PV112 Programování grafických aplikací

Jaro 2017

Výukový materiál

---

5. přednáška: Osvětlení a materiály

Osvětlení

Nové OpenGL již nepodporuje vytváření osvětelní formou vestavěných funkcí, proto si uživatel musí vše nadefinovat sám. V této přednášce si ukážeme, jak tedy se světly pracovat.

Dále je v případě výpočtu osvětlení dávat pozor na to, ve kterých souřadnicových systémech budou osvětlení počítat. Na přednášce o transformacích jsme se seznámili se dvěma možnými pohledy na souřadné systémy, globálním a lokálním. Na cvičeních budete používat globální přístup, kdy se budou vrcholy a jejich normály (které hrají při osvětlení zásadní roli) násobit Model maticí.

Při osvětlování scény jsou zásadní normálové vektory. **Normála roviny** je vektor délky 1, který je kolmý na tuto rovinu. Podobně **normála trojúhelníka** je tedy jednotkový vektor kolmý na trojúhelník. Dá se jednoduše spočítat jako vektorový součin dvou jeho hran a výsledek následně normalizovat. Následující pseudokód demonstruje tento výpočet.

```
triangle(v1, v2, v3);
edge1 = v2 - v1;
edge2 = v3 - v1;
triangle.normal = cross(edge1, edge2).normalize();
```

Pozor na možnou záměnu pojmů normála a normalizace. Normalizací je myšleno dělení vektoru jeho délkou, tedy výsledný vektor má délku 1.

Normálu vrcholu lze spočítat z normál okolních trojúhelníků, jak ukazuje následující pseudokód.

```
vertex v1, v2, v3, ....;
triangle tr1, tr2, tr3;       // všechny sdílí vrchol v1
v1.normal = normalize(tr1.normal + tr2.normal + tr3.normal);
```

Použití normál v OpenGL je jednoduché. Normála je podobně jako pozice či barva dalším atributem vrcholu, bude se s ní tedy zacházet stejným způsobem.

```
GLuint normalbuffer;
glGenBuffers(1, &normalbuffer);
glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
glBufferData(GL_ARRAY_BUFFER, normals.size() *
        sizeof(glm::vec3), &normals[0], GL_STATIC_DRAW);


glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
glVertexAttribPointer(
     2,         // attribute
     3,         // size
     GL_FLOAT, // type
     GL_FALSE, // normalized?
     0,         // stride
     (void*)0  // array buffer offset
);
```

Normálová matice

Normálové vektory nejsou transformovány stejným způsobem jako vrcholy nebo vektory určující pozici. Matematicky je lepší si normálové vektory představit ne jako vektory, ale jako roviny kolmé k těmto vektorům. Následné transformace normálových vektorů mohou být tedy popsány transformacemi jejich kolmých rovin.

Homogenní rovina je popsána řádkovým vektorem *(a, b, c, d)*, kdy alespoň jedna z proměnných *a, b, c, d* je nenulová. Pokud vezmeme *q* jako nenulové číslo, pak *(qa, qb, qc, qd)* reprezentuje stejnou rovinu. Bod $(x,y,z,w)^T$ leží v rovině *(a, b, c, d)*, jestliže $ax+by+cz+dw = 0$ (pokud $w = 1$, jedná se o rovnici Euklidovské roviny). Aby rovnice mohla reprezentovat Euklidovskou rovinu, musí alespoň jedna z hodnot *a, b, c* být nenulová. Pokud jsou všechny nulové, pak *(0, 0, 0, d)* reprezentuje nekonečnou rovinu obsahující všechny body v nekonečnu.

Pokud **p** je homogenní rovina a **v** je homogenní vrchol, pak skutečnost, že **v** leží na **p** je matematicky zapsáno jako **pv** = 0, kde **pv** je násobení normálovou maticí. Pokud nyní vezmeme nějakou transformaci **M** (zapsánu maticí 4x4, která má inverzní matici $\mathbf{M}^{-1}$), pak **pv** = 0 je ekvivalentní k $\mathbf{pM}^{-1}\mathbf{Mv} = 0$, tedy **Mv** leží v rovině $\mathbf{pM}^{-1}$. $\mathbf{pM}^{-1}$ je tedy obrazem roviny **p** transformované pomocí **M**.

Pokud se nyní vrátíme k reprezentaci normálového vektoru pomocí vektoru místo k němu kolmé roviny, pak mějme vzájemně kolmé vektory **v** a **n**. Pak $\mathbf{n}^T\mathbf{v} = 0$. Pak pro libovolnou

transformaci **M** máme $\mathbf{n}^T\mathbf{M}^{-1}\mathbf{Mv} = 0$, což znamená, že $\mathbf{n}^T\mathbf{M}^{-1}$ je transponovaná matice transformovaného normálového vektoru. Transformovaný normálový vektor je tedy $(\mathbf{M}^{-1})^T\mathbf{n}$. Jinými slovy, normálové vektory jsou transformovány **inverzní transponovanou maticí** transformace, která je aplikována na vrcholy.

Osvětlení a materiály

V OpenGL je barva objektů a výsledných pixelů na obrazovce reprezentována trojicí barevných složek – RGB. Jejich kombinací lze vytvořit jakoukoliv viditelnou barvu.

Pro výpočet barvy objektů ve scéně lze použít takzvaný Phongův osvětlovací model, který je jednoduchý a dává dobré výsledky. Tento model se snaží co nejvíce zjednodušit fyzikální a optické vlastnosti reálného světla tak, aby výpočty osvětlení byly rychlé a aby se přitom výsledný obrázek dostatečně blížil realitě. V Phongově osvětlovacím modelu se světlo, které na určitý objekt dopadá a následně se od něj odráží, rozkládá na tři světelné složky – **ambientní**, **difúzní** a **spekulární**.

Ambientní složka

Tato složka vyjadřuje okolní světlo, které není přímo vyzařováno ze žádných světelných zdrojů. Toto světlo vzniká tak, že se světlo z nějakého zdroje několikrát odrazí (například od stěn v místnosti), a není tedy jasně patrný směr, odkud přichází. V Phongově osvětlovacím modelu je toto světlo považováno za všesměrové, tj. osvětluje objekt ze všech směrů nezávisle na poloze a orientaci zdrojů světla. Nezávisle na 3D tvaru objektu se objekt jeví jako plošný, ztrácíme tedy prostorový vjem.

Difúzní složka

Tato světelná složka vyjadřuje světlo, které na povrch objektu dopadá z konkrétního světelného zdroje a od povrchu objektu se odráží do všech směrů se stejnou intenzitou. Pomocí difúzní složky lze tedy vytvářet matné materiály. V Phongově osvětlovacím modelu jsou výpočty difúzní složky světla upraveny tak, že výsledná barva difúzní složky je závislá pouze na vzájemné orientaci světelného zdroje a objektu. Poloha pozorovatele (kamery) nemá vliv na barvu ani intenzitu difúzní složky, čehož se dá využít například při urychlování vykreslovacích algoritmů.

Spekulární složka

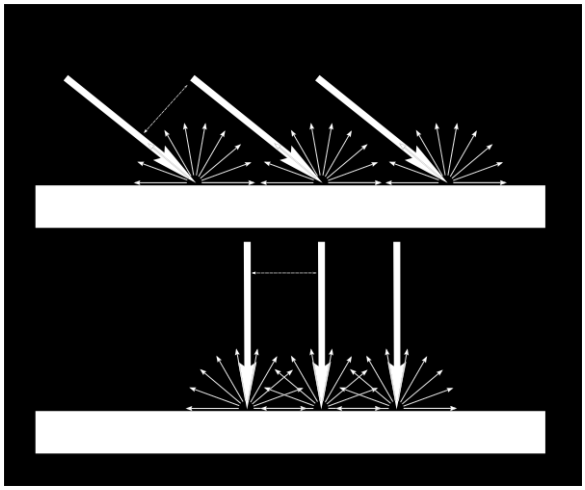
Poslední světelnou složkou použitou v Phongově osvětlovacím modelu jsou *odlesky* (*specular*). Odlesky vznikají ve chvíli, kdy na povrch tělesa dopadá paprsek ze zdroje světla a tento paprsek se od povrchu odráží podle známého zákona odrazu a lomu. Ve skutečnosti se paprsek nikdy neodrazí ideálně, vždy nastává určité rozptýlení. Phongův osvětlovací model tento fenomén modeluje tak, že se stanoví koeficient změny intenzity odraženého světla podle úhlu odklonu počítaného odraženého paprsku od paprsku ideálně odraženého. Vektory těchto dvou paprsků se vynásobí a výsledek se umocní zadaným koeficientem.

Při vykreslování reálných těles se téměř nikdy nepoužívá pouze jedna světelná složka, vždy se jedná o jejich vzájemnou kombinaci. Obrázky ukazují libovolné kombinace.

Vlevo nahoře – ambientní a difúzní složka
Vpravo nahoře – ambientní složka s odlesky
Vlevo dole – difúzní složka s odlesky
Vpravo dole – ambientní a difúzní složka s odlesky

Vliv normály

U difúzní složky hrají normály zásadní úlohu. Při styku paprsku světla s povrchem objektu se světlo odrazí do všech směrů. Úroveň osvětlení povrchu však závisí na tom, z jakého úhlu světlo přišlo. Když světlo dopadá kolmo na povrch, je odražené světlo koncentrováno na poměrně malém povrchu. Čím menší je úhel mezi paprskem a povrchem, tím větší povrch je osvětlen. To znamená, že každý bod povrchu se bude jevit jako tmavší, ale tím, že paprsek osvětlí větší plochu, tak nedojde ke ztrátě celkového množství světla – to zůstane stejné. To znamená, že když počítáme barvu daného pixelu, hraje roli i úhel mezi příchozím paprskem a normálou povrchu.



Pokud si v OpenGL chceme nastavit parametry světelného zdroje, musíme si vše nastavit sami. Například si nadefinujeme a budeme používat následující uniform proměnné:

```
uniform vec4 light_position;

uniform vec3 light_ambient_color;

uniform vec3 light_diffuse_color;

uniform vec3 light_specular_color;
```

Materiály v OpenGL

V OpenGL se kromě parametrů světelných zdrojů musí také nastavovat optické vlastnosti materiálů, ze kterých jsou tělesa vytvořena. Vlastnost materiálu se může měnit pro přední a zadní stranu vykreslovaných polygonů, musíme si ale sami tyto vlastnosti nadefinovat zvlášť.

Phongův osvětlovací model počítá difúzní světelnou složku a odlesky s využitím informací o normálách ve vrcholech povrchu tělesa.

Vlastnosti materiálu si podobně jako u světel musíme nastavit sami. Můžeme si tedy nadefinovat a poté využívat následující parametry:

```
uniform vec3 material_ambient_color;

uniform vec3 material_diffuse_color;

uniform vec3 material_specular_color;

uniform float material_shininess;
```

Pro materiály platí následující skutečnosti:

- Difúzní složka je pro rozpoznávání barev nejdůležitější
- Difúzní odrazivost (reflectance) nezávisí na pozici pozorovatele
- Spekulární odrazivost závisí na pozici pozorovatele
- V reálném světě je difuzní a ambientní složka ekvivalentní

Typy světel v OpenGL

V OpenGL již nejsou defaultně podporována žádná světla, ale mezi nejjednodušší, které lze použít (a které byly v dřívějších verzích OpenGL podporovány) patří směrová, bodová a reflektorová světla. Proto se v dalším vysvětlování zaměříme právě na tyto tři typy světel.

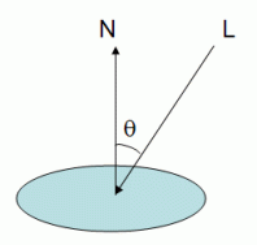
Směrová světla

Směrové světlo nemá žádný pevně daný světelný zdroj. Světlo se šíří z jednoho směru (je zadáno vektorem) nezávisle na vzájemné poloze a vzdálenosti objektů v zobrazované scéně. Za směrové světlo lze považovat například slunce, neboť světelné paprsky, které dopadají na povrch těles ze slunce, jsou víceméně paralelní (vzhledem k poměru vzdáleností slunce a těles od sebe).

Difúzní složka se spočítá podle rovnice

$$I_o = L_d * M_d * \cos(\theta)$$

kde I_0 je intenzita odraženého světla, L_d je barva difúzního světla a M_d je difúzní koeficient materiálu.



Podle Lambertova zákona je množství odraženého světla přímo úměrné kosinu úhlu Θ. Povrch objektu je osvětlen světelným zdrojem pouze tehdy, když je velikost úhlu Θ mezi 0 a 90ti stupni.

Ve vertex shaderu je aktuální pozice vrcholu transformována do souřadnic oka (pomocí vynásobení modelview maticí). Normála je poslána do fragment shaderu.

Vertex shader může vypadat následovně:

```
in vec4 position;
in vec3 normal;
out vec3 N;
out vec3 v;
uniform mat4 myModelViewMatrix;
uniform mat4 myModelViewProjectionMatrix;
uniform mat3 myNormalMatrix;
void main()
{
     v = vec3(myModelViewMatrix * position);
     N = normalize(myNormalMatrix * normal);
     gl_Position = myModelViewProjectionMatrix * position;
}
```

Pro každý fragment je spočten Lambertův zákon. Odpovídající fragment shader může vypadat následovně:

```
in vec3 N;
in vec3 v;
out vec4 final_color;
uniform vec3 material_diffuse_color;
uniform vec4 light_position;
uniform vec3 light_diffuse_color;
void main()
{
vec3 L = normalize(light_position.xyz); // Toto platí pro
          směrové světlo, tedy za předpokladu, že
          light_position je místo v nekonečnu, ODKUD světlo
          svítí
     vec3 normN = normalize(N);
     vec3 Idiff = light_diffuse_color * material_diffuse_color
          * max(dot(normN, L), 0.0);
     Idiff = clamp(Idiff, 0.0, 1.0);
     final_color = vec4(Idiff, 1.0);
}
```

Ambientní složka ovlivní i povrchy, které nebyly přímo osvětleny světelným zdrojem.

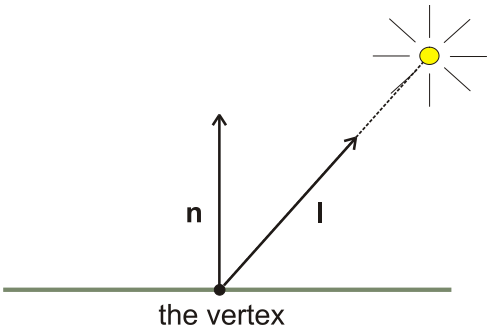
```
ambient = light_ambient_color * material_ambient_color
```

Barva je počítána podle vzorců

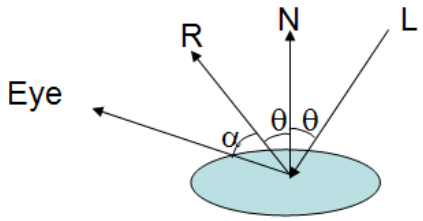
$$ambient = ambient_{light} * ambient_{material}$$

$$diffuse = (max\{\mathbf{l}.\mathbf{n},0\})\ diffuse_{light} * diffuse_{material}$$

Jak již bylo řečeno, difúzní složka nezávisí na pozici pozorovatele. Difúze je největší, když světlo dopadá kolmo na plochu.



Spekulární složka se počítá následovně. Phongův model říká, že spekulární složka směrového světla odpovídá kosinu mezi vektorem odrazu a vektorem směřujícím k oku pozorovatele.



- L = vektor směřující od zdroje světla do vrcholu na povrchu objektu
- N = normálový vektor

- Eye = vektor směřující od vrcholu do oka pozorovatele (nebo kamery)
- R = vektor odrazu (světla ze směru vektoru L)

Spekulární složka je pak úměrná kosinu úhlu α.

Pokud má Eye vektor stejný směr jako R, je intenzita spekulárního odrazu největší. Úbytek intenzity (útlum) je úměrný vektoru mezi Eye a R. Útlum je řízen parametrem *shininess.* Vysoká hodnota parametru shininess zmenšuje velikost osvětlené oblasti na objektu a naopak. Jinými slovy, hodnota shininess řídí v OpenGL velikost osvětleného bodu na objektu.

Shininess = 8 *Shininess = 64* *Shininess = 128*

Rovnice pro vektor odrazu pak vypadá následujícím způsobem:

$$R = -2N(L \cdot N) + L$$

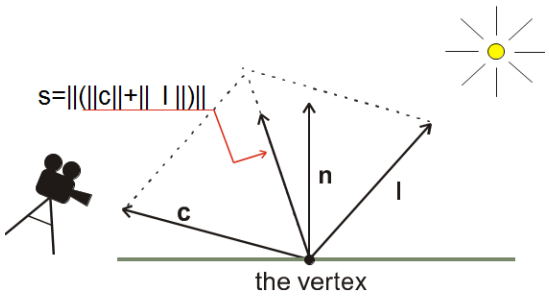
Spekulární složka Phongova modelu se spočítá jako:

$$Spec = (R \cdot Eye)^{s} * L_s * M_s$$

kde exponent *s* odpovídá hodnotě parametru *shininess*, L_s je intenzita spekulárního světla a M_s je spekulární koeficient materiálu.

$$\text{specular} = (\max\{\mathbf{s}.\mathbf{n},0\})^{\text{shininess}}\ \text{specular}_{\text{light}} * \text{specular}_{\text{material}}$$

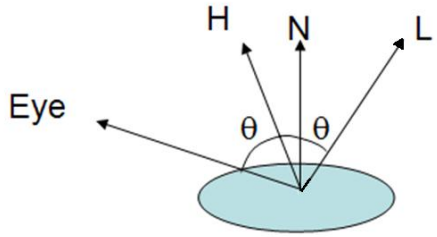
Spekulární odraz závisí na pozici pozorovatele



Blinn-Phongův model

Blinn-Phongův osvětlovací model je zjednodušením Phongova modelu.

Blinn navrhl jednodušší a rychlejší model, jeho myšlenka je založena na použití tzv. half-vektoru. Half-vektor je vektor se směrem mezi Eye vektorem a vektorem světla.



Intenzita spekulární složky světla je nyní odvozena z kosinu úhlu mezi half-vektorem a normálou povrchu.

$$H = Eye + L$$

Rovnice pro výpočet half-vektoru je tedy mnohem jednodušší než rovnice pro výpočet vektoru odrazu.

Vektory Eye, L a následně i H musí být normalizovány

Spekulární složka se spočte následovně:

$$Spec = (N \cdot H)^{s} * L_{s} * M_{s}$$

Směrové světlo lze počítat i na úrovni pixelů. V podstatě se jedná o rozdělení práce mezi vertex a fragment shader, některé operace jsou tedy prováděny ne ve vertex shaderu, ale ve fragment shaderu. Možné vizuální změny jsou vidět na obrázku. Pro více detailů odkazujeme na referenční příručku.

Per Vertex

Per Pixel

Bodová světla

Nejjednodušší (alespoň pro pochopení, programová implementace je poměrně složitá) je *bodové světlo*. Toto světlo si můžeme představit například jako žárovku v prostoru, která vyzařuje světlo na všechny strany se stejnou intenzitou. Intenzita klesá se vzdáleností od světelného zdroje.

Bodové světlo je definováno dvěma základními parametry:

- *w* = pozice světla (hodnota 0 odpovídá směrovému světlu)
- *attenuation* = útlum světla (konstatní, lineární, kvadratický)

Útlum světla lze počítat různými způsoby. Útlum je způsoben rozbíháním světelných paprsků, odrazy, lomy a pohlcováním světla vlivem pevných prachových částeček a páry, které se ve vzduchu nacházejí. Následujícím způsobem získáme jednoduchou simulaci útlumu světla. Jednoduchá simulace kombinuje konstantní c_1, lineární c_2 a kvadratický c_3 útlum. Tyto koeficienty jsou kombinovány v rovnici

$$attenuation = \frac{1}{(c_1 + c_2 d + c_3 d^2)}$$

kde *d* je vzdálenost světla od osvětlovaného vrcholu.

Výsledná barva se opět počítá tak, že ambientní, difúzní a spekulární složky světla jsou **sečteny**!!!

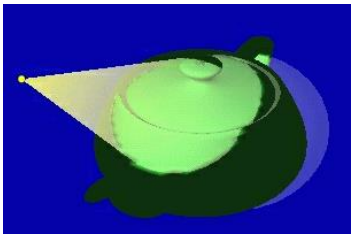
$$color = global\ ambient_{light} * ambient_{material} + \Sigma\ light$$

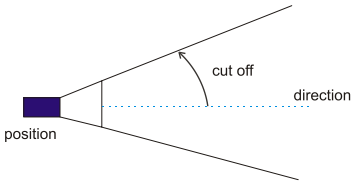
$$light = attenuation * spot * (ambient + diffuse + specular)$$

$$attenuation = 1/(const + d * linear + d^2 * quadratic)$$

Reflektorová světla

Nejsložitějším světlem, které si ukážeme, je reflektorové světlo. Tento typ světelného zdroje se chová podobně jako klasický reflektor, tj. světlo se šíří (stejně jako u bodového světla) z jednoho bodu, ale ne do všech směrů, nýbrž pouze v předem zadaném kuželu. U tohoto světla se také musí zadat nejvíce parametrů. Kromě pozice světla (tj. kde je umístěn reflektor) se musí zadat také hlavní směr šíření světelných paprsků (kam je reflektor namířen) a úhel síření světla (světelný kužel), ve kterém intenzita světla postupně klesá směrem k nule.





`spotDirection` směr refl. světla (0,0,-1)

`spotCosCutoff` úhel ořezání refl. světla

`spotExponent` jak se intenzita světla snižuje od středu světelného kužele k jeho stěnám

Výpočet osvětlení je velmi podobný jako pro bodové světlo, pouze útlum musí být vynásoben efektem pro reflektorové světlo podle následující rovnice:

$$spotEffect = (spotDirection \cdot lightDir)^{spotExp}$$

kde *spotDirection* je definovaný parametr světla, *lightDir* je vektor od zdroje světla k vrcholu a *spotExp* je útlum v rámci kužele světla (parametr *spotExponent* světla).

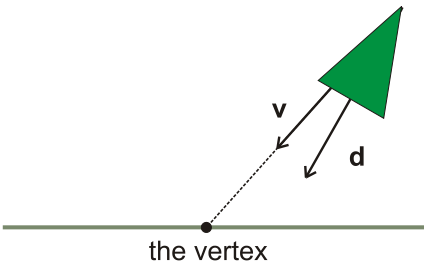
Čím větší hodnota parametru *spotExp*, tím větší útlum. Hodnota 0 značí situaci, kdy je v kuželu světla všude stejná intenzita osvětlení.

spot = 1 jestliže světlo není reflektorové (CUTOFF je 180^{o})

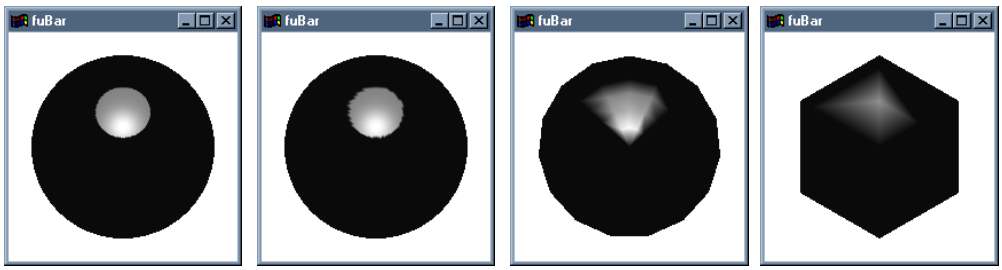
0 jestliže je bodové a vrchol je mimo kužel

$\max\{\mathbf{v}.\mathbf{d},0\}^{GL\_SPOT\_EXPONENT}$

Ve skutečnosti se provádí kontrola, jestli je vrchol mimo kužel a provádí skalární součin dvou vektorů.



Je nutné si uvědomit, že bodová světla pracují dobře pouze s precizní reprezentací objektů. To je důsledkem toho, že světlo je počítáno na vrcholech objektu.



Toto lze optimalizovat tím, že převedeme část funkcionality z vertex shaderu na fragment shader. Výsledek může vypadat následovně:

Per Pixel

Toon shading

Toon shading je pravděpodobně ten nejjednodušší nefotorealistický shader, který lze vytvořit. Používá velmi málo barev, často pouze tóny jedné barvy a přitom zachovává 3D dojem hloubky.

Tóny použité na konvičce jsou vybrány na základě kosinu úhlu mezi směrem světla a normálou povrchu. Pokud tedy máme normálu, která je velmi podobná směru světla, použijeme nejsvětlejší tón barvy. Čím větší je úhel mezi těmito dvěma vektory, tím tmavší tón je použit.

V následujícím příkladě se počítá toon shading pro jednotlivé vrcholy objektu. Ve vertex shaderu si nadefinujeme uniform proměnnou definující směr světla:

```
uniform vec3 lightDir;
```

Následuje výpočet kosinu úhlu a uložení výsledku do proměnné *intensity*:

```
intensity = dot(lightDir, gl_Normal);
```

Fragment shader poté spočítá barvu fragmentu s ohledem na intenzitu světla. Používá interpolovanou hodnotu intenzity na vrcholech pro výpočet tónu barvy fragmentu.

```
vec4 color;

if (intensity > 0.95)
        color = vec4(1.0,0.5,0.5,1.0);
else if (intensity > 0.5)
        color = vec4(0.6,0.3,0.3,1.0);
else if (intensity > 0.25)
        color = vec4(0.4,0.2,0.2,1.0);
else
        color = vec4(0.2,0.1,0.1,1.0);
```

Výsledné nefotorealistické zobrazení pak může vypadat následovně: