Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Oświetlenie

Potok renderowania. Techniki oświetlenia i cieniowania.

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

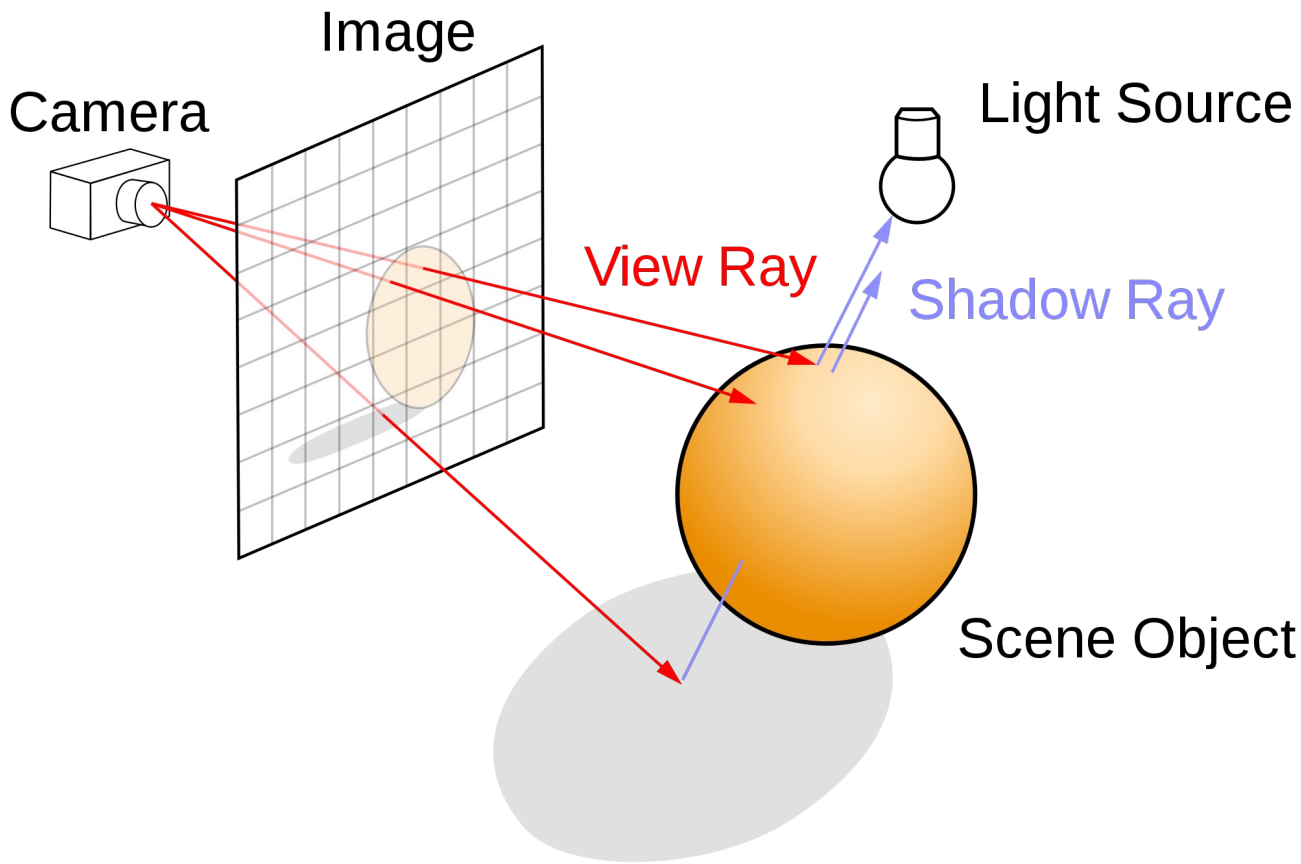
POTOK RENDEROWANIA

Jest to operacja **niezwykle kosztowna**. Istnieją jedynie bardzo ograniczone *ray tracery* działające w czasie rzeczywistym.

Najbardziej kosztowne jest **śledzenie promieni światła** i ich odbić.

Źródło obrazu: Wikipedia

- W **grafice realistycznej** stosuje się zwykle podejścia oparte na **śledzeniu promieni** (ang. *ray tracing*)
  - Dla **każdego piksela** wynikowego obrazu **śledzony jest promień** i sprawdzane jest, w jaki element sceny trafia i czy jest on oświetlony

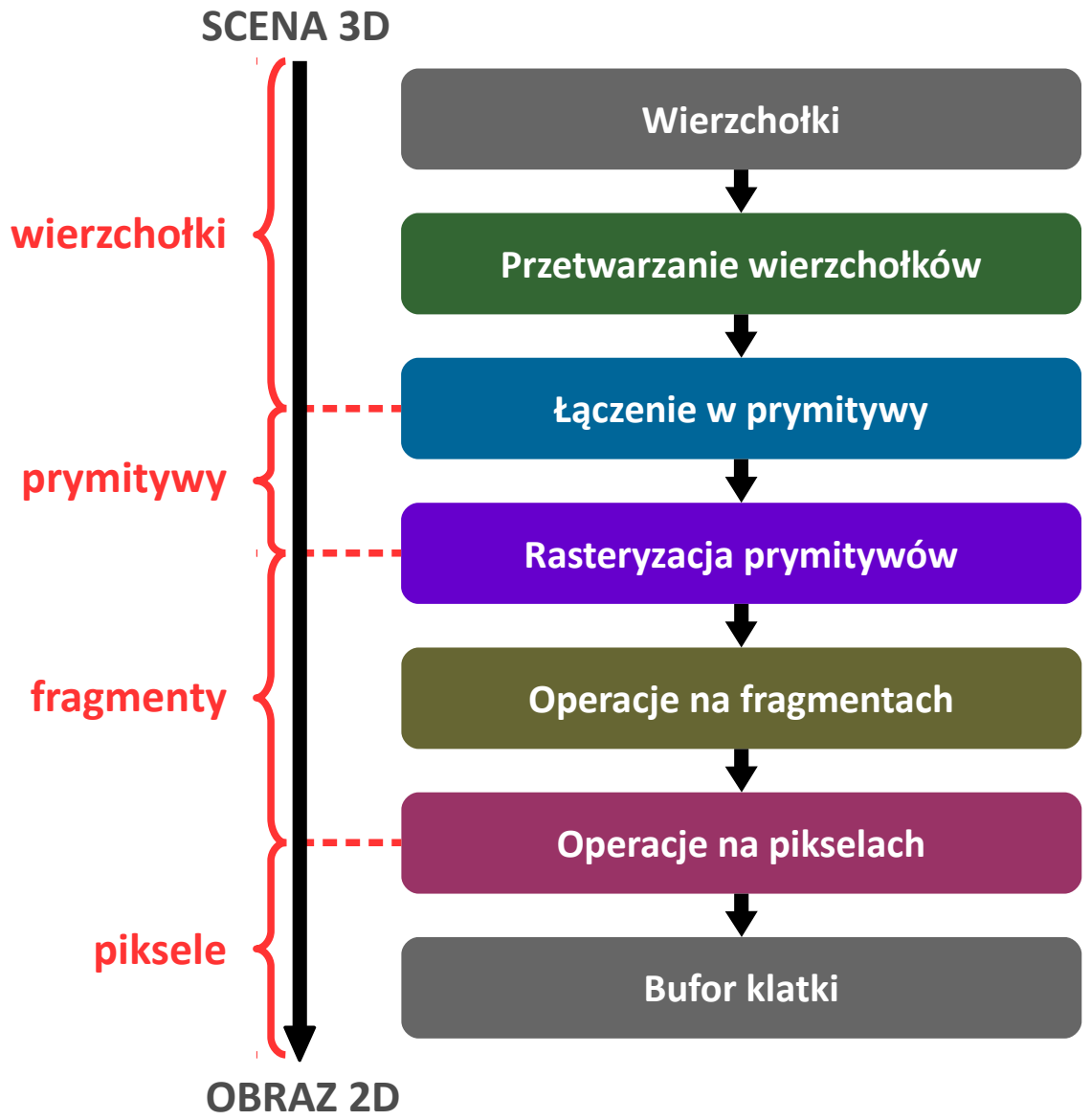


POTOK RENDEROWANIA

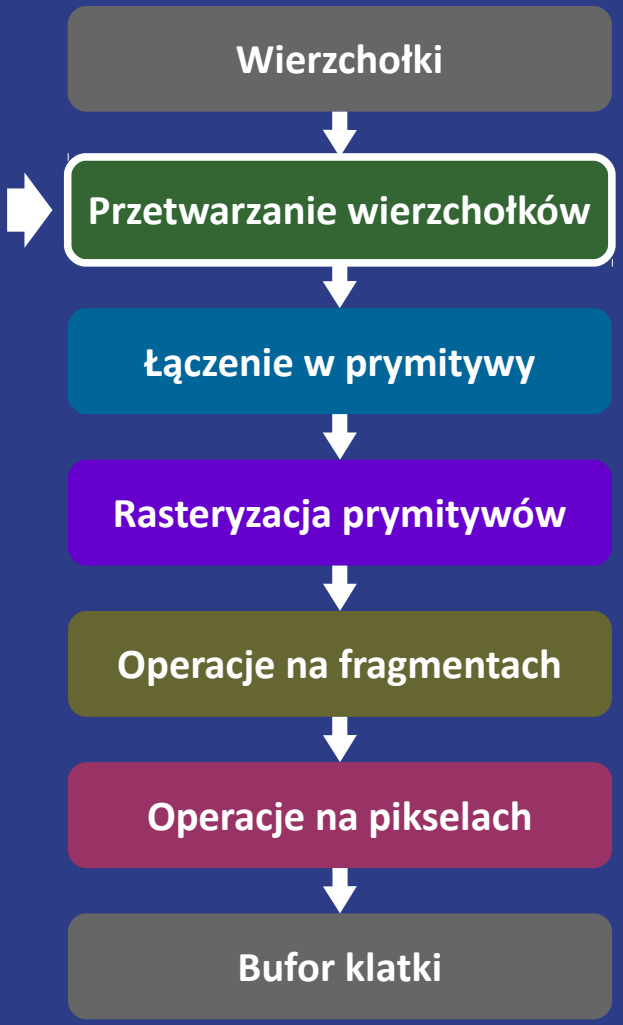
Jest to podejście w pewnym stopniu **odwrotne**:

Zaczynamy od geometrii i to ją **rzutujemy** na ekran.

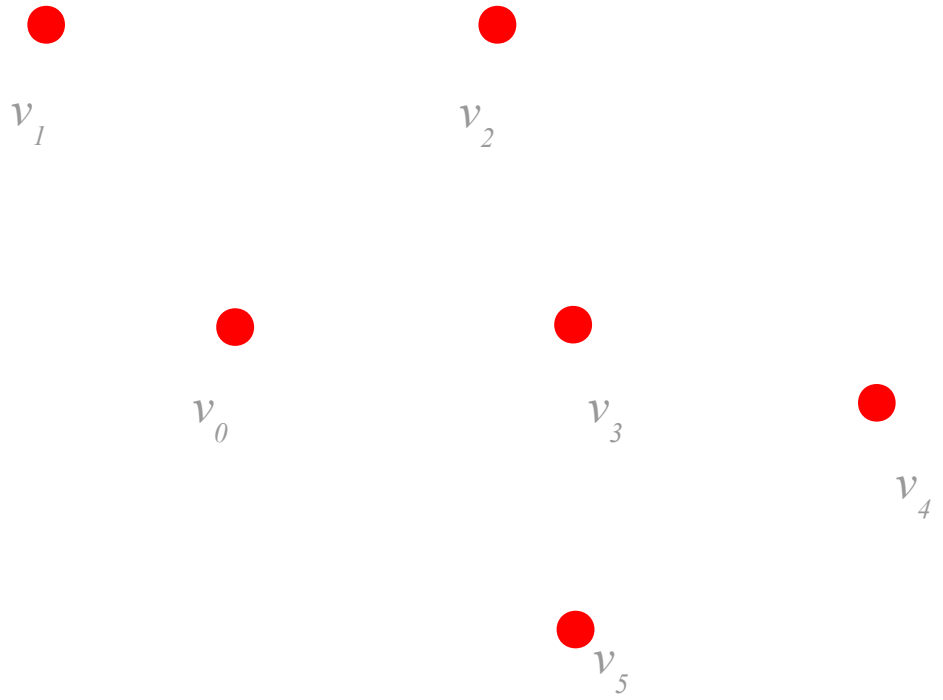
- Potok **renderowania geometrii** stosowany w grafice czasu rzeczywistego



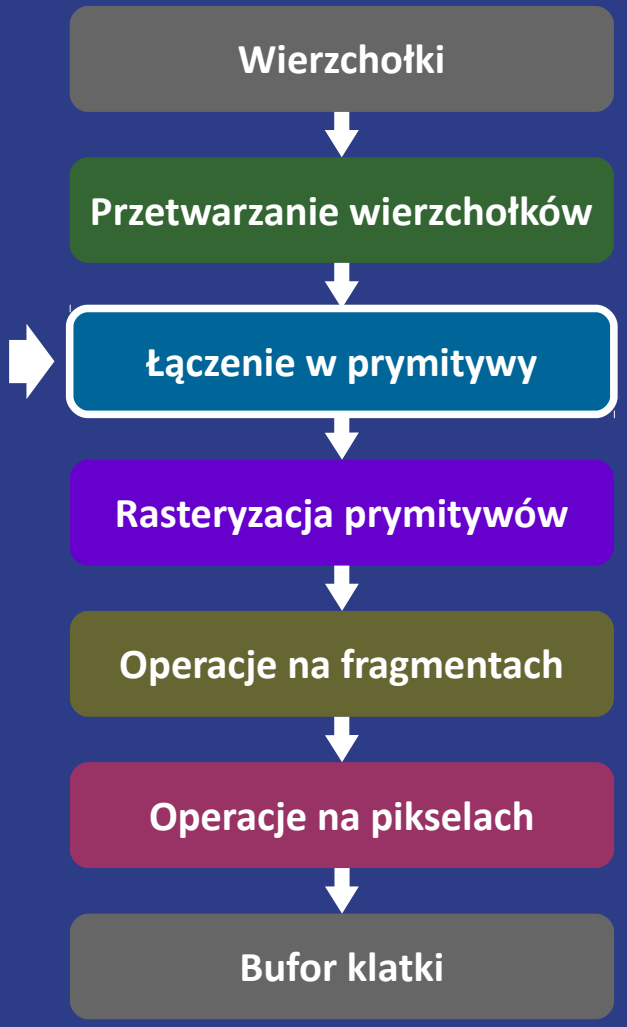
POTOK RENDEROWANIA



- **Wierzchołki** i **operacje na wierzchołkach**
  - Geometria sceny opisana jest za pomocą wierzchołków
  - Atrybuty (np. położenie) są poddawane **przekształceniom**
    - **Transformacje** geometryczne
    - **Vertex shaders**: np. **Morphing**, **symulacja ruchu** itp.

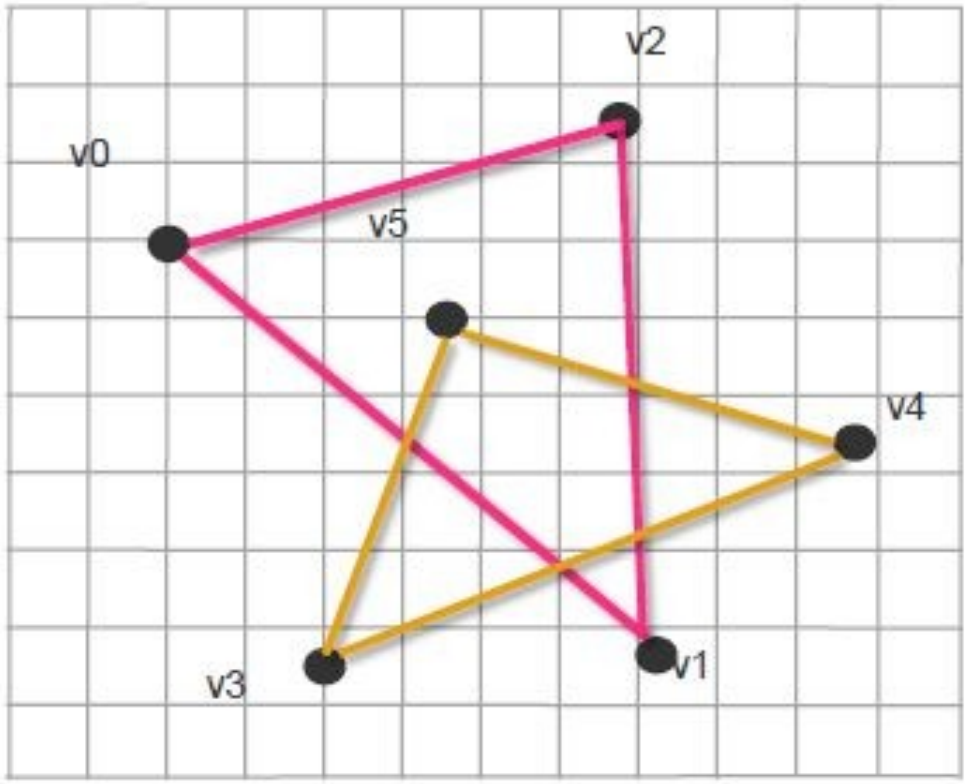


POTOK RENDEROWANIA

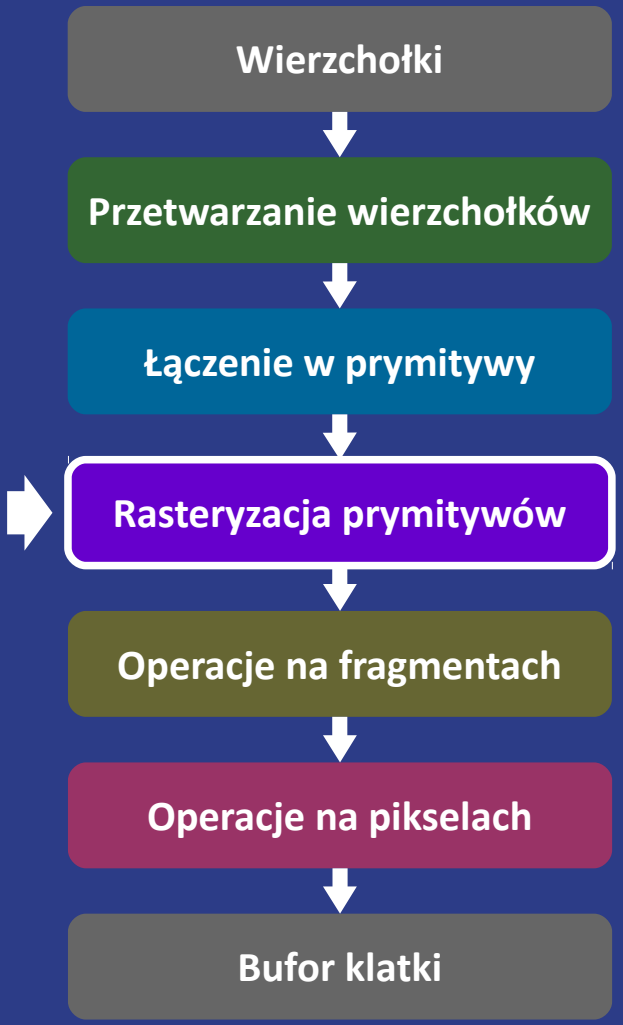


Źródło obrazu:
GPU Design

- Łączenie wierzchołków w **prymitywy**
  - Grupy wierzchołków **łączone są w prymitywy**, w sposób określony przez twórcę
    - Np. **trójkąty**, **czworokąty**, **linie** itp.

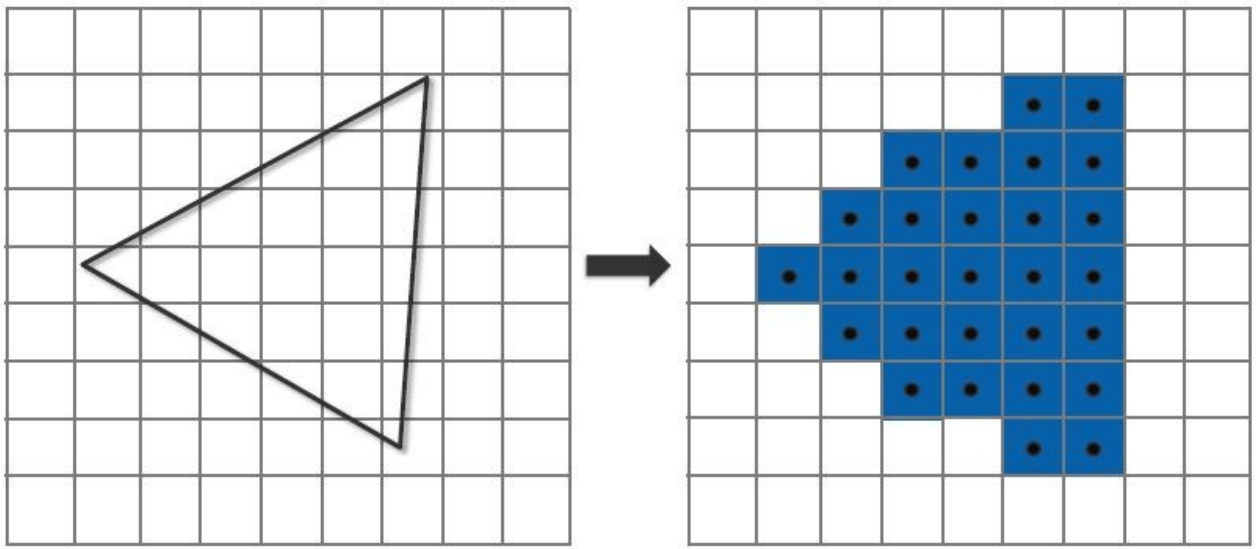


POTOK RENDEROWANIA

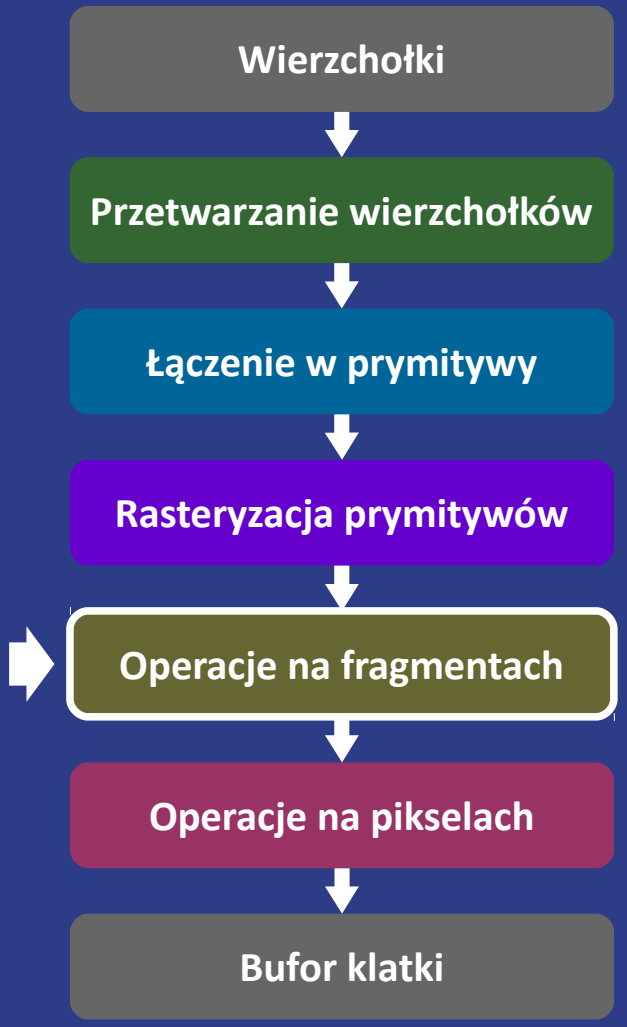


Źródło obrazu:
GPU Design

- **Rasteryzacja prymitywów**
  - Etap na którym **generowane są fragmenty**
  - **Próbkowanie** prymitywów w **przestrzeni ekranu**
  - **Fragment** to **próbka powierzchni prymitywu**, odpowiadająca **jednemu pikselowi z bufora klatki**
    - Czyli wycinek powierzchni obiektu na scenie, który przyczyni się do obliczenia wyjściowej wartości piksela w buforze klatki
  - Z fragmentem związane są interpolowane wartości atrybutów wierzchołków

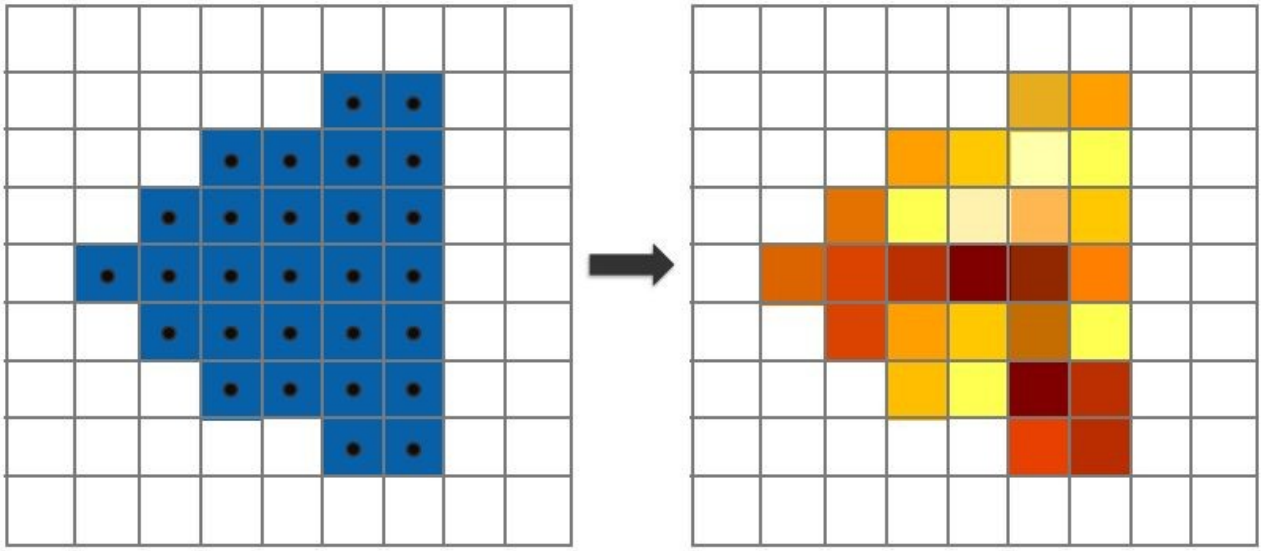


POTOK RENDEROWANIA

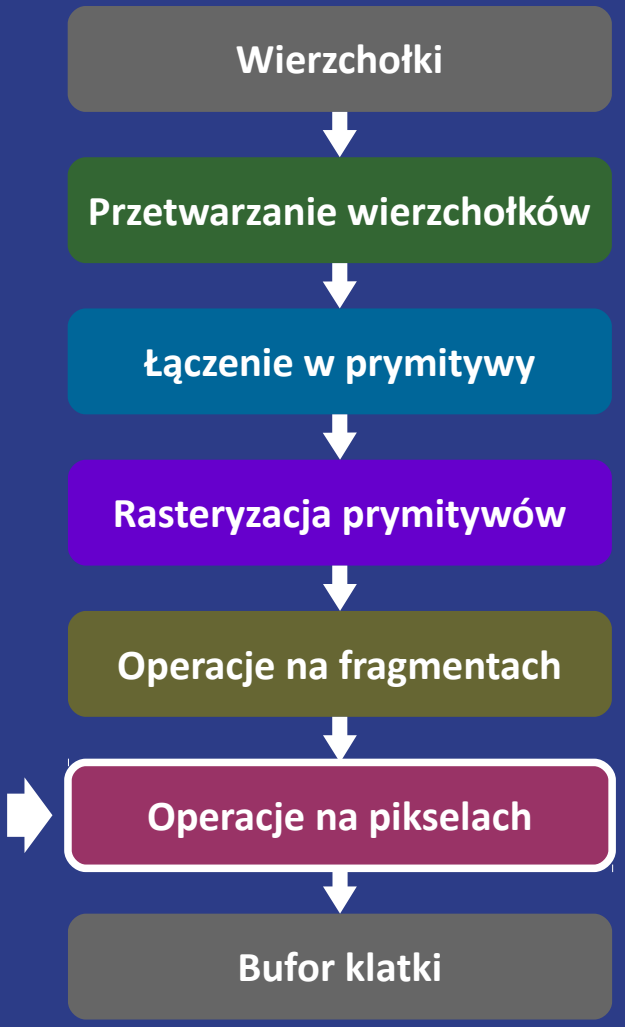


Źródło obrazu:
GPU Design

- **Operacje na fragmentach**
  - Wyznaczenie **wartości fragmentu**, która zostanie wzięta pod uwagę podczas wypełniania bufora klatki
  - Oblicza się go **na podstawie** np.:
    - **Koloru wierzchołków**
    - **Oświetlenia**
    - **Tekstury**

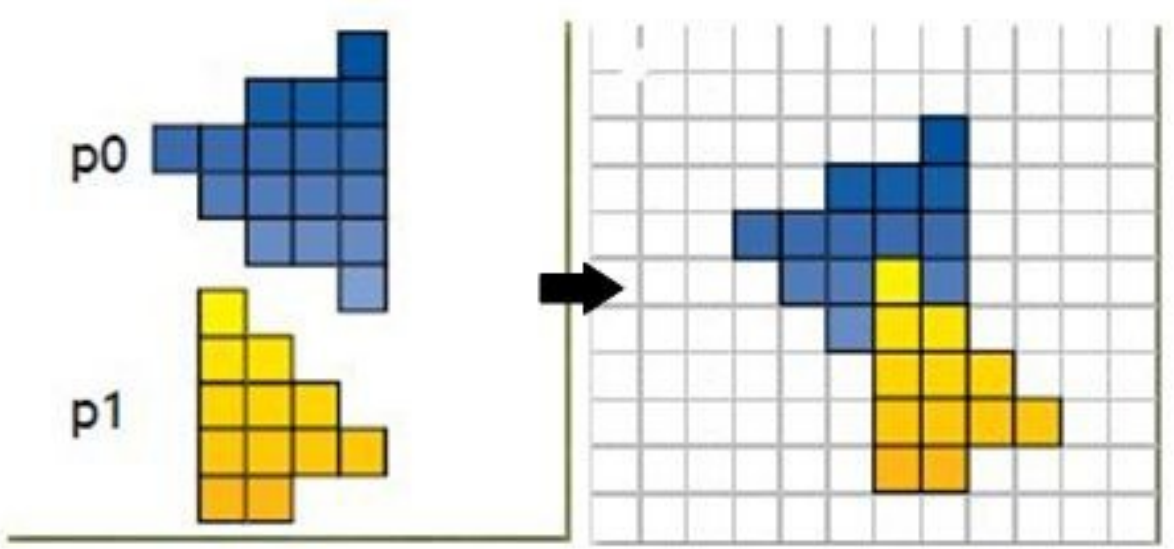


POTOK RENDEROWANIA



Źródło obrazu: GPU Design

- **Operacje na pikselach**
  - **Wypełnienie bufora klatki** na podstawie wyliczonych wartości fragmentów dla wszystkich prymitywów
  - Operacje takie jak np.:
    - **Test głębokości** (algorytm *bufora Z*)
    - **Alpha blending** (przezroczystość)
    - **Anti-aliasing**

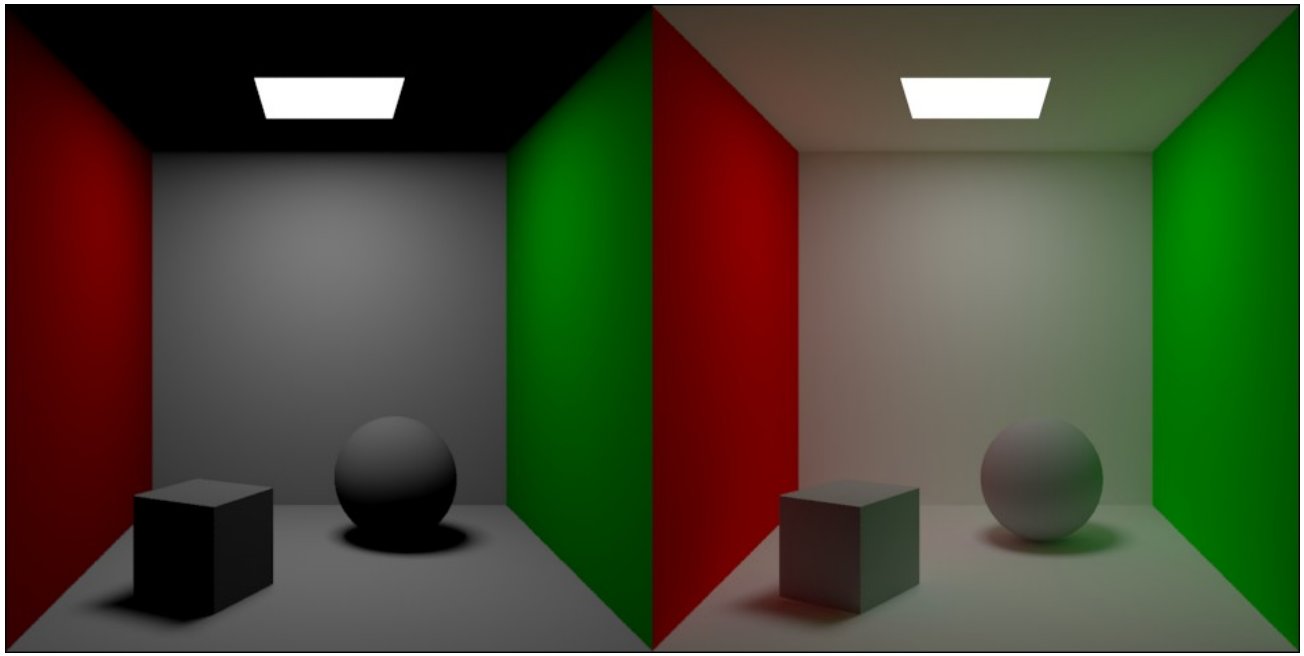


Oświetlenie *lokalne* a *globalne*

W **grafice czasu rzeczywistego** najczęściej spotyka się **oświetlenie lokalne** lub bardzo uproszczone formy oświetlenia globalnego.

Oświetlenie globalne to technika używana w **grafice realistycznej**.

- Techniki oświetlenia można podzielić na dwa rodzaje:
  - Oświetlenie **lokalne**
    - Na kolor poszczególnych punktów wpływa **wyłącznie bezpośrednie oświetlenie** ze źródeł światła (*direct illumination*)
  - Oświetlenie **globalne**
    - Na kolor wpływa bezpośrednie oświetlenie ze źródeł światła **oraz światło odbite przez inne obiekty** (*indirect illumination*)

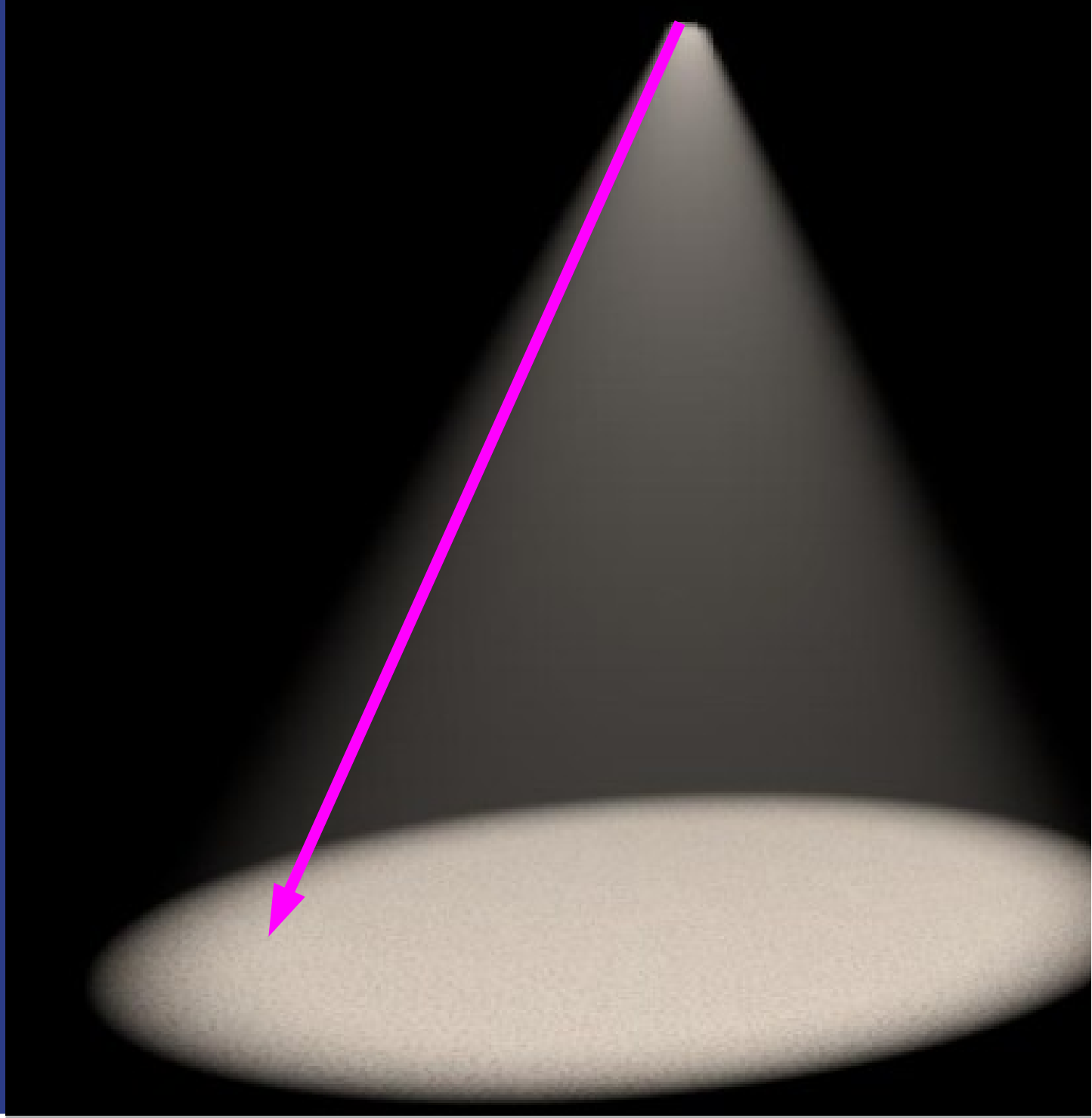


Oświetlenie lokalne Oświetlenie globalne

Oświetlenie *lokalne* a *globalne*

0 odbić światła

Tylko oświetlenie bezpośrednie
(ang. *direct illumination*)

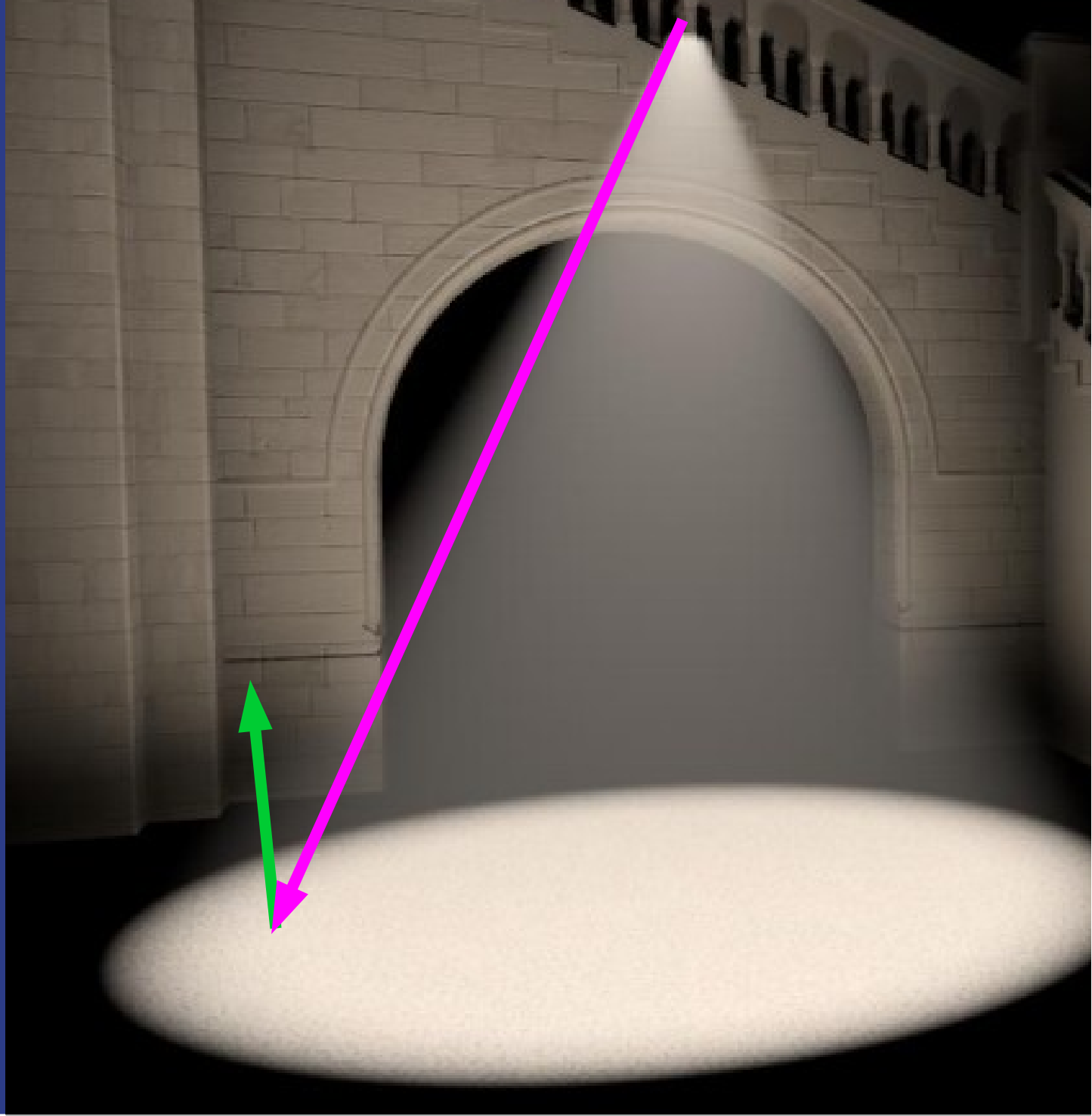


Źródło obrazu:
https://realtimeillumination.wordpress.com

Oświetlenie *lokalne* a *globalne*

1 odbicie światła

Oświetlenie niebezpośrednie
(ang. *indirect illumination*)



Źródło obrazu:
https://realtimeillumination.wordpress.com

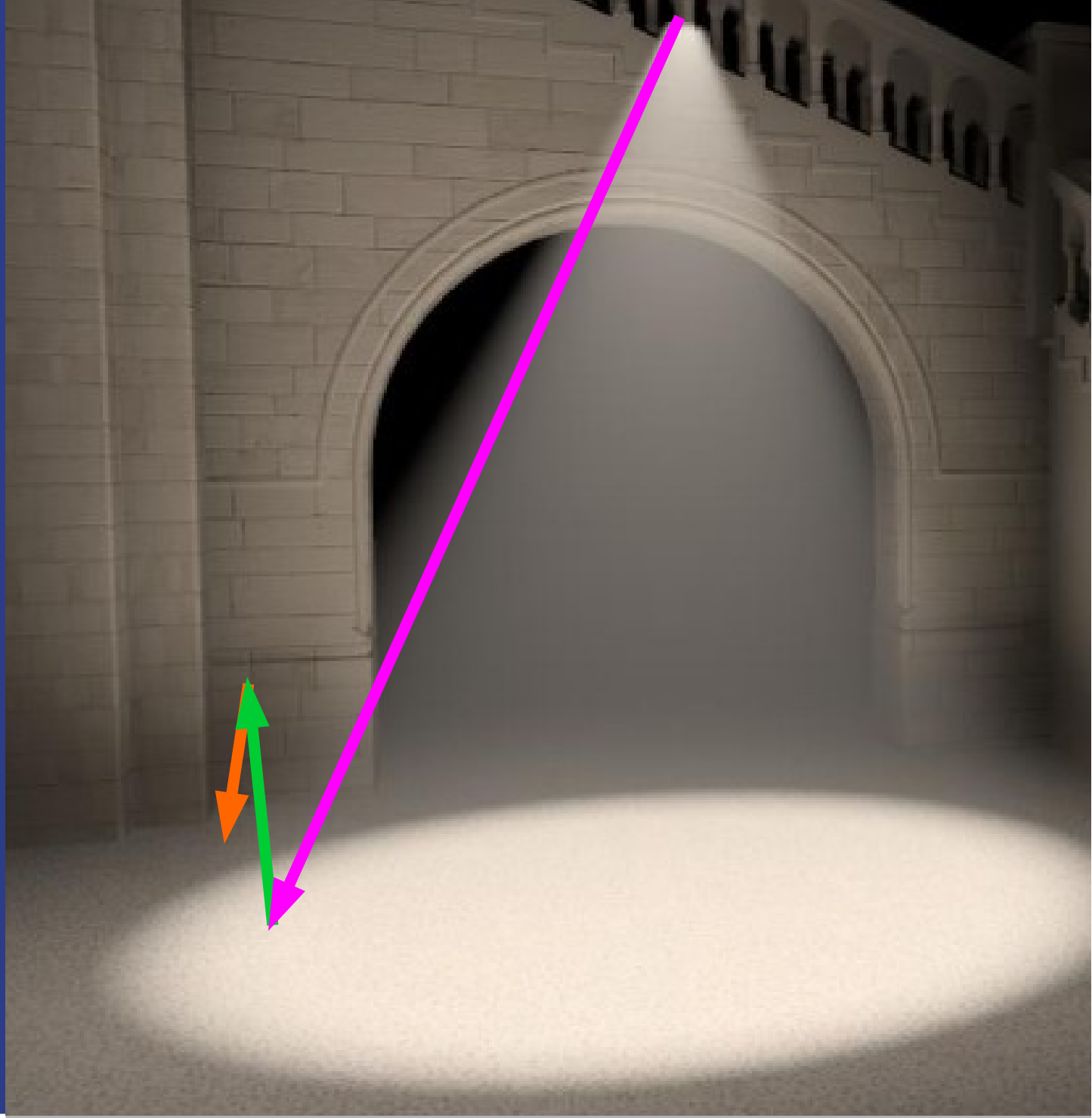
Oświetlenie *lokalne* a *globalne*

2 odbicia światła

Oświetlenie niebezpośrednie
(ang. *indirect illumination*)

...i tak dalej.

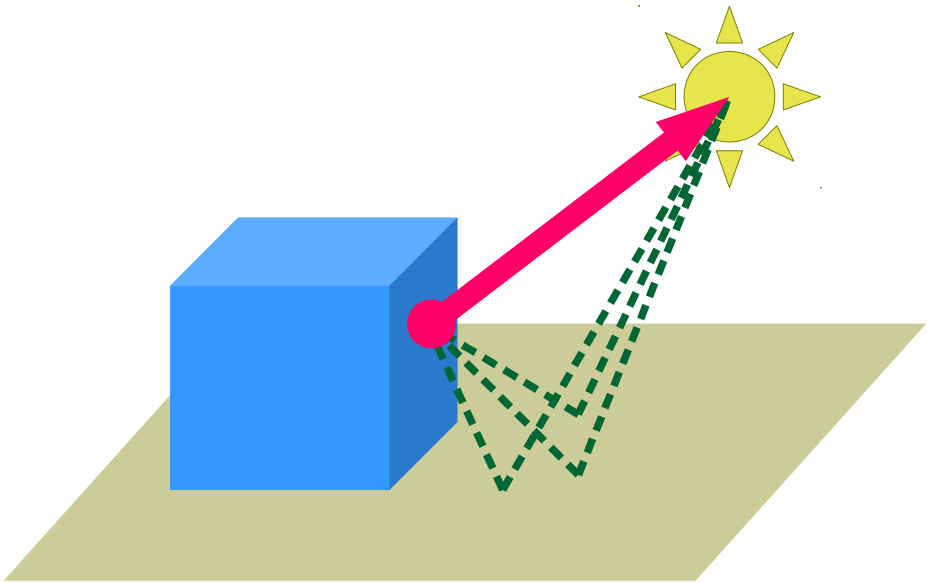
Im więcej odbić, tym dokładniejsza aproksymacja **oświetlenia globalnego**.

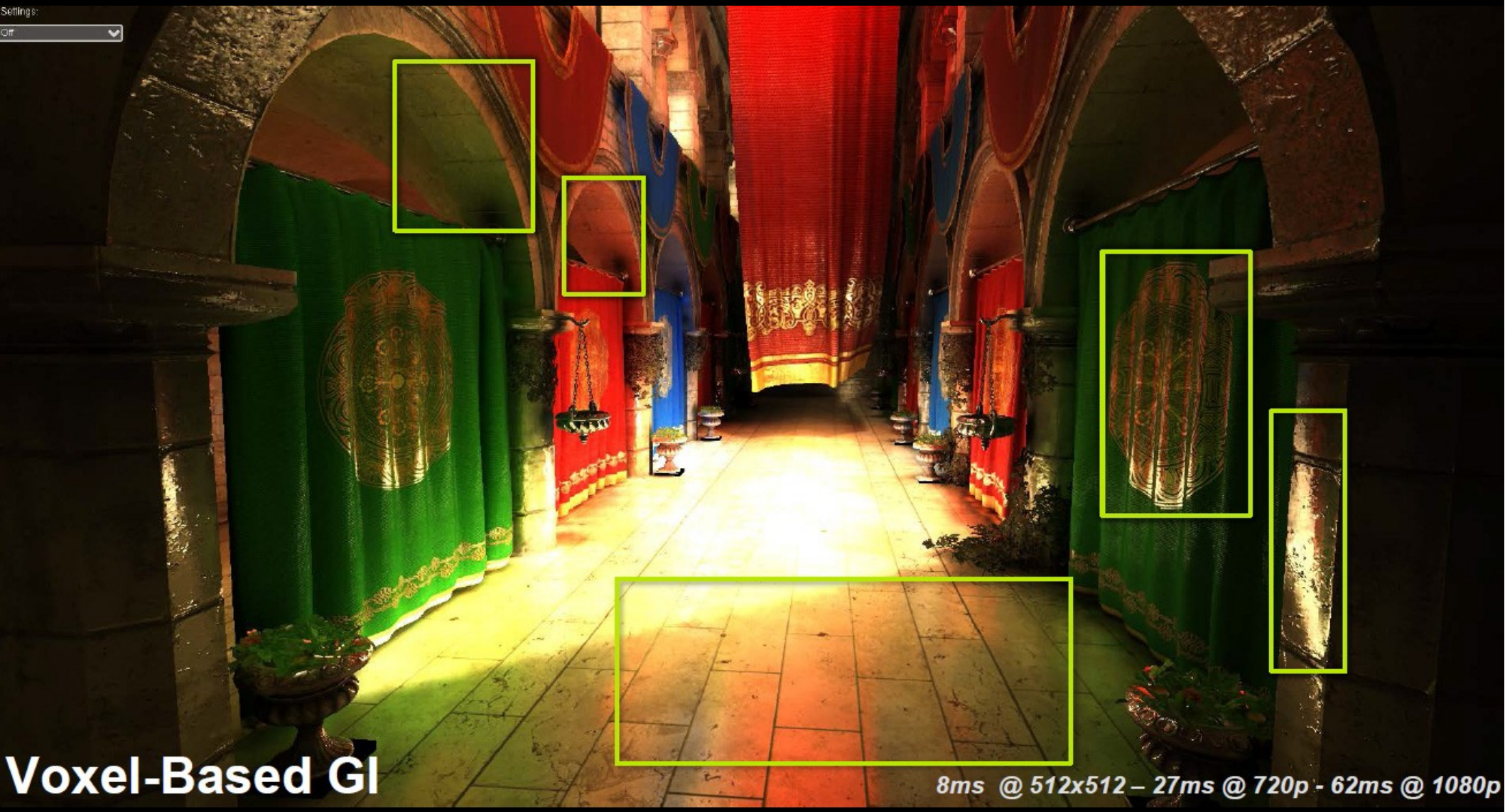


Źródło obrazu:
https://realtimeillumination.wordpress.com

Oświetlenie *lokalne* a *globalne*

- **Dlaczego** odbicia promieni światła są **trudne do uzyskania w grafice czasu rzeczywistego**?
  - Kolor obliczany jest **we fragmentach powierzchni**, które po projekcji odpowiadają pikselom bufora klatki
    - Czyli postępujemy niejako ***od końca*** z punktu widzenia światła (choć podejście nazywa się *forward rendering*)
  - **Nie śledzimy więc promienia** i jego odbić *od* źródła światła
    - Musielibyśmy przeanalizować *nieskończenie wiele* możliwości!
  - Obliczenia bazują na odcinku *łączącym* dany fragment ze źródłem światła

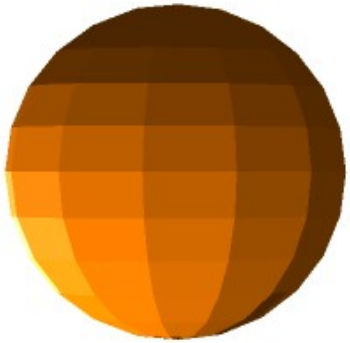




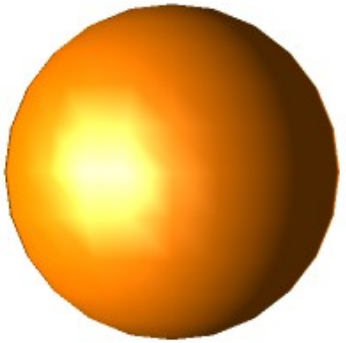
Źródło obrazu:
Cyril Crassin, NVIDIA

Oświetlenie a cieniowanie

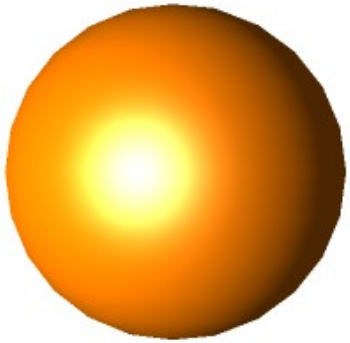
- Celem **symulacji oświetlenia** jest taka modyfikacja **kolorów** obiektów, aby sprawiały wrażenie oświetlonych fizycznymi źródłami światła
- Należy rozróżnić dwie rzeczy:
  - **Oświetlenie** – model matematyczny pozwalający na wyliczenie koloru z uwzględnieniem wpływu źródeł światła, cech materiału i środowiska
  - **Cieniowanie** (ang. *shading* – nie *shadowing*!) – sposób dystrybucji koloru pomiędzy miejscami, dla których jest on wyliczony



Płaskie



Gouraud



Phonga

Źródło obrazów:
http://docs.techsoft3d.com

Model oświetlenia

Radiancja – jednostka radiometryczna określająca strumień promieniowania na jednostkę powierzchni na jednostkę kąta bryłowego.

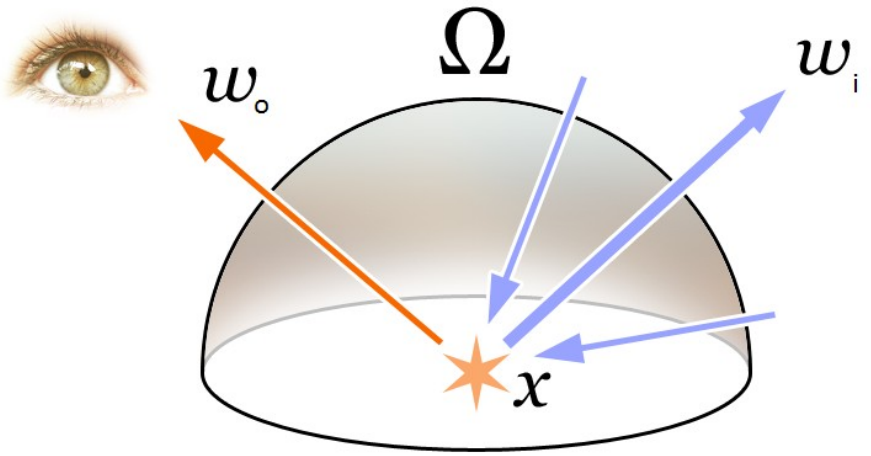
[W sr^{-1} m^{-2}]

L_o – wyjściowa radiancja
L_e – radiancja emitowana
L_i – radiancja nadchodząca
f_r – funkcja rozkładu odbicia
x – miejsce w przestrzeni
ω_o – wyjściowy kierunek światła
ω_i – wejściowy kierunek światła
λ – długość fali światła
n – wektor normalny w x
Ω – jednostkowa półsfera wokół x
t – czas

Źródło obrazu: Wikipedia

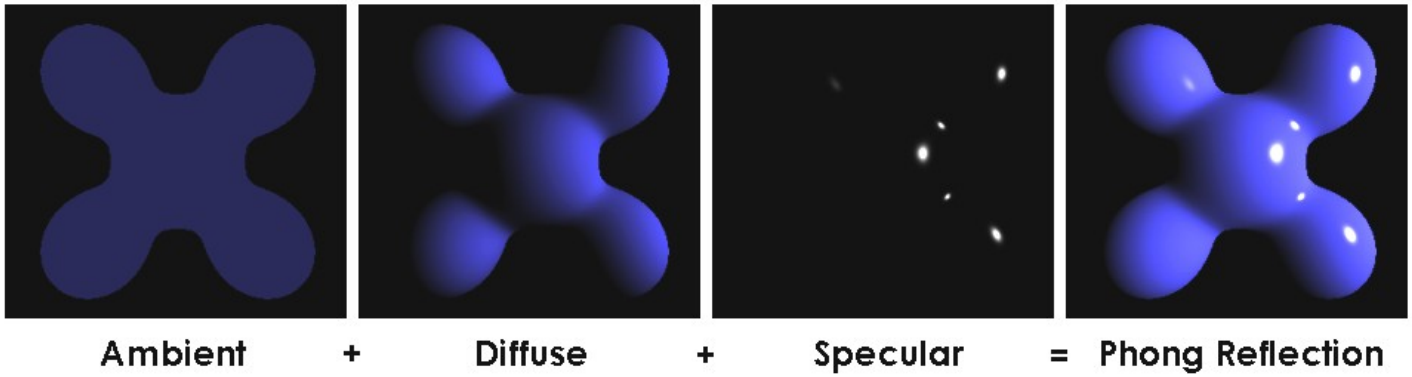
- Modele oświetlenia są próbami rozwiązania **równiania renderingu**, w uproszczeniu:
  - Obliczenie **radiancji** wyjściowej po odbiciu przez powierzchnię w zadanym punkcie *x*, na podstawie nadchodzącej fali światła i cech materiału którym pokryta jest ta powierzchnia

$$L_o(x, \omega_O, \lambda, t) = L_e(x, \omega_O, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t)\, L_i(x, \omega_i, \lambda, t)\, (\omega_i \circ n)\, \mathrm{d}\omega_i$$



Model oświetlenia

- **Najczęściej używanym** modelem oświetlenia w grafice czasu rzeczywistego jest **model Phonga**
  - **Niezależnie** od tego, czy stosujemy cieniowanie płaskie, *Gouraud*, czy *Phonga* – **model oświetlenia jest czymś osobnym**
- Model Phonga opiera się na rozbiciu światła na **trzy komponenty**:



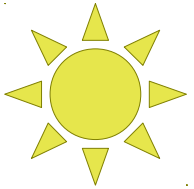
$$I_A = \begin{pmatrix} i_{r_A} \\ i_{g_A} \\ i_{b_A} \end{pmatrix} \quad I_D = \begin{pmatrix} i_{r_D} \\ i_{g_D} \\ i_{b_D} \end{pmatrix} \quad I_S = \begin{pmatrix} i_{r_S} \\ i_{g_S} \\ i_{b_S} \end{pmatrix}$$

$$I = I_A + I_D + I_S$$

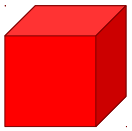
Źródło obrazów:
http://wikipedia.org

Światło i materiały

- Komponenty *Ambient*, *Diffuse* oraz *Specular* określamy zarówno **dla każdego ze źródeł światła**, jak i **dla materiału** z którego zbudowana jest **powierzchnia obiektu**



$$L_A = \begin{pmatrix} l_{r_A} \\ l_{g_A} \\ l_{b_A} \end{pmatrix} \quad L_D = \begin{pmatrix} l_{r_D} \\ l_{g_D} \\ l_{b_D} \end{pmatrix} \quad L_S = \begin{pmatrix} l_{r_S} \\ l_{g_S} \\ l_{b_S} \end{pmatrix}$$

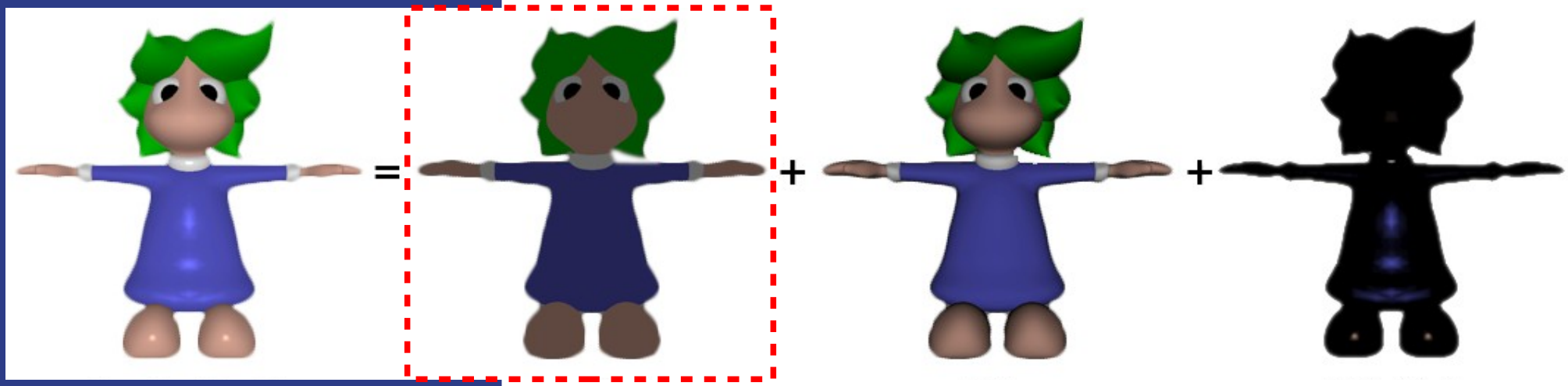


$$M_A = \begin{pmatrix} m_{r_A} \\ m_{g_A} \\ m_{b_A} \end{pmatrix} \quad M_D = \begin{pmatrix} m_{r_D} \\ m_{g_D} \\ m_{b_D} \end{pmatrix} \quad M_S = \begin{pmatrix} m_{r_S} \\ m_{g_S} \\ m_{b_S} \end{pmatrix}$$

- Wartości ***RGB***, najczęściej **znormalizowane** (czyli *<0;1>*)

Model oświetlenia Phonga

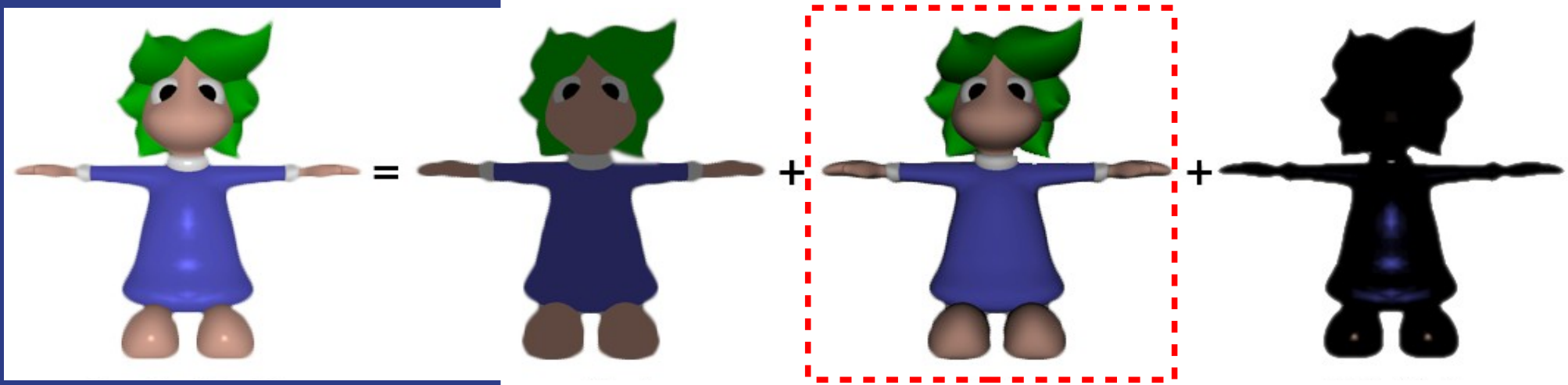
- Komponent **Ambient**
  - Wszystkie obiekty, wszystkie powierzchnie, **niezależnie od położenia czy orientacji**, oświetlone są **jednakowo**
  - Komponent ten służy jako bardzo (bardzo!) **uproszczona symulacja oświetlenia niebezpośredniego** (*indirect*).
    - Komponent *ambient* może być użyty w bardziej zaawansowanych technikach oświetlenia, np. *ambient occlusion*



Źródło obrazów:
http://evasion.imag.fr

Model oświetlenia Phonga

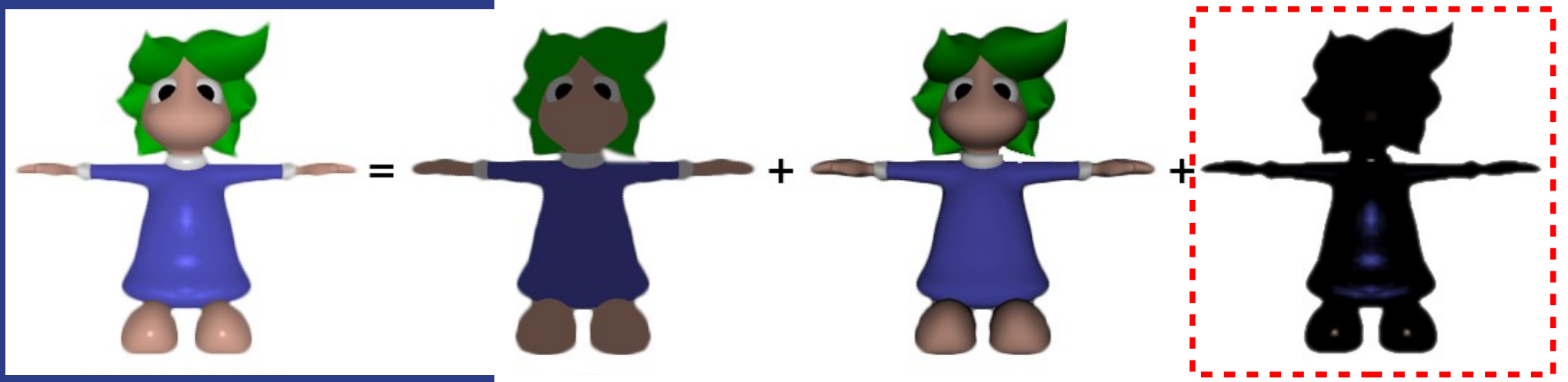
- Komponent **Diffuse**
  - Kolor uzależniony jest od kąta pomiędzy **kierunkiem padania światła**, a **orientacją powierzchni** w danym punkcie
    - Zależy więc od **położenia światła** względem naszego obiektu oraz **kąta, pod jakim ułożona jest powierzchnia** obiektu w danym punkcie
  - Jest to komponent związany z cechą tzw. ***powierzchni lambertowskich***, a więc takich które **rozpraszają światło jednakowo w każdym kierunku**
    - W rzeczywistości nie spotyka się takich idealnych powierzchni



Źródło obrazów:
http://evasion.imag.fr

Model oświetlenia Phonga

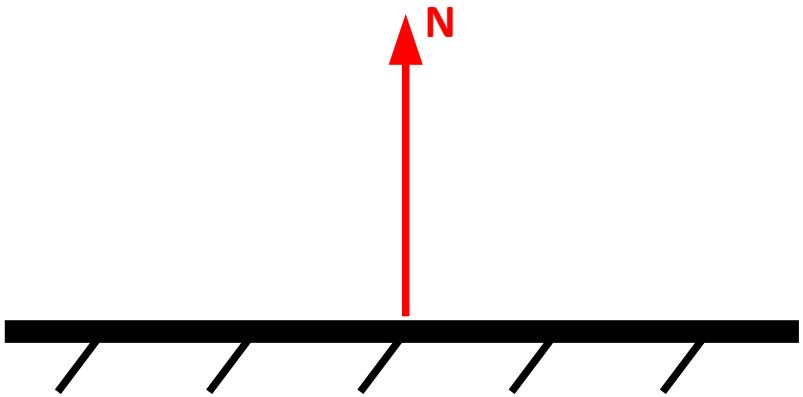
- Komponent **Specular**
  - Związany jest z powstawaniem **odbić światła**, tzw. *specular highlights*
    - Jest to jeden z najprostszych wyników nieregularności **powierzchni nielambertowskich**, które **odbijają światło**
    - Zależy od kąta pomiędzy **kierunkiem odbicia światła** w danym punkcie powierzchni, a **kierunkiem obserwacji** tego punktu
  - Przy symulacji rzeczywistych materiałów:
    - Dla powierzchni metalicznych powinien to być kolor biały
    - Dla innych materiałów (np. plastik) kolor powinien być zgodny z kolorem *diffuse*.



Źródło obrazów:
http://evasion.imag.fr

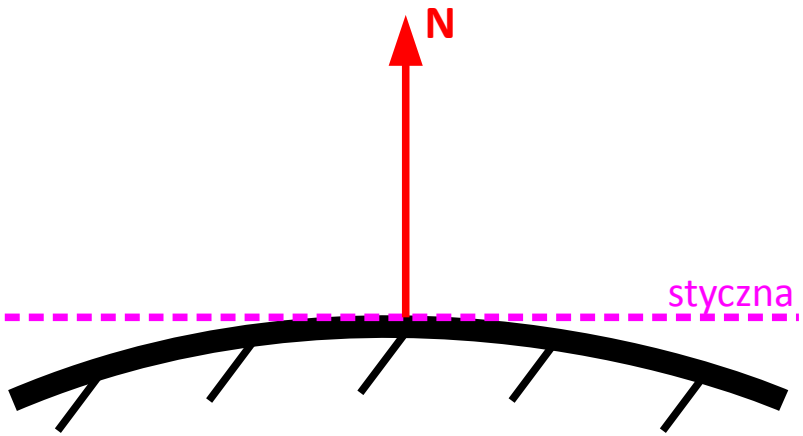
Wektory w oświetleniu

- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **normalny** (ang. *Normal*, często oznaczany jako *N*)
    - **Prostopadły do powierzchni** w danym punkcie
    - **Prostopadły do stycznej** w danym punkcie, gdy powierzchnia **nie jest płaska**



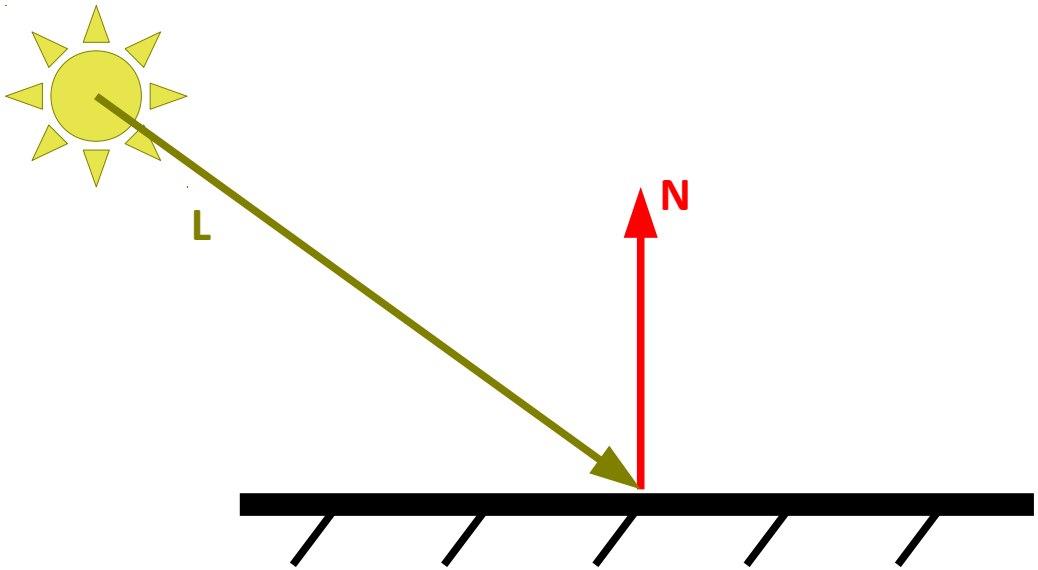
Wektory w oświetleniu

- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **normalny** (ang. *Normal*, często oznaczany jako *N*)
    - **Prostopadły do powierzchni** w danym punkcie
    - **Prostopadły do stycznej** w danym punkcie, gdy powierzchnia **nie jest płaska**



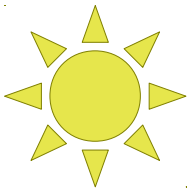
Wektory w oświetleniu

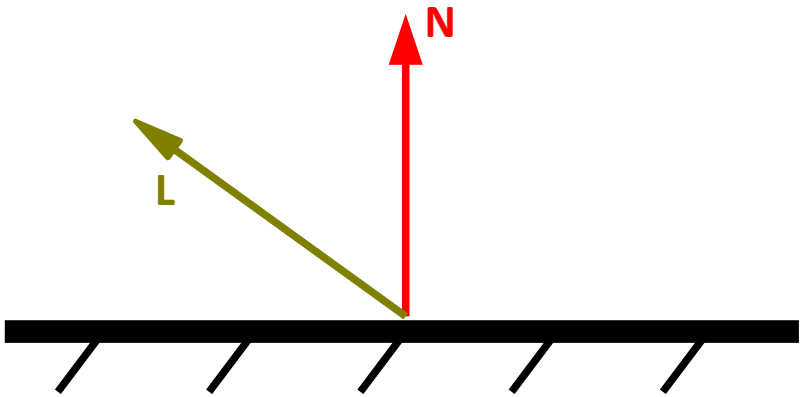
- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **światła** (ang. *Light*, często oznaczany jako *L*)
    - Kierunek od **danego punktu** na powierzchni obiektu do **źródła światła**
    - Kierunek **"padania"** światła
    - Najczęściej wektor **jednostkowy** (**znormalizowany**)
    - **Zwrot jest umowny**, zależy od naszej interpretacji i implementacji



Wektory w oświetleniu

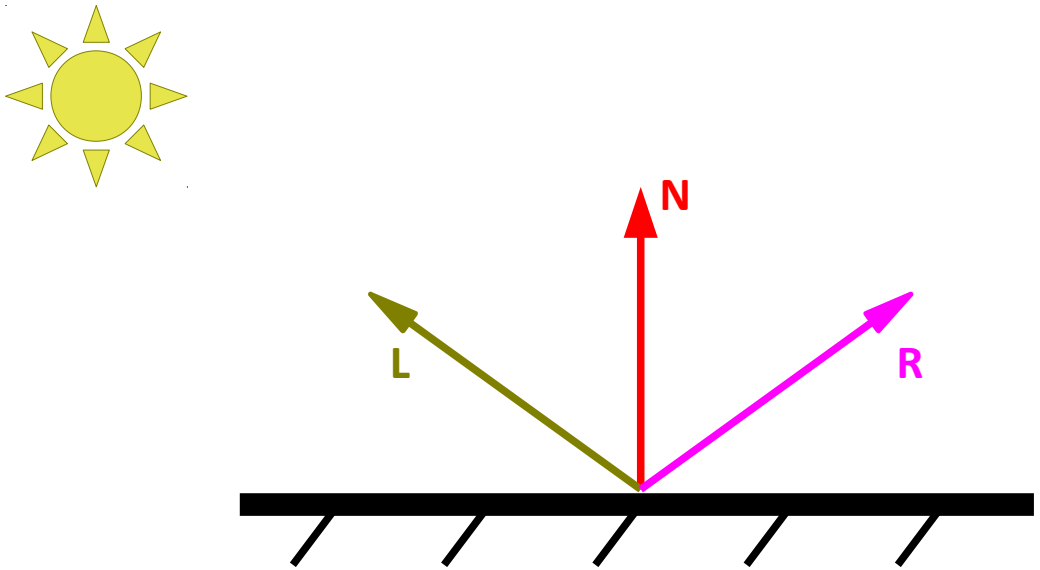
- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **światła** (ang. *Light*, często oznaczany jako *L*)
    - Kierunek od **danego punktu** na powierzchni obiektu do **źródła światła**
    - Kierunek **"padania"** światła
    - Najczęściej wektor **jednostkowy** (**znormalizowany**)
    - **Zwrot jest umowny**, zależy od naszej interpretacji i implementacji





Wektory w oświetleniu

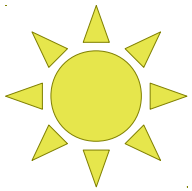
- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - **Odbity** wektor **światła** (ang. *Reflection*, często oznaczany jako *R*)
    - **Kąt** między *N* a *R* **jest taki sam**, jak kąt między *L* a *N*
    - Przydatny np. przy wyliczaniu komponentu ***specular*** który zależy od tego, czy promień odbity *trafia* do kamery

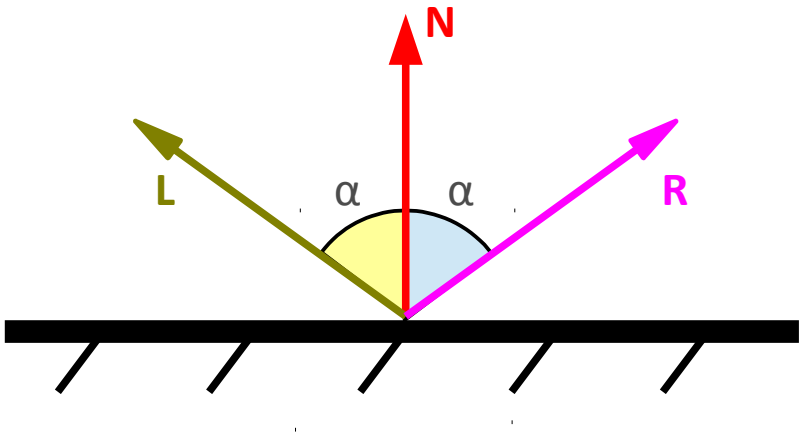


Wektory w oświetleniu

- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - **Odbity** wektor **światła** (ang. *Reflection*, często oznaczany jako *R*)
    - **Kąt** między *N* a *R* **jest taki sam**, jak kąt między *L* a *N*
    - Przydatny np. przy wyliczaniu komponentu ***specular*** który zależy od tego, czy promień odbity *trafia* do kamery

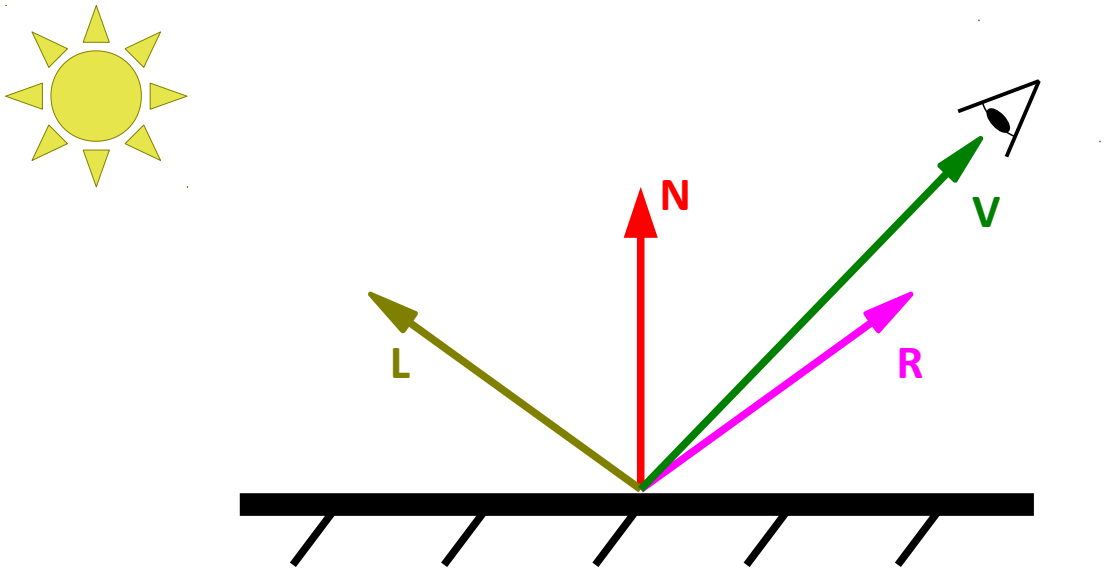
$$\vec{R} = 2(\vec{L} \circ \vec{N})\vec{N} - \vec{L}$$





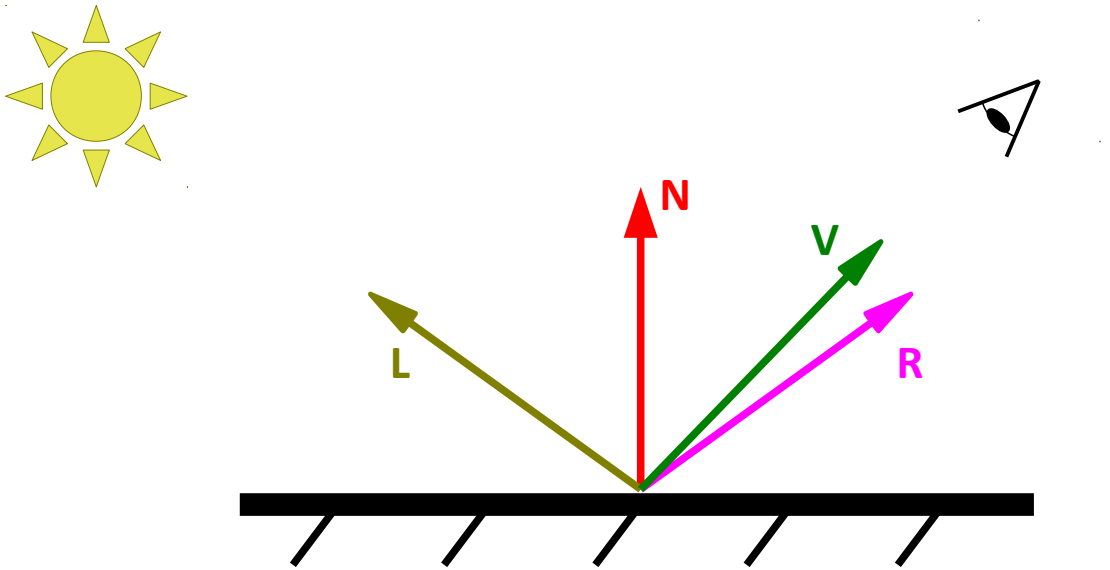
Wektory w oświetleniu

- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **widzenia** (ang. *View*, często oznaczany jako *V*)
    - **Łączy punkt z którego obserwujemy scenę z danym punktem** na powierzchni obiektu
    - Przydatny np. przy wyliczaniu komponentu ***specular*** który zależy od tego, czy promień odbity *trafia* do kamery



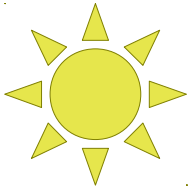
Wektory w oświetleniu

- Wektory przydatne przy obliczeniach związanych z oświetleniem:
  - Wektor **widzenia** (ang. *View*, często oznaczany jako *V*)
    - **Łączy punkt z którego obserwujemy scenę z danym punktem** na powierzchni obiektu
    - Przydatny np. przy wyliczaniu komponentu ***specular*** który zależy od tego, czy promień odbity *trafia* do kamery

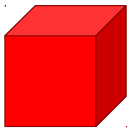


Światło i materiały

- Komponenty *Ambient*, *Diffuse* oraz *Specular* określamy zarówno **dla każdego ze źródeł światła**, jak i **dla materiału** z którego zbudowana jest **powierzchnia obiektu**



$$L_A = \begin{pmatrix} l_{r_A} \\ l_{g_A} \\ l_{b_A} \end{pmatrix} \quad L_D = \begin{pmatrix} l_{r_D} \\ l_{g_D} \\ l_{b_D} \end{pmatrix} \quad L_S = \begin{pmatrix} l_{r_S} \\ l_{g_S} \\ l_{b_S} \end{pmatrix}$$

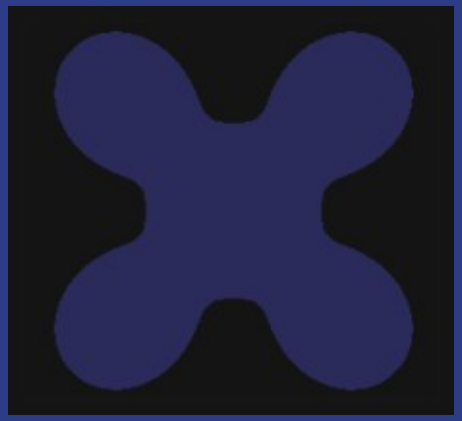


$$M_A = \begin{pmatrix} m_{r_A} \\ m_{g_A} \\ m_{b_A} \end{pmatrix} \quad M_D = \begin{pmatrix} m_{r_D} \\ m_{g_D} \\ m_{b_D} \end{pmatrix} \quad M_S = \begin{pmatrix} m_{r_S} \\ m_{g_S} \\ m_{b_S} \end{pmatrix}$$

Model oświetlenia Phonga

W tym wypadku **iloczyn dwóch wektorów** rozumiemy jako **iloczyn ich komponentów**!

(jak mnożenie „z kropką” w MATLABie)



- Kolor w danym wierzchołku/fragmencie (zależnie, czy cieniowanie *Gouraud*, czy *Phonga*), oblicza się **sumując wartości dla wszystkich źródeł światła**:

$$I = \sum_{i=1}^{lights} \left(I_{i_A} + I_{i_D} + I_{i_S} \right)$$

- Dla każdego źródła światła:
  - Komponent ***Ambient*** to prosty iloczyn wartości *RGB* światła i wartości *RGB* materiału

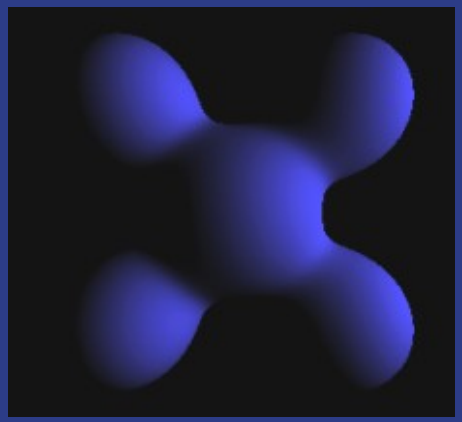
$$I_A = L_A M_A$$

    - Nie zależy od kierunku padania światła!

Model oświetlenia Phonga

W tym wypadku **iloczyn dwóch wektorów** rozumiemy jako **iloczyn ich komponentów**!

(jak mnożenie „z kropką” w MATLABie)

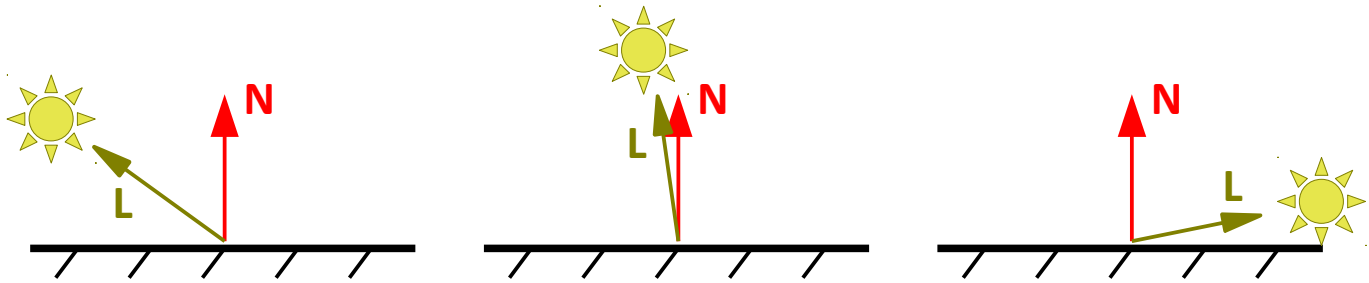


- Kolor w danym wierzchołku/fragmencie (zależnie, czy cieniowanie *Gouraud*, czy *Phonga*), oblicza się **sumując wartości dla wszystkich źródeł światła**:

$$I = \sum_{i=1}^{lights} \left(I_{i_A} + I_{i_D} + I_{i_S} \right)$$

- Dla każdego źródła światła:
  - Komponent ***Diffuse*** dodatkowo zależy od *współczynnika lambertowskiego*, zależnego od kąta pomiędzy wektorem normalnym *N* a wektorem kierunku padania światła *L*

$$I_D = k_D L_D M_D, \quad k_D = \vec{L} \circ \vec{N}$$

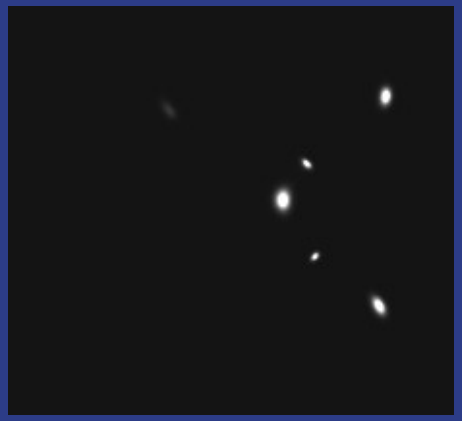


średnie k_D | k_D bliskie 1 | k_D bliskie 0

Model oświetlenia Phonga

W tym wypadku **iloczyn dwóch wektorów** rozumiemy jako **iloczyn ich komponentów**!

(jak mnożenie „z kropką” w MATLABie)

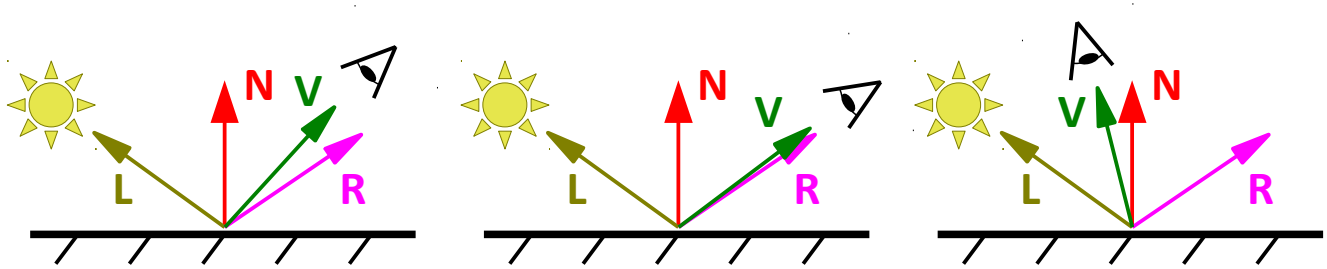


- Kolor w danym wierzchołku/fragmencie (zależnie, czy cieniowanie *Gouraud*, czy *Phonga*), oblicza się **sumując wartości dla wszystkich źródeł światła**:

$$I = \sum_{i=1}^{lights} \left(I_{i_A} + I_{i_D} + I_{i_S} \right)$$

- Dla każdego źródła światła:
  - Komponent ***Specular*** zależy od spójności wektora odbitego *R* oraz wektora kierunku obserwacji *V*
  - Wielkość rozbłysku zależy od wykładnika *shininess* (*α*)

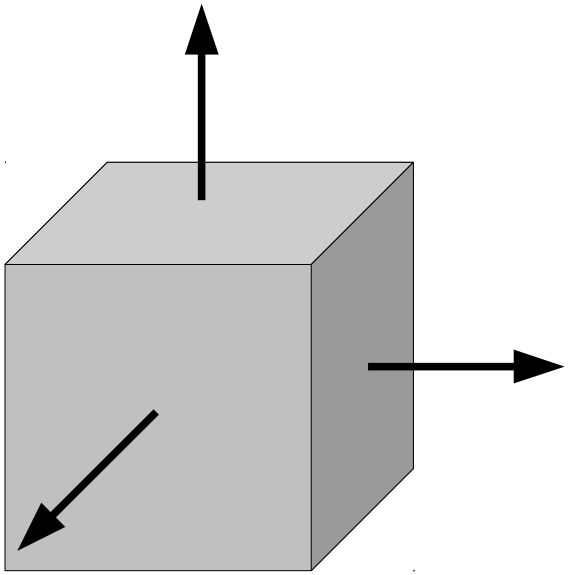
$$I_S = k_S L_S M_S, \quad k_S = (\vec{R} \circ \vec{V})^{\alpha}, \quad \vec{R} = 2(\vec{L} \circ \vec{N})\vec{N} - \vec{L}$$



średnie k_s | k_s bliskie 1 | k_s bliskie 0

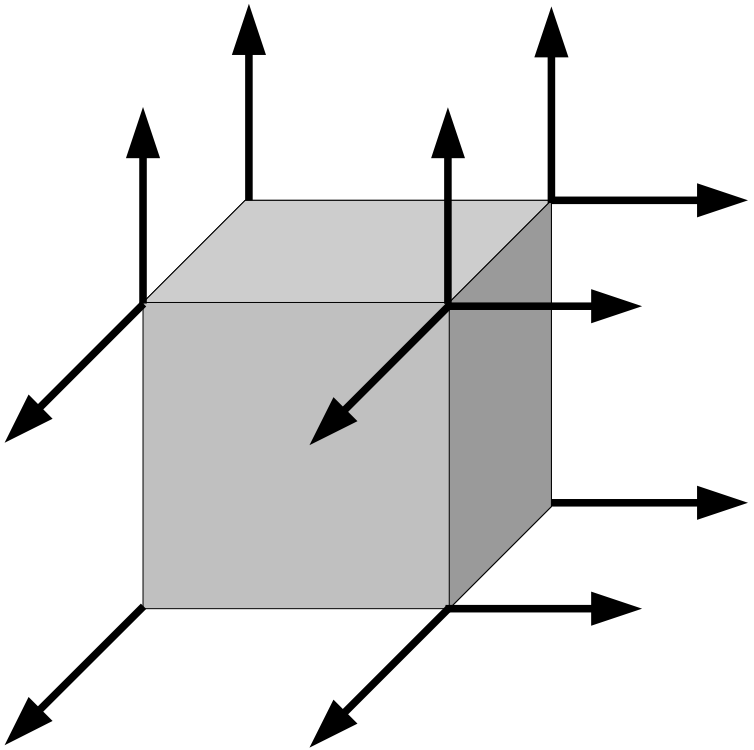
Wektory normalne

- W celu określenia skierowania punktu na powierzchni bryły, definiuje się tzw. **wektory normalne**
  - Są to wektory, które są **prostopadłe do powierzchni**
  - Zazwyczaj są to **wektory jednostkowe**



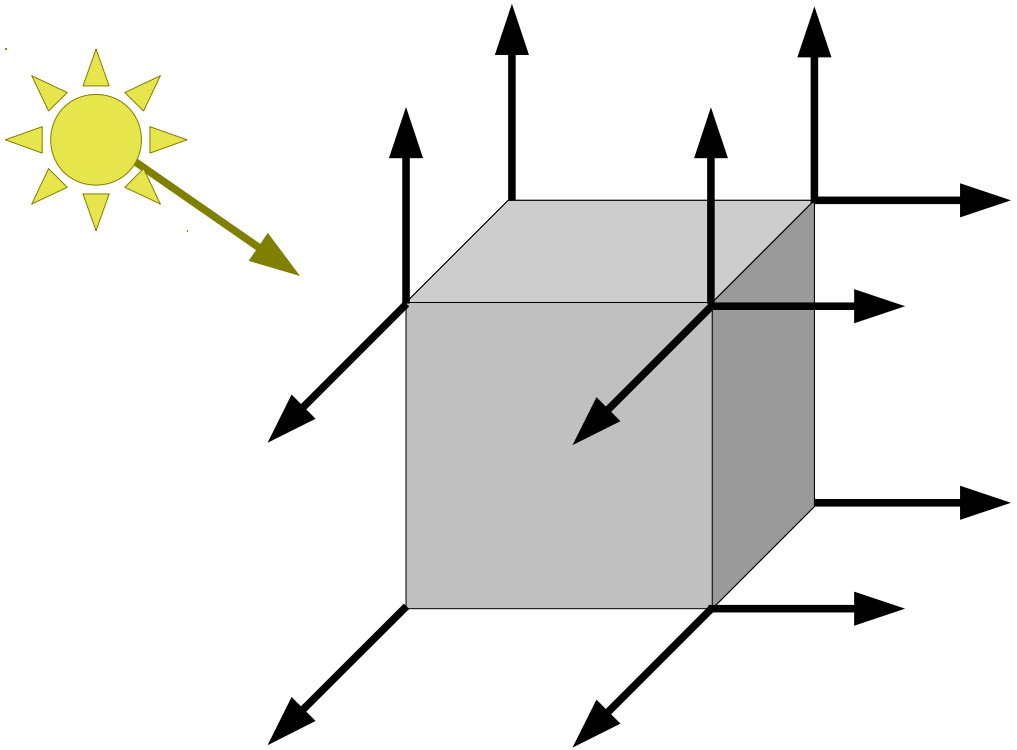
Wektory normalne

- W celu określenia skierowania punktu na powierzchni bryły, definiuje się tzw. **wektory normalne**
  - Są to wektory, które są **prostopadłe do powierzchni**
  - Zazwyczaj są to **wektory jednostkowe**
  - Wektory normalne określa się **dla wierzchołków**



Wektory normalne

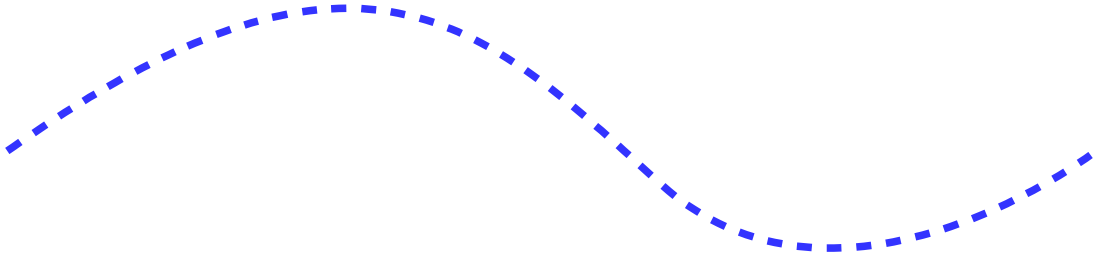
- W celu określenia skierowania punktu na powierzchni bryły, definiuje się tzw. **wektory normalne**
  - Są to wektory, które są **prostopadłe do powierzchni**
  - Zazwyczaj są to **wektory jednostkowe**
  - Wektory normalne określa się **dla wierzchołków**
  - Na ich podstawie można określić **wpływ światła padającego w zadanym kierunku na kolor powierzchni obiektu**



Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

    Chcemy przybliżyć niebieską, **gładką** powierzchnię.

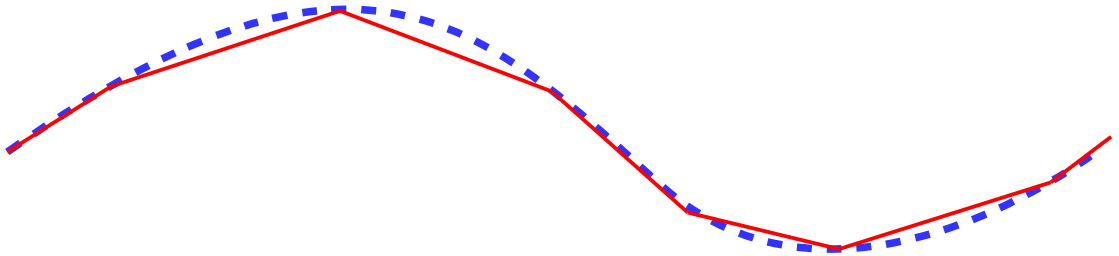


(widok z boku)

Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

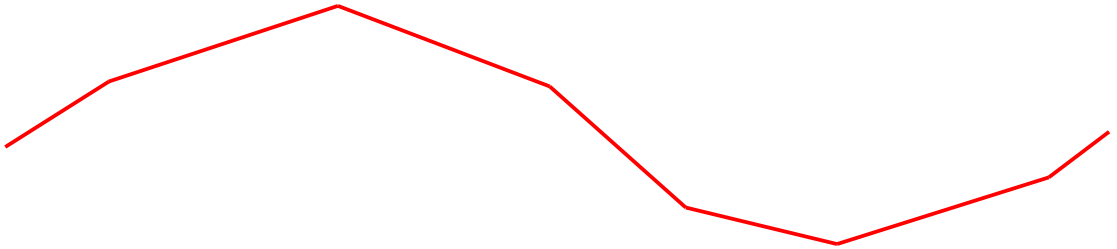
    W tym celu posłużymy się **ścianami**, które są pokazane w kolorze czerwonym.



Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

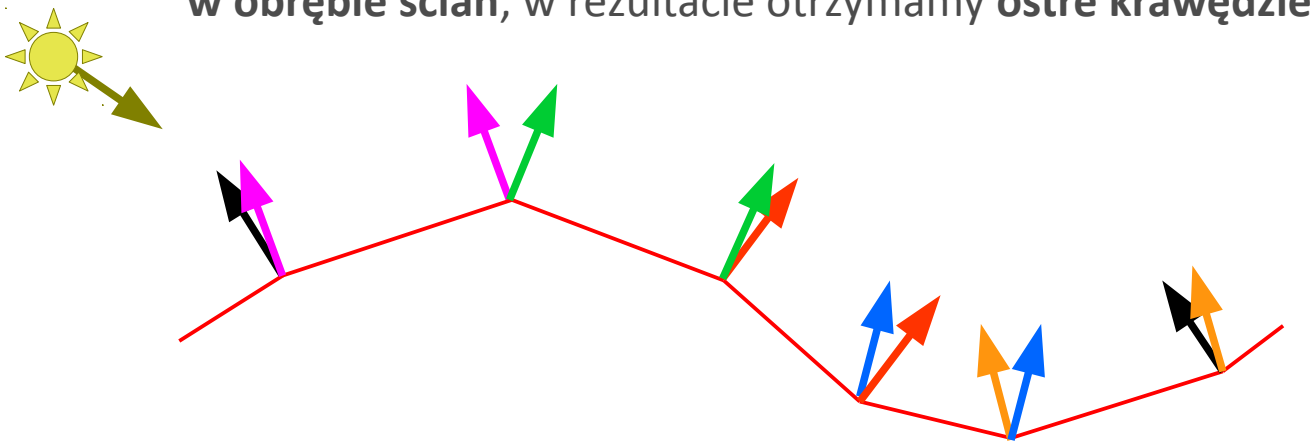
    Nasza "powierzchnia" **nie jest gładka**, ale możemy **cieniować** ją tak, *jakby była gładka*.



Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

    Jeśli wierzchołki będą miały **identyczne wektory normalne w obrębie ścian**, w rezultacie otrzymamy **ostre krawędzie**.

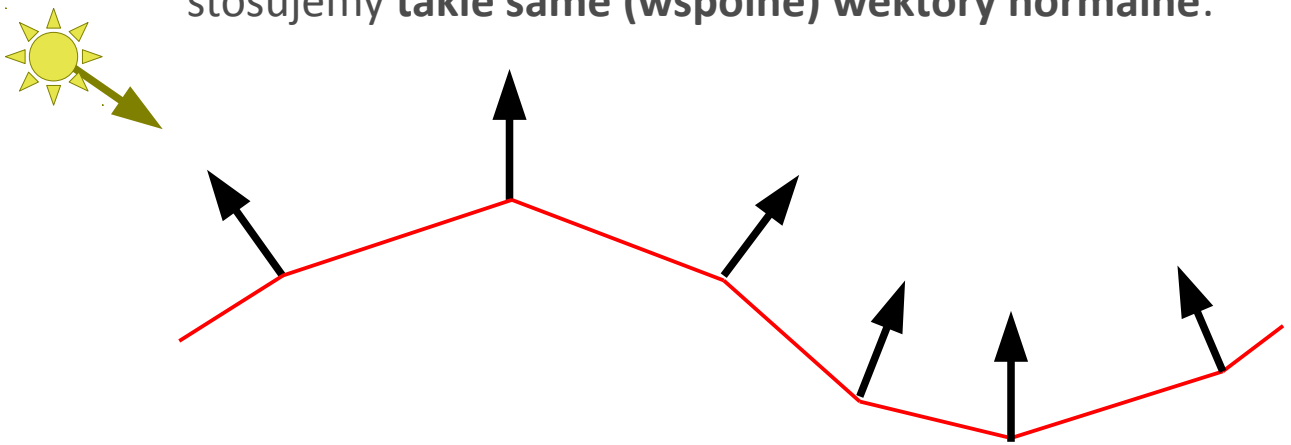


(widok z góry):

Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

    W **wierzchołkach wspólnych** dla dwóch lub więcej ścian, stosujemy **takie same (wspólne) wektory normalne**.

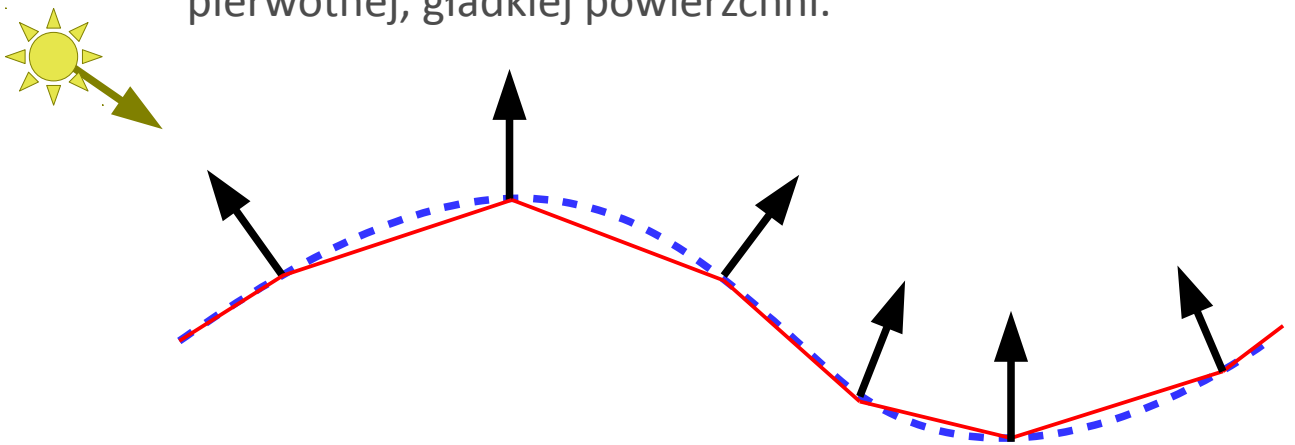


(widok z góry):

Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

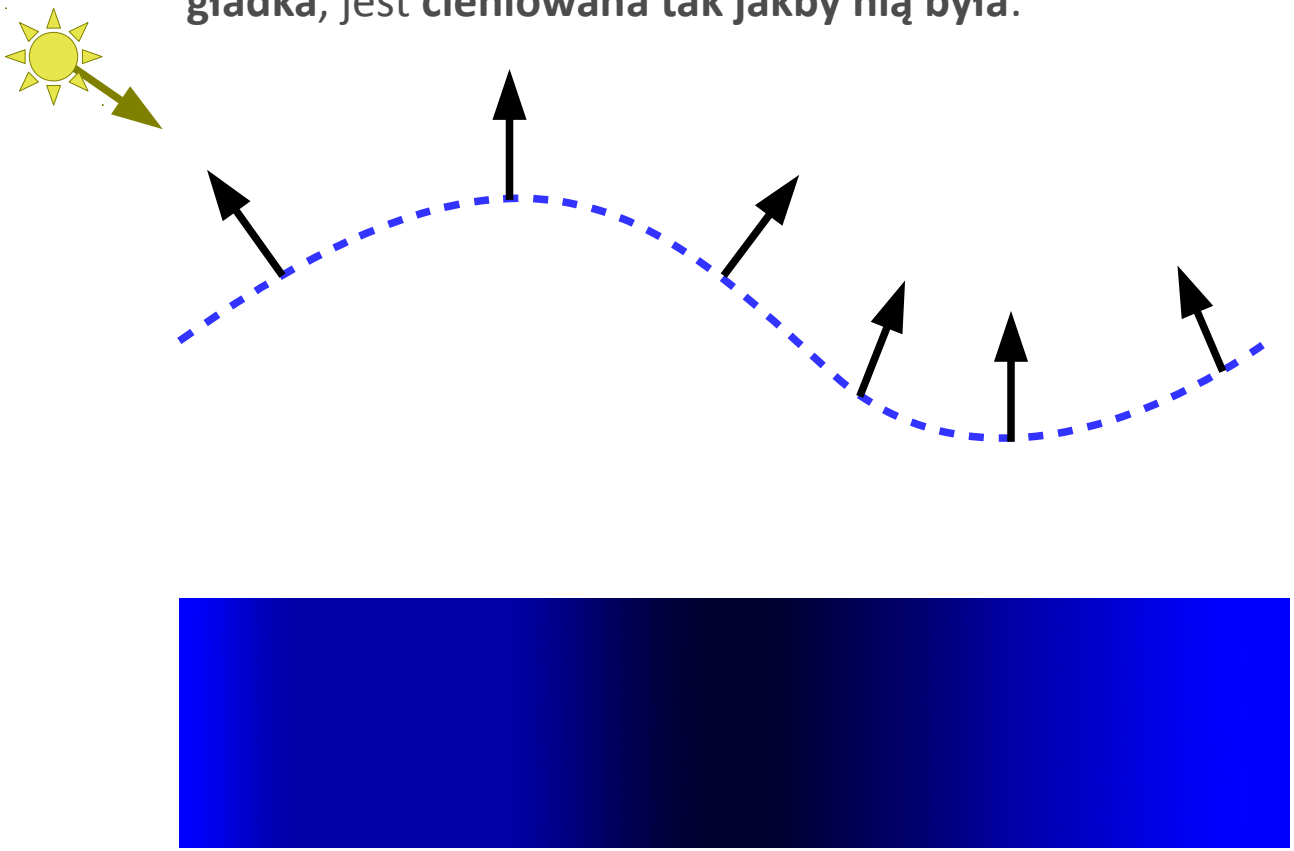
    W założeniu są to wektory **prostopadłe do stycznej** naszej pierwotnej, gładkiej powierzchni.



Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (rzut boczny):

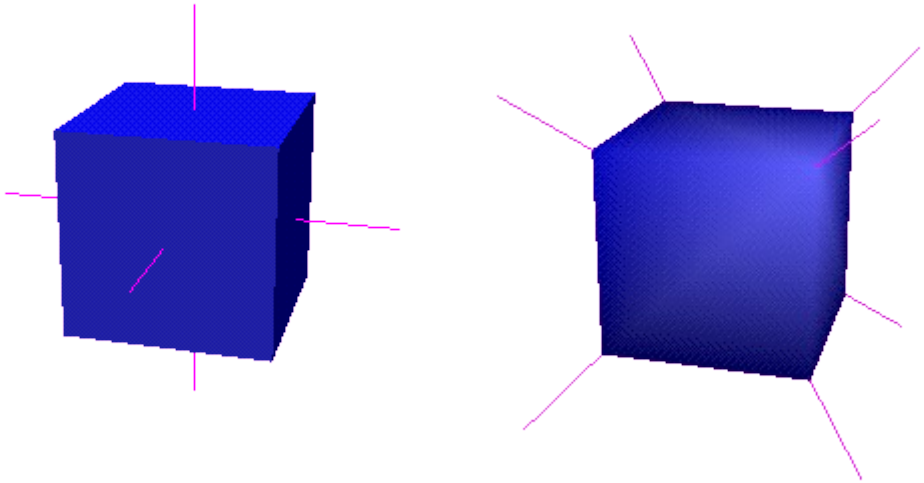
    Pomimo tego że nasza renderowana powierzchnia **nie jest gładka**, jest **cieniowana tak jakby nią była**.



Wektory normalne

- W celu uzyskania **łagodnych krawędzi**, wektory normalne powinny być **identyczne dla wszystkich ścian zbiegających się w danym wierzchołku**
  - **Przykład** (cieniowanie sześcianu):

    Ostre krawędzie są niekiedy pożądane. Cieniowanie sześcianu z użyciem współdzielonych wektorów normalnych może dać niepożądany efekt.



Źródło obrazów:
http://cs.berkeley.edu

Światło w OpenGL

- Model oświetlenia Phonga jest częścią nieprogramowalnego potoku renderowania w OpenGL
  - Włączenie obliczania oświetlenia:

```
glEnable(GL_LIGHTING);
```

- Nieprogramowalny potok renderowania OpenGL oferuje dwa rodzaje cieniowania
  - Cieniowanie płaskie

```
glShadeModel(GL_FLAT);
```

  - Cieniowanie *Gouraud* (domyślne; per-vertex)

```
glShadeModel(GL_SMOOTH);
```

  - Nie ma możliwości włączenia cieniowania Phonga (per-fragment) nie pisząc własnych programów cieniujących

Światło w OpenGL

- W nieprogramowalnym potoku renderowania OpenGL (ang. *fixed pipeline*) mamy możliwość zdefiniowania **przynajmniej ośmiu źródeł światła** (zależy od implementacji)
  - Do źródeł światła odwołujemy się **z użyciem stałych**:
    - **GL_LIGHT0**, **GL_LIGHT1**, **GL_LIGHT2**, ...
    - Można też użyć sposobu: **GL_LIGHT0 + 23**
  - Aby dane źródło światła działało, musi zostać **włączone**:

```
glEnable(GL_LIGHT1);
```

    - Domyślnie tylko GL_LIGHT0 jest włączone
  - Poszczególne komponenty źródła światła definiujemy następująco:

```
float l1Amb[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
float l1Dif[4] = { 1.0f, 0.0f, 1.0f, 1.0f };
float l1Spe[4] = { 0.5f, 0.5f, 0.5f, 1.0f };
glLightfv(GL_LIGHT1, GL_AMBIENT, l1Amb);
glLightfv(GL_LIGHT1, GL_DIFFUSE, l1Dif);
glLightfv(GL_LIGHT1, GL_SPECULAR, l1Spe);
```

Światło w OpenGL

- Dodatkowo w OpenGL istnieje **globalne** oświetlenie typu *ambient* **niezależne od źródeł światła**
  - Jego wartość jest **dodawana do wszystkich** wierzchołków, dla których używane jest oświetlenie
  - Często **warto je wyłączyć** już na samym początku, aby nie myliło i nie sugerowało błędów w innych miejscach kodu

```
float globAmb[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globAmb);
```

Materiały

- **Materiał jest cechą danego wierzchołka**
  (analogicznie do koloru ustawianego za pomocą *glColor*()*)
  - Po ustawienia materiału i włączeniu oświetlenia, **kolory** wierzchołków domyślnie **nie są już brane pod uwagę**
  - Oddzielnie ustawia się materiał **strony przedniej i tylnej**, w praktyce zazwyczaj używa się jedynie przedniej (*GL_FRONT*)
  - Materiał ustawia się następująco:

```
float mAmb[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
float mDif[4] = { 1.0f, 1.0f, 0.0f, 1.0f };
float mSpe[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
glMaterialfv(GL_FRONT, GL_AMBIENT, mAmb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mDif);
glMaterialfv(GL_FRONT, GL_SPECULAR, mSpe);
```

  - Dodatkowo można określić wartość eksponenty związanej z komponentem *specular*:

```
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
```

    - Im większe *shininess*, tym mniejszy rozbłysk

Wektory normalne

Definicja wierzchołków zostanie szczegółowo omówiona **na następnym wykładzie**!

- W OpenGL wektory normalne definiujemy w sposób podobny do pozycji wierzchołków

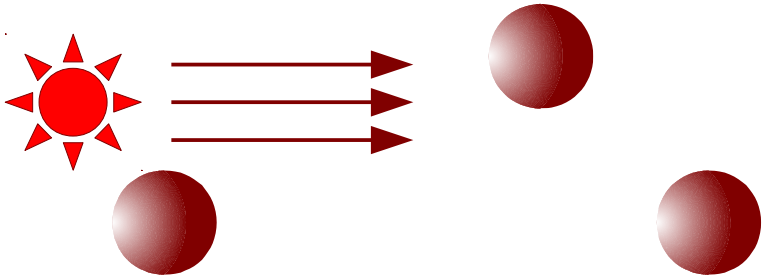
```
glNormal3f(0.0f, 1.0f, 0.0f);
```

- Ostatnio określony wektor normalny zostaje powiązany z każdym nowo tworzonym wierzchołkiem

```
glNormal3f(0.0f, 1.0f, 0.0f);
glVertex3f(0.0f, 0.0f, 0.0f); // normal = (0;1;0)
glNormal3f(0.0f, 0.0f, 1.0f);
glVertex3f(1.0f, 0.0f, 0.0f); // normal = (0;0;1)
glVertex3f(0.0f, 1.0f, 0.0f); // normal = (0;0;1)
glVertex3f(0.0f, 0.0f, 1.0f); // normal = (0;0;1)
```

Źródła światła

- Możemy wyróżnić trzy podstawowe rodzaje źródeł światła:
  - **Światła kierunkowe**
    - Wszystkie promienie są do siebie **równoległe**



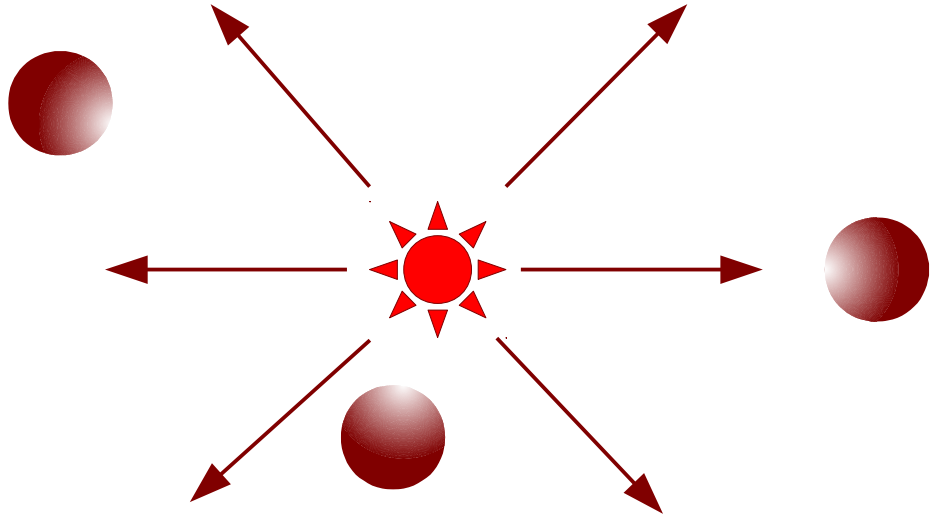
    - Konieczne jest zdefiniowanie **kierunku świecenia**
      - W *OpenGL*:

```
float lightPos[4] = { 0.0f, 0.0f, 1.0f, 0.0f };
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
```

      - **Ważne!** Ostatni komponent pozycji musi być **równy 0**, wtedy źródło światła traktowane jest jako kierunkowe. Inaczej jest ono światłem pozycyjnym.
      - **Ważne!** Podajemy wartości będące **przeciwieństwem kierunku** świecenia. W przykładzie promienie będą się rozchodzić w kierunku (0;0;-1).

Źródła światła

- Możemy wyróżnić trzy podstawowe rodzaje źródeł światła:
  - **Światła punktowe**
    - Światło **pozycyjne**, promienie rozchodzą się **dookólnie**



    - Konieczne jest określenie **pozycji** źródła światła
      - W *OpenGL*:

```
float lightPos[4] = { 0.0f, 0.0f, 1.0f, 1.0f };
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
```

      - **Ważne!** Ostatni komponent pozycji musi być **równy 1**, wtedy źródło światła traktowane jest jako pozycyjne. Inaczej jest ono światłem kierunkowym.

Źródła światła

- Możemy wyróżnić trzy podstawowe rodzaje źródeł światła:
  - **Światła punktowe (c.d.)**
    - Dodatkowo można określić **tłumienie wraz z odległością**
      - Składa się z trzech komponentów: **stałego, liniowego i kwadratowego**

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

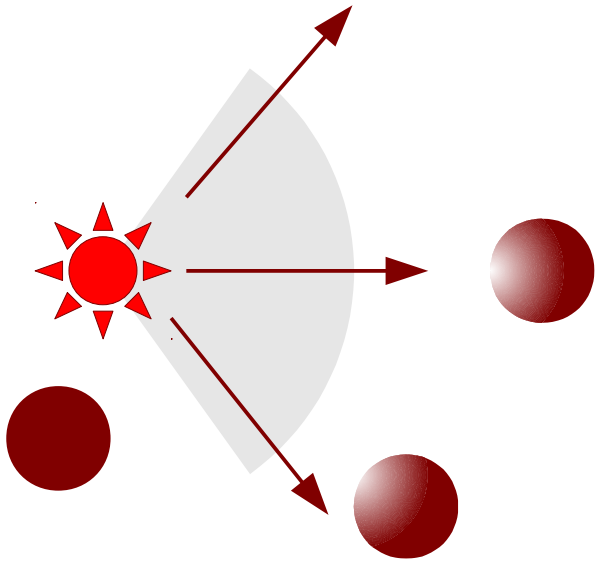
      - W *OpenGL*:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0f);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0f);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5f);
```

      - Tłumienie wpływa na **wszystkie komponenty** światła

Źródła światła

- Możemy wyróżnić trzy podstawowe rodzaje źródeł światła (c.d.):
  - **Światła stożkowe (ang. *spotlights*)**
    - Skierowane światło punktowe o ograniczonym kącie

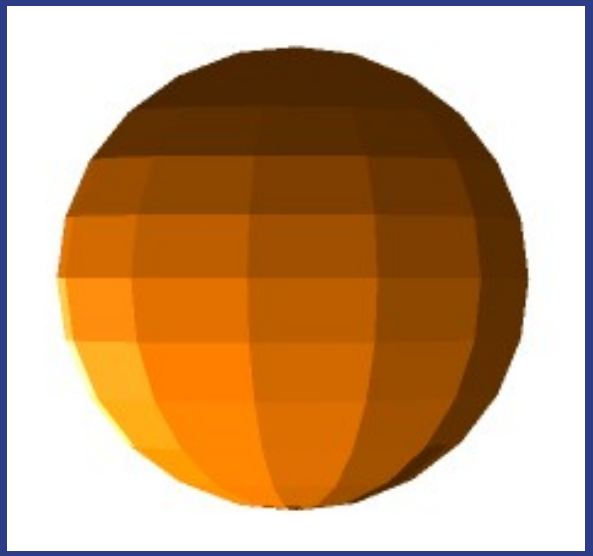


    - Oprócz **pozycji**, określa się także **kierunek** światła oraz szerokość **kąta** świecenia:

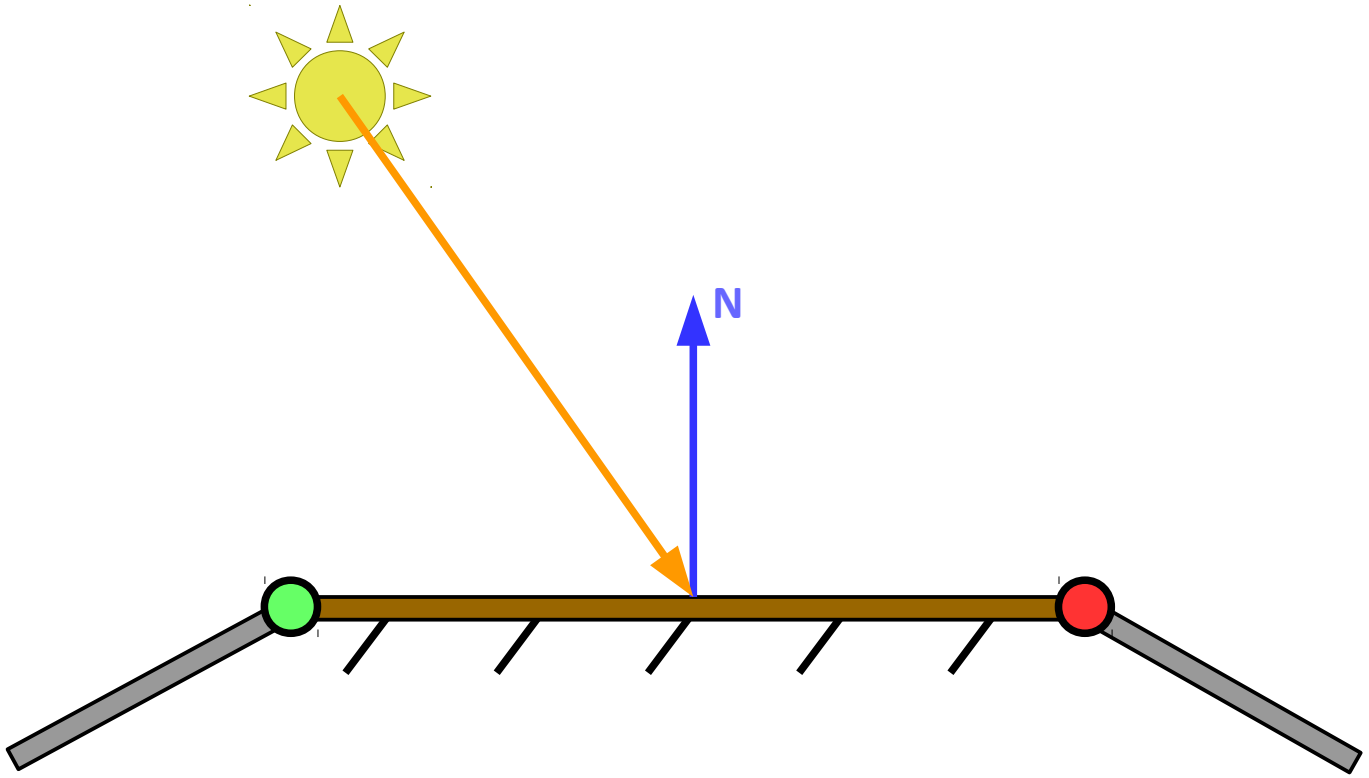
```
float lightDir[3] = { 0.0f, 0.0f, 1.0f };
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, lightDir);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0f);
```

Techniki cieniowania

- **Cieniowanie płaskie** (ang. *flat shading*)
  - Podejście *per-face*
  - Wykorzystywany jest tylko **jeden wektor normalny**
  - Model oświetlenia rozwiązywany tylko **w jednym miejscu**
  - Fragmenty pomiędzy wierzchołkami wypełniane tym samym, **jednym kolorem** będącym rezultatem powyższego kroku

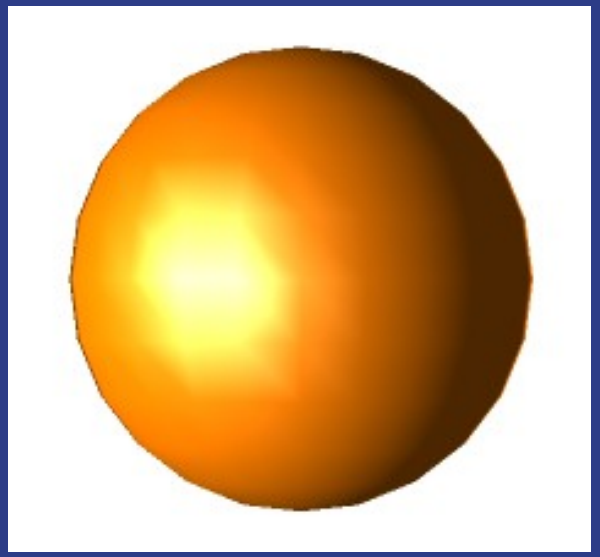


Źródło obrazu:
http://docs.techsoft3d.com

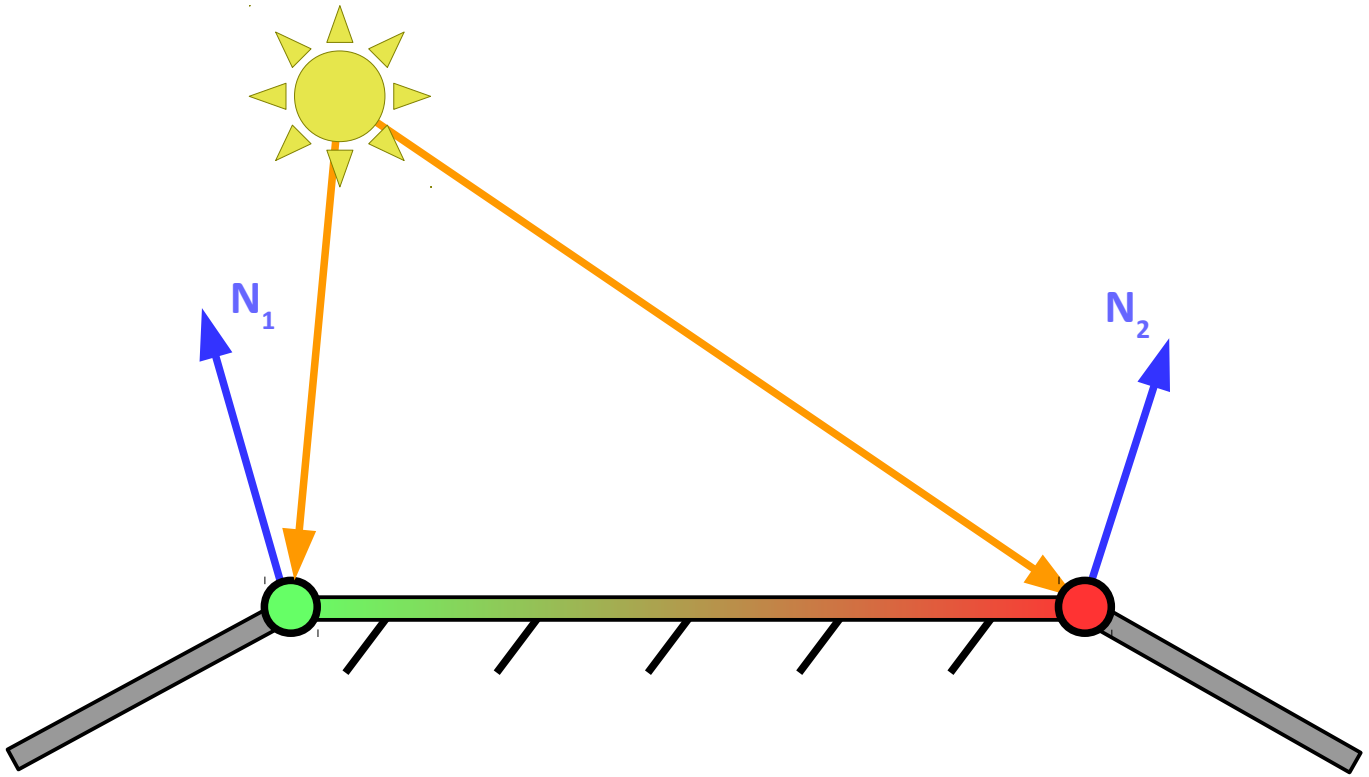


Techniki cieniowania

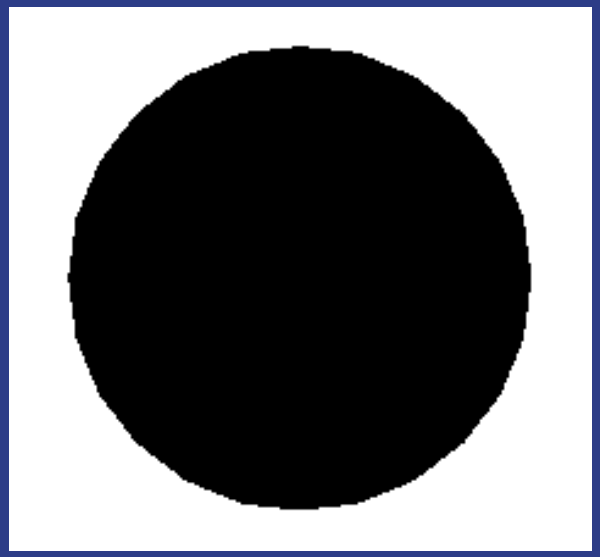
- **Cieniowanie Gouraud** (od nazwiska *Henri Gouraud*)
  - Podejście *per-vertex*
  - Wykorzystywane są **wektory normalne w wierzchołkach**
  - Oświetlenie obliczane jest **we wszystkich wierzchołkach**
  - Fragmenty pomiędzy wierzchołkami wypełniane są poprzez **interpolację obliczonego koloru wierzchołków**



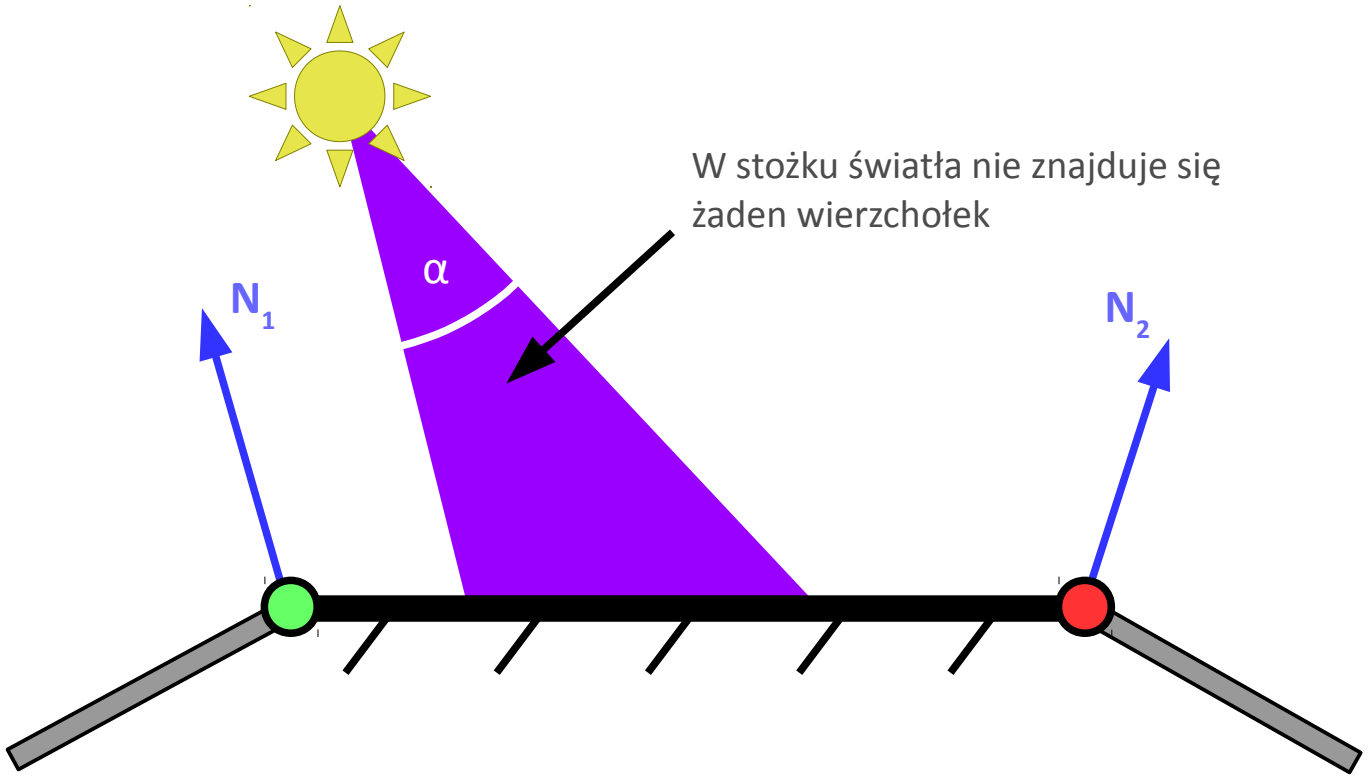
Źródło obrazu:
http://docs.techsoft3d.com



Techniki cieniowania

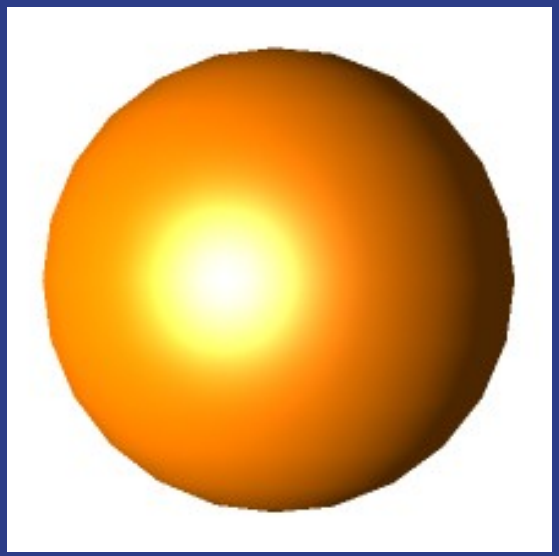


- **Cieniowanie Gouraud** (od nazwiska *Henri Gouraud*)
  - Dlaczego to podejście **nie jest wystarczające**?
    - Kolor nie powinien zmieniać się liniowo, ważna jest relacja między wektorem normalnym a kierunkiem padania światła
    - Niskiej jakości reprezentacja tłumienia, bardziej złożonych świateł i technik jak np. *spotlight*, *specular* itp.
      - Jeśli stożek światła nie trafi w żaden wierzchołek, to **zostanie zignorowany**!

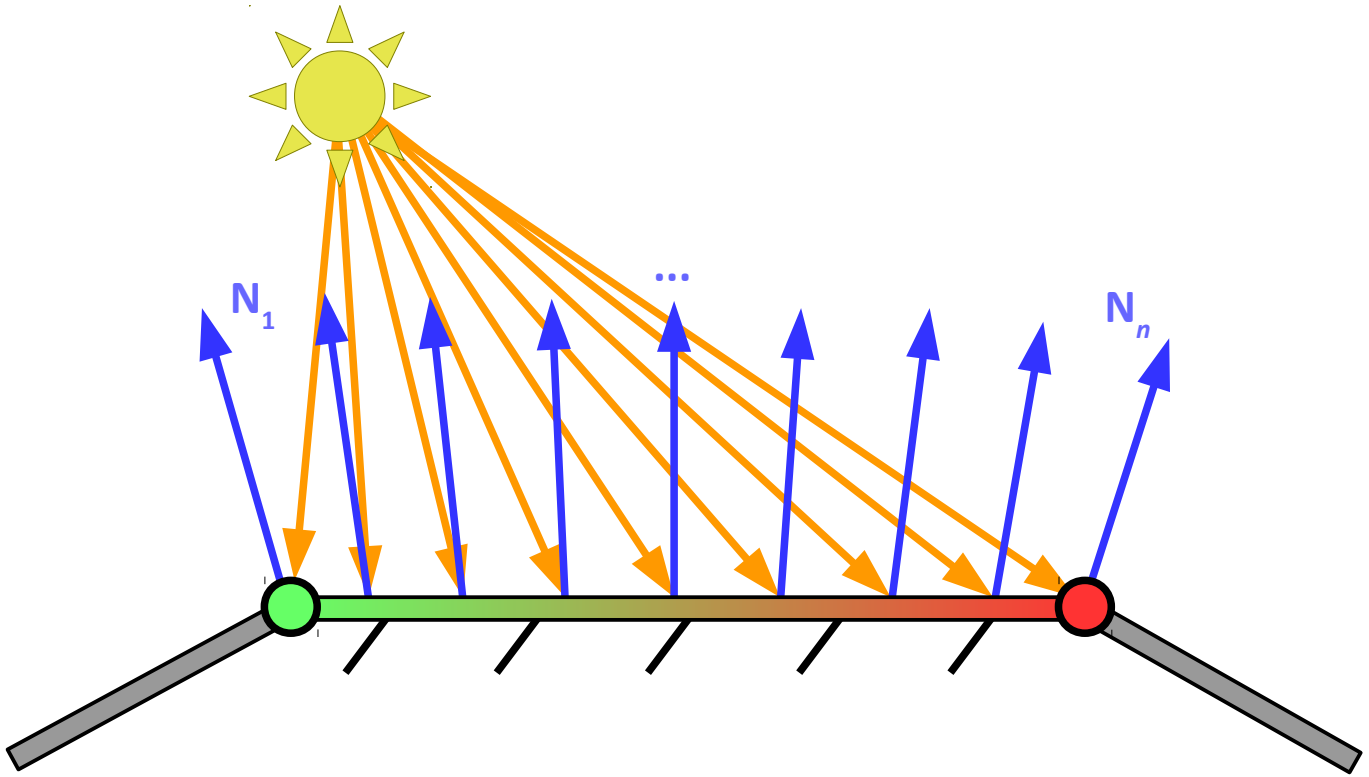


Techniki cieniowania

- **Cieniowanie Phonga** (od nazwiska *Bui Thuong Phong*)
  - Podejście *per-fragment*
  - Wektory **normalne** są **interpolowane pomiędzy wierzchołkami**
  - Cały **model oświetlenia jest rozwiązywany dla każdego fragmentu osobno**

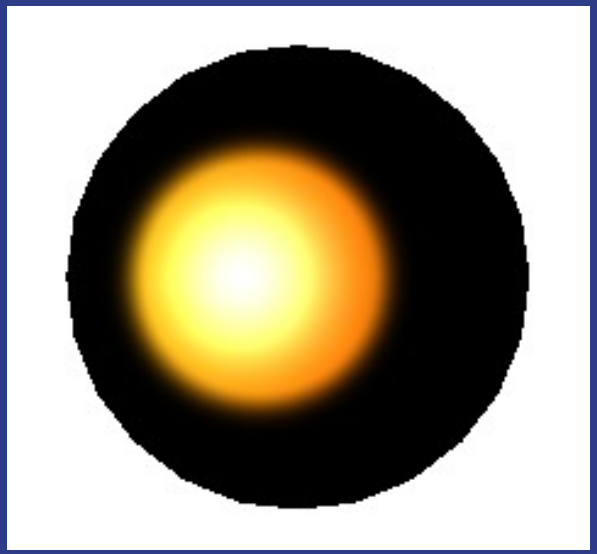


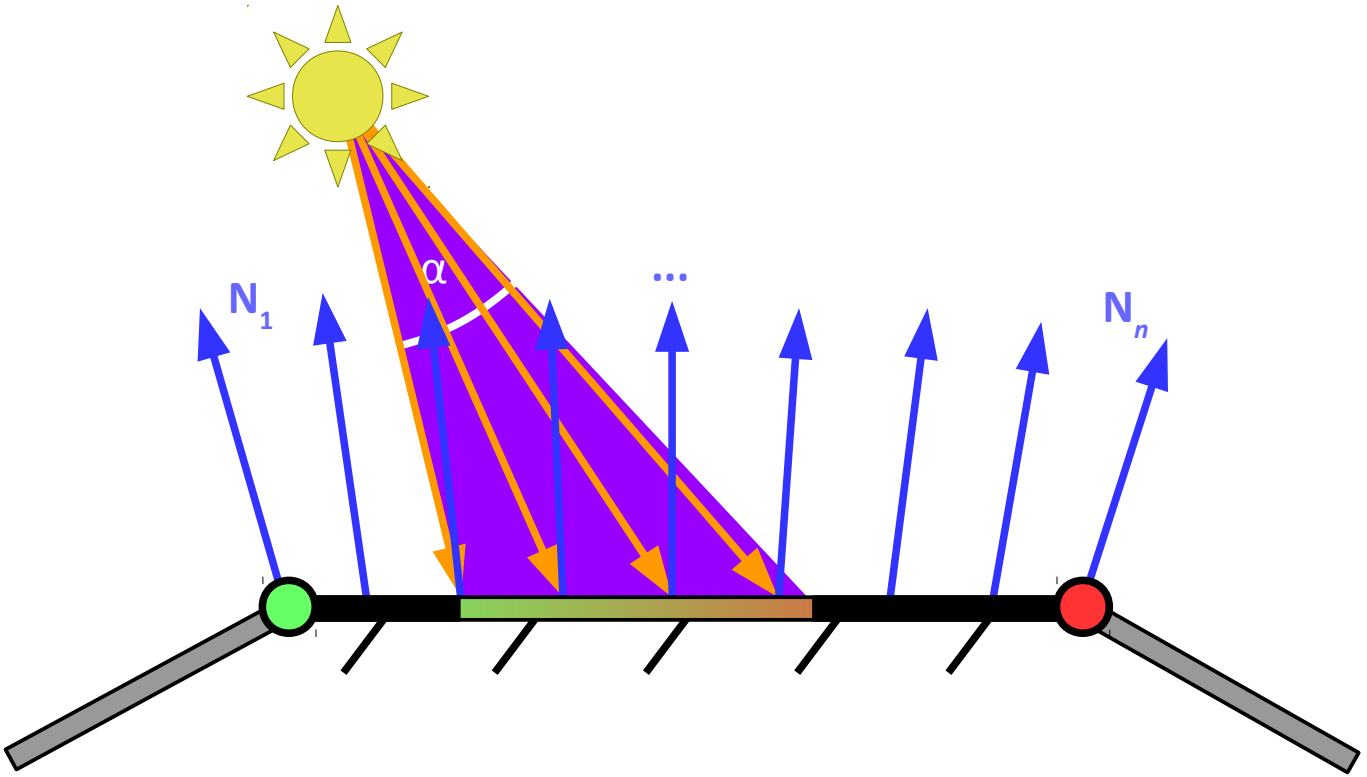
Źródło obrazu:
http://docs.techsoft3d.com



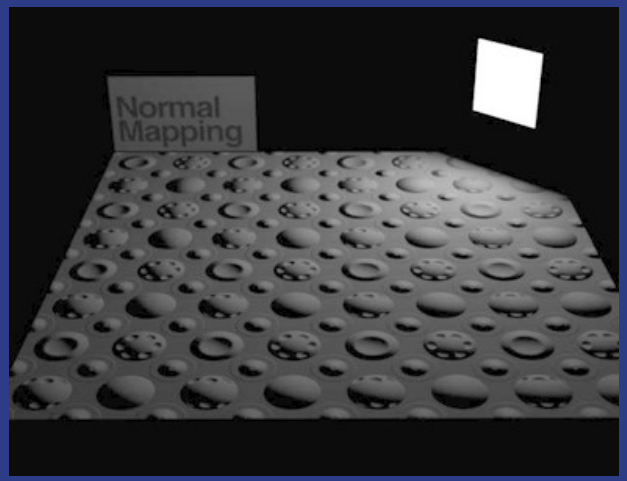
Techniki cieniowania

- **Cieniowanie Phonga** (od nazwiska *Bui Thuong Phong*)
  - Jako że **obliczamy wpływ światła osobno dla każdego fragmentu powierzchni**, możemy dokładniej wyliczyć np. *spotlight* i komponent *specular*





Techniki cieniowania

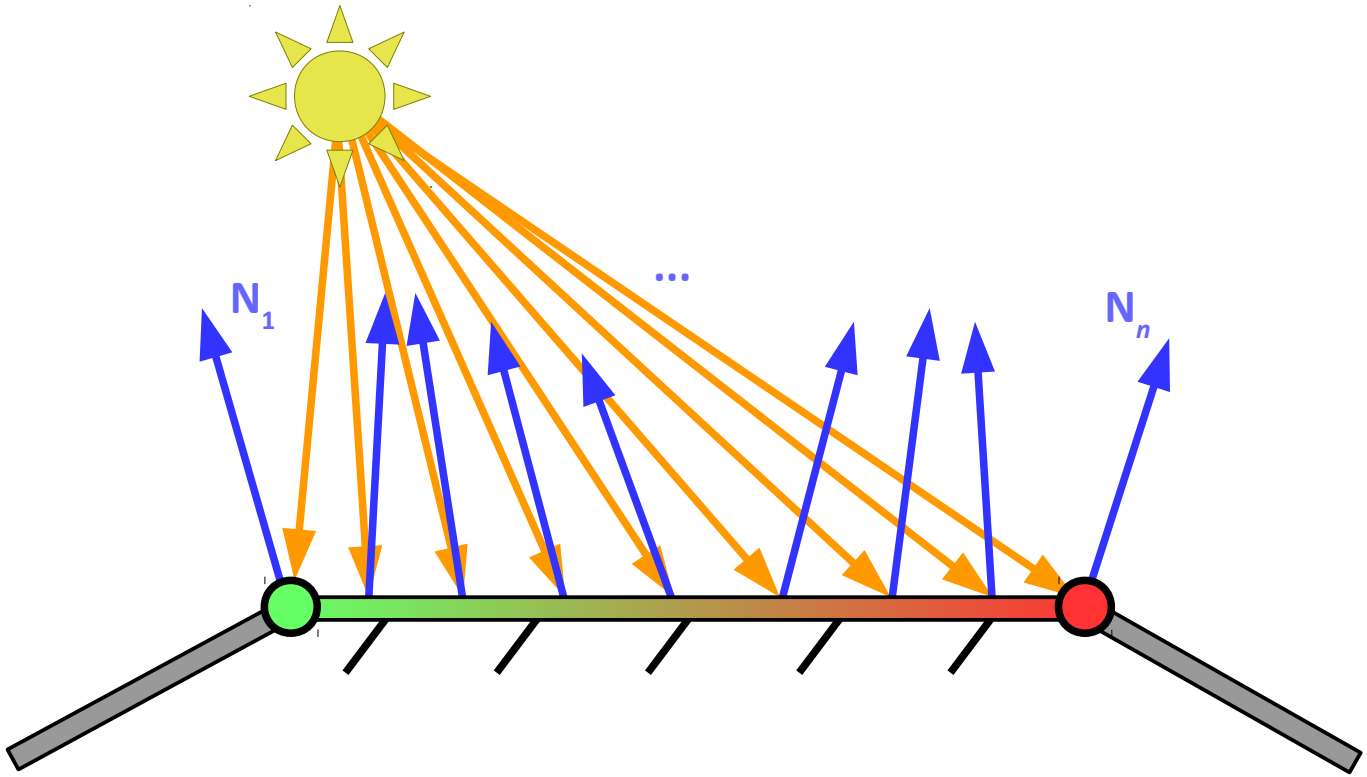


Podłoga jest prostokątem złożonym tylko z czterech wierzchołków.

Kiedy światło się przemieści, nierówności będą inaczej oświetlone.

Źródło obrazu:
http://nextlimit.com

- A gdyby móc kontrolować wektory normalne pomiędzy wierzchołkami?
  - Podejście nazywa się **mapowaniem wektorów normalnych** (ang. *normal mapping*) i wykorzystuje specjalne **tekstury**
  - Pozwala na **oświetlenie powierzchni płaskiej tak, jakby nie była płaska** – oszczędność geometrii przy zadowalającym efekcie wizualnym
  - Technika **mapowania nierówności** (ang. *bump mapping*)



Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Oświetlenie

Potok renderowania. Techniki oświetlenia i cieniowania.

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok