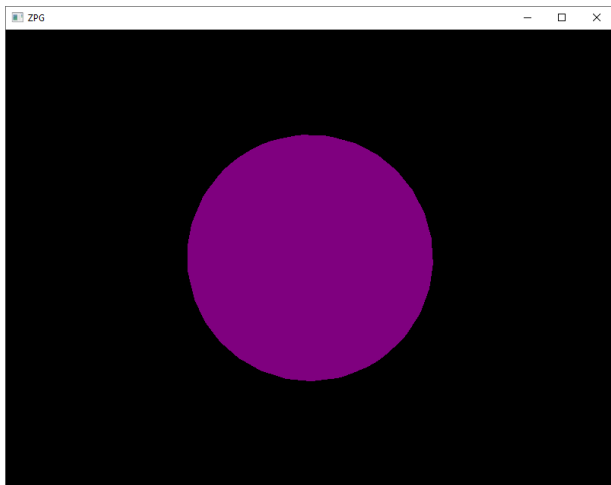
```
//glDrawArrays(GL_TRIANGLES, 0, 2880);
```

```
const float sphere[17280] = {  
-0.831,-0.555,0.000,-0.833,-0.552,0.000,  
-0.923,-0.382,0.000,-0.924,-0.380,0.000,  
-0.815,-0.555,-0.162,-0.817,-0.552,-0.162,  
...  
-0.375,-0.923,0.074,-0.380,-0.921,0.075,  
-0.555,-0.831,0.000,-0.559,-0.828,0.000};
```

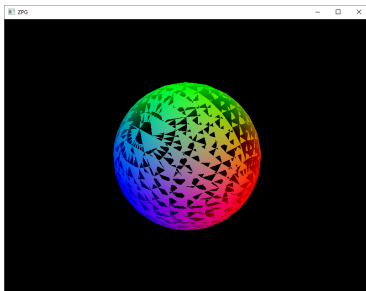


```
glVertexAttribPointer(  
    0,3,GL_FLOAT,GL_FALSE,6*sizeof(float),NULL);  
glVertexAttribPointer(  
    1,3,GL_FLOAT,GL_FALSE,6*sizeof(float), (void*)3);
```

Přidání tělesa (kulové plochy) do scény. Jsme spokojení?



Problém se "špatným" vykreslováním trojúhelníků vyřešíte takto.
Budeme se mu věnovat na další přednášce.

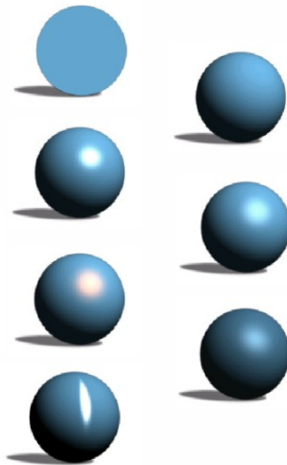


```
glEnable(GL_DEPTH_TEST);  
while (!glfwWindowShouldClose(window)) {  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```


Výpočet barvy na povrchu těles

- Constant
- Lambert
- Phong
- Blinn
- Cook-Torrence
- Strauss
- Anisotropic

(kvalita vs. rychlost)



Konstantní shader

Ambientní složka zabraňuje tomu, aby povrchy odvrácené od světelných zdrojů byly zcela černé.

- pouze konstantní barva
- ignorují se normály
- stejná orientace ploch
- stejná vzdálenost



```
outColor = vec4( 0.385, 0.647, 0.812, 1.0 );
```

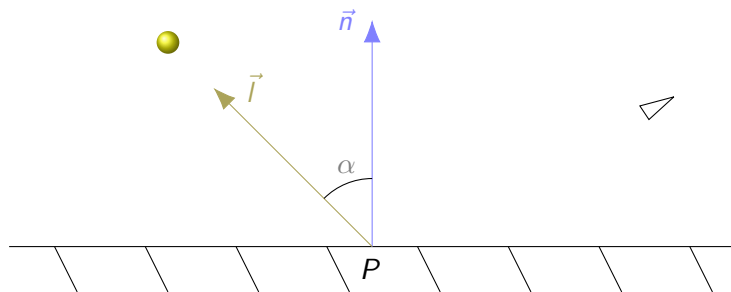
- ambientní a difúzní složka
- neobsahuje zrcadlovou složku
- matný povrch



```
I=max(dot(lightVector,worldNor), 0.0);
```

Lambertův shader

Intenzita světla je tím větší, čím je úhel mezi směrem ke světlu a normálou menší.



```
I=max(dot(lightVector,worldNor), 0.0);
```

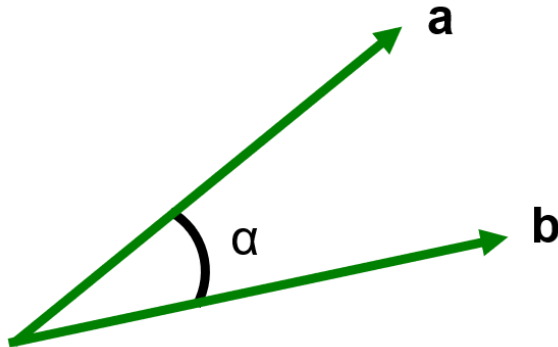
Skalární součin

Definuje se mezi dvěma vektory a zachycuje vztah mezi velikostí těchto vektorů a jejich úhlem.

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \alpha$$

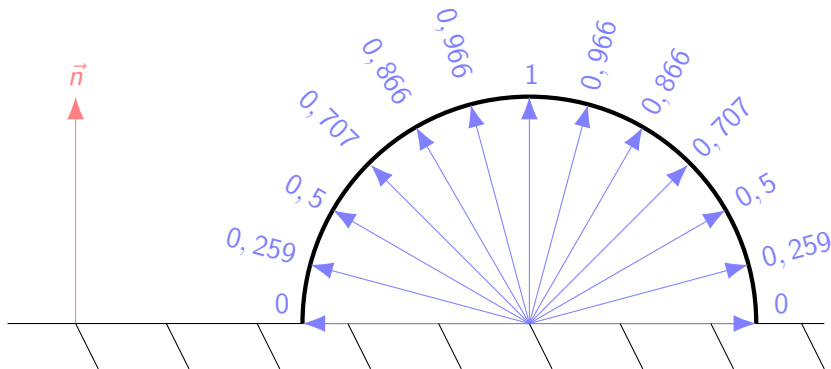
$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$\vec{a} \cdot \vec{b} = \text{dot}(\vec{n}, \vec{l})$$



Lambertův shader

Proč kosinus? Význam $(\max(\text{dot}(\vec{n}, \vec{l}), 0))$.



Vertex shader

```
#version 400
//atribut 0 in VAO
layout(location = 0) in vec3 vp; //position
//atribut 1 in VAO
layout(location = 1) in vec3 vn; //normal
out vec4 worldPos;
out vec3 worldNorm;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main(void)
{
    worldPos = model * vec4(vp,1.0);
    mat4 normal=model;//problem - priste vysvetlime
    worldNorm = vec3(normal * vec4(vn,1.0));
    gl_Position = projection * view * model * vec4(vp,1.0);
}
```

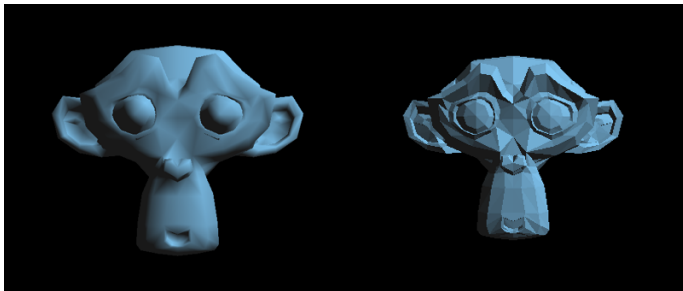
Lambert - Fragment shader

```
#version 400
in vec4 worldPos;
in vec3 worldNor;
out vec4 fragColor;

void main(void)
{
    vec3 lightPosition= vec3(10.0,10.0,10.0);
    vec3 lightDir = ?;

    float diff=
        max(dot(normalize(lightDir),normalize(worldNor))0.0);
    vec4 ambient=vec4(0.1,0.1,0.1,1.0);
    vec4 objectColor = vec4(0.385,0.647,0.812,1.0);
    fragColor=ambient + (diff * objectColor); //lze upravit
}
```


Rozdíl v nastavení Smooth vs. Flat?



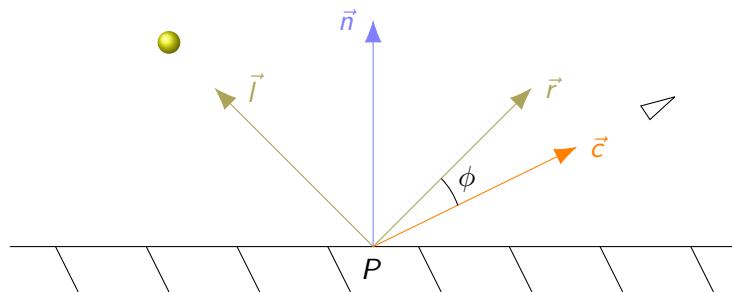
Phongův model navrhnul v roce 1973 v rámci své disertační práce Bui Tuong Phong.

- Ambientní, difúzní a zrcadlová složka
- Vytváří na povrchu odlesky
- Empirický model není založený na fyzice



$$I = I_a + I_d + I_s ;$$

Zrcadlová složka je závislá na úhlu mezi odraženým paprskem \vec{r} a směrem ke kameře \vec{c} .



```
Is=max(dot(reflect(light, normal),camera), 0.0);
```

Phong - Fragment shader

```
#version 400
in vec4 worldPos;
in vec3 worldNor;
out vec4 fragColor;

void main(void)
{
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec= pow(max(dot(viewDir,reflectDir))0.0),32);

    //vec4 specular=specularStrength*spec*lightColor;;

    vec4 objectColor = vec4(0.385,0.647,0.812,1.0);
    //lze upravit I=Ia+Id+Is
    fragColor=ambient+(diff*objectColor)+
                (spec * vec4(1.0,1.0,1.0,1.0));
}
```

Ambientní složka

$$I_{ambient} = I_a \cdot r_a ,$$

Difúzní složka

$$I_{diffuse} = I_d \cdot r_d \max(0, \vec{n} \cdot \vec{l}) ,$$

Zrcadlová složka

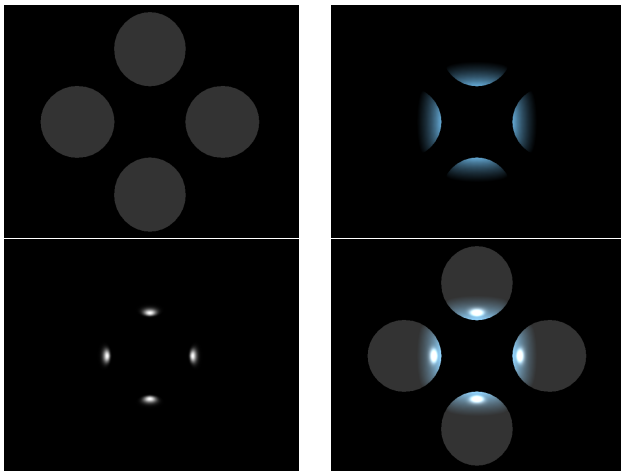
$$I_{specular} = I_s \cdot r_s \max(0, \vec{r} \cdot \vec{c})^s .$$

Obecně lze vzorec zapsat takto:

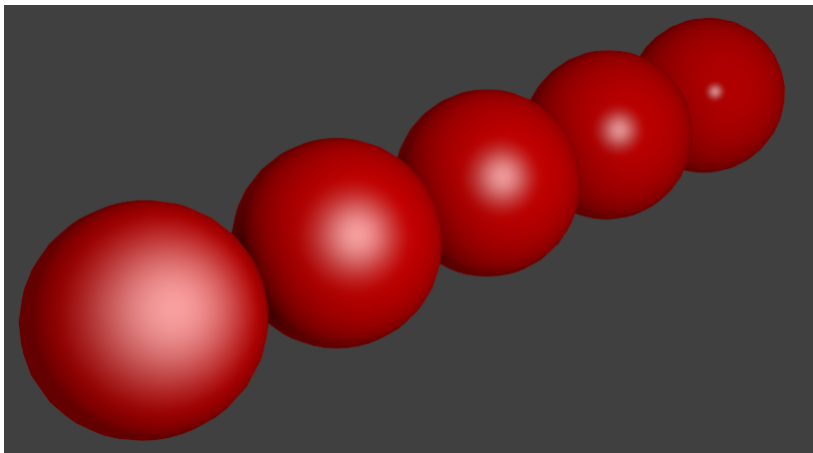
$$I_v = I_a r_a + \sum_{i=0}^m (I_{d,i} r_d \cos \alpha_i + I_{s,i} r_s \cos^h \phi_i)$$

$$I_v = I_a r_a + \sum_{i=0}^m \left(I_{d,i} r_d (\hat{l} \cdot \hat{n}) + I_{s,i} r_s (\hat{r} \cdot \hat{v})^h \right)$$

Phongův osvětlovací model

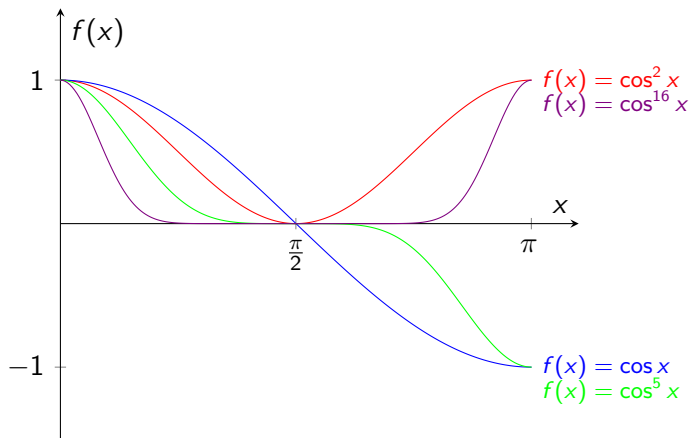


Shininess constant



Shininess constant

Ukázka změny průběhu funkce cosinus s různými koeficienty.




```
const float sphere[17280] = {  
-0.831,-0.555,0.000,-0.833,-0.552,0.000,  
-0.923,-0.382,0.000,-0.924,-0.380,0.000,  
-0.815,-0.555,-0.162,-0.817,-0.552,-0.162,  
...  
-0.375,-0.923,0.074,-0.380,-0.921,0.075,  
-0.555,-0.831,0.000,-0.559,-0.828,0.000};
```

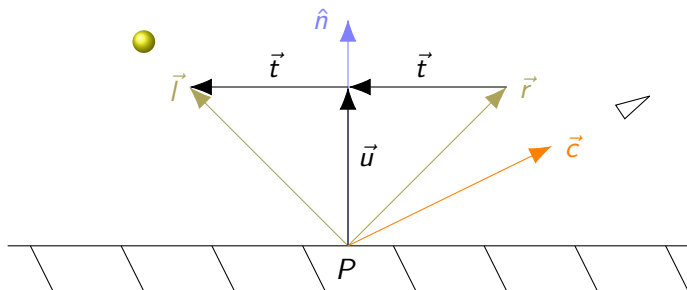


```
glVertexAttribPointer(  
    0,3,GL_FLOAT,GL_FALSE,6*sizeof(float),NULL);  
glVertexAttribPointer(  
    1,3,GL_FLOAT,GL_FALSE,6*sizeof(float),  
    (GLvoid*)(3*sizeof(GLfloat)));
```

Odražený paprsek

Odražený paprsek \vec{r} lze v případě normalizované normály (jednotkové) získat jako

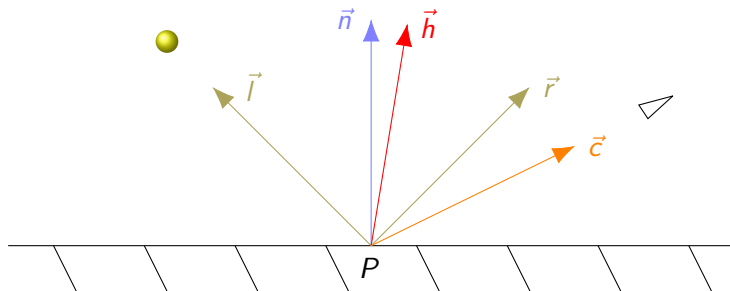
$$\vec{r} = \vec{l} - 2\vec{t} = \vec{l} - 2\hat{n}(\hat{n} \cdot \vec{l}).$$



V GLSL máme funkci `reflect(genType I, genType N);`

Blinn-Phong reflection model

James F. Blinn upravil v roce 1977 výpočet osvětlovacího modelu.



$$\vec{h} = \frac{\vec{l} + \vec{c}}{|\vec{l} + \vec{c}|}$$

```
vec3 halfwayDir = normalize(lightDir + viewDir);
```

$$I_{\text{specular}} = I_d \cdot r_d \max(0, (\vec{n} \cdot \vec{h}))^s$$

Obecně se vychází z toho, že intenzita klesá s druhou mocninou vzdálenosti od světelného zdroje.

$$f_{att}(d) = \frac{1}{k_c + k_l d + k_q d^2}$$

Hodnoty nejsou předem dané, upravujeme je podle požadavků na výslednou scénu.

```
glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);  
glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, 2.0);  
glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 2.0);
```

Stínování popisuje to, jak se nanáší barva na jednotlivé plochy

- Konstantní - jednoduchá a rychlá metoda, používaná pro náhled, vypočítám intenzitu (osvětlovací model) v jednom bodě a použiji pro celou plochu.
- Gouraudovo stínování - vypočítám intenzitu ve vrcholech a pak interpoluji na celé ploše.
- Phongovo stínování - intenzitu (osvětlovací model) určuji v každém bodě samostatně.

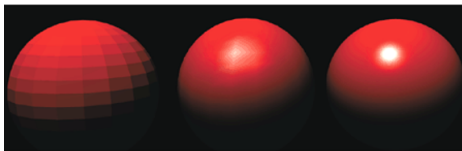
Jak by to fungovalo v našem vertex a fragment shaderu?

Konstantní, Gouraudovo a Phongovo stínování

Stínování popisuje to, jak se nanáší barva na jednotlivé plochy



From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Interglyph Computer Systems



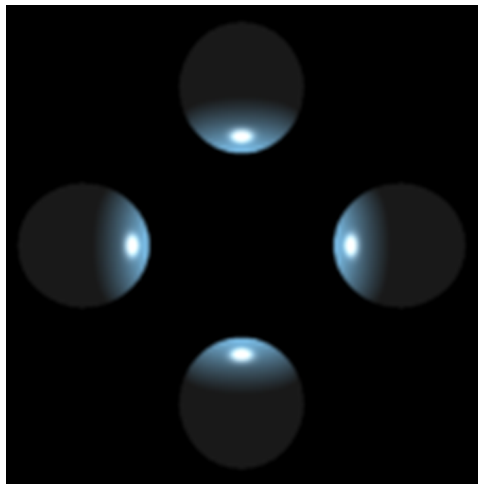
Flat

Gouraud

Phong

Světlo v $[0,0,0]$, čtyři kulové plochy souměrně rozmístěny a kamera nad objekty.

Problém s transformací normály (normálová matice)
... příště na přednášce



Dotazy?