Grafika interaktywna

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Oświetlenie

Oświetlenie występujące w świecie rzeczywistym jest zbyt złożone, aby dokładnie je odwzorować w aplikacjach komputerowych (zwłaszcza czasu rzeczywistego). Konieczne są pewne modele i uproszczenia (czasem bardzo duże).

W grafice komputerowej rozpatruje się dwa rodzaje oświetlenia:

- globalne – rozpatrywana jest cała scena jednocześnie, uwzględniane jest dowolnie wiele odbić,
- lokalne – rozpatrywane są tylko źródła światła i obiekt, obliczane jest tylko jedno odbicie.

W grafice czasu rzeczywistego na ogół stosuje się oświetlenie lokalne z pewnymi elementami oświetlenia globalnego np.

- cienie,
- mapowanie środowiska.

Radiometria

Wiemy, że światło to fala elektromagnetyczna. Radiometria zajmuje się pomiarami energii fal elektromagnetycznych.

Irradiancję (natężenie napromieniowania powierzchni) E definiujemy jako strumień padający na element powierzchni odbiornika do wielkości tej powierzchni.

Intensywność (natężenie promieniowania) I jest to strumień energii wypromieniowany przez punktowe źródło światła w danym kierunku w kącie bryłowym.

Radiancja (luminancja energetyczna) L jest to liczba fotonów docierających w jednostce czasu do małej powierzchni z określonego kierunku. Opisuje ona ilość energii wysyłanej przez skończone źródło światła.

Modele źródeł światła

W grafice czasu rzeczywistego wyróżniamy kilka rodzajów źródeł światła:

- kierunkowe (ang. directional),
- punktowe wielokierunkowe (ang. omni),
- punktowe typu reflektor (ang. spotlight),
- powierzchniowe (ang. area).

Źródło światła kierunkowego

Jest to źródło oddalone bardzo daleko, np. Słońce. Jego światło podróżuje w jednym kierunku na całej scenie, a promienie są równoległe.

Na potrzeby renderingu kierunek tego światła jest opisany za pomocą wektora l, który najczęściej jest unormowany oraz irradiancji E_L. Kierunek wektora l najczęściej pokazuje kierunek przeciwny do kierunku podróżowania światła. Irradiancja jest reprezentowana przez kolor.

Źródło światła punktowego

Źródło takie dane jest przez pozycję p_L oraz intensywność I_L.

W ogólności I_L jest funkcją kierunku. Światło o stałej wartości I_L nazywane jest światłem wielokierunkowym (ang. omni light), a samo I_L reprezentowane jest przez kolor.

Przy obliczeniach cieniowania potrzebować będziemy kierunku światła l oraz irradiancji E_L. Można je obliczyć następująco:

$$r = \|p_L - p_S\|,$$
$$l = \frac{p_L - p_S}{r},$$
$$E_L = \frac{I_L}{r^2},$$

gdzie p_S jest to punkt na powierzchni.

Zmniejszanie E_L proporcjonalnie do $1/r^2$ jest fizycznie poprawne, ale często preferowane jest użycie innej funkcji zmniejszania E_L w zależności od odległości. Funkcję taką nazywamy funkcją tłumienia.

W takim przypadku wzór na irradiancję przyjmuje postać:

$$E_L = I_L f_d(r),$$

gdzie f_d jest funkcją tłumienia.

W OpenGL (z nieprogramowalną funkcjonalnością) i DirectX używana jest następująca funkcja tłumienia:

$$f_d(r) = \frac{1}{s_c + s_l r + s_q r^2},$$

gdzie s_c, s_l, s_q są właściwościami źródła światła. Funkcja ta ma tę wadę, że nigdy nie osiąga 0.

Często stosowaną w grach i aplikacjach graficznych funkcją tłumienia jest:

$$f_d(r) = \begin{cases} 1, & \text{gdy } r \leqslant r_{start}, \\ \frac{r_{end} - r}{r_{end} - r_{start}}, & \text{gdy } r_{start} < r < r_{end}, \\ 0, & \text{gdy } r \geqslant r_{end}, \end{cases}$$

gdzie r_{start}, r_{end} są właściwościami źródła światła.

Przy renderowaniu filmów Pixar korzysta m.in. z następującej funkcji tłumienia:

$$f_d(r) = \begin{cases} f_{max} e^{k_0 (r/r_c)^{-k_1}}, & \text{gdy } r \leqslant r_c, \\ f_c \left(\frac{r_c}{r}\right)^{s_e}, & \text{gdy } r > r_c. \end{cases}$$

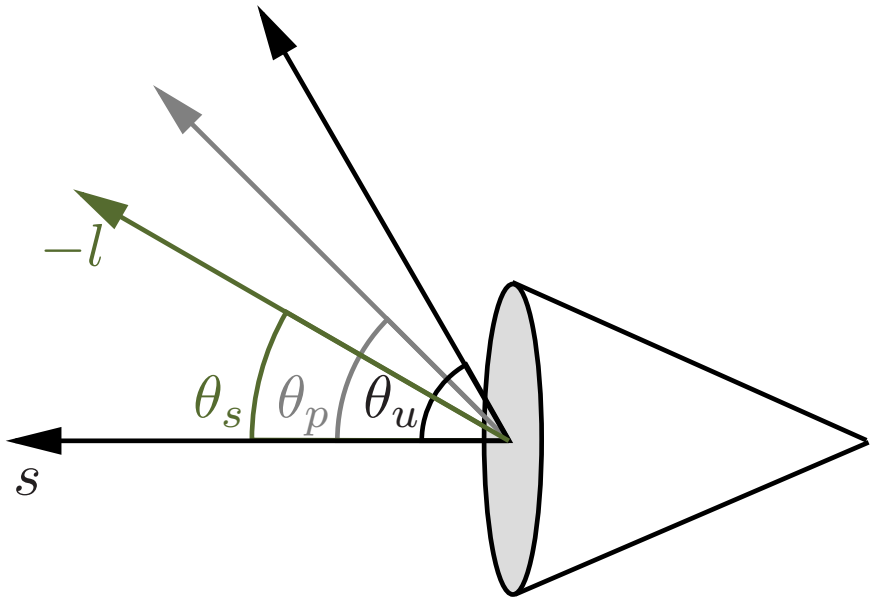
Funkcja osiąga wartość f_c w odległości r_c. W odległościach mniejszych niż r_c stopniowo zbliża się do f_{max}. Wykładnik s_e kontroluje w jakim stopniu funkcja maleje wraz z odległością większą niż r_c. Stałe k_0, k_1 są ustalane tak, aby zapewnić ciągłość pochodnej w punkcie przejścia, tzn. $k_0 = \ln(f_c/f_{max})$, $k_1 = s_e/k_0$.

Funkcja ta nie osiąga 0 i jest kosztowna do obliczenia.

Źródło światła punktowego typu reflektor

W odróżnieniu od światła wielokierunkowego w świetle typu reflektor I_L zmienia się wraz z kierunkiem.

Światło tego typu oprócz pozycji ma wyróżniony kierunek padania s.



W OpenGL (z nieprogramowalną funkcjonalnością) funkcja do obliczania I_L jest postaci:

$$I_L(l) = \begin{cases} I_{L_{max}}(\cos\theta_s)^{s_{exp}}, & \text{gdy } \theta_s \leqslant \theta_u, \\ 0, & \text{gdy } \theta_s > \theta_u, \end{cases}$$

gdzie θ_s jest to kąt pomiędzy wektorami s i $-l$, θ_u jest to kąt odcięcia, który nie może wynosić więcej niż 90°, a s_{exp} jest to wykładnik tłumienia.

W przypadku gdy wektory s i $-l$ są unormowane wartość $\cos\theta_s$ możemy policzyć w następujący sposób:

$$\cos\theta_s = s \cdot -l,$$

gdzie $\cdot$ oznacza iloczyn skalarny.

W DirectX stosowana jest funkcja:

$$I_L(l) = \begin{cases} I_{L_{max}}, & \text{gdy } \cos\theta_s \geqslant \cos\theta_p, \\ I_{L_{max}} \left(\frac{\cos\theta_s - \cos\theta_u}{\cos\theta_p - \cos\theta_u} \right)^{s_{exp}}, & \text{gdy } \cos\theta_u < \cos\theta_s < \cos\theta_p, \\ 0, & \text{gdy } \cos\theta_s \leqslant \cos\theta_u, \end{cases}$$

gdzie θ_p jest to kąt przy którym intensywność światła zaczyna maleć.

Funkcja ta pozwala na to, aby kąty θ_u i θ_p miały wartości aż do 180° pod warunkiem, że $\theta_p \leqslant \theta_u$.

Funkcja obliczająca I_L może również być innej postaci, np. zmieniać się w czasie tworząc efekt migoczącej pochodni.

BRDF

W radiometrii funkcja opisująca zdolność odbijania światła przez obiekt zbudowany z określonego materiału nazywana jest dwukierunkową funkcją rozkładu odbicia (ang. bidirectional reflectance distribution function – BRDF).

Wykorzystywana jest ona w symulacji syntetycznego oświetlenia i jest jedną z ważniejszych funkcji wpływających na realizm tworzonych scen wirtualnego świata.

Służy ona do opisu rozkładu światła odbitego od powierzchni.

Precyzyjna definicja jest następująca:

$$f(l, v) = \frac{dL_o(v)}{dE(l)},$$

gdzie l – kierunek (przychodzący) światła, v – kierunek (wychodzący) obserwatora, L_o – wychodząca radiancja, E – irradiancja.

Funkcja BRDF przyjmuje wartości z przedziału $[0, 1]$. Ponadto wartości te zależą również od długości fali, więc w renderingu funkcja BRDF jest reprezentowana jako wektor wartości RGB.

Funkcja BRDF posiada dwie ważne własności:

- wzajemność Helmholtza, tzn. jeśli zamienione zostaną kierunki odbicia i padania światła, to wartość funkcji BRDF pozostanie bez zmian,
- normalizacja, tzn. całkowita energia odbita od powierzchni nie może być większa od energii, która do powierzchni dotarła.

Funkcja BRDF może zostać skonstruowana dla rzeczywistych materiałów na podstawie pomiarów lub wyprowadzona dla modeli oświetlenia. Każdy z modeli oświetlenia wpływa na kształt funkcji BRDF.

W przypadku gdy źródło światła nie jest powierzchniowe (np. światło punktowe, kierunkowe) wzór na funkcję BRDF przyjmuje postać:

$$f(l, v) = \frac{L_o(v)}{E_L \overline{\cos}\theta_i},$$

gdzie E_L – irradiancja źródła światła zmierzona w płaszczyźnie prostopadłej do kierunku światła l, $L_o(v)$ – wynikowa wychodząca radiancja w kierunku obserwatora v, θ_i – kąt pomiędzy wektorem l oraz normalem powierzchni n oraz

$$\overline{\cos}\theta = \max(\cos\theta, 0).$$

Korzystając z funkcji BRDF możemy otrzymać równanie cieniowania (ang. shading equation):

$$L_o(v) = f(l, v) E_L \overline{\cos}\theta_i,$$

gdzie mnożenie $f(l, v)E_L$ rozumiane jest jako mnożenie po współrzędnych.

W przypadku N źródeł światła równanie przyjmuje postać:

$$L_o(v) = \sum_{k=1}^{N} f(l_k, v) E_{L_k} \overline{\cos}\theta_{i_k}.$$

Modele oświetlenia

W renderingu mamy wiele różnych modeli oświetlenia. Przykładowe to:

- model Lamberta,
- model Phonga,
- model Blinna-Phonga,
- model Warda,
- model Orena-Nayara,
- model Cooka-Torrance'a.

W OpenGL z nieprogramowalną funkcjonalnością używany był model Blinna-Phonga i nie było możliwości skorzystania z innego modelu. Dopiero wprowadzenie shaderów dało taką możliwość.

Model Lamberta (1971 r.)

Jest to najprostszy z modeli. Jego funkcja BRDF wygląda następująco:

$$f(l, v) = \frac{c_{diff}}{\pi}.$$

Tym samym otrzymujemy równanie cieniowania:

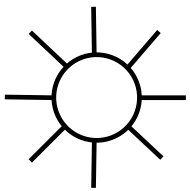
$$L_o(v) = \frac{c_{diff}}{\pi} E_L \overline{\cos}\theta_i.$$

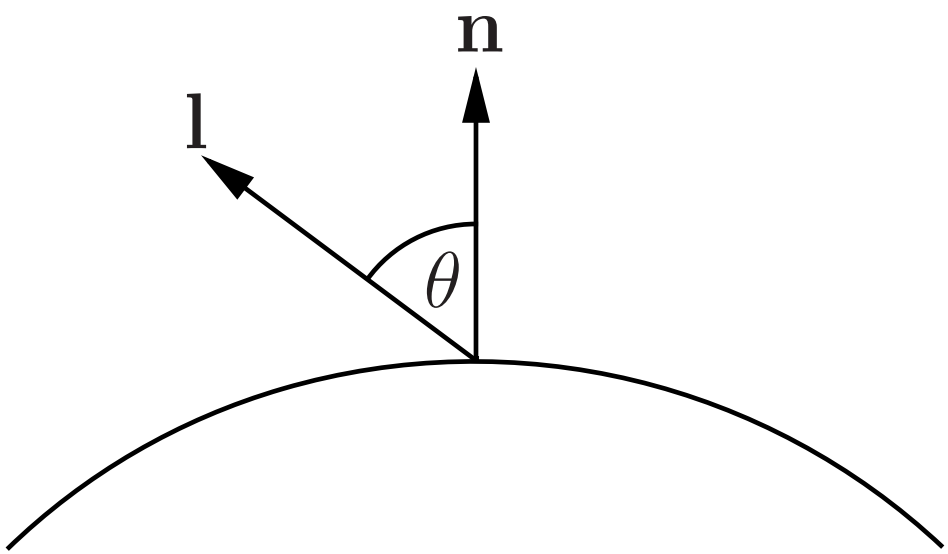
W równaniu używanym zazwyczaj w renderingu czasu rzeczywistego brakuje wyrażenia $1/\pi$. Spowodowane jest to tym, że wyrażenie to jest łączone z E_L, tzn. E_L/π, a nie E_L jest właściwością źródła światła.

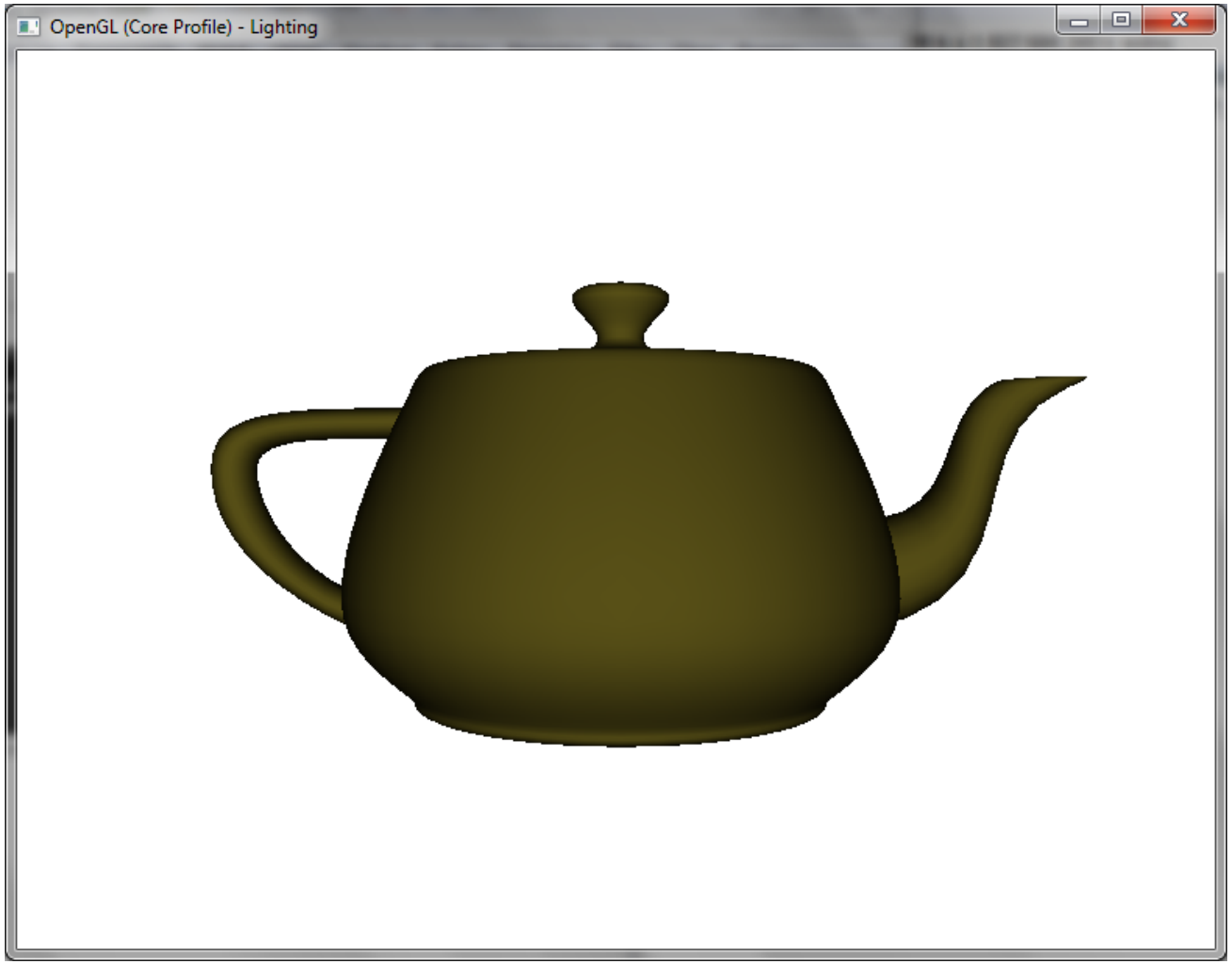
W literaturze na temat shaderów najczęściej podawane jest równanie:

$$c = c_d k_d \cos\theta = c_d k_d (n \cdot l),$$

gdzie c_d – kolor światła, k_d – stopień rozproszenia światła (kolor materiału).






OpenGL (Core Profile) - Lighting

Model Phonga (1975 r.)

W modelu tym dodane są odbicia. Równanie cieniowania przedstawione przez Phonga wygląda następująco:

$$L_o(v) = \begin{cases} B_L(\overline{cos}\theta_i c_{diff} + (\overline{\cos}\alpha_r)^m c_{spec}), & \text{gdy } \theta_i > 0, \\ 0, & \text{gdy } \theta_i \leqslant 0, \end{cases}$$

gdzie $B_L = E_L/\pi$, a α_r jest to kąt pomiędzy odbitym wektorem światła r a wektorem kierunku obserwatora v.

Funkcja BRDF tego równania ma postać:

$$f(l, v) = \begin{cases} \frac{c_{diff}}{\pi} + \frac{c_{spec}(\overline{\cos}\alpha_r)^m}{\pi \overline{\cos}\theta_i}, & \text{gdy } \theta_i > 0, \\ 0, & \text{gdy } \theta_i \leqslant 0. \end{cases}$$

W celu osiągnięcia lepszego fizycznego efektu usunięto $\overline{\cos}\theta_i$ z funkcji BRDF otrzymując:

$$f(l, v) = \frac{c_{diff}}{\pi} + \frac{c_{spec}(\overline{\cos}\alpha_r)^m}{\pi},$$

Mając taką formę funkcji BRDF wprowadzono tzw. unormowaną funkcję BRDF dla modelu Phonga:

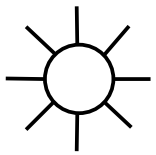
$$f(l, v) = \frac{c_{diff}}{\pi} + \frac{m+2}{2\pi} c_{spec}(\overline{\cos}\alpha_r)^m.$$

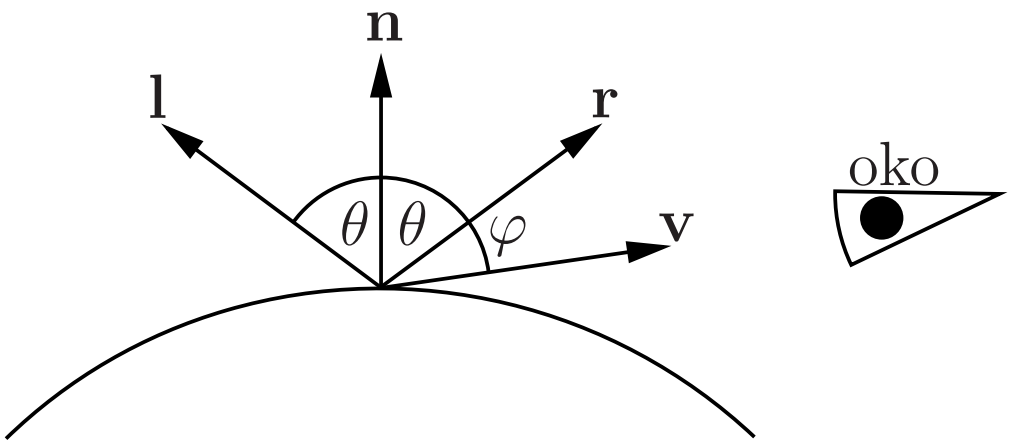
W literaturze na temat shaderów najczęściej podawane jest równanie:

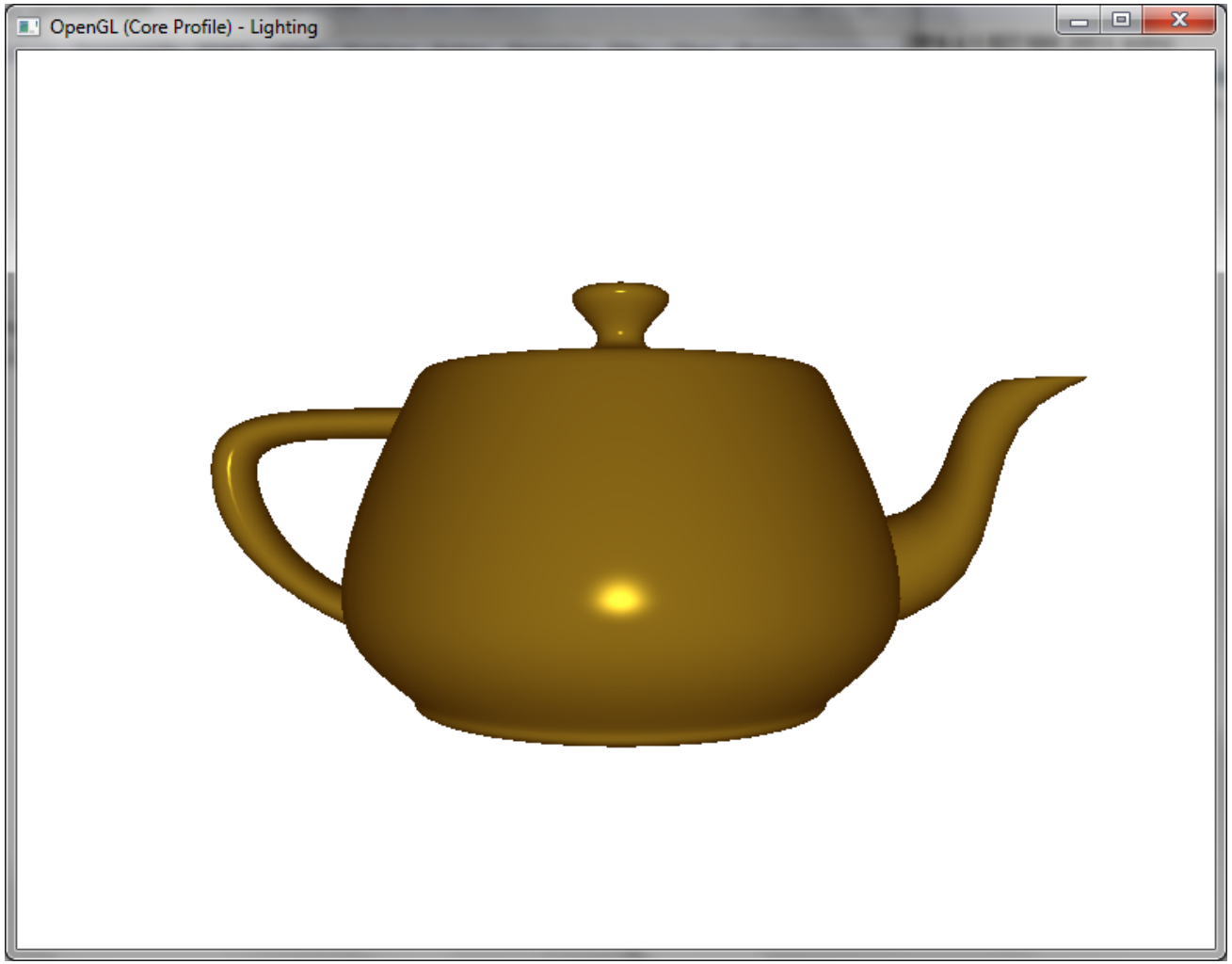
$$c = c_d k_d \cos\theta + c_s k_s (\cos\varphi)^m = c_d k_d (n \cdot l) + c_s k_s (r \cdot v)^m,$$

gdzie c_s – kolor światła odbitego, k_s – stopień odbicia światła odbitego (kolor materiału).
Do obliczenia wektora odbicia r stosuje się wzór:

$$r = 2(n \cdot l)n - l.$$






OpenGL (Core Profile) - Lighting

Model Blinna-Phonga (1977 r.)

Jest to modyfikacja modelu Phonga. Używany w nim jest wektor połówkowy h reprezentujący normala mikrościanki, która prowadzi do odbicia lustrzanego światła w kierunku oka.

Wektor połówkowy obliczany jest następująco:

$$h = \frac{l + v}{\|l + v\|}.$$

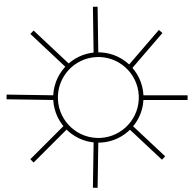
Znormalizowana funkcja BRDF dla tego modelu ma postać:

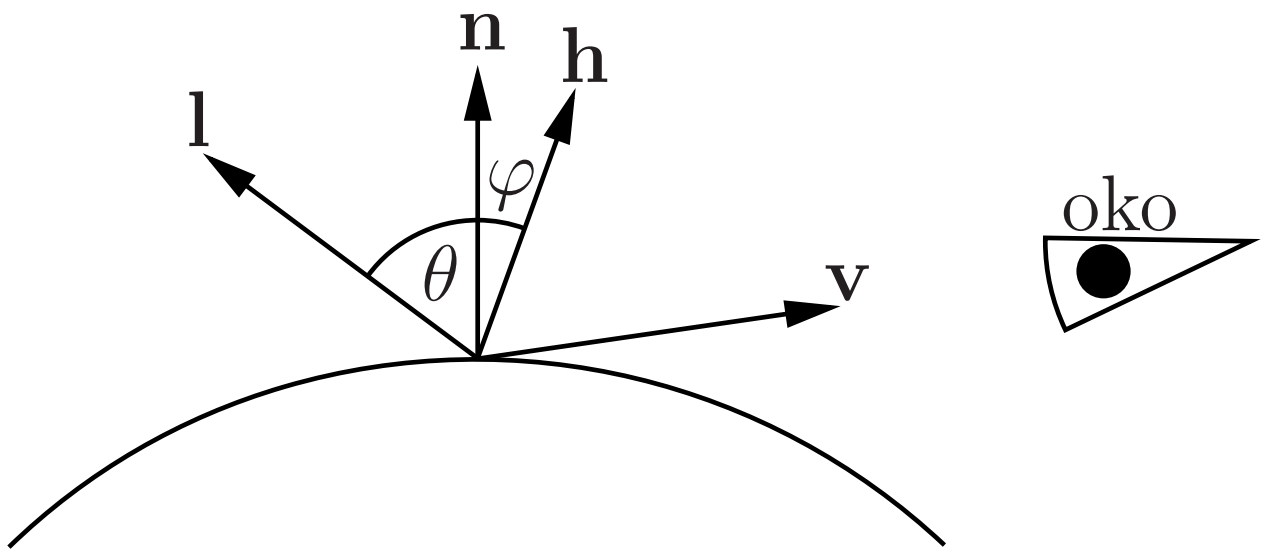
$$f(l, v) = \frac{c_{diff}}{\pi} + \frac{m + 8}{8\pi} c_{spec} (\overline{\cos}\theta_h)^m,$$

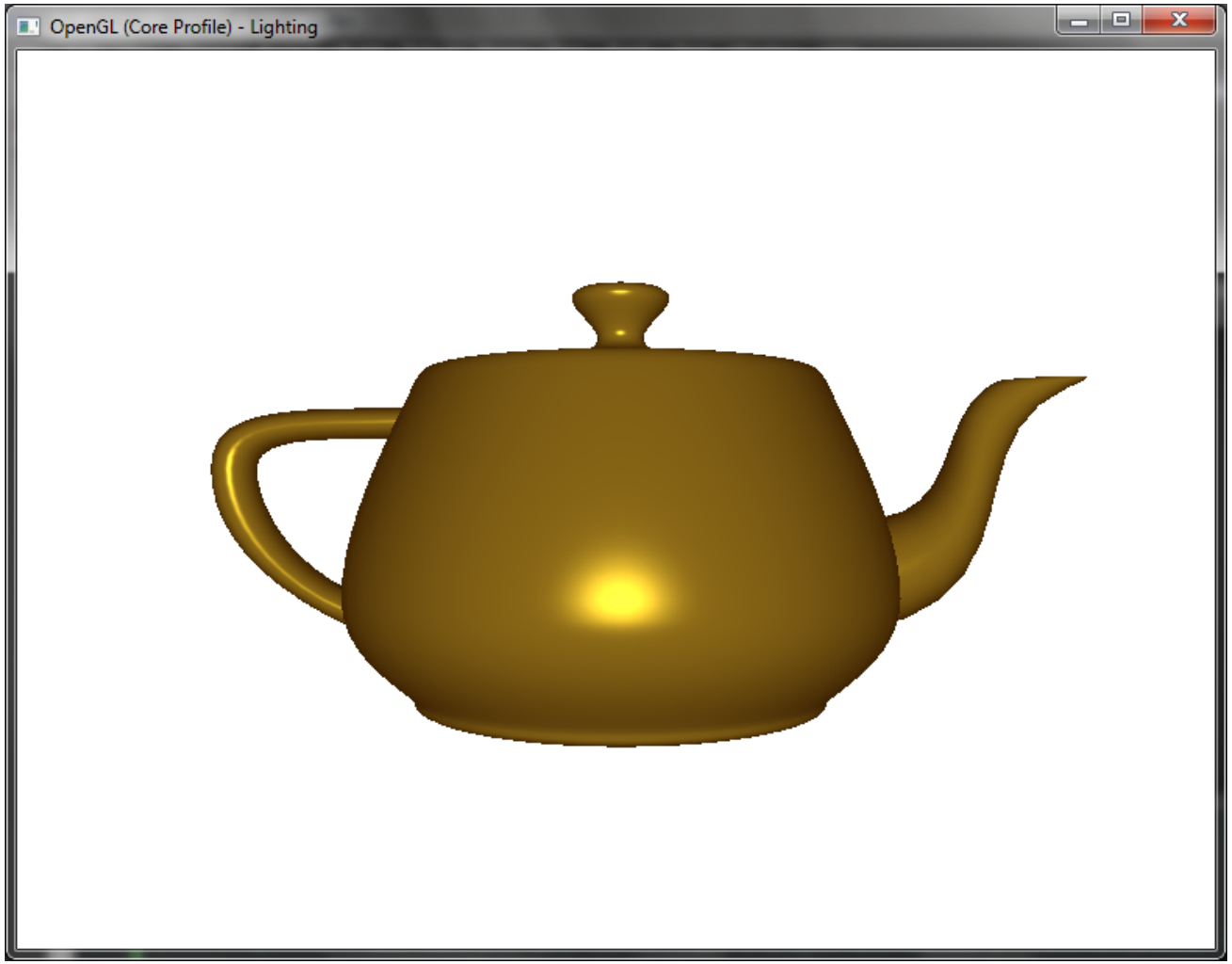
gdzie θ_h jest to kąt pomiędzy wektorem połówkowym a normalem.

W literaturze na temat shaderów najczęściej podawane jest równanie:

$$c = c_d k_d \cos\theta + c_s k_s (\cos\varphi)^m = c_d k_d (n \cdot l) + c_s k_s (n \cdot h)^m.$$






OpenGL (Core Profile) - Lighting

Światło otoczeniowe (ang. ambient)

Światło otoczeniowe jest najprostszym modelem oświetlenia niebezpośredniego, w którym niebezpośrednia radiancja nie zmienia się wraz ze zmianą kierunku i ma stałą wartość L_A.

Najczęściej światło otoczeniowe dodawane jest do pozostałych modeli. Wówczas równanie cieniowania przyjmuje postać:

$$L_o(v) = c_{amb} L_A + \sum_{k=1}^{N} f(l_k, v) E_{L_k} \overline{\cos}\theta_{i_k}.$$

Jest to globalne światło otoczeniowe. Do każdego z N świateł w tym równaniu możemy dodać lokalne (dla danego światła) wyrażenie symulujące światło otoczeniowe.


OpenGL (Core Profile) - Lighting

W literaturze na temat shaderów model Phonga i Blinna-Phonga z dodanym światłem otoczeniowym nosi również nazwę modelu ADS (Ambient-Diffuse-Specular).

Wektory normalne

We wszystkich modelach oświetlenia, które widzieliśmy używany był wektor normalny (prostopadły). W jaki sposób go obliczyć dla danego modelu?

Wektory normalne

We wszystkich modelach oświetlenia, które widzieliśmy używany był wektor normalny (prostopadły). W jaki sposób go obliczyć dla danego modelu?

Powierzchnie parametryczne

Załóżmy, że mamy powierzchnię parametryczną $S : [0,1]^2 \to \mathbb{R}^3$ daną wzorem:

$$\forall_{u,v\in[0,1]} \quad S(u,v) = [x(u,v), y(u,v), z(u,v)]^T$$

Wówczas wektor normalny n do powierzchni S w punkcie (u_0, v_0) dany jest wzorem:

$$\mathrm{n} = \frac{\partial S}{\partial u}(u_0, v_0) \times \frac{\partial S}{\partial v}(u_0, v_0),$$

gdzie $\times$ oznacza iloczyn wektorowy.

Wielokąty

Niech $P_1 = (x_1, y_1, z_1)^T$, $P_2 = (x_2, y_2, z_2)^T$, $P_3 = (x_3, y_3, z_3)^T$.

Wówczas

$\mathrm{v}_1 = [x_2 - x_1, y_2 - y_1, z_2 - z_1]^T$,

$\mathrm{v}_2 = [x_3 - x_1, y_3 - y_1, z_3 - z_1]^T$

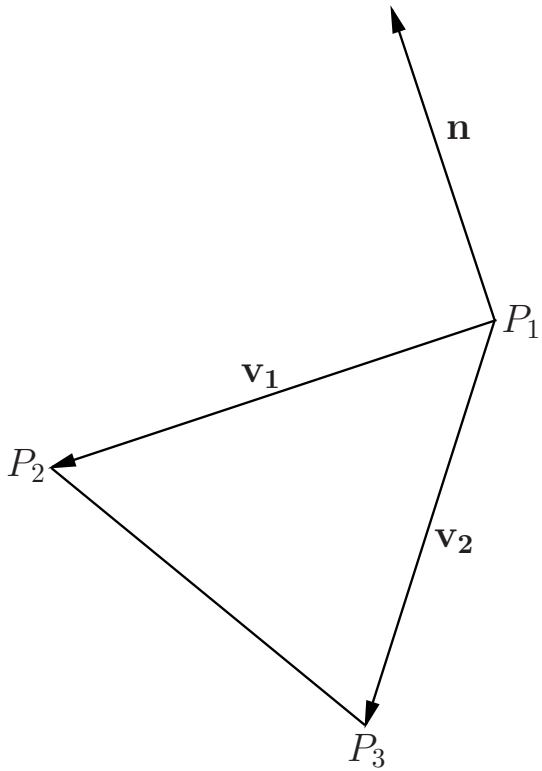
oraz

$\mathrm{n} = \mathrm{v}_1 \times \mathrm{v}_2$,

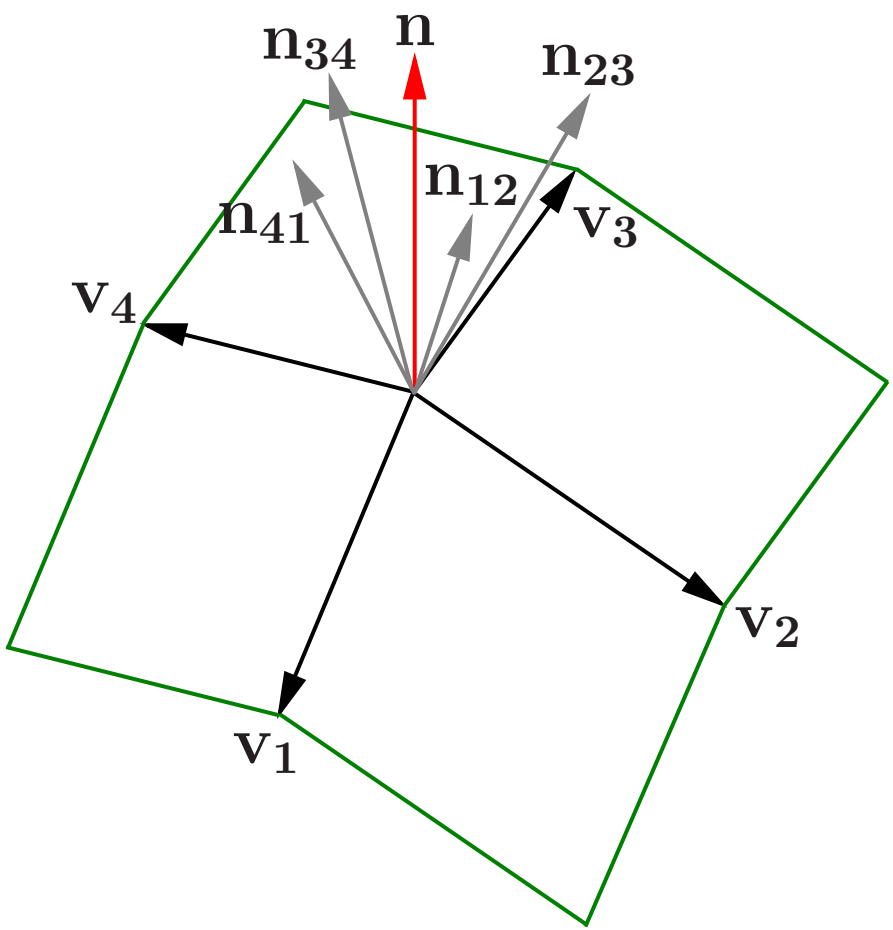
gdzie

$[v_x^1, v_y^1, v_z^1]^T \times [v_x^2, v_y^2, v_z^2]^T =$

$[v_y^1 v_z^2 - v_z^1 v_y^2, v_z^1 v_x^2 - v_x^1 v_z^2, v_x^1 v_y^2 - v_y^1 v_x^2]^T$

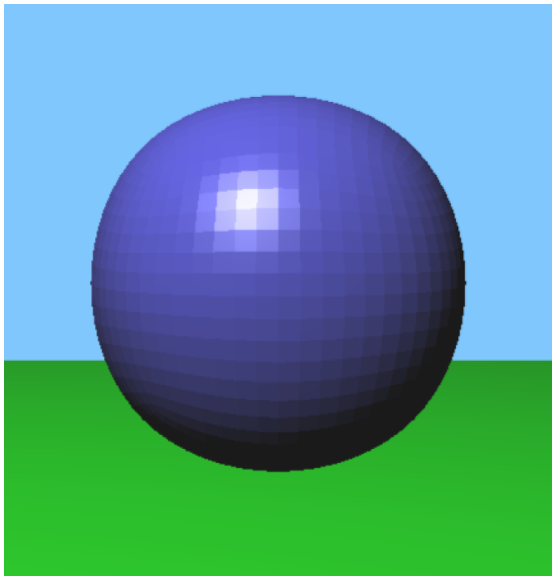


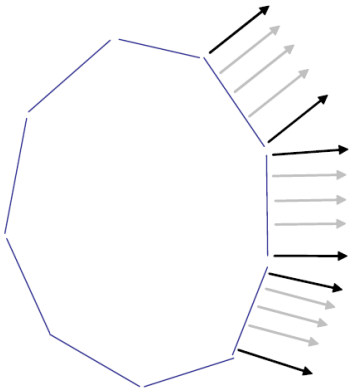
Uśrednianie wektorów normalnych

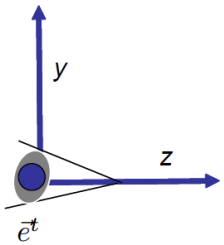


$$\mathsf{n} = \frac{\mathsf{n}_{12} + \mathsf{n}_{23} + \mathsf{n}_{34} + \mathsf{n}_{41}}{\left\|\mathsf{n}_{12} + \mathsf{n}_{23} + \mathsf{n}_{34} + \mathsf{n}_{41}\right\|}$$

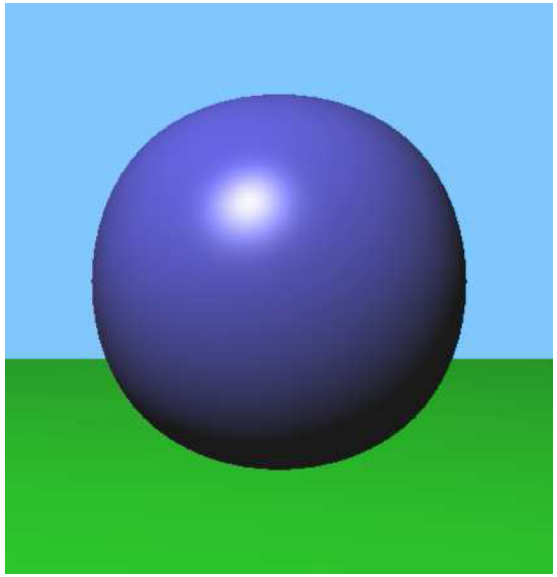
„per face”

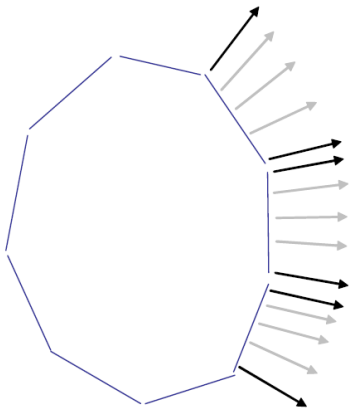


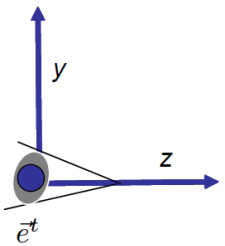




„per vertex” (uśrednianie normali)



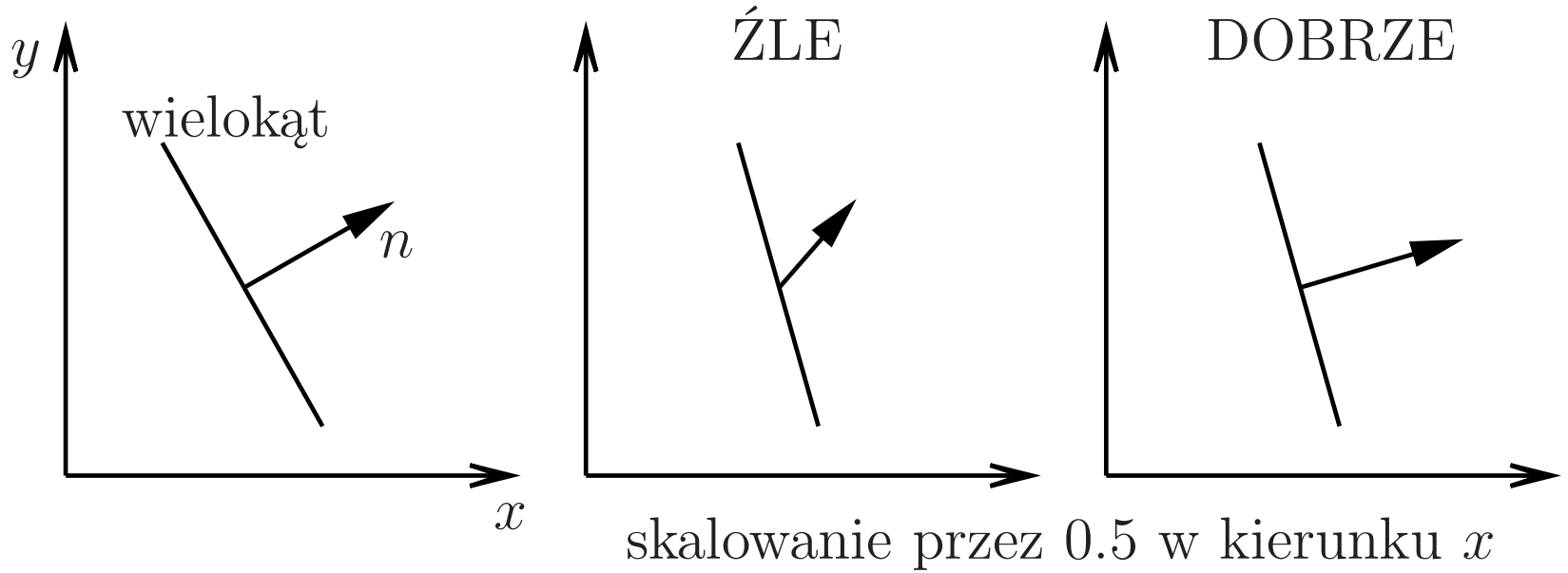




Transformacja normala

Podobnie jak wierzchołki również wektory normalne muszą zostać przekształcone do odpowiedniego układu współrzędnych.

Do tego celu nie możemy użyć tej samej macierzy transformacji co dla wierzchołków.



Załóżmy, że M jest macierzą transformacji wierzchołków. Wówczas macierz transformacji normala R ma postać:

$$R = (M^{-1})^T.$$

Normal jest wektorem więc translacja nie ma na niego wpływu. Ponadto, większość transformacji jest afiniczna, więc nie zmienia składowej w. W związku z tym do obliczeń macierzy R wystarczy wziąć podmacierz 3×3 macierzy M.

Jeśli wiemy, że będziemy wykonywać jedynie translacje i rotacje, to wówczas macierz transformacji normala jest taka sama jak macierz M. Jest to konsekwencją tego, że translacja nie wpływa na wektory, a macierzą odwrotną do macierzy rotacji jest jej macierz transponowana.

Cieniowanie geometrii

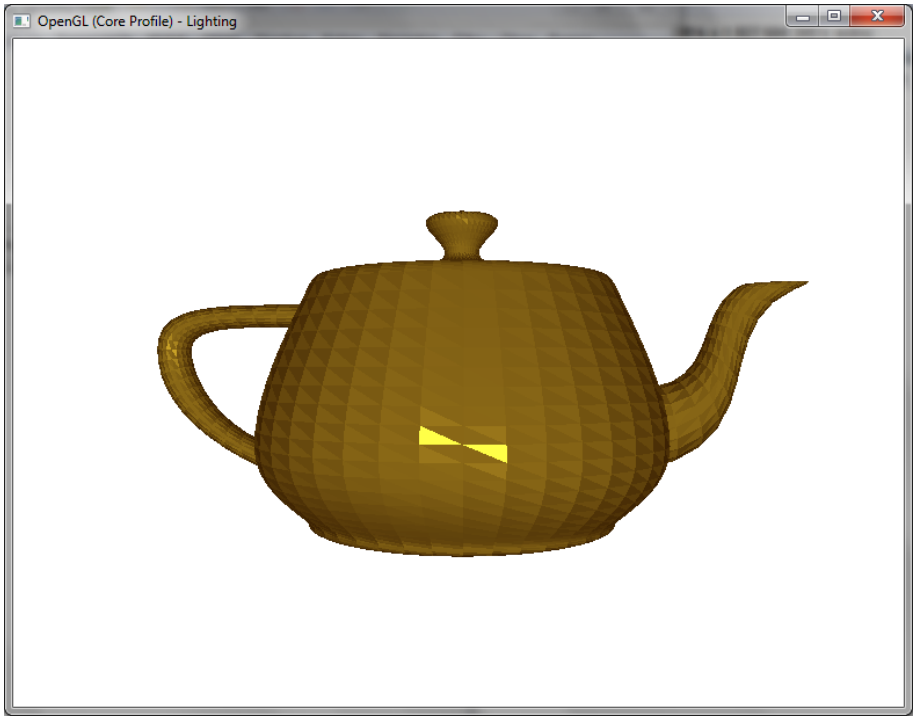
Wyróżniamy trzy rodzaje cieniowania:

- płaskie (ang. flat) lub dla ścianki (ang. per face),
- Gouraud lub dla wierzchołka (ang. per vertex),
- Phonga lub dla fragmentu (ang. per fragment).

W OpenGL bez programowalnej funkcjonalności dostępne było jedynie cieniowanie płaskie i Gouraud.

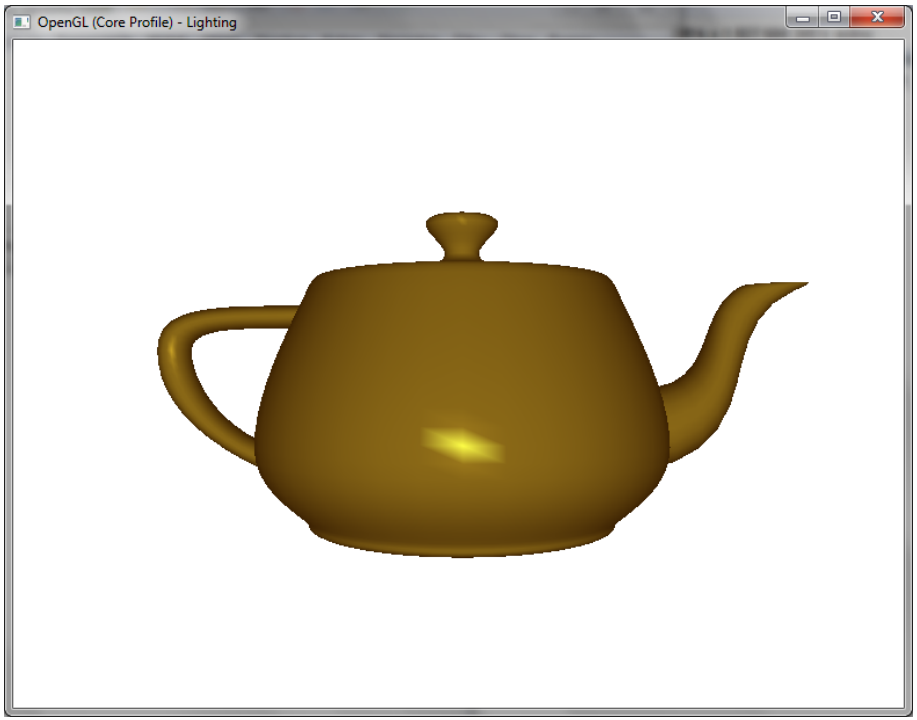
Cieniowanie płaskie

- Obliczenia koloru wykonywane jednokrotnie dla ścianki.
- Zaletą jest szybkość.
- Wadą jest to, że widzimy ścianki.



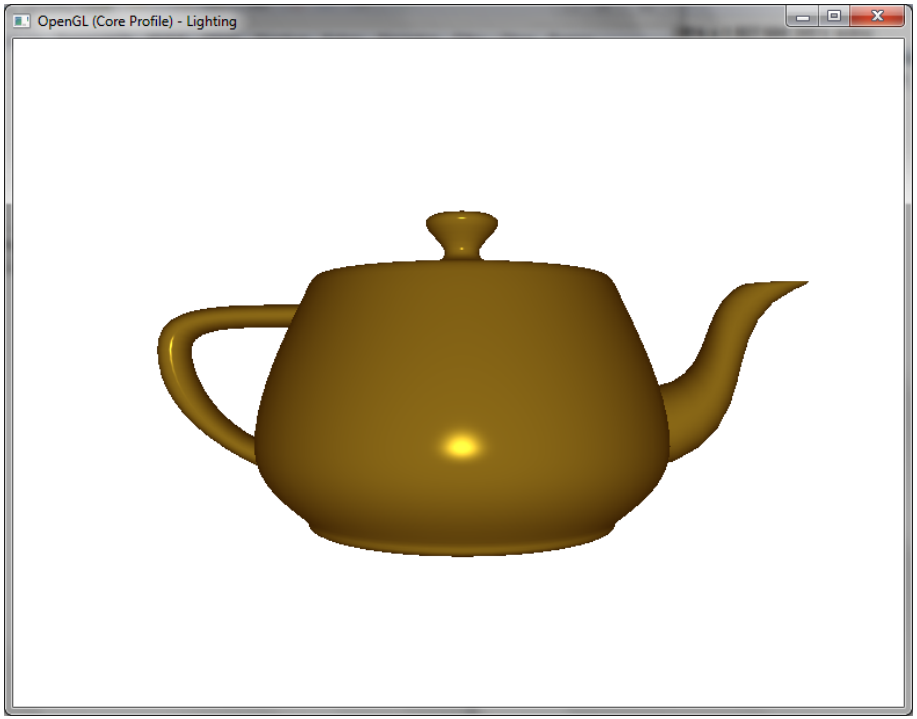
Cieniowanie Gouraud

- Obliczenia koloru wykonywane są dla każdego wierzchołka reszta punktów ścianki jest interpolowana.
- Zaletami są szybkość i bardziej gładki wygląd powierzchni.
- Wadą są problemy z małymi rozjaśnieniami.



Cieniowanie Phonga

- Rasteryzer interpoluje normale a nie kolory wierzchołków. Obliczenia koloru odbywają się dla każdego fragmentu.
- Zaletą jest lepsza jakość w porównaniu do cieniowania Gouraud.
- Wadą jest to, że jest wolniejsze.



Gouraud (1024, 6320, 40000 trójkątów) vs Phong (6320 trójkątów)

Elementy GLSL ES

Przy pisaniu shaderów związanych z oświetleniem (i nie tylko) pomocne mogą być różne funkcje języka GLSL ES.

Wprowadźmy oznaczenia:

- `genType` oznacza jeden z typów: `float`, `vec2`, `vec3`, `vec4`,
- `mat` oznacza typ macierzowy: `mat2`, `mat3`, `mat4`.

Dla danej funkcji każdy argument oznaczony jednym z ogólnych typów musi mieć taką samą liczbę składowych.

Funkcje trygonometryczne:

```
genType sin(genType angle);
genType cos(genType angle);
genType tan(genType angle);
```

`angle` jest to kąt w radianach

```
genType asin(genType x);
genType acos(genType x);
```

Funkcje zwracają arc sin (liczba z przedziału $[-\pi/2, \pi/2]$) i arc cos (liczba z przedziału $[0, \pi]$) podanej wartości `x`. W przypadku gdy $|x| > 1$ wynik nie jest zdefiniowany.

```
genType atan(genType y, genType x);
```

Zwraca kąt z przedziału $[-\pi, \pi]$, dla którego tangens wynosi y/x. Znaki x i y używane są do określenia ćwiartki kąta.

```
genType atan(genType y_over_x);
```

Zwraca kąt z przedziału $[-\pi/2, \pi/2]$, dla którego tangens wynosi `y_over_x`.

Funkcje do konwersji stopni na radiany i odwrotnie:

```
genType radians(genType degrees);
genType degrees(genType radians);
```

Potęgowanie, x^y (niezdefiniowane gdy $x < 0 \vee (x = 0 \wedge y \leqslant 0)$):

```
genType pow(genType x, genType y);
```

Funkcja eksponencjalna, e^x:

```
genType exp(genType x);
```

Logarytm naturalny, $\ln x$ (niezdefiniowane gdy $x \leqslant 0$):

```
genType log(genType x);
```

Potęga dwójki, 2^x:

```
genType exp2(genType x);
```

Logarytm o postawie 2, $\log_2 x$ (niezdefiniowane gdy $x \leqslant 0$):

```
genType log2(genType x);
```

Pierwiastek, $\sqrt{x}$ (niezdefiniowane gdy $x < 0$):

```
genType sqrt(genType x);
```

Odwrotność pierwiastka, $1/\sqrt{x}$ (niezdefiniowanie gdy $x \leqslant 0$):

```
genType inversesqrt(genType x);
```

Moduł, $|x|$:

```
genType abs(genType x);
```

Znak:

```
genType sign(genType x);
```

Podłoga, $\lfloor x \rfloor$, sufit, $\lceil x \rceil$:

```
genType floor(genType x);
genType ceil(genType x);
```

Modulo, $x \mod y$:

```
genType mod(genType x, float y);
genType mod(genType x, genType y);
```

Minimum, maximum (pełna lista w specyfikacji GLSL ES):

```
genType min(genType x, genType y);
genType max(genType x, genType y);
```

Przycięcie (niezdefiniowane gdy `minVal` $>$ `maxVal`):

```
genType clamp(genType x, float minVal, float maxVal);
```

Funkcja zwraca `min(max(x, minVal), maxVal)`. Pełna lista wersji funkcji w specyfikacji GLSL ES.

Długość wektora:

```
float length(genType x);
```

Odległość dwóch punktów:

```
float distance(genType p0, genType p1);
```

Iloczyn skalarny, $x \cdot y$:

```
float dot(genType x, genType y);
```

Iloczyn wektorowy, $x \times y$:

```
vec3 cross(vec3 x, vec3 y);
```

Normalizacja wektora:

```
genType normalize(genType x);
```

Wektor odbicia, $I - 2(N \cdot I)N$:

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