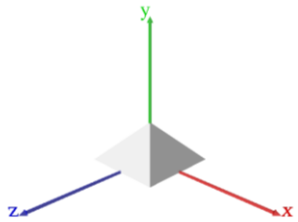
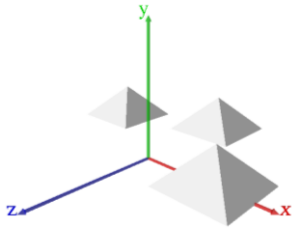
GRK 5

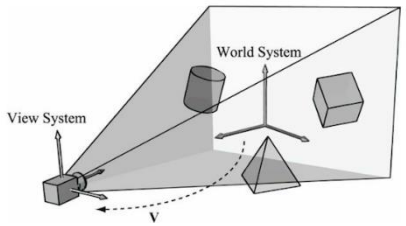
dr Wojciech Palubicki

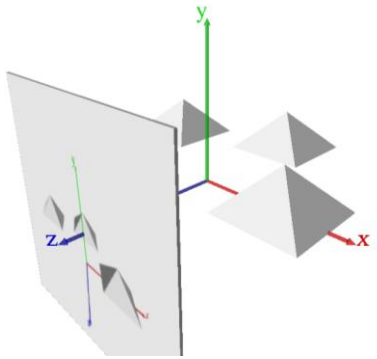
Uproszczony Potok Graficzny (Rendering)

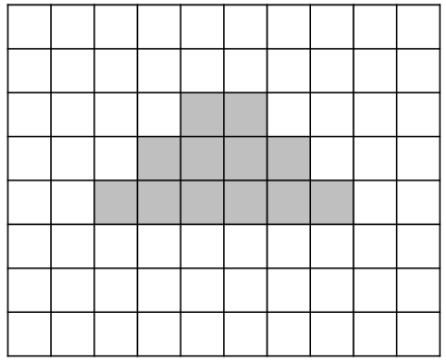
Model Matrix → View Matrix → Projection Matrix → Viewport Transform





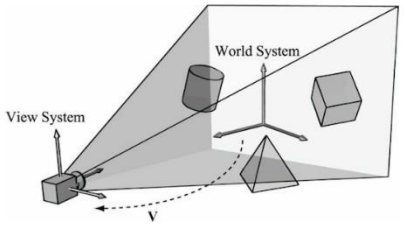


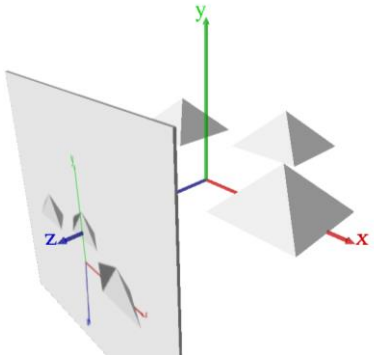




Object Space | World Space | View Space | Clip Space | Screen Space

Projection Matrix

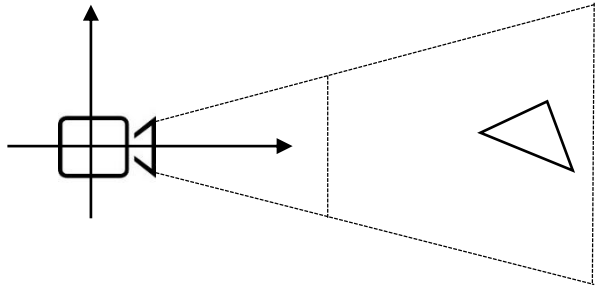




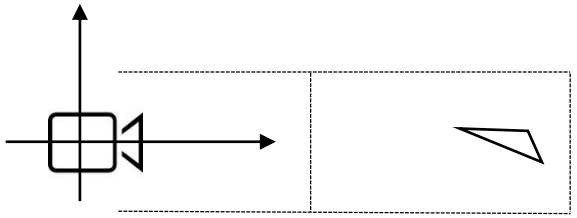
View Space

Clip Space

Potok graficzny

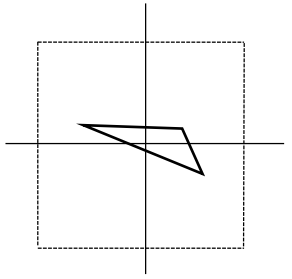


View Space

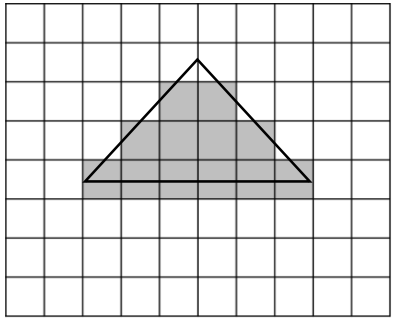


Orthographic view

Canonical view

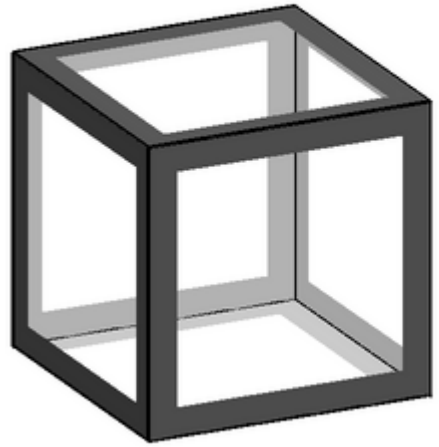


Window space



Rzutowania

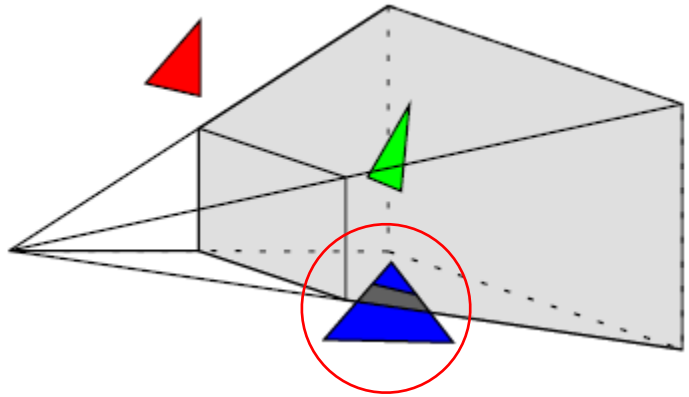
Perspektywicznie rzutowanie

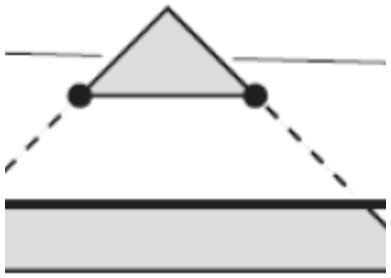


Ortograficzne rzutowanie

Clipping

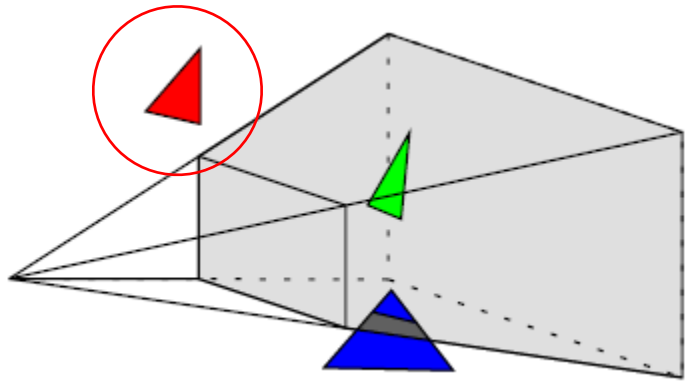
- Żeby zdecydować czy pociąć trójkąt musimy:
  - Sprawdzić czy on przecina **hiperpłaszczyznę**
  - Stworzyć **nowy** trójkąt(y)





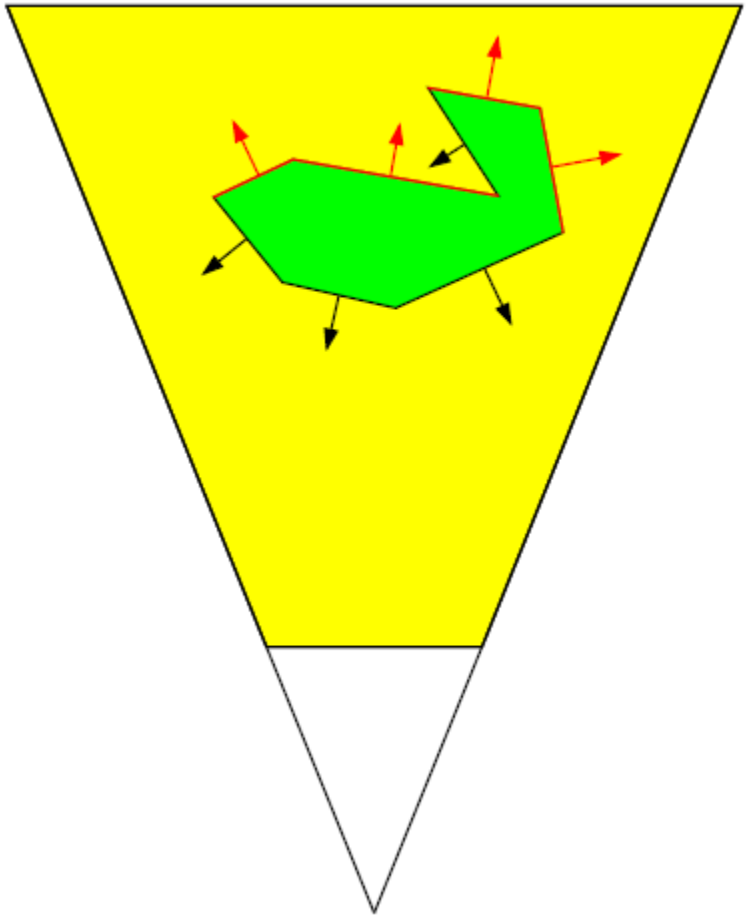
Culling

- Gdy jednak trójkąt leży poza bryłą widzenia usuwamy go całkowicie z dalszych obliczeń
- Testowanie wierzchołków jest ale jednak kosztowne…

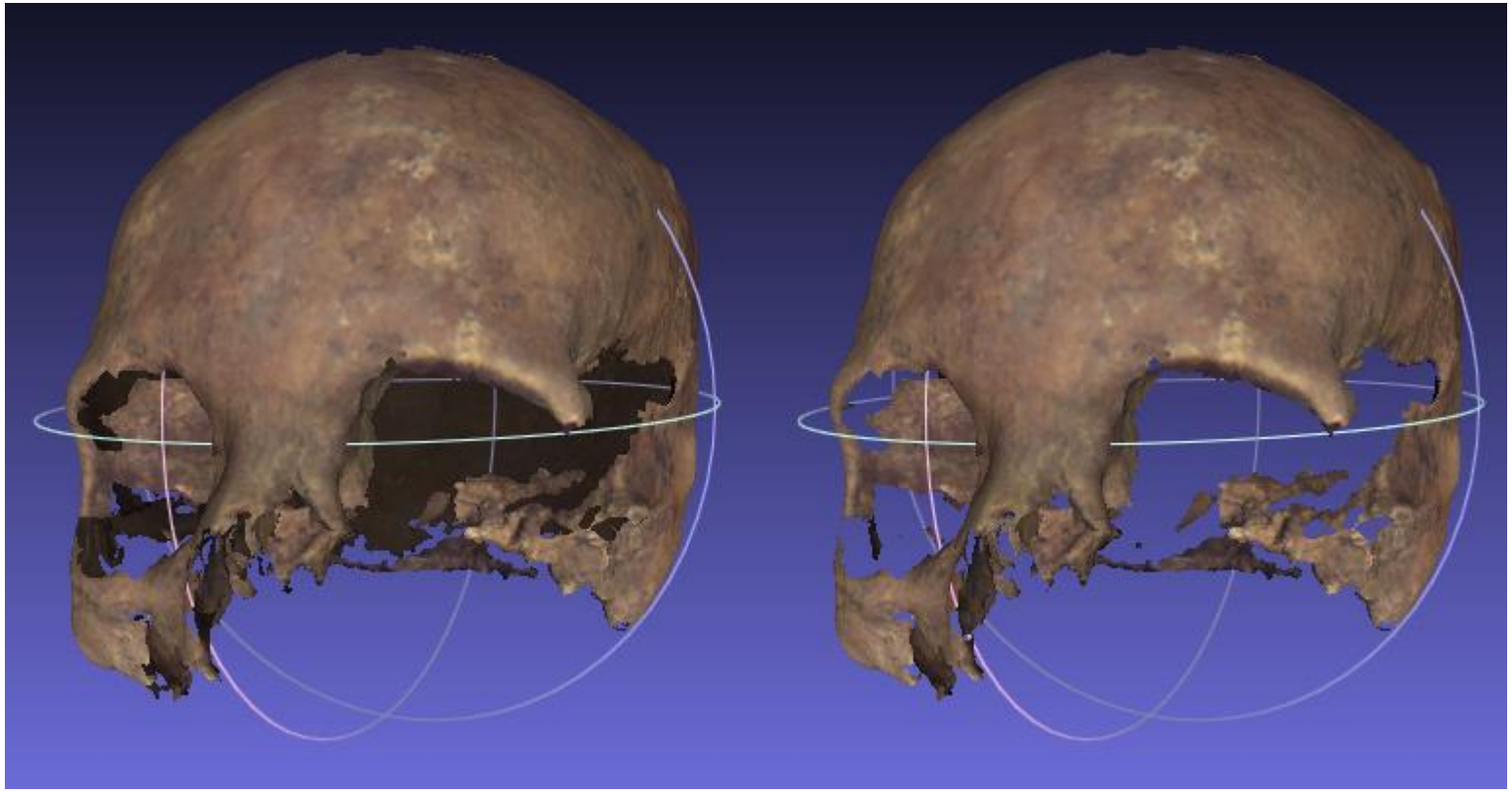


Backface culling

- Gdy modelujemy obiekty geometryczne za pomocy trójkątów to **normalna trójkątów** jest ustawiona na **zewnątrz** obiektu
- Odrzucanie trójkątów których normalna jest skierowana **od punktu widzenia** nazywamy to **backface culling** (usuwanie powierzchni tylnych)

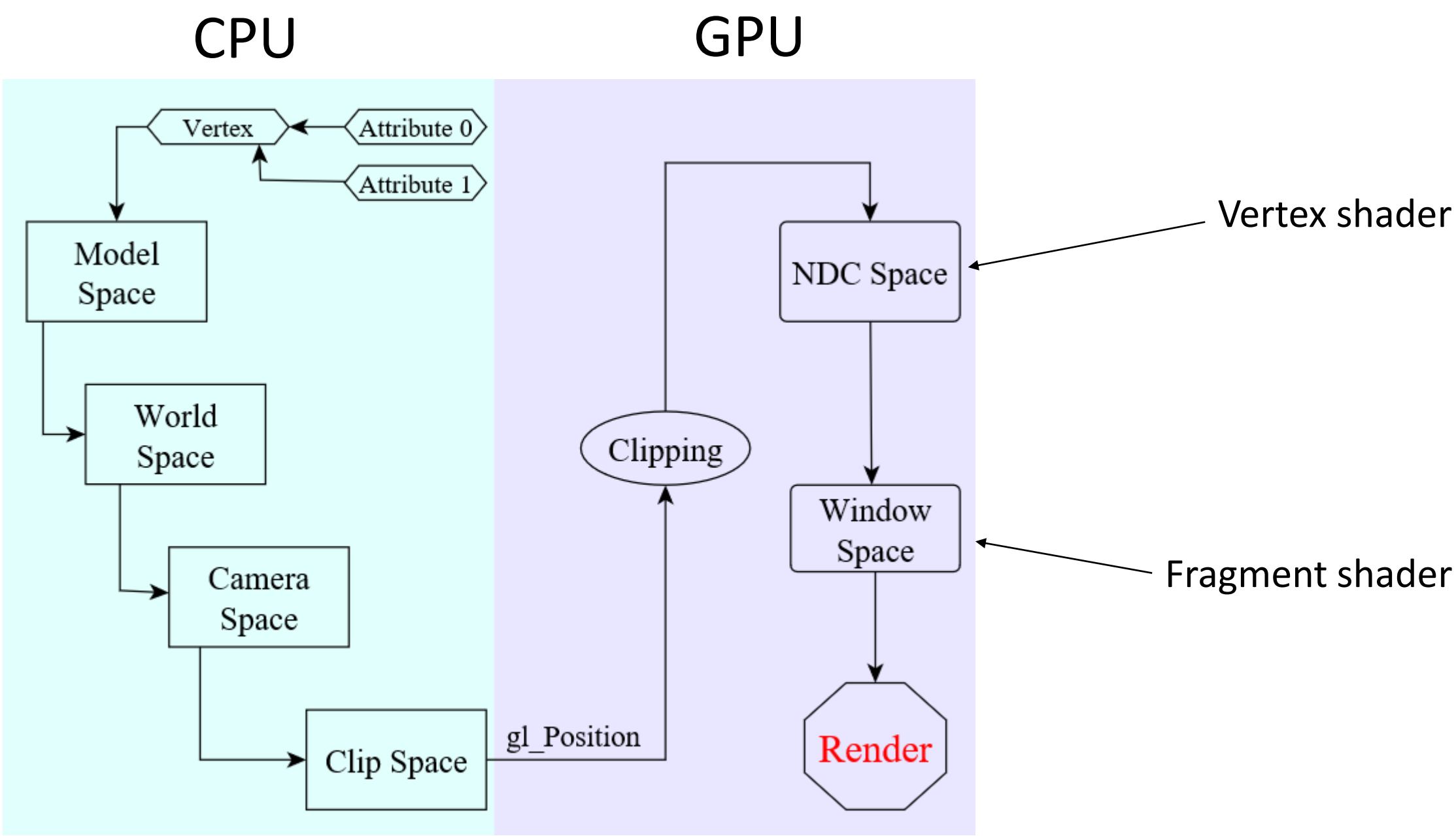


Przykład

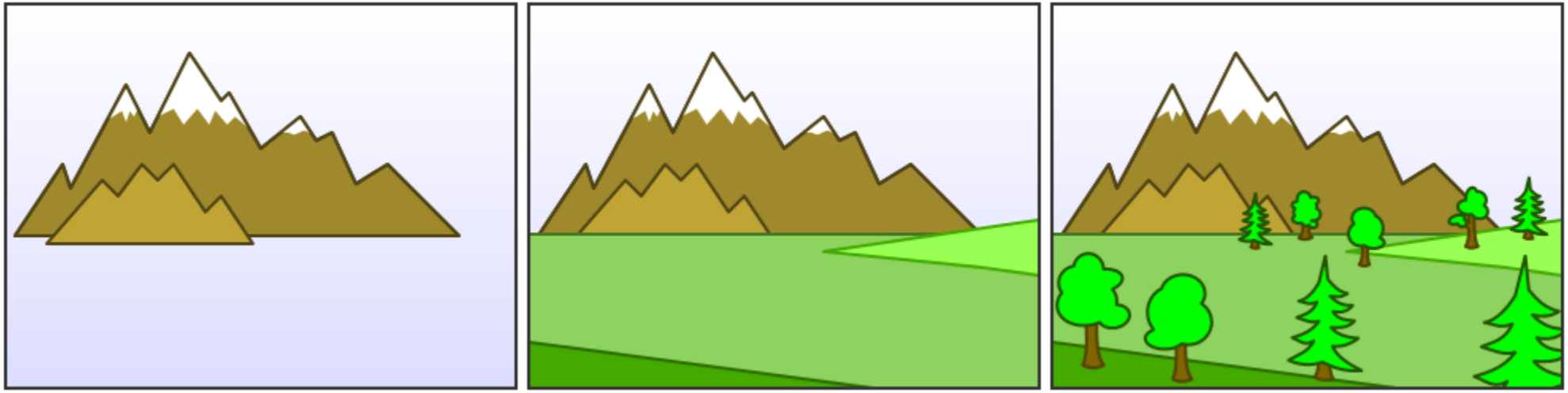


Bez backface culling

Backface culling


CPU
GPU
Vertex
Attribute 0
Attribute 1
Model Space
World Space
Camera Space
Clip Space
gl_Position
Clipping
NDC Space
Window Space
Render
Vertex shader
Fragment shader

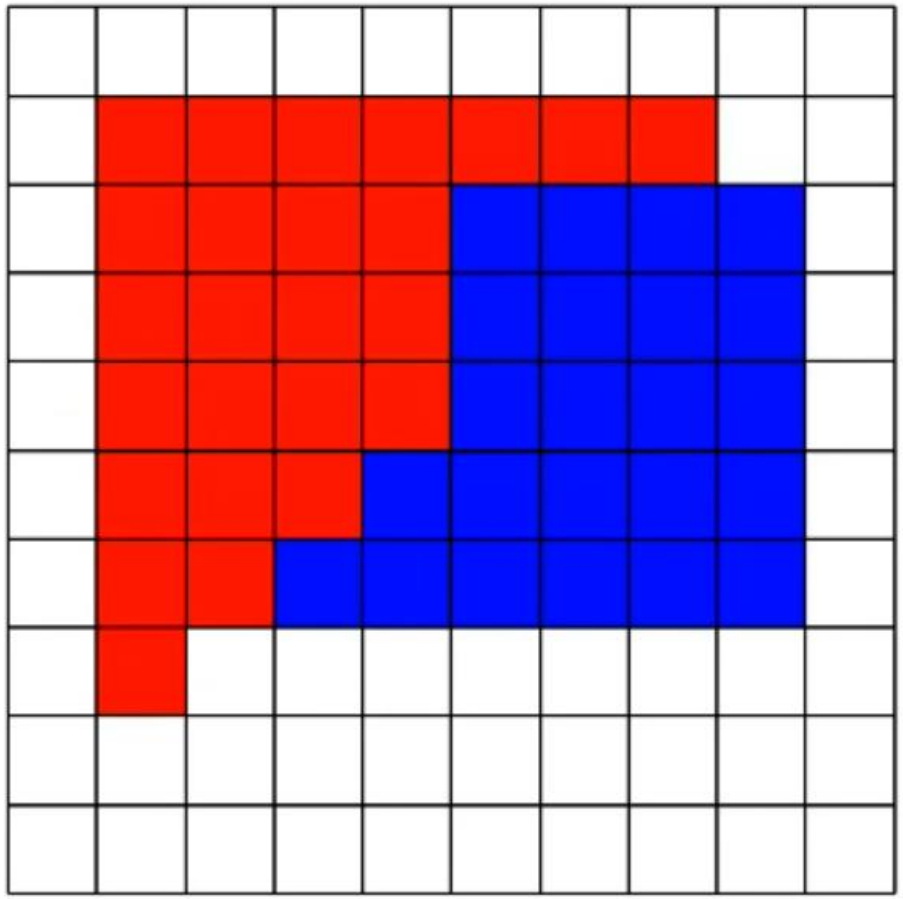
Algorytm malarza (painter's algorithm)



Z-bufor

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	1	2	3	4	5	6	7	∞	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	5	4	3	2	1	∞
∞	1	2	6	5	4	3	2	1	∞
∞	1	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

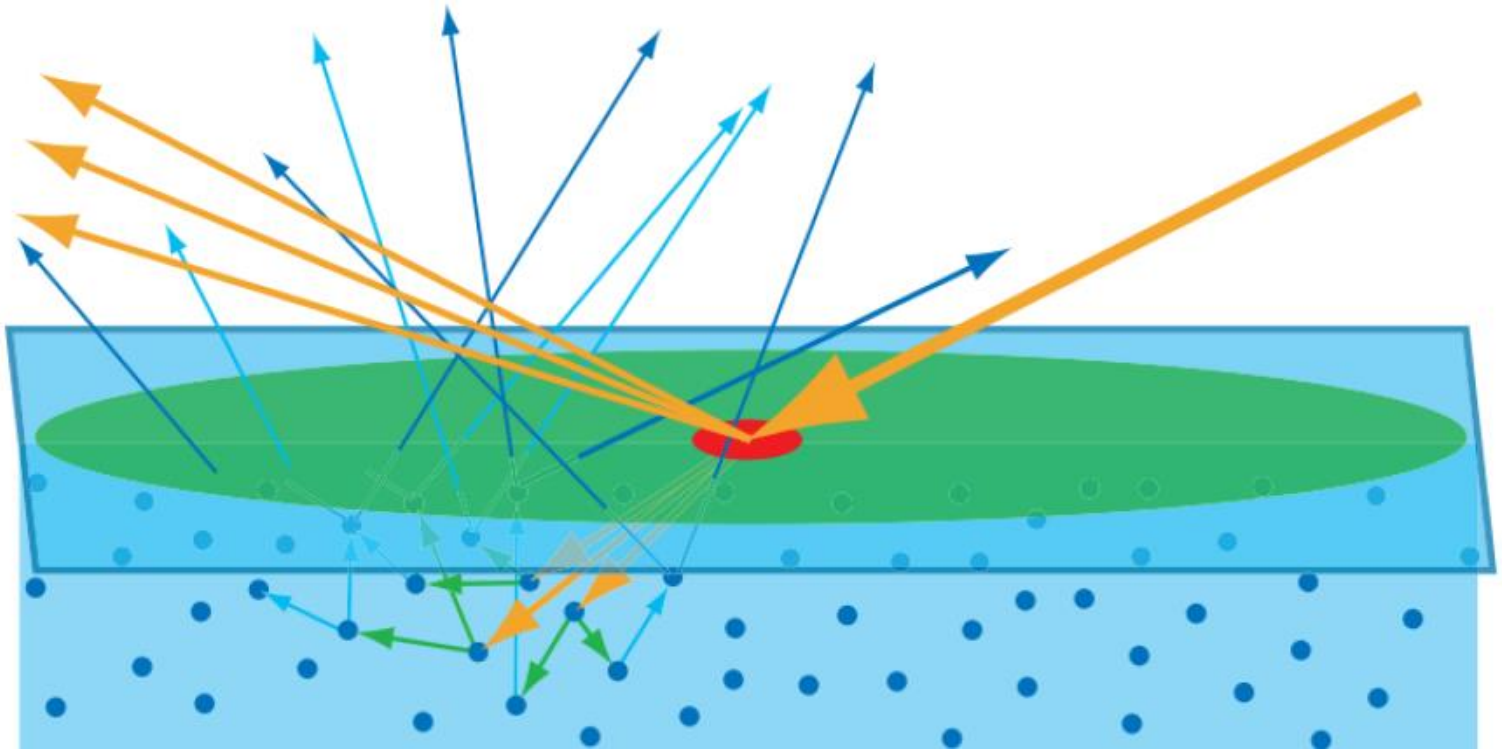
Z buffer



Frame buffer

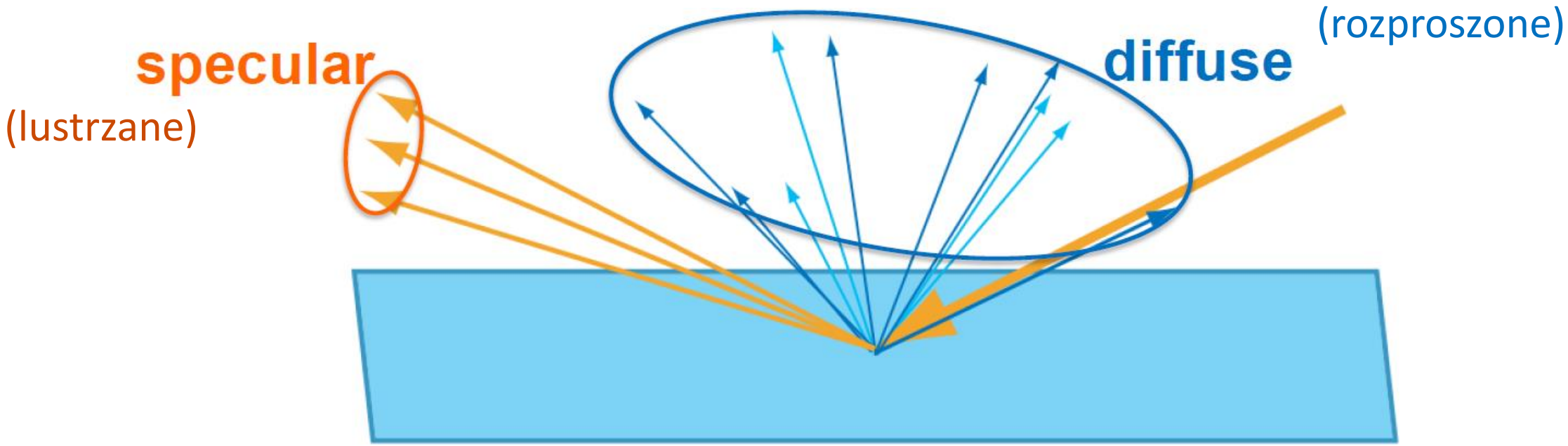
Podpowierzchniowe rozpraszanie (subsurface scattering)

- Dystans pomiędzy punktem wejścia i wyjścia promieni światła jest wyznaczony przez materiał

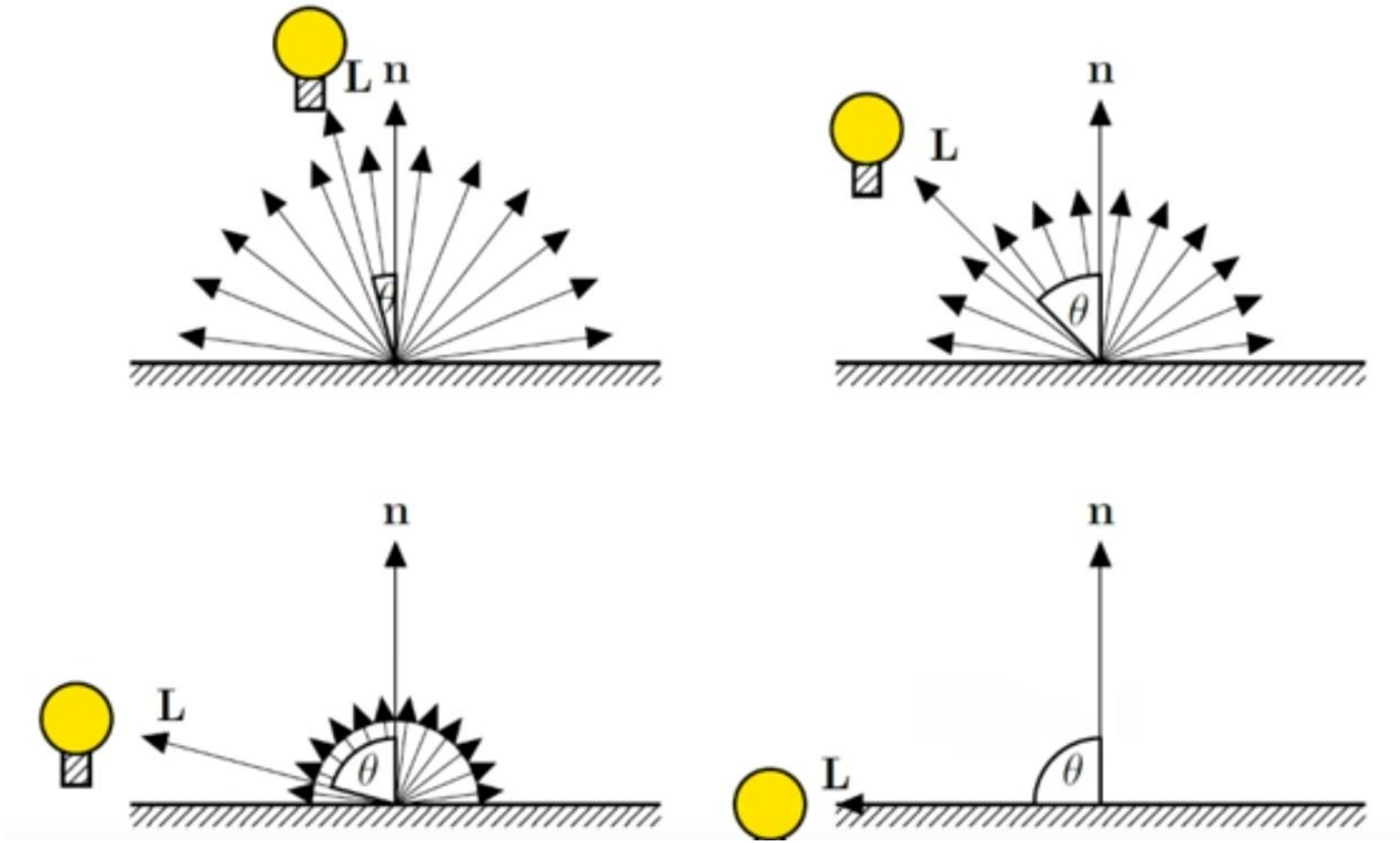


Modelowanie za pomocy BRDF

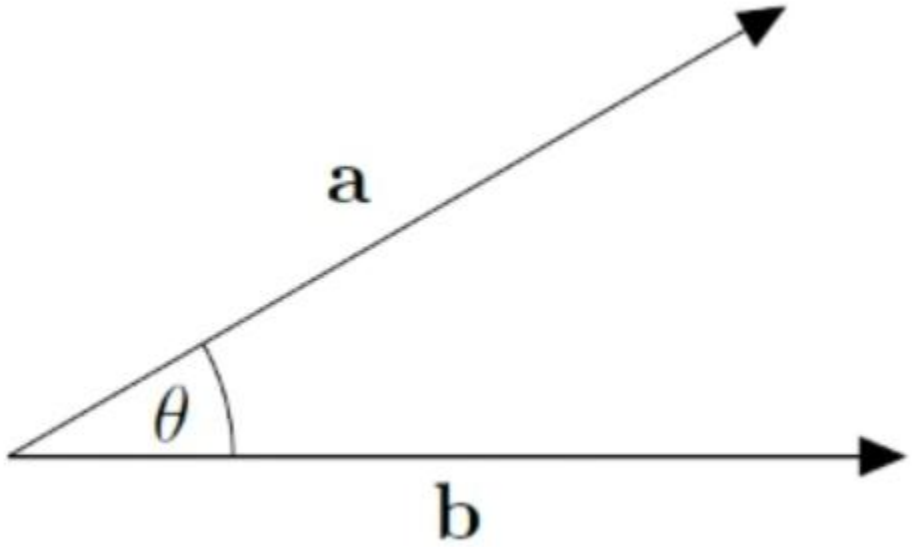
- BRDF to *Bidirectional Reflectance Distribution Function* lub "dwukierunkowa funkcja rozkładu odbicia "
- Modelujemy światło ignorując różnice dystansu punktu wejścia i wyjścia światła



Model Phonga odbicia rozproszonego



- Definicja iloczynu skalarnego to:
  - $a \cdot b = |a|\,|b| \cos(\theta)$
- Ale jak L i n są wektory jednostkowe to:
  - $L \cdot n = \cos(\theta)$
- Czyli kosztowne obliczenie kosinusa da się zastąpić prostym iloczynem skalarnym
  - $\boldsymbol{D = I_p k_d \max[L \cdot n, 0]}$



$k_d = 0.25$

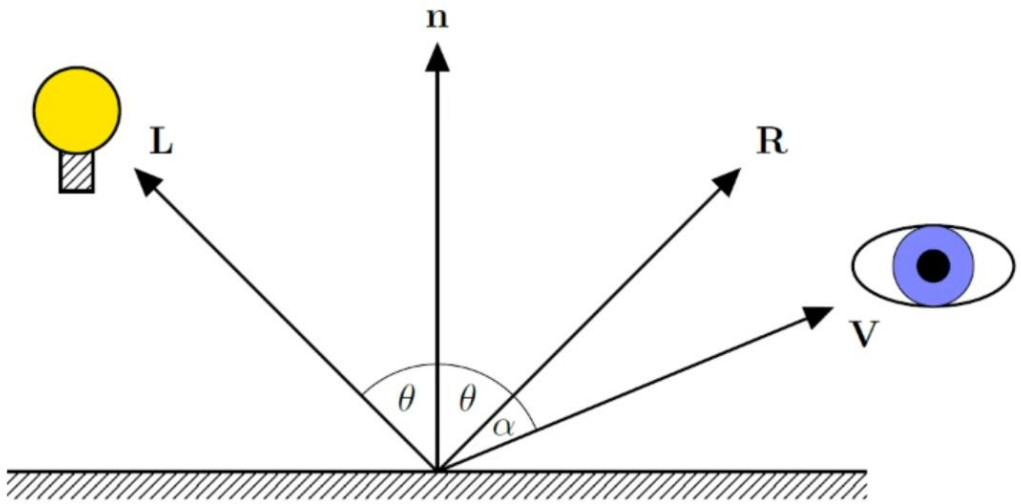
$k_d = 0.5$

$k_d = 0.75$

$k_d = 1$

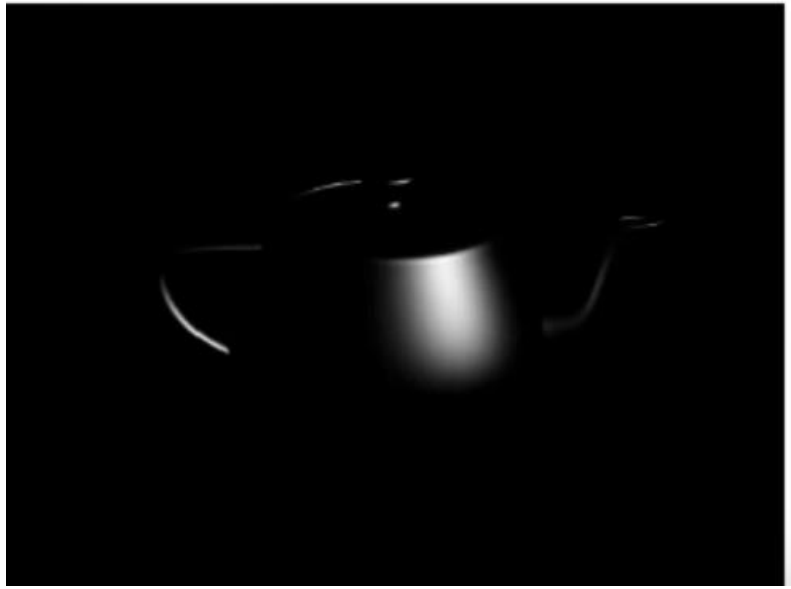
Odbicie geometryczne lustrzane - wektory

- Model Phonga odbicia geometrycznego lustrzanego jest
- $\boldsymbol{S = I_p k_s cos^n(\alpha) = I_p k_s (V \cdot R)^n}$
- Gdzie
  - $k_s \in [0, 1]$ jest współczynnik geometryczny lustrzany
  - n jest wykładnik geometryczny lustrzany
  - α jest kątem pomiędzy R i V
- $cos^n(\alpha)$ ustala ile światła jest odbite



n = 50

n = 20

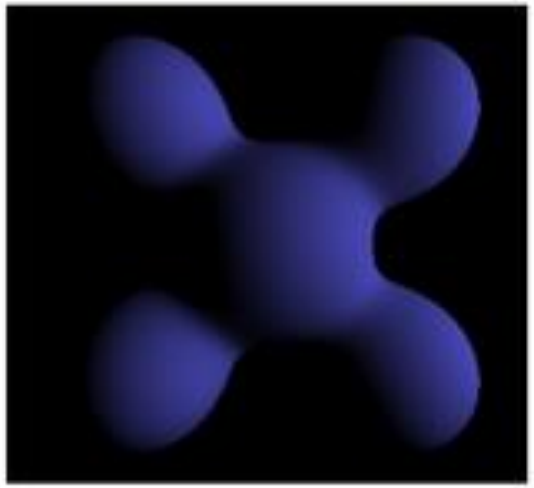


n = 5

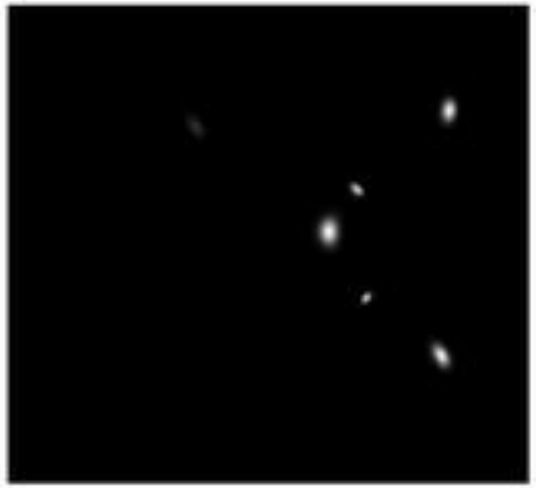
n = 1

Wynik

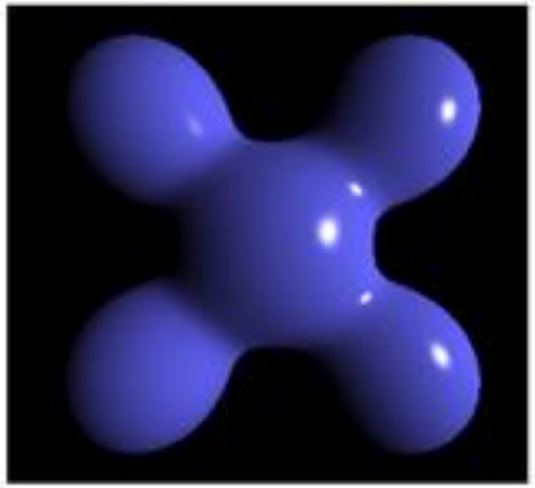
światło otoczenia
(ambient)



światło rozproszone
(diffuse)



światło odbite
geometryczne lustrzane
(specular)

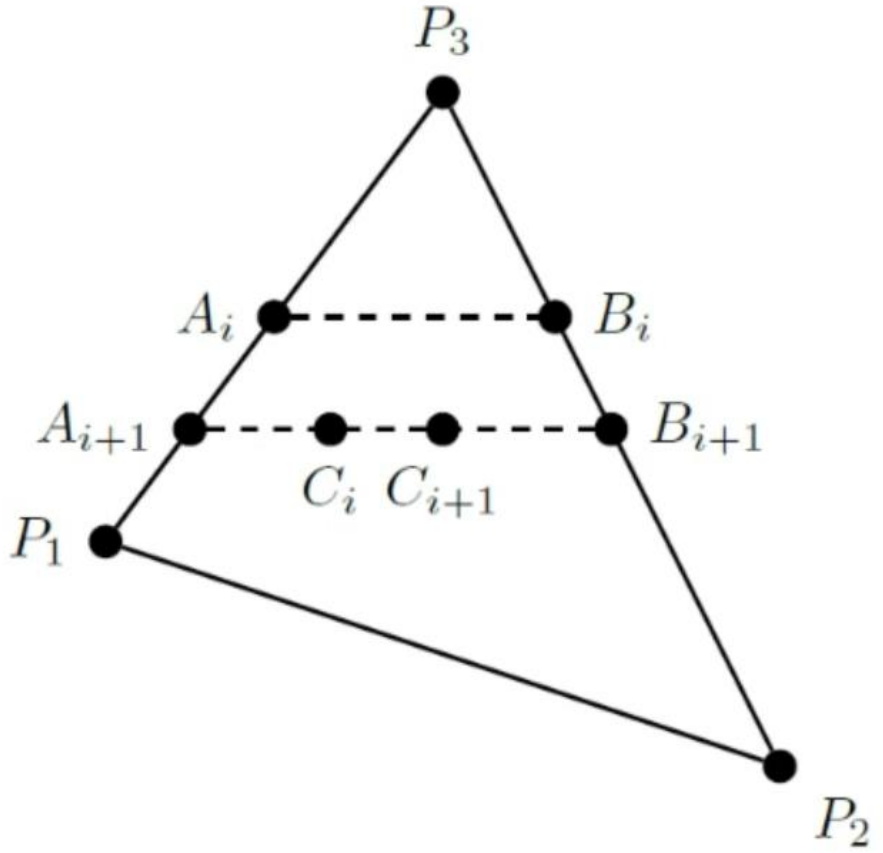


= model odbicia światła Phonga

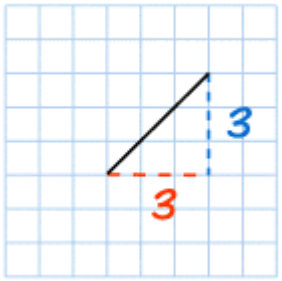
Uśrednianie intensywności

- Gdy *I* jest intensywność oświetlenia w danym punkcie P:
- $I_{A,i+1} = I_{A,i} - \Delta I_A$
- $I_{B,i+1} = I_{B,i} - \Delta I_B$
- $I_{C,i+1} = I_{C,i} - \Delta I_C$

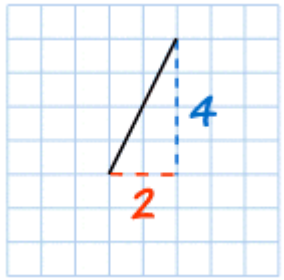
- Gdzie
- $\Delta I_A = \frac{I_3 - I_1}{y_3 - y_1}, \Delta I_B = \frac{I_3 - I_2}{y_3 - y_2}, \Delta I_C = \frac{x_B - x_A}{I_B - I_A}$



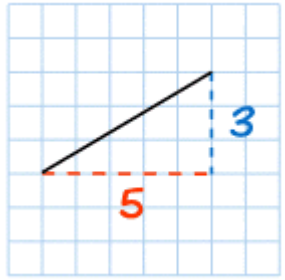
Gradient Δy prostej



$\frac{3}{3} = 1$



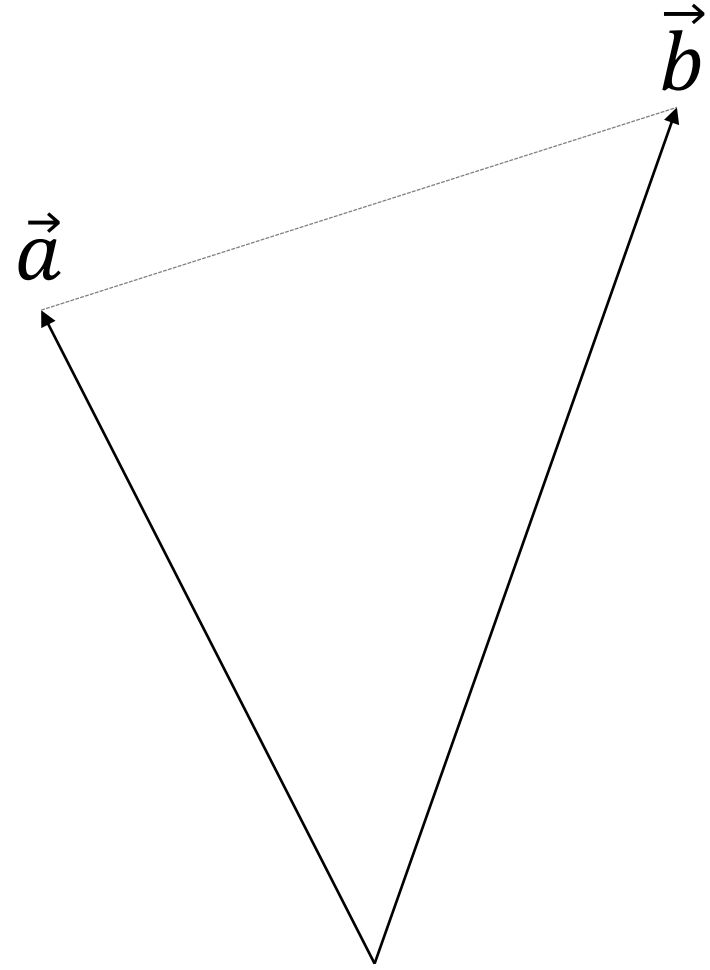
$\frac{4}{2} = 2$



$\frac{3}{5} = 0.6$

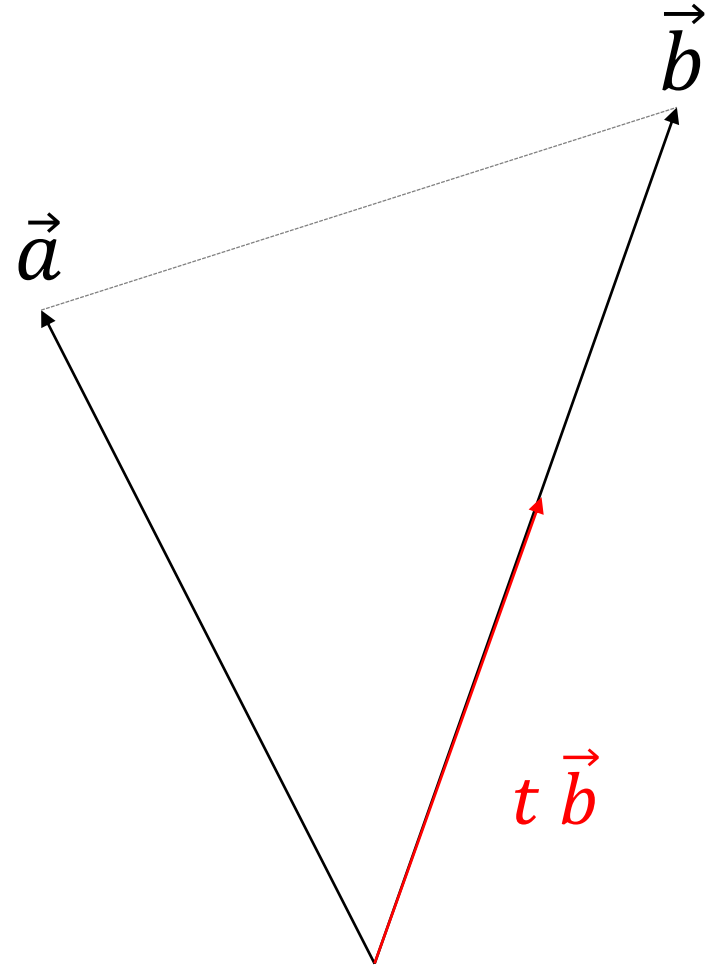
Interpolacja liniowa

- Gdy są dane dwa wektory $\vec{a}$, $\vec{b}$, liniowa interpolacja jest zdefiniowana:
- $\vec{p}(t) = (1-t)\vec{a} + t\vec{b}$
- Gdzie $t \in [0, 1]$



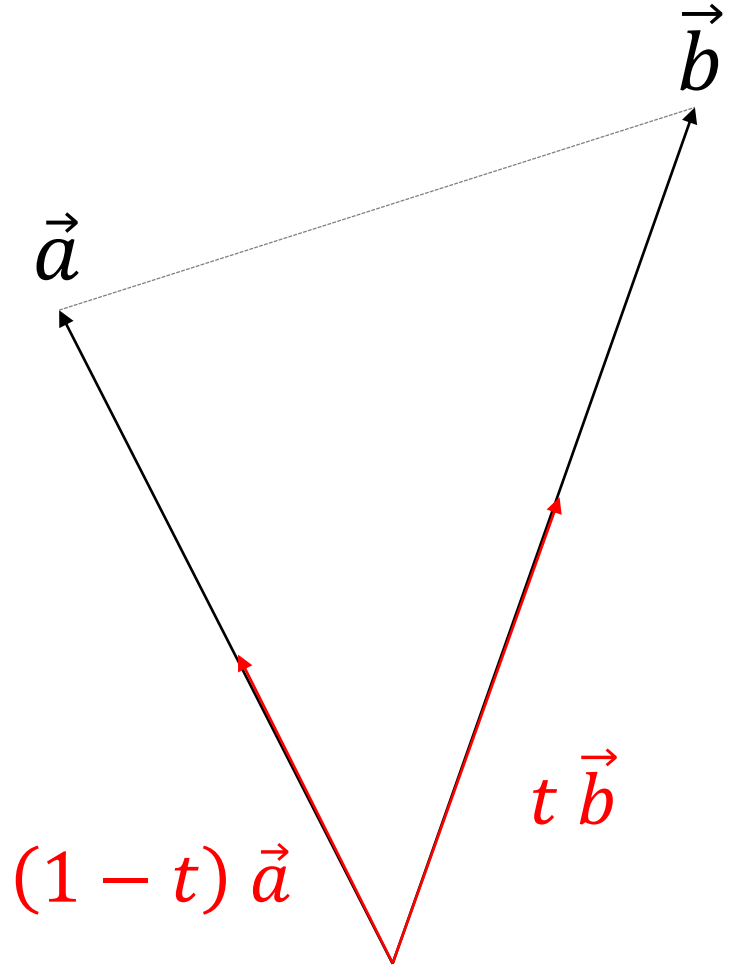
Interpolacja liniowa

- Gdy są dane dwa wektory $\vec{a}$, $\vec{b}$, liniowa interpolacja jest zdefiniowana:
- $\vec{p}(t) = (1-t)\vec{a} + t\vec{b}$
- Gdzie $t \in [0, 1]$



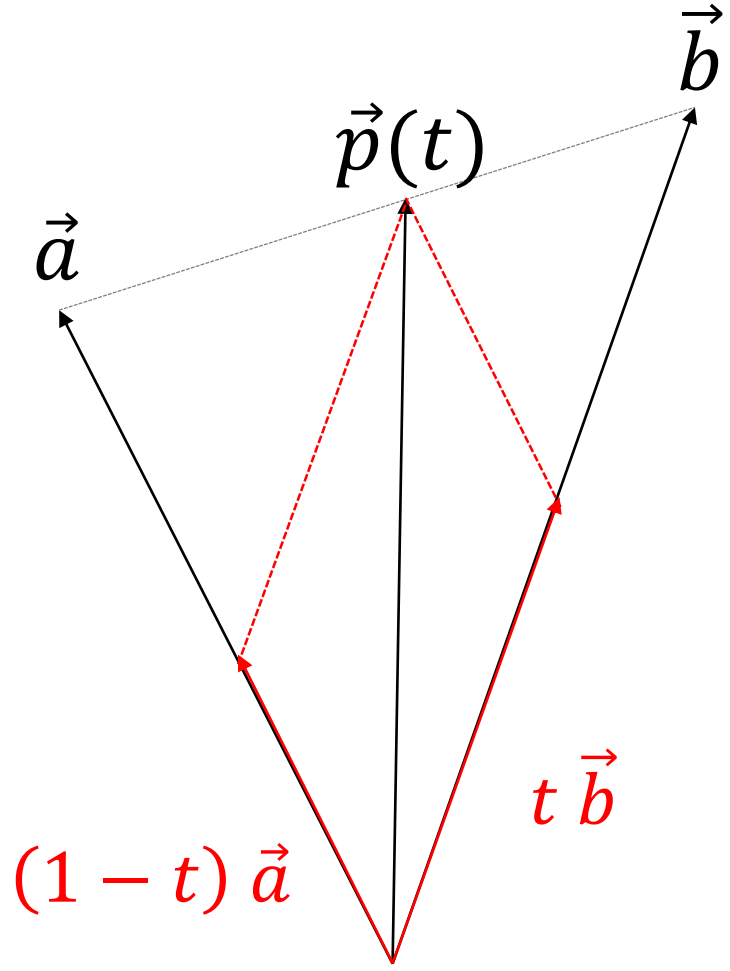
Interpolacja liniowa

- Gdy są dane dwa wektory $\vec{a}$, $\vec{b}$, liniowa interpolacja jest zdefiniowana:
- $\vec{p}(t) = (1-t)\vec{a} + t\vec{b}$
- Gdzie $t \in [0, 1]$



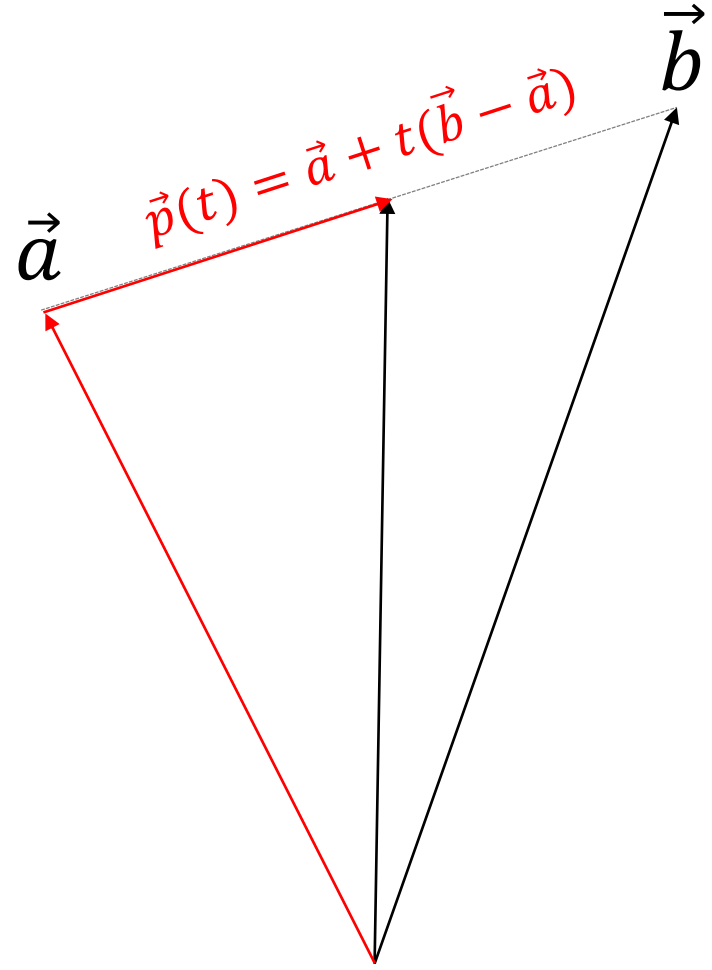
Interpolacja liniowa

- Gdy są dane dwa wektory $\vec{a}$, $\vec{b}$, liniowa interpolacja jest zdefiniowana:
- $\vec{p}(t) = (1-t)\vec{a} + t\vec{b}$
- Gdzie $t \in [0, 1]$



Interpolacja liniowa

- Geometryczna interpretacja:
- $\vec{p}(t) = (1-t)\vec{a} + t\vec{b} = \vec{a} - t\vec{a} + t\vec{b}$
$= \vec{a} + t(\vec{b} - \vec{a})$
- Dla $0 \leq t \leq 1$ daje nam to wszystkie możliwe pozycje pomiędzy $\vec{a}$ i $\vec{b}$

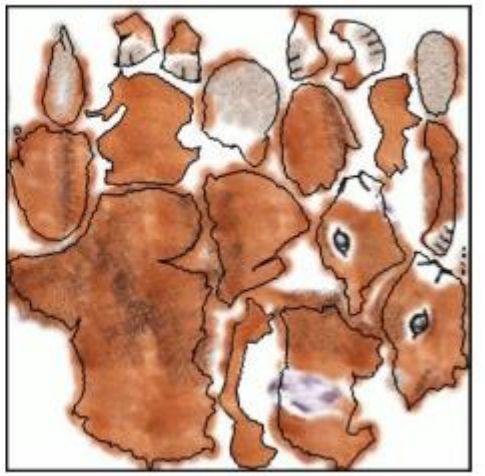


Wariacja kolorów w przestrzeni

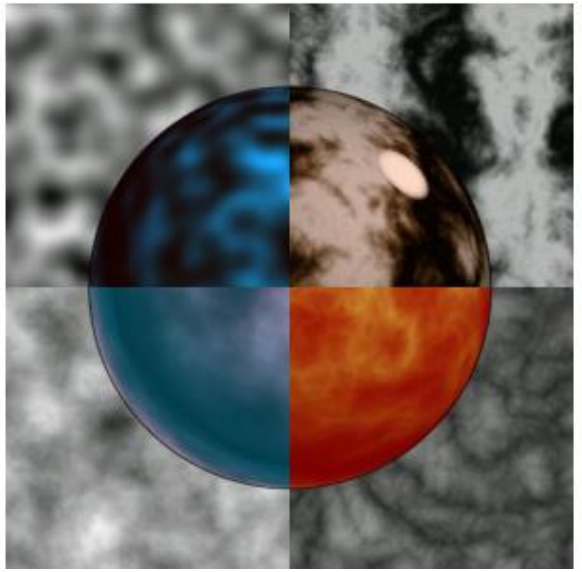
- Kolor światła rozproszonego jest identyczny w każdym punkcie
- Możemy zakładać że kolor k_d rożni się z x

Teksturowanie

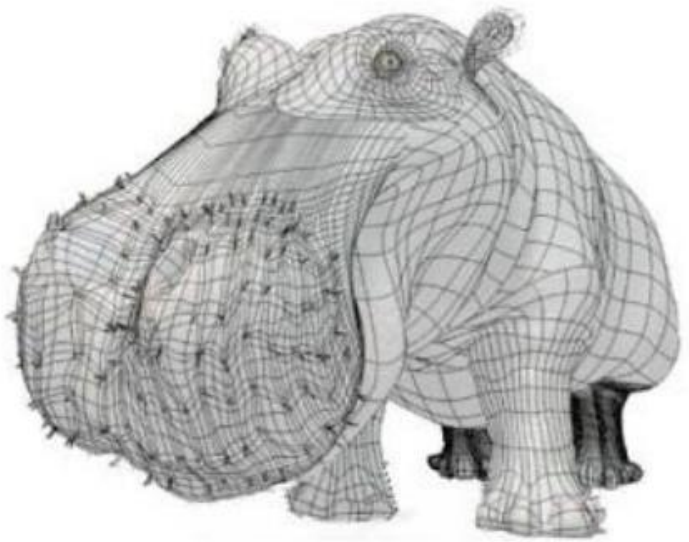
- Z danych:
  - Wczytaj informacje o kolorach z 2D obrazów



- Proceduralnie:
  - Napisać program który oblicza kolor jako funkcja pozycji w przestrzeni

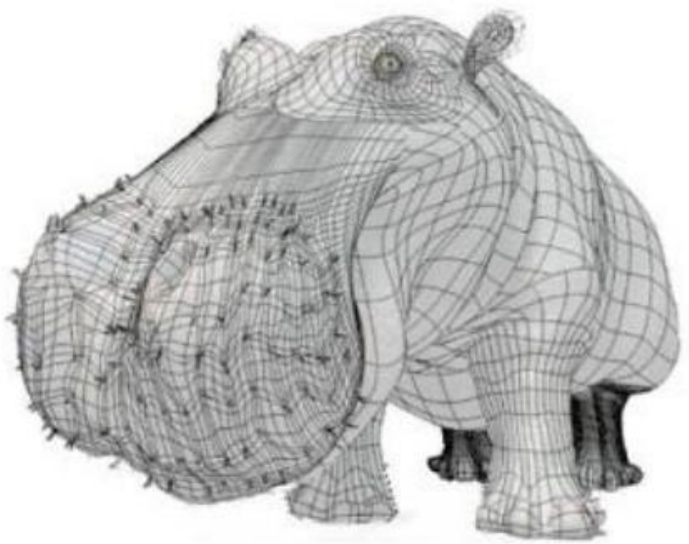


Efekt tekstur



Model

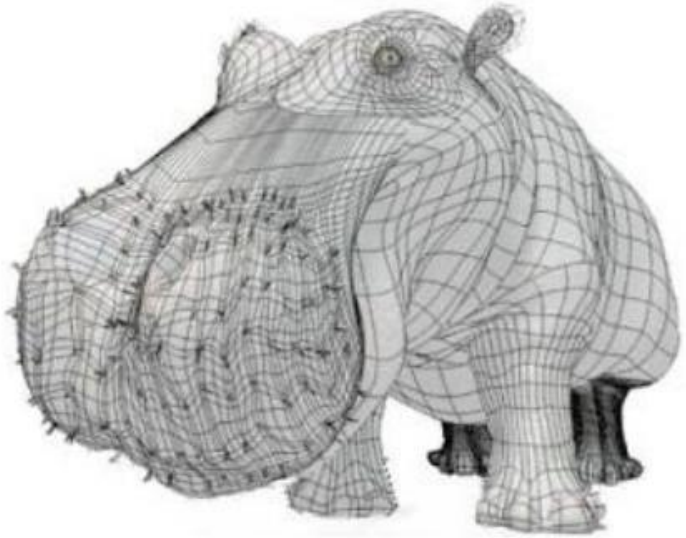
Efekt tekstur



Model

Model + Cieniowanie

Efekt tekstur

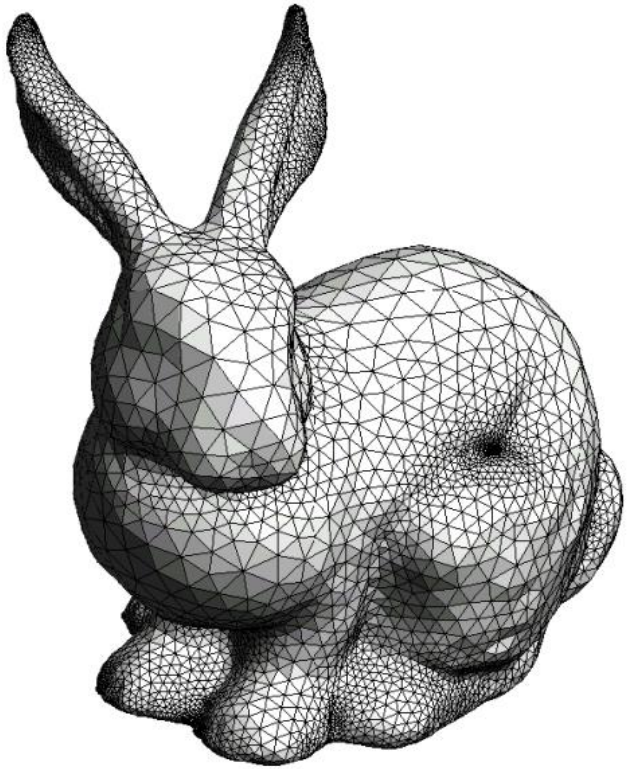


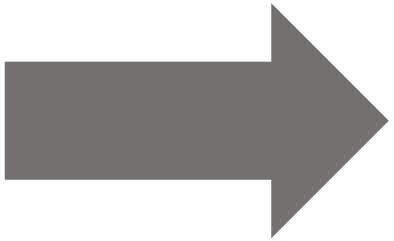
Model

Model + Cieniowanie

Model + Cieniowanie + Tekstury

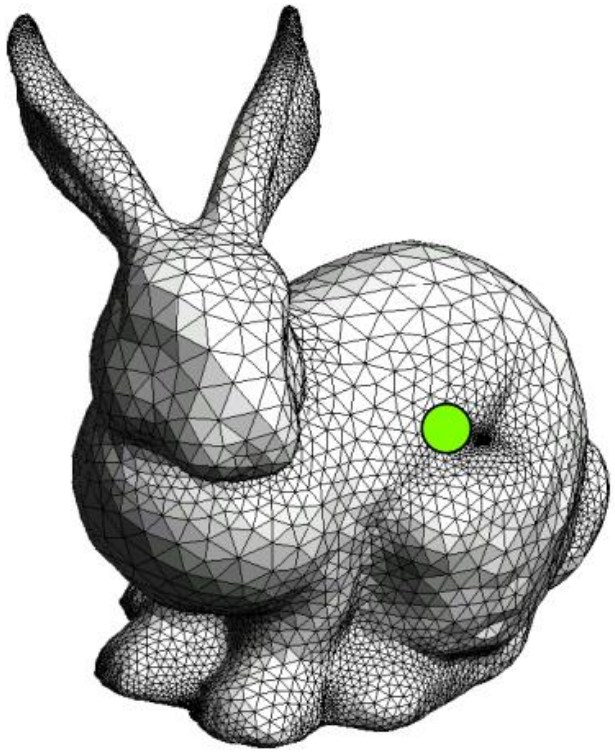
Teksturowanie

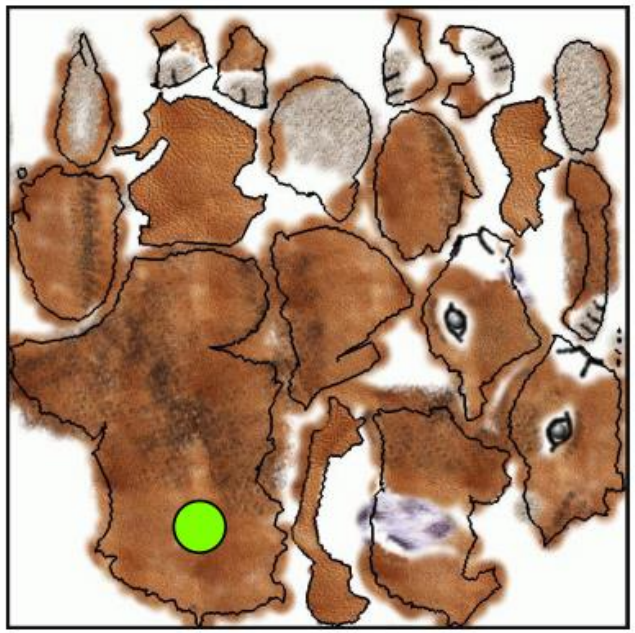




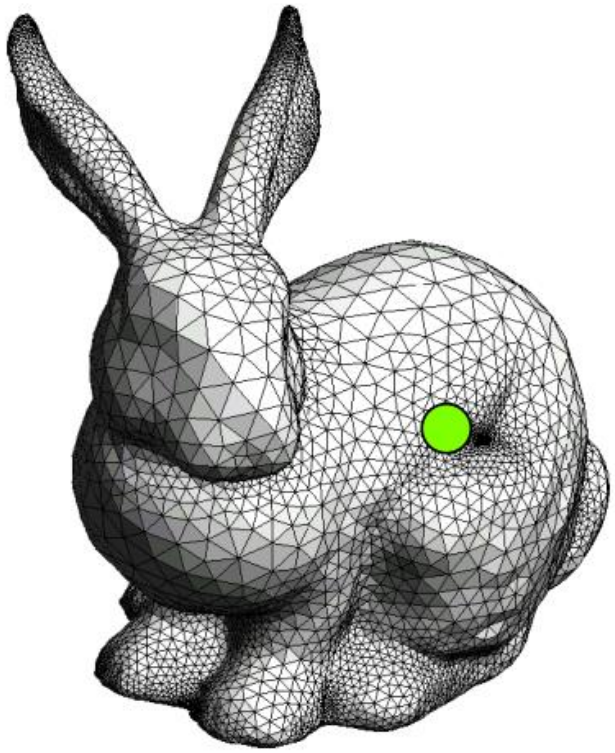
3D model

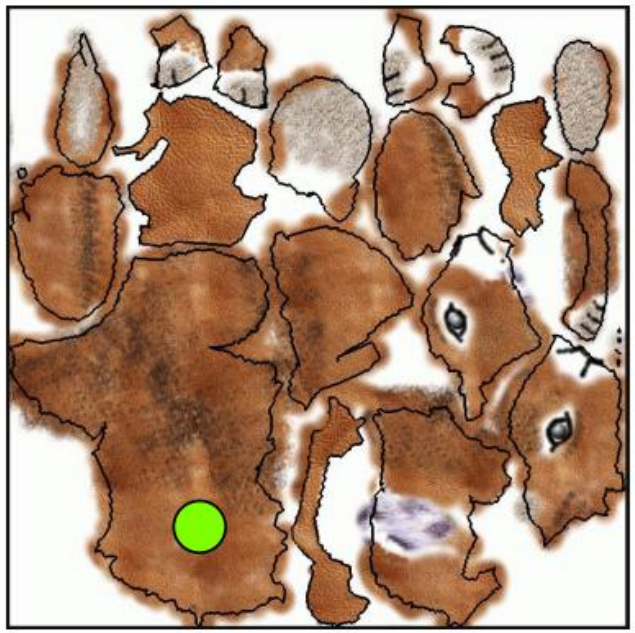
Teksturowany model

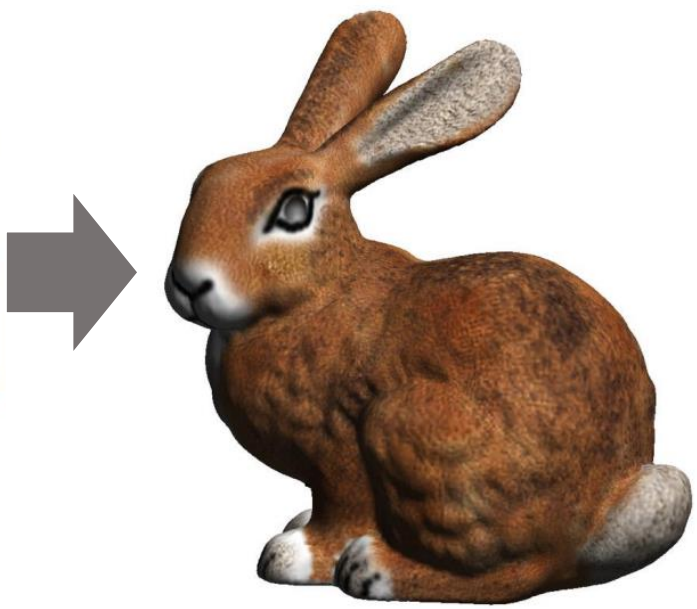




Potrzebujemy funkcje która asocjuje każdy punkt na powierzchni naszego modelu z współrzędną w teksturze



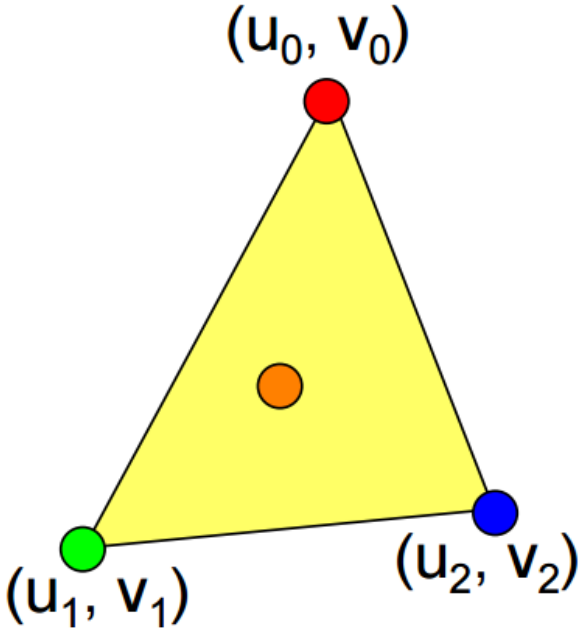


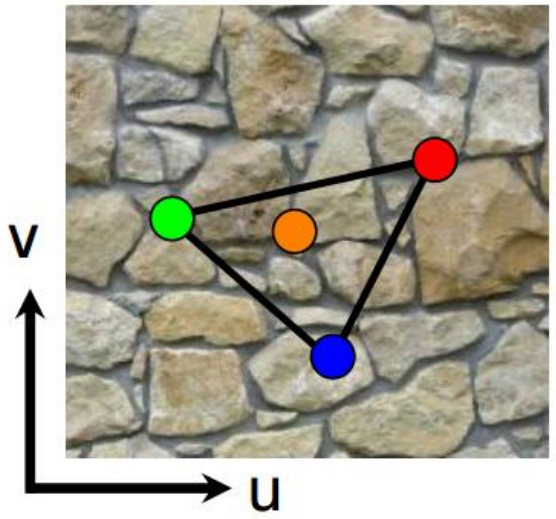


Dla każdego punktu którego rysujemy chcemy odzyskać informacje kolorystyczne z tekstury

Współrzędne u, v

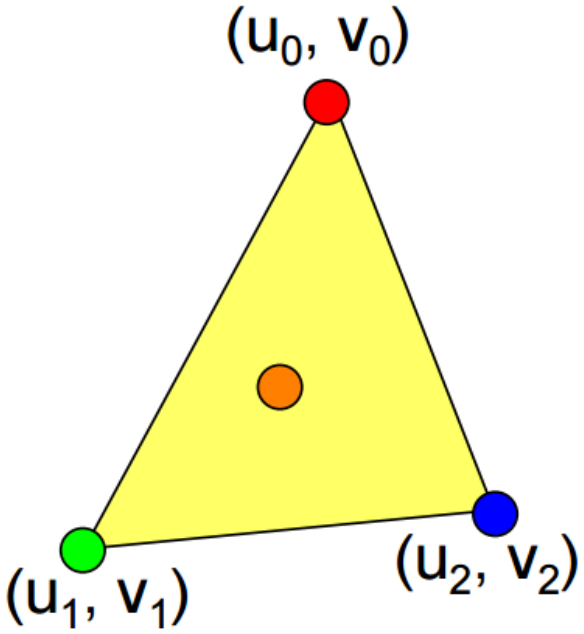
- Każdy wierzchołek zapisuje współrzędne 2D (u, v) tekstury
  - Współrzędne u, v definiują nam pozycje w teksturze każdego wierzchołka
- Informacje kolorystyczne dla pozycji pomiędzy wierzchołkami jest uzyskane za pomocy interpolacji

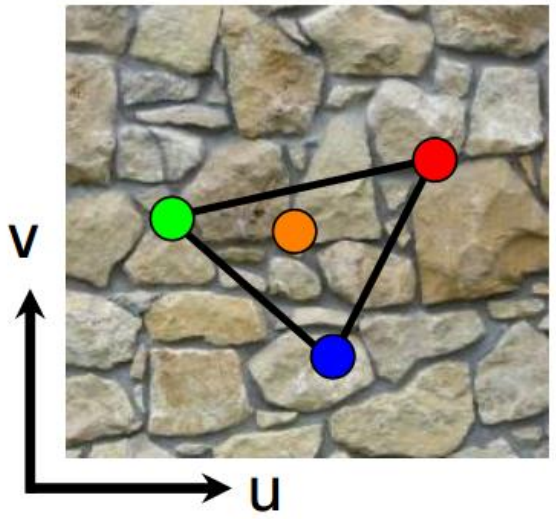




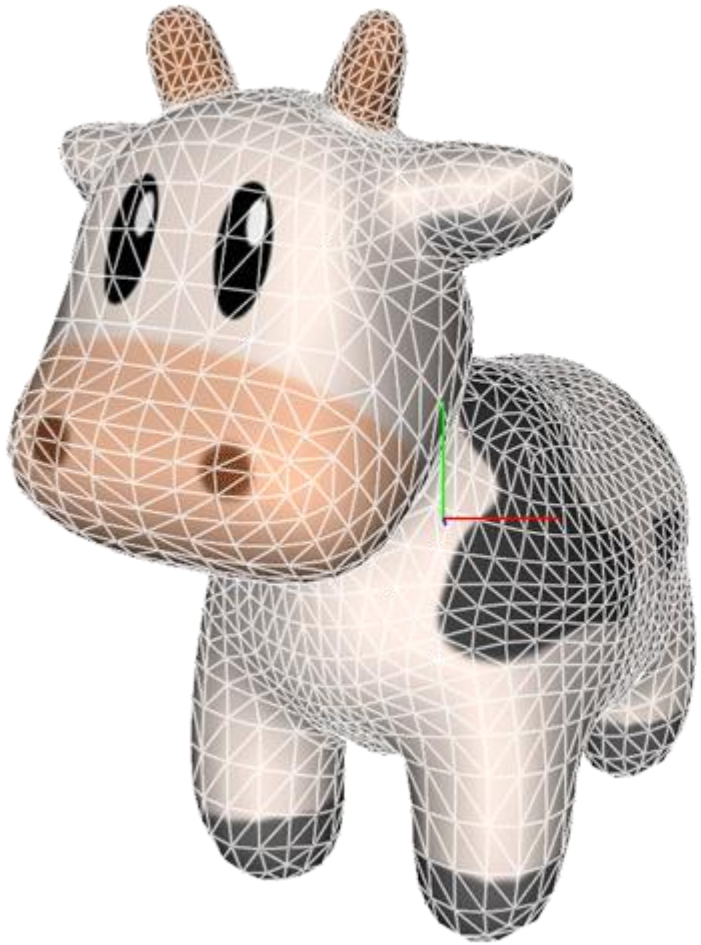
Współrzędne u, v

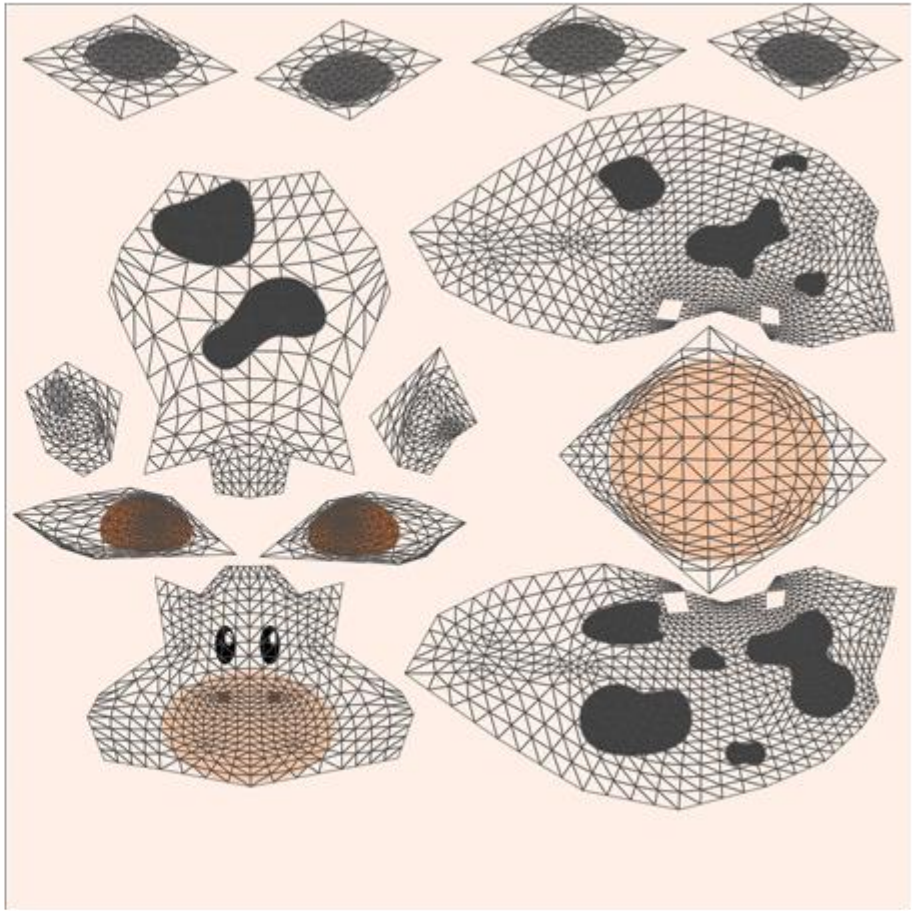
- Współrzędne tekstury u, v są specyfikowane:
  - Manualnie przez użytkownika (pracochłonne)
  - Automatycznie, używając optymizacje parametryczną





Optymizacja parametryczna

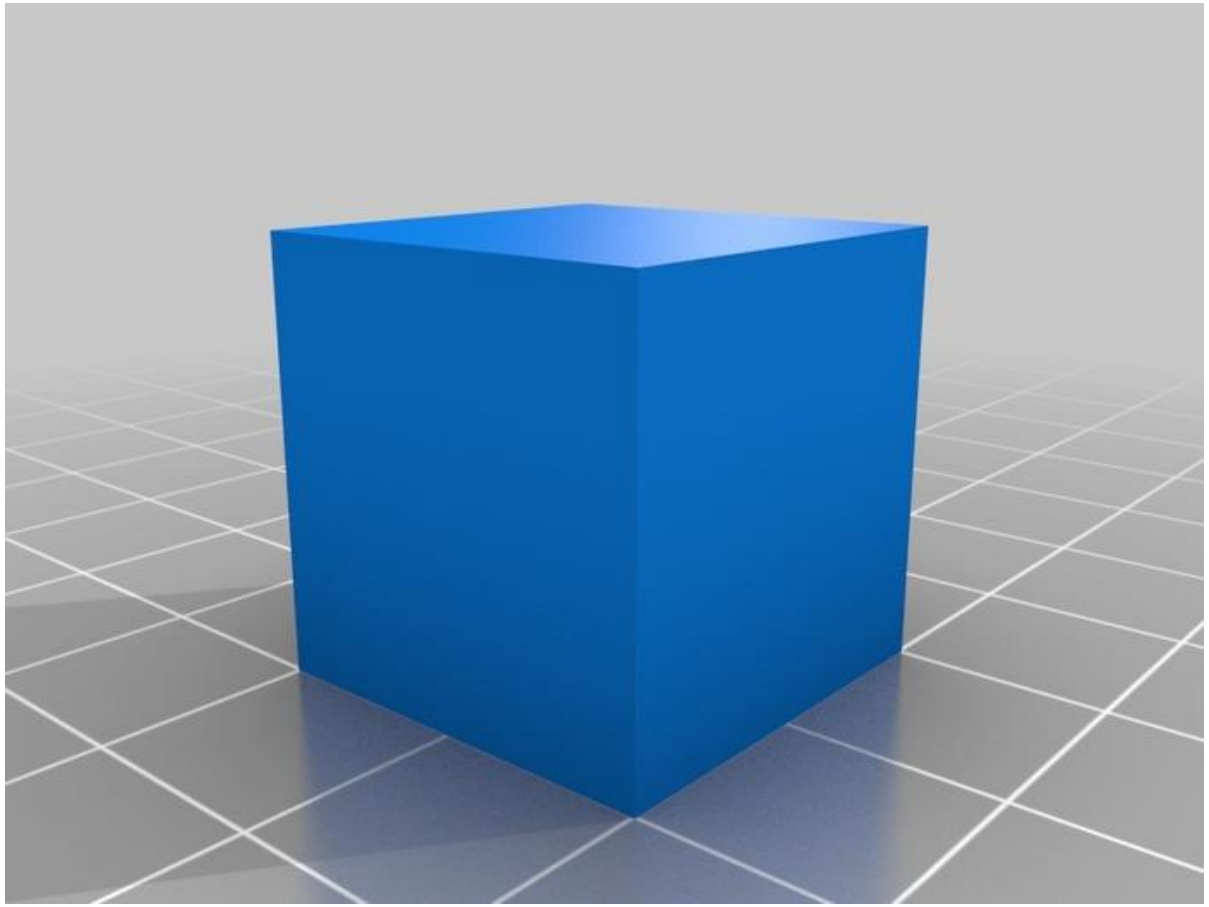




3D model

- Potrzebne dane:
- Dla wierzchołków:
  - Pozycja (3D współrzędne)
  - Normalna
  - 2D u, v współrzędne
- Inne informacje:
  - Parametry i sposób cieniowania
  - Obraz dla naszej tekstury

Wavefront .OBJ plik

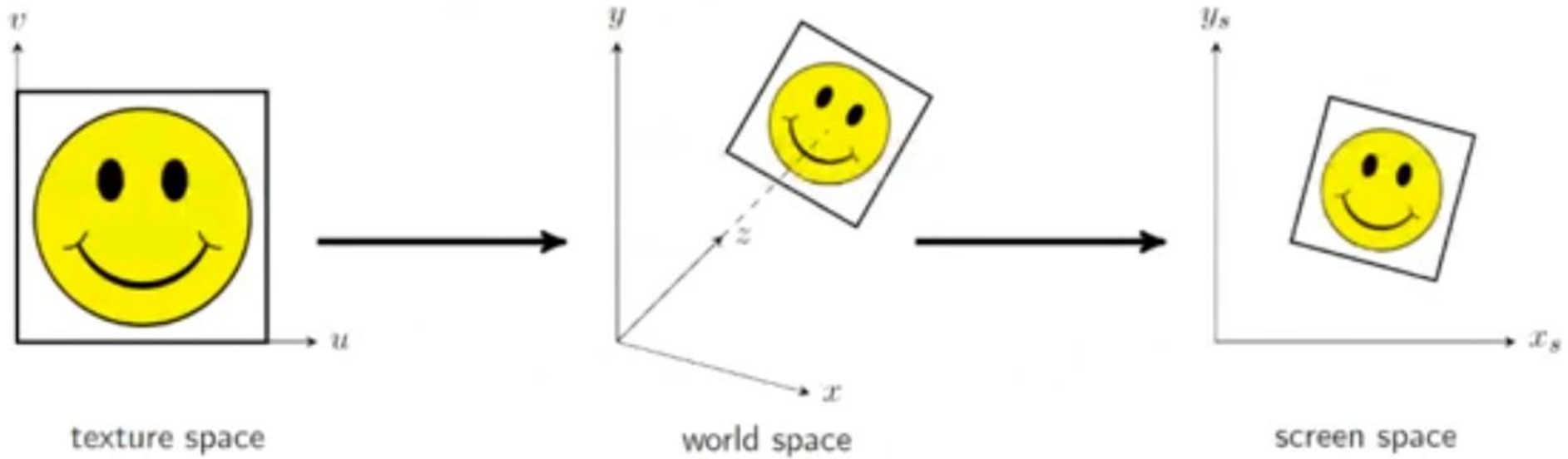


```
# Notes:
# v - vertices in x,y,z coordinates
# vt - texture in u,v coordinates
# vn - normals in nx,ny,nz coordinates
# f - faces: map vertices, UVs, normals

g cube

v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0

vt 0.0 0.0
vt 1.0 0.0
vt 1.0 1.0
vt 0.0 1.0

vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn -1.0 0.0 0.0

f 1/1/2 7/3/2 5/2/2
f 1/1/2 3/4/2 7/3/2
f 1/1/6 4/3/6 3/4/6
f 1/1/6 2/2/6 4/3/6
f 3/1/3 8/3/3 7/4/3
```

Teksturowanie (texture mapping)

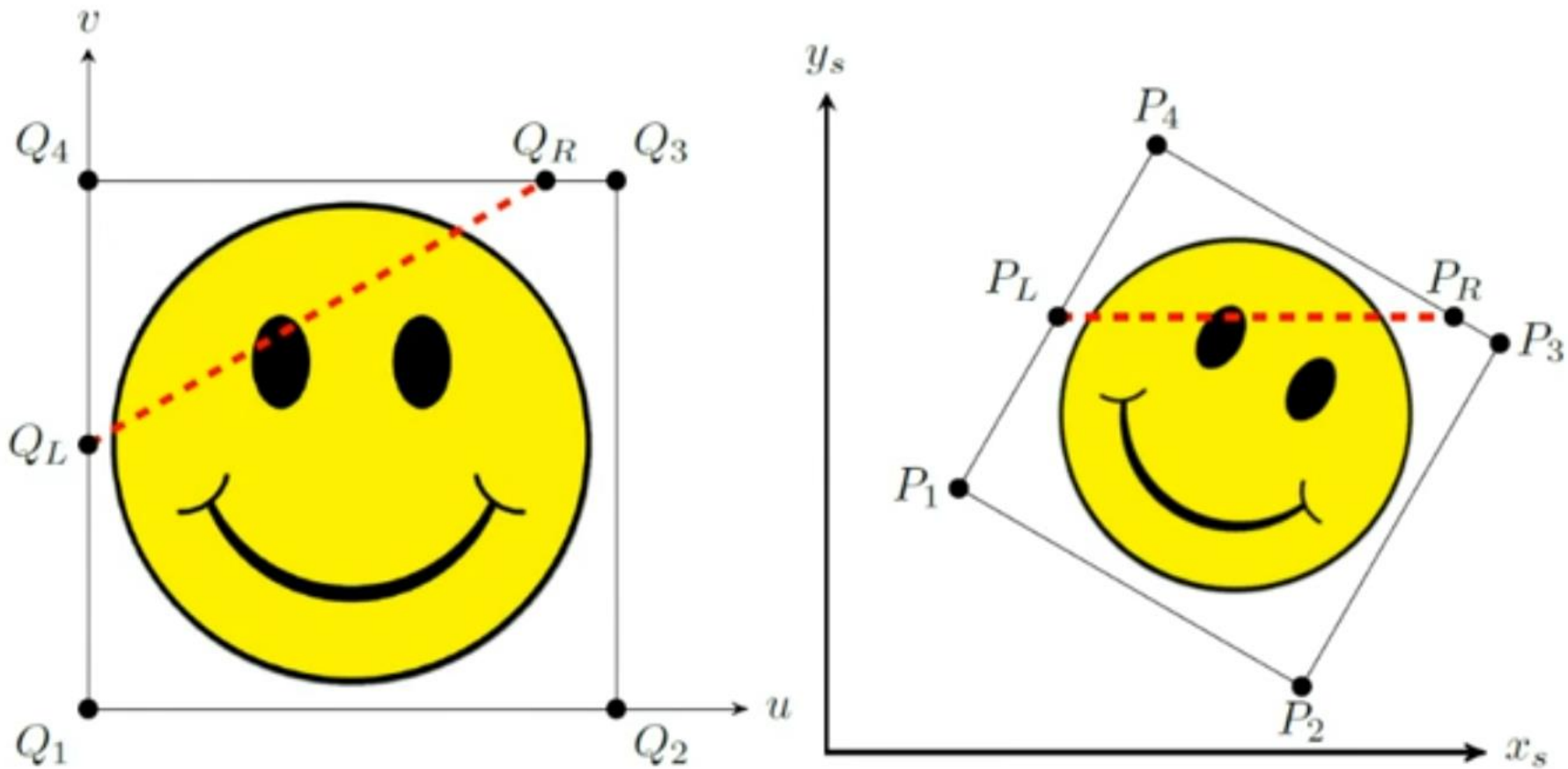
- **Teksturowanie** to zwyczajnie jest procesem odwzorowywania **dwu-wymiarowego** obrazu na **trójwymiarowego** wielokąta
- Obraz który odwzorowujemy nazywamy **teksturą**
- Piksel w teksturze nazywamy **teksel**
- Kolory pikselów są dane przez kolory tekselów w teksturze

Teksturowanie



Przestrzeń tekstury

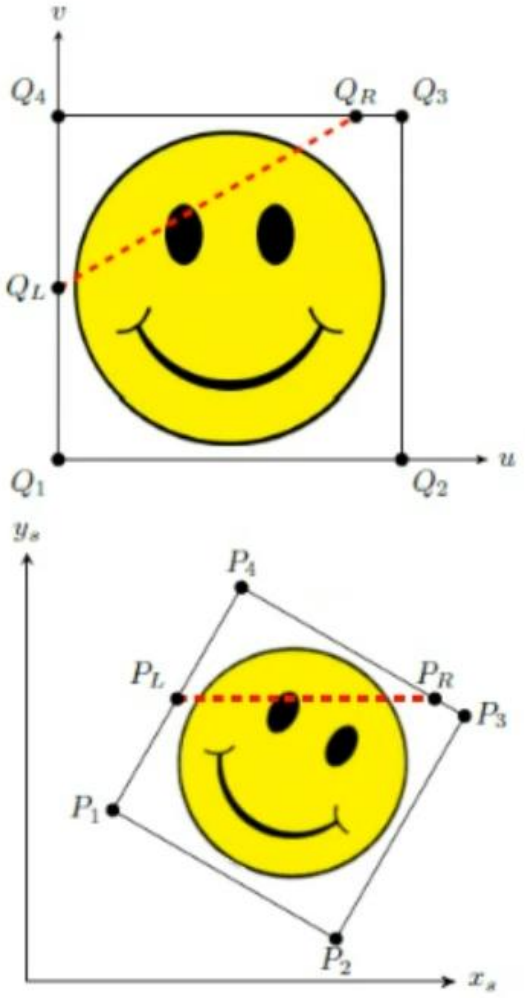
- **Przestrzeń tekstury** to przestrzeń w jakiej tekstura istnieje
- Jest to dwuwymiarowa tekstura gdzie horyzontalne i wertykalne osie są nazywane $u \in [0, 1]$ i $v \in [0, 1]$
- Tekstura wypełnia przestrzeń tekstury, tzn. lewy dolny róg jest w pozycji (0,0) i górny prawy róg jest w (1, 1)
- Współrzędne **tekseli** (U, V) dla M_x x M_y tekstur są
  - $U = \lfloor M_x \cdot u + 1 \rfloor$
  - $V = \lfloor M_y \cdot v + 1 \rfloor$
  - Wtedy gdy u = 0 to U = 0 i gdy u = 1 to U = M_x + 1, dlatego musimy sprawdzić czy 0 < u, v < 1



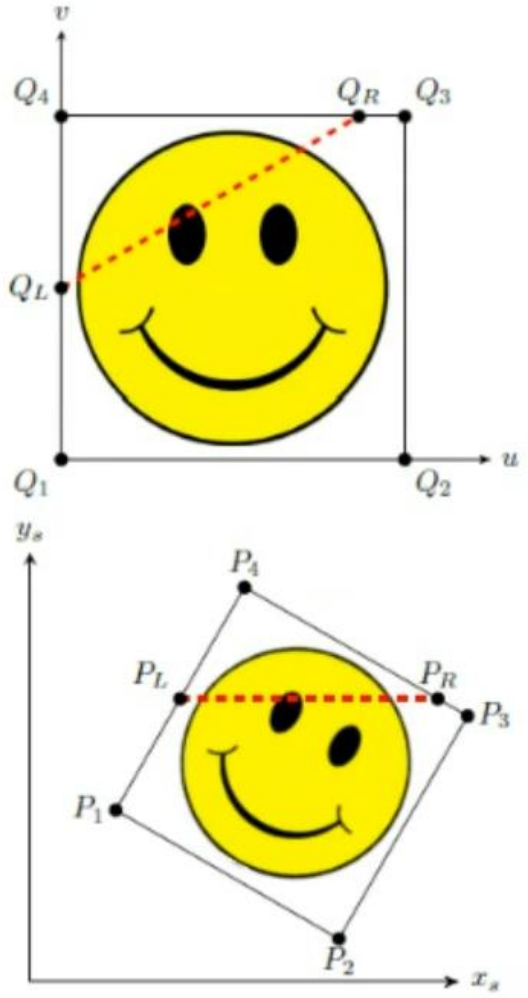
a texture space

b screen space

- Wierzchołki wielokąta P, korespondują do wierzchołków Q w przestrzeni tekstury
- Każda transformacja wielokątów powinna być uwzględniona w teksturowaniu
- Używając metody skanowania wierszy, ekstremum P_L jest obliczony interpolując pomiędzy P_1 i P_4
- Podobnie ekstremum P_R jest obliczony interpolując pomiędzy P_4 i P_3



- Korespondujące ekstremy w przestrzeni tekstury Q_L i Q_R, są obliczone interpolując krawędzie tekstury
- Piksele pomiędzy P_L i P_R przybierają ten sam kolor jak teksele pomiędzy Q_L i Q_R



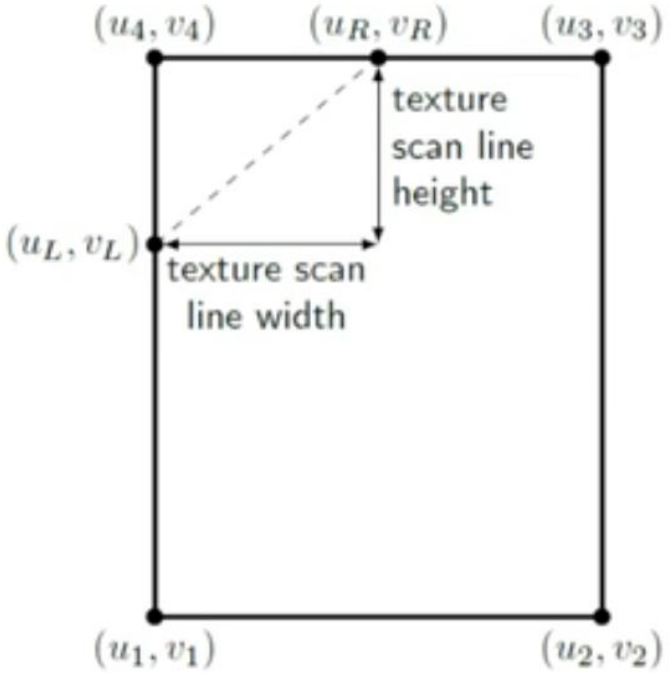
Obliczenie ekstremy współrzędnych

- Współrzędne wierzchołków są skalowane do wielkości wyświetlacza
- $x = max\left[1, (x_s + 1)\frac{N_x}{2}\right], \qquad y = max\left[1, (y_s + 1)\frac{N_y}{2}\right]$
- Ekstremy P_L i P_R są inicjalizowane z wartością największej współrzędnej y_s (tzn. P_4)
- Współrzędna y od P_L i P_R jest obliczana odejmując 1 od obecnej współrzędnej
- x_L jest obliczony interpolując pomiędzy P_4 i P_1
- $x_L = x_4 - \frac{x_4 - x_1}{y_4 - y_1}$

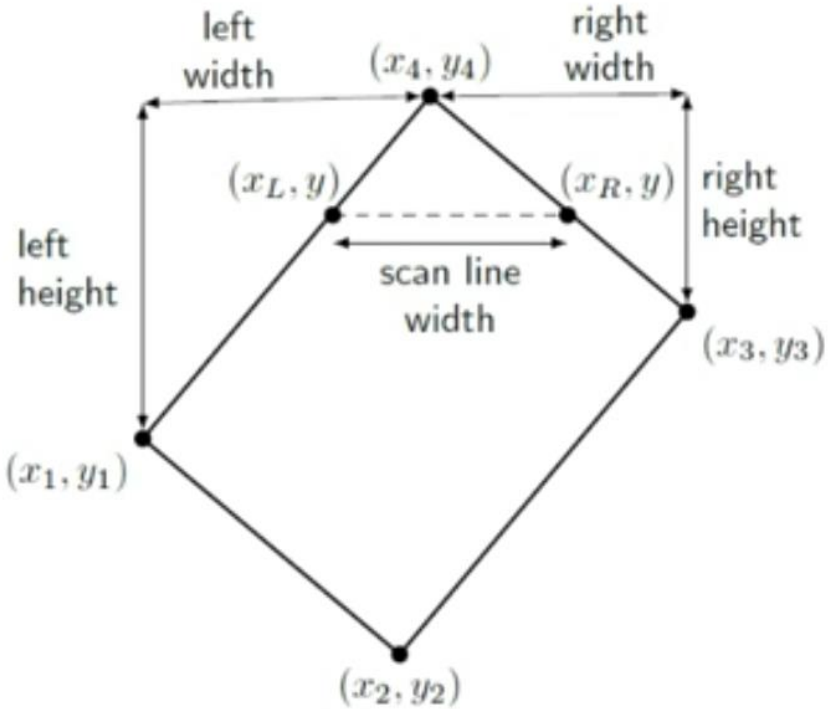
Obliczenie ekstremy współrzędnych

- $x_L = x_4 - \frac{x_4 - x_1}{y_4 - y_1}$
- Gradient jest stały wzdłuż krawędzi wielokąta, tzn. możemy ułamek przekalkulować
- $x_L = x_L - \Delta x_L$
- $x_R = x_R - \Delta x_R$
- Gdzie
- $\Delta x_L = \frac{dlugosc\ lewej\ krawedzi}{wysokosc\ lewej\ krawedzi} = \frac{x_4 - x_1}{y_4 - y_1}$
- $\Delta x_R = \frac{dlugosc\ prawej\ krawedzi}{wysokosc\ prawej\ krawedzi} = \frac{x_4 - x_3}{y_4 - y_3}$

Obliczenie ekstremy współrzędnych



a texture

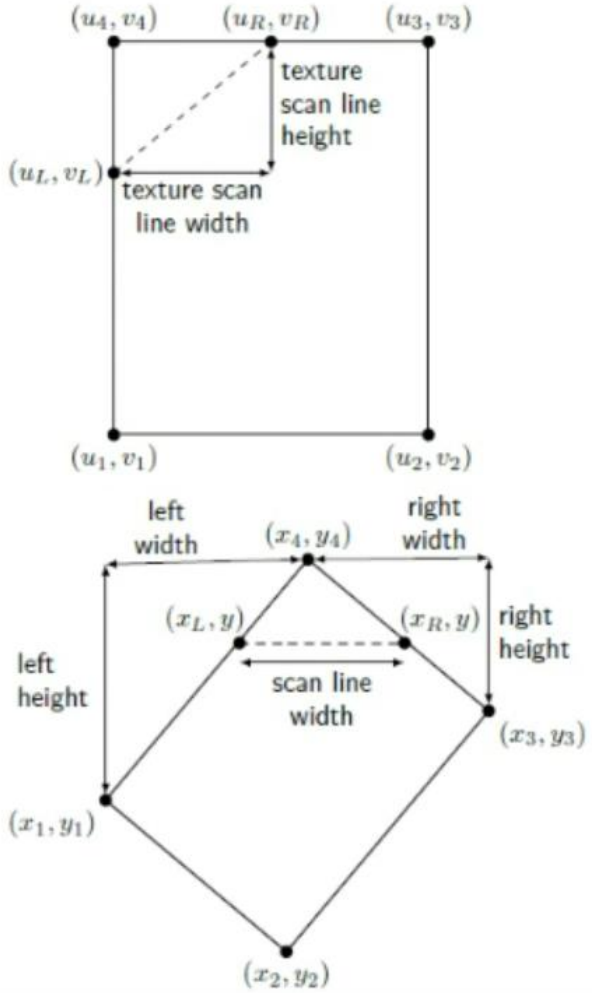


b polygon

Obliczenie ekstremy współrzędnych w przestrzeni teksturowej

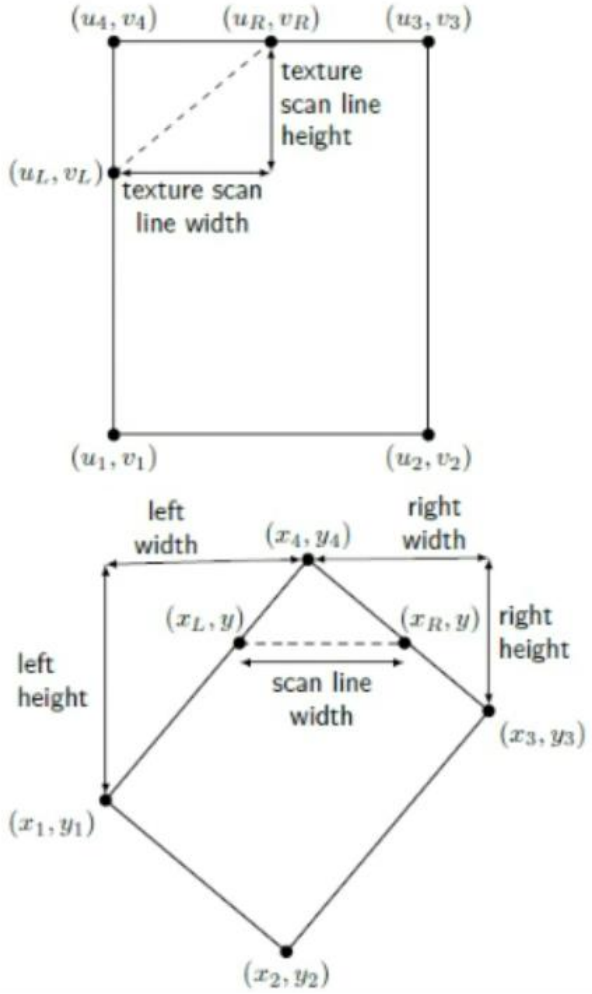
- v_L przemieszcza się pomiędzy v_4 i v_1 tak samo jak y pomiędzy y_4 i y_1
- $\Delta u_L = \frac{u_4 - u_1}{y_4 - y_1}$

- $\Delta v_L = \frac{v_4 - v_1}{y_4 - y_1}$
- Gdy P_L obniża się do następnego wiersza skanowania
- $u_L = u_L - \Delta u_L$
- $v_L = v_L - \Delta v_L$
- Podobnie dla u_R i v_R



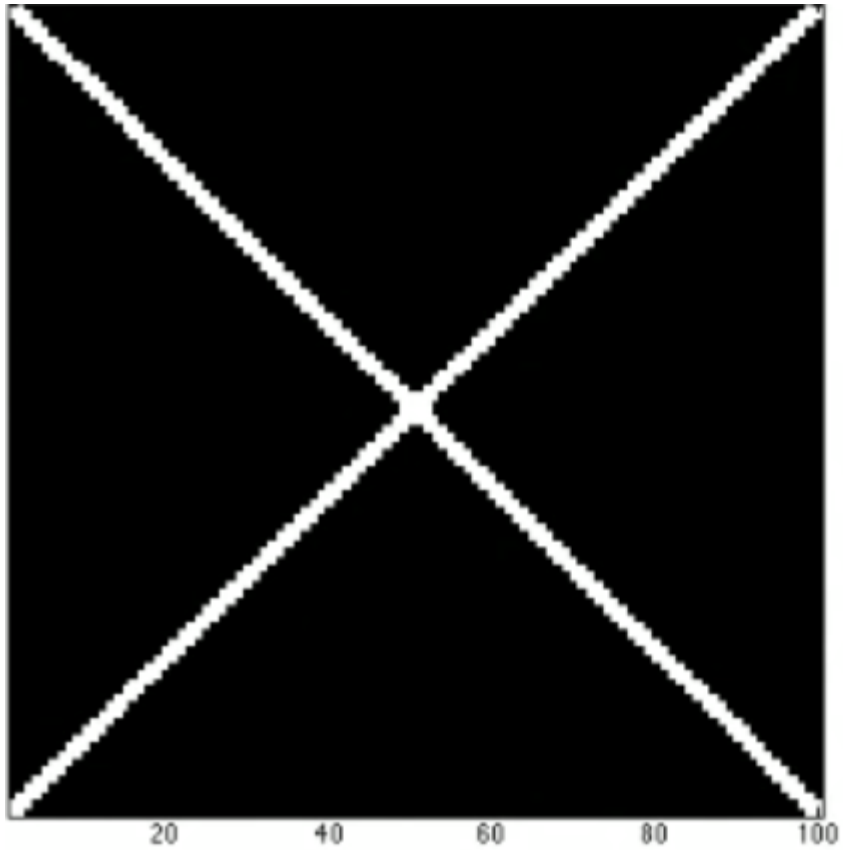
Interpolacja wierszu skanowania

- Zaczynajac w (x_L, x_R) inicjalizujemy u = u_Li v = v_L
- Dla każdego piksela przemieszczamy się na wierszu skanowania i aktualizujemy współrzędne (u, v)
- $u = u + \Delta u$
- $v = v + \Delta v$
- Idziemy od u_L do u_R w tej samej powłoce liniowej jak dla x_L do x_R, dlatego
- $\Delta u = \frac{u_R - u_L}{x_R - x_L}$
- $\Delta v = \frac{v_R - v_L}{x_R - x_L}$

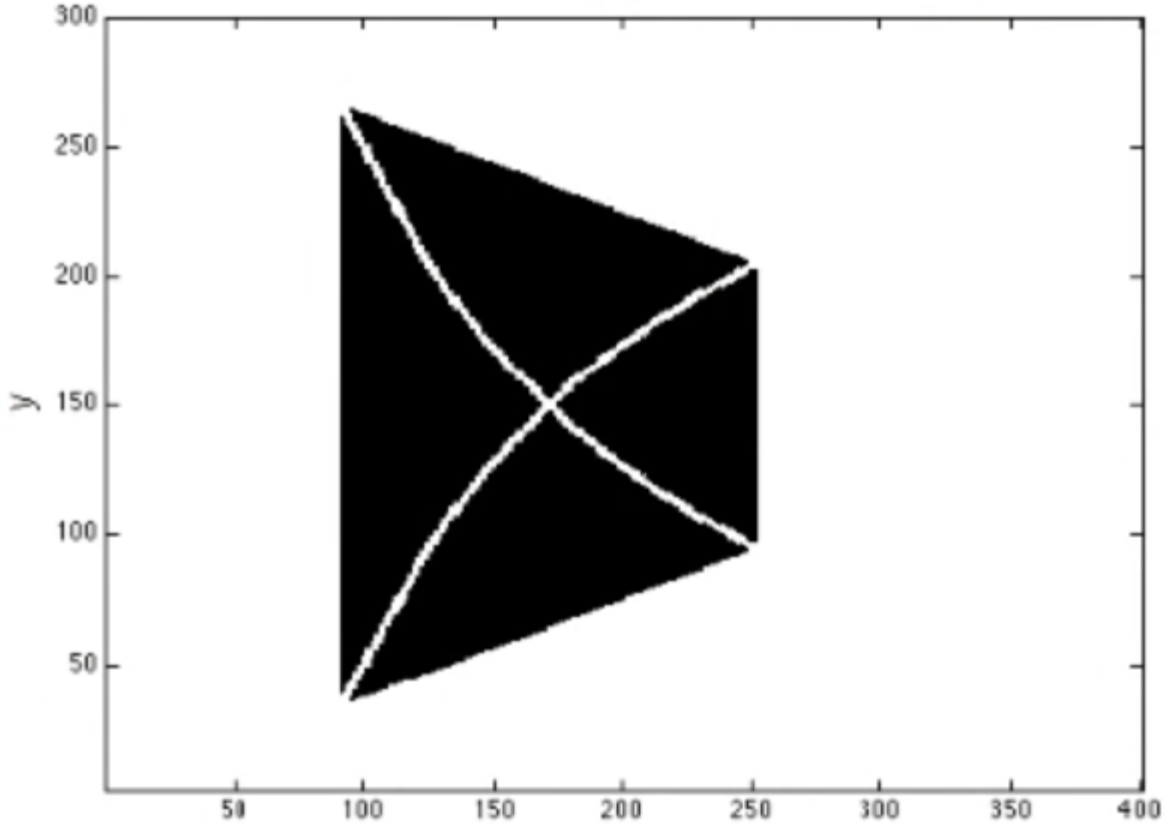


Texture space

Screen space



Texture space

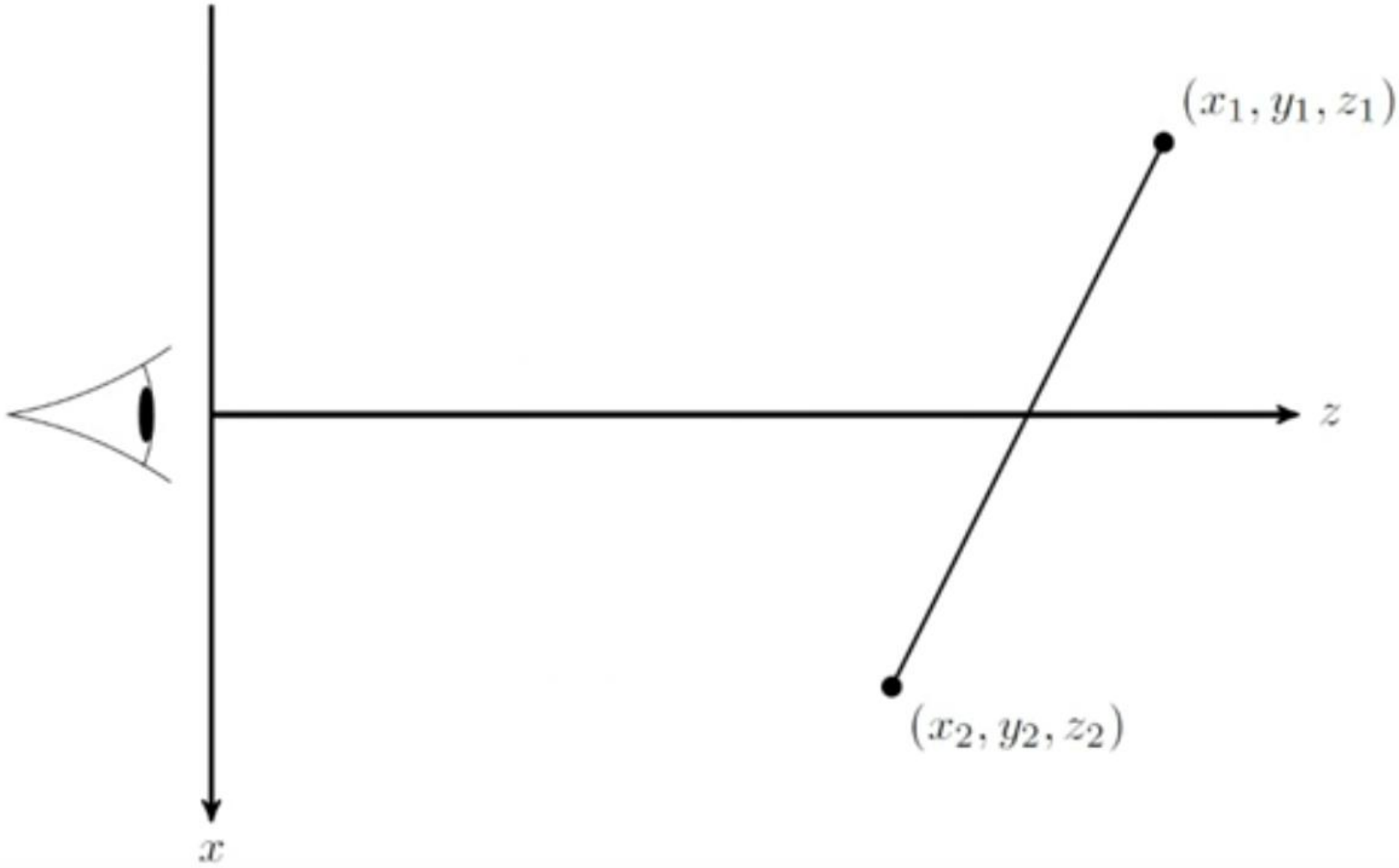


Screen space

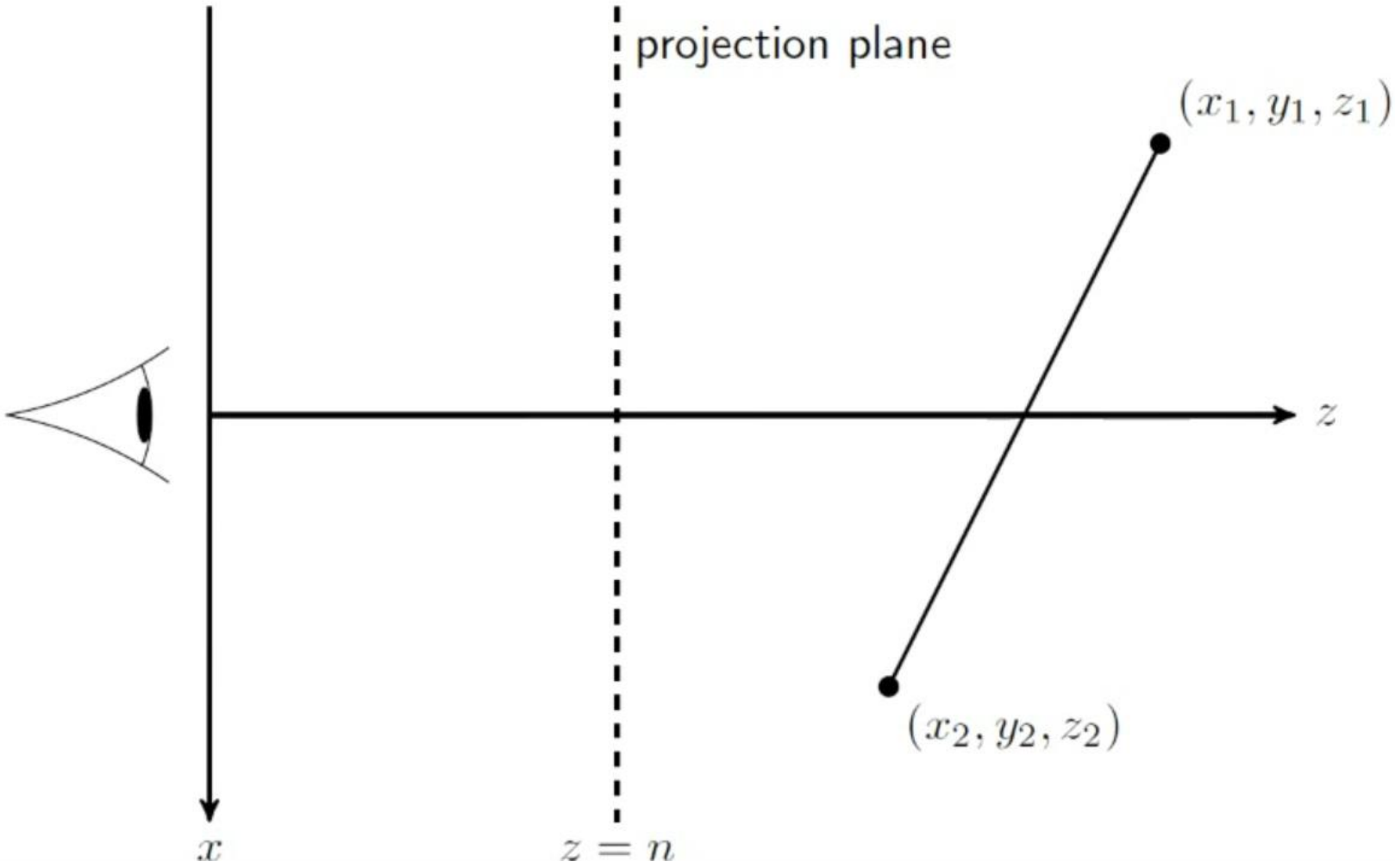
Dlaczego widzimy artefakty?

- Nasza metoda teksturowania opiera się na założeniu że rzutowanie z 3D na 2D jest liniowa transformacja
- Relacja pomiędzy x i x_s i y i y_s jest liniowa, relacja pomiędzy z i z_s **nie jest liniowa**
- Żeby pozbyć się artefaktów musimy dystans od obserwatora brać pod uwagę

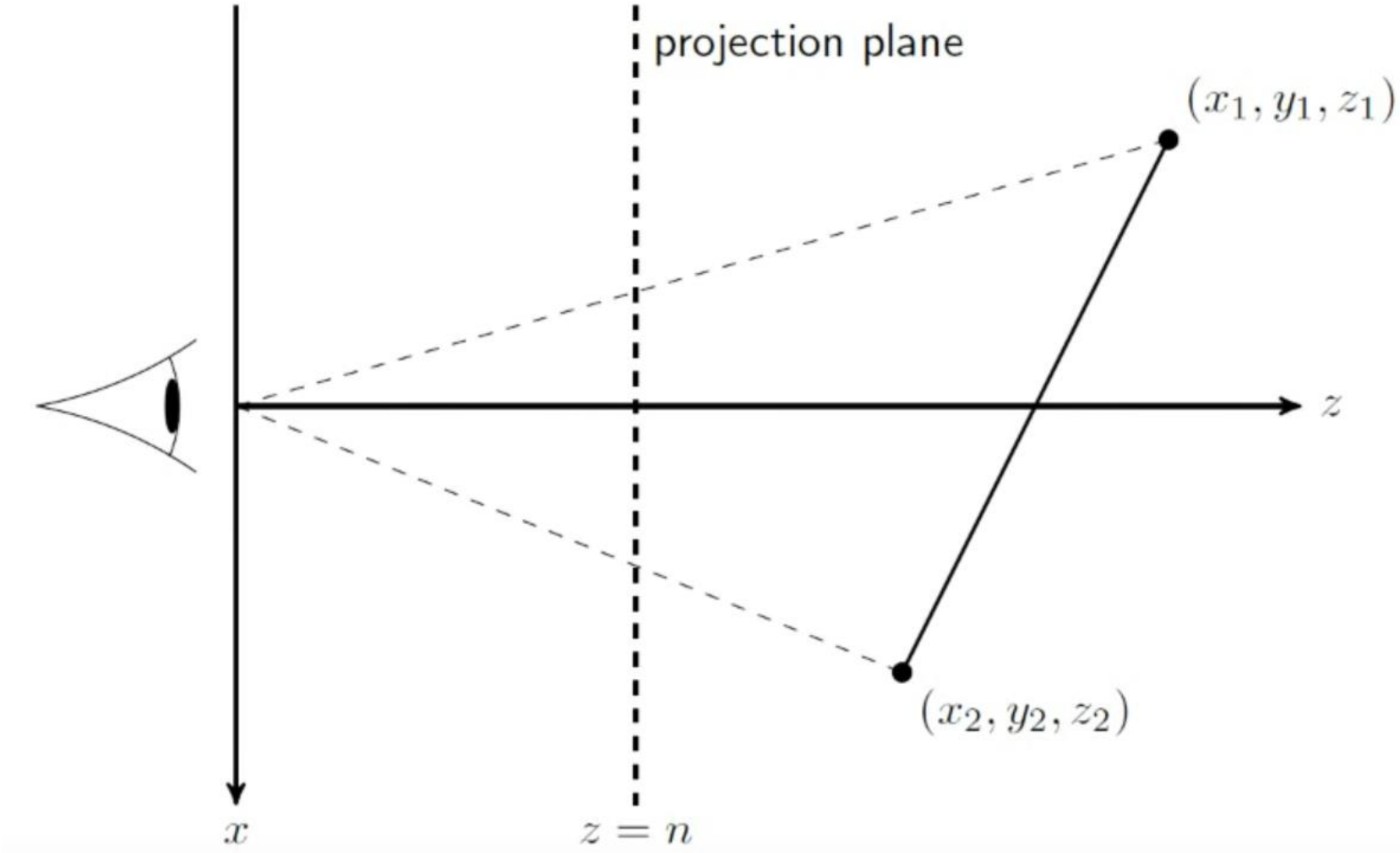
Rzutowanie perspektywiczne



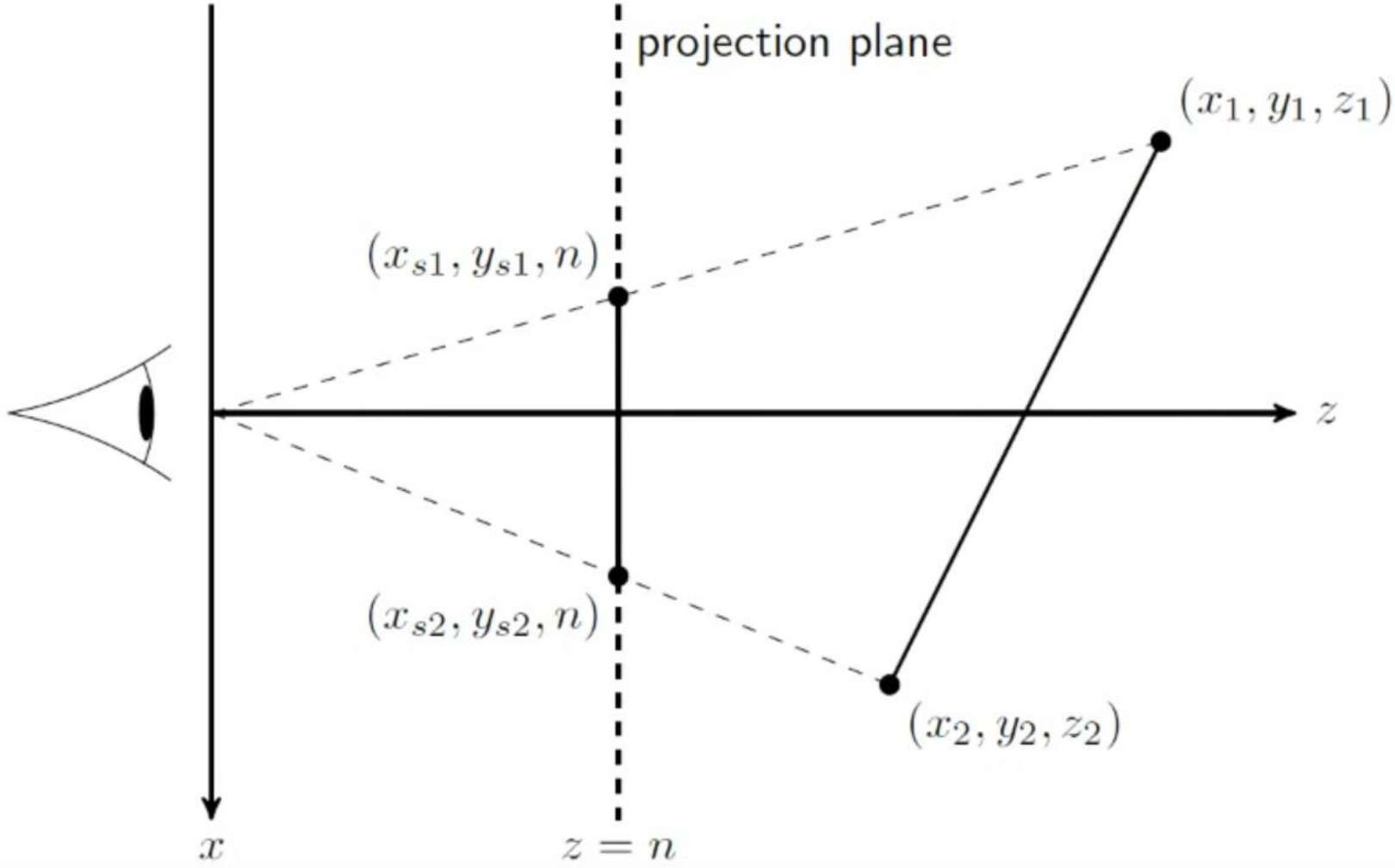
Rzutowanie perspektywiczne



Rzutowanie perspektywiczne

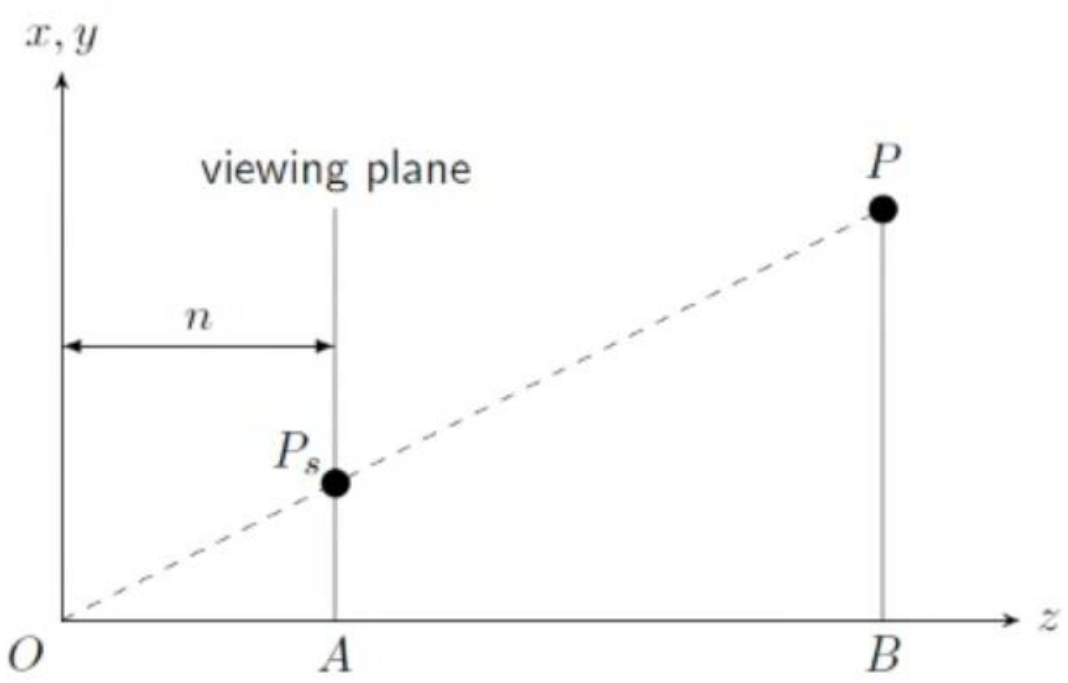


Rzutowanie perspektywiczne



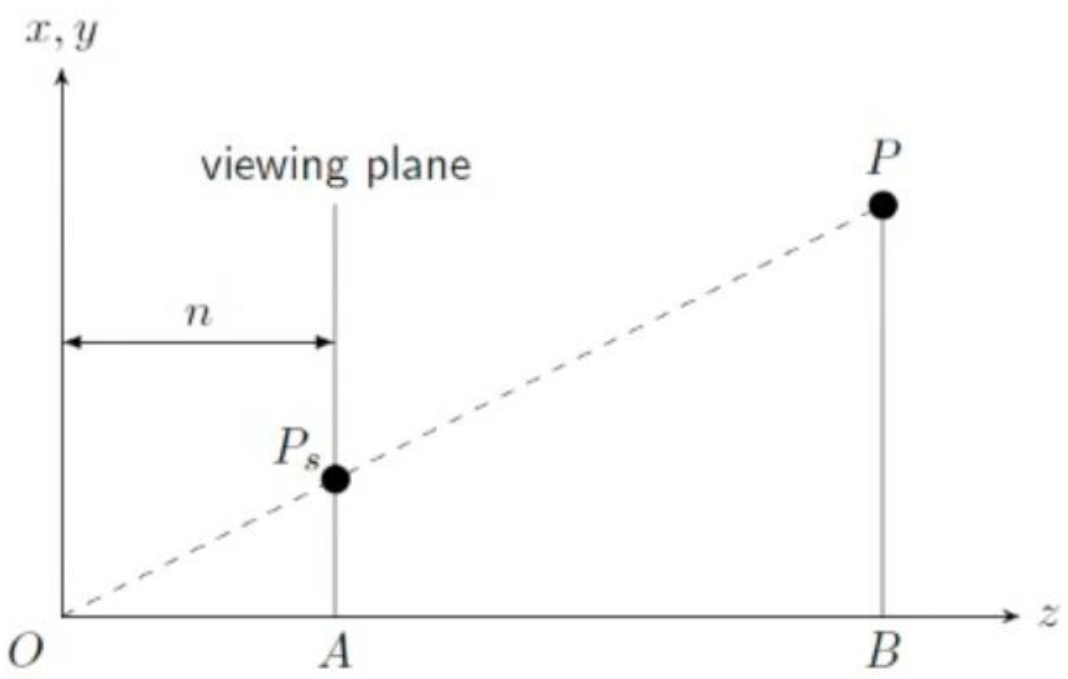
Rzutowanie perspektywiczne

- Punkt w przestrzeni świata P jest rzutowany na przestrzeń okna P_s
- Trójkąty OAP i OBP są podobne:
- $\frac{x_s}{n} = \frac{x}{z} \quad \Rightarrow x_s = \frac{n}{z}x$
- $\frac{y_s}{n} = \frac{y}{z} \quad \Rightarrow y_s = \frac{n}{z}y$
- $z_s = n$



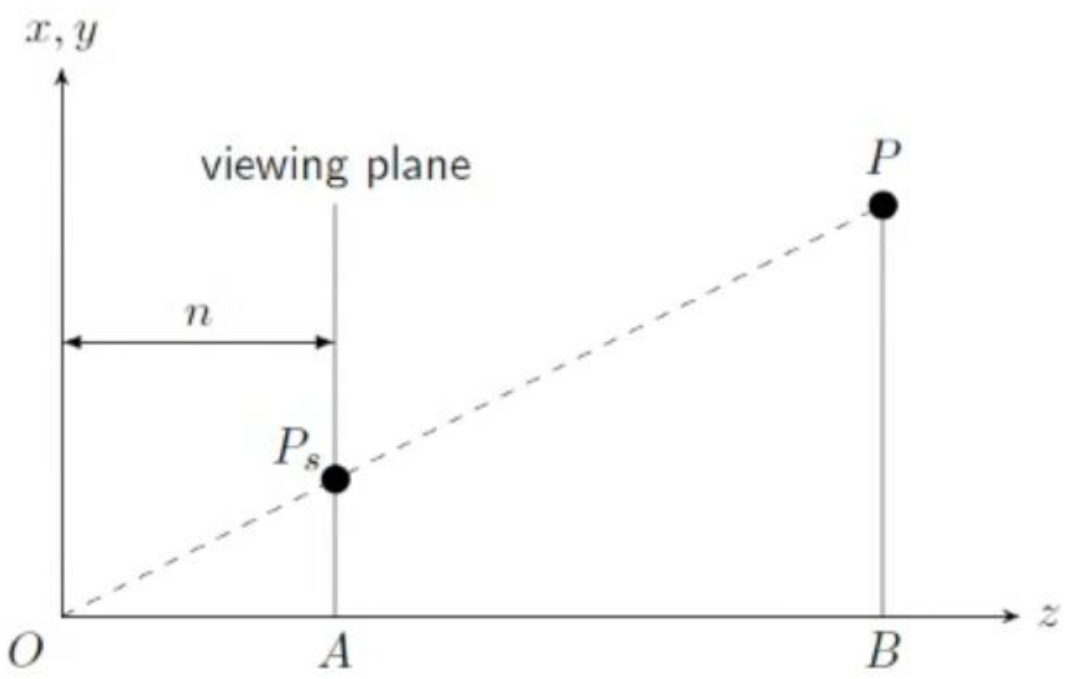
Rzutowanie perspektywiczne

- Rownanie krawędzi w przestrzeni świata jest dane przez:
- $x = \alpha z + \beta$
- Odwracając rzutowanie perspektywiczne
- $x = \frac{z}{n} x_s$



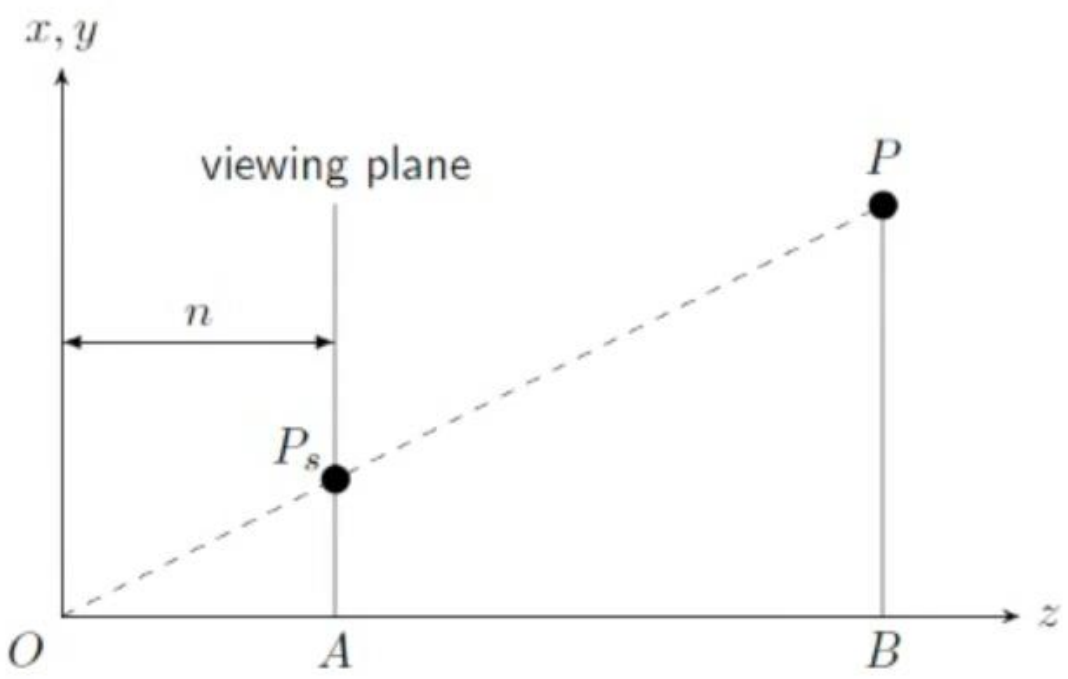
Rzutowanie perspektywiczne

- Rownanie krawędzi w przestrzeni świata jest dane przez:
- $x = \alpha z + \beta$
- Odwracając rzutowanie perspektywiczne
- $x = \frac{z}{n} x_s$
- Daje nam $\frac{z}{n} x_s = \alpha z + \beta$

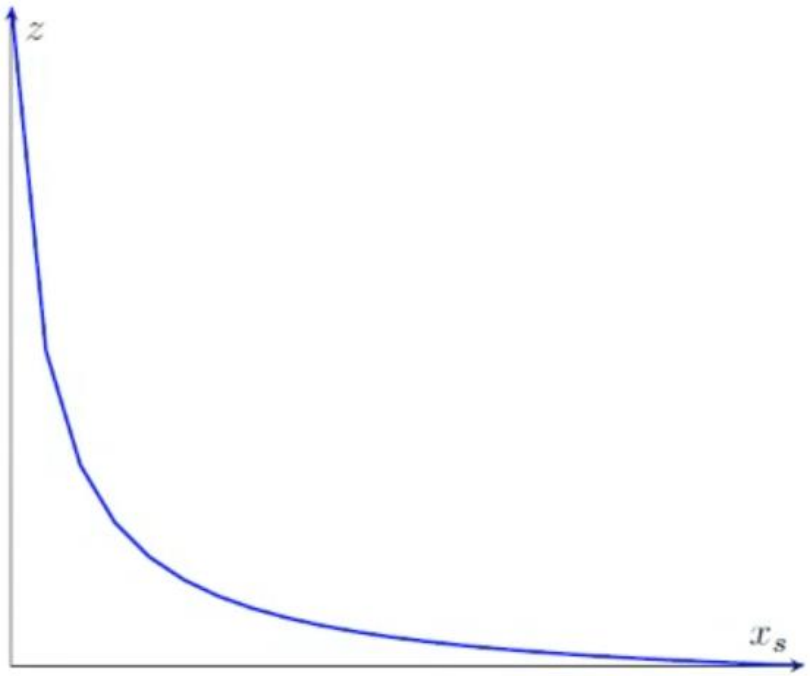


Rzutowanie perspektywiczne

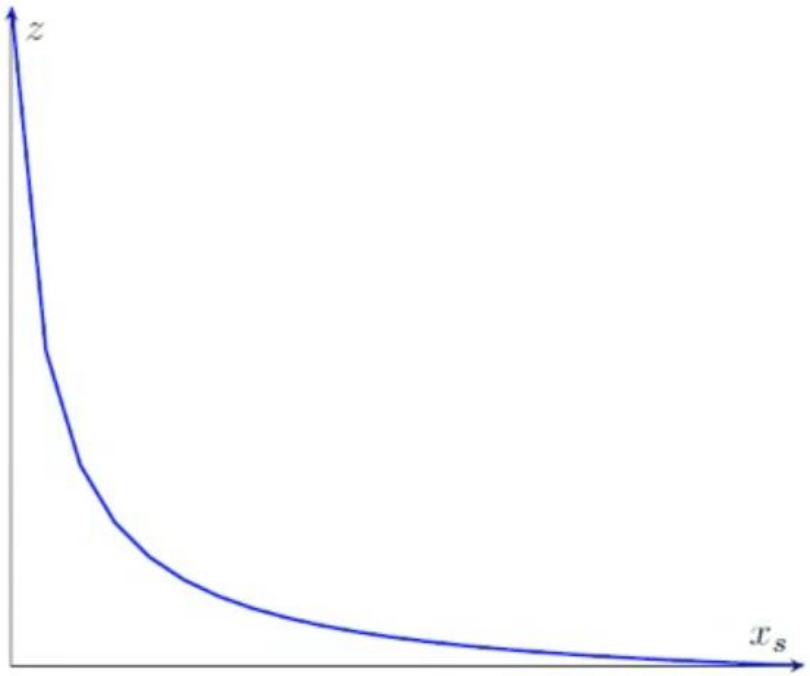
- Rownanie krawędzi w przestrzeni świata jest dane przez:
- $x = \alpha z + \beta$
- Odwracając rzutowanie perspektywiczne
- $x = \frac{z}{n} x_s$
- Daje nam $\frac{z}{n} x_s = \alpha z + \beta$
- $z = \beta n \frac{1}{x_s} + \frac{\beta}{\alpha}$



- $z = \beta n \frac{1}{x_s} + \frac{\beta}{\alpha}$
- To nie jest liniowa relacja pomiędzy z i x_s

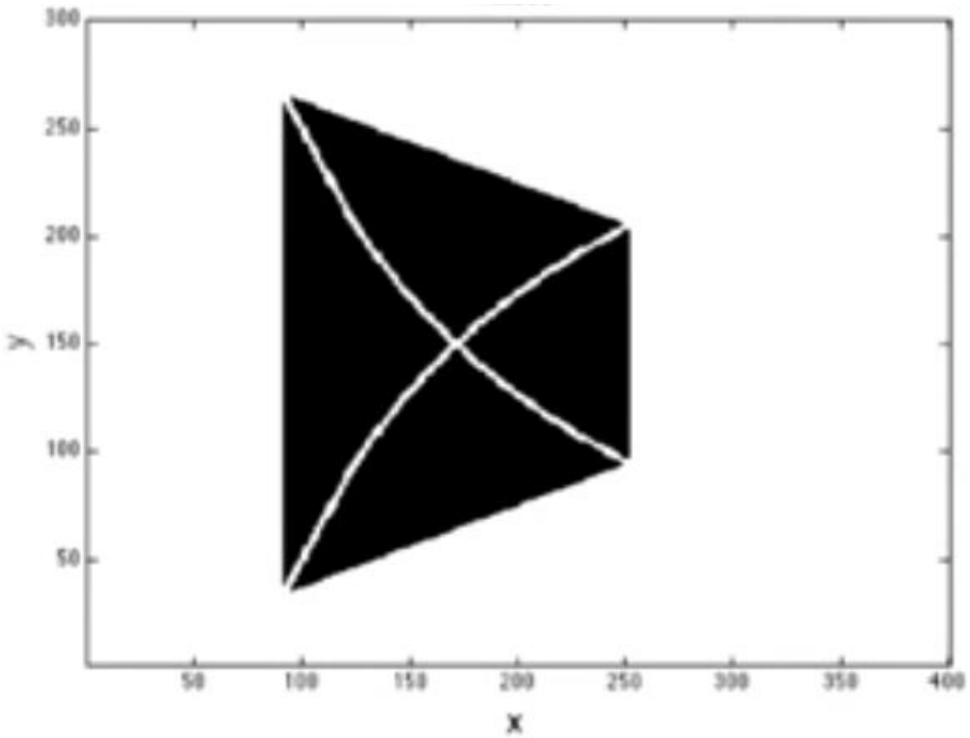


- $z = \beta n \frac{1}{x_s} + \frac{\beta}{\alpha}$
- To nie jest liniowa relacja pomiędzy z i x_s
- Ale możemy używać zwrotna relacje
- $\frac{1}{z} = \frac{x_s}{\beta n} - \frac{\alpha}{\beta}$
- Istnieje liniowa relacja pomiędzy $\frac{1}{z}$ i x_s

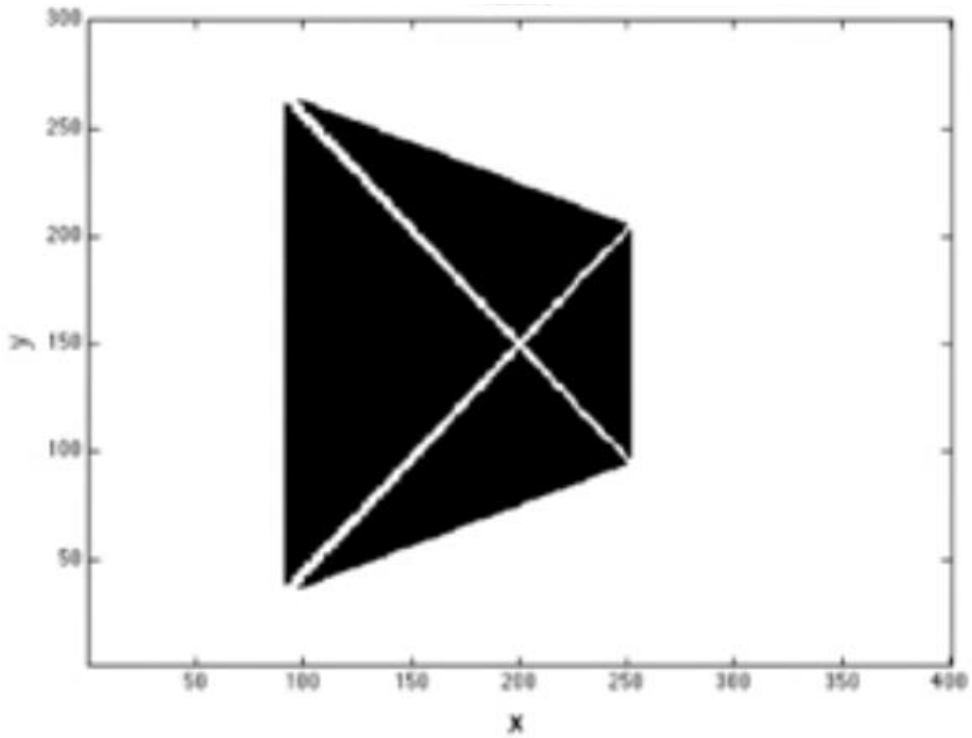


Perspektywiczne teksturowanie

- Żeby zastosować rzutowanie perspektywiczne ale nadal używając liniową interpolacje musimy obliczyć wartość $\frac{1}{z}$ dla ekstremy P_L i P_R
- Wierzchołki przestrzeni teksturowej są podzielone przez z dając $\frac{u}{z}$ i $\frac{v}{z}$
- Liniowa interpolacja jest wtedy zastosowana jak poprzednio żeby obliczyć współrzędne przestrzeni teksturowej $(\frac{u}{z}, \frac{v}{z})$ które korespondują do współrzędnych przestrzeni okna $(x_s, y_s, \frac{1}{z})$



Tylko liniowa interpolacja

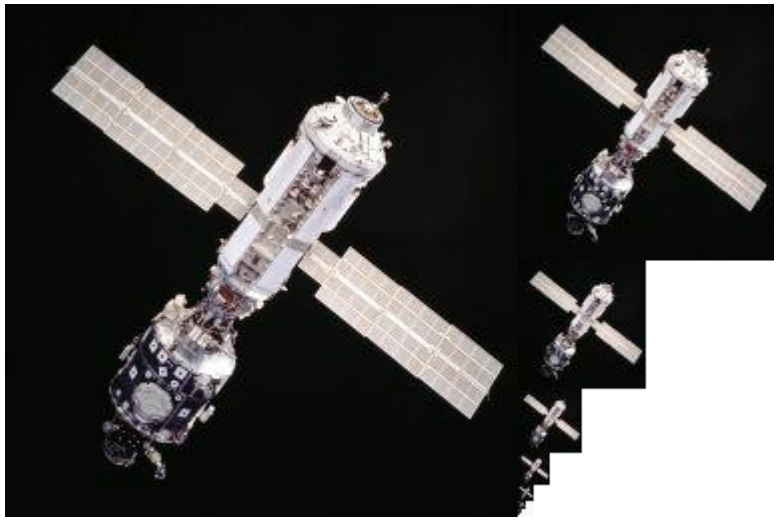


Rzutowanie perspektywiczne

Mipmapping

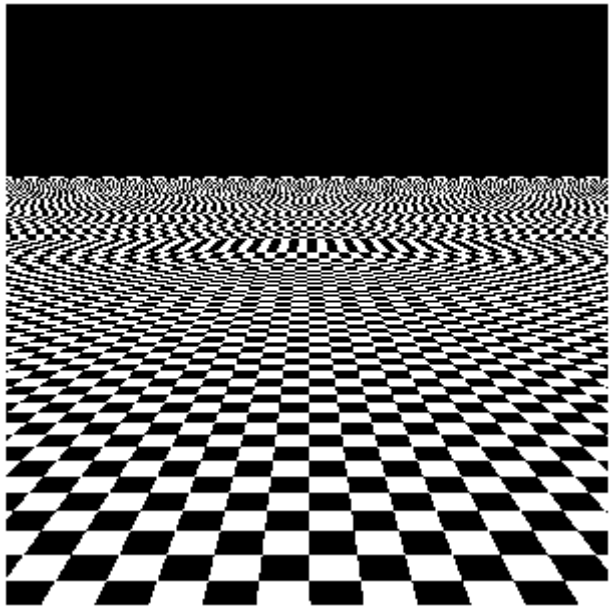
Mipmapping

- Przyspiesza teksturowanie



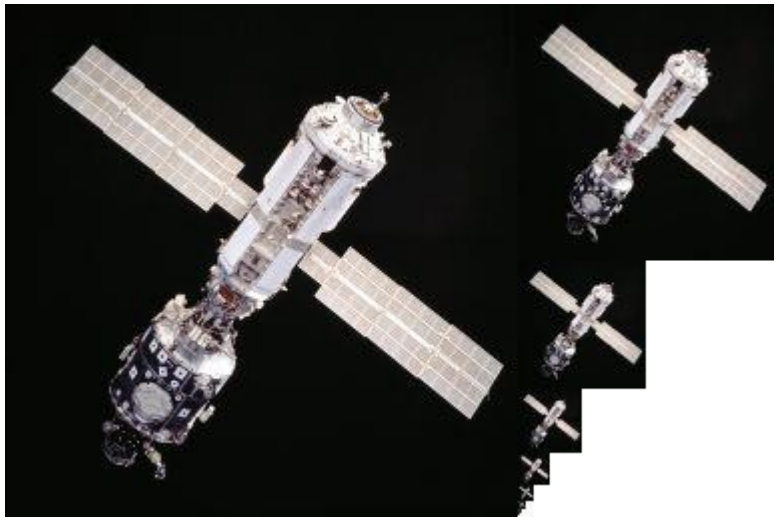
Mipmapping

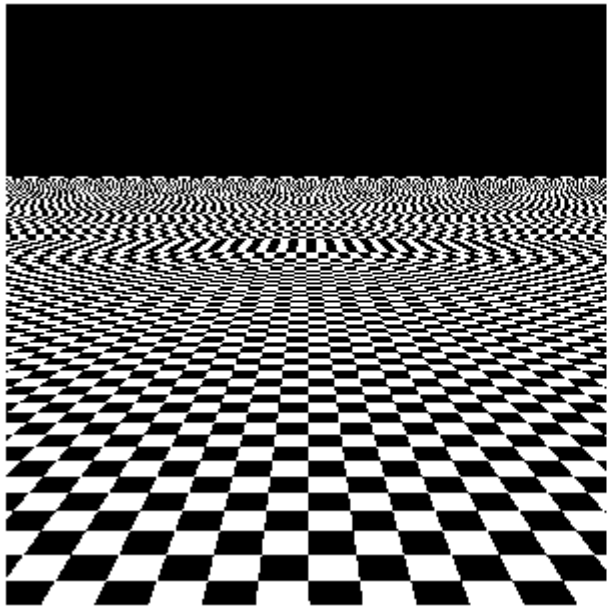
- Przyspiesza teksturowanie



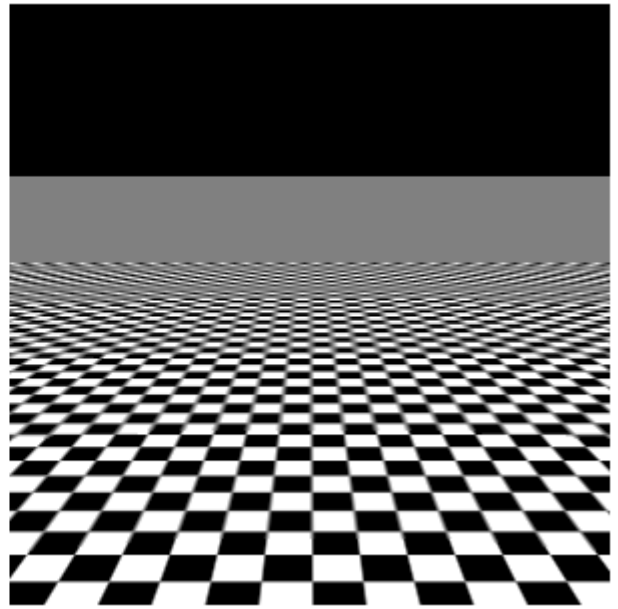
Mipmapping

- Przyspiesza teksturowanie





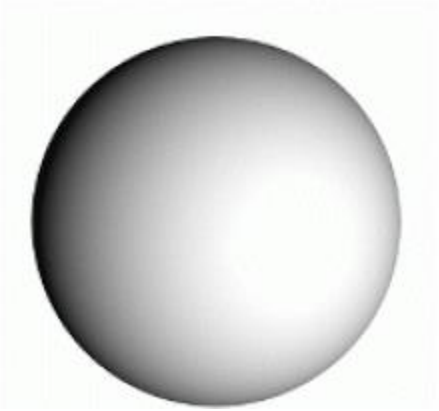
bez mipmapping



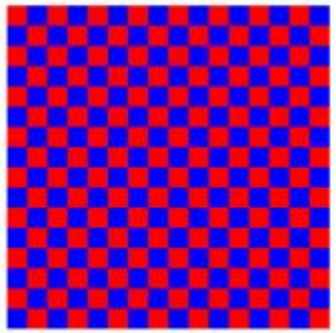
mipmapping

Teksturowanie i oświetlenie

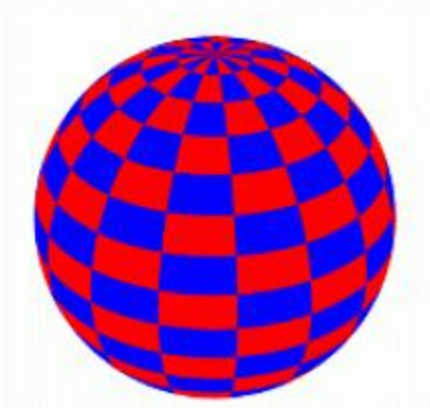
- Teksturowanie może być używane żeby zmienić wartości parametrów modelu oświetlenia (np. parametr k_d)



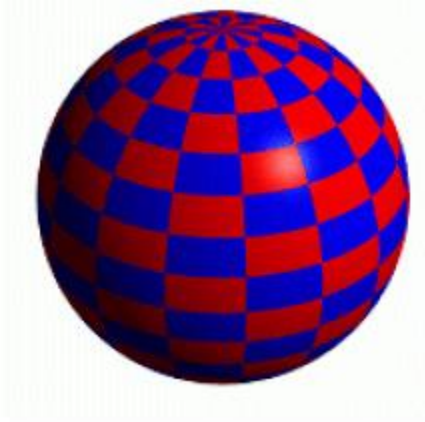
Cieniowanie



Tekstura



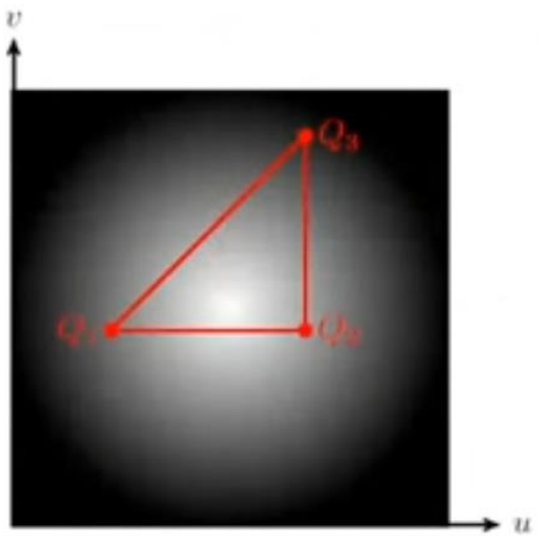
Nałożona tekstura



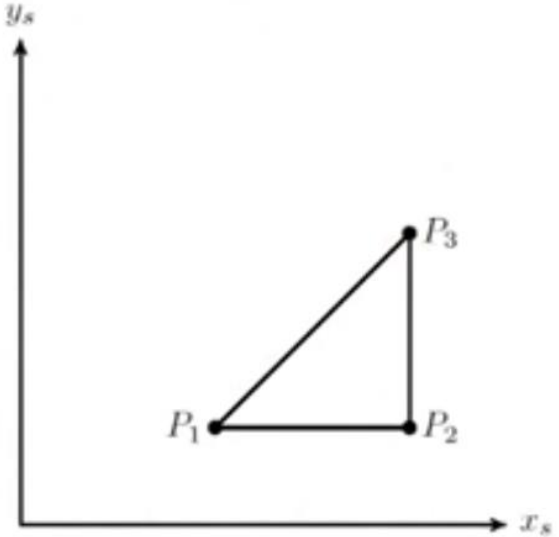
Oświetlenie i tekstura

Szybkie cieniowanie Phonga

- Teksturowanie może być używane żeby przyspieszyć obliczenia cieniowania Phonga



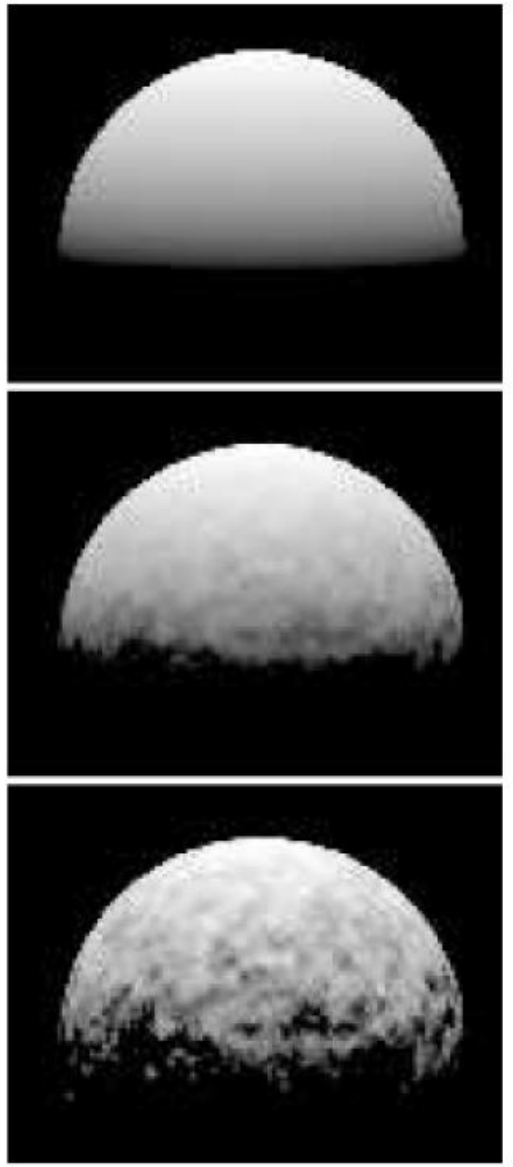
Phong map



Screen space

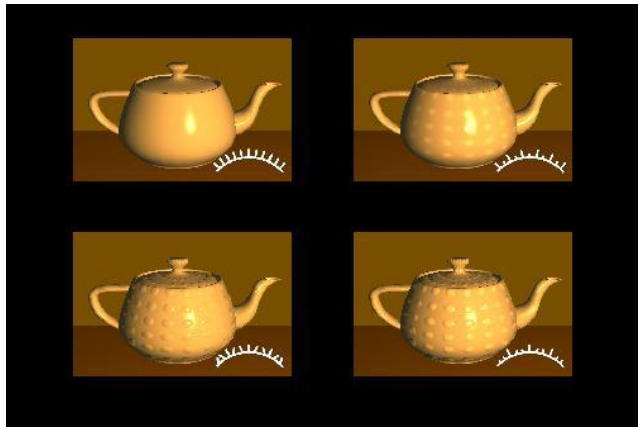
Mapowanie normalnych

- Jeden z powodów dlaczego używamy tekstury jest żeby modelować powierzchnie które nie są płaskie ale często nierówne.
- Te nierówności mogą uczynić ze światło jest odbijane pod różnych kątach.



Mapowanie normalnych

- Ludzkie oko dobrze rozpoznaje formę obiektów z bodźców oświetlenia
- Możemy ten fakt wykorzystać i użyć obraz normalnych dla naszych płaszczyzn żeby stworzyć wrażenie nierównej powierzchni
- Nazywamy ten proces „mapowanie normalnych” lub „bump mapping”

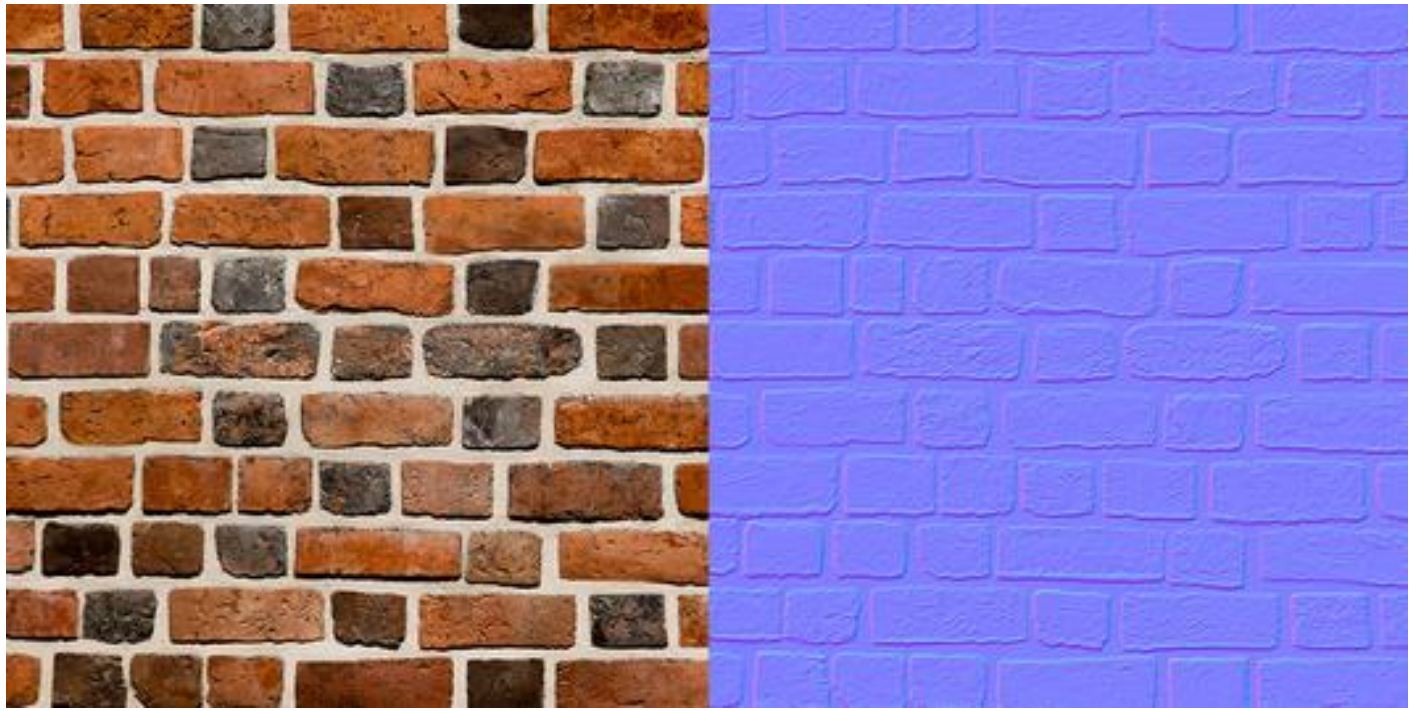


Mapowanie normalnych

- Dla każdego cieniowanego punktu w naszej scenie podajemy trójwymiarowe normalne przechowane w dwuwymiarowej mapie normalnych

- `Dla każdego punktu`
  - `Interpoluj UV`
  - `Normalna = mapaNormalnych[U, V]`
  - `Oblicz cieniowanie używając tą normalną`

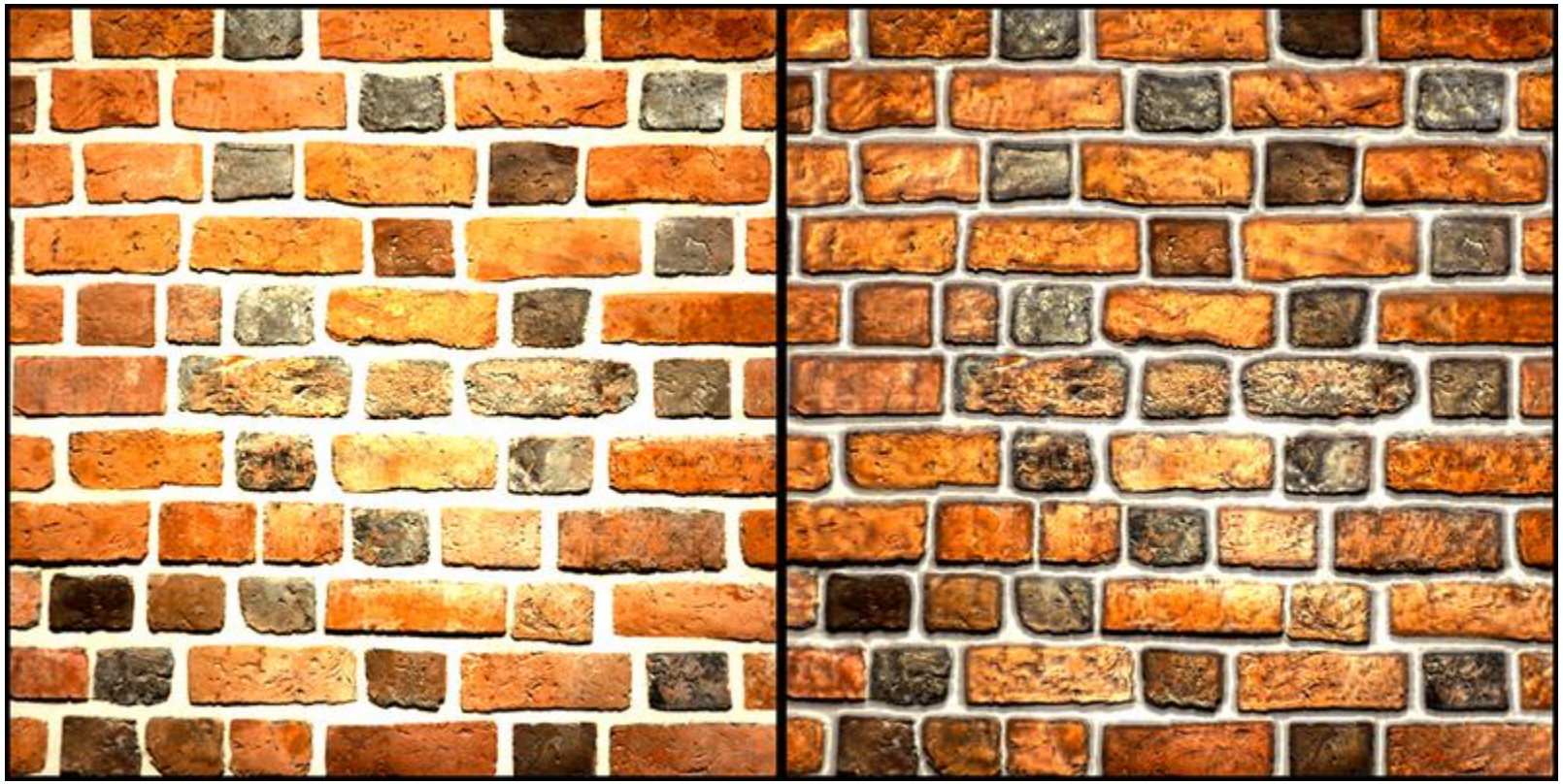
Mapowanie normalnych



Tekstura

Mapa normalnych

Mapowanie normalnych

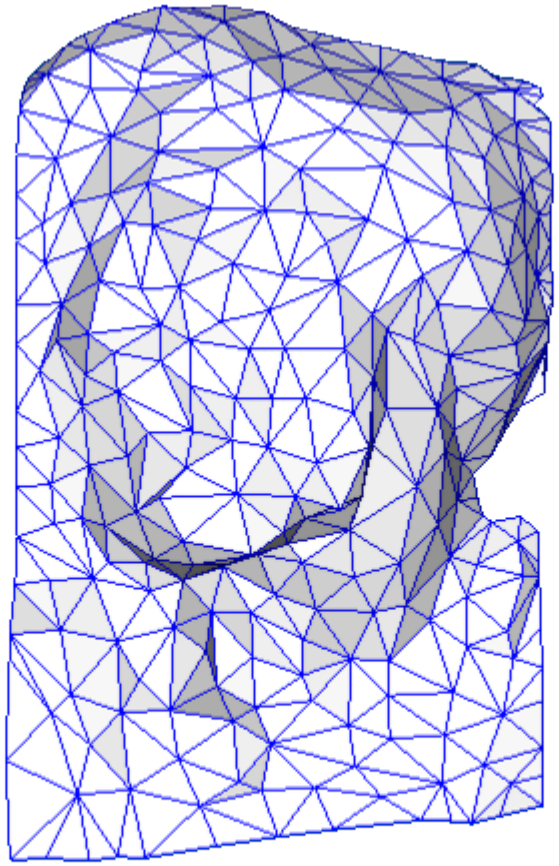


Bez bump mapping

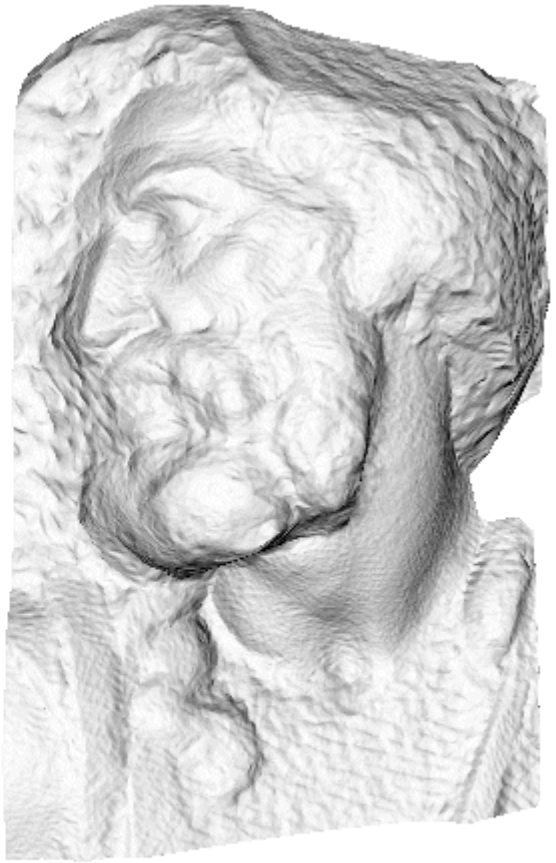
Bump mapping

Mapowanie normalnych

original mesh
4M triangles

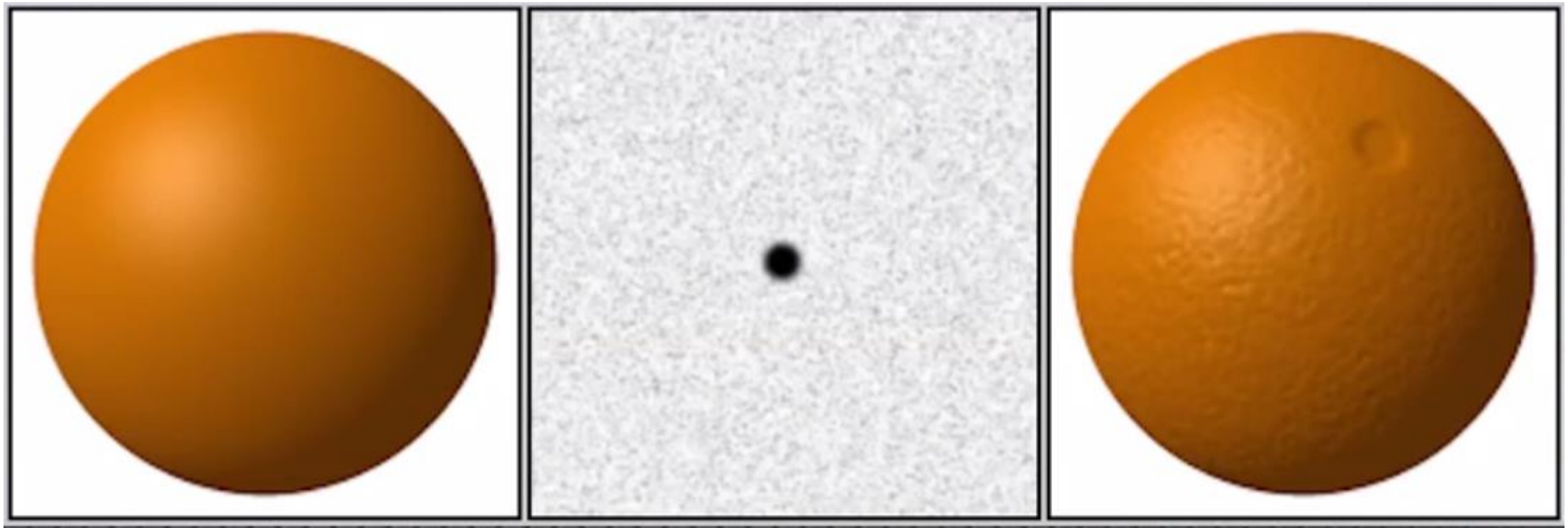


simplified mesh
500 triangles

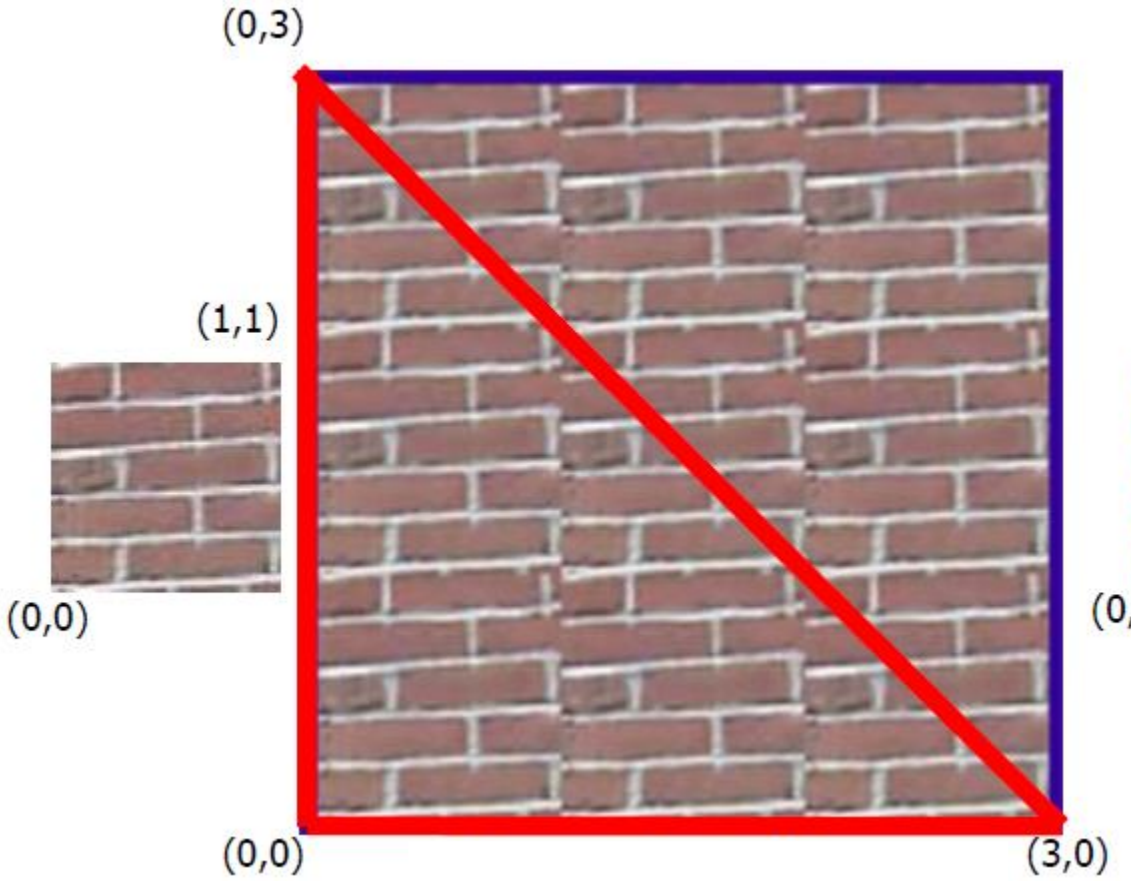


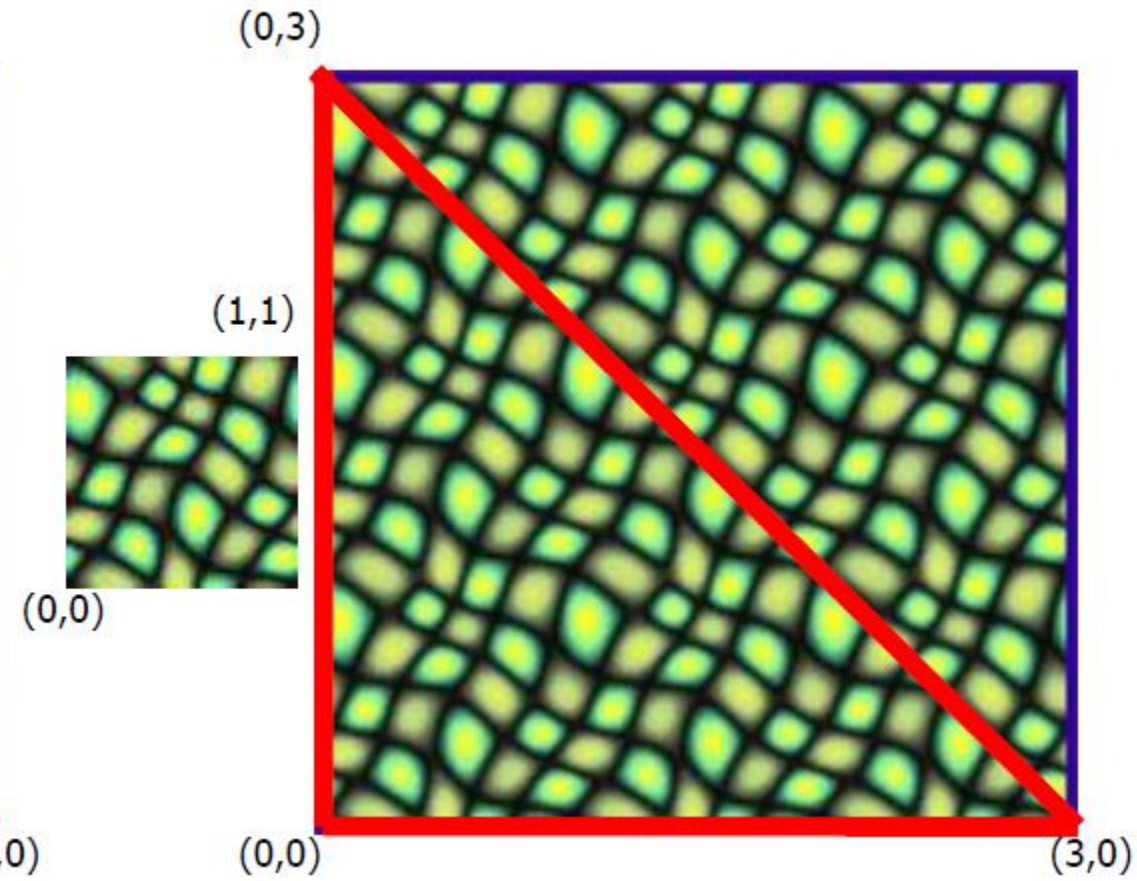
simplified mesh
and normal mapping
500 triangles

Normalne obliczone z mapy wysokości



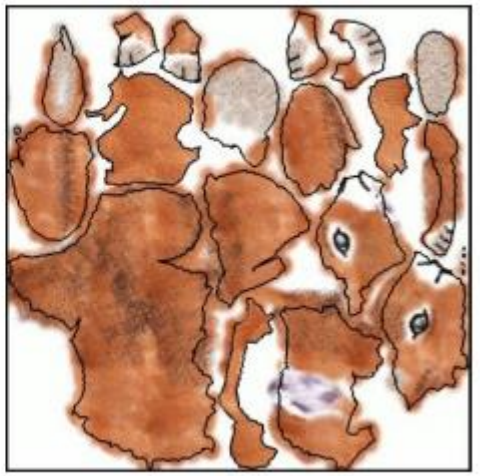
Układanie tekstur



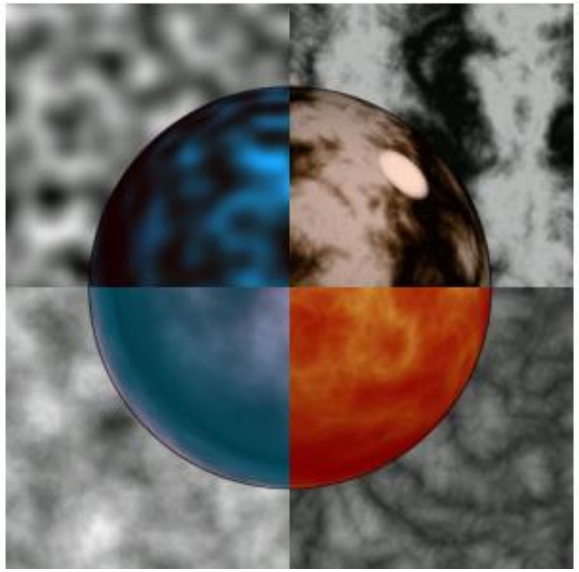


Dwa główne sposoby

- Z danych: teksturowanie
  - Wczytaj informacje o kolorach z 2D obrazów



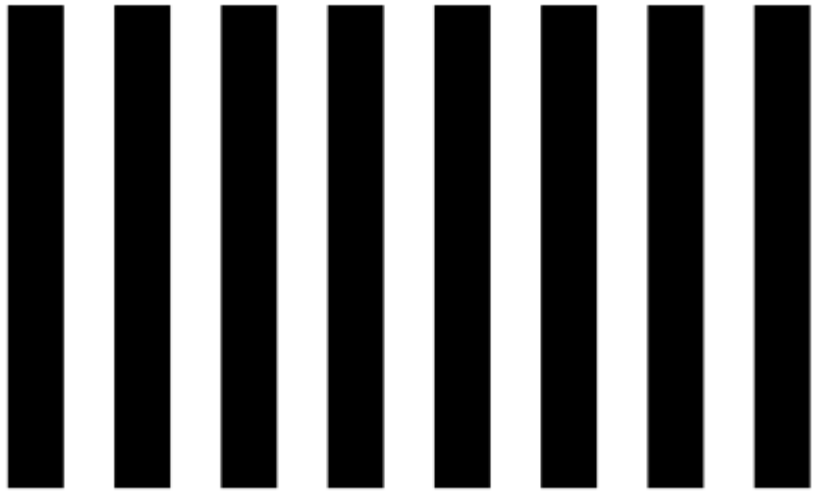
- Proceduralnie: w shaderze
  - Napisać program który oblicza kolor jako funkcja pozycji w przestrzeni



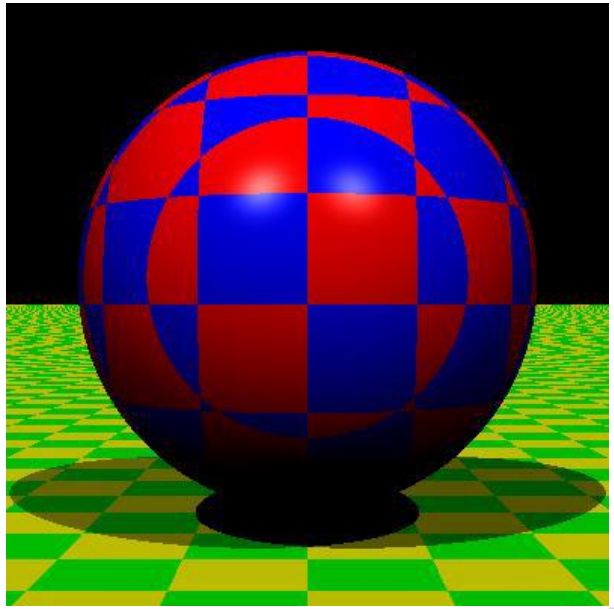
Tekstury proceduralne

- Alternatywnie do teksturowania możemy stworzyć program który oblicza kolor jako funkcja pozycji (x, y, z)

- $f(x, y, z) \rightarrow kolor$

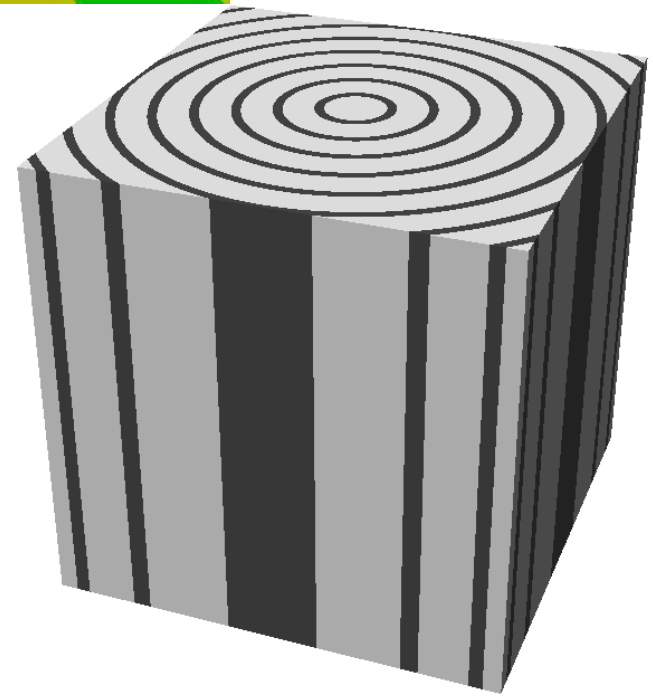


Tekstury proceduralne



- Zalety:
  - Zajmują mniej miejsca w pamięci
  - Nieskończona rezolucja

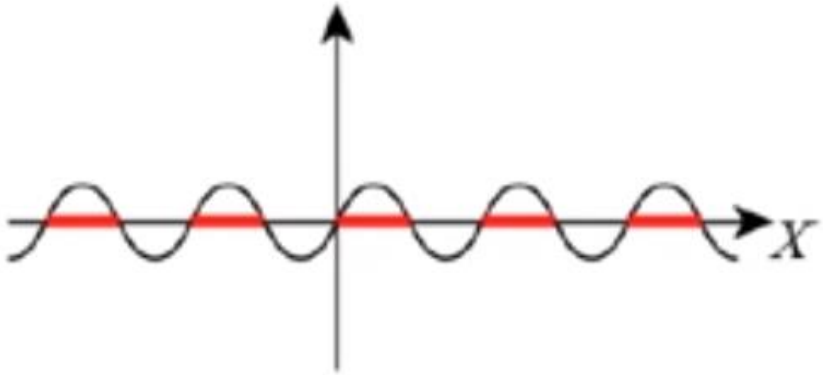
- Wady:
  - Mało intuicyjne
  - Często trudno istniejący wzorzec zamodelować za pomocy funkcji



Przykład: 3D paski

- Paski wzdłuż osi x:

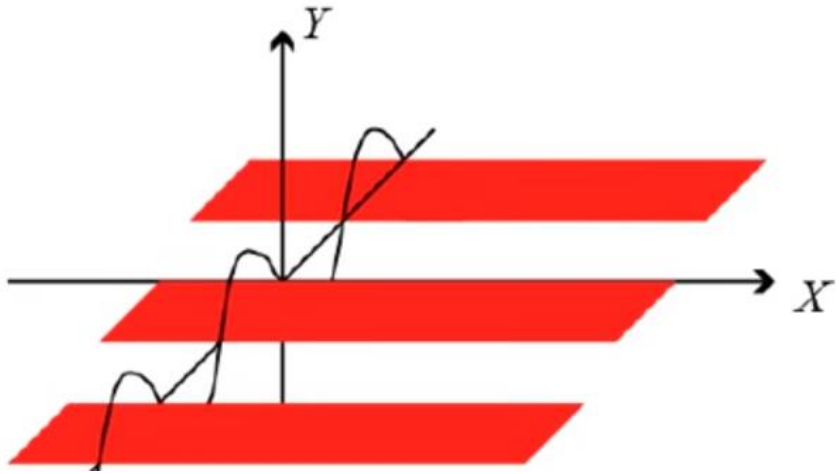
```
Paski(x_s, y_s, z_s)
{
   If(sin x_s > 0) return color0
   Else return color1
}
```



Przykład: 3D paski

- Paski wzdłuż osi z:

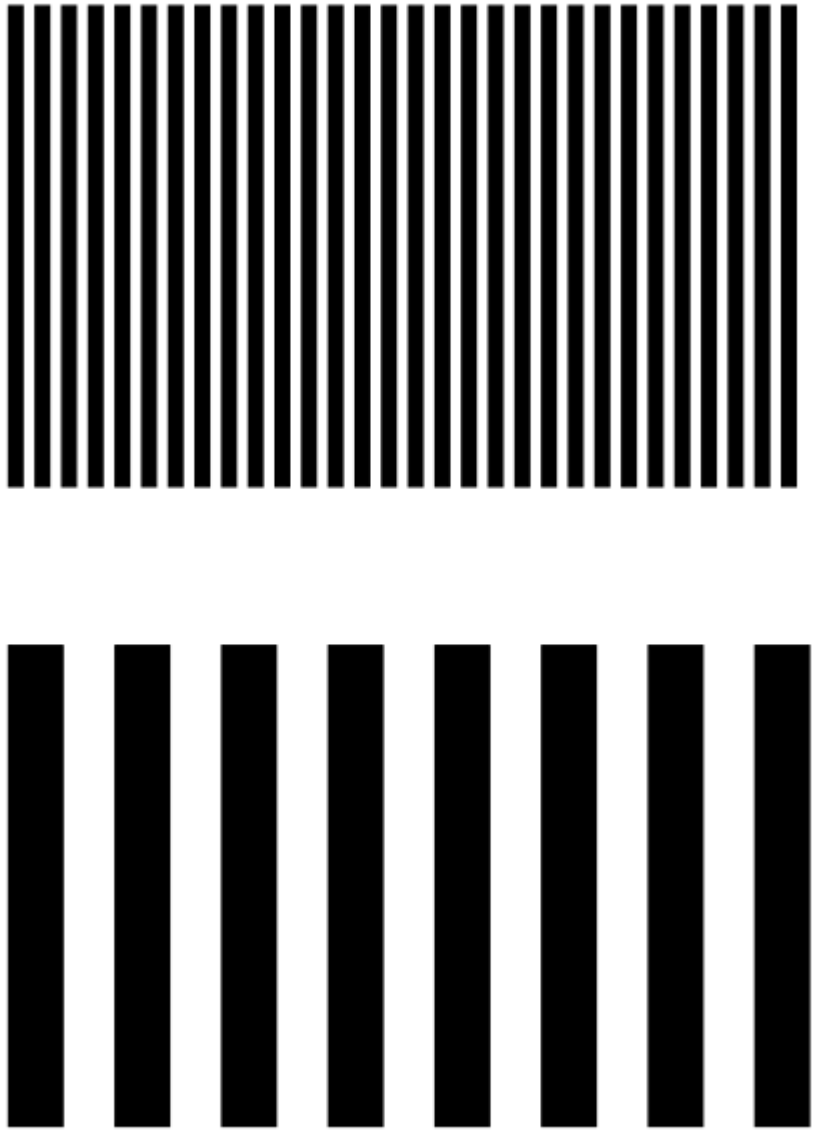
```
Paski(x_s, y_s, z_s)
{
   If(sin z_s > 0) return color0
   Else return color1
}
```



Przykład: 3D paski

- Paski z kontrolowaną szerokością

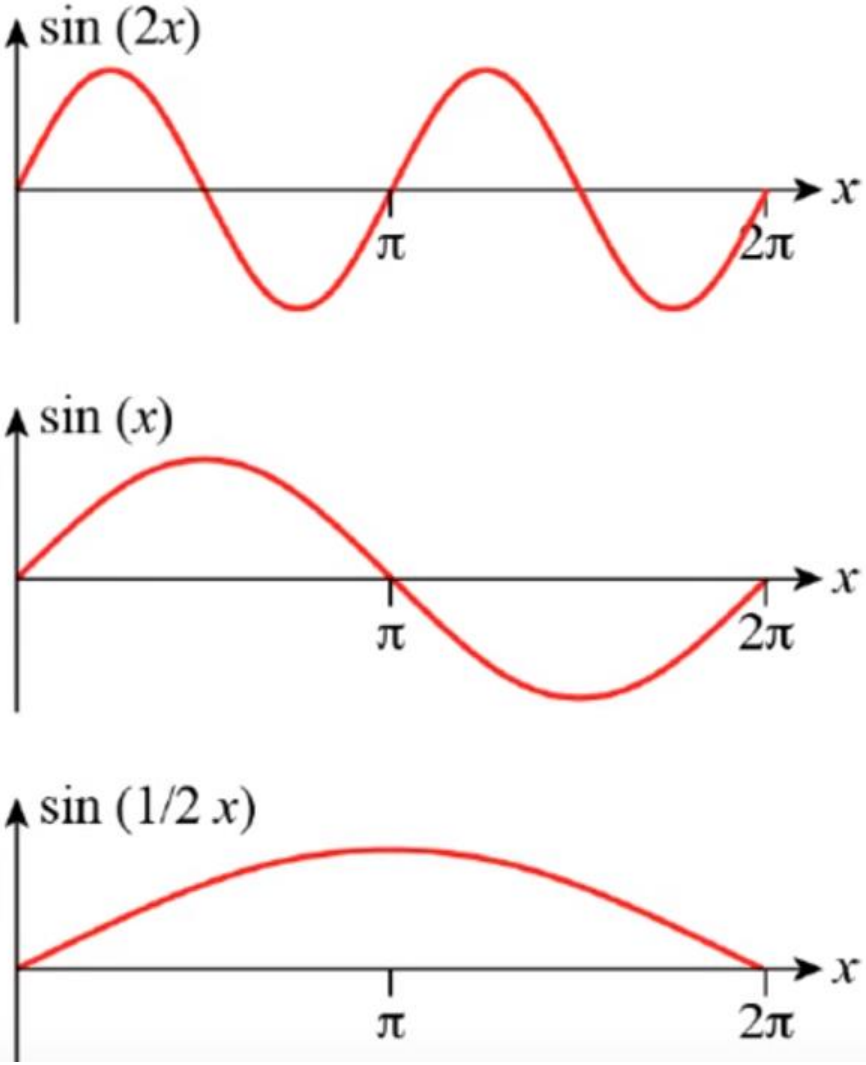
```
Paski(x_s, y_s, z_s, szerokość)
{
   If(sin (π * x_s / szerokość) > 0)
   return color0
   Else return color1
}
```



Przykład: 3D paski

- Paski z kontrolowaną szerokością

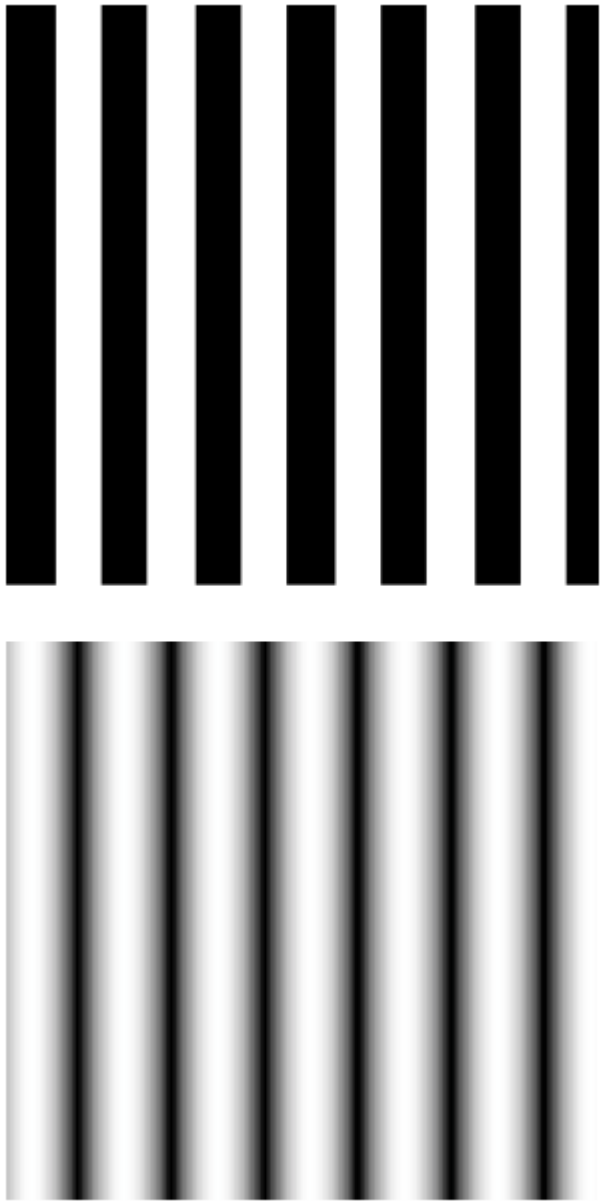
```
Paski(xs, ys, zs, szerokość)
{
   If(sin (π * xs / szerokość) > 0)
   return color0
   Else return color1
}
```



3D paski

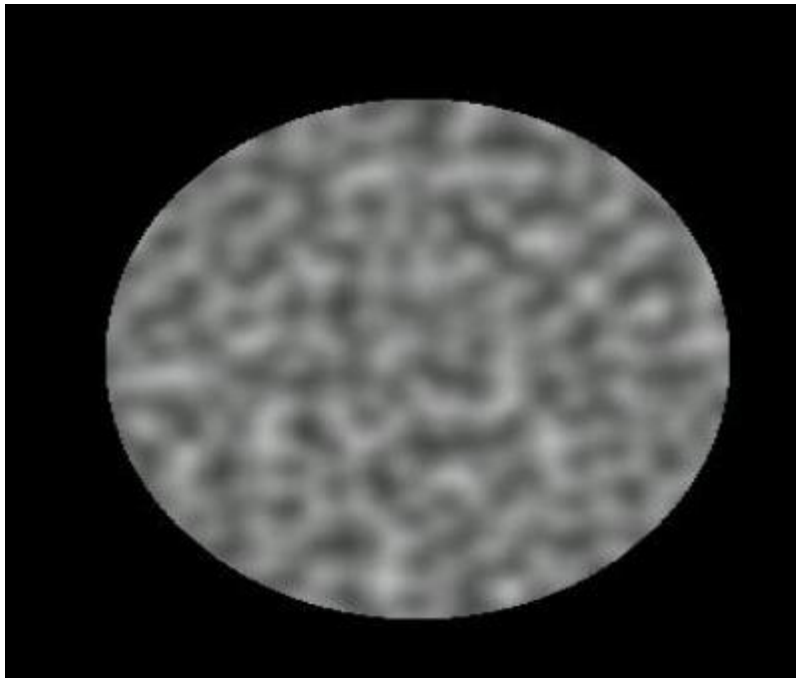
- Ciągła wariacja pomiędzy dwoma kolorami

```
Paski(x_s, y_s, z_s, szerokość)
{
   t = (1 + sin (π * x_s / szerokość)) / 2
   Return (1 - t) color0 + t color1
}
```



Szum Perlina

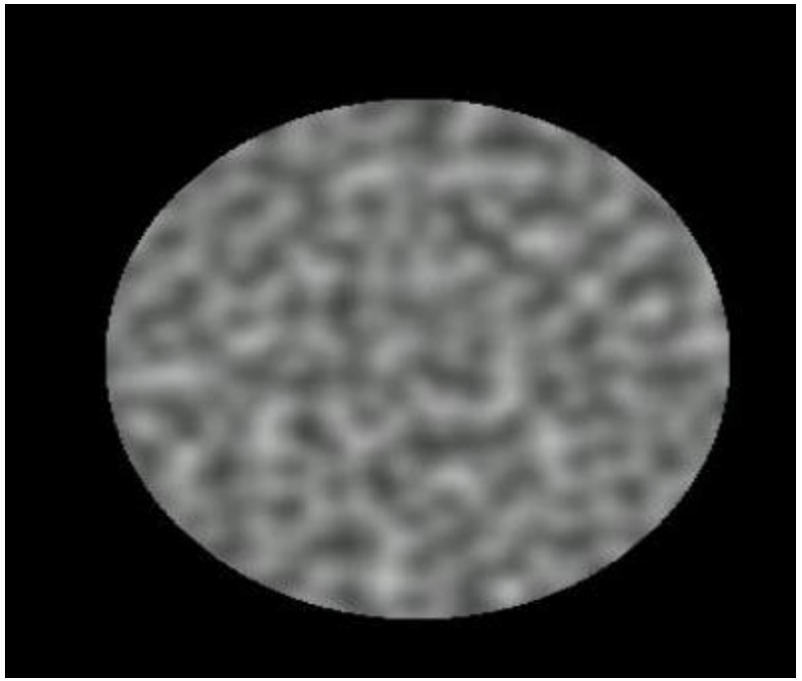
- Ważna komponenta tekstur proceduralnych żeby uzyskać wariacje w kolorach



Szum Perlina

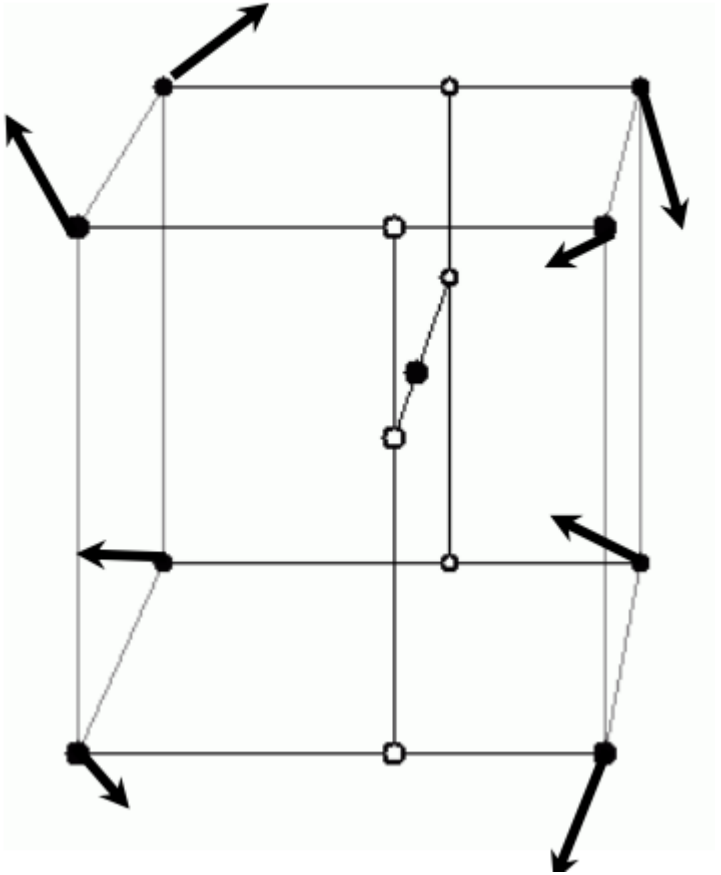
- Ważna komponenta tekstur proceduralnych żeby uzyskać wariacje w kolorach

Teren generowany za pomocy szumu Perlina - Minecraft

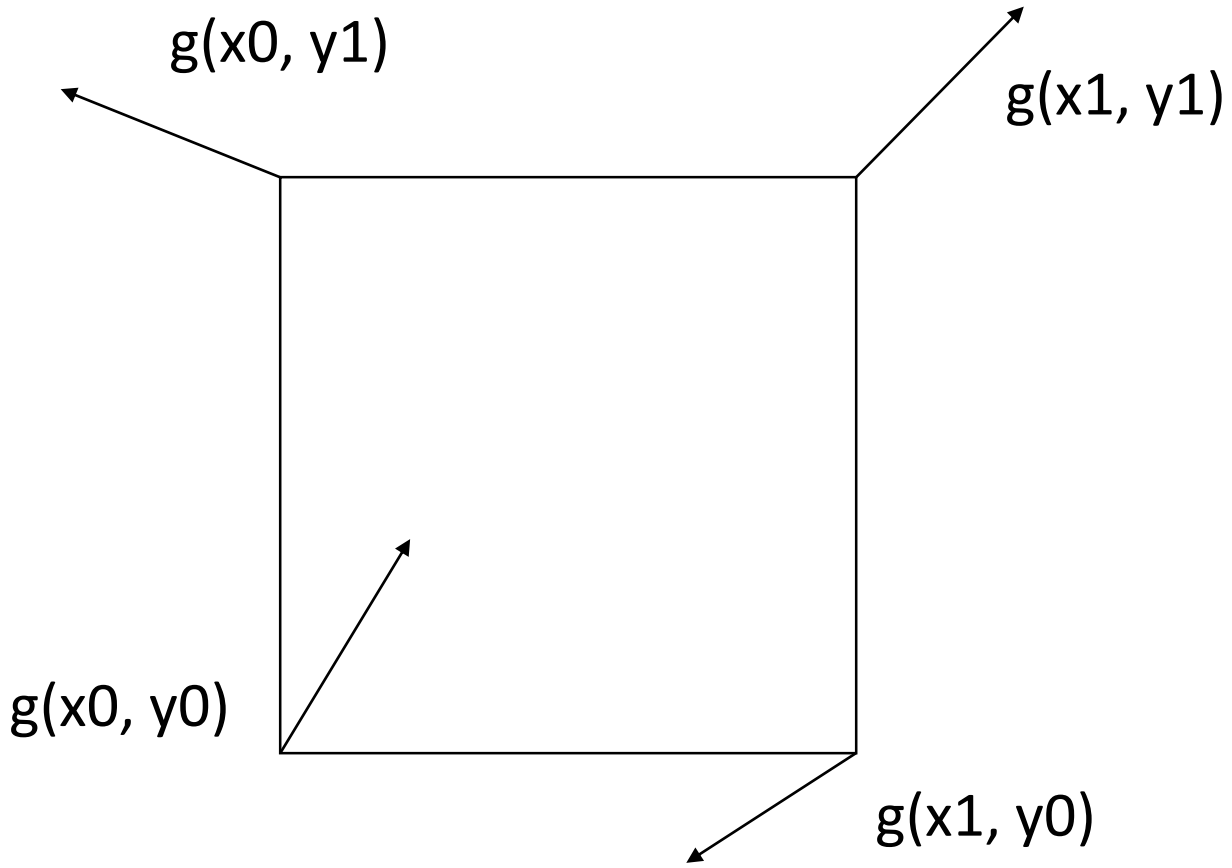


Szum Perlina

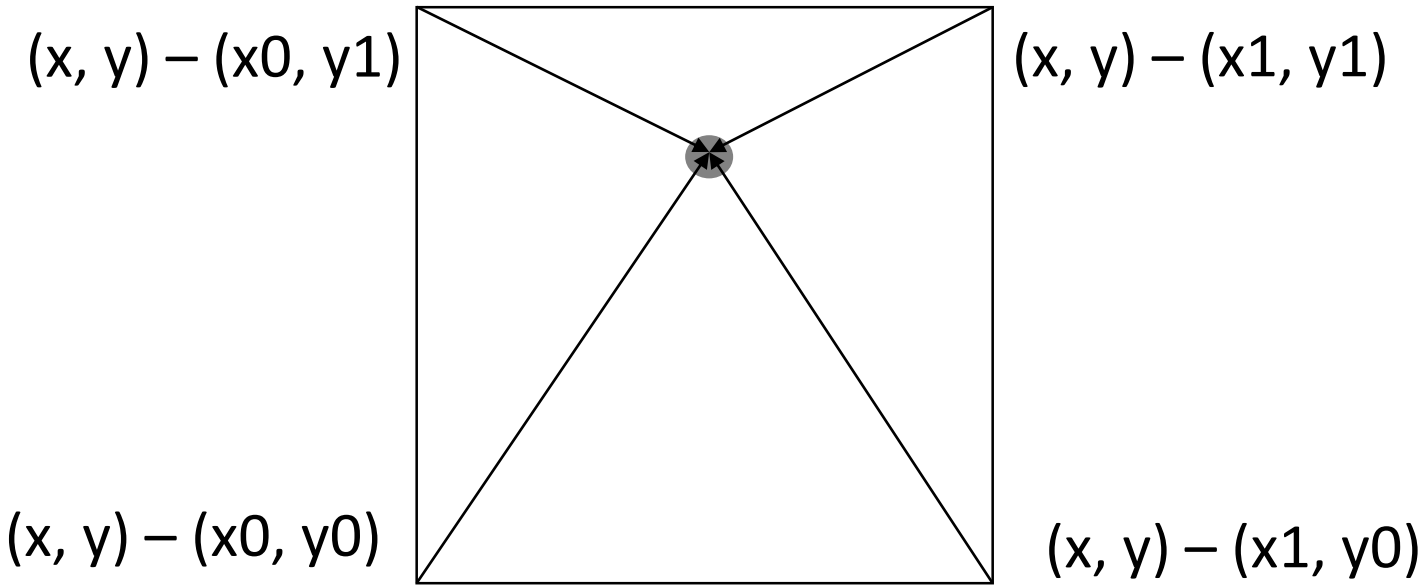
- Kubiczna siatka
- Wartość zerowa we wierzchołkach
- Pseudo-przypadkowa pochodna w wierzchołkach
- Funkcje sklejane interpolują wartości pozostałych punktów



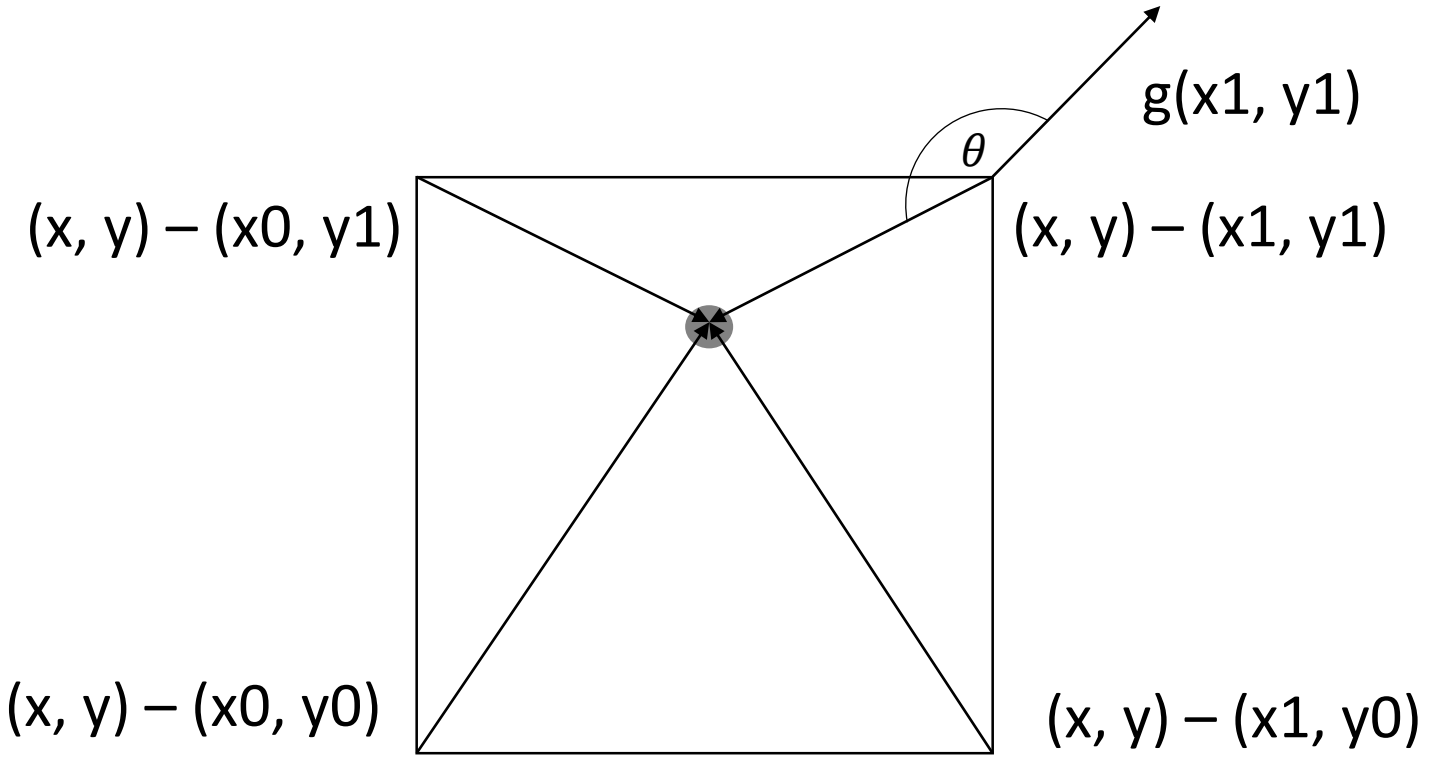
Pseudo-przypadkowe wektory gradientowe



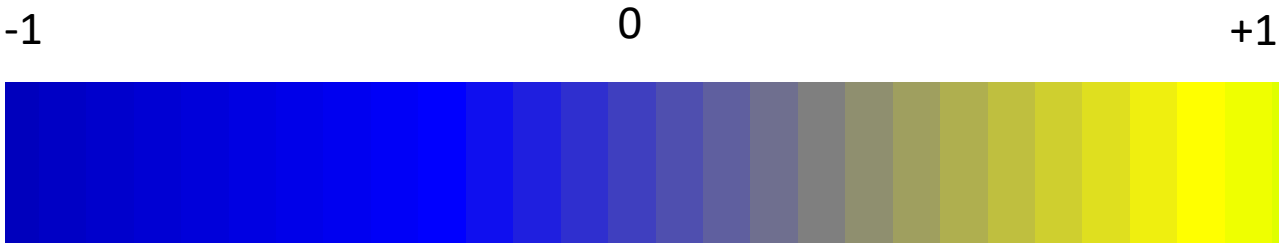
Wektory odległości (distance vectors)



Wektory odległości (distance vectors)

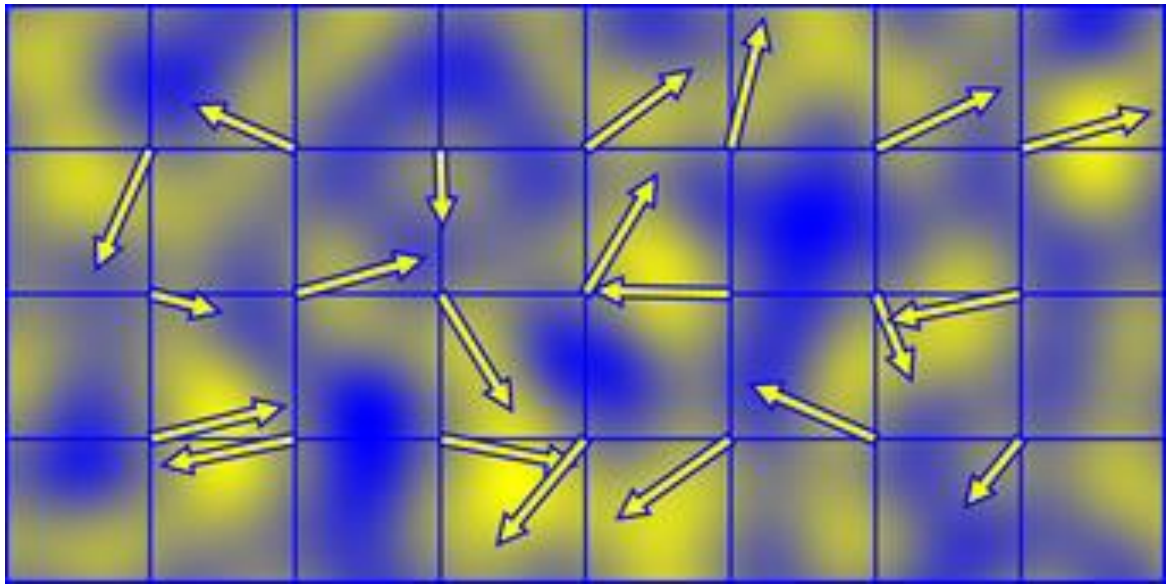


Mapa kolorów



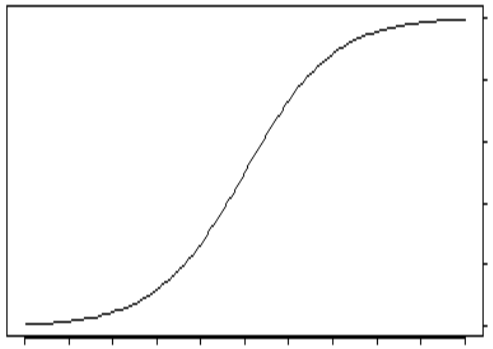
Iloczyn skalarny

- Obliczamy iloczyn skalarny **wektorów odległości** i **wektorów gradientowych**
- Interpolujemy wyniki żeby uzyskać uśrednianie



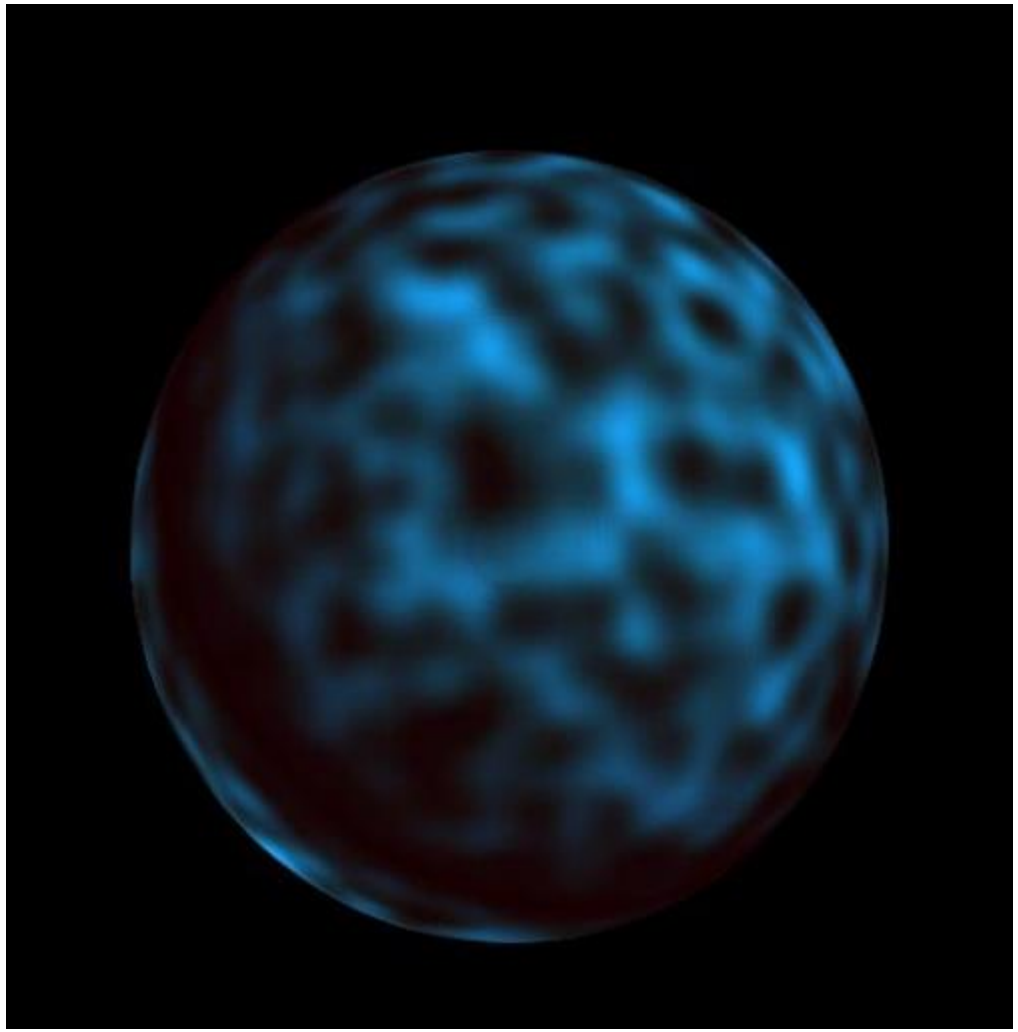
Algorytm w 3D

- Dla danego punktu wejściowego P
- Dla każdego z sąsiednich punktów S_i na siatce
  - Oblicz pseudo-przypadkowy wektor gradientowy g
  - Oblicz iloczyn skalarny g i PS_i
- Oblicz ważoną sumę wyników używając kubiczne wagi (fade function)
- Znajdź odpowiedny kolor z mapy kolorów

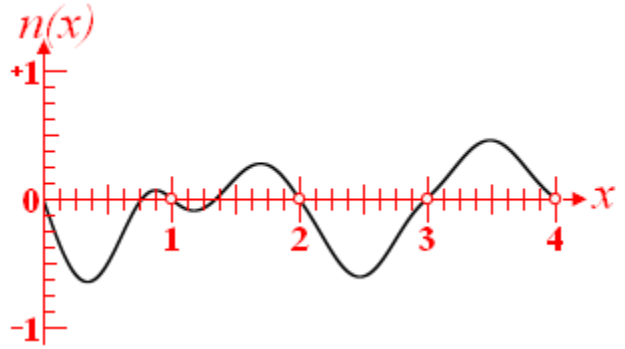


np. $6t^5 - 15t^4 + 10t^3$

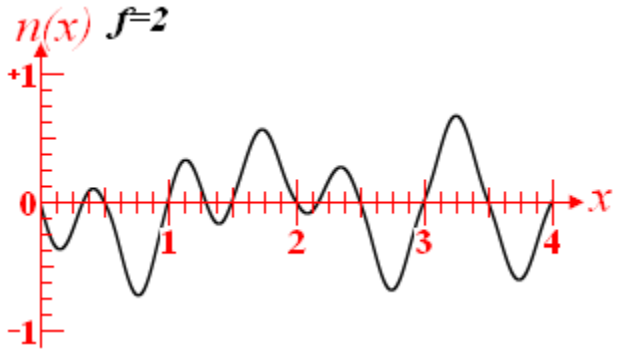
Szum Perlina



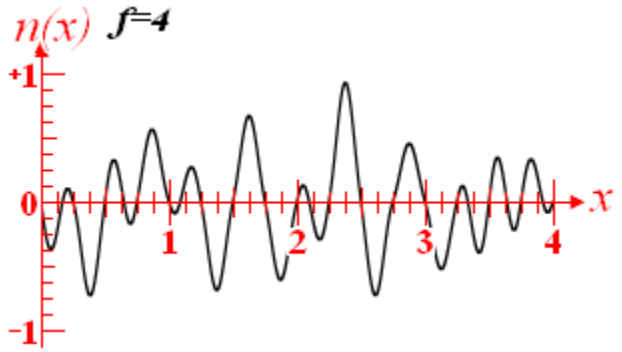
Częstotliwość Funkcji Szumu



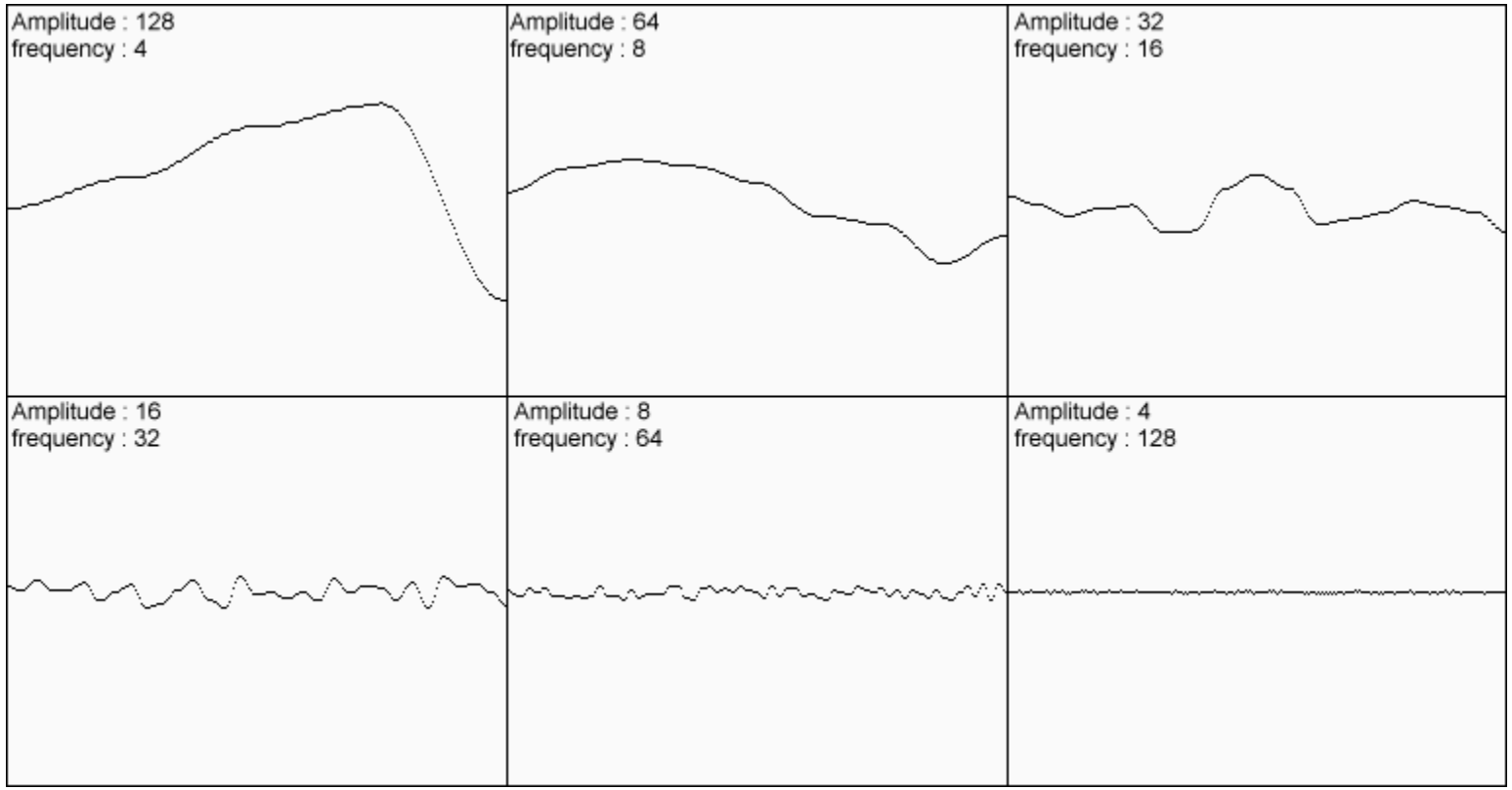
Częstotliwość Funkcji Szumu



Częstotliwość Funkcji Szumu

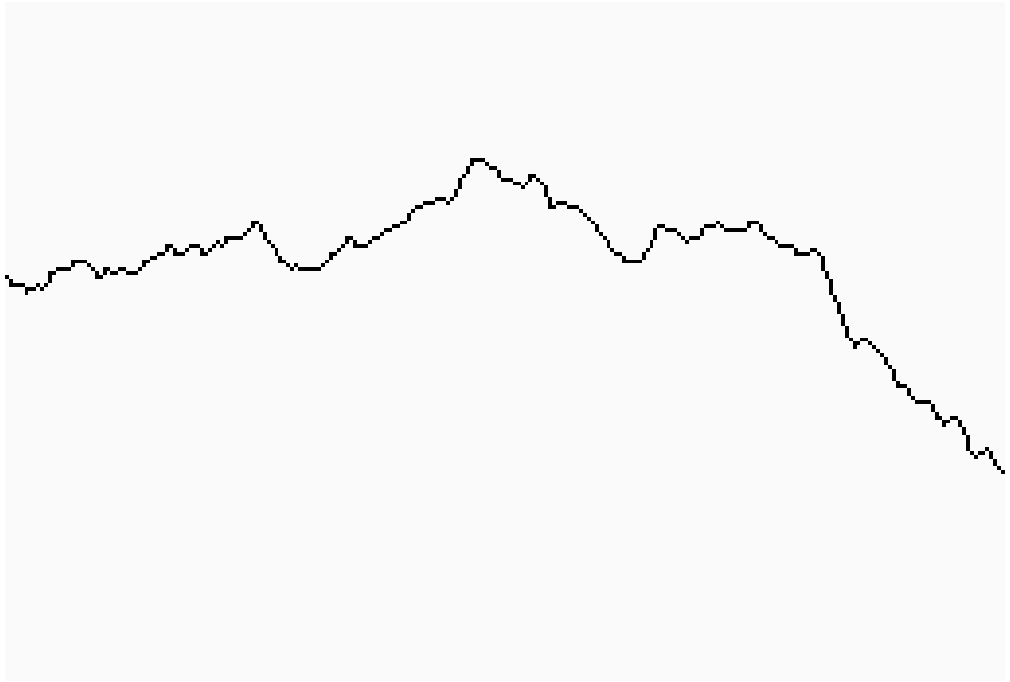


Szum Perlina - oktawy



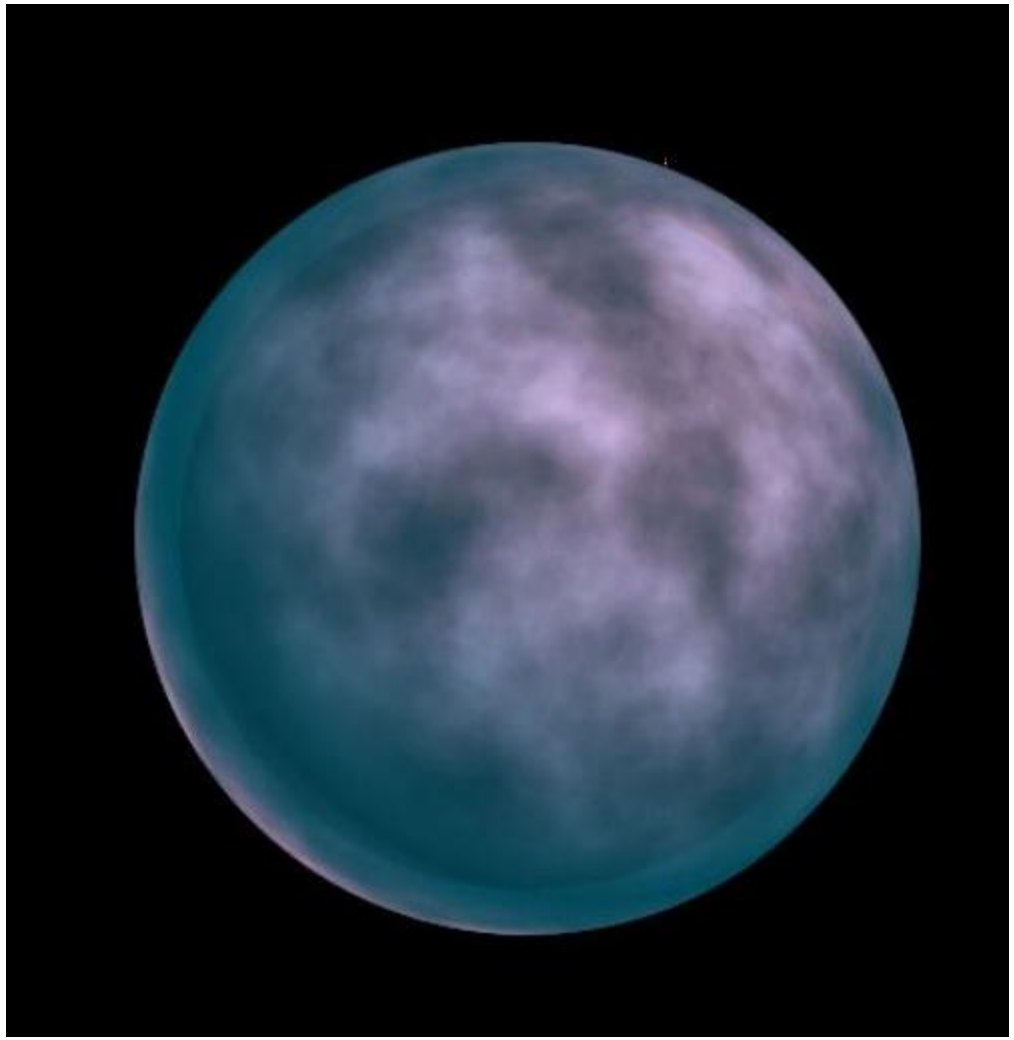
Szum Perlina - oktawy

- Suma sześciu oktaw:



Szum w kilku oktawach

- Każda oktawa o ma wagę 1/o



Absolutna wartość szumu

- abs(iloczyn skalarny g i Ps_i)

Sinus (absolutną wartość szumu)

- Szum Perlin przypominający marmur