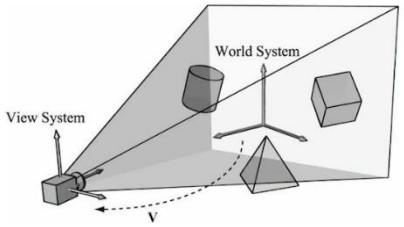
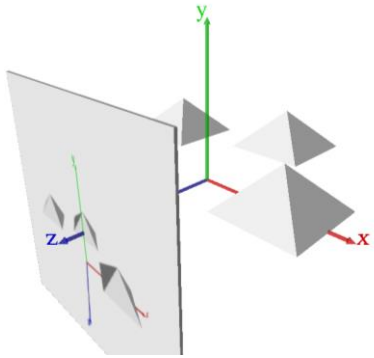
GRK 3

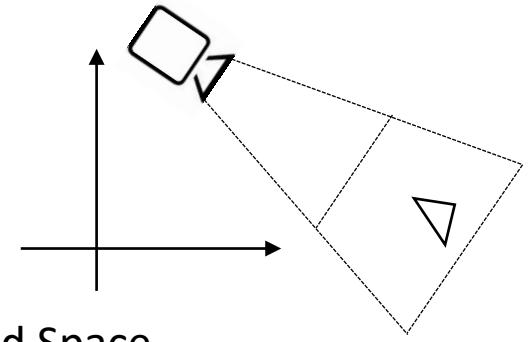
Dr Wojciech Palubicki

Potok graficzny

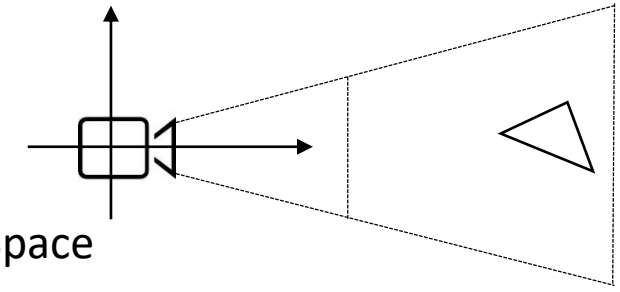




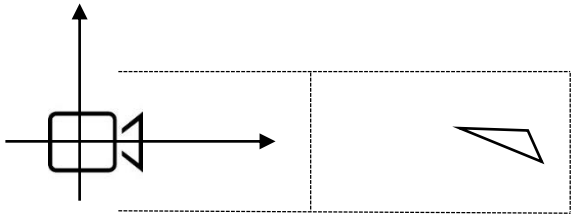
Potok graficzny



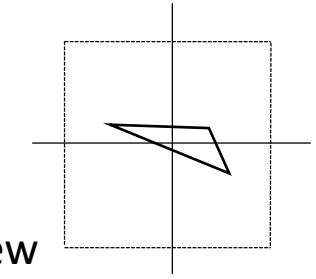
World Space



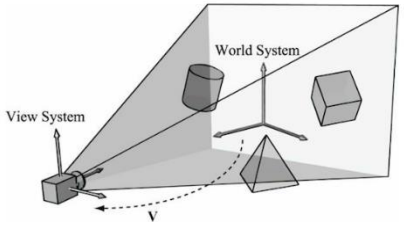
View Space

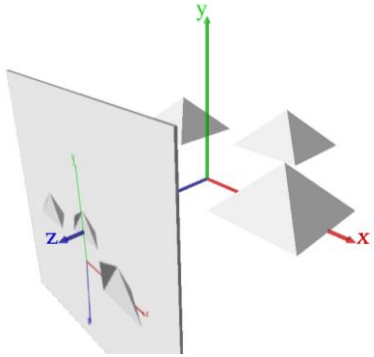


Orthographic view

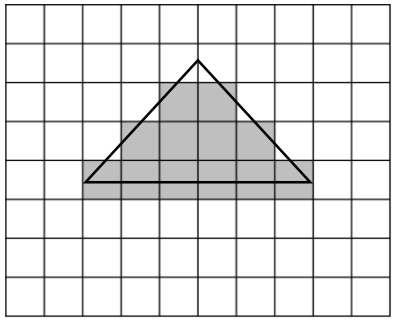


Canonincal view

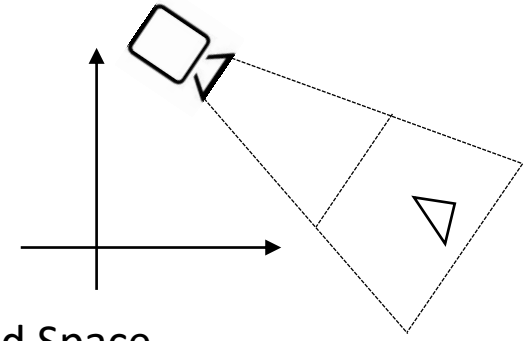




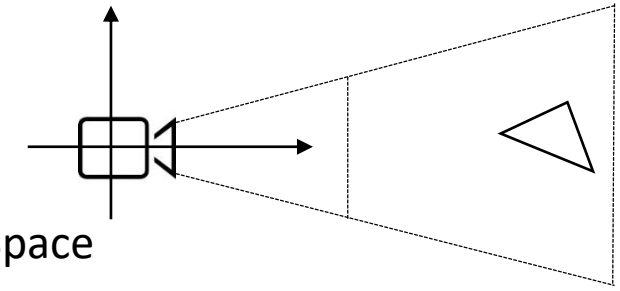
Window space



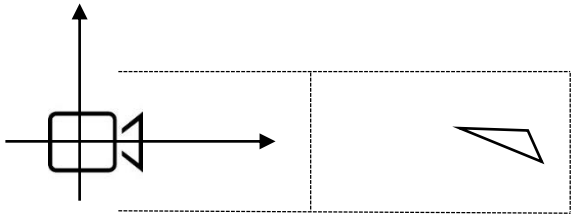
Potok graficzny



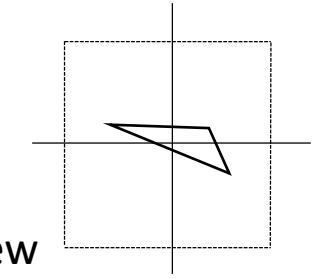
World Space



View Space

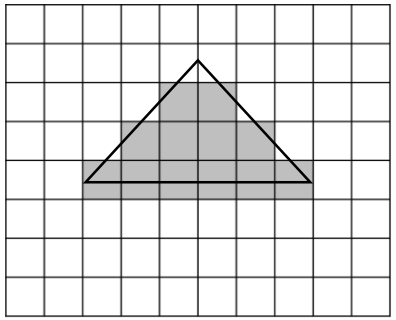


Orthographic view



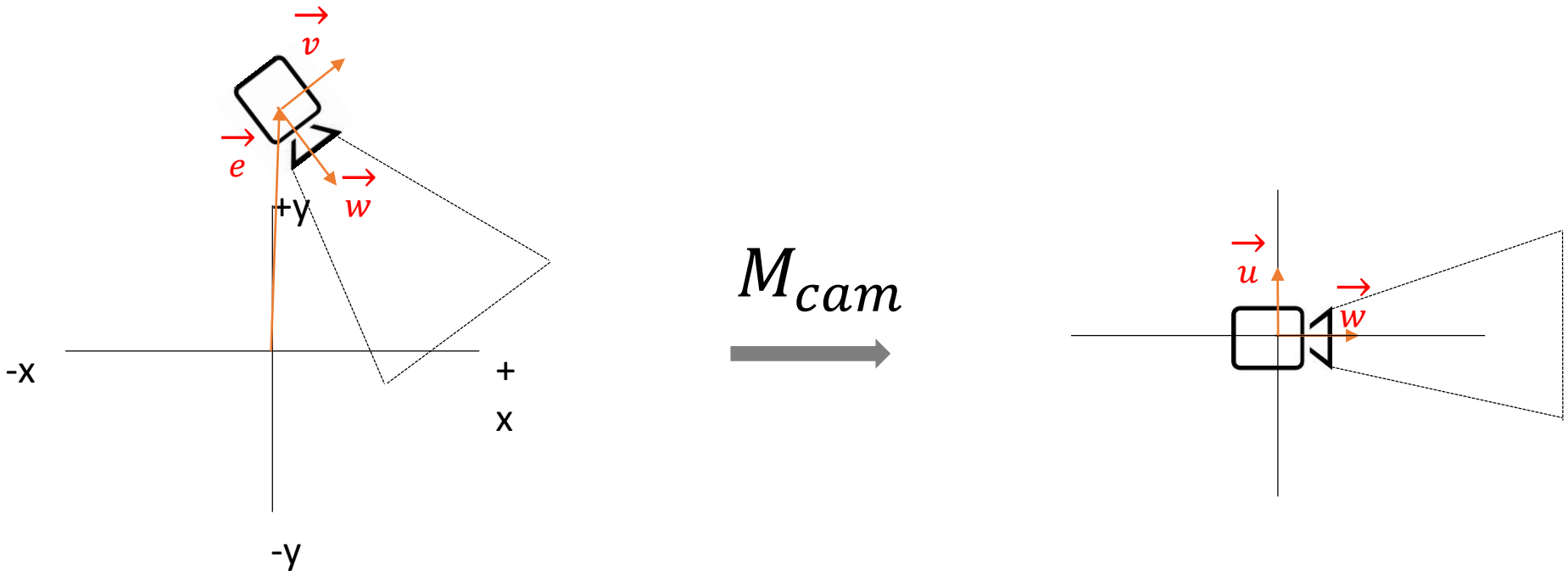
Canonincal view

Window space



- Potok graficzny z **ortograficznym** rzutem:
- $M_{vp}M_{ort}M_{cam}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$
- Potok graficzny z **perspektywicznym** rzutem:
- $M_{vp}M_{per}M_{cam}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

Wyrównanie systemów współrzędnych

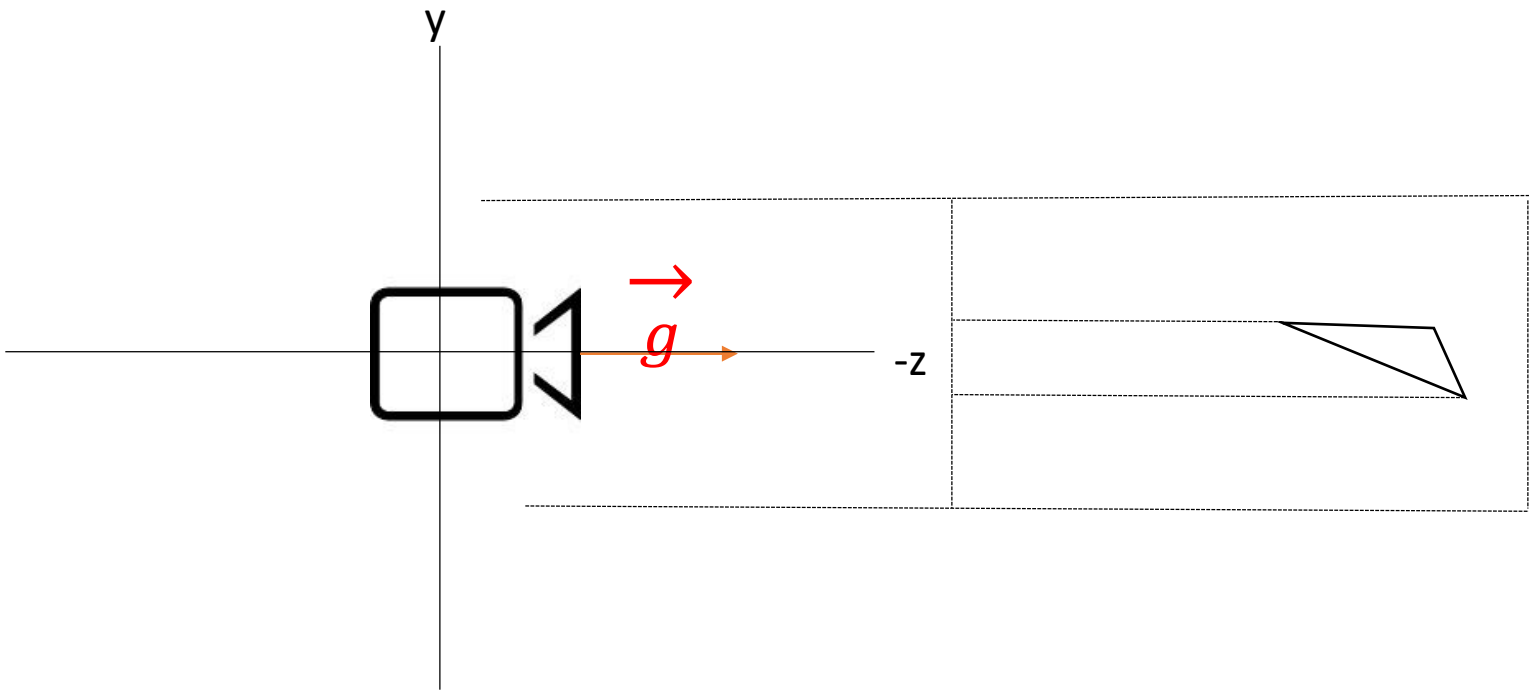


M_{cam}

- Macierz transformacji z world space na view space jest wtedy:

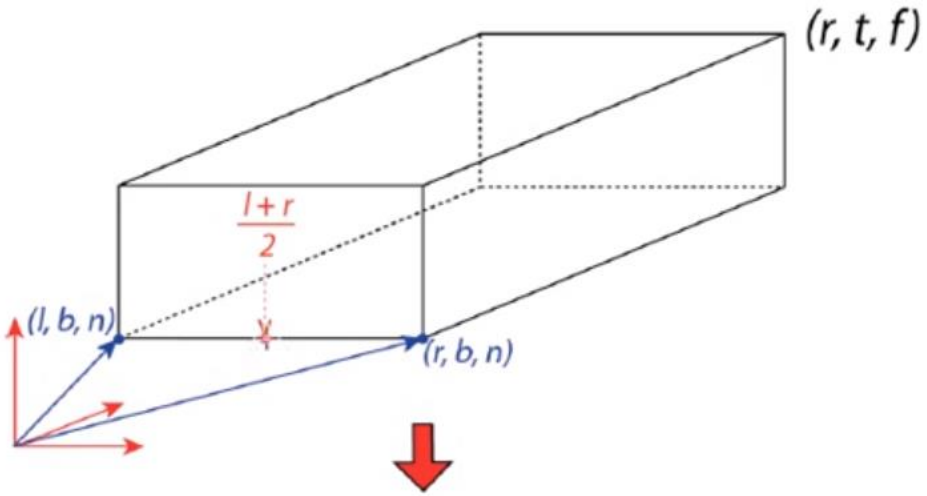
$$M_{cam} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

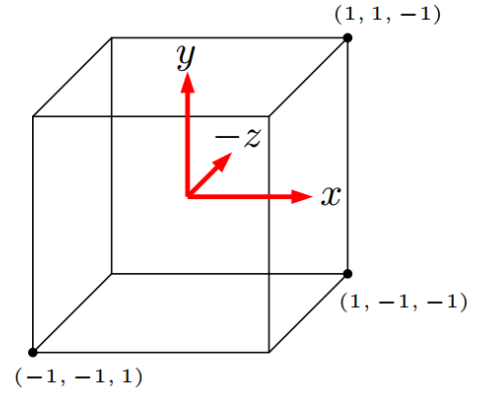
Rzutowanie równoległe



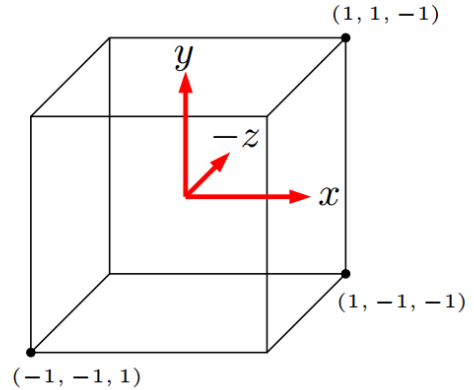
Ortograficzna bryła widzenia

- $M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{b+t}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

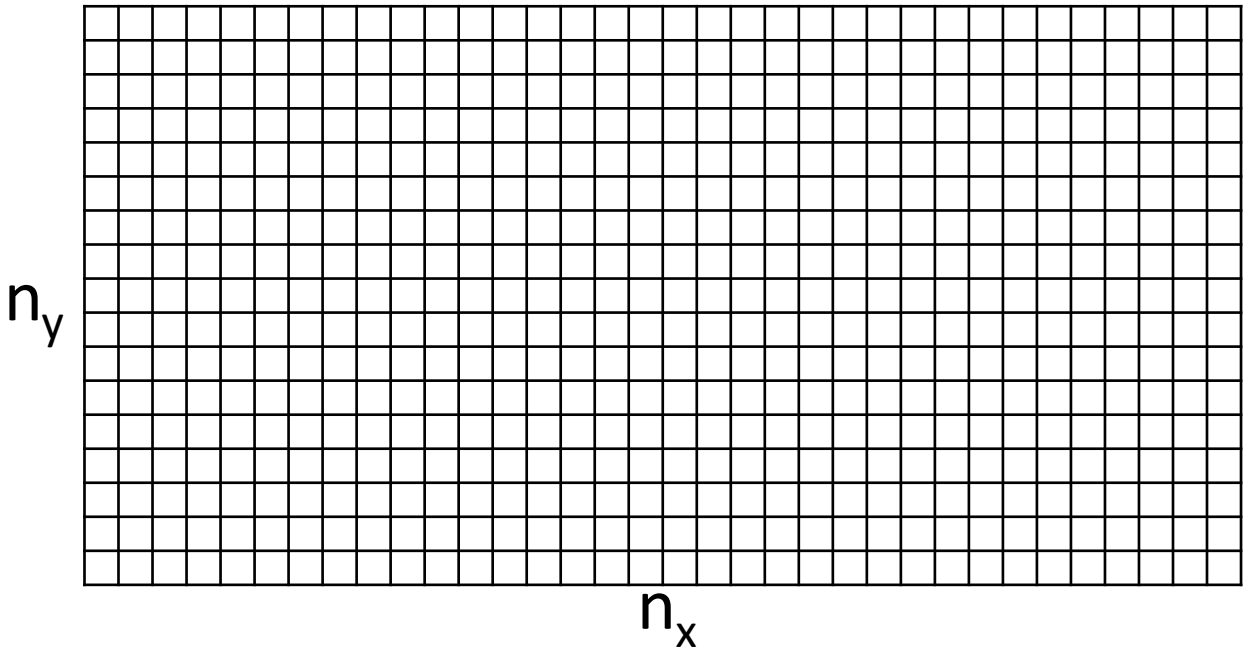




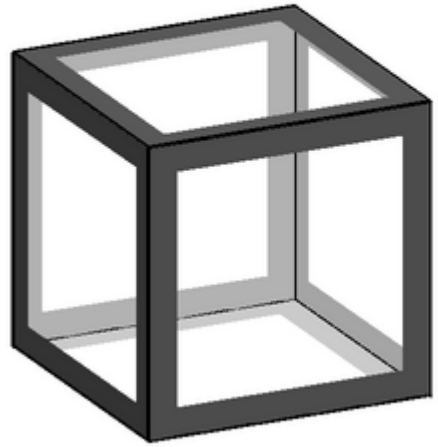
Przekształcenie bryły kanonicznej



$$M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x}{2} - \frac{1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y}{2} - \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



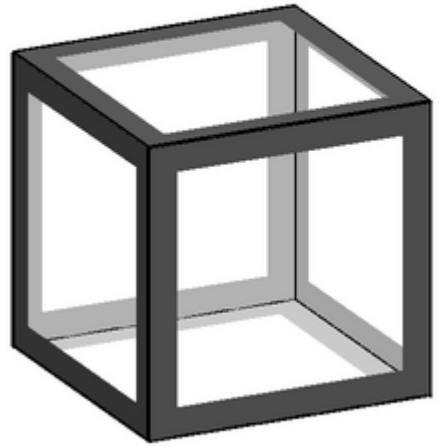
Rzutowania



Ortograficzne rzutowanie

Rzutowania

Perspektywicznie rzutowanie

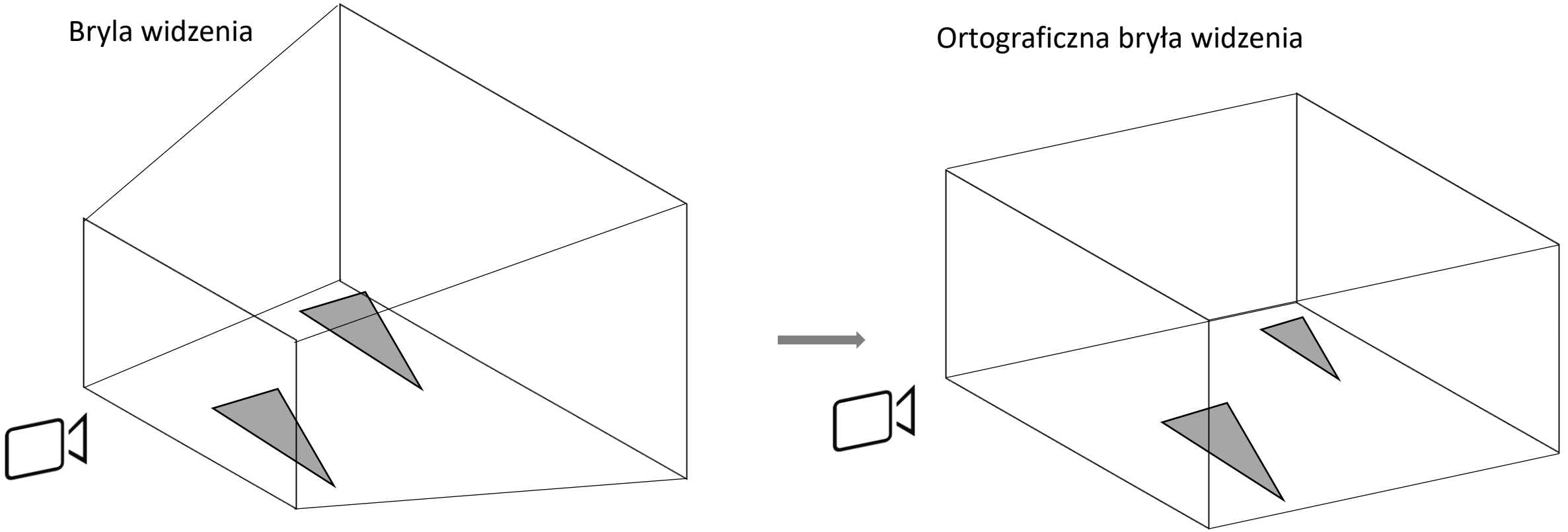


Ortograficzne rzutowanie

Macierz P

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n+f-\frac{nf}{z} \\ 1 \end{bmatrix}$$

Transformacja bryły widzenia



System współrzędnych jednorodnych

Nasze multiplikacje macierzami do liniowe transformacje, tzn. tylko mogą wygenerować wartości jak np.:

$$x' = a_1 x + b_1 y + c_1 z$$

Wprowadzenie współrzędnych jednorodnych umożliwia reprezentacje punktów (x, y, z) jako (x, y, z, 1) i umożliwia nam skorzystanie z afinicznych transformacji, np.:

$$x' = a_1 x + b_1 y + c_1 z + d_1$$

System współrzędnych jednorodnych

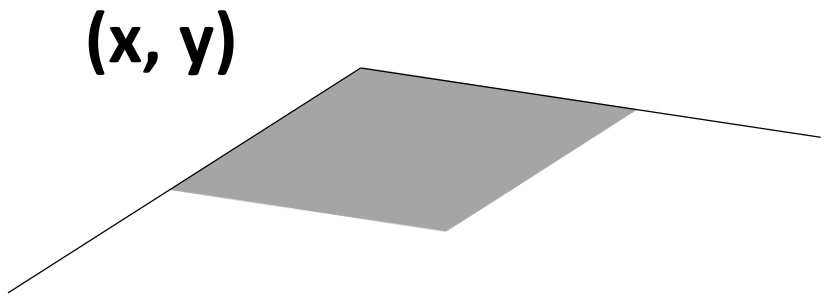
- We współrzędnych jednorodnych wektor:

(x, y, z, 1) reprezentuje punkt **(x, y, z)**

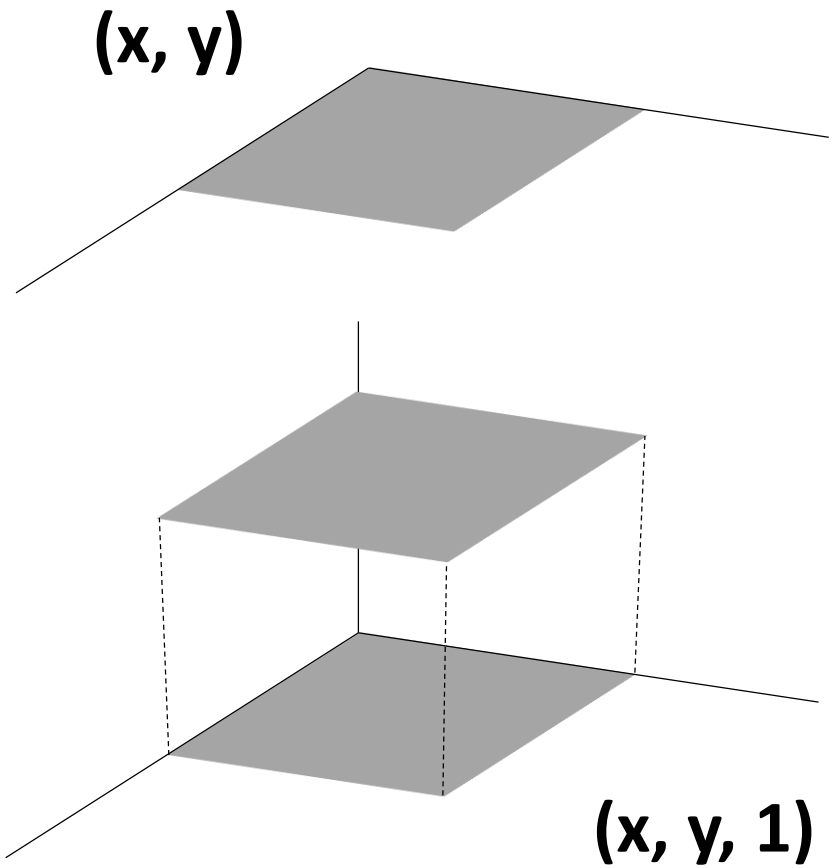
Teraz wprowadzamy możliwość reprezentacji punktów za pomocy:

(x, y, z, w) co daje nam punkt **(x/w, y/w, z/w)**

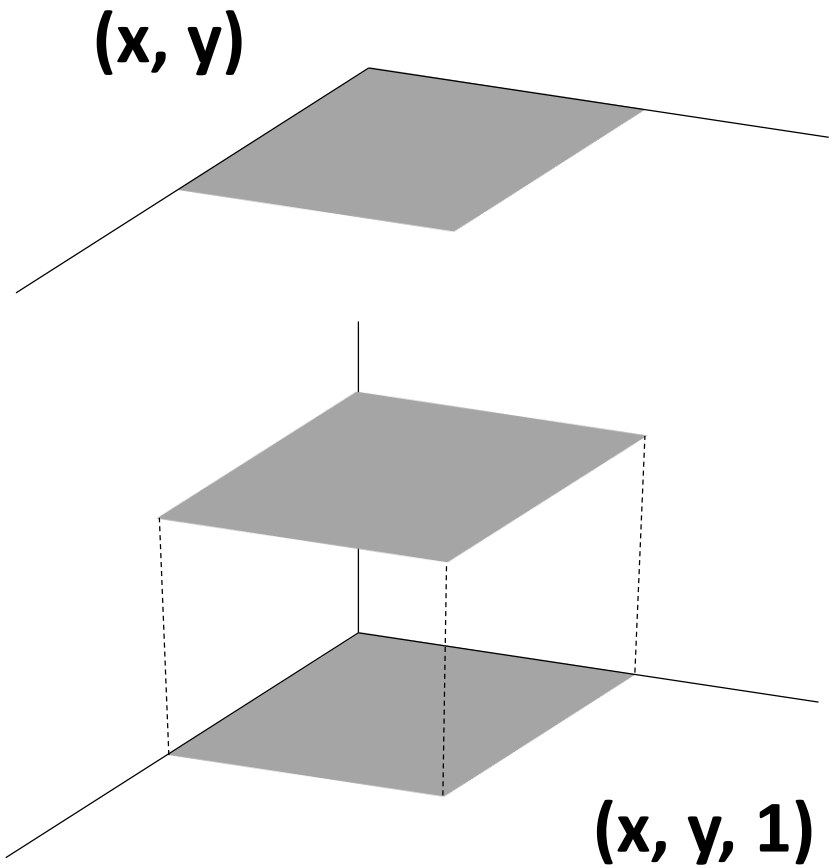
Współrzędne jednorodne

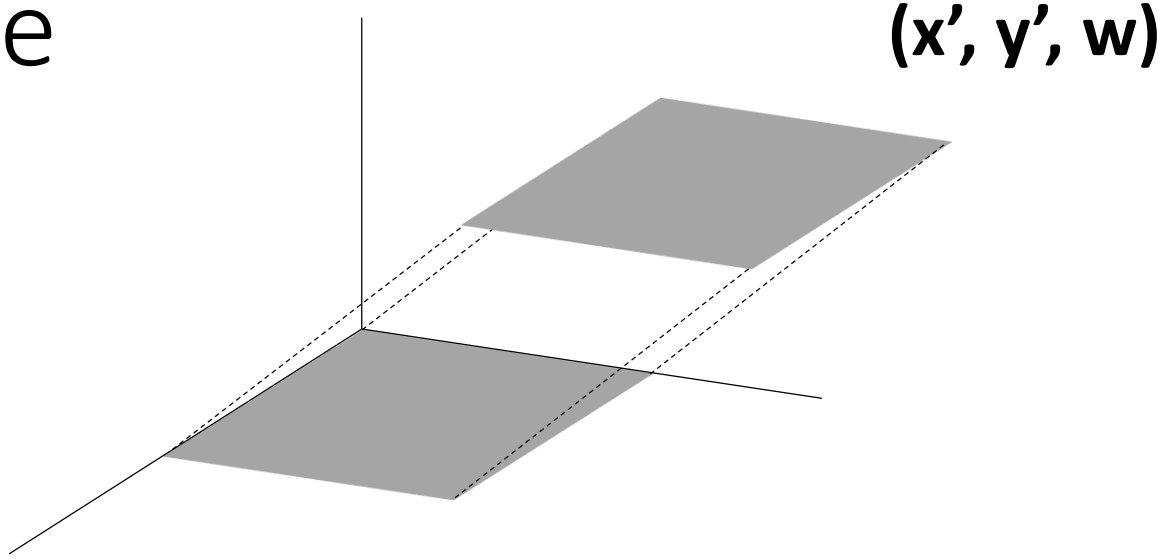


Współrzędne jednorodne

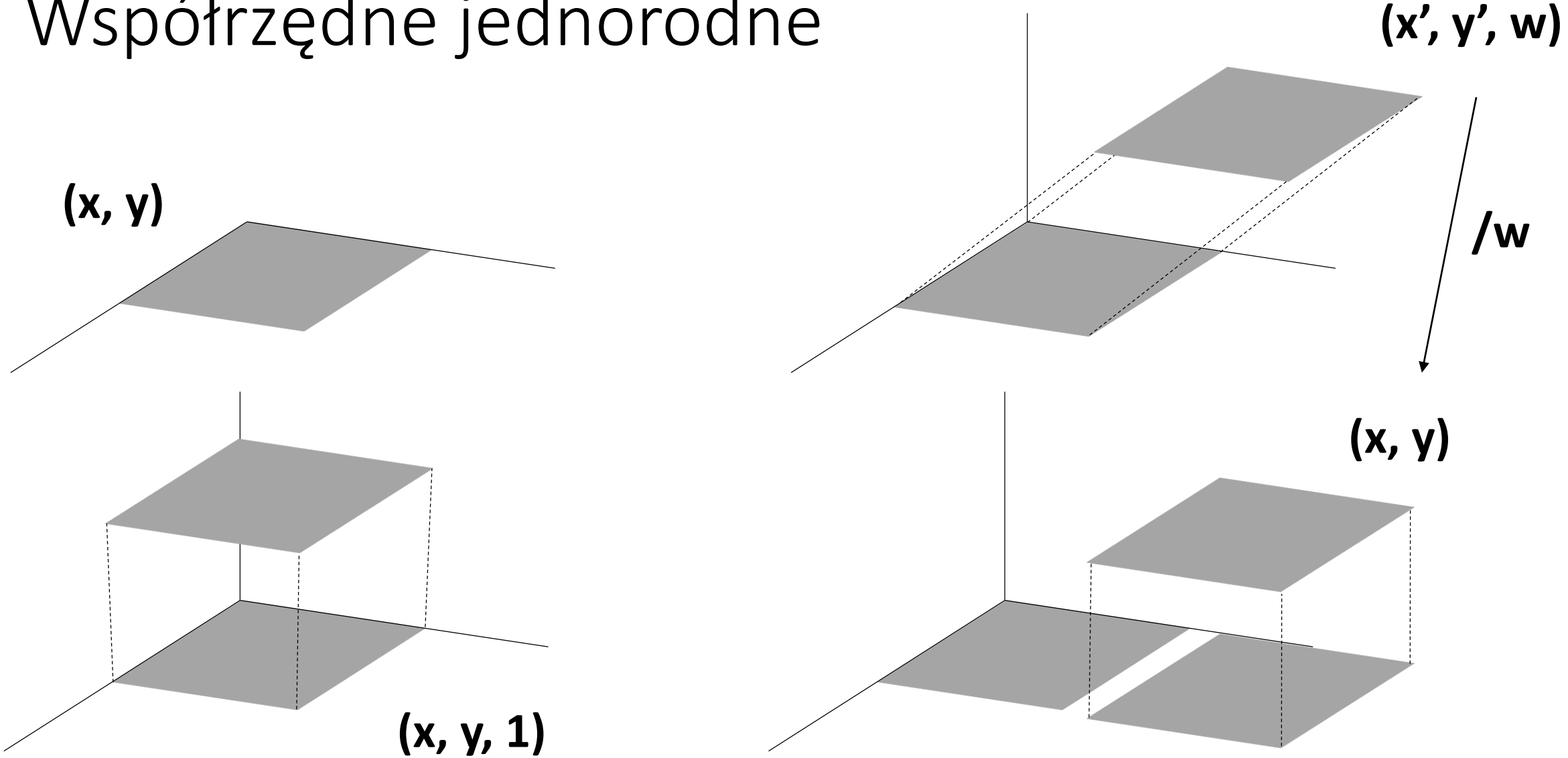


Współrzędne jednorodne

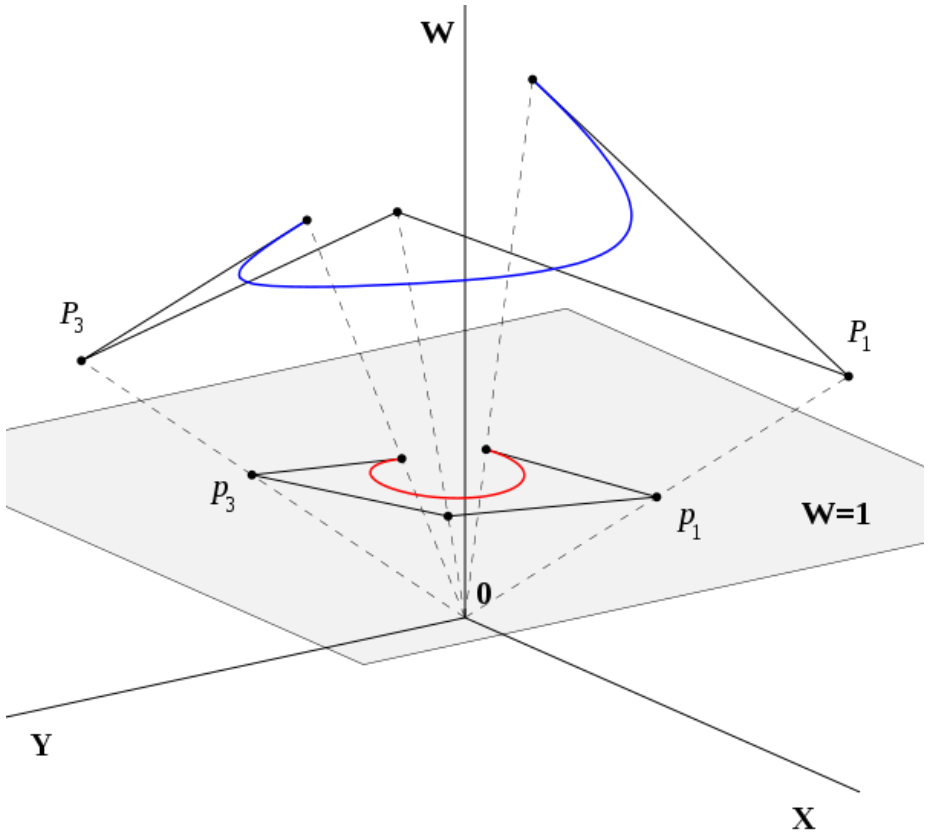




Współrzędne jednorodne



August Möbius



https://en.wikipedia.org/wiki/Homogeneous_coordinates

System współrzędnych jednorodnych

- Wtedy transformacja macierzowa staje się:

- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & T_x \\ a_2 & b_2 & c_2 & T_y \\ a_3 & b_3 & c_3 & T_z \\ a_4 & b_4 & c_4 & w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} a_1x + b_1y + c_1z + T_x \\ a_2x + b_2y + c_2z + T_y \\ a_3x + b_3y + c_3z + T_z \\ a_4x + b_4y + c_4z + w \end{bmatrix}$$

- $$\begin{bmatrix} a_1x + b_1y + c_1z + T_x \\ a_2x + b_2y + c_2z + T_y \\ a_3x + b_3y + c_3z + T_z \\ a_4x + b_4y + c_4z + w \end{bmatrix} \xrightarrow{homogenizacja} \begin{bmatrix} \frac{a_1x+b_1y+c_1z+T_x}{a_4x+b_4y+c_4z+w} \\ \frac{a_2x+b_2y+c_2z+T_y}{a_4x+b_4y+c_4z+w} \\ \frac{a_3x+b_3y+c_3z+T_z}{a_4x+b_4y+c_4z+w} \\ 1 \end{bmatrix}$$

Transformacja bryły widzenia

Kalkulacja:

$$\begin{bmatrix} \frac{a_1x + b_1y + c_1z + T_x}{a_4x + b_4y + c_4z + w} \\ \frac{a_2x + b_2y + c_2z + T_y}{a_4x + b_4y + c_4z + w} \\ \frac{a_3x + b_3y + c_3z + T_z}{a_4x + b_4y + c_4z + w} \\ 1 \end{bmatrix}$$

Cel:

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

Transformacja bryły widzenia

Kalkulacja:

$$\begin{bmatrix} \frac{a_1x + b_1y + c_1z + T_x}{a_4x + b_4y + c_4z + w} \\ \frac{a_2x + b_2y + c_2z + T_y}{a_4x + b_4y + c_4z + w} \\ \frac{a_3x + b_3y + c_3z + T_z}{a_4x + b_4y + c_4z + w} \\ 1 \end{bmatrix}$$

Cel:

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Czwarty wiersz

Transformacja bryły widzenia

Kalkulacja:

$$\begin{bmatrix} \frac{a_1x + b_1y + c_1z + T_x}{a_4x + b_4y + c_4z + w} \\ \frac{a_2x + b_2y + c_2z + T_y}{a_4x + b_4y + c_4z + w} \\ \frac{a_3x + b_3y + c_3z + T_z}{a_4x + b_4y + c_4z + w} \\ 1 \end{bmatrix}$$

Cel:

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Czwarty wiersz

Transformacja bryły widzenia

Kalkulacja:

$$\begin{bmatrix} \frac{a_1x + b_1y + c_1z + T_x}{a_4x + b_4y + c_4z + w} \\ \frac{a_2x + b_2y + c_2z + T_y}{a_4x + b_4y + c_4z + w} \\ \frac{a_3x + b_3y + c_3z + T_z}{a_4x + b_4y + c_4z + w} \\ 1 \end{bmatrix}$$

Cel:

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Czwarty wiersz

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Pierwszy i drugi wiersz

Transformacja bryły widzenia

Kalkulacja:

$$\begin{bmatrix} \frac{a_1x + b_1y + c_1z + T_x}{a_4x + b_4y + c_4z + w} \\ \frac{a_2x + b_2y + c_2z + T_y}{a_4x + b_4y + c_4z + w} \\ \frac{a_3x + b_3y + c_3z + T_z}{a_4x + b_4y + c_4z + w} \\ 1 \end{bmatrix}$$

Cel:

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Czwarty wiersz

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Pierwszy i drugi wiersz

Co z trzecim wierszem?

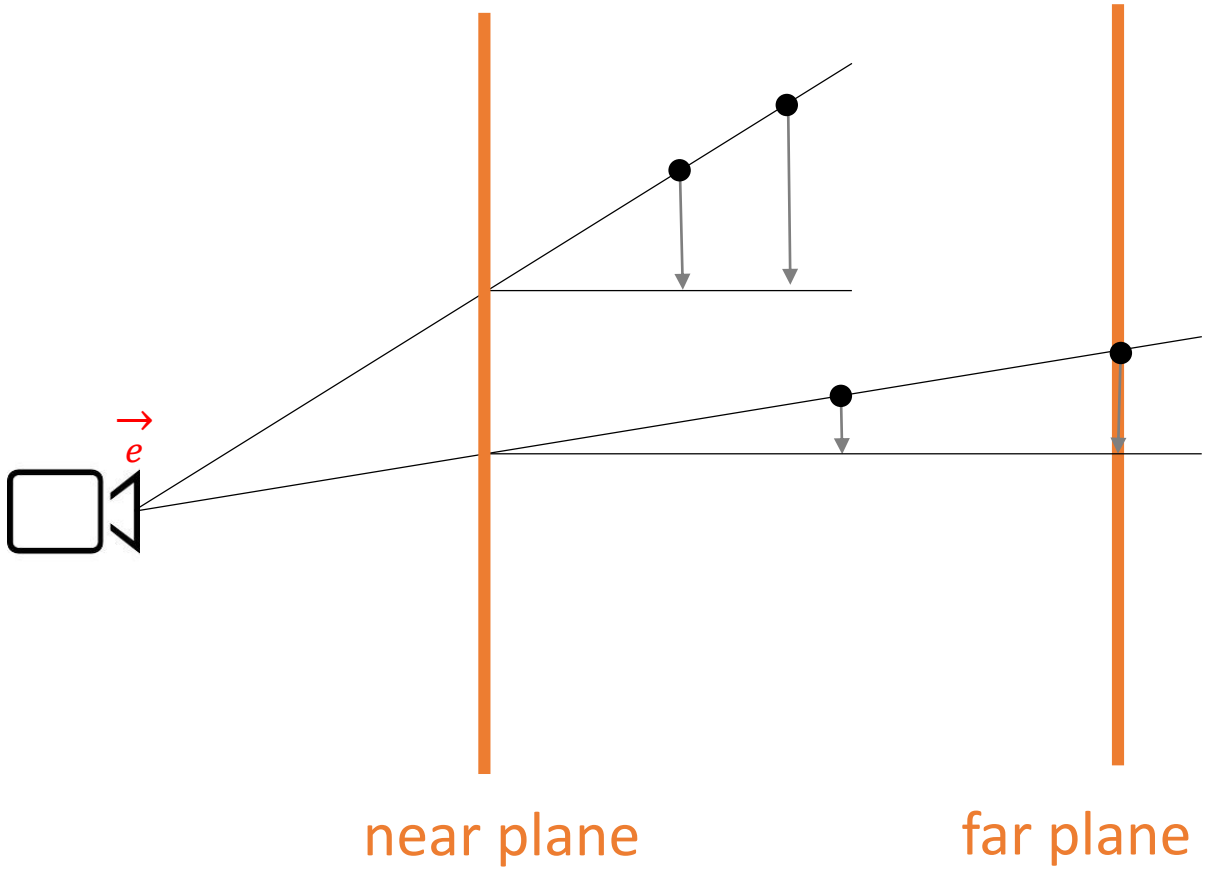
$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \\ 1 \end{bmatrix}$$

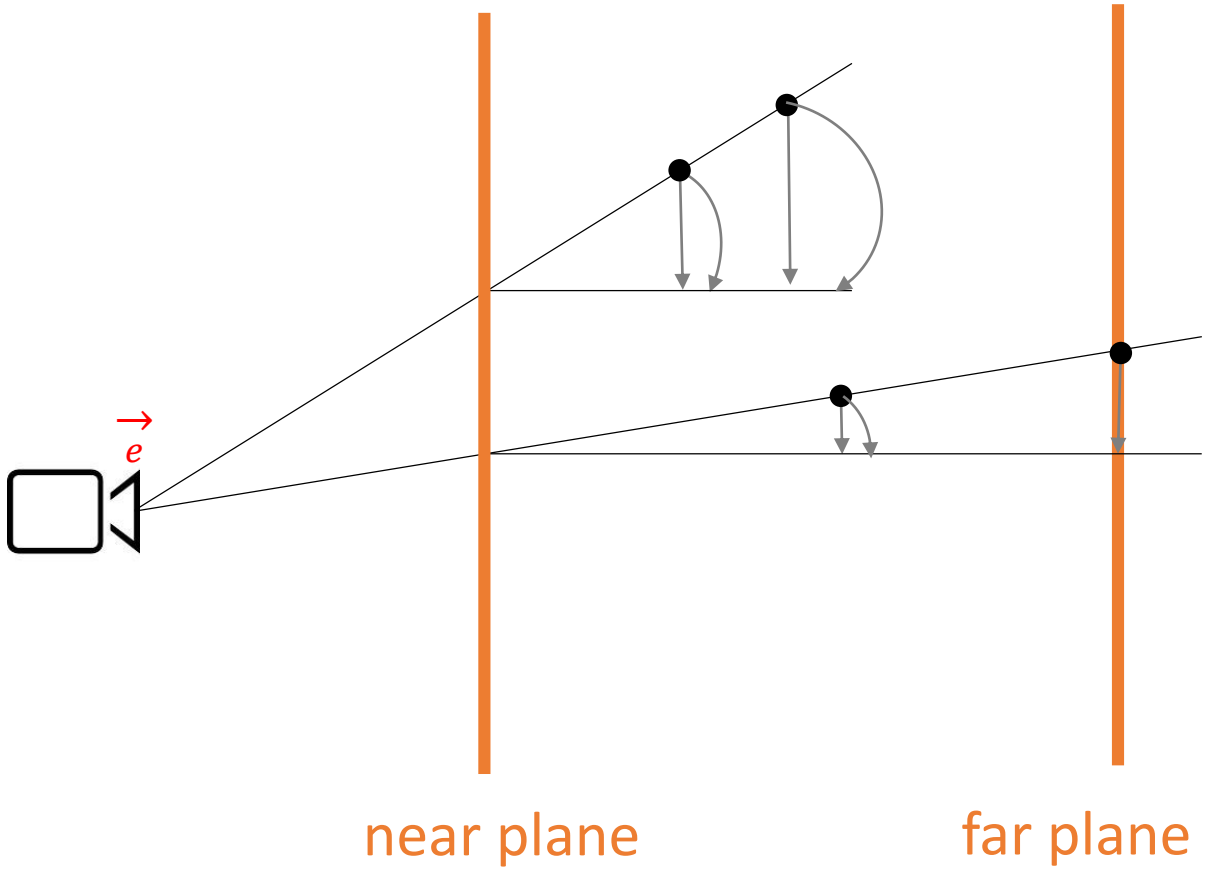
Macierz P

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n+f-\frac{nf}{z} \\ 1 \end{bmatrix}$$

Transformacja bryły widzenia

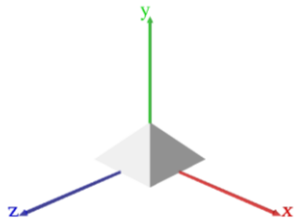


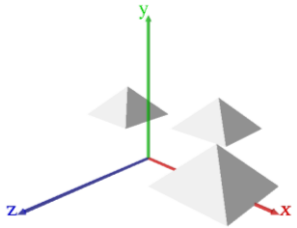
Transformacja bryły widzenia

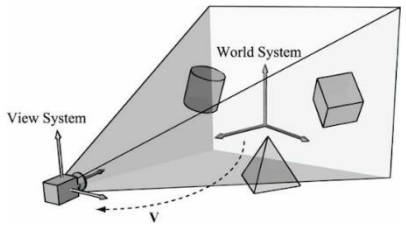


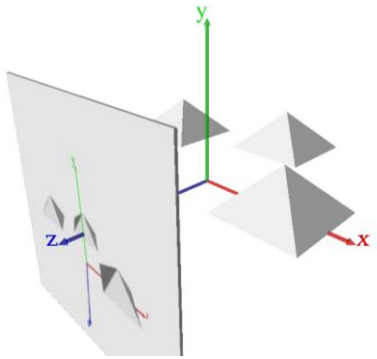
Uproszczony Potok Graficzny (Rendering)

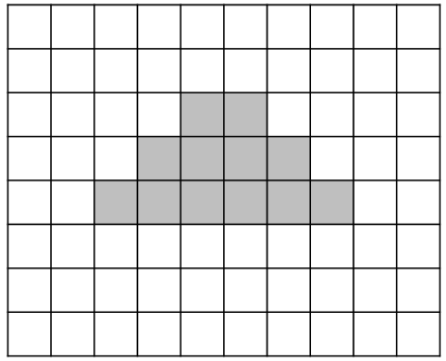
Model Matrix → View Matrix → Projection Matrix → Viewport Transform







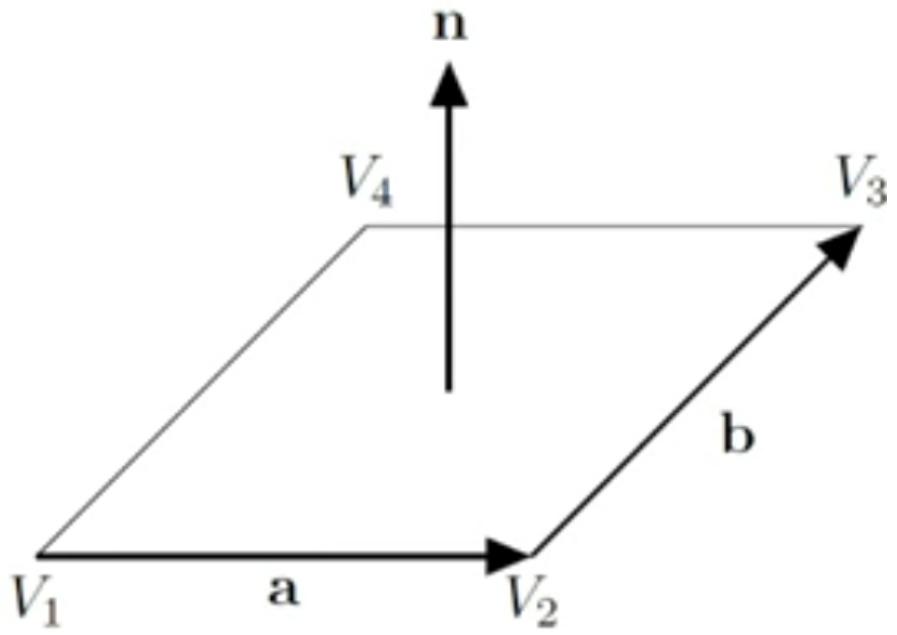




Object Space | World Space | View Space | Clip Space | Screen Space

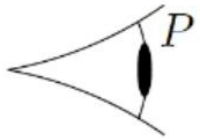
Wektor normalny

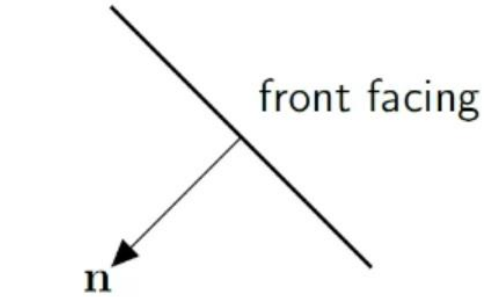
- **Wektor normalny** jest skierowany **prostopadle** do płaszczyzny
- Możemy skorzystać z **iloczynu wektorowego** żeby obliczyć wartość wektora normalnego **n**
- $n = a \times b$
- Konwencją jest podawać wierzchołki w kierunku przeciwnym do zegara
- $n = (V_2 - V_1) \times (V_3 - V_2)$

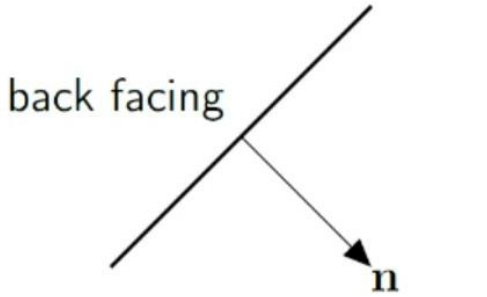


Skierowanie wielokątów

- Wielokąt jest skierowany do przodu (**front facing**) gdy normalna powierzchni jest skierowane do punktu widzenia, w innym przypadku jest skierowany do tylu (**back facing**)

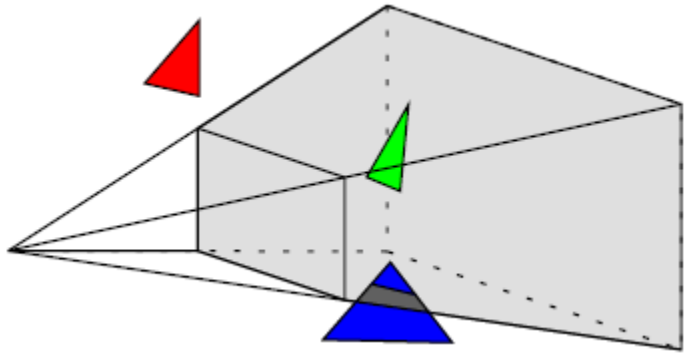






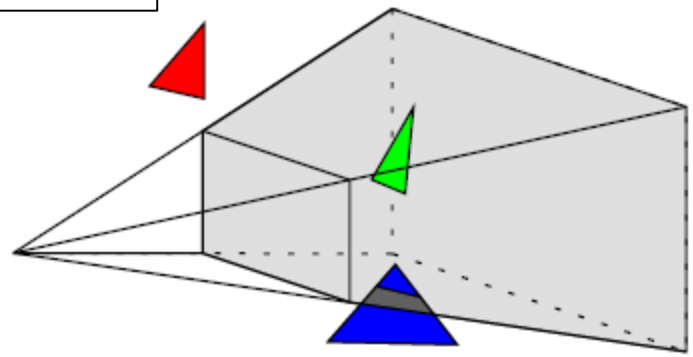
Jakie trójkąty rzutować?

- Trójkąty które lezą (częściowo) poza bryłą widzenia nie musza być rzutowane i są cięte (**clipping/culling**)



Jakie trójkąty rzutować?

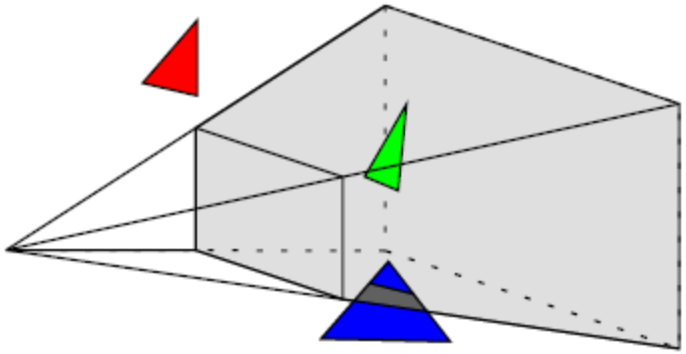
CULLING



- Trójkąty które lezą (częściowo) poza bryłą widzenia nie musza być rzutowane i są cięte (**clipping/culling**)

Jakie trójkąty rzutować?

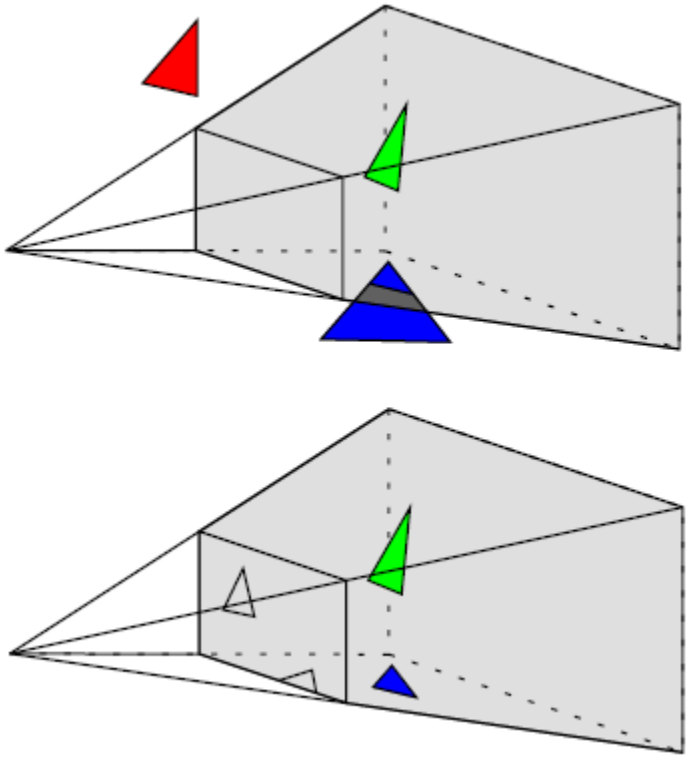
- Trójkąty które lezą (częściowo) poza bryłą widzenia nie musza być rzutowane i są cięte (**clipping/culling**)



CLIPPING

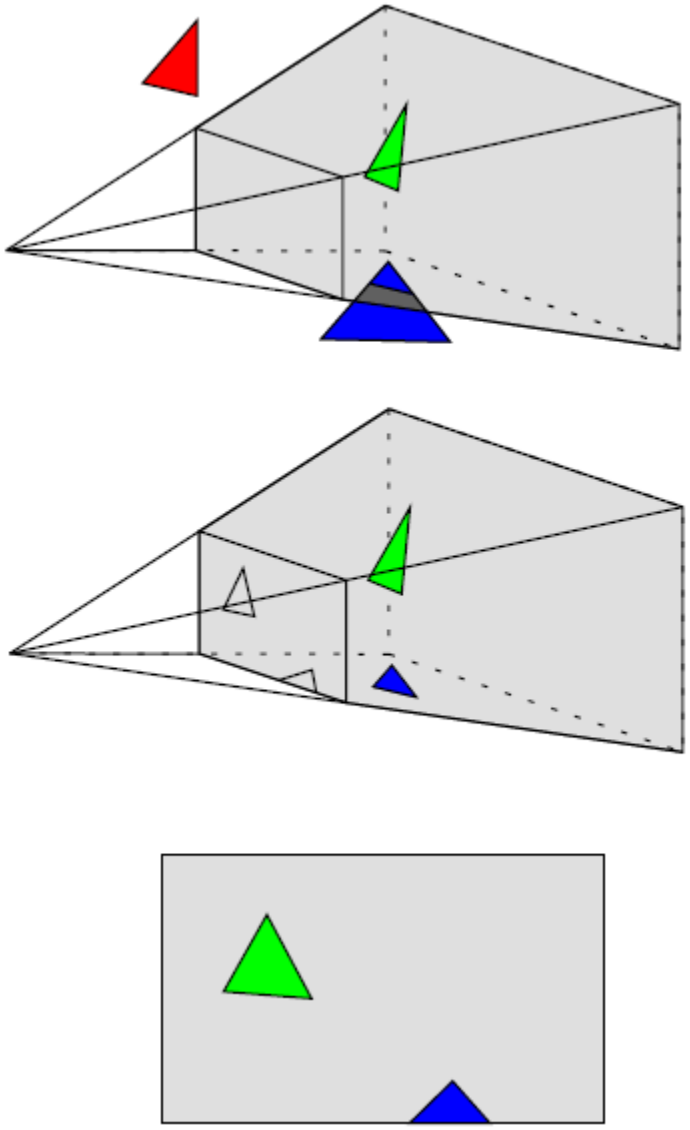
Jakie trójkąty rzutować?

- Trójkąty które lezą (częściowo) poza bryłą widzenia nie musza być rzutowane i są cięte (**clipping/culling**)
- Pozostałe trójkąty są rzutowane na powierzchnie widzenia



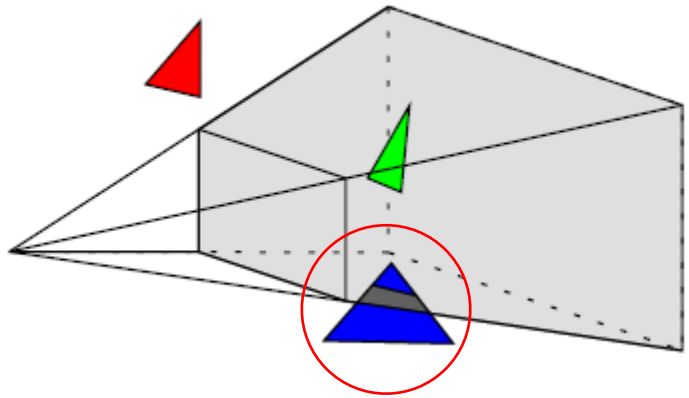
Jakie trójkąty rzutować?

- Trójkąty które lezą (częściowo) poza bryłą widzenia nie musza być rzutowane i są cięte (**clipping/culling**)
- Pozostałe trójkąty są rzutowane na powierzchnie widzenia



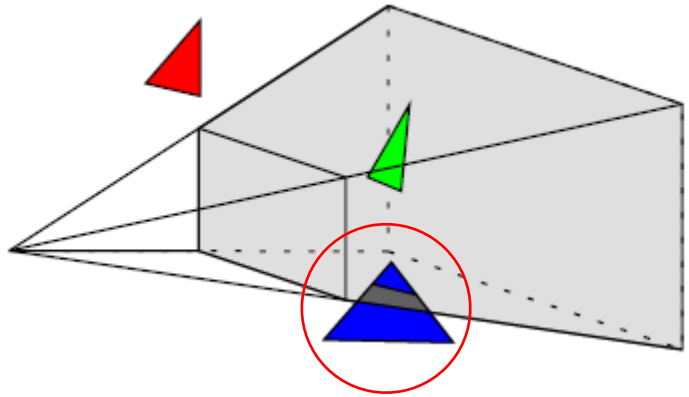
Clipping

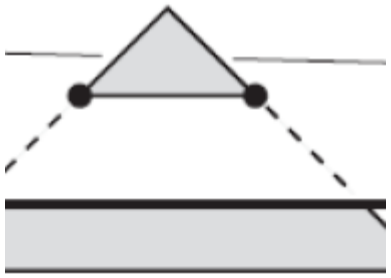
- Żeby zdecydować czy pociąć trójkąt musimy:
  - Sprawdzić czy on przecina **hiperpłaszczyznę**



Clipping

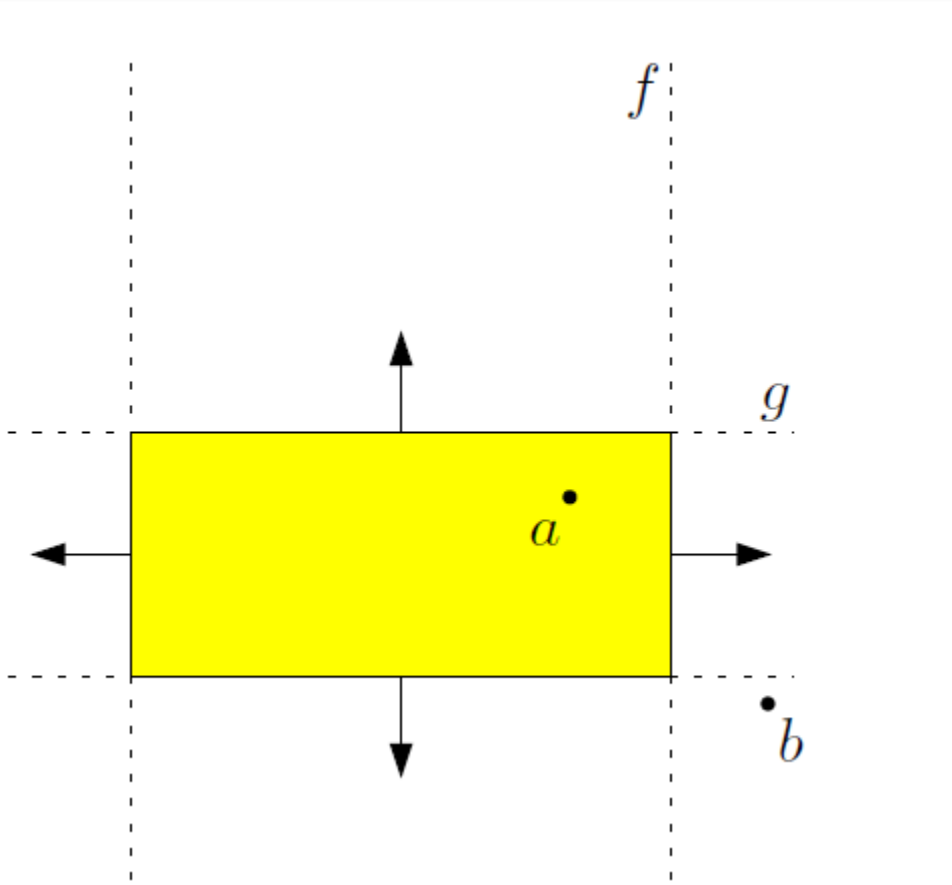
- Żeby zdecydować czy pociąć trójkąt musimy:
  - Sprawdzić czy on przecina **hiperpłaszczyznę**
  - Stworzyć **nowy** trójkąt(y)





Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$



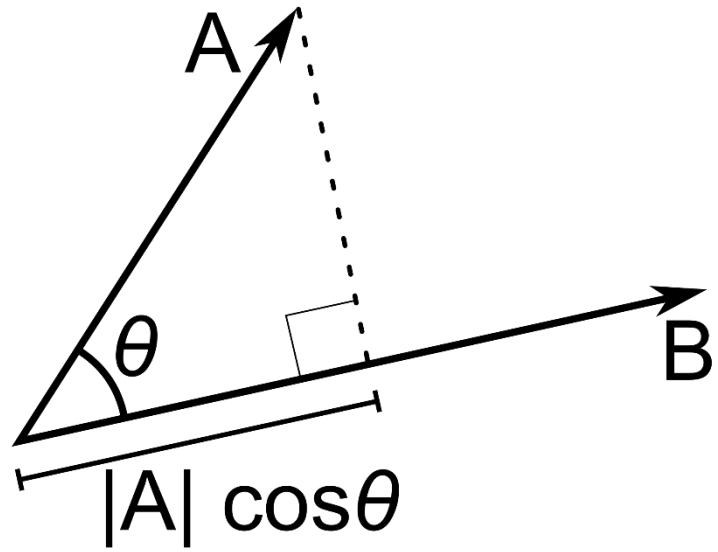
3D - iloczyn skalarny $a \cdot b$

Definicja 1: $a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3$

3D - iloczyn skalarny $a \cdot b$

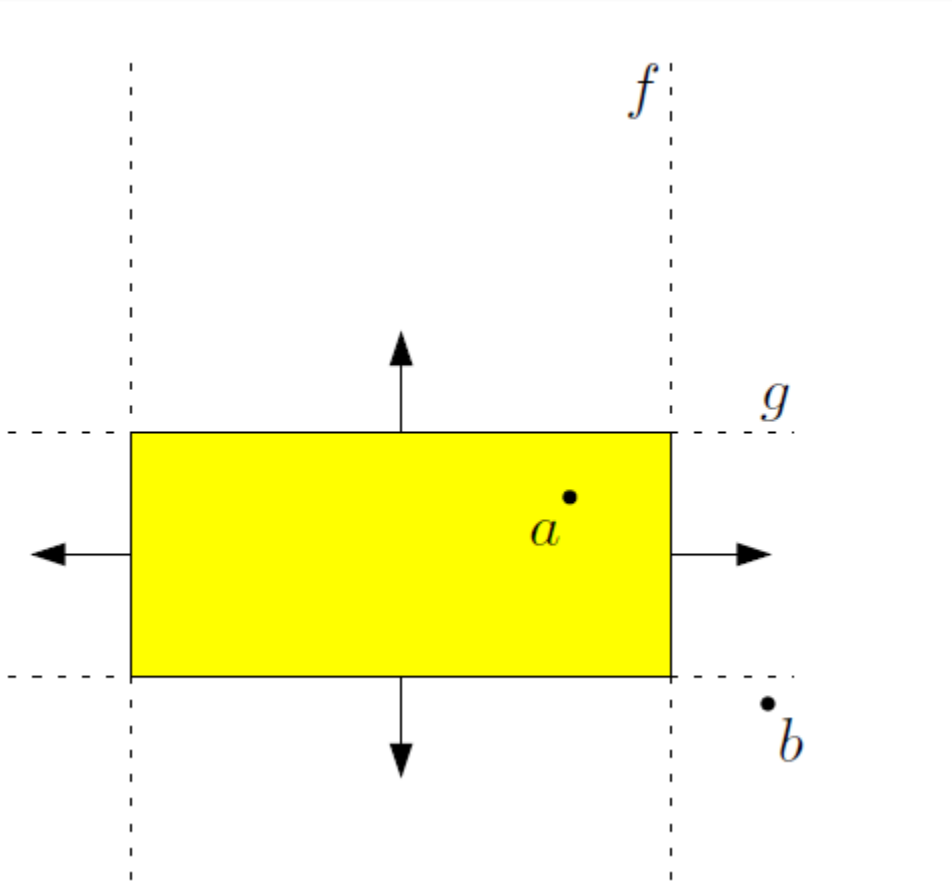
Definicja 1: $a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3$

Definicja 2: $a \cdot b = |a|\,|b| \cos(\theta)$



Sprawdzanie przecinania

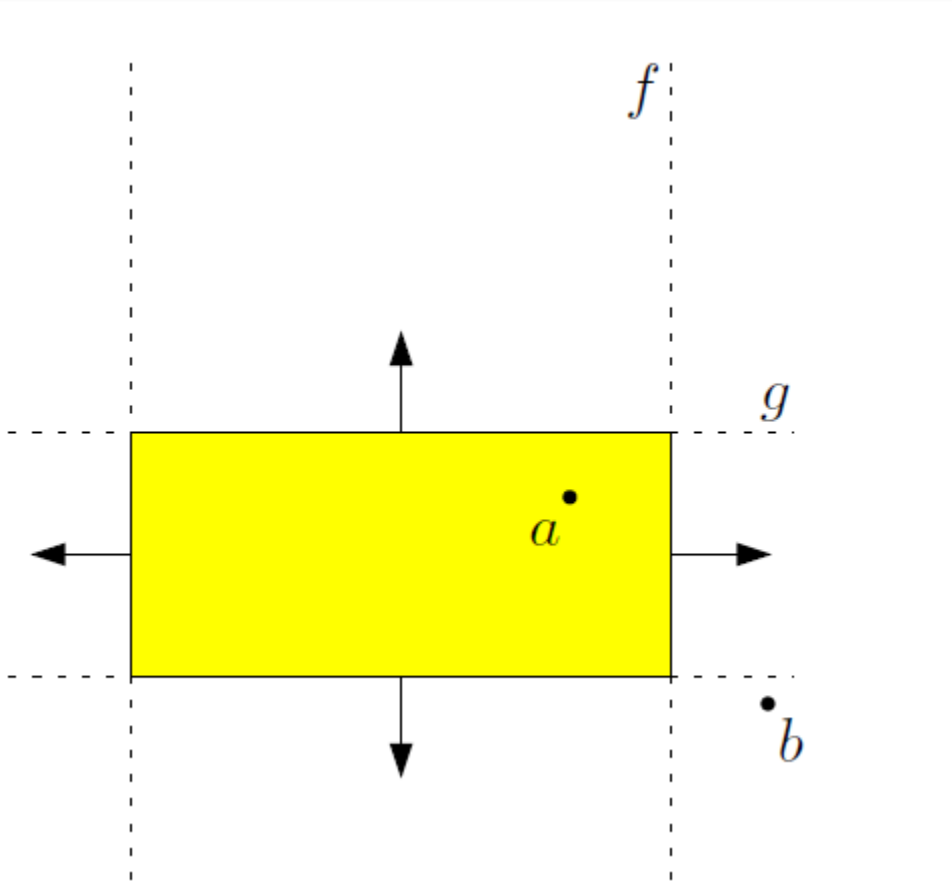
- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$



Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$

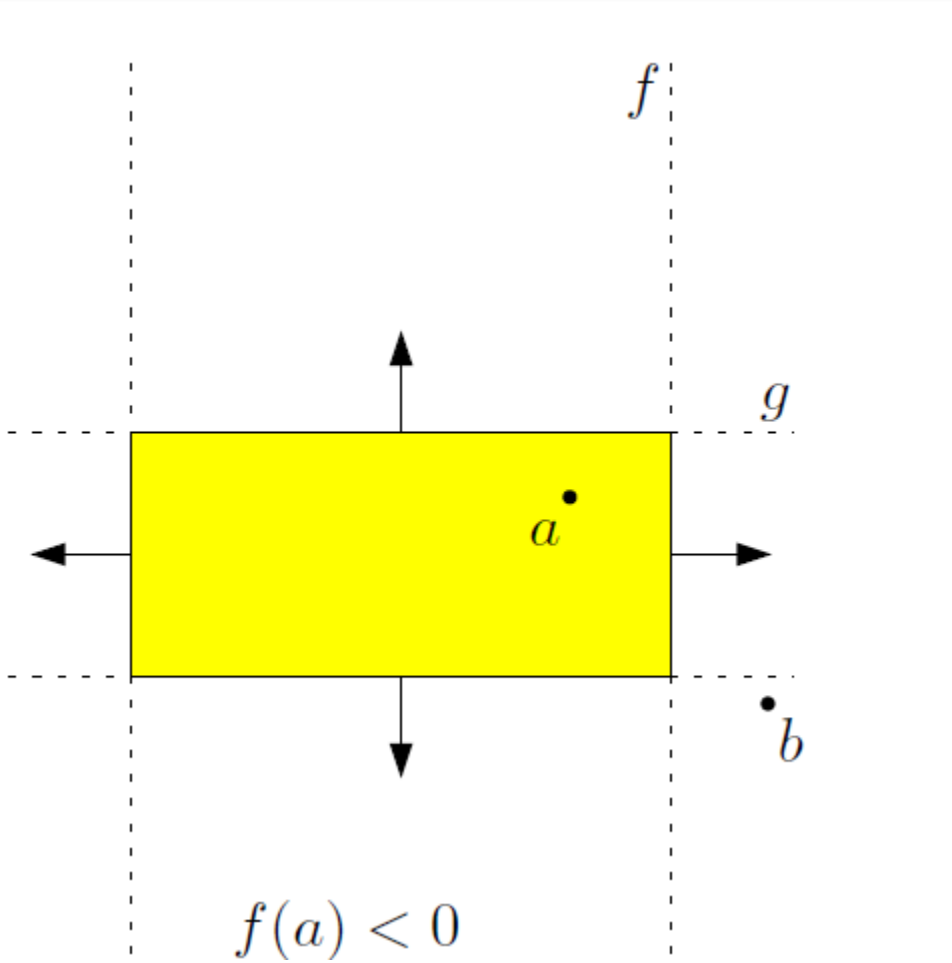
- Konwencja: normalne hiperpłaszczyzn są z kierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$

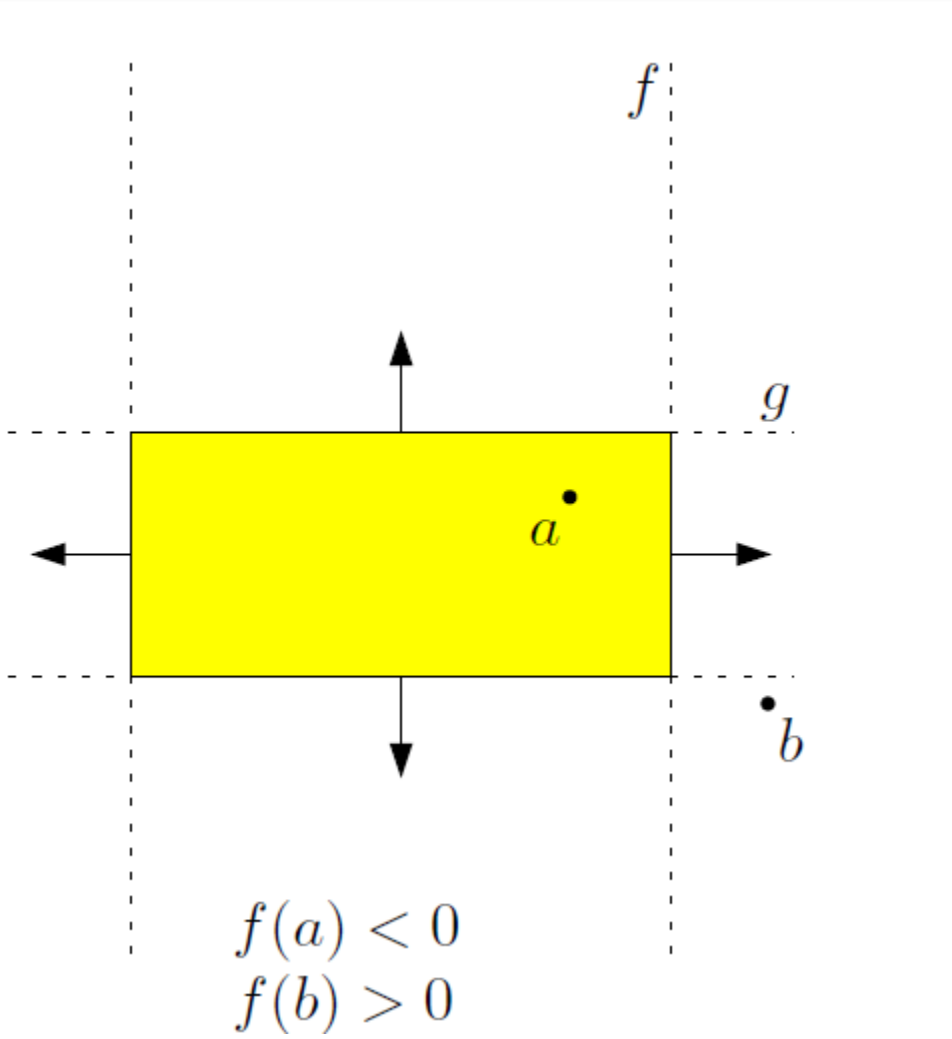
- Konwencja: normalne hiperpłaszczyzn są z kierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$

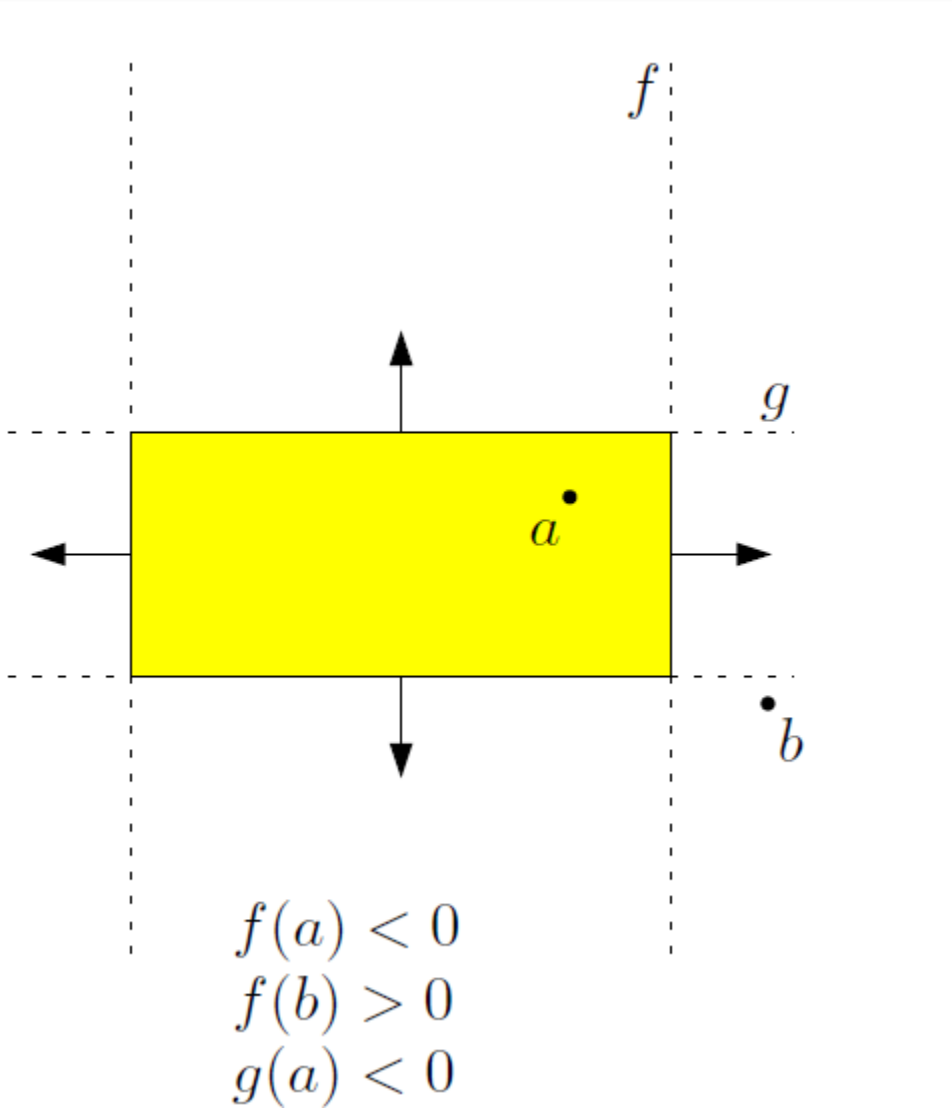
- Konwencja: normalne hiperpłaszczyzn są z kierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$

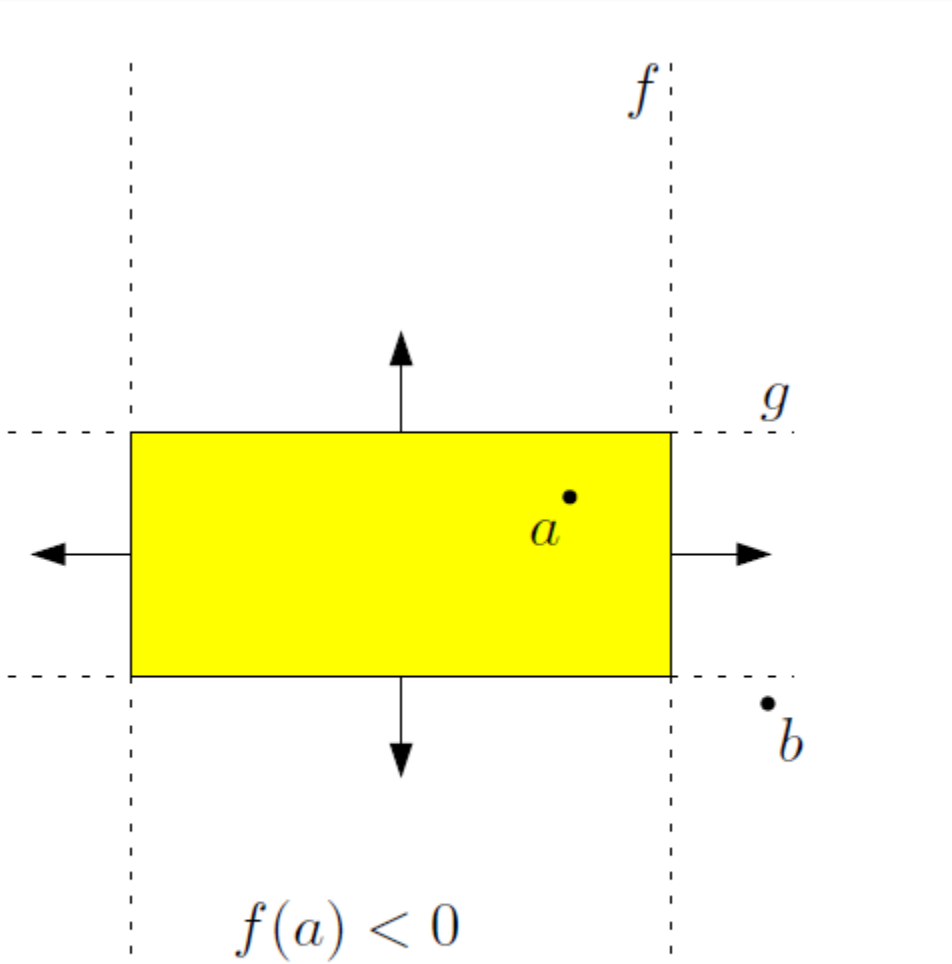
- Konwencja: normalne hiperpłaszczyzn są z kierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$

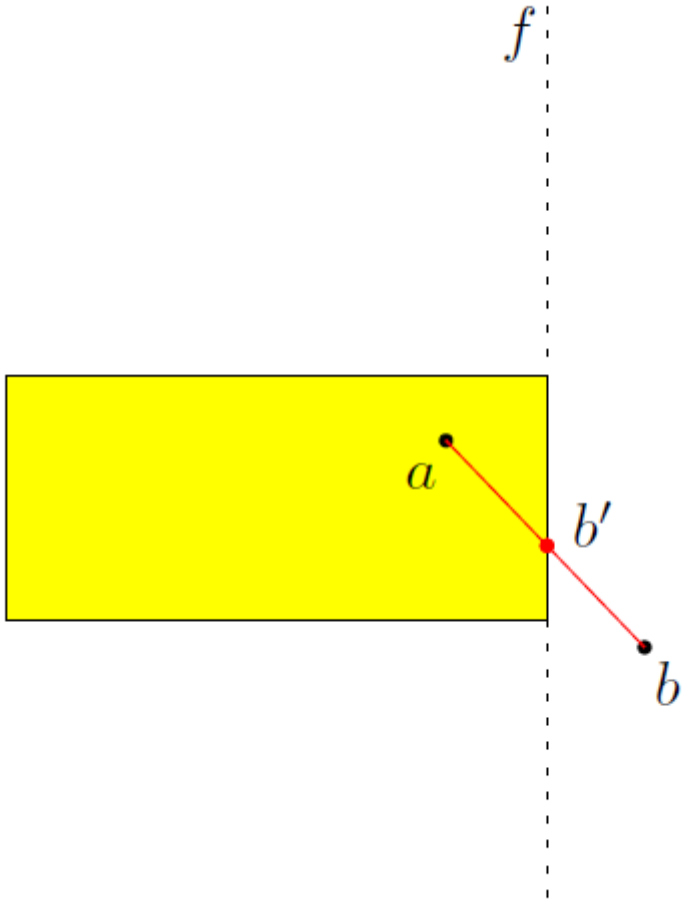
- Konwencja: normalne hiperpłaszczyzn są z kierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



$f(a) < 0$
$f(b) > 0$
$g(a) < 0$
$g(b) < 0$

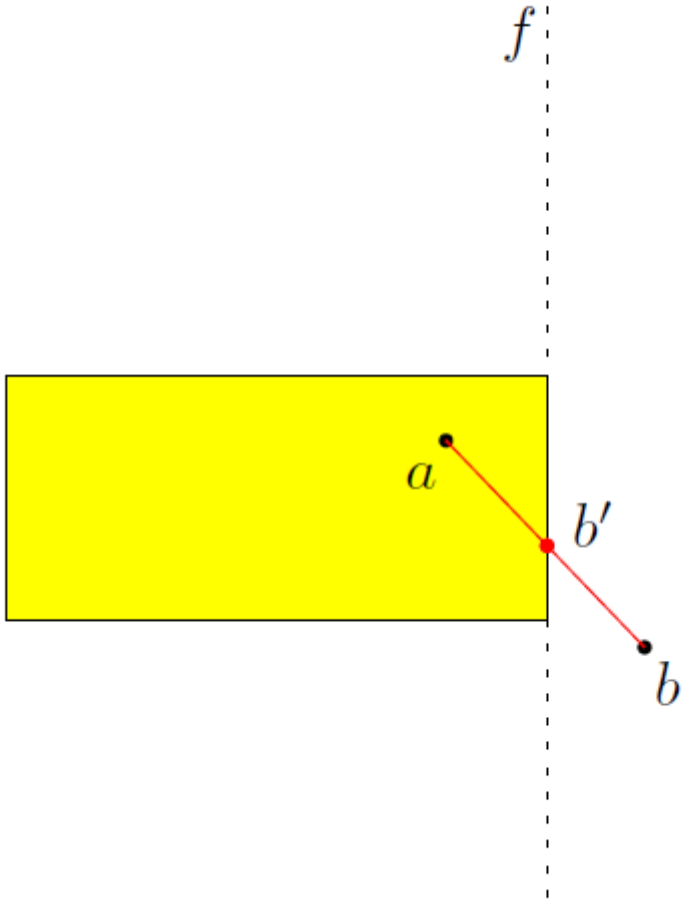
Obliczenie punktów przecięcia

- Jak dwa punkty $\vec{a}$ i $\vec{b}$ są po różnych stronach hiperpłaszczyzny stwarzamy równanie krzywej która przez te punkty przechodzi:
- $\vec{p}(t) = \vec{a} + t\left(\vec{b} - \vec{a}\right)$



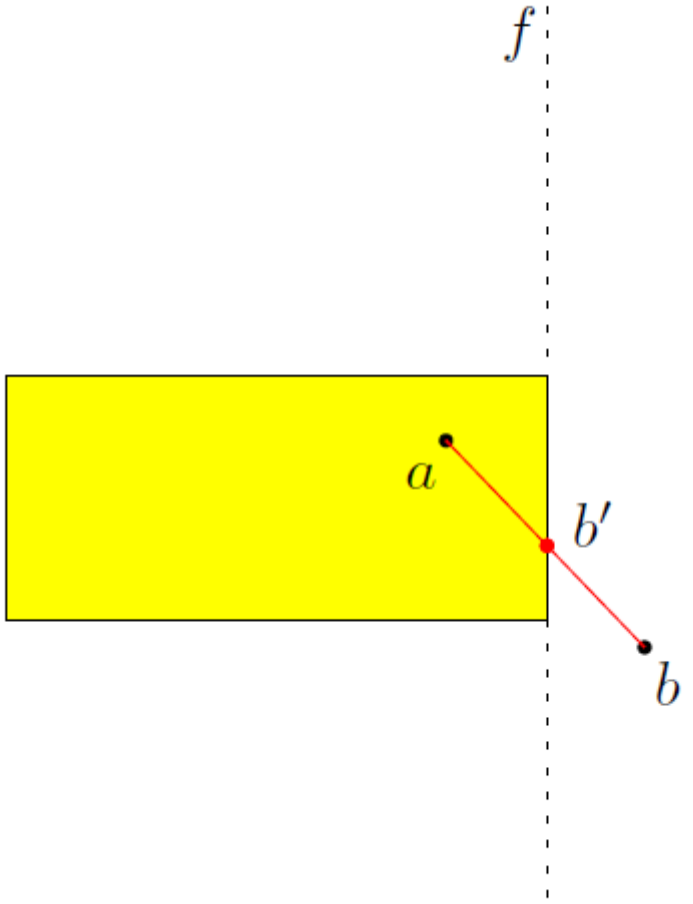
Obliczenie punktów przecięcia

- Jak dwa punkty $\vec{a}$ i $\vec{b}$ są po różnych stronach hiperpłaszczyzny stwarzamy równanie krzywej która przez te punkty przechodzi:
- $\vec{p}(t) = \vec{a} + t(\vec{b} - \vec{a})$
- Podkładając te równanie pod nasze równanie hiperpłaszczyzny daje nam:
- $\vec{n} \cdot (\vec{p} - \vec{q}) = 0$



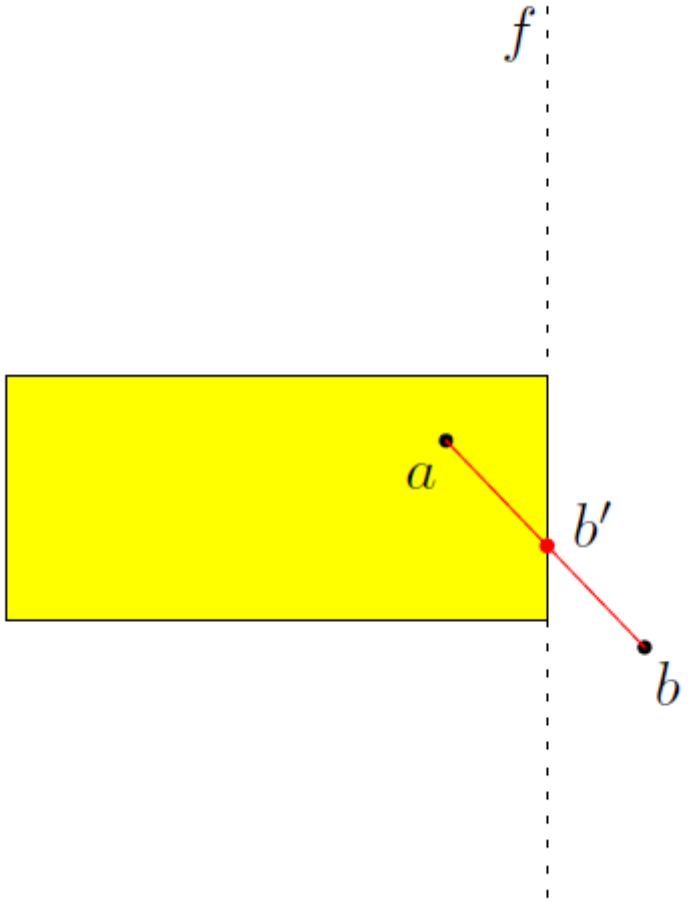
Obliczenie punktów przecięcia

- Jak dwa punkty $\vec{a}$ i $\vec{b}$ są po różnych stronach hiperpłaszczyzny stwarzamy równanie krzywej która przez te punkty przechodzi:
- $\vec{p}(t) = \vec{a} + t(\vec{b} - \vec{a})$
- Podkładając te równanie pod nasze równanie hiperpłaszczyzny daje nam:
- $\vec{n} \cdot (\vec{p} - \vec{q}) = 0$
- $\vec{n} \cdot (\vec{a} + t(\vec{b} - \vec{a}) - \vec{q}) = 0$



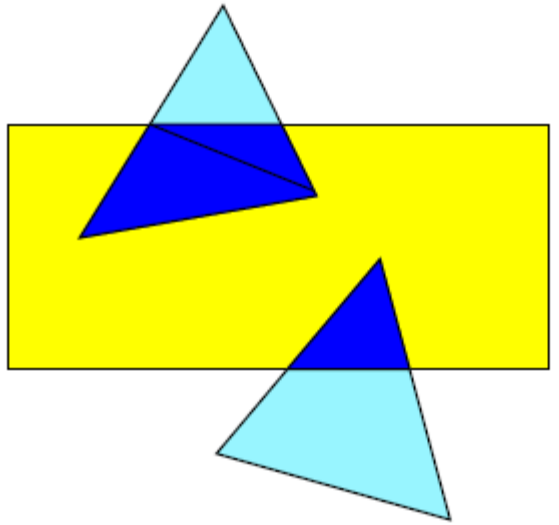
Obliczenie punktów przecięcia

- Jak dwa punkty $\vec{a}$ i $\vec{b}$ są po różnych stronach hiperpłaszczyzny stwarzamy równanie krzywej która przez te punkty przechodzi:
- $\vec{p}(t) = \vec{a} + t(\vec{b} - \vec{a})$
- Podkładając te równanie pod nasze równanie hiperpłaszczyzny daje nam:
- $\vec{n} \cdot (\vec{p} - \vec{q}) = 0$
- $\vec{n} \cdot (\vec{a} + t(\vec{b} - \vec{a}) - \vec{q}) = 0$
- $t = \frac{\vec{n} \cdot \vec{a} + \vec{n} \cdot \vec{q}}{\vec{n} \cdot (\vec{a} - \vec{b})}$



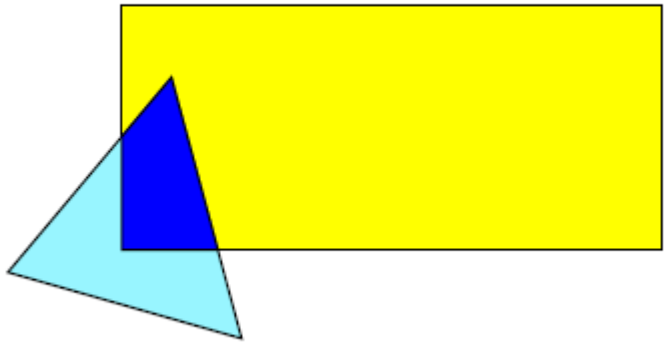
Stwarzanie nowych trójkątów

- Na podstawie **dwóch punktów intersekcji** możemy pociąć nasz trójkąt według naszej hiperpłaszczyzny w następujący sposób:
  - Jak **dwa wierzchołki** są zewnątrz hiperpłaszczyzny dostajemy **nowy trójkąt**
  - Jak **jeden wierzchołek** jest zewnątrz hiperpłaszczyzny dostajemy **dwa nowe trójkąty**



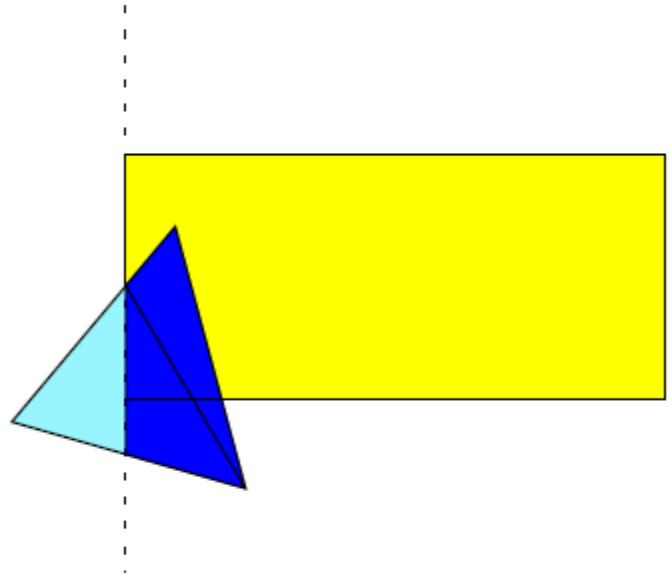
Stwarzanie nowych trójkątów

- Ale co gdy trójkąt jest przecinany przez dwie płaszczyzny?



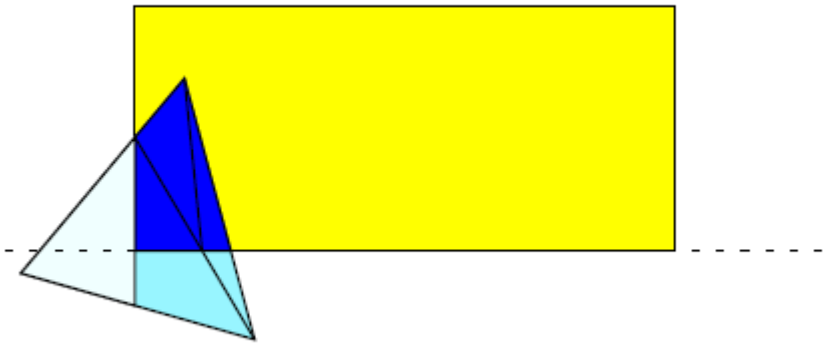
Stwarzanie nowych trójkątów

- Przecinamy względem jednej



Stwarzanie nowych trójkątów

- Przecinamy względem drugiej



$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

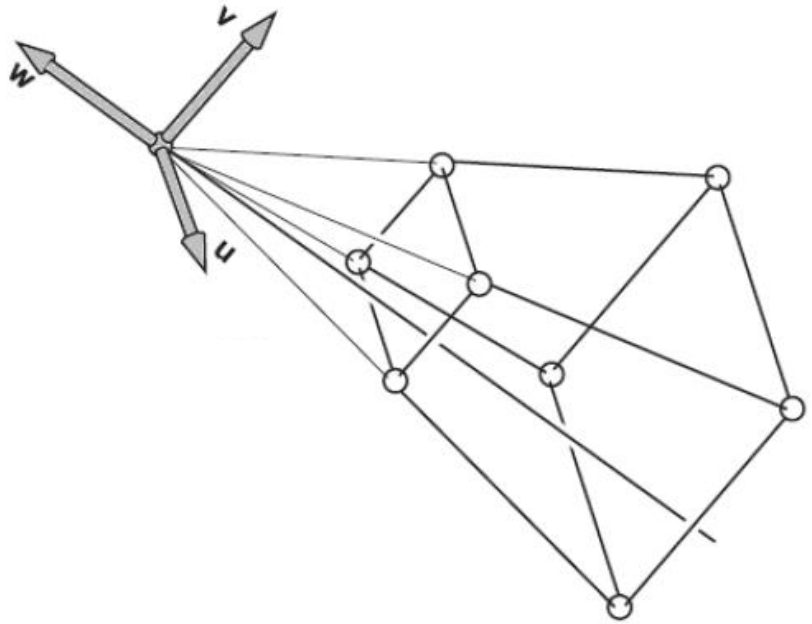
Macierz transformacji

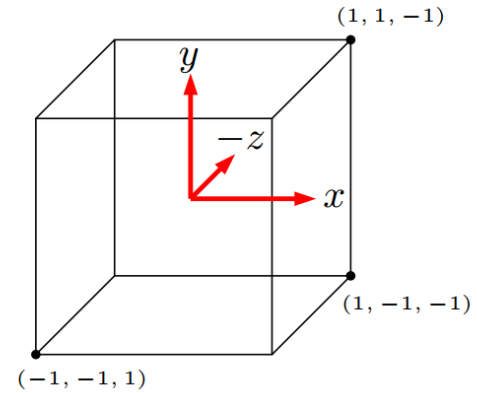
$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix}$$

Homogenizacja

$$\begin{bmatrix} x'/w \\ y'/w \\ z'/w \\ 1 \end{bmatrix}$$

Rasteryzacja

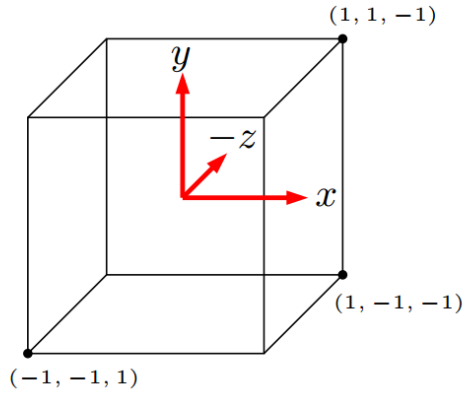




Clipping po homogenizacji

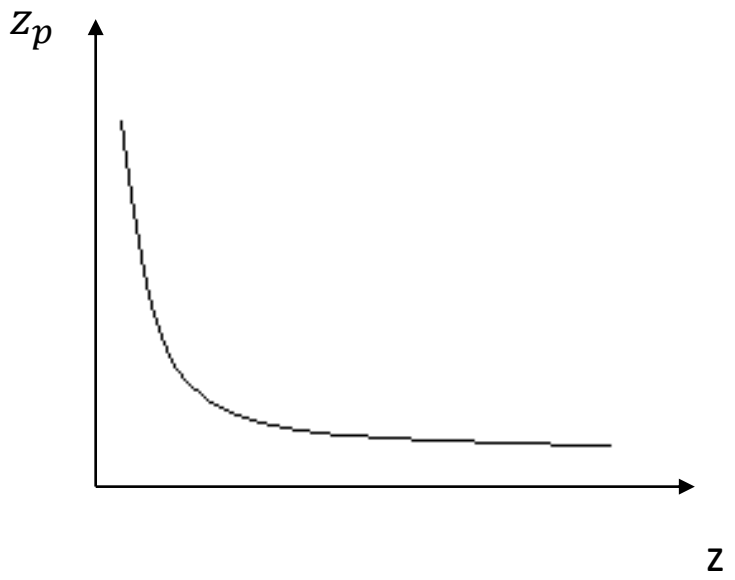
- Proste równania dla hiperpłaszczyzn:

$$
\begin{aligned}
-x + l &= 0 \\
x - r &= 0 \\
-y + b &= 0 \\
y - t &= 0 \\
-z + n &= 0 \\
z - f &= 0
\end{aligned}
$$



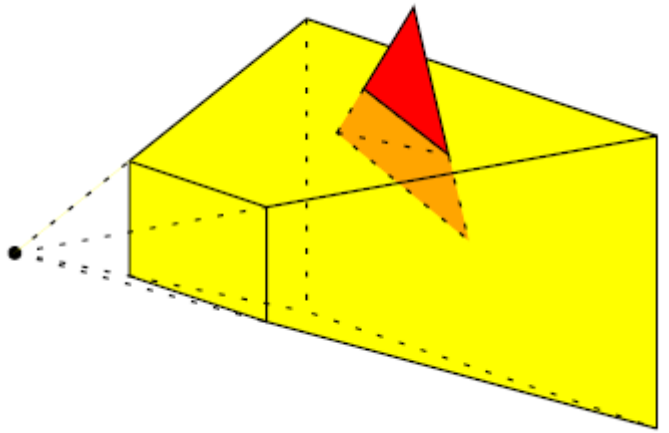
Problem z płaszczyzną XY

$$z_p = n + f - \frac{fn}{z} \quad \rightarrow \quad z_p \sim \frac{1}{z}$$



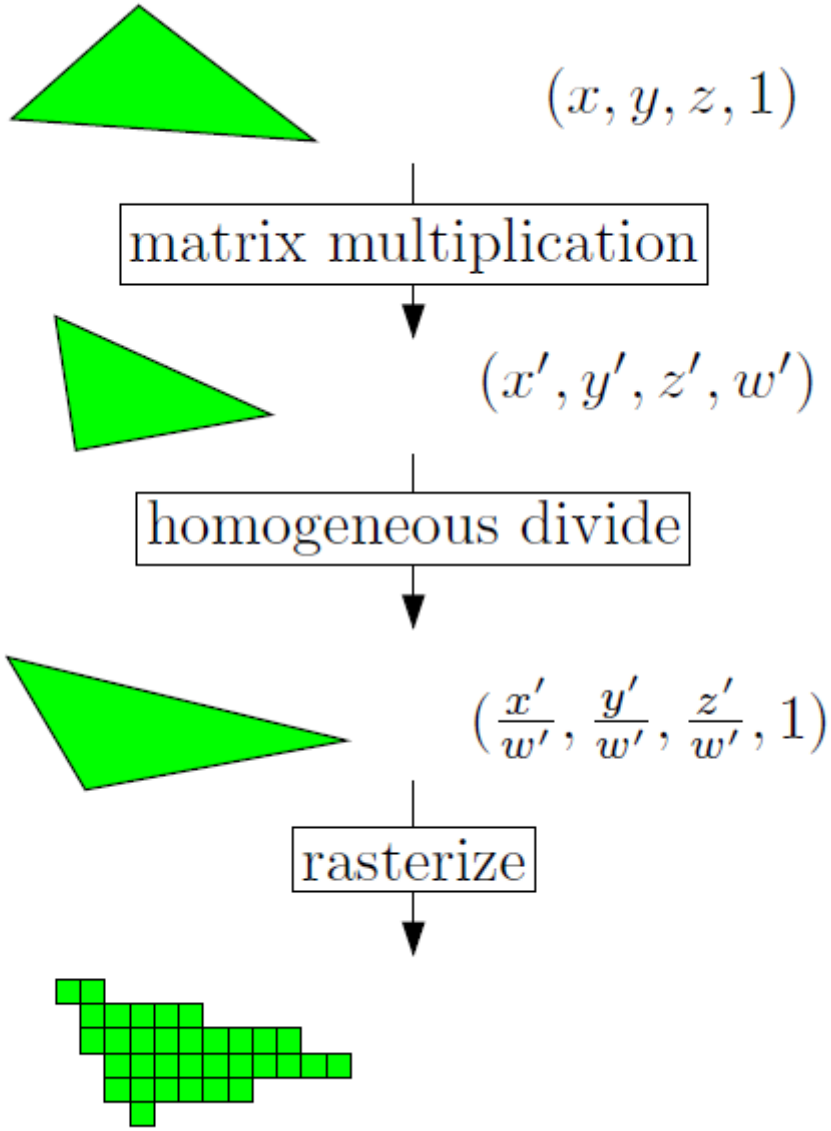
Clipping przed homogenizacją

- Wierzchołki bryły widzenia da się obliczyć przez $M_{per}{}^{-1}$
- Wtedy równania płaszczyzn bryły widzenia można wyprowadzić



Clipping w systemie współrzędnych jednorodnych

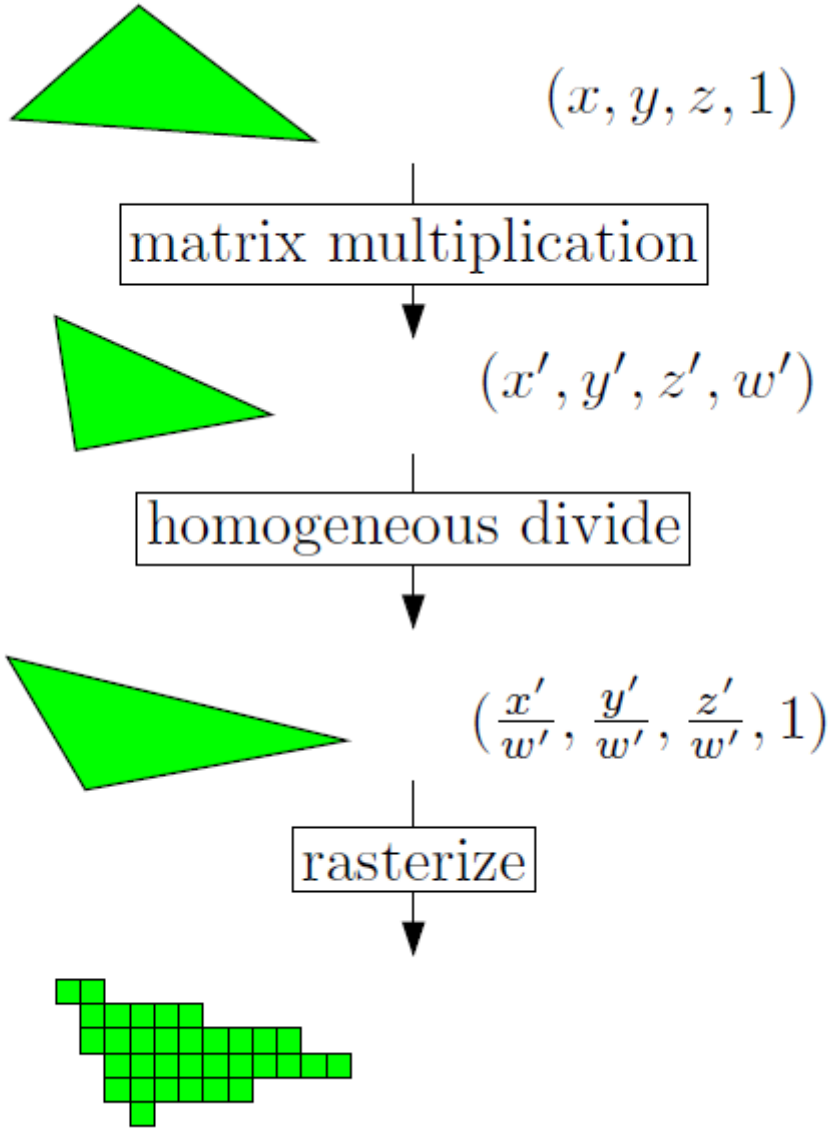
- Jednak najłatwiej da się wykonać **clipping** w systemie **współrzędnych jednorodnych**. To znaczy: ucinamy trójkąty w **czterech wymiarach** według **trójwymiarowych hiperpłaszczyzn**.



Clipping w systemie współrzędnych jednorodnych

- Jednak najłatwiej da się wykonać **clipping** w systemie **współrzędnych jednorodnych** To znaczy: ucinamy trójkąty w **czterech wymiarach** według **trójwymiarowych hiperpłaszczyzn**.

$$
\begin{aligned}
-x' + lw' &= 0 \\
x' - rw' &= 0 \\
-y' + bw' &= 0 \\
y' - tw' &= 0 \\
-z' + nw' &= 0 \\
z' - fw' &= 0
\end{aligned}
$$

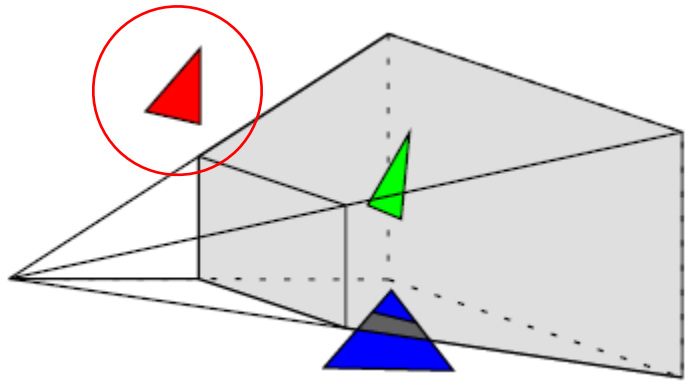


Algorytmy clippingu dla dowolnych wielokątów

- Greiner-Hormann
- Sutherland–Hodgman
- Weiler–Atherton
- Vatti

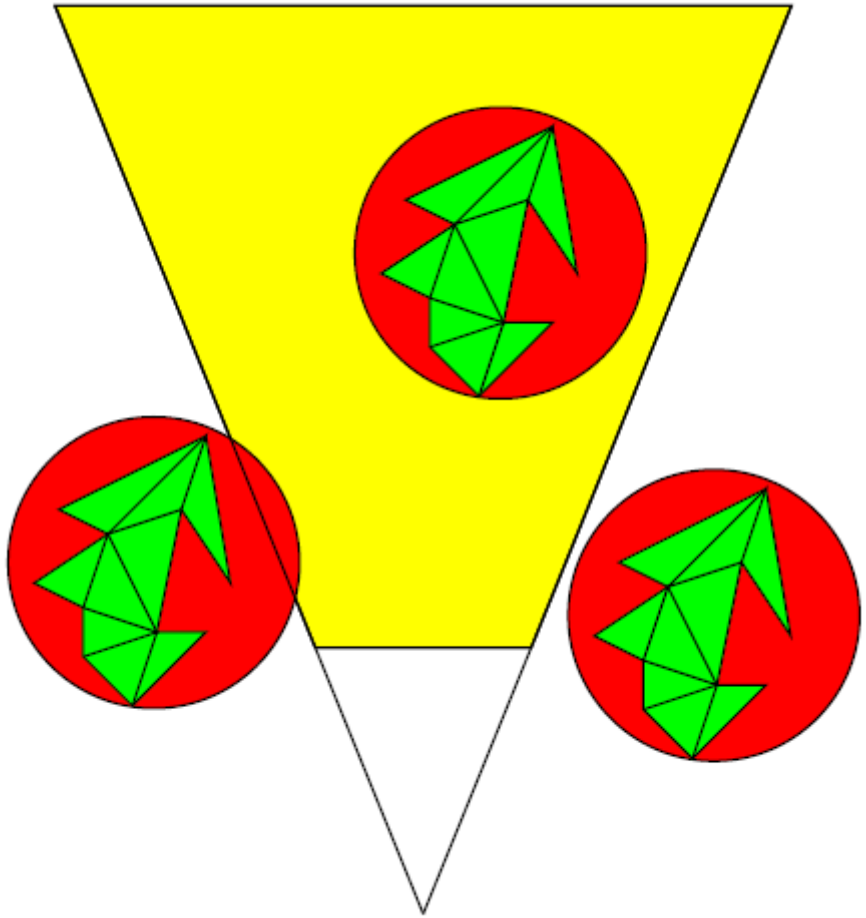
Culling

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



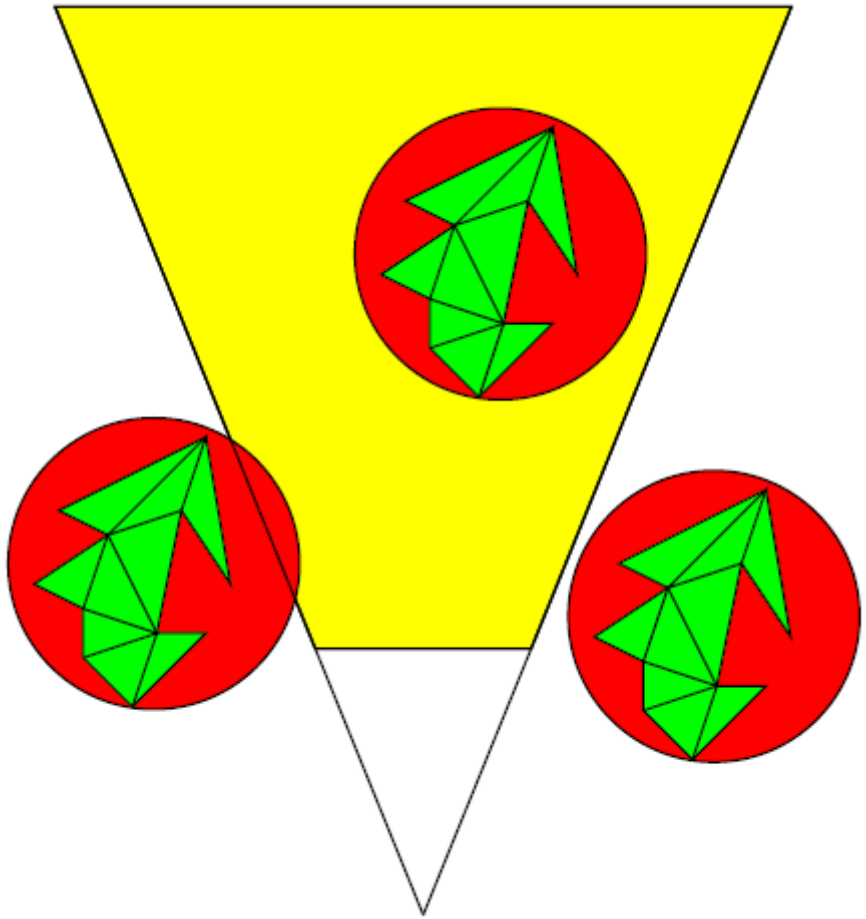
Bryły brzegowe (bounding volumes)

- Używanie brył brzegowych dla skomplikowanych obiektów geometrycznych przyspiesza potok graficzny



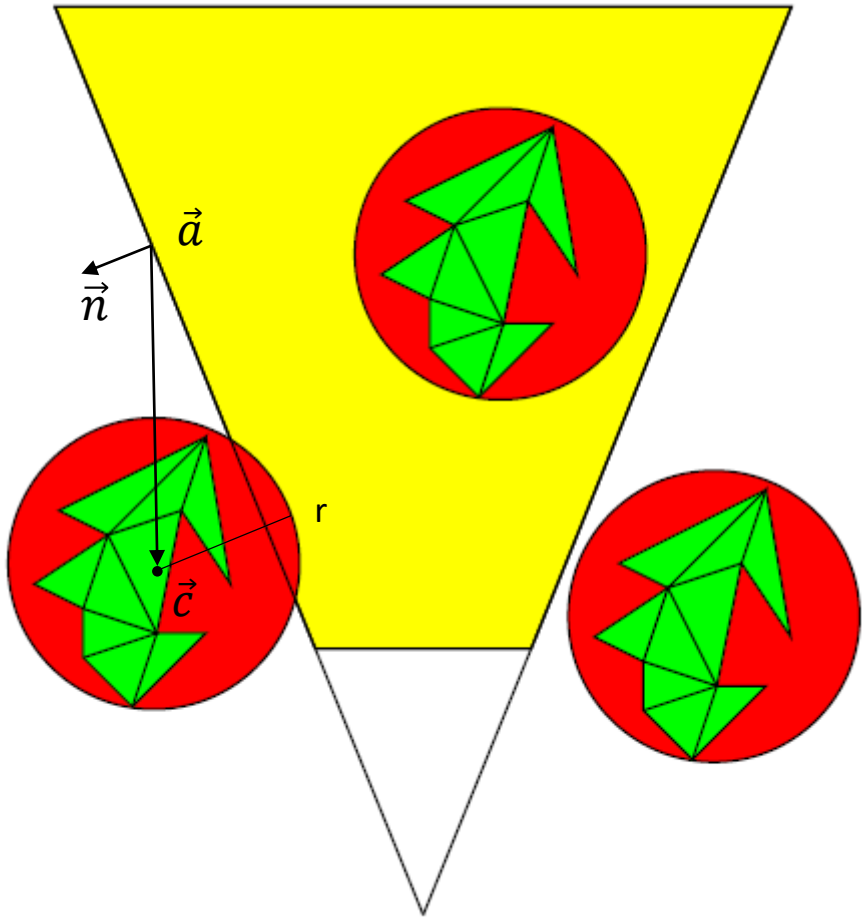
Bryły brzegowe (bounding volumes)

- Często kule są używane jako bryły brzegowe
- Jak płaszczyzna jest dana przez
- $(\vec{p} - \vec{a}) \cdot \vec{n} = 0$
- i kula ma środek $\vec{c}$ i promień r, wtedy sprawdzamy nierówność
- $\frac{(\vec{c}-\vec{a})\cdot\vec{n}}{\|n\|} > r$



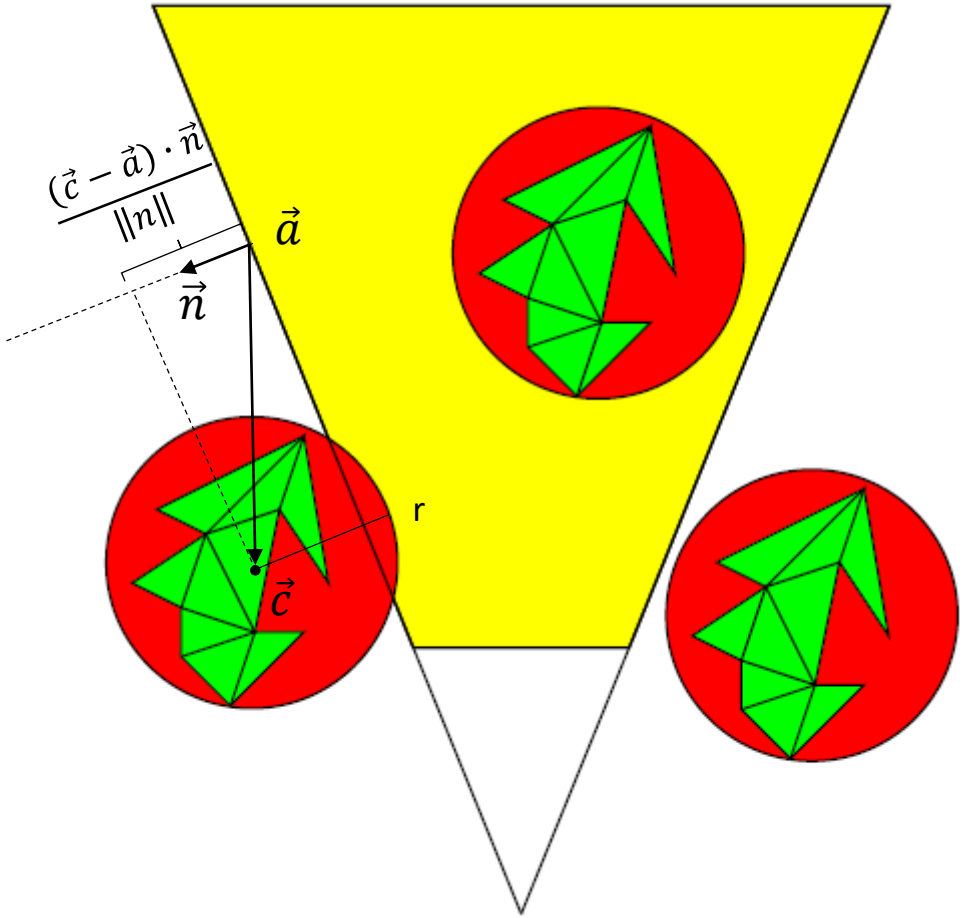
Bryły brzegowe (bounding volumes)

- Często kule są używane jako bryły brzegowe
- Jak płaszczyzna jest dana przez
- $(\vec{p} - \vec{a}) \cdot \vec{n} = 0$
- i kula ma środek $\vec{c}$ i promień r, wtedy sprawdzamy nierówność
- $\frac{(\vec{c}-\vec{a})\cdot\vec{n}}{\|n\|} > r$



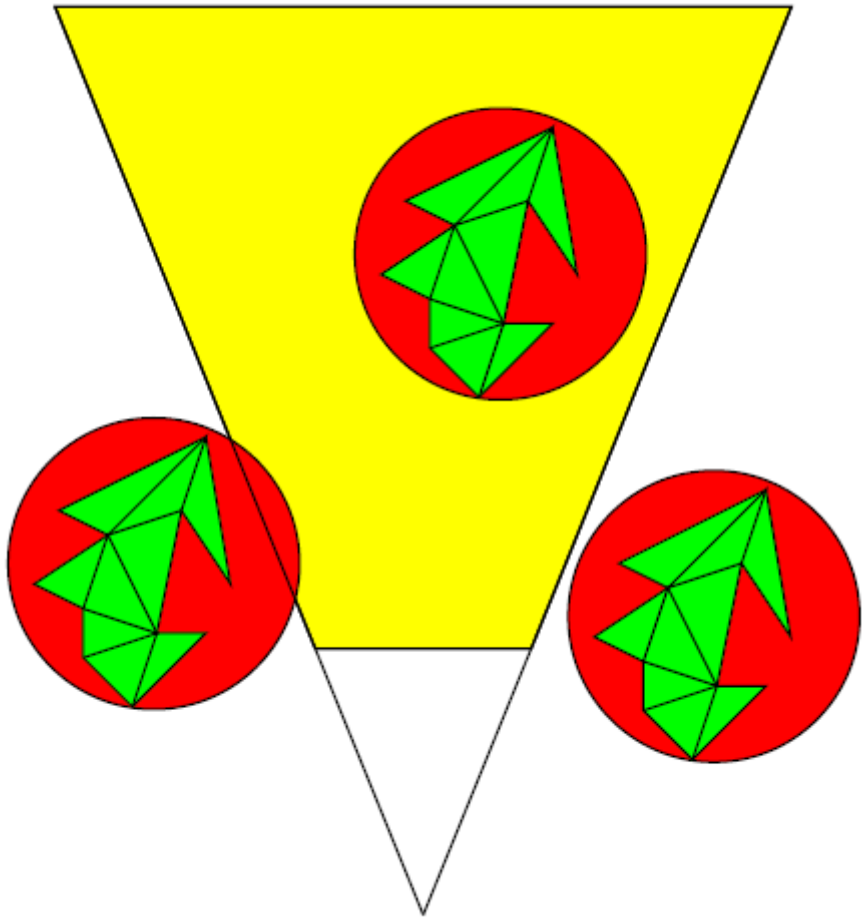
Bryły brzegowe (bounding volumes)

- Często kule są używane jako bryły brzegowe
- Jak płaszczyzna jest dana przez
- $(\vec{p} - \vec{a}) \cdot \vec{n} = 0$
- i kula ma środek $\vec{c}$ i promień r, wtedy sprawdzamy nierówność
- $\frac{(\vec{c}-\vec{a})\cdot\vec{n}}{\|n\|} > r$



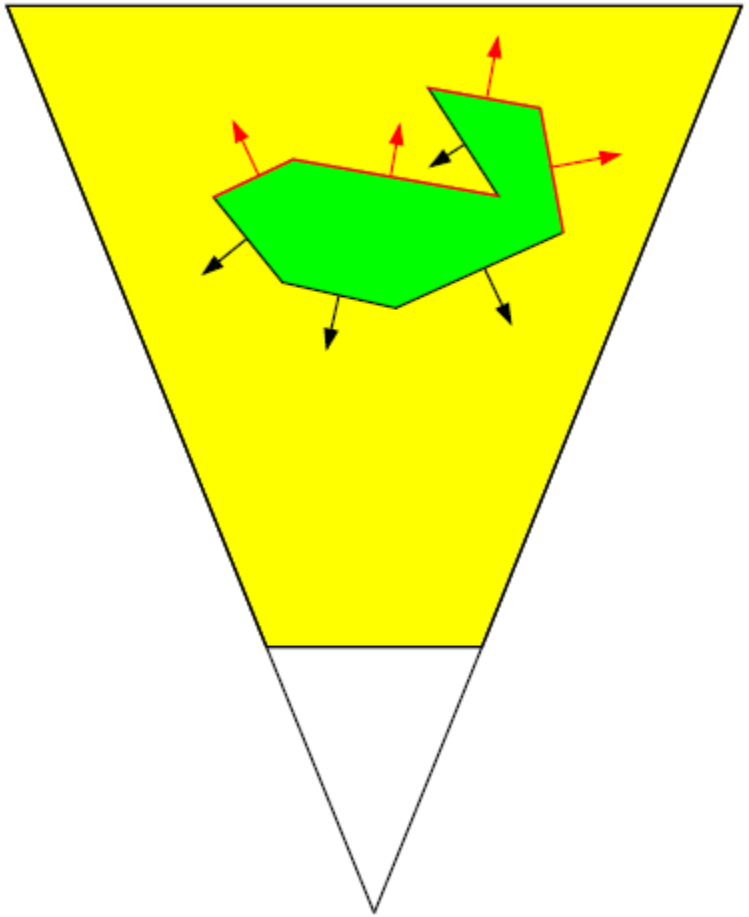
Culling

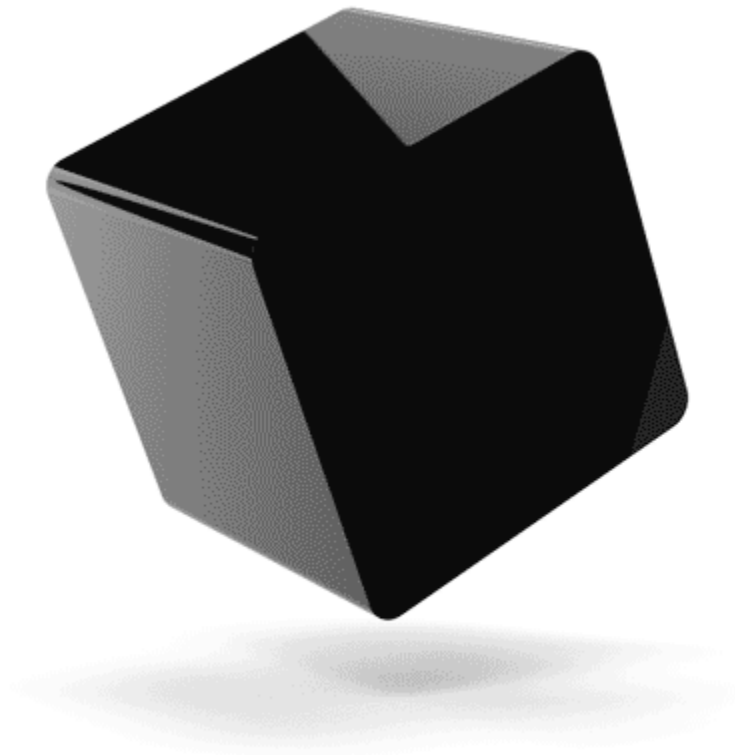
- **Frustum culling**
  usuwanie trójkątów poza bryły widzenia
- **Backface culling**
  usuwanie trójkątów orientowane od pozycji kamery

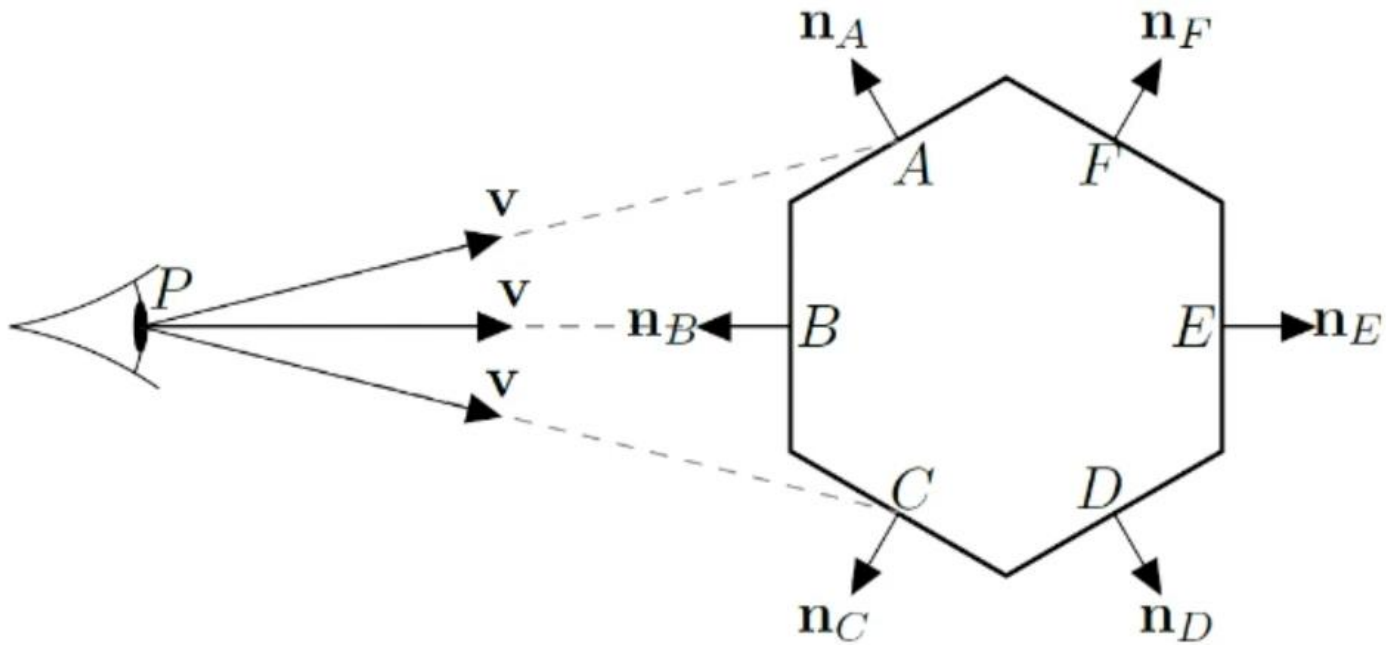


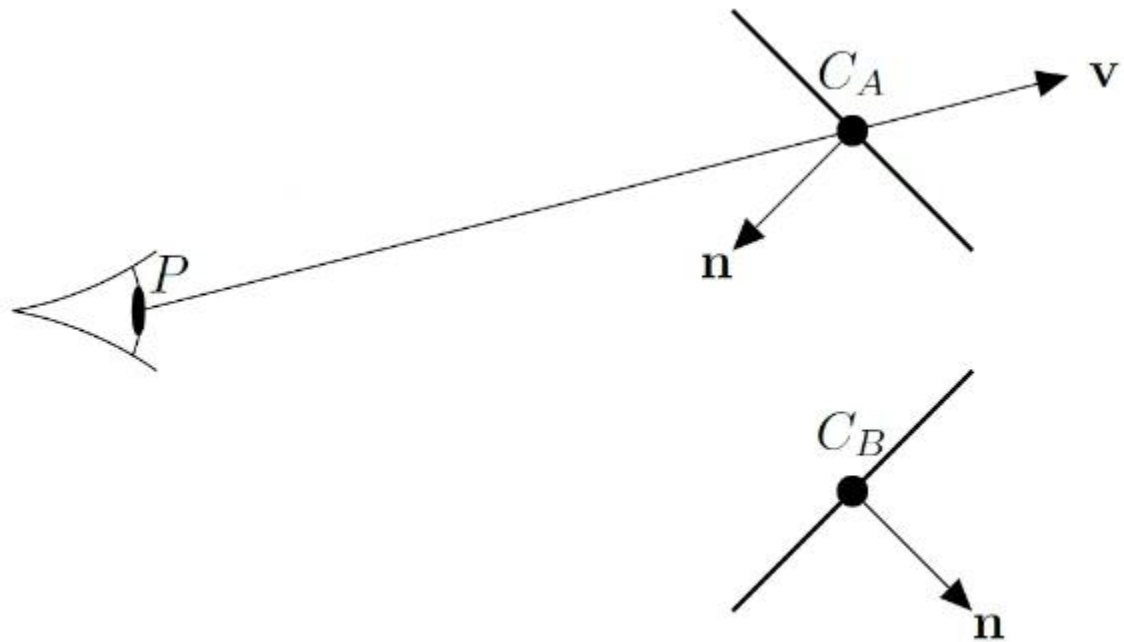
Backface culling

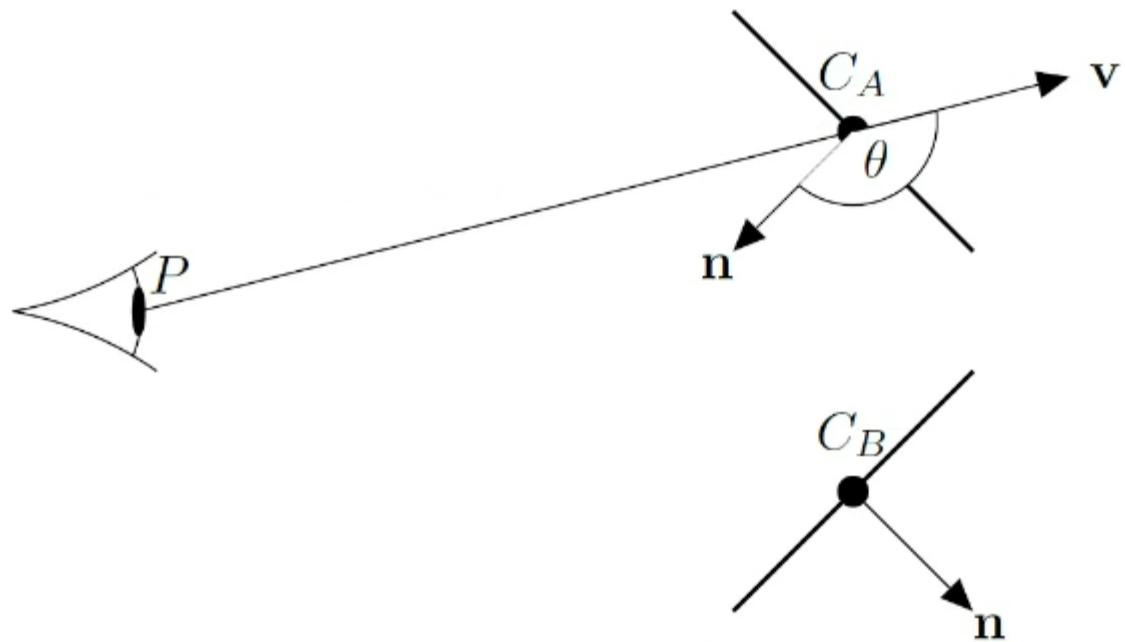
- Gdy modelujemy obiekty geometryczne za pomocy trójkątów to **normalna trójkątów** jest ustawiona na **zewnątrz** obiektu
- Odrzucanie trójkątów których normalna jest skierowana **od punktu widzenia** nazywamy to **backface culling** (usuwanie powierzchni tylnych)

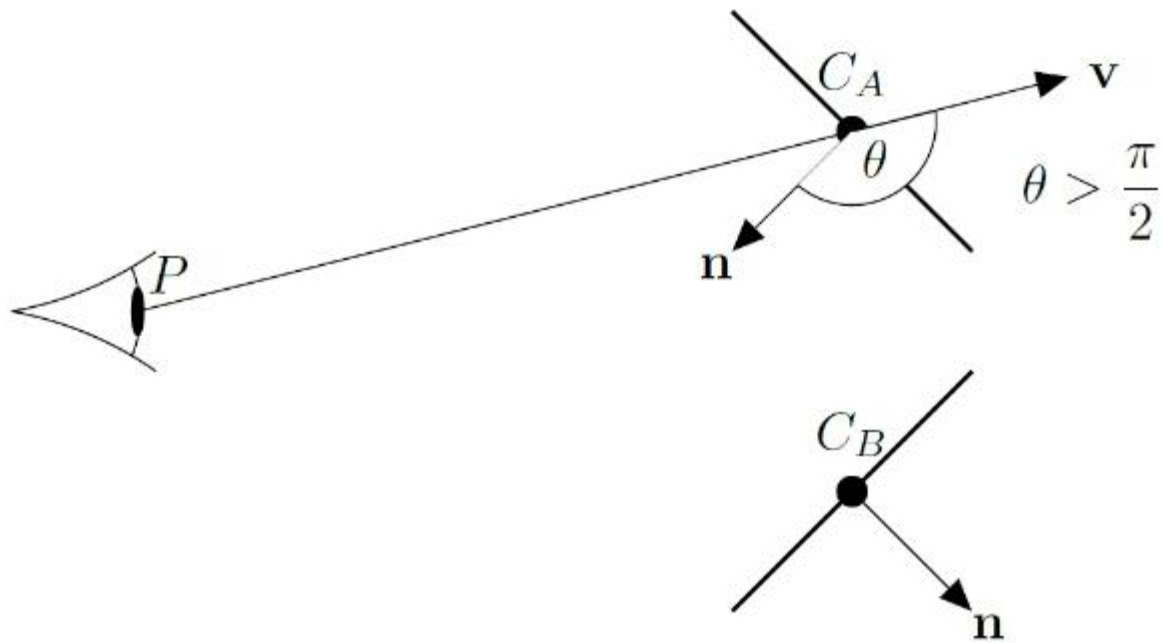


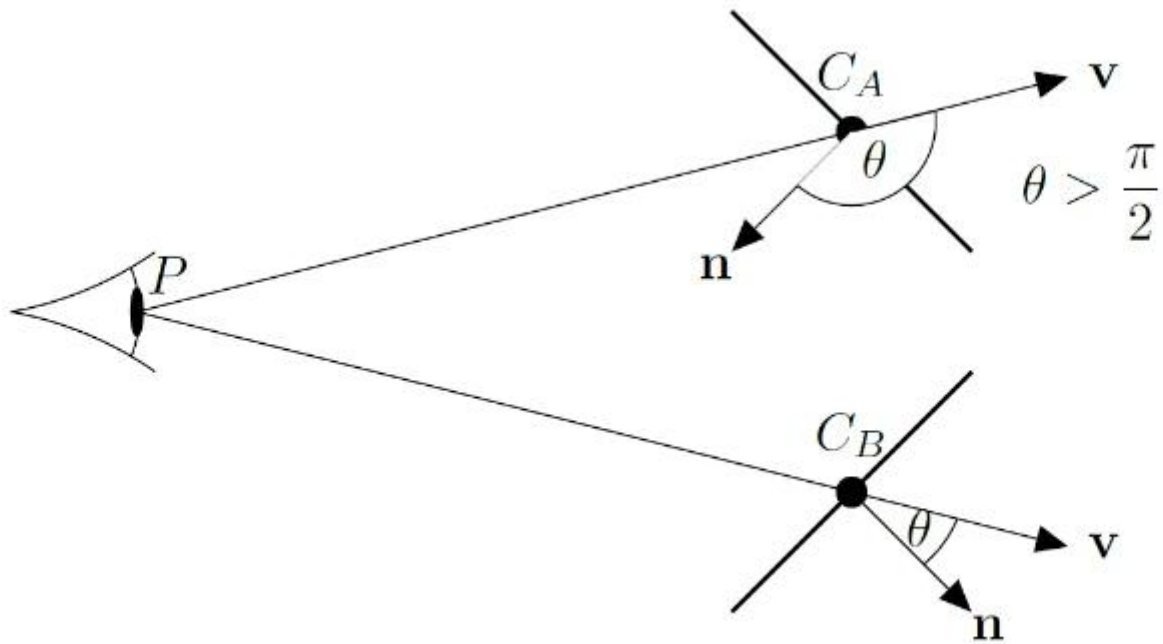


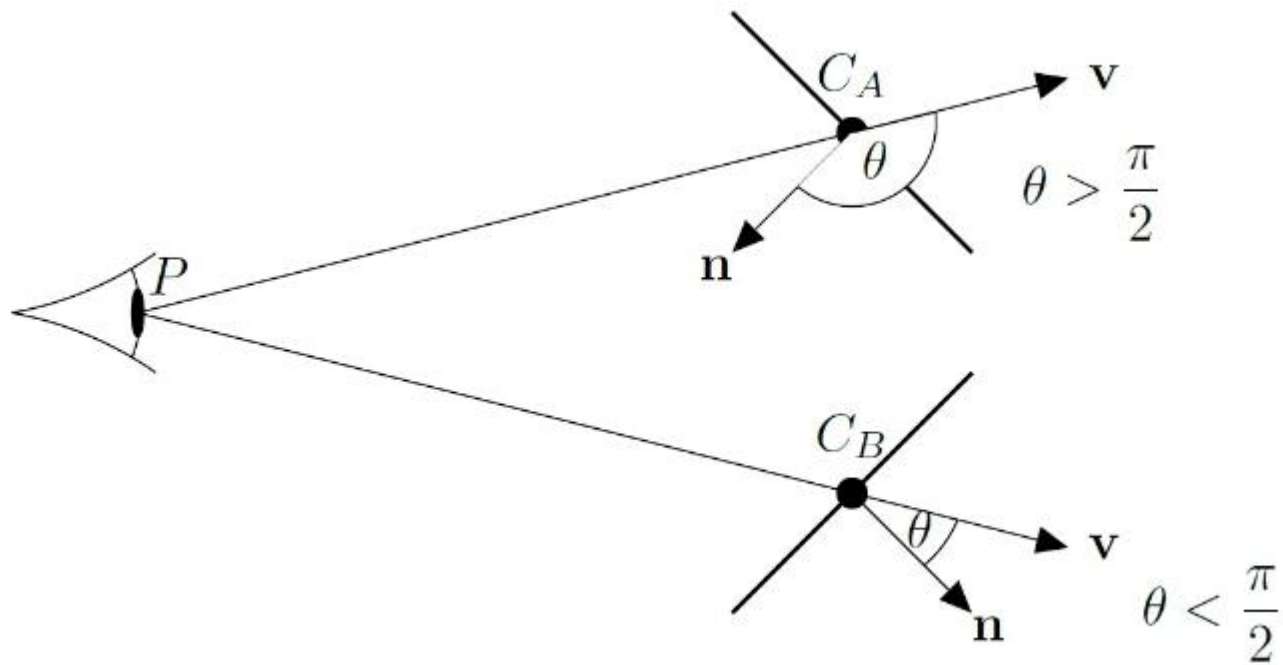

$\mathbf{n}_A$
$\mathbf{n}_F$
A
F
$\mathbf{v}$
$\mathbf{v}$
$\mathbf{v}$
P
$\mathbf{n}_B$
B
E
$\mathbf{n}_E$
C
D
$\mathbf{n}_C$
$\mathbf{n}_D$

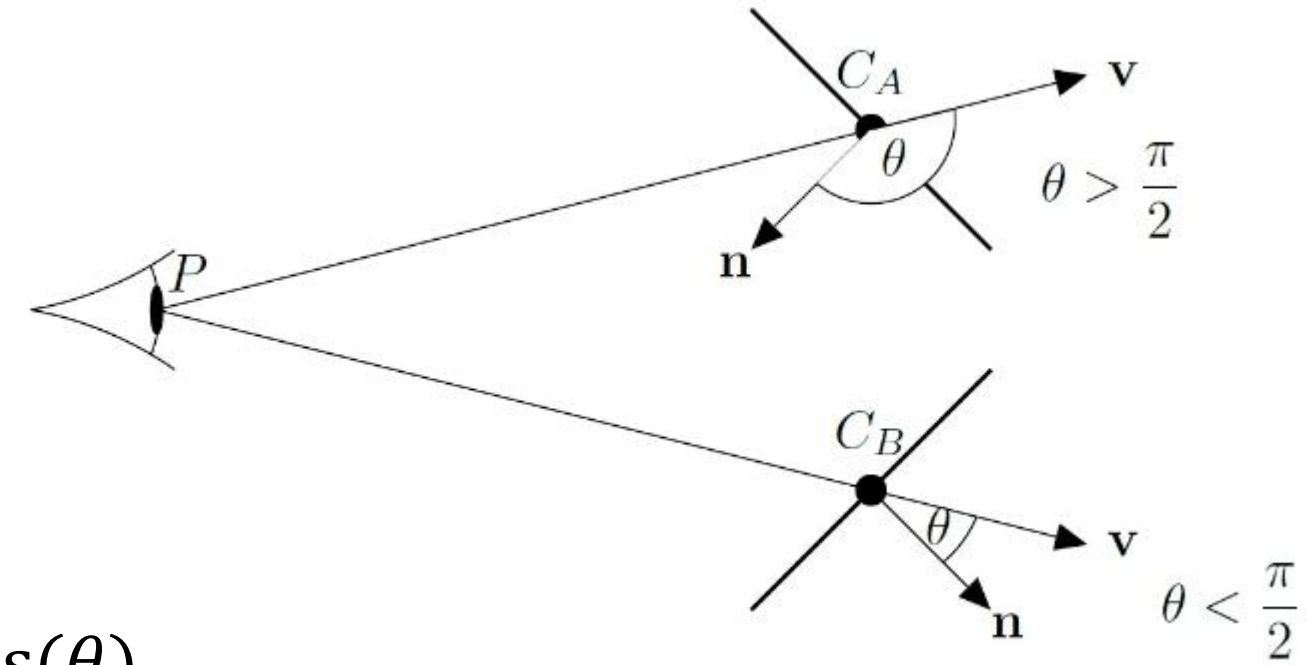

C_A
$\mathbf{v}$
$\mathbf{n}$
P
C_B
$\mathbf{n}$


C_A
$\mathbf{v}$
θ
$\mathbf{n}$
P
C_B
$\mathbf{n}$

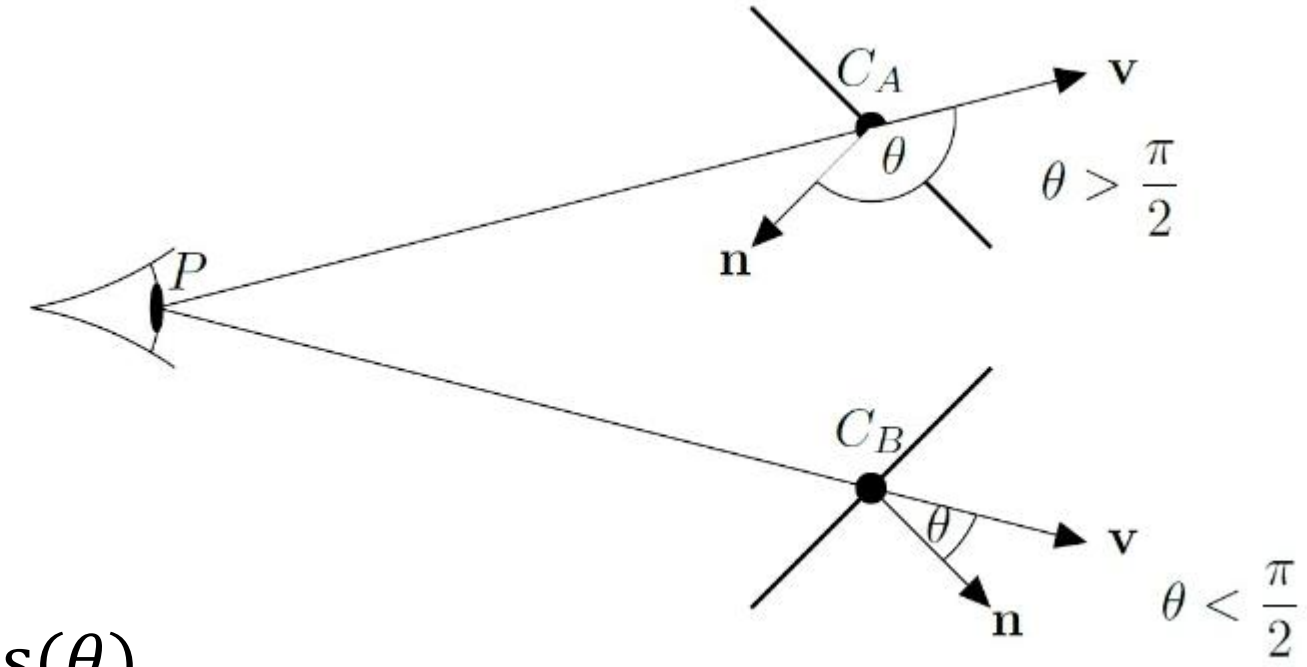

C_A
$\mathbf{v}$
θ
$\theta > \frac{\pi}{2}$
$\mathbf{n}$
P
C_B
$\mathbf{n}$


C_A
$\mathbf{v}$
θ
$\theta > \frac{\pi}{2}$
$\mathbf{n}$
P
C_B
θ
$\mathbf{v}$
$\mathbf{n}$


C_A
$\mathbf{v}$
θ
$\theta > \frac{\pi}{2}$
$\mathbf{n}$
P
C_B
θ
$\mathbf{v}$
$\mathbf{n}$
$\theta < \frac{\pi}{2}$



- $\boldsymbol{v} \cdot \boldsymbol{n} = |\boldsymbol{v}||\boldsymbol{n}|\cos(\theta)$



- $\boldsymbol{v} \cdot \boldsymbol{n} = |\boldsymbol{v}||\boldsymbol{n}|\cos(\theta)$

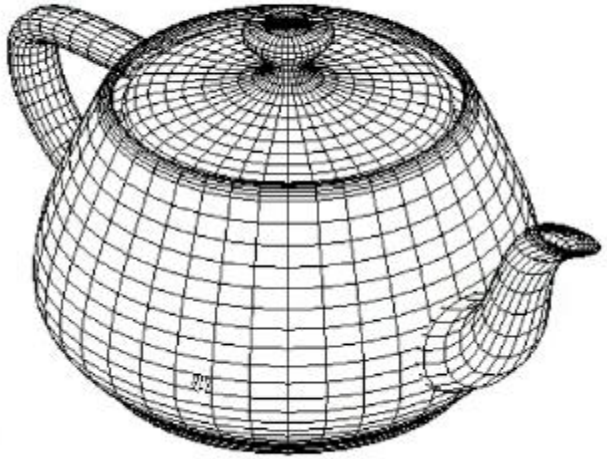
- Skierowany do przodu gdy $\theta > \frac{\pi}{2}$, dlatego $\cos(\theta) < 0$ i $\boldsymbol{v} \cdot \boldsymbol{n} < 0$

Algorithm Back face culling

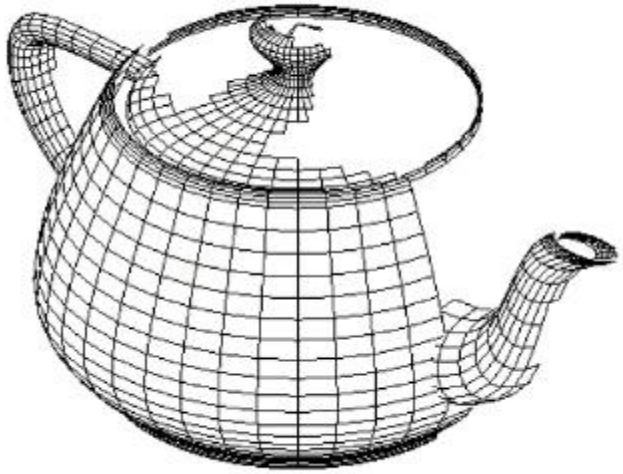
Require: Vertex co-ordinates of polygons and an viewpoint P
  for all polygons in the virtual world **do**
    calculate the normal vector $\mathbf{n}$ of the current polygon
    calculate the centre C of the current polygon
    calculate the viewing vector $\mathbf{v} = C - P$
    if $\mathbf{v} \cdot \mathbf{n} < 0$ **then**
      render current polygon
    end if
  end for

wszystkie wielokąty

backface culling

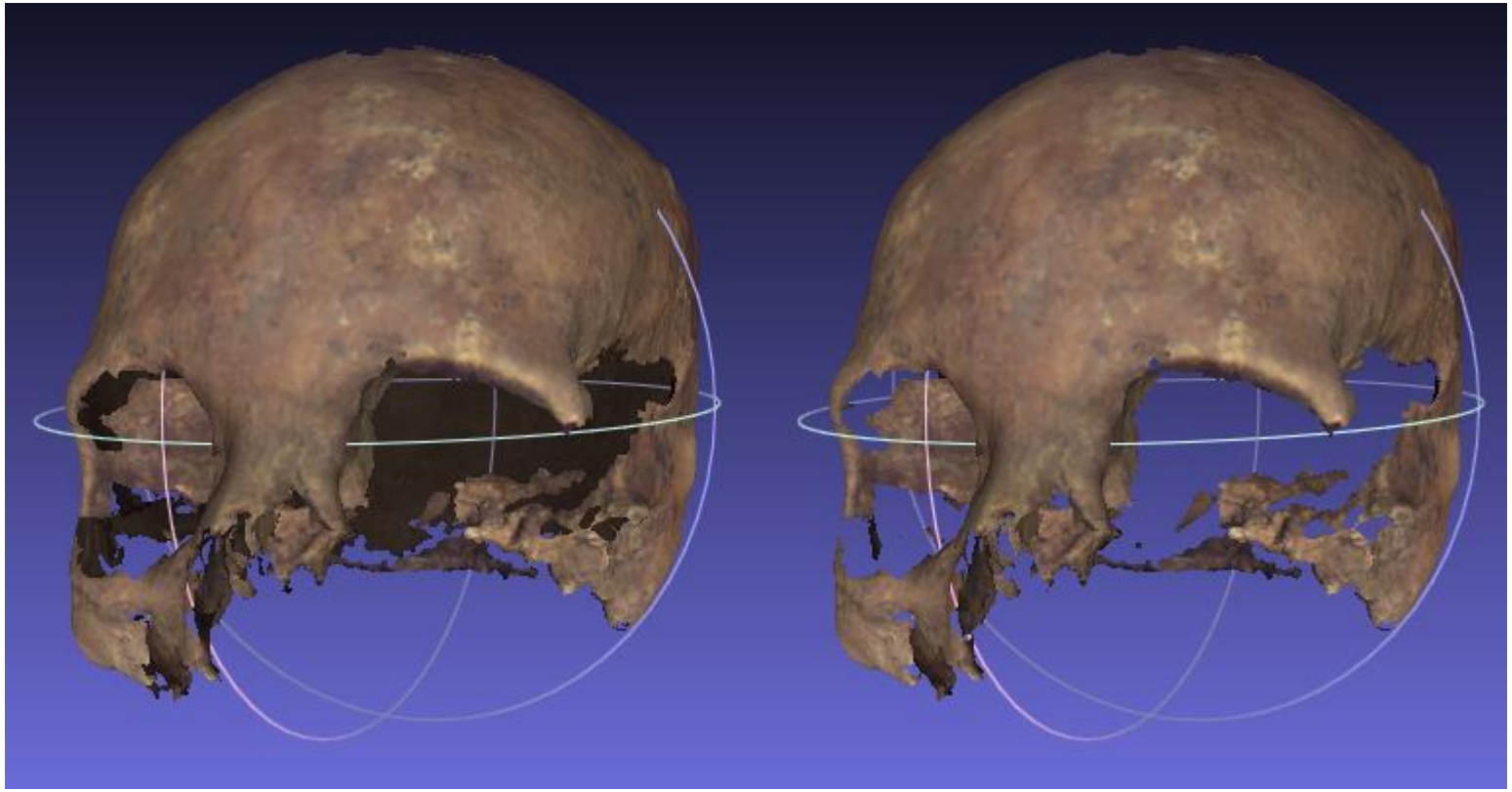


wszystkie wielokąty



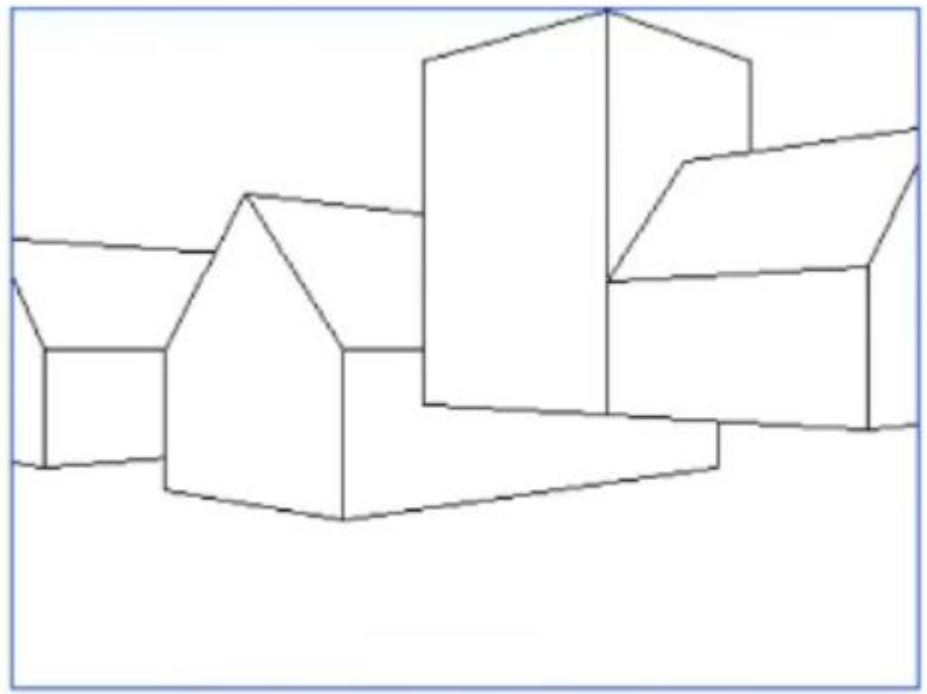
backface culling

Przykład

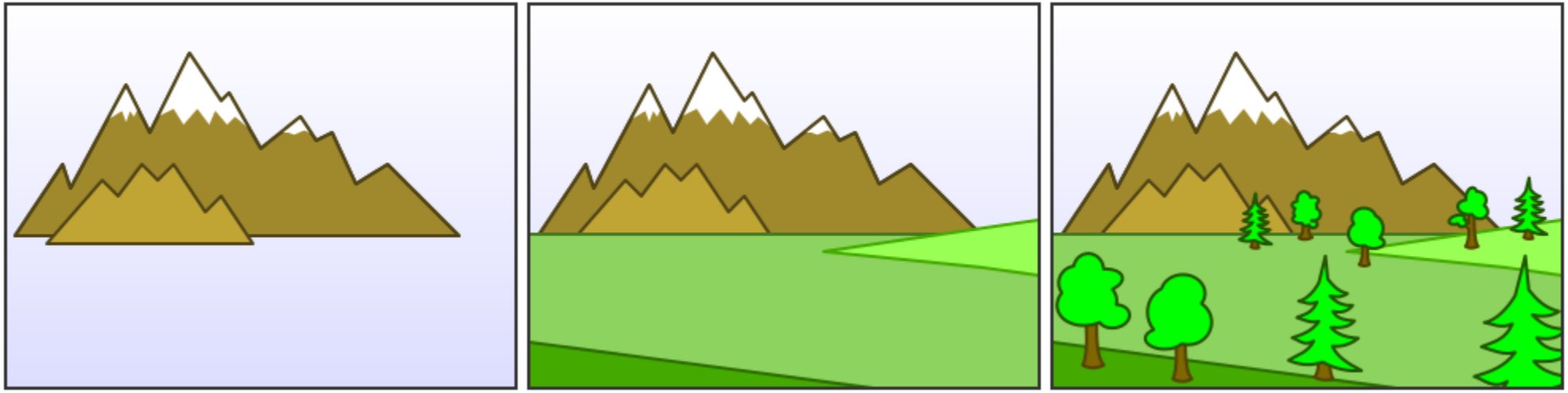


Bez backface culling

Backface culling



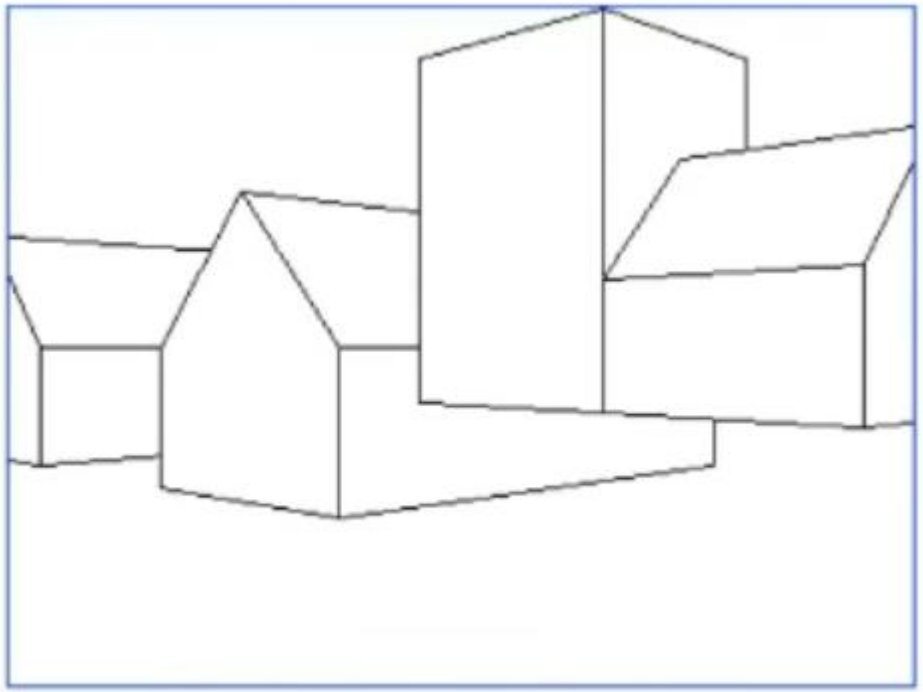
Algorytm malarza (painter's algorithm)

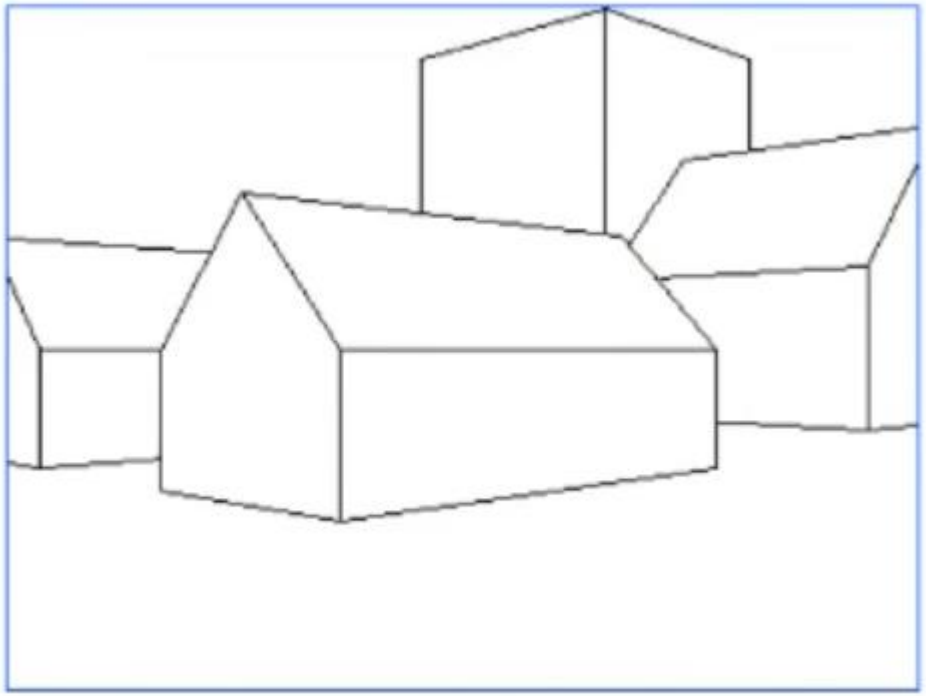


Algorytm malarza (painter’s algorithm)

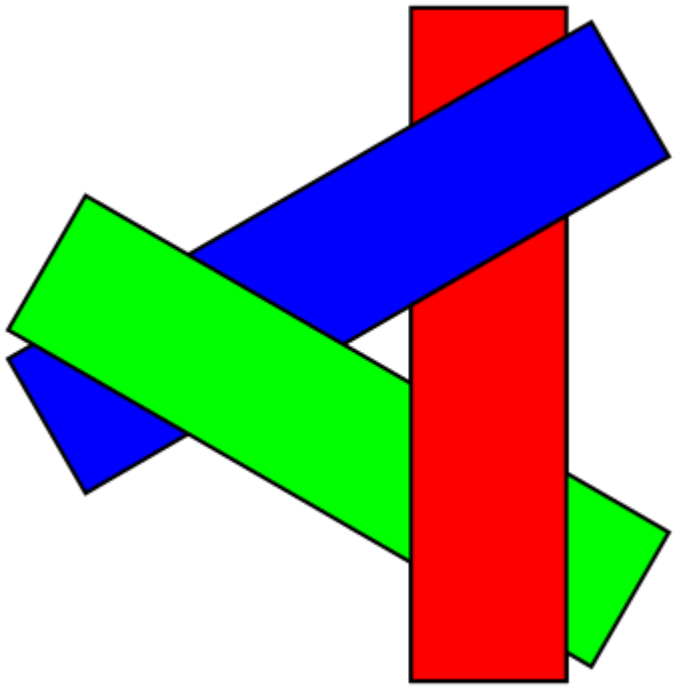
- Algorytm rysuje wielokąty w kolejności odległości od kamery
- Najmniejsza współrzędna z_s od wszystkich wielokątów jest używana żeby odległość obliczyć
- Bliższe wielokąty zasłaniają te które są dalej

Przykład





Częściowe pokrywanie wielokątów



Z-bufor

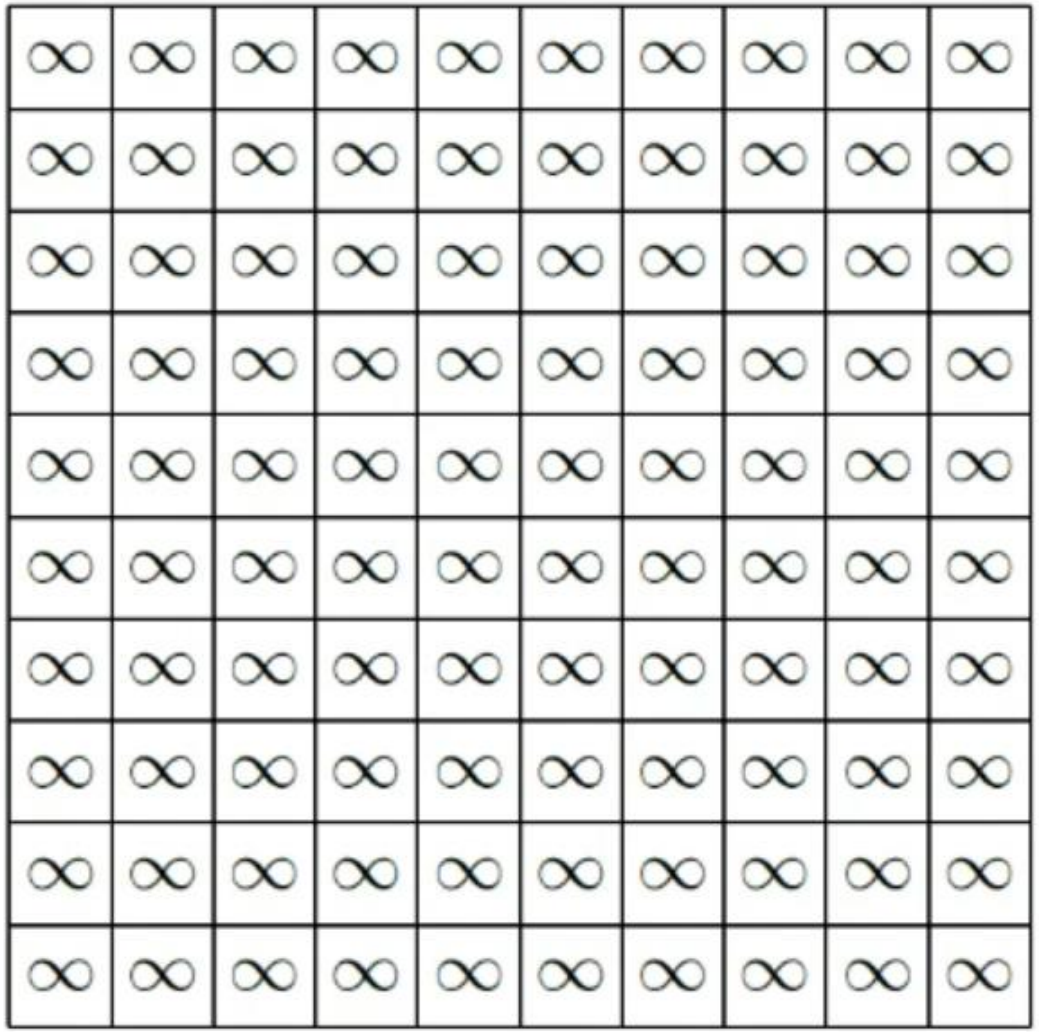
- Algorytm z-buforu jest podobny w koncepcji do algorytmu malarza ale na skali pikseli
- Dwie tablice są stworzone gdzie każdy piksel koresponduje jednemu elementu tablicy
- **Bufor głębokości** (depth buffer) zatrzymuje dystans **z** od najbliższej powierzchni w ułożonej przestrzeni świata dla każdego pikselu
- **Bufor klatek** (frame buffer) zatrzymuje indeksy wielokątów (żeby później wybrać właściwe kolory)

Z-bufor

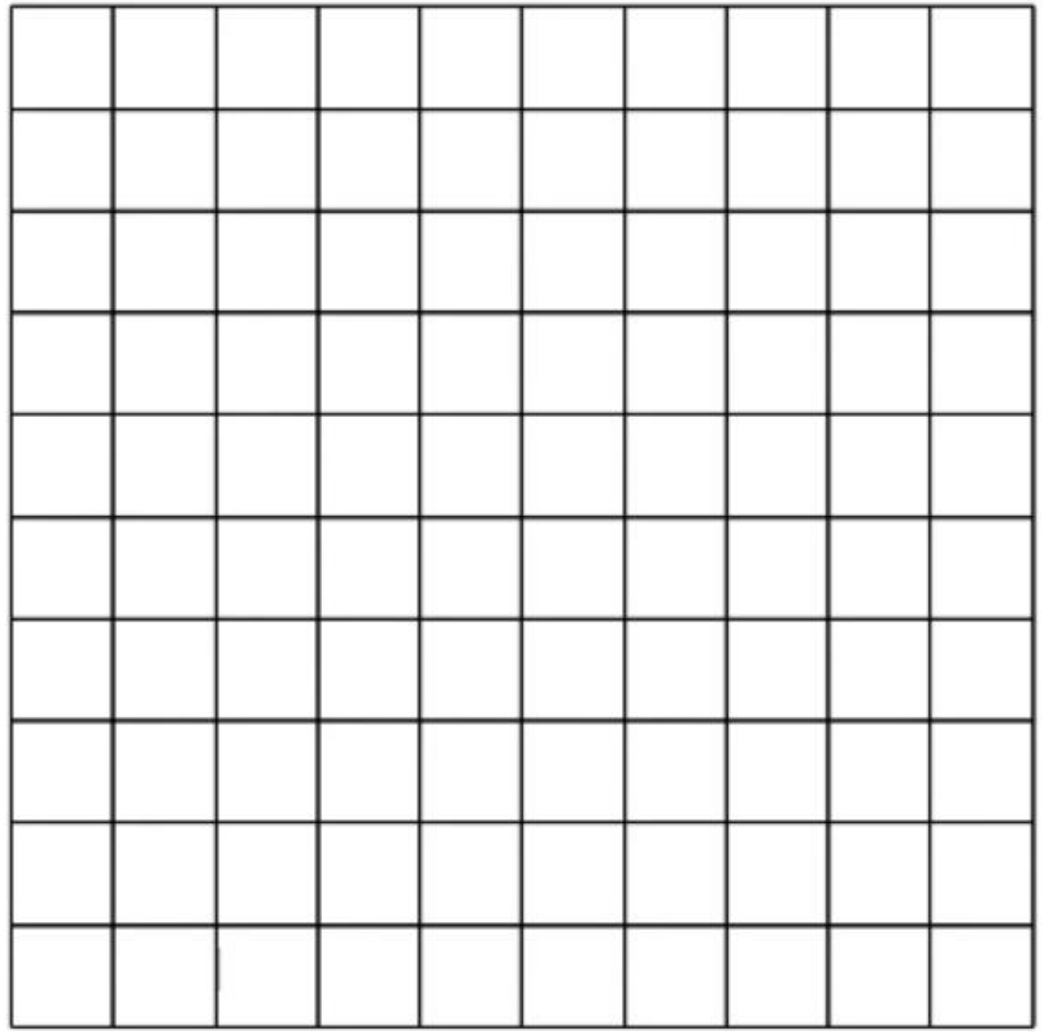
Algorithm 1 Z buffer

Require: a set of polygons P, a depth buffer array Z and a frame buffer array F

```
initialise Z to z_max
for all polygons in P do
    for all pixels in the current polygon do
        calculate the z co-ordinate of the point corresponding to the current pixel
        if z < Z(x, y) then
            Replace Z(x, y) with z
            Replace F(x, y) with the colour of the current polygon
        end if
    end for
end for
Display F on screen
```



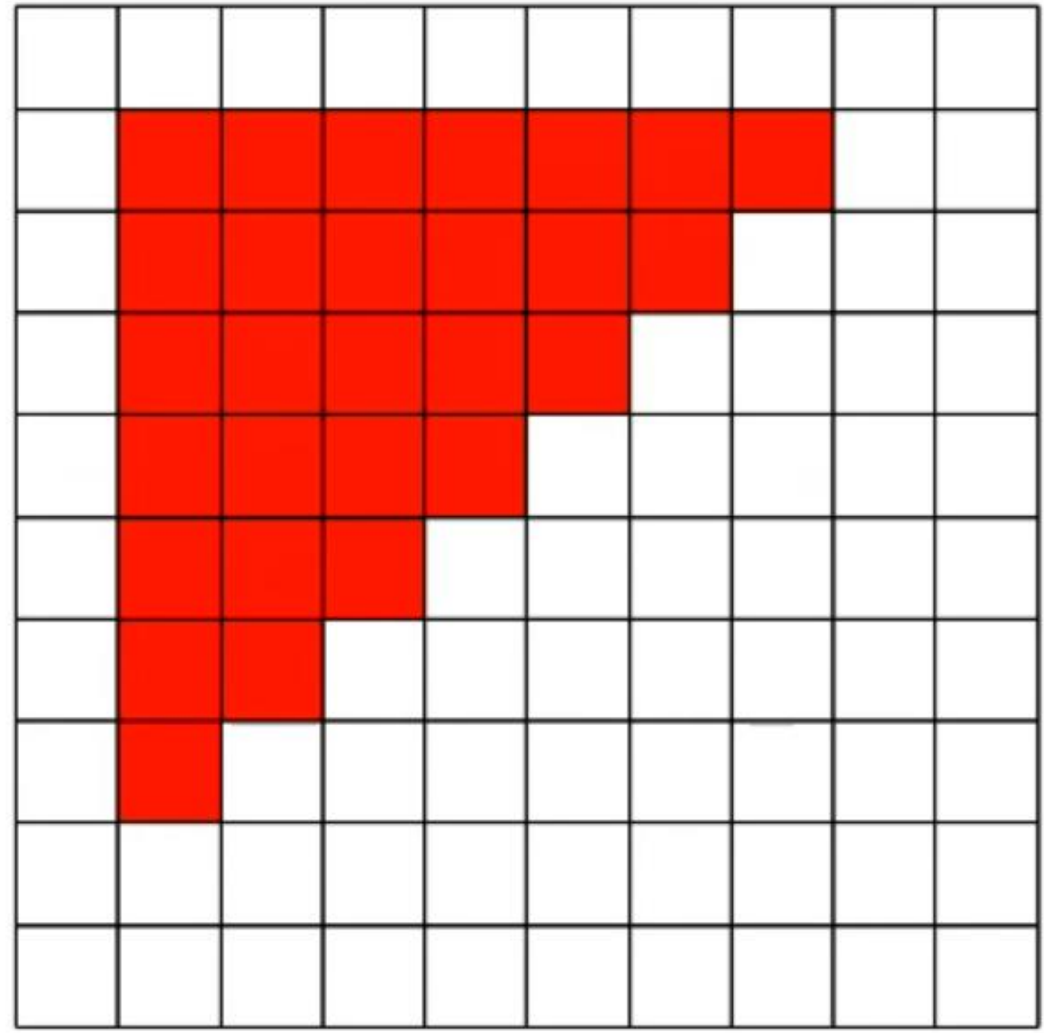
Z buffer



Frame buffer

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

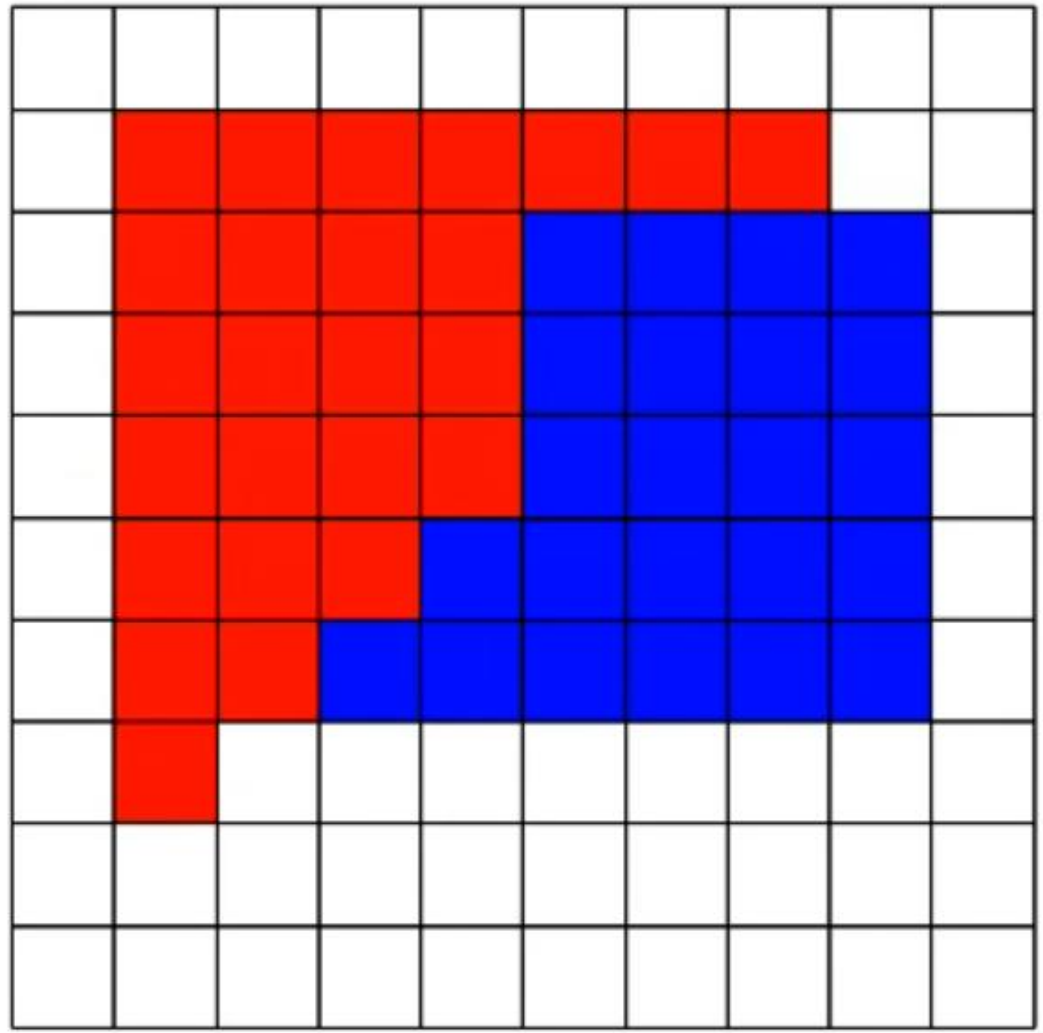
Z buffer



Frame buffer

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	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	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

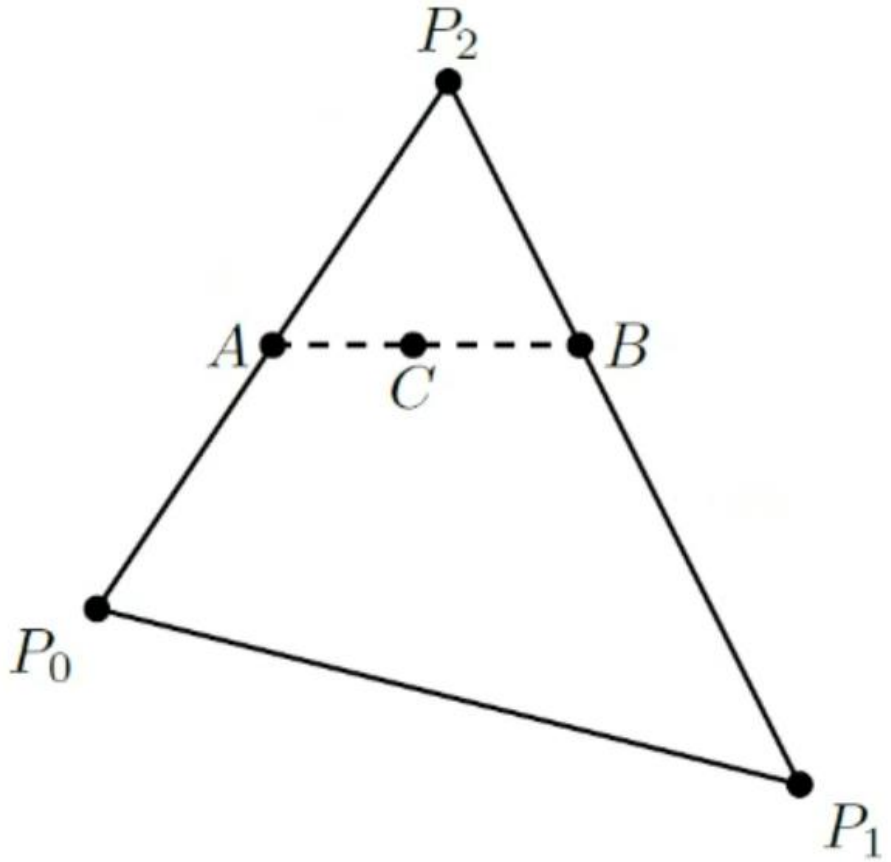
Z buffer



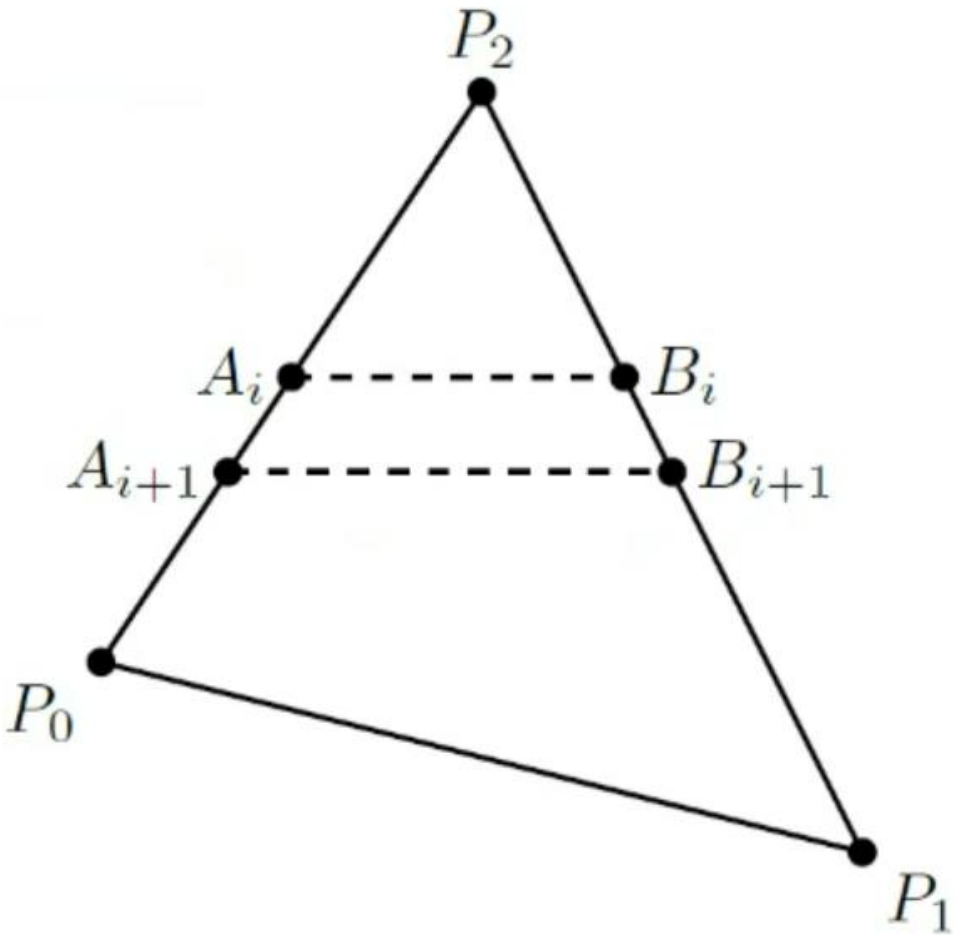
Frame buffer

Jak wewnętrzne punkty są obliczone

- Przejdź wszystkie horyzontalne wiersze pikseli (scan lines) od góry do dołu
- Oblicz piksele na lewym i prawym rogu linii skanowania przez interpolacje pomiędzy punktami P
- Oblicz piksele na linii skanowania C interpolując między nimi A i B



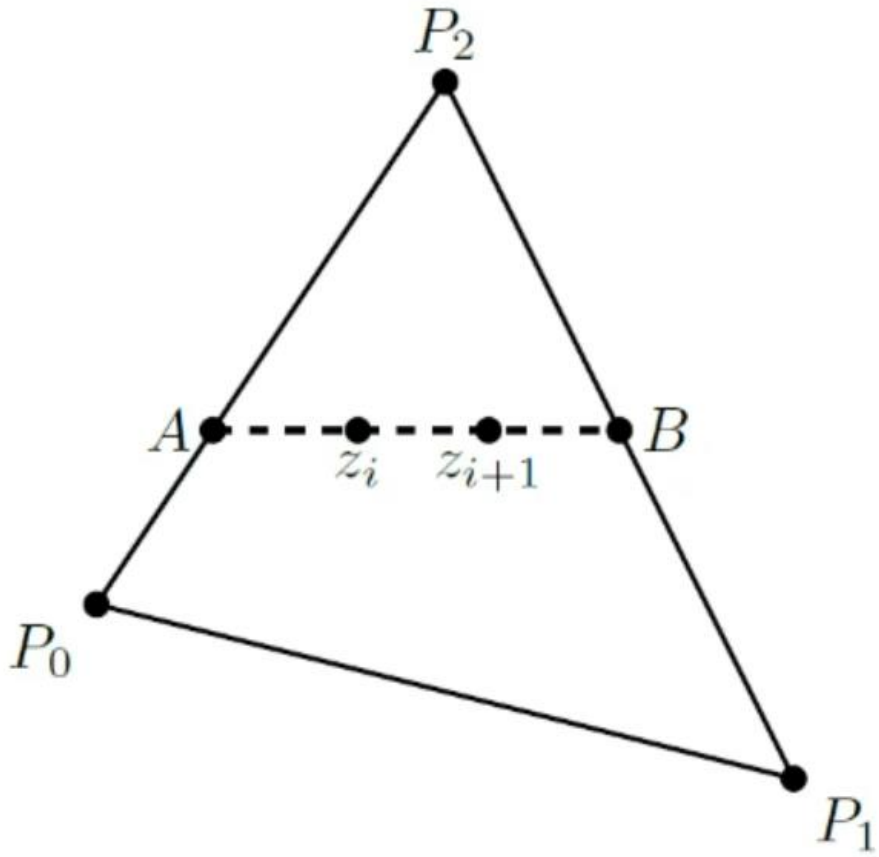
- Interpolowanie między wierzchołkami:
- $x_{A,\,i+1} = x_{A,\,i} - \Delta x_A$
- $x_{B,\,i+1} = x_{B,\,i} - \Delta x_B$
- $y_{A,\,i+1} = y_{A,\,i} - 1$
- $y_{B,\,i+1} = y_{B,\,i} - 1$
- Gdzie
- $\Delta x_A = \frac{y_A - y_0}{y_2 - y_0}$
- $\Delta x_B = \frac{y_B - y_1}{y_2 - y_1}$



Obliczanie z

- Używając równanie wektorowe płaszczyzny:
- N r = s
- Wartość z_{i+1} jest $z_{i+1} = z_i - \frac{n_x}{n_z}$

- Gdzie $\frac{n_x}{n_z}$ jest stała



Wymogi pamięciowe

- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:

Wymogi pamięciowe

- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:
- $pamiec = bufor\ glebokosci + bufor\ klatek$

Wymogi pamięciowe

- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:
- $$pamiec = bufor\ glebokosci + bufor\ klatek$$
$$= (1920\ \times 1080 \times 24) + (1920 \times 1080 \times 16)$$

Wymogi pamięciowe

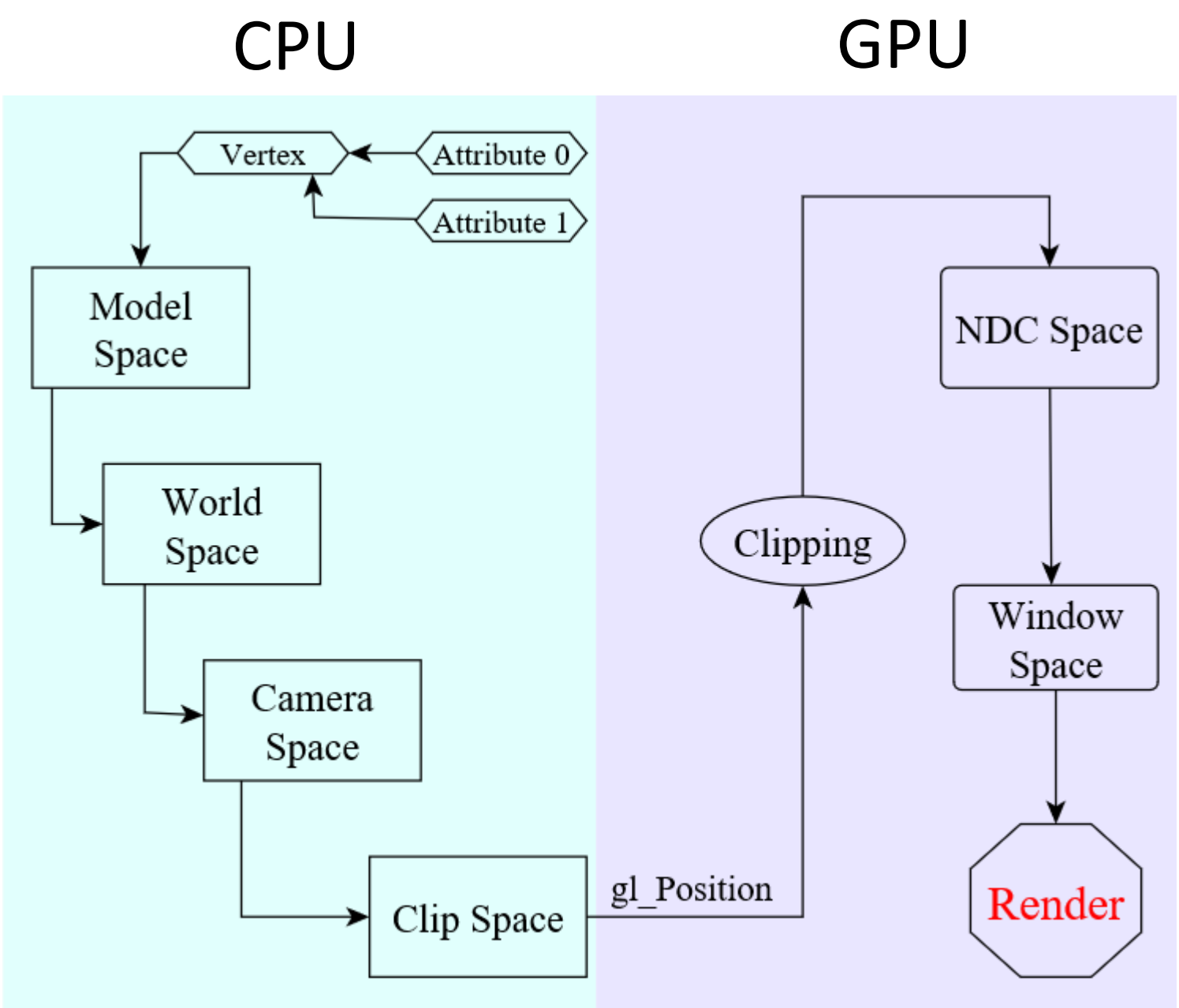
- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:
- $$pamiec = bufor\ glebokosci + bufor\ klatek$$
$$= (1920\ \times 1080 \times 24) + (1920 \times 1080 \times 16)$$
$$= 49766400 + 33{,}177{,}600$$

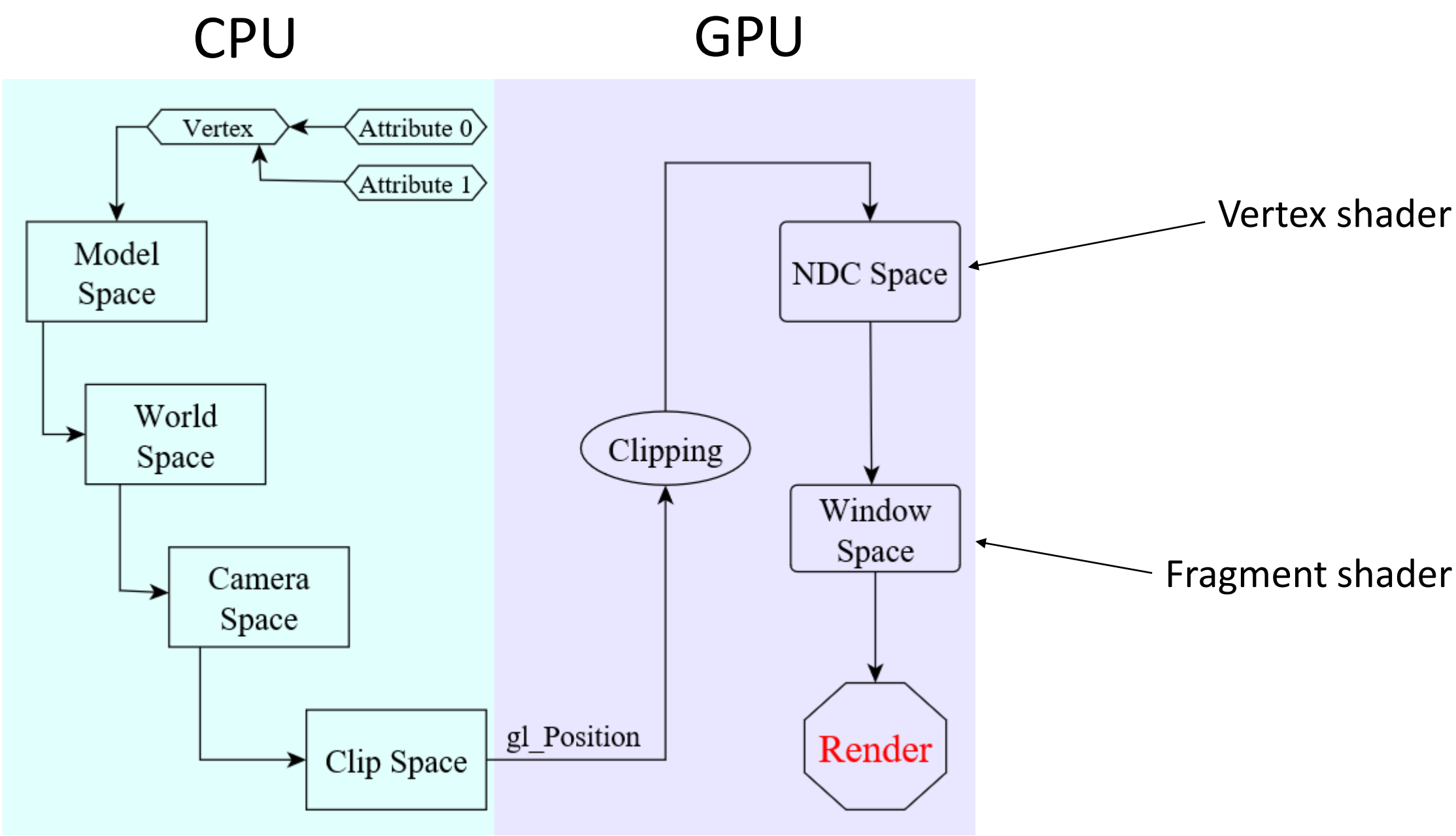
Wymogi pamięciowe

- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:
- $$pamiec = bufor\ glebokosci + bufor\ klatek$$
$$= (1920\ \times 1080 \times 24) + (1920 \times 1080 \times 16)$$
$$= 49766400 + 33{,}177{,}600$$
$$= 82{,}944{,}000b$$

Wymogi pamięciowe

- Dla wyświetlacza 1920 x 1080 pikseli używając 16b RGB model kolorowy i 24b buforu głębokości zużycie pamięci jest:
- $$pamiec = bufor\ glebokosci + bufor\ klatek$$
$$= (1920\ \times 1080 \times 24) + (1920 \times 1080 \times 16)$$
$$= 49766400 + 33{,}177{,}600$$
$$= 82{,}944{,}000b$$
$$= 10{,}368{,}600B \sim 10MB$$

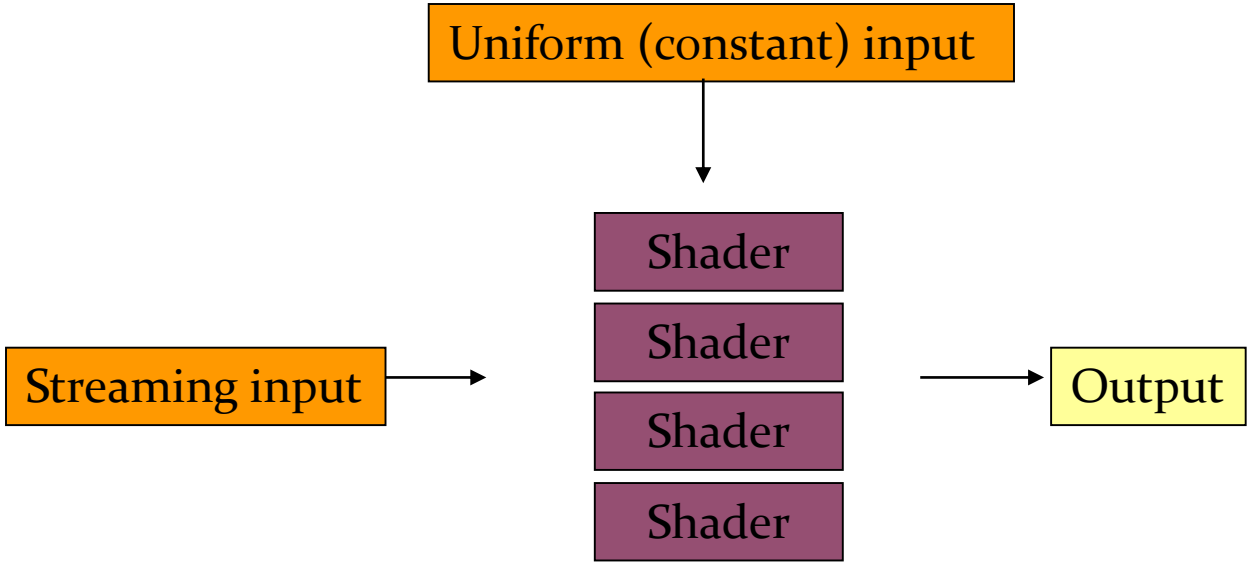

CPU
GPU
Vertex
Attribute 0
Attribute 1
Model Space
World Space
Camera Space
Clip Space
gl_Position
Clipping
NDC Space
Window Space
Render


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

GLSL syntaks

- GLSL jest jak C ale bez
  - Wskaźników
  - Rekursji
  - Dynamicznej alokacji pamięci
- GLSL jest jak C ale z
  - Domyślną obsługą wektorów i macierzy
  - Bardzo dobrą biblioteką matematyczną

GLSL syntaks: `in`/ `out`/ `uniform`



GLSL syntaks: `in`/ `out`/ `uniform`

- Przykład

```
#version 430

uniform mat4 u_ModelView;
layout (location = 0) in vec4 Position;
layout (location = 1) in vec4 Color;
out vec3 fs_Color;

void main(void)
{
  fs_Color = Color;
  gl_Position = u_ModelView * Position;
}
```

`uniform`: dane wejściowe które są stałe przez jedne wywołanie `glDraw`

`in`: dane wejściowe dla poszczególnych shaderow

`out`: dane wyjściowe shadera

GLSL syntaks

- GLSL ma preprocesora

```
#version 330

#ifdef FAST_EXACT_METHOD
  FastExact();
#else
  SlowApproximate();
#endif

// ... many others
```

- Wszystkie shadery mają funkcje main()

```
void main(void)
{
}
```

GLSL: Typy

- Typy skalarne: `float`, `int`, `uint`, `bool`
- Wektory tez są typy domyslne:
  - `vec2`, `vec3`, `vec4`
  - `ivec*`, `uvec*`, `bvec*`
- Elementy wektorów można na kilka sposobów wywołać:
  - .x, .y, .z, .w pozycja lub kierunek
  - .r, .g, .b, .a kolor
  - .s, .t, .p, .q współrzędne tekstur

`mojKolor.xyz`

~~`mojKolor.xgb`~~

GLSL: Wektory

- Konstruktory:

```
vec3 xyz = vec3(1.0, 2.0, 3.0);

vec3 xyz = vec3(1.0); // [1.0, 1.0, 1.0]

vec3 xyz = vec3(vec2(1.0, 2.0), 3.0);
```

GLSL: Wektory

- Swizzle: selekcja lub zamiana elementów

```
vec4 c = vec4(0.5, 1.0, 0.8, 1.0);

vec3 rgb = c.rgb;  // [0.5, 1.0, 0.8]

vec3 bgr = c.bgr;  // [0.8, 1.0, 0.5]

vec3 rrr = c.rrr;  // [0.5, 0.5, 0.5]

c.a = 0.5;         // [0.5, 1.0, 0.8, 0.5]
c.rb = 0.0;        // [0.0, 1.0, 0.0, 0.5]

float g = rgb[1];  // 1.0, indeksowanie
```

GLSL: Macierze

- Macierze są typy domyślne:
  - kwadratowa: `mat2`, `mat3`, `mat4`
  - M x N: `mat`*`m`*`x`*`n`* `m` kolumn, `n` wierszy

- Zapisywanie macierzy:

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

GLSL: Macierze

- Konstruktory

```
mat3 i = mat3(1.0); // 3x3 macierz jednostkowa

mat2 m = mat2(1.0, 2.0,  // [1.0 3.0]
              3.0, 4.0); // [2.0 4.0]
```

- Elementy macierzy

```
float f = m[kolumna][wiersz];

float x = m[0].x; // element x pierwszej kolumny

vec2 yz = m[1].yz; // elementy yz drugiej kolumny
```

Macierz jako tablica wektorów

Swizzle

GLSL: Wektory i Macierze

- Macierz wektor operacje:

```
vec3 xyz = // ...

vec3 v0 = 2.0 * xyz; // skalowanie
vec3 v1 = v0 + xyz;  // addycja
vec3 v2 = v0 * xyz;  // iloczyn elementów

mat3 m = // ...
mat3 v = // ...

mat3 mv = v * m;       // macierz * macierz
mat3 xyz2 = mv * xyz; // macierz * wektor
mat3 xyz3 = xyz * mv; // wektor * macierz
```

GLSL domyślne funkcje

- Wybrane funkcje trygonometryczne

```
float s = sin(theta);
float c = cos(theta);
float t = tan(theta);

float theta = asin(s);
// ...

vec3 angles = vec3(/* ... */);
vec3 vs = sin(angles);
```

W radianach

Funkcje również obsługują wektory

GLSL domyślne funkcje

- Funkcje nieliniowe

```
float x1 = pow(x, y);
float x2 = exp(x);
float x3 = exp2(x);

float l = log(x);   // ln
float l2 = log2(x); // log2

float s = sqrt(x);
float is = inversesqrt(x);
```

GLSL domyślne funkcje

- Inne funkcje

```
float ax = abs(x);  // wartość absolutna
float sx = sign(x); // -1.0, 0.0, 1.0

float m0 = min(x, y); // wartość minimalna
float m1 = max(x, y); // wartość maksymalna
float c = clamp(x, 0.0, 1.0);

// wiele innych:  floor(), ceil(),
// step(), smoothstep(), ...
```

GLSL domyślne funkcje

- Przykład

```
float x = // ...
float f;

if (x > 0.0)
{
  f = 2.0;
}
else
{
  f = -2.0;
}
```

```
sign(x)*2.0
```

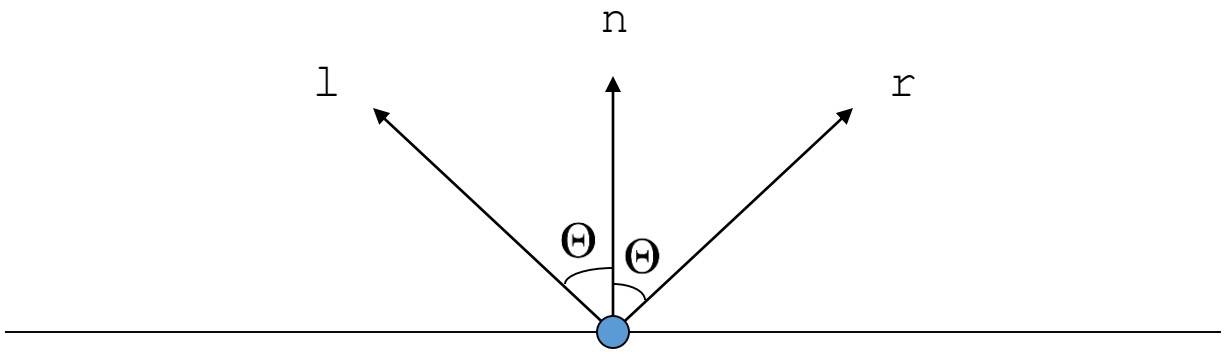
GLSL domyślne funkcje

- Wybrane funkcje geometryczne

```
vec3 l = // ...
vec3 n = // ...
vec3 p = // ...
vec3 q = // ...

float f = length(l);       // długość wektora
float d = distance(p, q); // dystans pomiędzy dwoma punktami

float d2 = dot(l, n);      // iloczyn skalarny
vec3 v2 = cross(l, n);     // iloczyn wektorowy
vec3 v3 = normalize(l);    // normalizacja

vec3 v3 = reflect(l, n);   // refleksja
```

GLSL domyślne funkcje

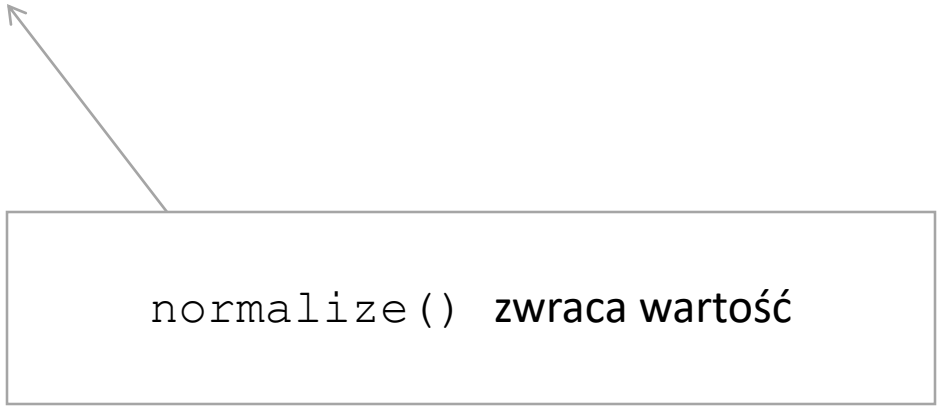
- `reflect(-l, n)`
  - Mając ***l*** i ***n***, znajdź ***r***.



GLSL domyślne funkcje

- Co jest nie poprawne w tym kodzie?

```
vec3 n = // ...

normalize(n);
```



GLSL domyślne funkcje

- Wybrane metody numeryczne

```
mat4 m = // ...

mat4 t = transpose(m);              //transpozycja
float d = determinant(m);           //wyznacznik
mat4 d = inverse(m);                //macierz odwrotna
```