

# Základy počítačové grafiky

## Přednáška 6

Martin Němec

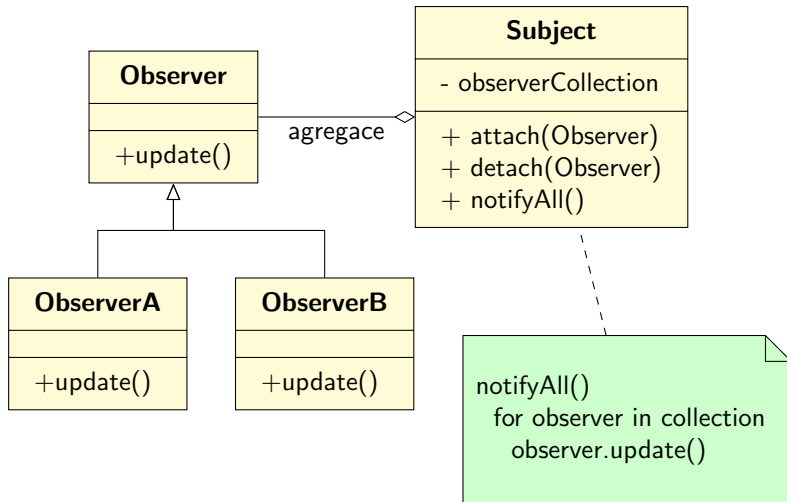
VŠB-TU Ostrava

2024

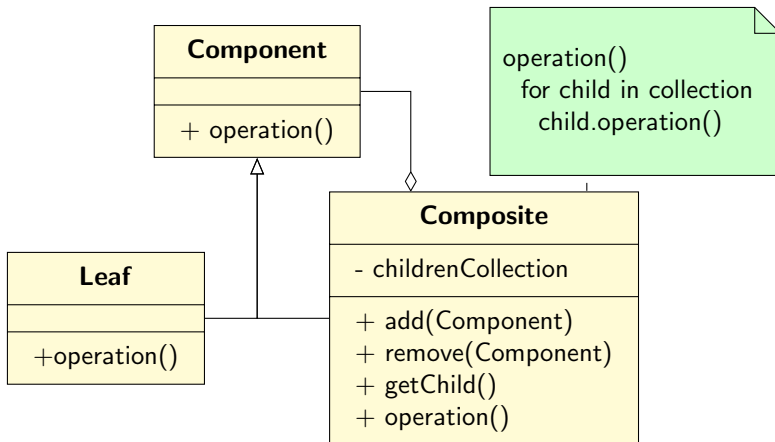
## Doporučuji

- Neprogramujte formou Ctrl+C a Ctrl+V.
- Snažte se dělat menší, ale zkontrolovatelné změny.
- Verzovací systém.

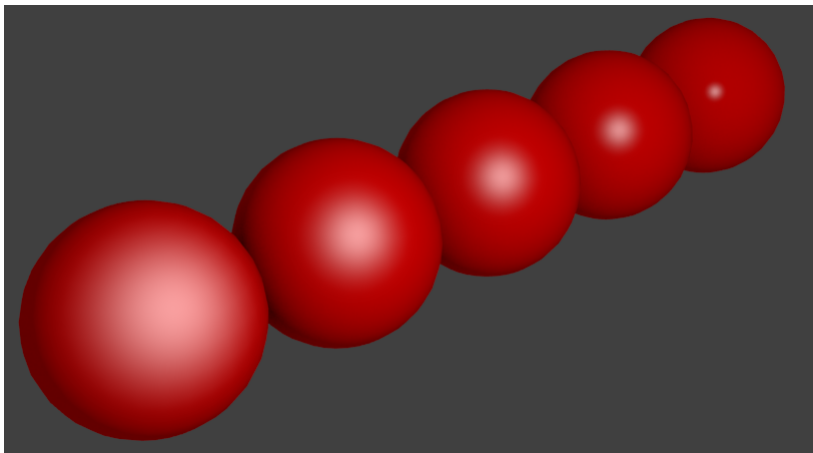
# Observer



# Composite

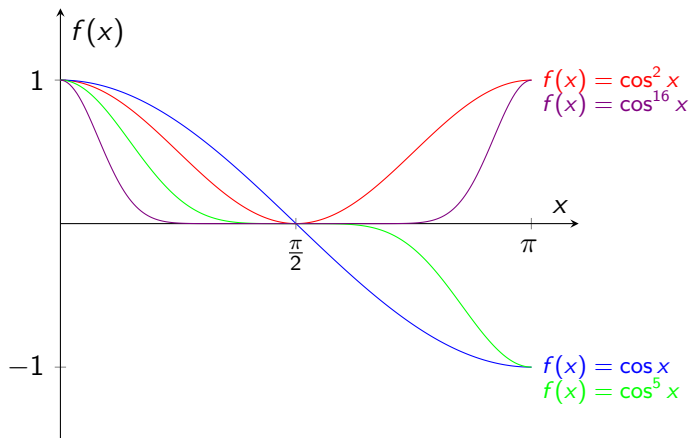


# Shininess constant



# Shininess constant

Ukázka změny průběhu funkce cosinus s různými koeficienty.



# Kulová plocha


```
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)));
```

# Syntax highlighting

Syntax highlighting (file extensions: glsl, frag, vert, geom, comp, tess, tessc).



## GLSL language integration (for VS2022)

Daniel Scherzer | 14,306 installs | ★★★★★ (1) | Free

VSIX Project that provides GLSL language integration. Includes syntax highlighting (file extensions: glsl, frag, vert, geom, comp, tess, tessc), code completion (OpenGL 4.5 + identifiers in shader file), error tagging with squiggles and in error list (error list support is ve...

Download

```
#version 440

out vec4 frag_colour;

in vec4 worldPosition;
in vec3 worldNormal;

void main () {
    //frag_colour = vec4 (0.5, 0.0, 0.5, 1.0);
    frag_colour = vec4(worldNormal,1);
};
```

GLSL language integration (for VS2022).



# Jakou verzi GLSL?

S vývojem grafických karet se vyvíjelo jak OpenGL tak jazyk GLSL.  
Generické datové typy

- `genBType`: booleans
- `genIType`: signed integers
- `genUType`: unsigned integers
- `genType`: floats
- `genDType`: double floats
- `mat`: float matrices
- `dmat`: double matrices

## Nebojte se použít dokumentaci.

### Name

reflect — calculate the reflection direction for an incident vector

### Declaration

```
genType reflect( genType I,  
                 genType N);  
  
genDType reflect( genDType I,  
                 genDType N);
```

### Parameters

*I*

Specifies the incident vector.

*N*

Specifies the normal vector.

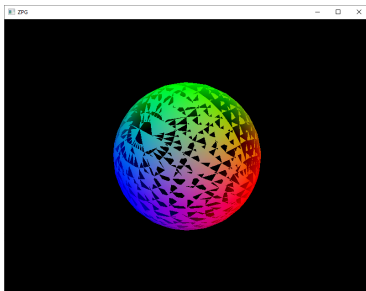
### Description

For a given incident vector *I* and surface normal *N* **reflect** returns the reflection direction calculated as  $I - 2.0 * \text{dot}(N, I) * N$ .  
*N* should be normalized in order to achieve the desired result.

### Version Support

Function Name	OpenGL Shading Language Version											
	1.10	1.20	1.30	1.40	1.50	3.30	4.00	4.10	4.20	4.30	4.40	4.50
reflect (genType)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
reflect (genDType)	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓

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



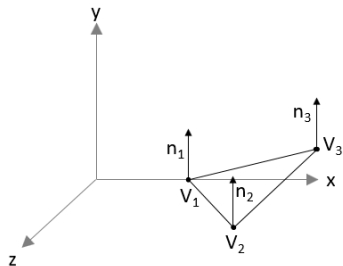
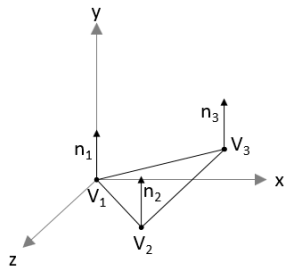
```
glEnable(GL_DEPTH_TEST);  
while (!glfwWindowShouldClose(window)) {  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

# Transformace normálových vektorů

Mějme trojúhelník o vrcholech  $V_1$ ,  $V_2$  a  $V_3$ , kde  $V_1 = [0, 0, 0]$ , normálový vektor  $\vec{n}_1 = (0, 1, 0)$ .

Modelová matice  $\mathbf{M}$  obsahuje translaci  $T(2, 0, 0)$ .

Převedte vrchol  $V_1$  i normálu  $\vec{n}_1$  z object space do world space.



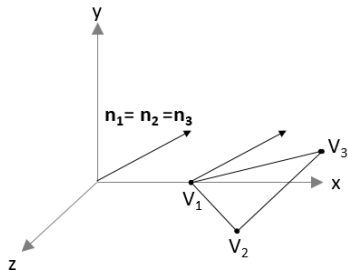
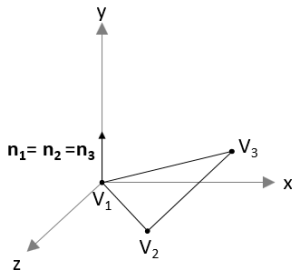
Platí, že  $\vec{n}_1 = \vec{n}_2 = \vec{n}_3 = (0, 1, 0)$ .

# Transformace normálových vektorů

Mějme trojúhelník o vrcholech  $V_1$ ,  $V_2$  a  $V_3$ , kde  $V_1 = [0, 0, 0]$ , normálový vektor  $\vec{n}_1 = (0, 1, 0)$ .

Modelová matice  $\mathbf{M}$  obsahuje translaci  $T(2, 0, 0)$ .

Převedte vrchol  $V_1$  i normálu  $\vec{n}_1$  z object space do world space.



Platí, že  $\vec{n}_1 = \vec{n}_2 = \vec{n}_3 = (2, 1, 0)$ .

Normálové (obecně) vektory se nesmí transformovat běžnými transformačními maticemi!

Můžeme převést pozici světla (bod) do souřadnicového systému modelu (z globalního do lokálního).

Můžeme využít inverzní matici  $\mathbf{M}\mathbf{M}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$ .

$$V'_1 = \mathbf{M}V_1$$

Potom platí:

$$\mathbf{M}^{-1}V'_1 = \mathbf{M}^{-1}\mathbf{M}V_1$$

$$\mathbf{M}^{-1}V'_1 = \mathbf{I}V_1$$

$$V_1 = \mathbf{M}^{-1}V'_1$$

Tečna  $\vec{t}$  je zadána dvěma vrcholy ( $\vec{t} = V_2 - V_1$ ).

$$\mathbf{M}\vec{t} = \mathbf{M}(V_2 - V_1)$$

resp.

$$\mathbf{M}\vec{t} = \mathbf{M}V_2 - \mathbf{M}V_1$$

Transformovaný tečný vektor je dán

$$\vec{t}' = V'_2 - V'_1$$

Normálový vektor není dán dvěma vrcholy, ale kolmostí k povrchu. Nelze použít přímo modelovou matici, chceme provést pouze rotace.

$$\text{dot}(\vec{n}, \vec{t}) = 0$$

$$\text{dot}(\vec{n}', \vec{t}') = 0$$

$$\vec{n}^T \vec{t} = 0$$

$$\vec{n}'^T \vec{t}' = 0$$

$$\vec{t}' = \mathbf{M} \vec{t}$$

$$\vec{n}' = \mathbf{N} \vec{n}$$

$$\text{dot}(\vec{n}', \vec{t}') = 0$$

$$(\mathbf{N} \vec{n})^T (\mathbf{M} \vec{t}) = 0$$



$$\vec{t}' = \mathbf{M}\vec{t}$$

$$\vec{n}' = \mathbf{N}\vec{n}$$

$$\begin{aligned} \text{dot}(\vec{n}', \vec{t}') &= 0 \\ (\mathbf{N}\vec{n})^T (\mathbf{M}\vec{t}) &= 0 \end{aligned}$$

Protože transpozice součinu matic je součin transponovaných matic v opačném pořadí  $(A \cdot B)^T = B^T \cdot A^T$ , dostaneme:

$$\vec{n}^T \mathbf{N}^T \mathbf{M} \vec{t} = 0$$

A protože  $\vec{n}^T \vec{t} = 0$ , pak musí platit

$$\mathbf{N}^T \mathbf{M} = \mathbf{I}$$

a odtud

$$\mathbf{N} = (\mathbf{M}^{-1})^T$$

# Vertex shader

```
uniform mat4 model;  
uniform mat4 view;  
uniform mat4 projection;  
  
void main(void)  
{  
    worldPos = model * vec4(vp,1.0);  
    //normalova matice  
    mat3 normal=transpose(inverse(mat3(model))) ;  
    worldNorm = normal*vn;  
    gl_Position = projection*view*model*vec4(vp,1.0);  
}
```

Počítat inverzní matici je možné jak na CPU tak i GPU.

```
glm::mat3 N=glm::transpose(glm::inverse(glm::mat3(M)));
```

K čemu nám může být toto ?

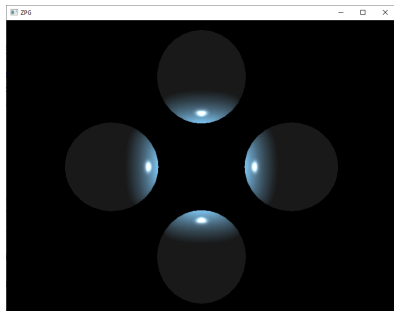
```
glm::mat3 N=glm::transpose(glm::inverse(glm::mat3(VM)));
```

# Testování 1

Světlo ( $S=[0,0,0]$ ,  $h=40$ ) ,  
čtyři kulové plochy souměrně  
rozmístěny kolem světla na  
jednotlivých osách a kamera  
nad objekty.

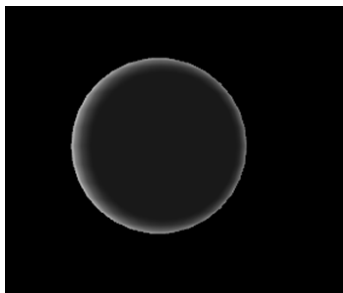
Kontrola správnosti Phongova  
osvětlovacího modelu (správně  
ambientní, difuzní i zrcadlová  
složka.).

Problémy se správným  
výpočtem (příp. s transfor-  
mací normály).



## Testování 2

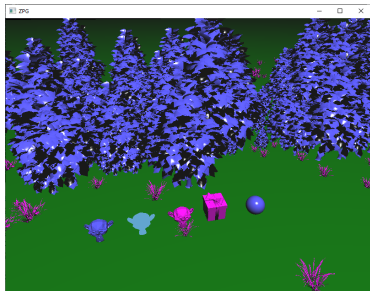
Světlo na špatné straně. Kulová plocha mezi světlem a kamerou, světlo má  $h=1$ . Přesvit na druhou stranu.



```
// light source on the wrong side?  
if (dot(normalDirection, lightDirection) < 0.0) {  
    specularReflection = vec4(0.0, 0.0, 0.0, 0.0);  
}
```

# Testování 3

Test na více modelů ve scéně s více shadery (min. konstantní, lambert, phong a blinn). Při změně obrazovky se nesmí tělesa deformovat (glViewport a perspective).



**Dotazy?**