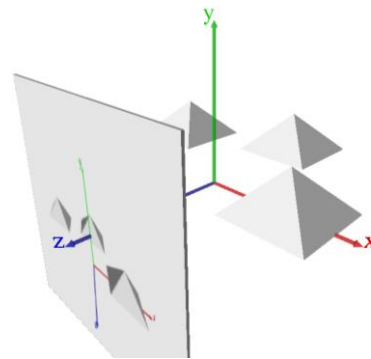
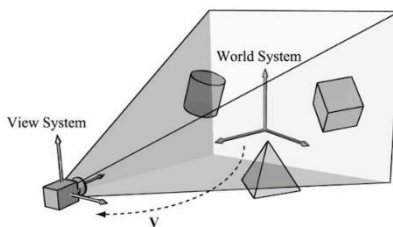


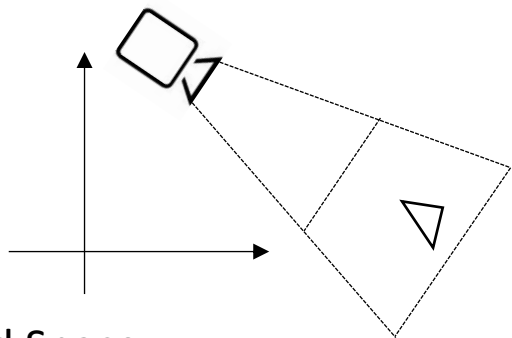
GRK 3

dr Wojciech Palubicki

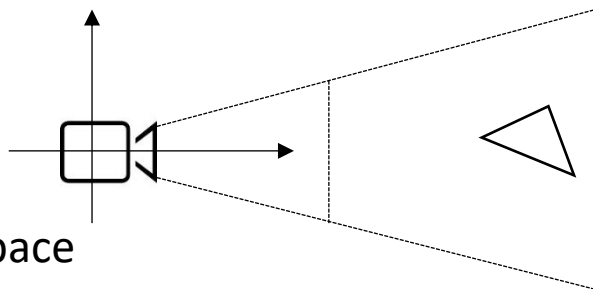
Potok graficzny



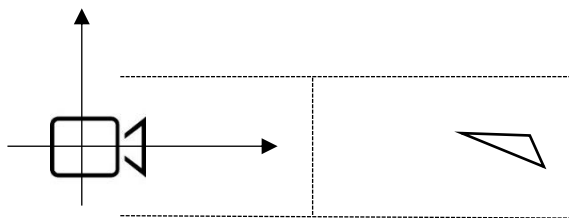
Potok graficzny



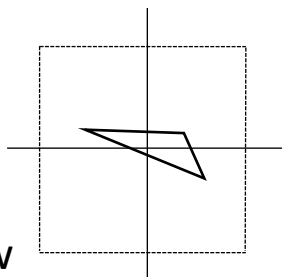
World Space



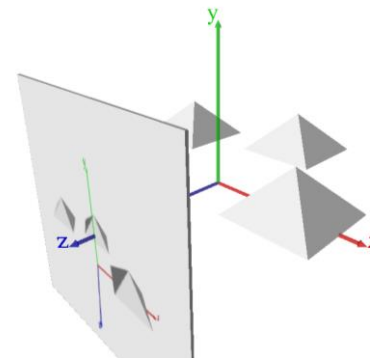
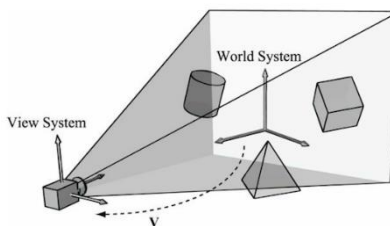
View Space



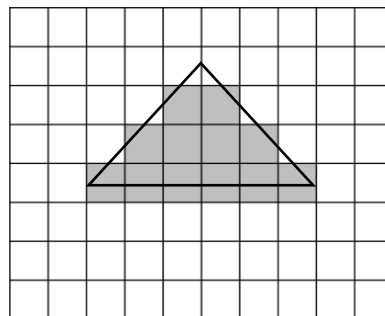
Orthographic view



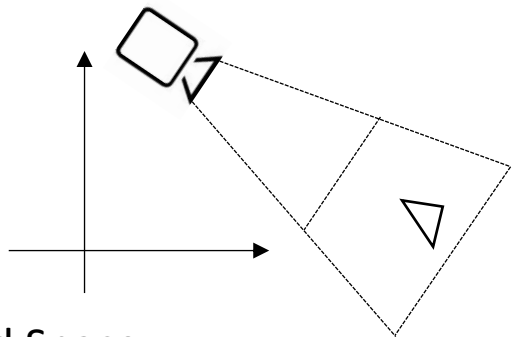
Canonincal view



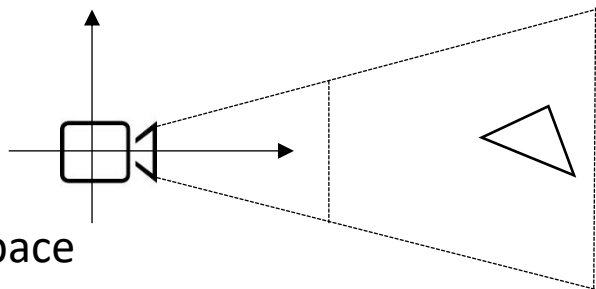
Window space



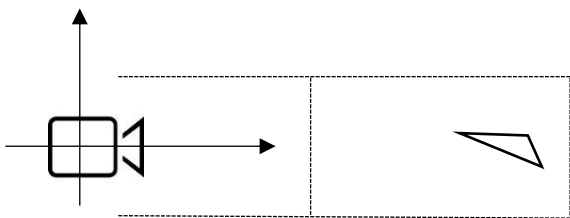
Potok graficzny



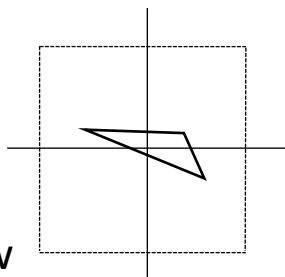
World Space



View Space

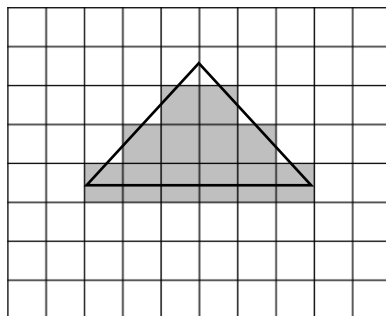


Orthographic view



Canonincal view

Window space



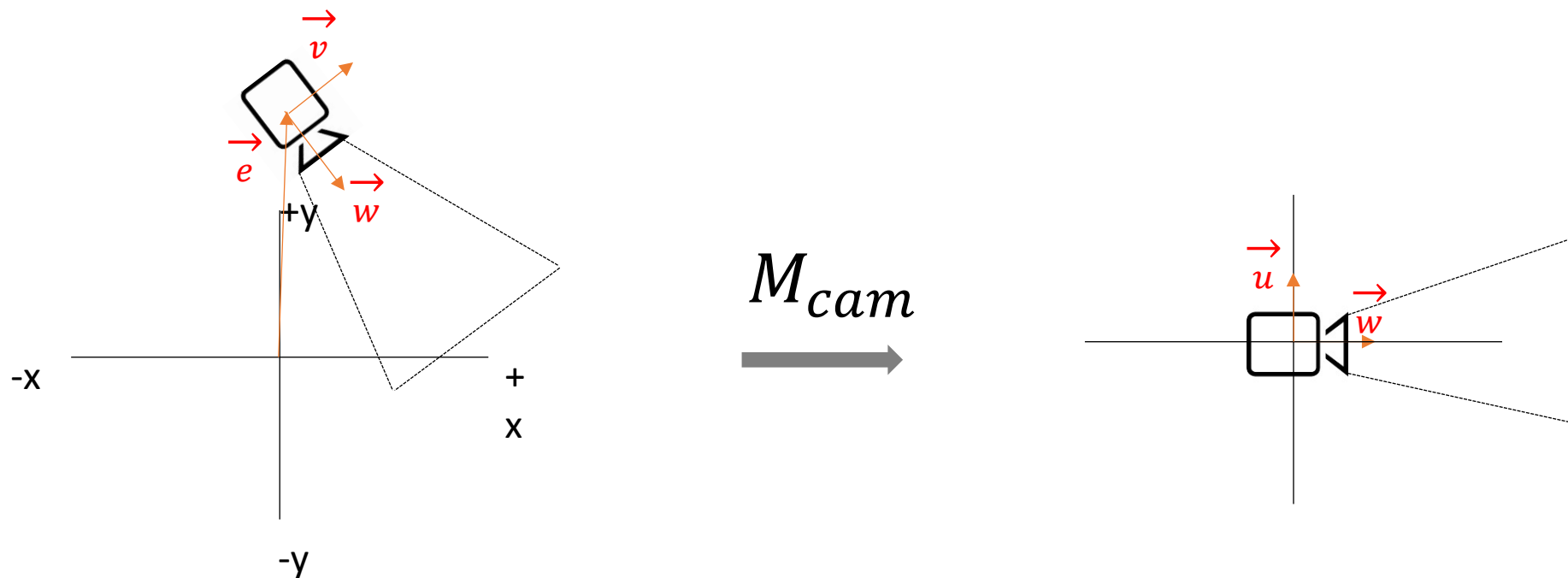
- Potok graficzny z **ortograficznym** rzutem:

$$M_{vp} M_{ort} M_{cam} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Potok graficzny z **perspektywicznym** rzutem:

$$M_{vp} M_{per} M_{cam} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Wyrównanie systemów współrzędnych

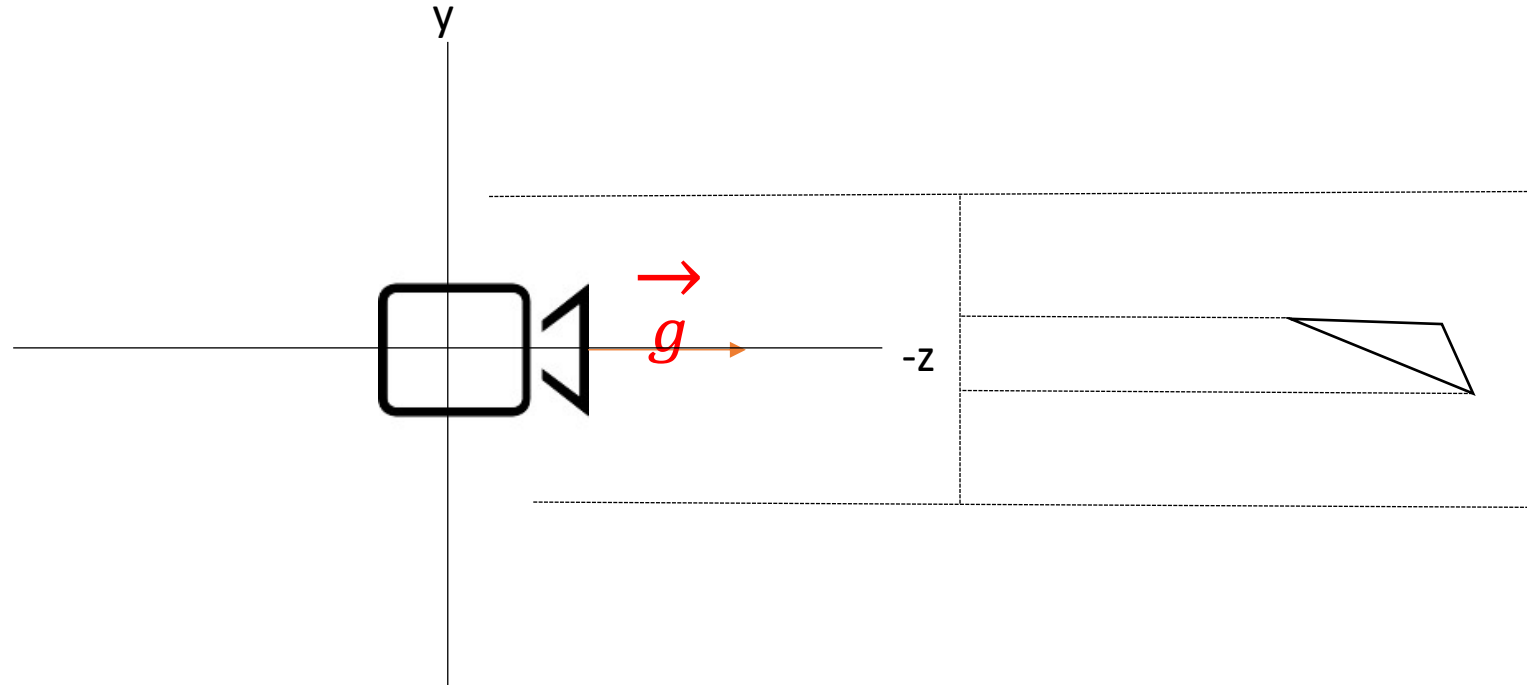


$$M_{cam}$$

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

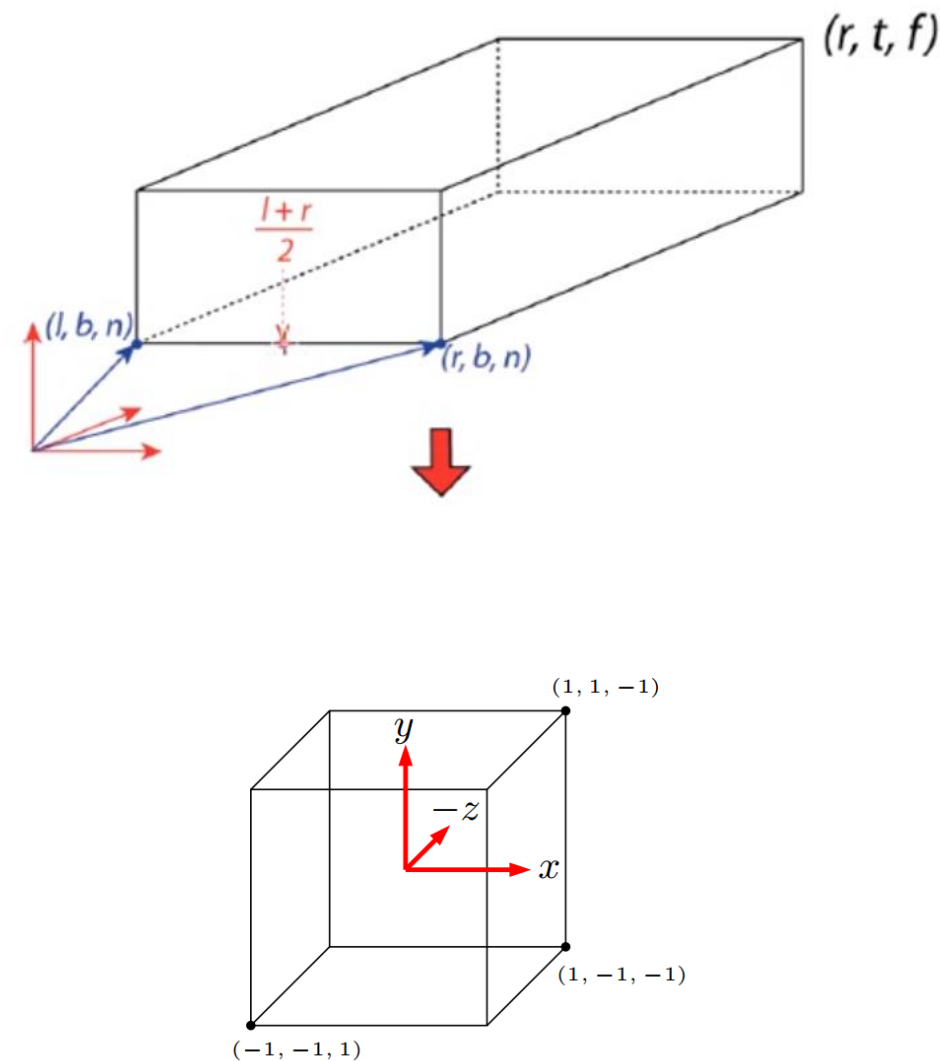
$$M_{cam} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rzutowanie równoległe



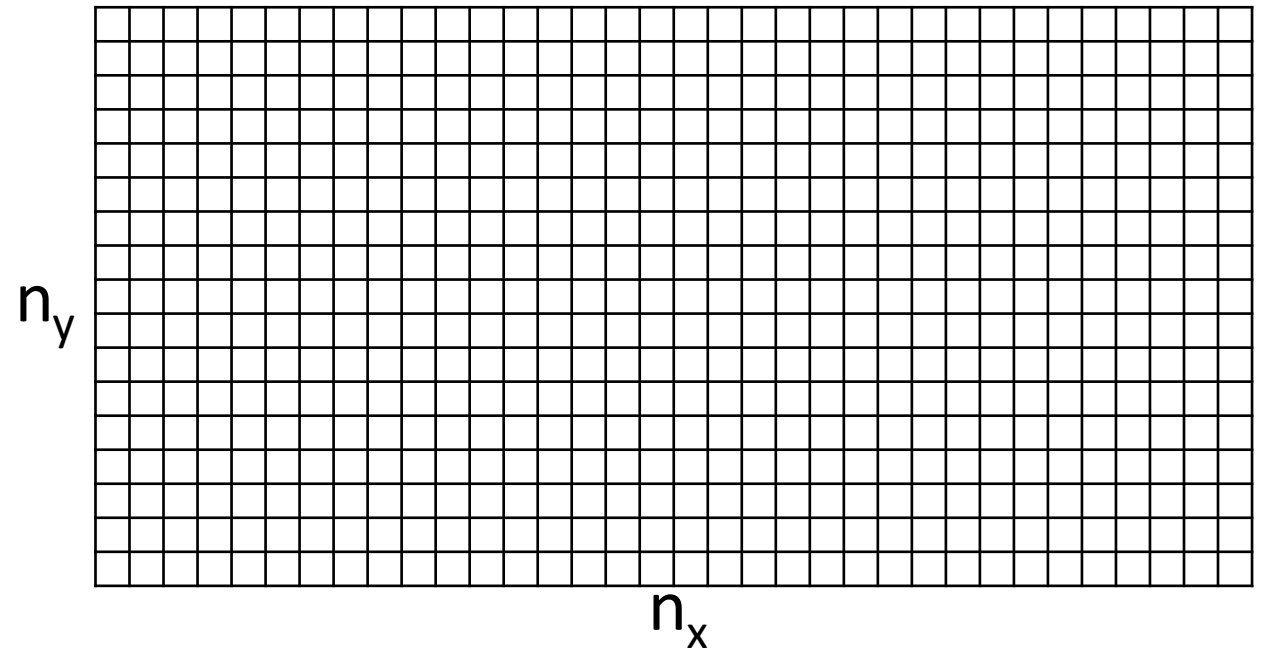
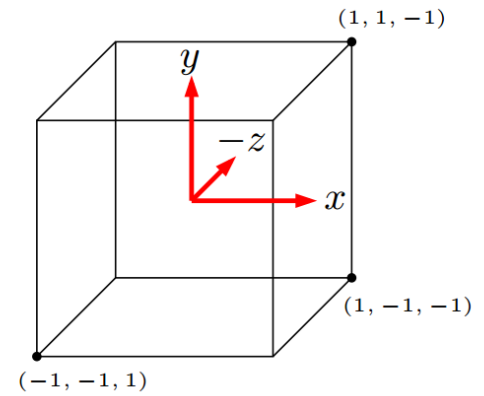
Ortograficzna bryła widzenia

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

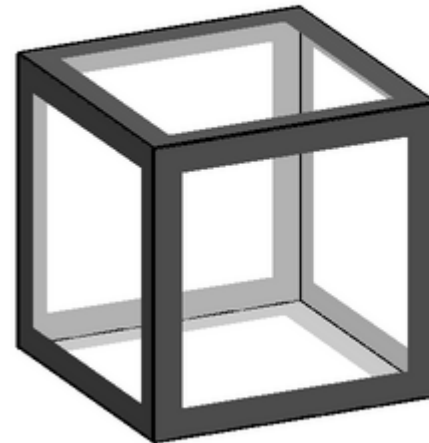


Przekształcenie bryły kanonicznej

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



Rzutowania

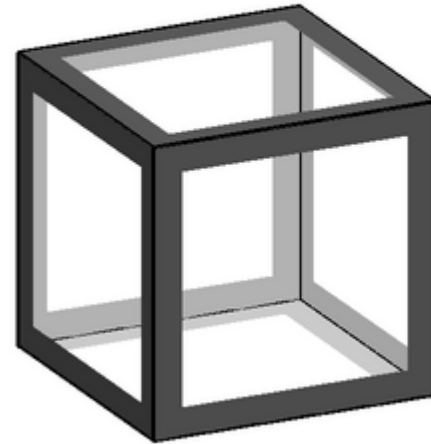


Ortograficzne rzutowanie

Rzutowania



Perspektywicznie rzutowanie



Ortograficznie rzutowanie

Macierz P

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

System współrzędnych jednorodnych

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

$$x' = a_1x + b_1y + c_1z$$

System współrzędnych jednorodnych

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

$$x' = a_1x + b_1y + c_1z$$

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

$$x' = a_1x + b_1y + c_1z + d_1$$

System współrzędnych jednorodnych

- We współrzędnych jednorodnych wektor:

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

Teraz wprowadzamy możliwość reprezentacji punktów za pomocą:

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

System współrzędnych jednorodnych

- Wtedy transformacja macierzowa staje się:

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

System współrzędnych jednorodnych

- Wtedy transformacja macierzowa staje się:

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

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

System współrzędnych jednorodnych

- Wtedy transformacja macierzowa staje się:

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

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

System współrzędnych jednorodnych

- Wtedy transformacja macierzowa staje się:

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

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

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:

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

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:

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

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

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:

$$\begin{bmatrix} nx \\ \hline \textcolor{red}{z} \\ ny \\ \hline \textcolor{red}{z} \\ z \\ 1 \end{bmatrix}$$

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

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:

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

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

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:

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

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

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?

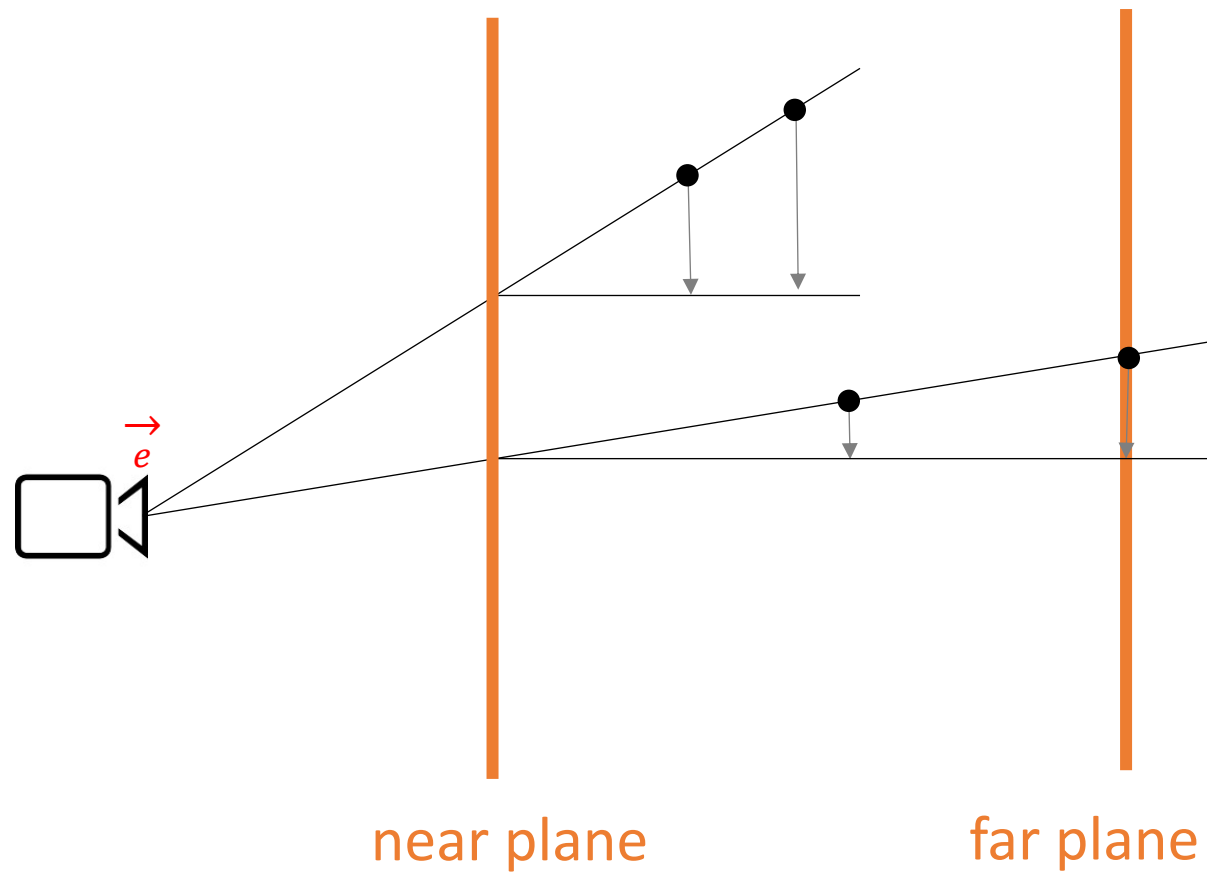
$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} nx \\ \hline z \\ ny \\ \hline z \\ \textcolor{red}{z} \\ 1 \end{bmatrix}$$

Macierz P

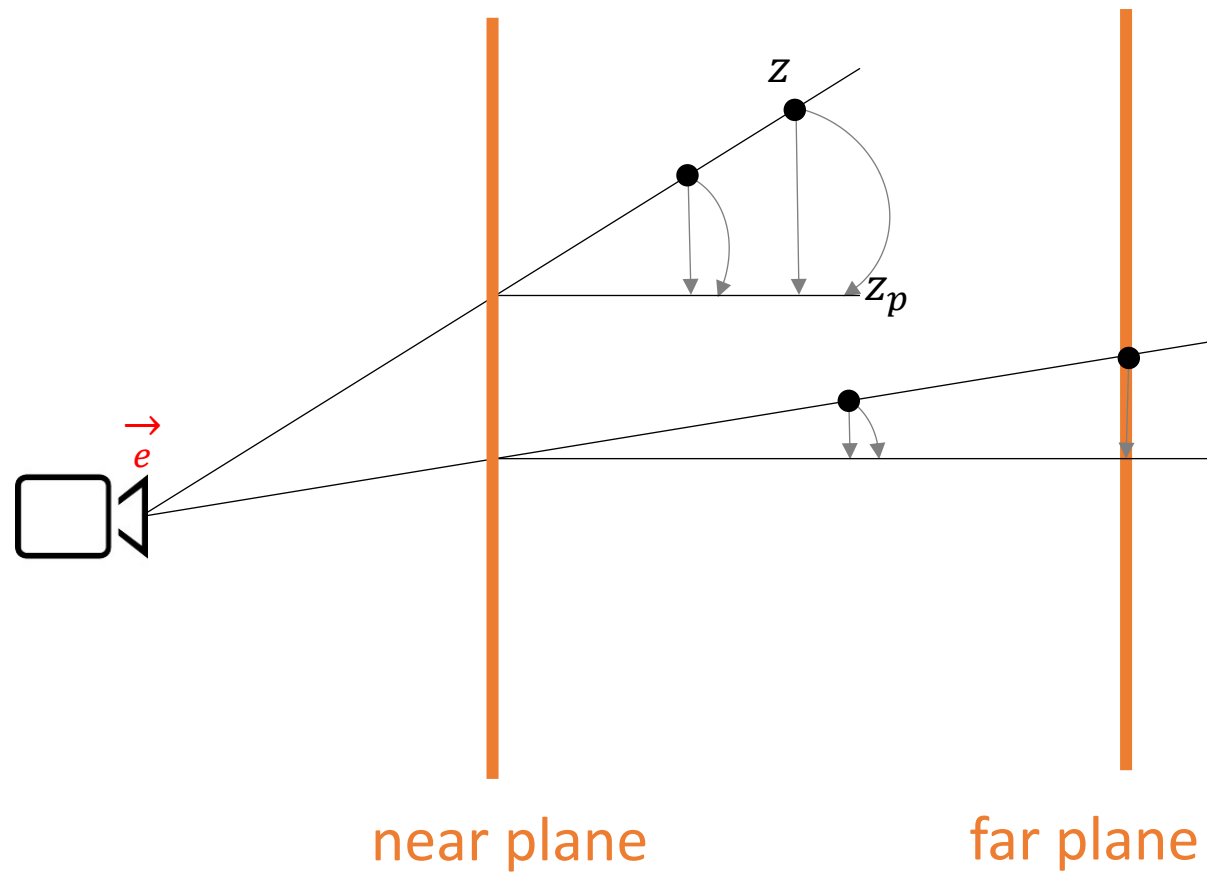
$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{nf}{z} \\ 1 \end{bmatrix}$$

Transformacja bryły widzenia



Transformacja bryły widzenia

$$z_p \sim \frac{1}{z}$$



Uproszczony Potok Graficzny (Rendering)

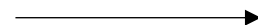
Model Matrix



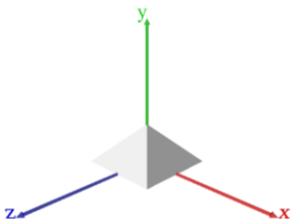
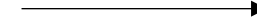
View Matrix



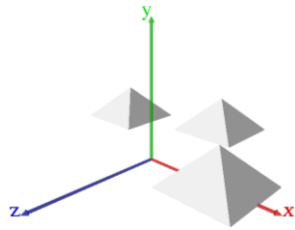
Projection Matrix



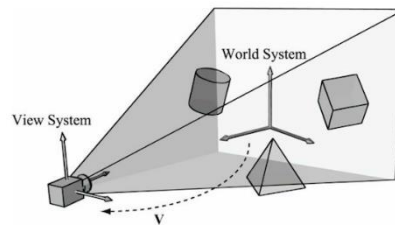
Viewport Transform



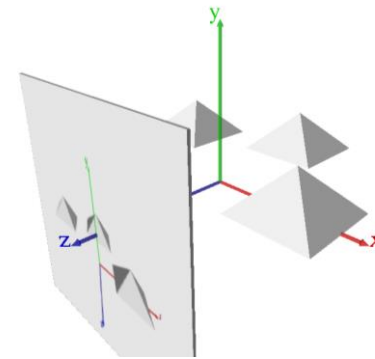
Object Space



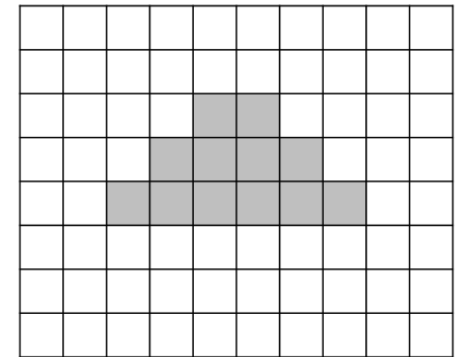
World Space



View Space



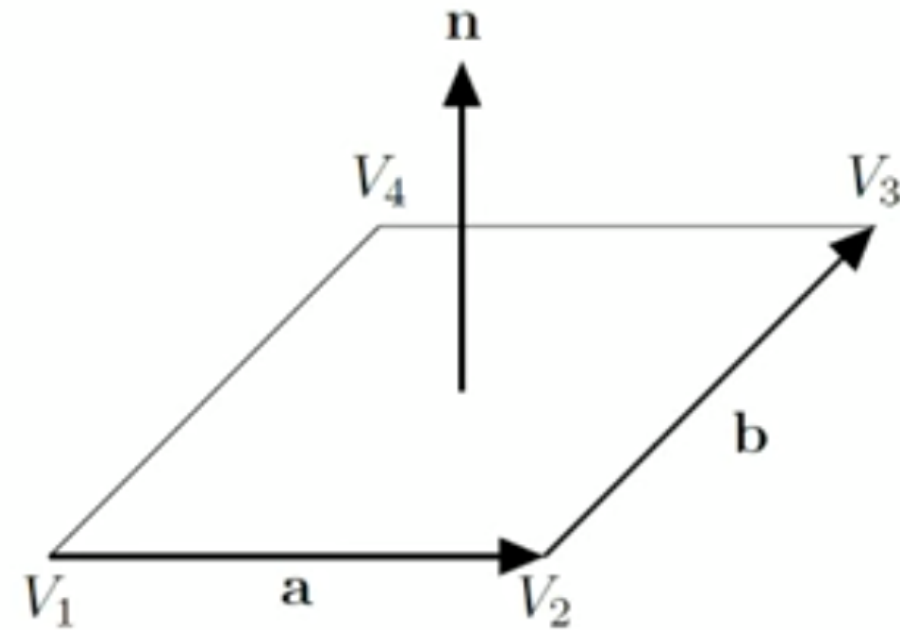
Clip Space



Screen Space

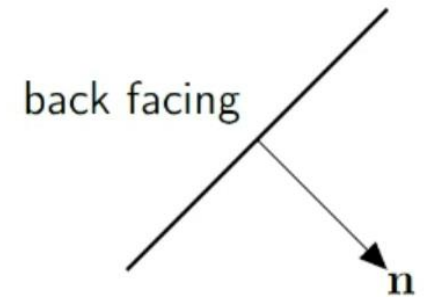
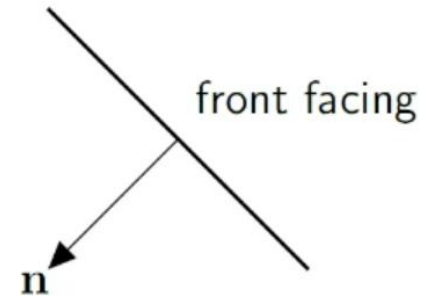
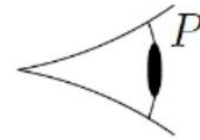
Wektor normalny

- **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
- $n = (V_2 - V_1) \times (V_3 - V_2)$



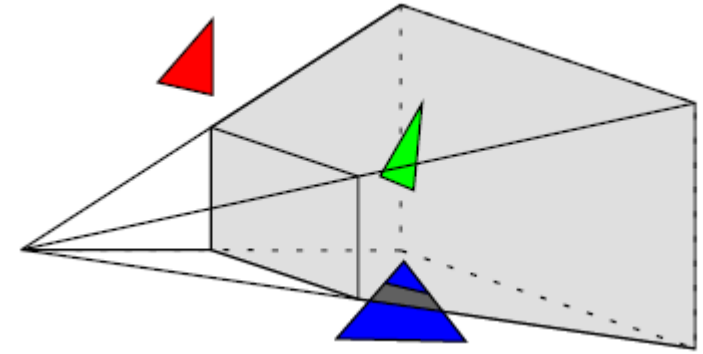
Skierowanie wielokątów

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



Jakie trójkąty rzutować?

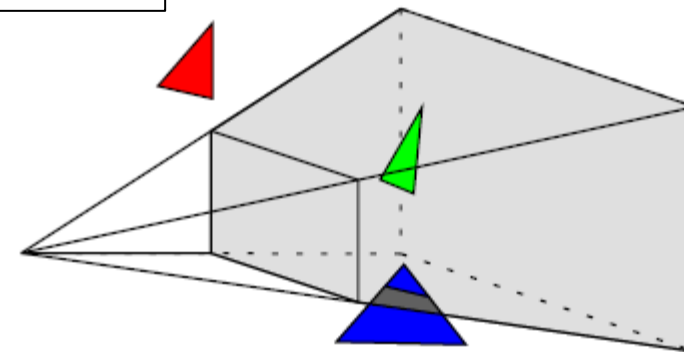
- Trójkąty które leżą (częściowo) poza bryłą widzenia nie muszą być rzutowane i są cięte (**clipping/culling**)



Jakie trójkąty rzutować?

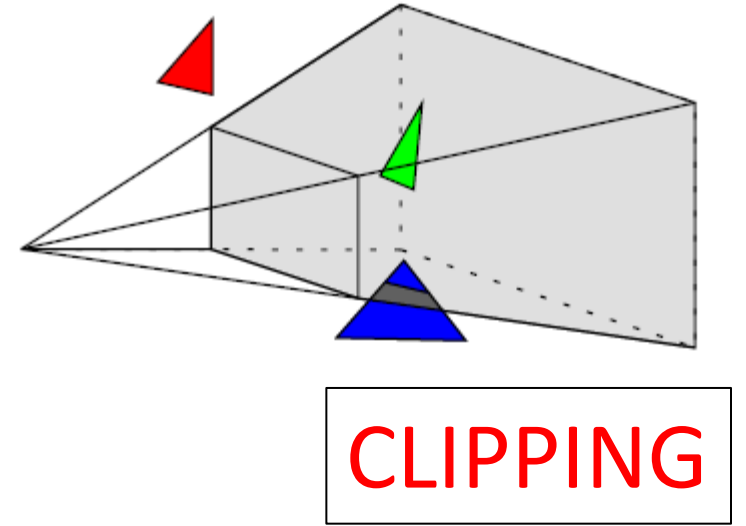
CULLING

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



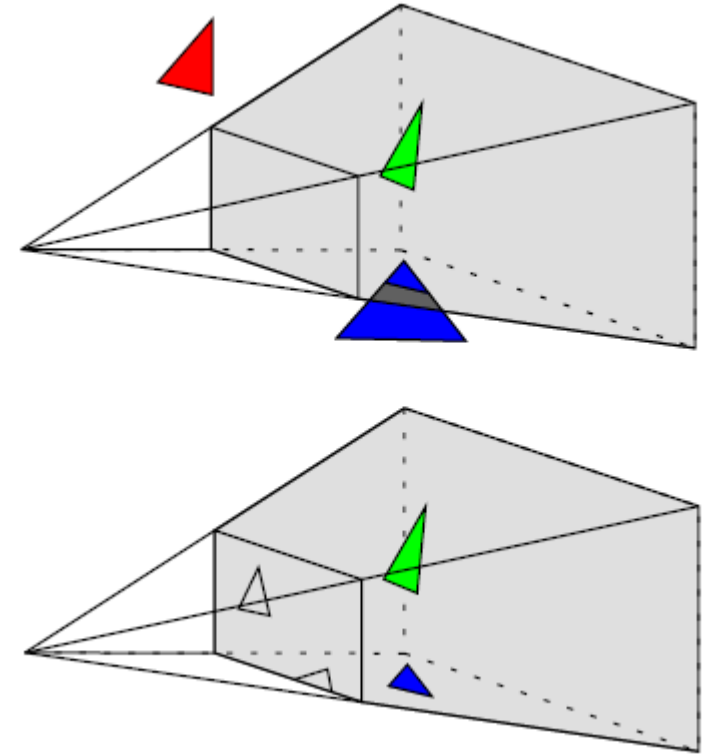
Jakie trójkąty rzutować?

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



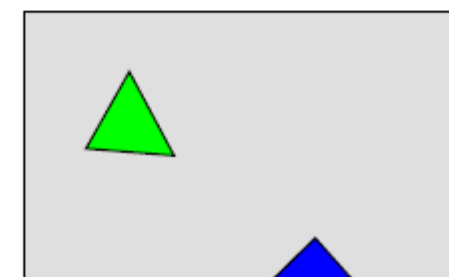
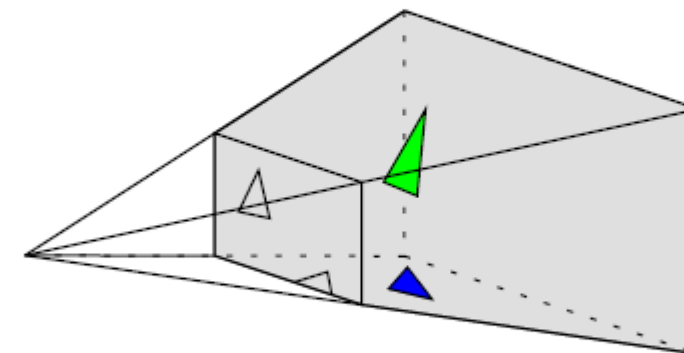
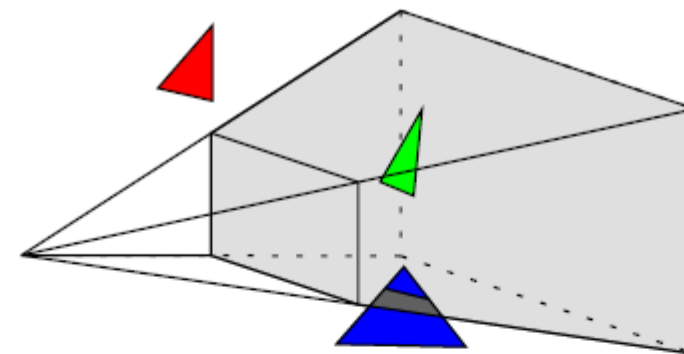
Jakie trójkąty rzutować?

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



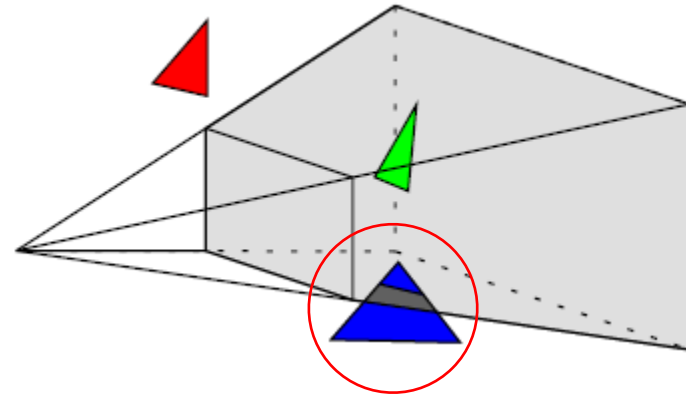
Jakie trójkąty rzutować?

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



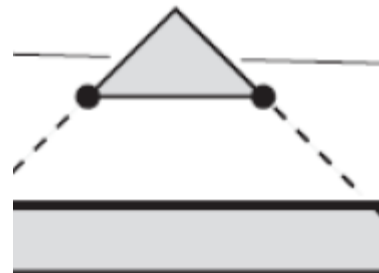
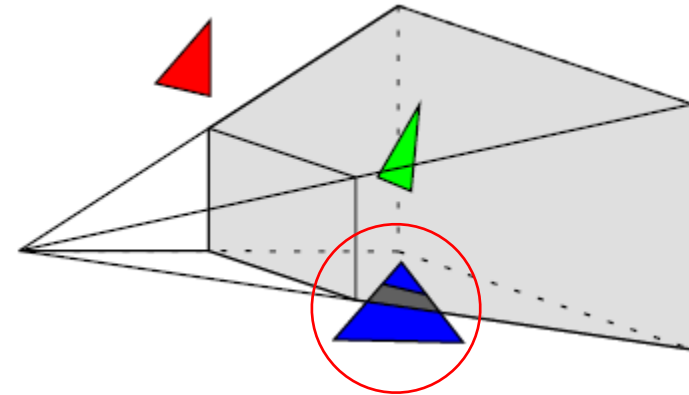
Clipping

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



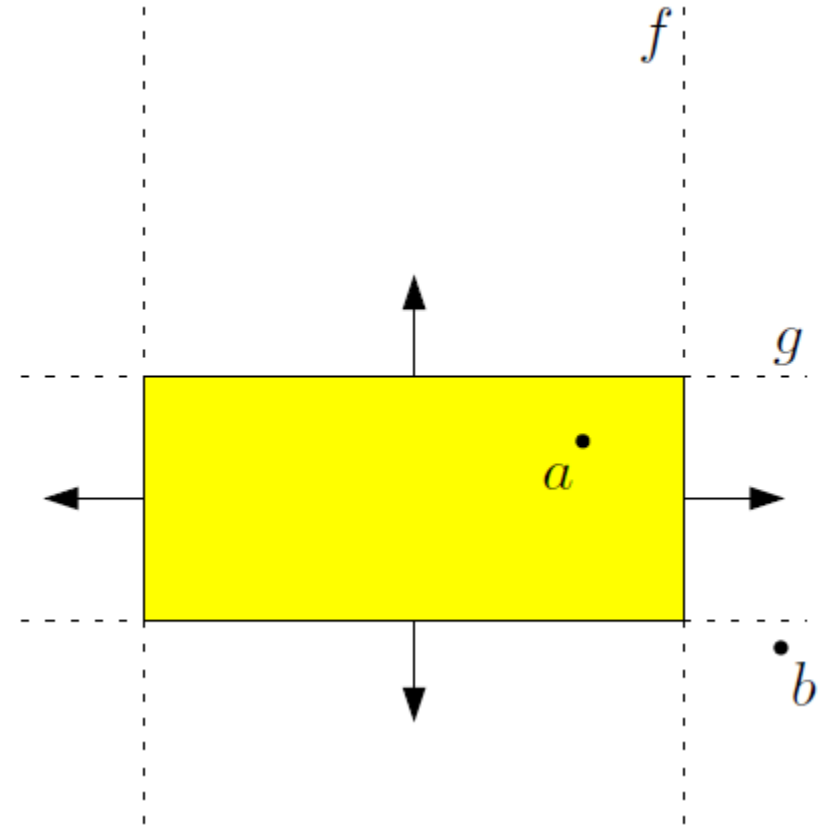
Clipping

- 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:
- $f(p) = n \cdot (p - q) = 0$



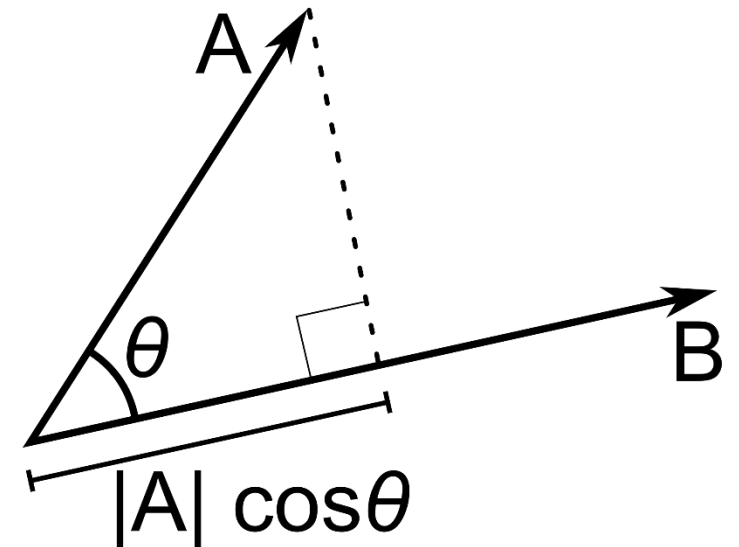
3D - iloczyn skalarny $a \cdot b$

Definicja 1: $a \cdot b = a_1b_1 + a_2b_2 + a_3b_3$

3D - iloczyn skalarny $a \cdot b$

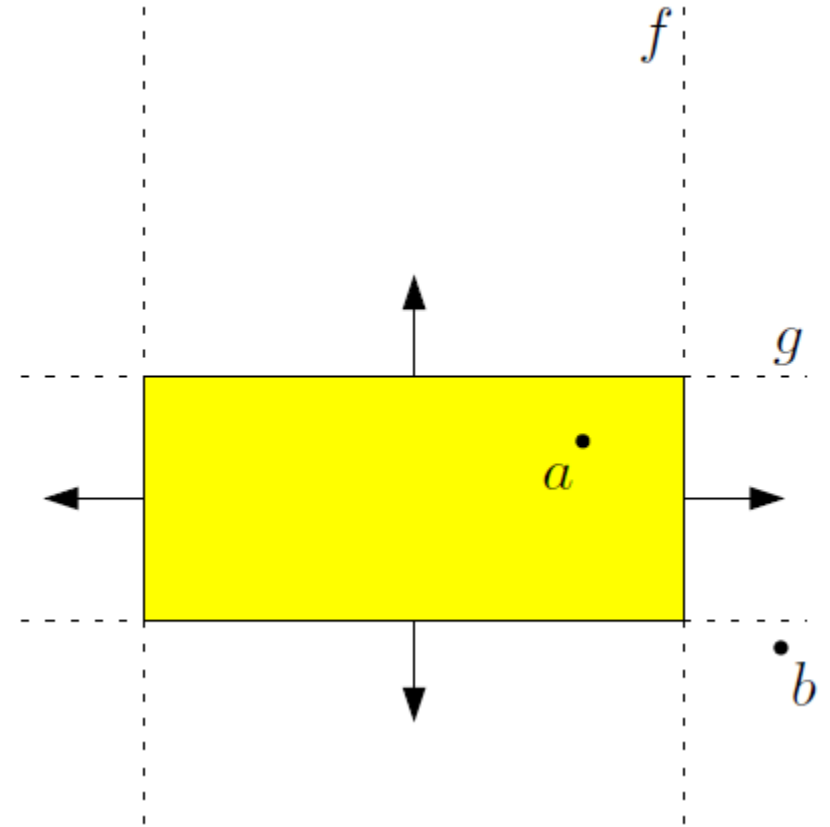
Definicja 1: $a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3$

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



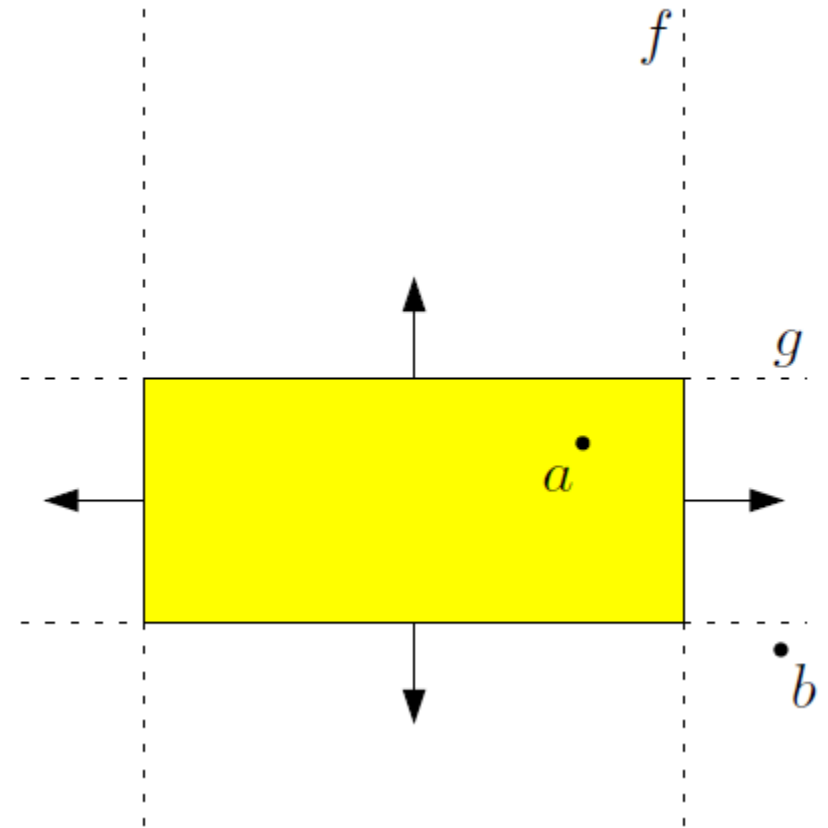
Sprawdzanie przecinania

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



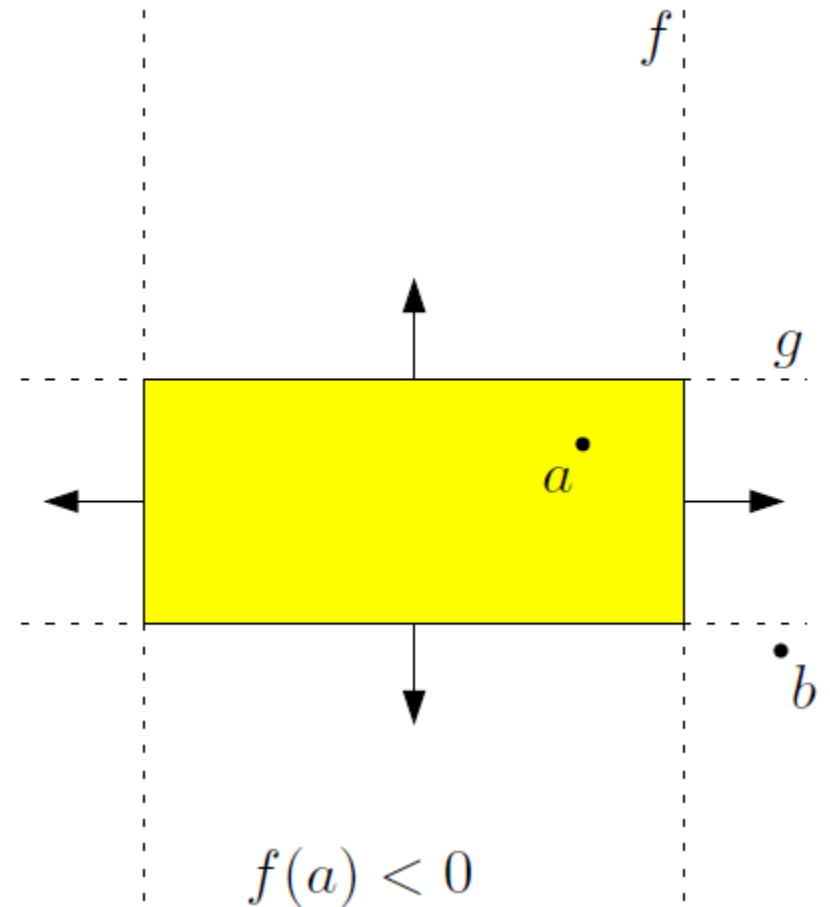
Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$
- Konwencja: normalne naszych hiperpłaszczyzn są skierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



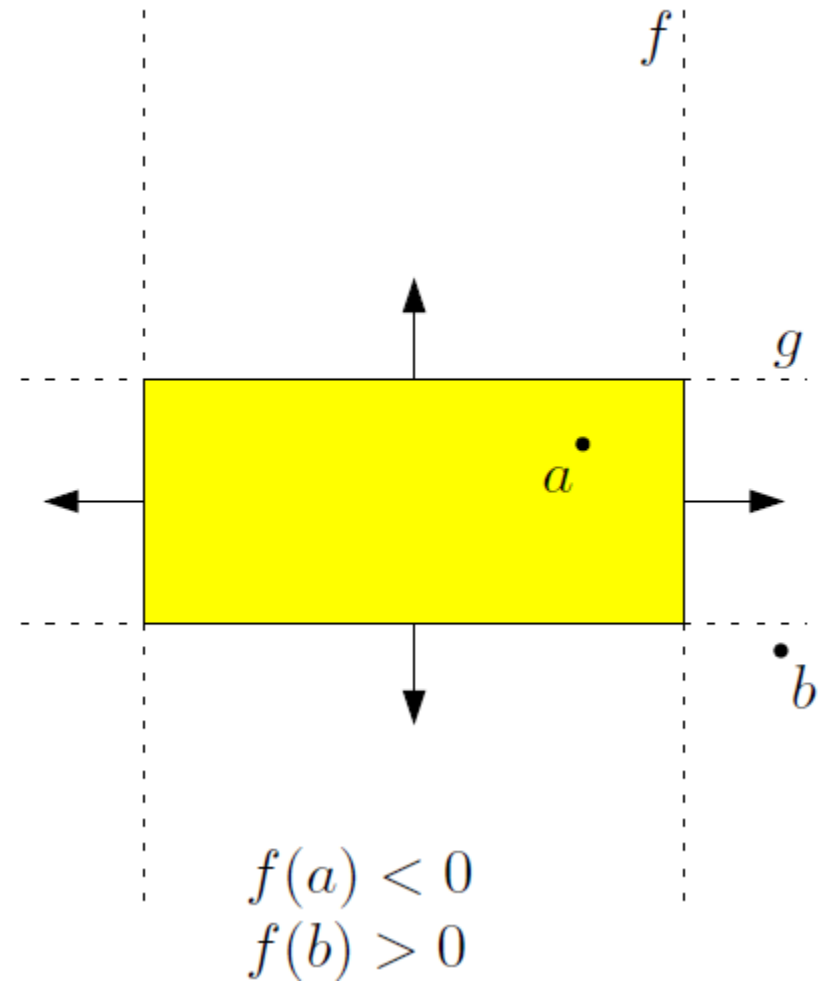
Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$
- Konwencja: normalne naszych hiperpłaszczyzn są skierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



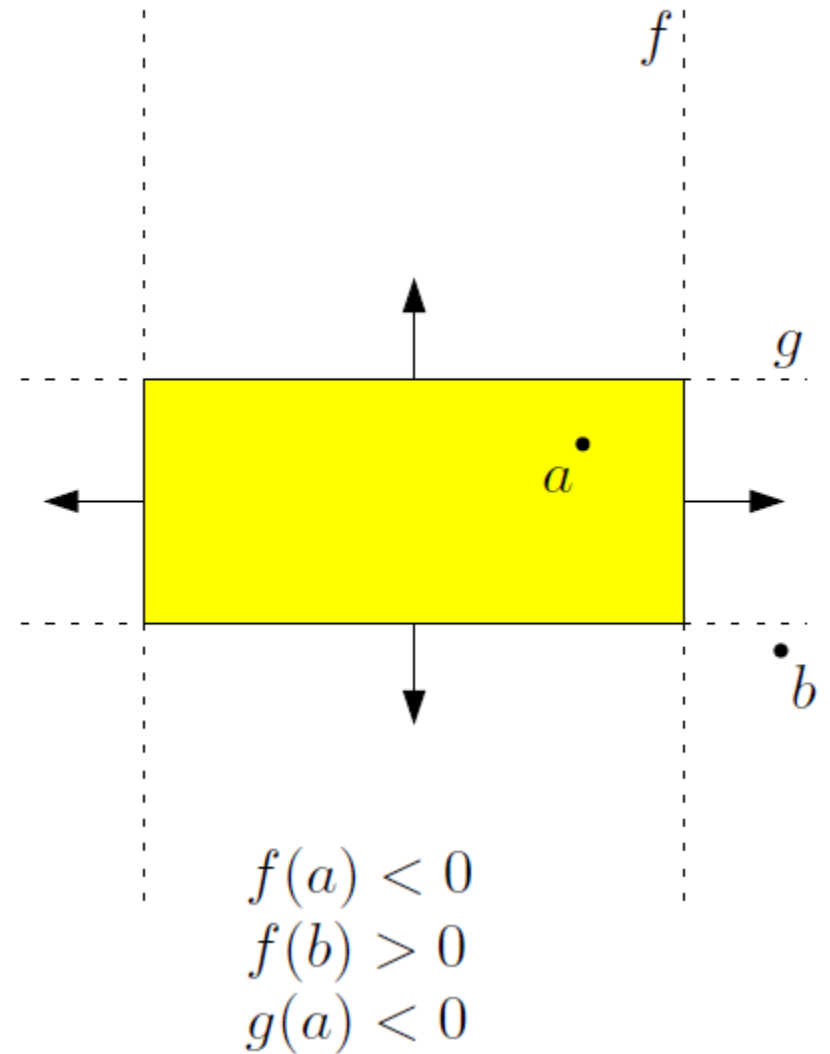
Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$
- Konwencja: normalne naszych hiperpłaszczyzn są skierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



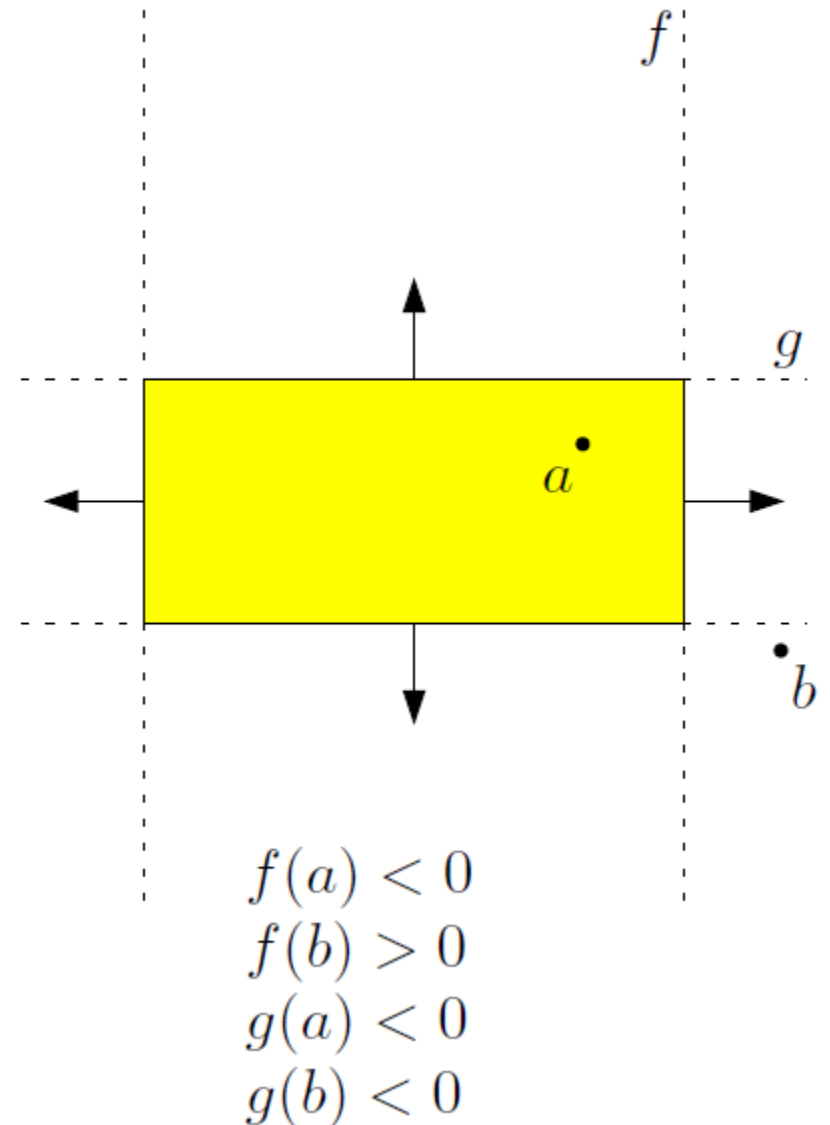
Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$
- Konwencja: normalne naszych hiperpłaszczyzn są skierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



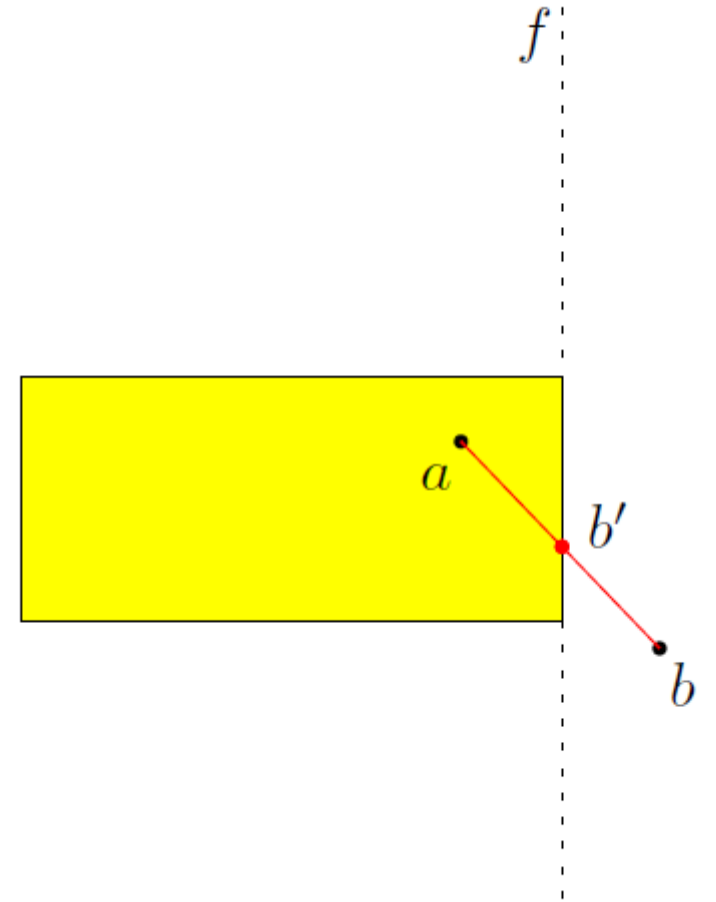
Sprawdzanie przecinania

- Równanie dla hiperpłaszczyzny przez punkt q i normalną n jest dane przez:
- $f(p) = n \cdot (p - q) = 0$
- Konwencja: normalne naszych hiperpłaszczyzn są skierowane na **zewnątrz** (bryły widzenia), czyli gdy $f(p) < 0$ to p jest **wewnątrz**, a jak $f(p) > 0$ to p jest **zewnątrz** płaszczyzny.



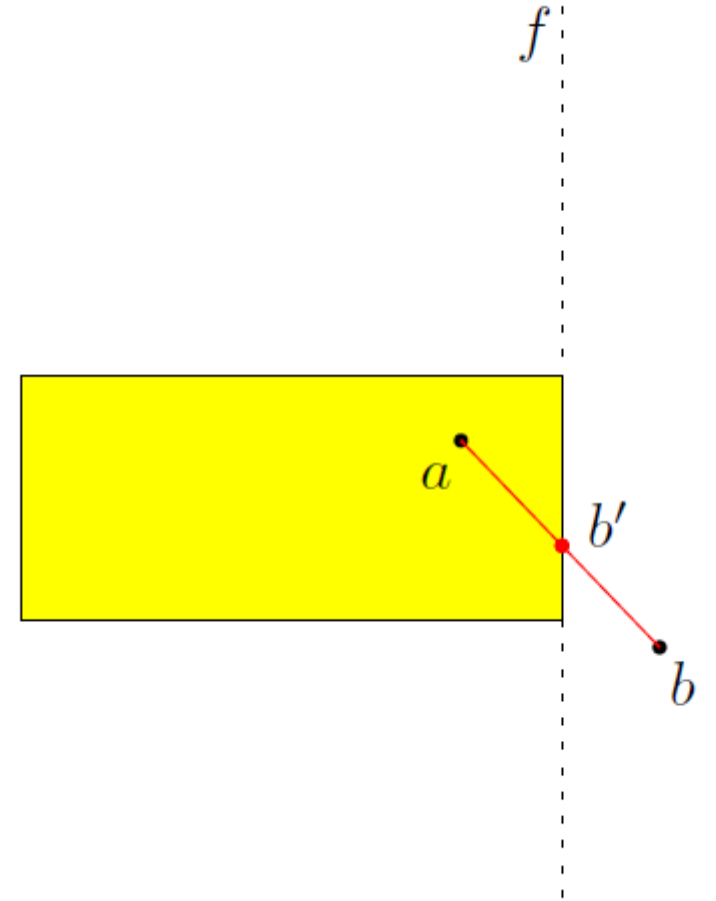
Obliczenie punktów przecięcia

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



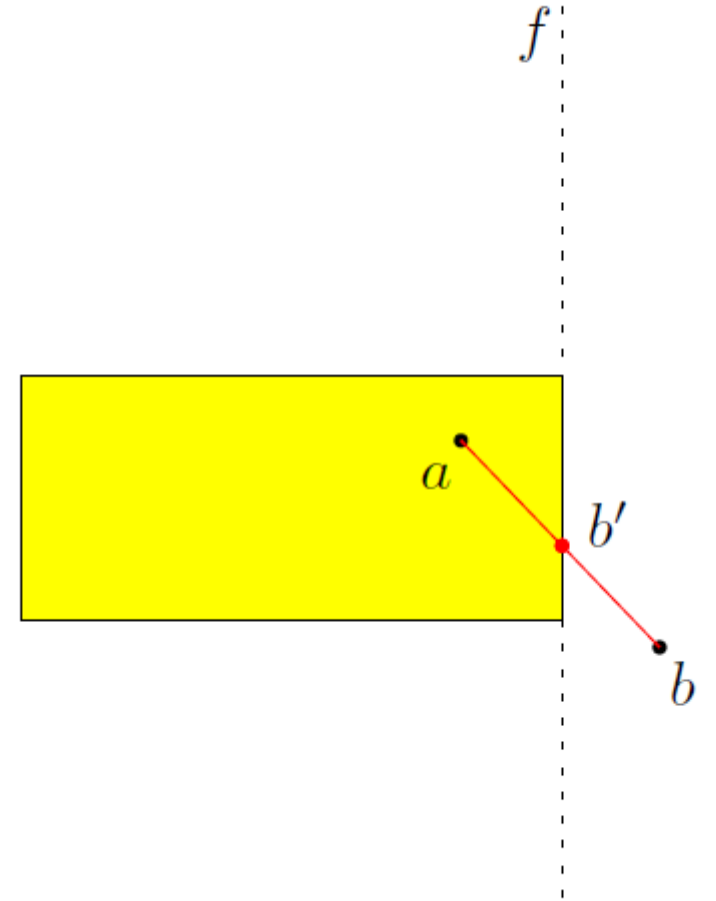
Obliczenie punktów przecięcia

- 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:

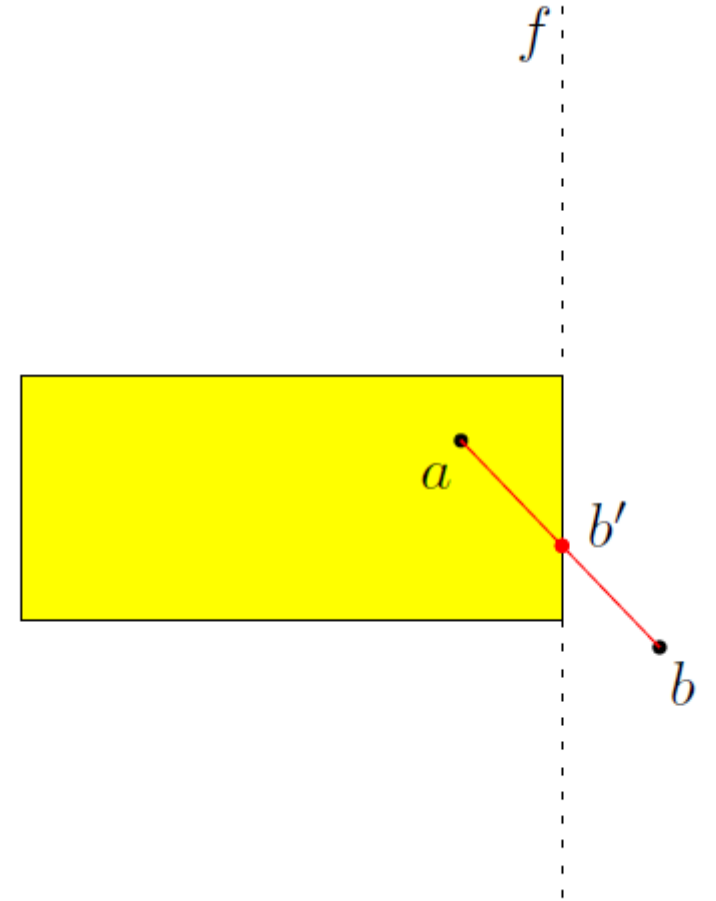
- $\vec{p}(t) = \vec{a} + t(\vec{b} - \vec{a})$

- Podkładając te równanie pod nasze równanie hiperpłaszczyzny daje nam:

- $\vec{n} \cdot (\vec{p} - \vec{q}) = 0$

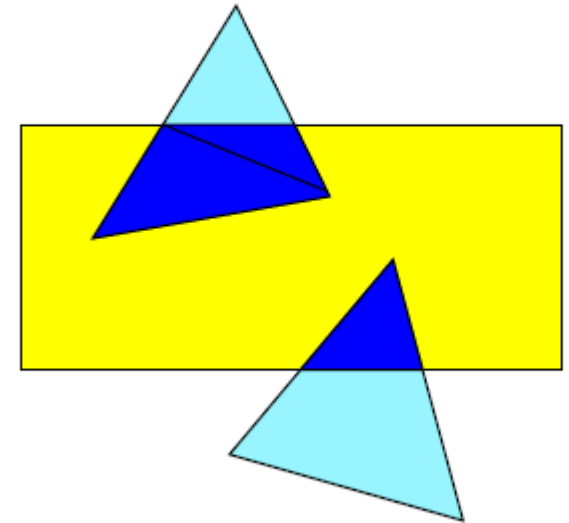
- $\vec{n} \cdot \left(\vec{a} + t(\vec{b} - \vec{a}) - \vec{q} \right) = 0$

- $t = \frac{\vec{n} \cdot \vec{a} - \vec{n} \cdot \vec{q}}{\vec{n} \cdot (\vec{a} - \vec{b})}$



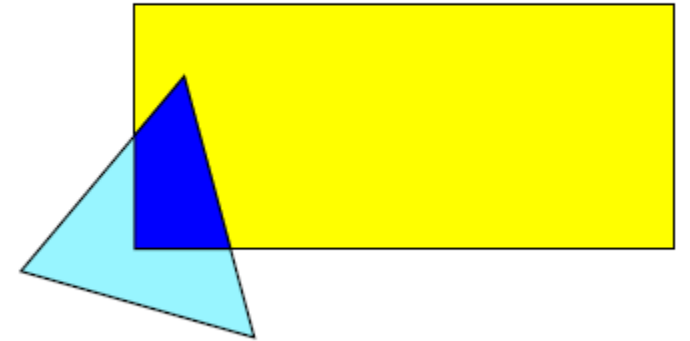
Stwarzanie nowych trójkątów

- 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 **dwie nowe trójkąty**



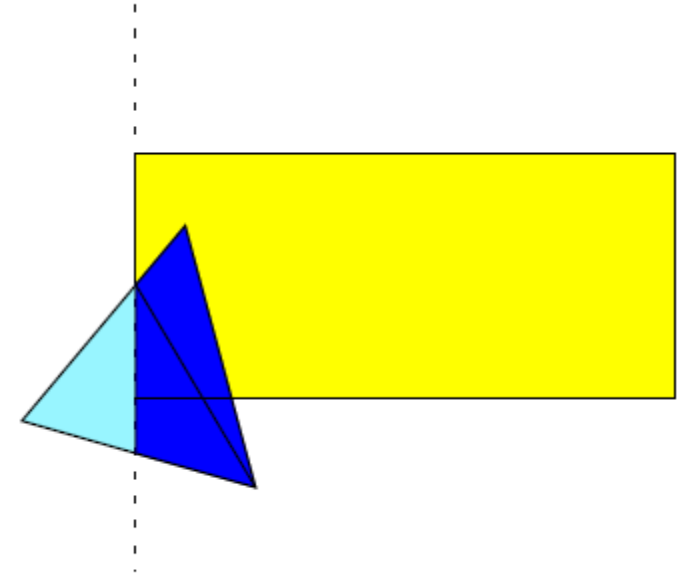
Stwarzanie nowych trójkątów

- Ale co jak trójkąt jest przecinany przez dwie płaszczyzny?



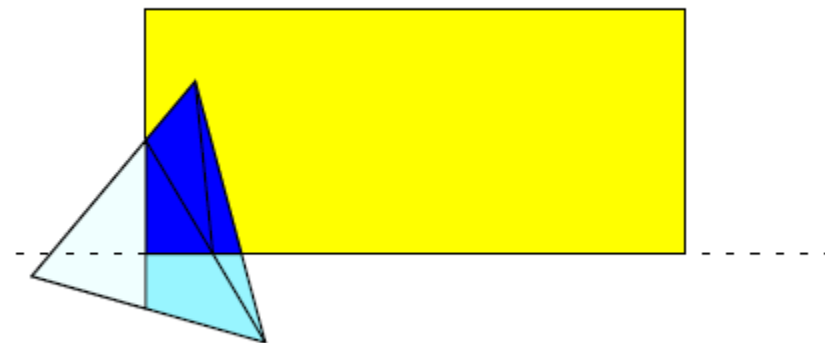
Stwarzanie nowych trójkątów

- Przycinamy względem jednej



Stwarzanie nowych trójkątów

- Przycinamy względem drugiej



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

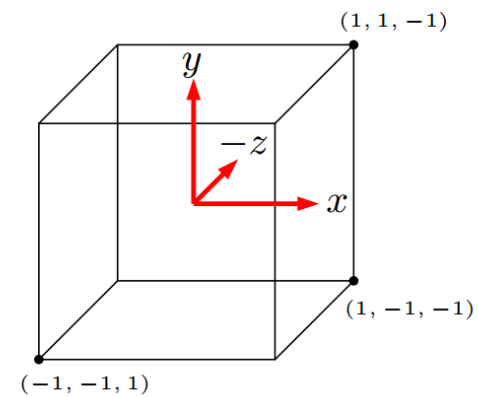
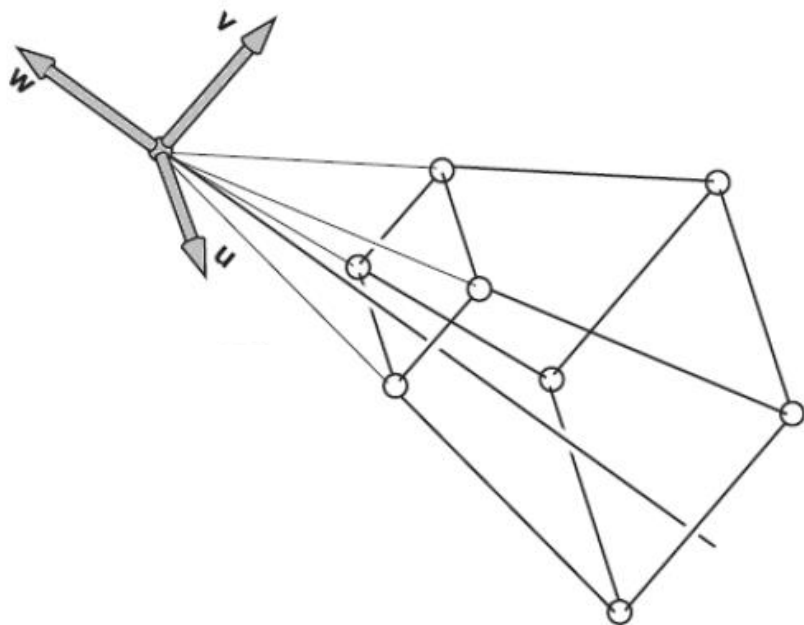
Macierz
transformacji

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

Homogenizacja

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

Rasteryzacja



Clipping po homogenizacji

- Proste równania dla hiperpłaszczyzn:

$$-x + l = 0$$

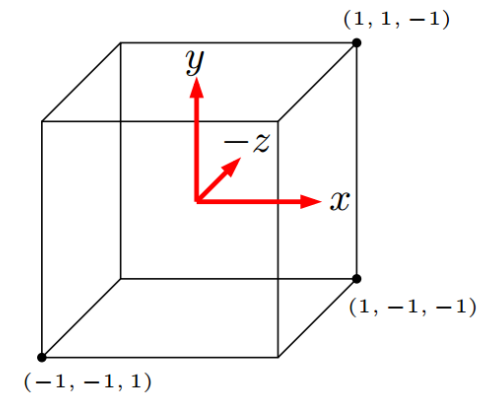
$$x - r = 0$$

$$-y + b = 0$$

$$y - t = 0$$

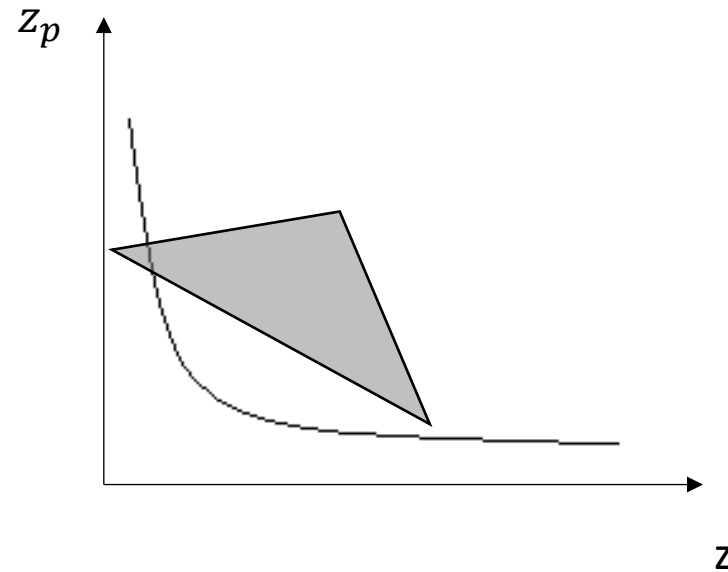
$$-z + n = 0$$

$$z - f = 0$$



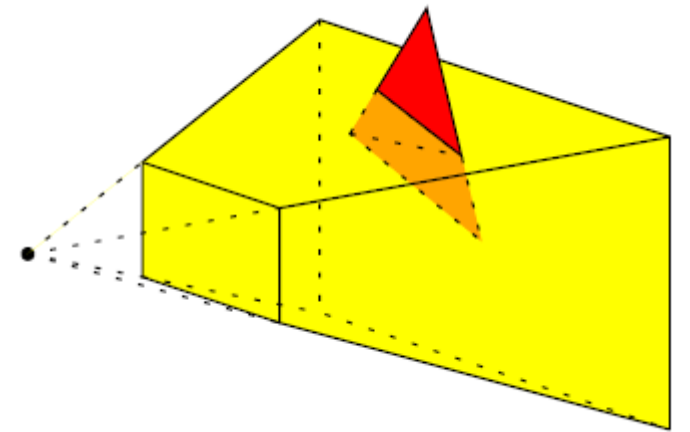
Problem z płaszczyzną XY

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



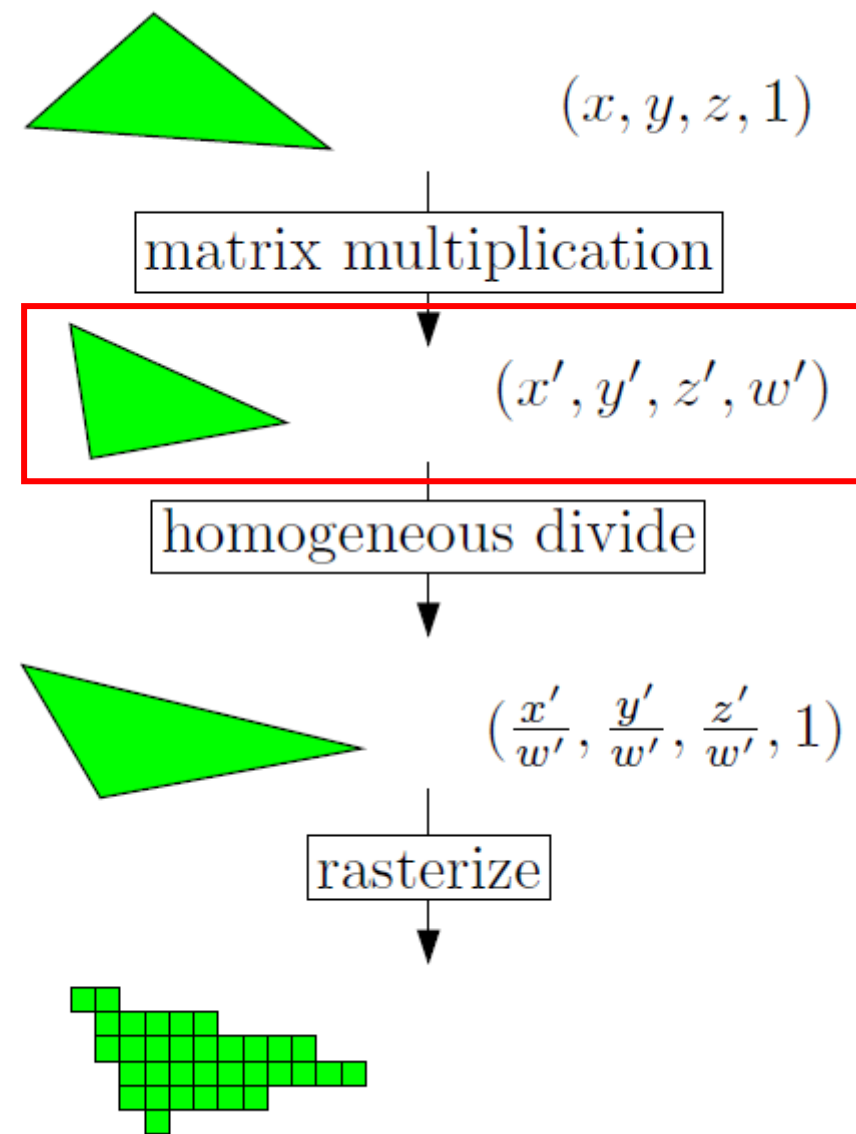
Clipping przed homogenizacją

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



Clipping w systemie współrzędnych jednorodnych

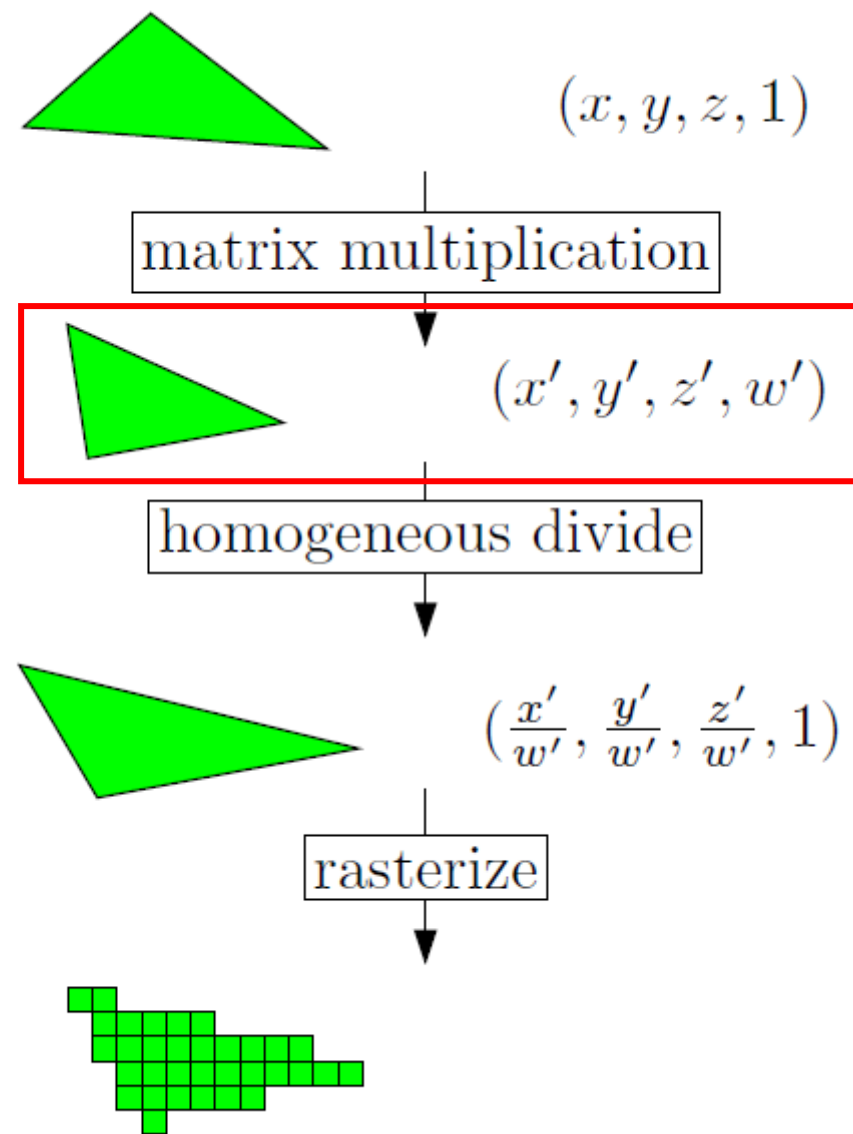
- Jednak najłatwiej da się wykonać **clipping** w systemie **współrzędnych jednorodnych**. To znaczy że ucinamy trójkąty w **czterech wymiarach** według **trójwymiarowych hiperpłaszczyzn**.



Clipping w systemie współrzędnych jednorodnych

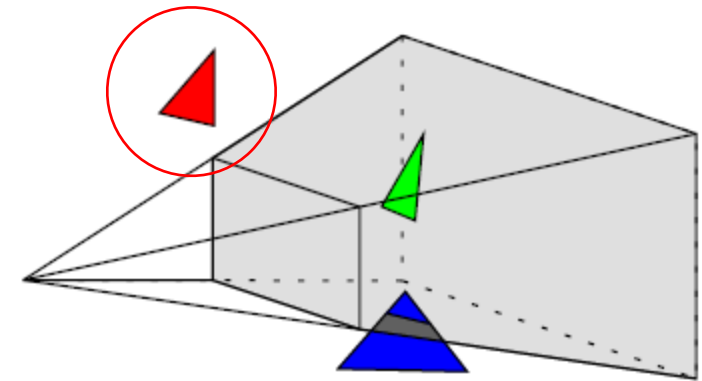
- Jednak najłatwiej da się wykonać **clipping** w systemie **współrzędnych jednorodnych**. To znaczy że ucinamy trójkąty w **czterech wymiarach** według **trójwymiarowych hiperpłaszczyzn**.

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



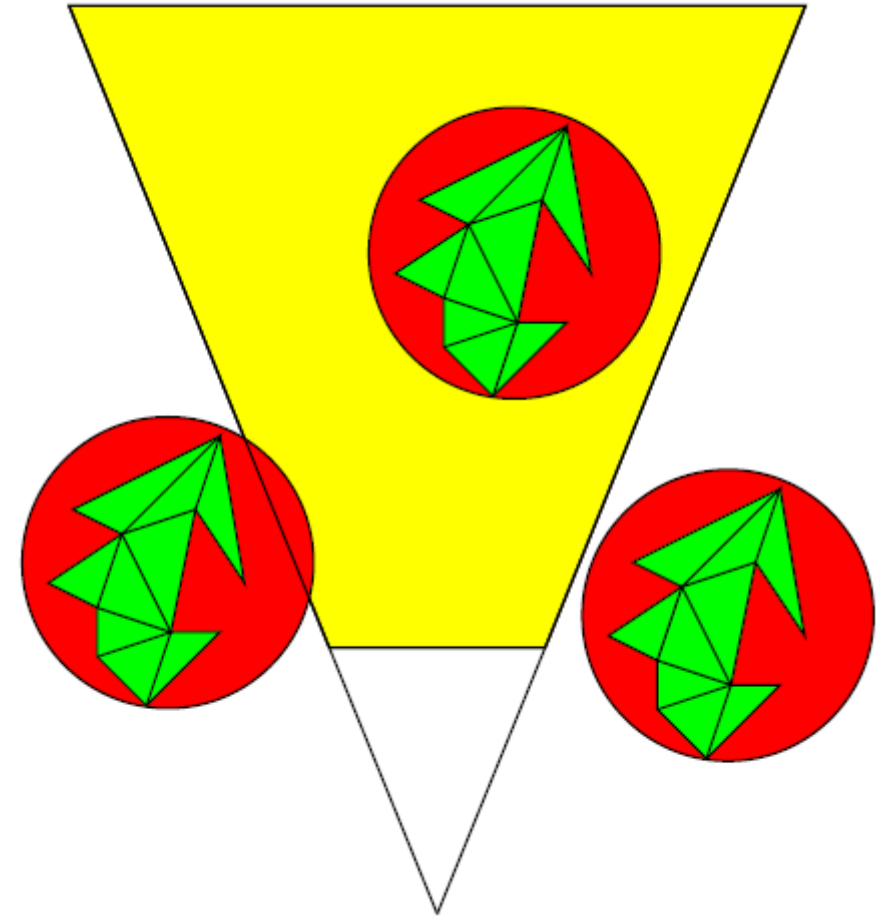
Culling

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



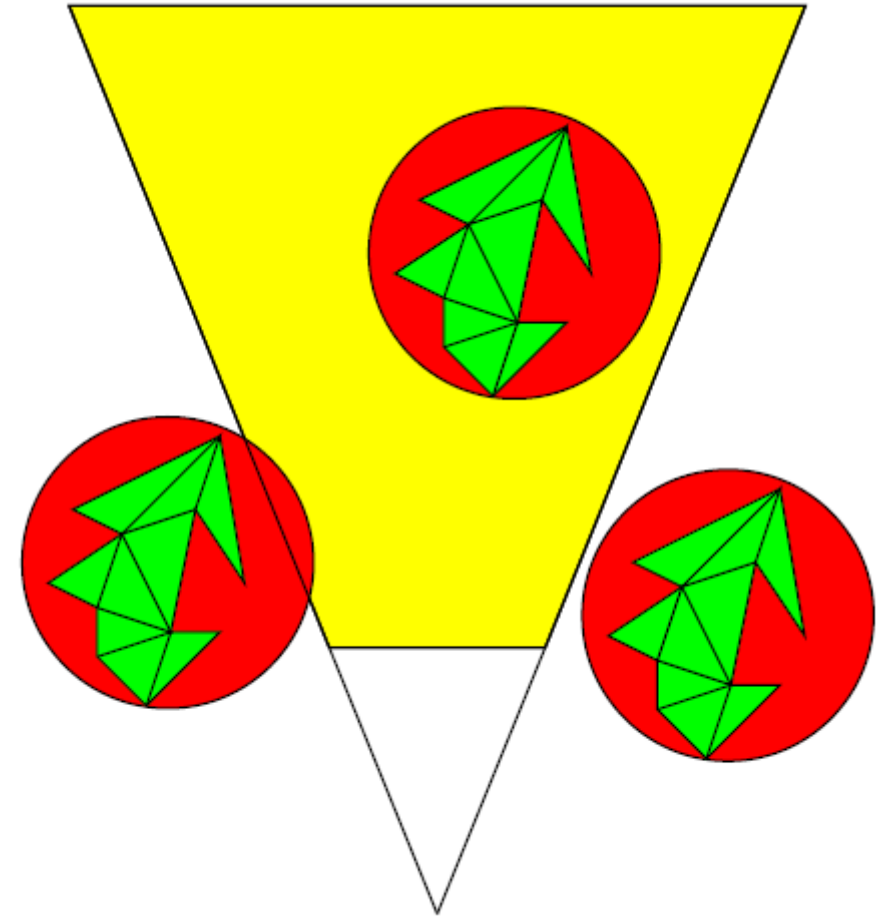
Bryły brzegowe (bounding volumes)

- Używanie brył brzegowych dla skomplikowanych obiektów geometrycznych przyspiesza potok graficzny



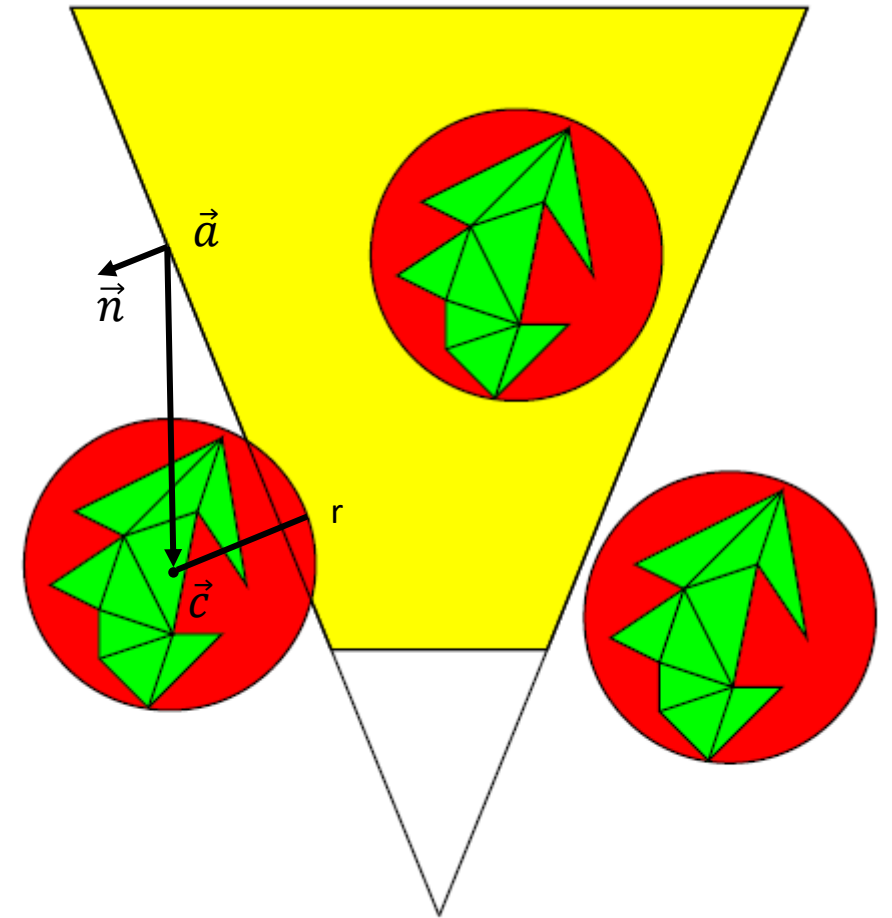
Bryły brzegowe (bounding volumes)

- 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
- $\frac{(\vec{c} - \vec{a}) \cdot \vec{n}}{\|\vec{n}\|} > r$



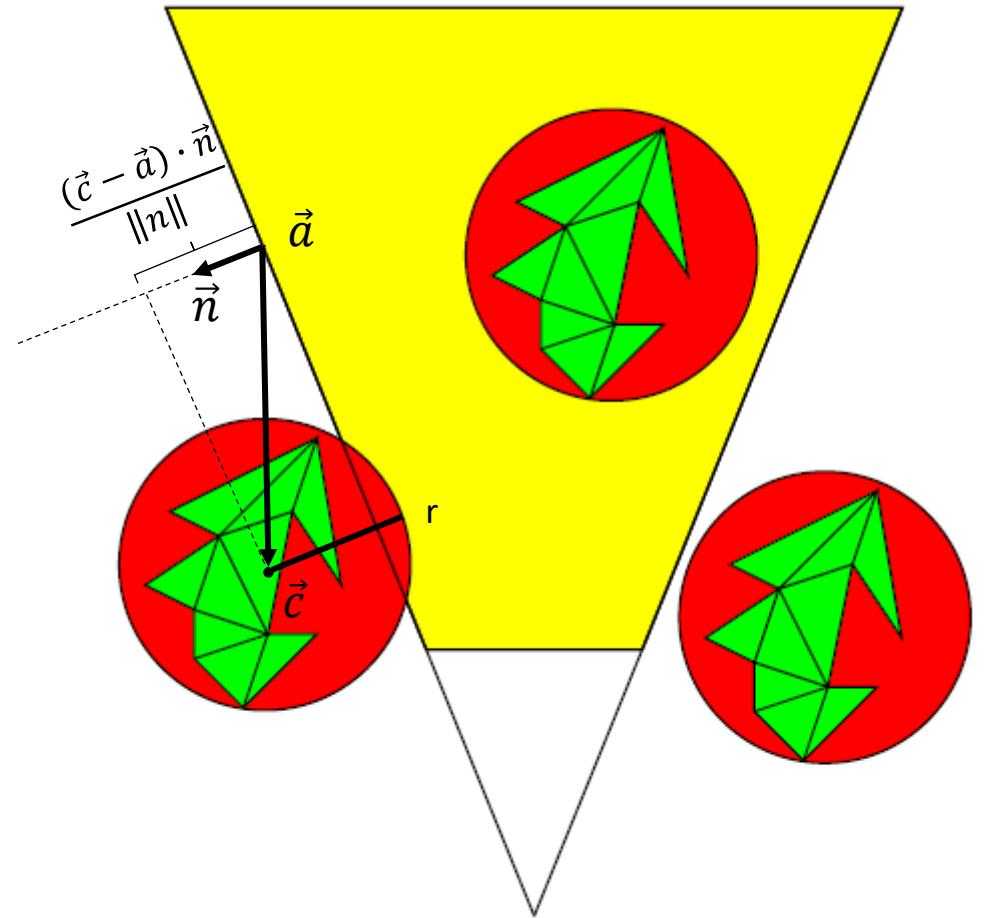
Bryły brzegowe (bounding volumes)

- 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
- $\frac{(\vec{c} - \vec{a}) \cdot \vec{n}}{\|\vec{n}\|} > r$



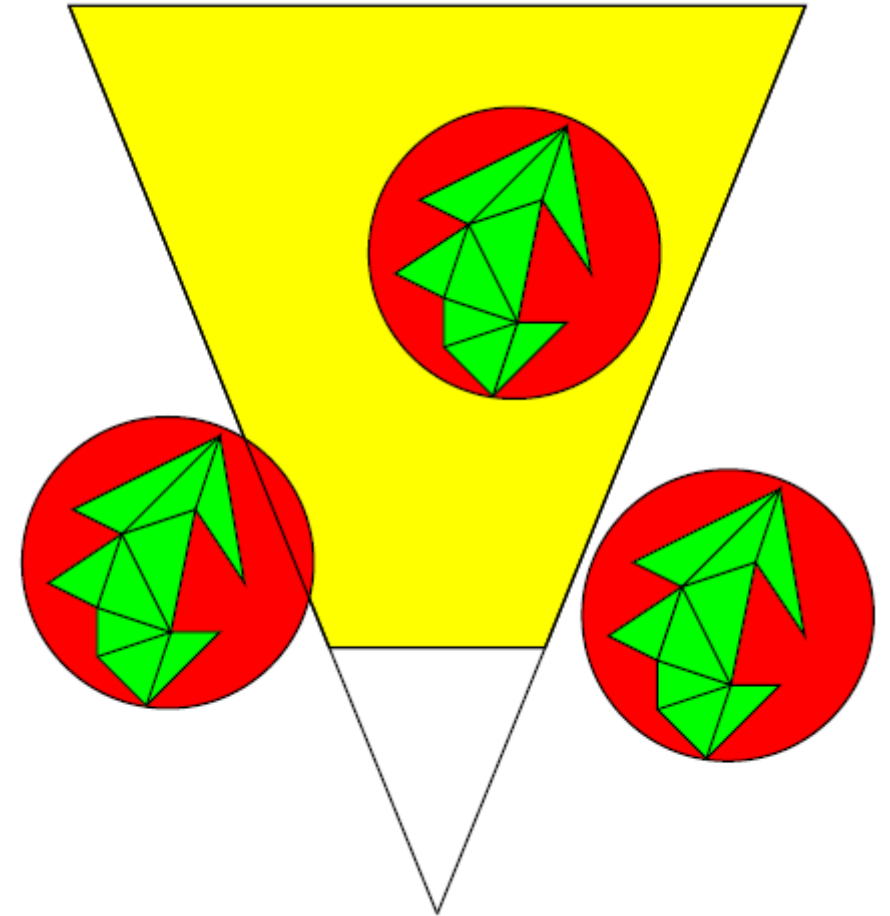
Bryły brzegowe (bounding volumes)

- 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
- $\frac{(\vec{c} - \vec{a}) \cdot \vec{n}}{\|\vec{n}\|} > r$



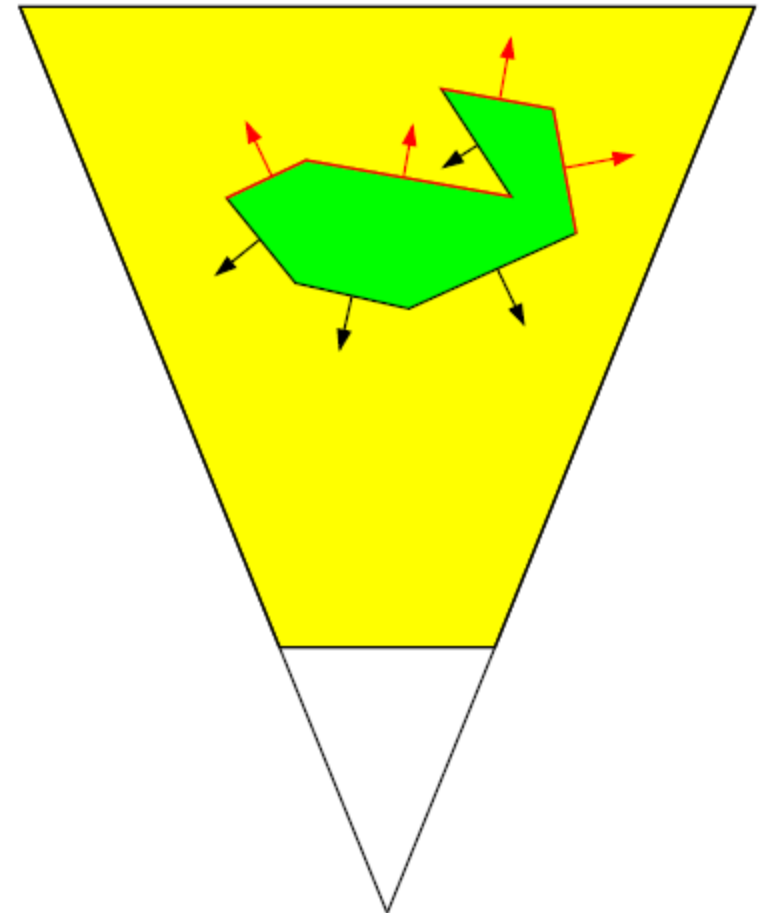
Culling

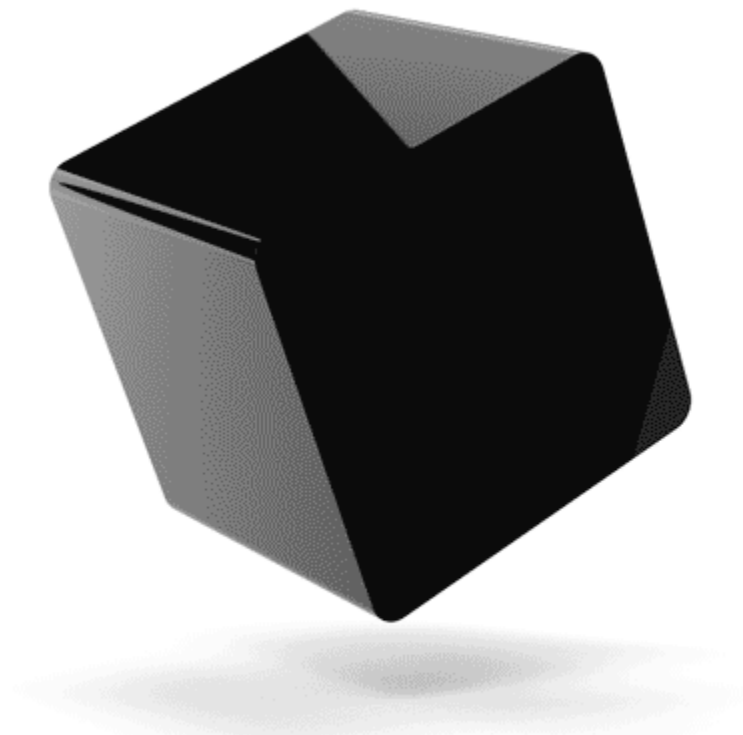
- **Frustum culling**
usuwanie trójkątów poza bryły widzenia
- **Backface culling**
usuwanie trójkątów orientowane od pozycji kamery



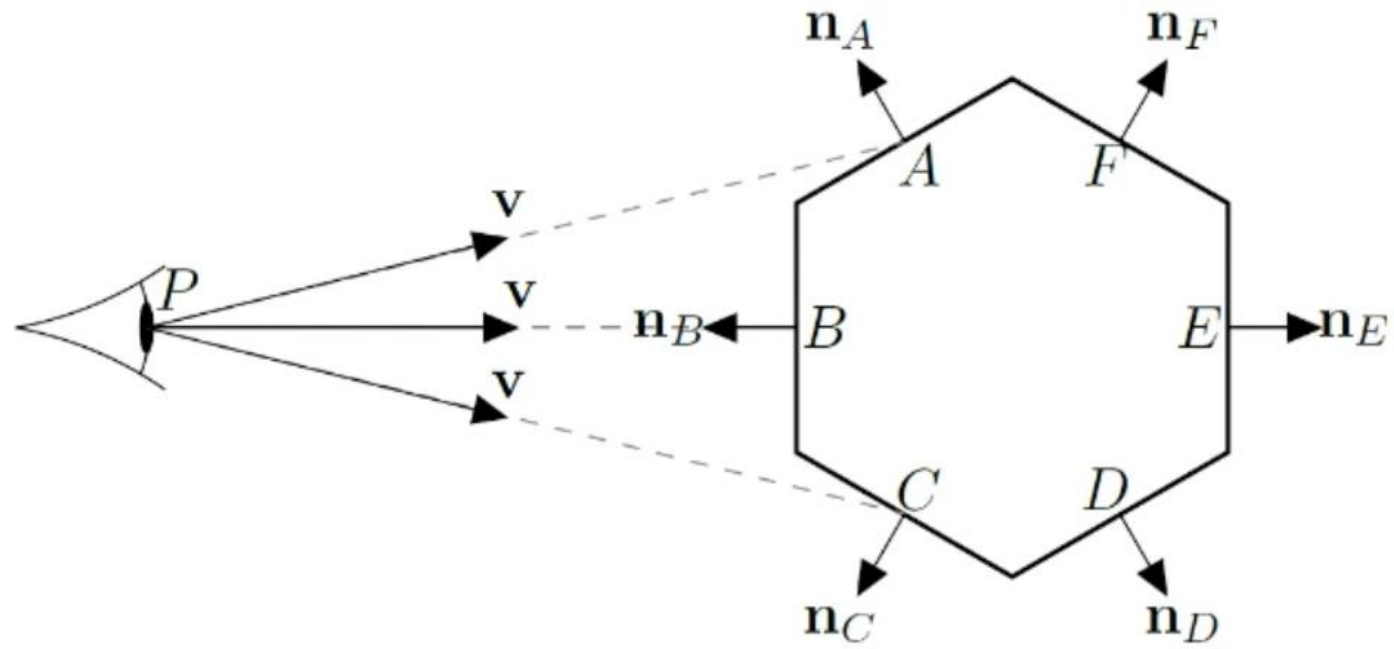
Backface culling

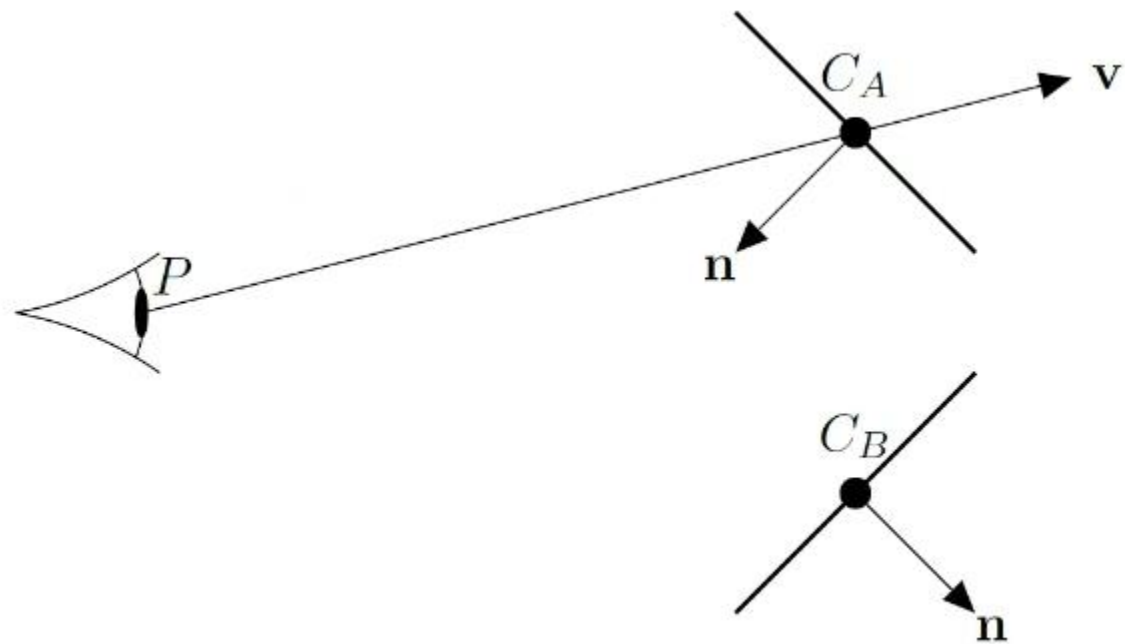
- Gdy modelujemy obiekty geometryczne za pomocą trójkątów to **normalna trójkątów** jest ustawiona na **zewnątrz** obiektu
- Odrzucanie trójkątów których normalna jest skierowana **od punktu widzenia** nazywamy to **backface culling** (usuwanie powierzchni tylnych)

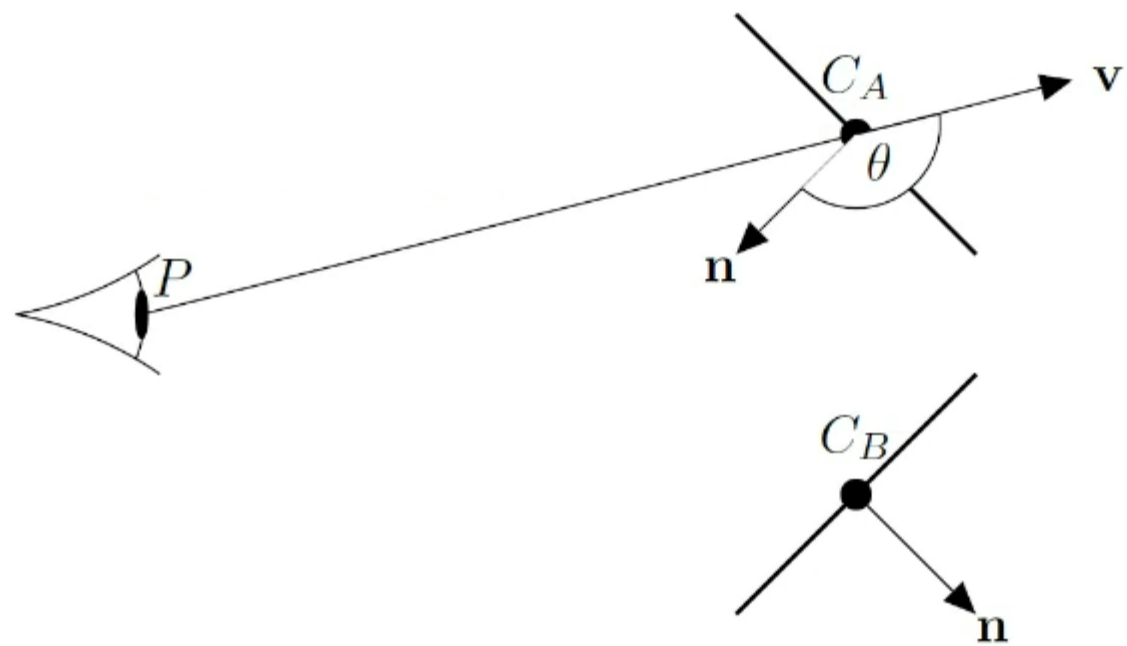


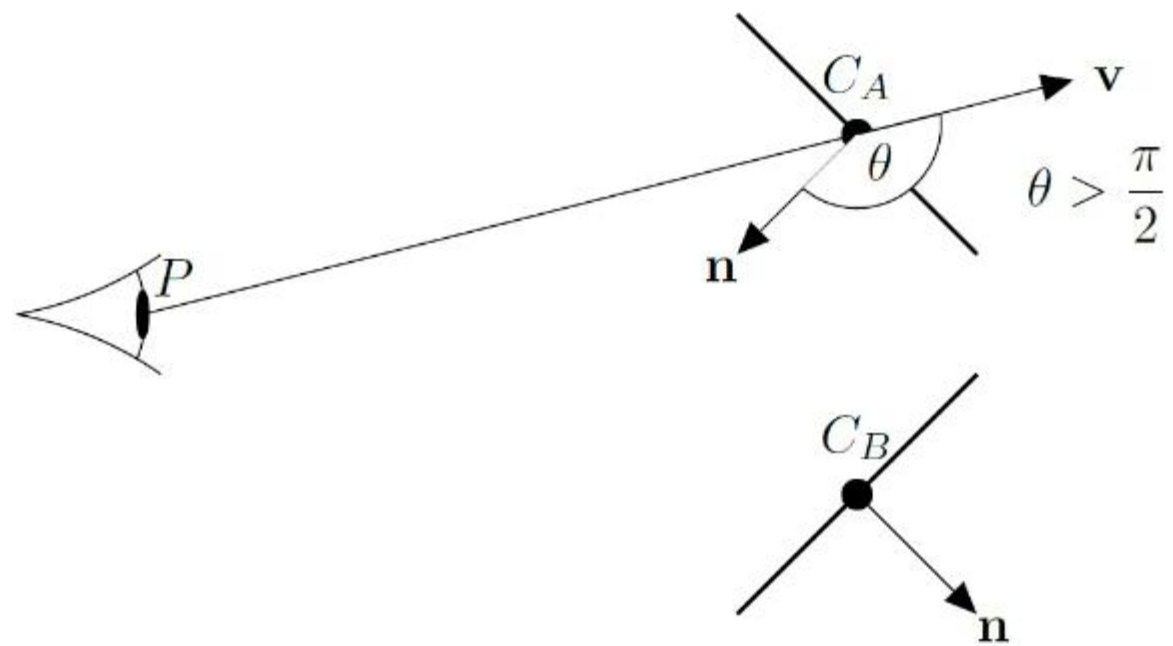


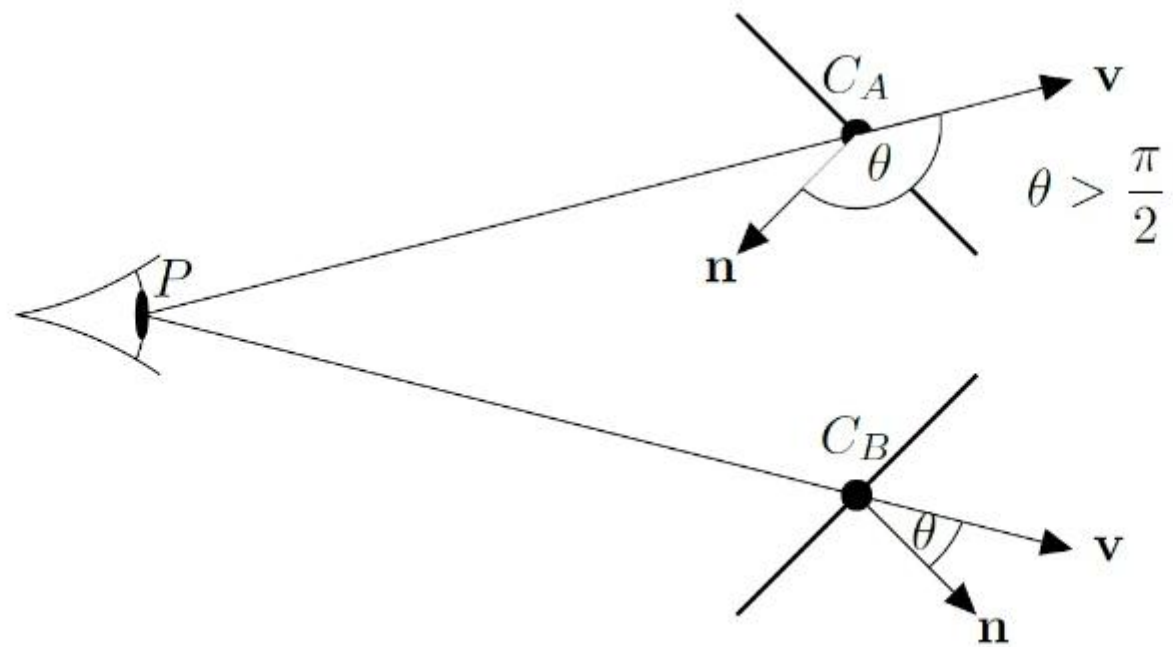
- Gdy kamera jest skierowana do tyłu płaszczyzn nie musimy je rysować

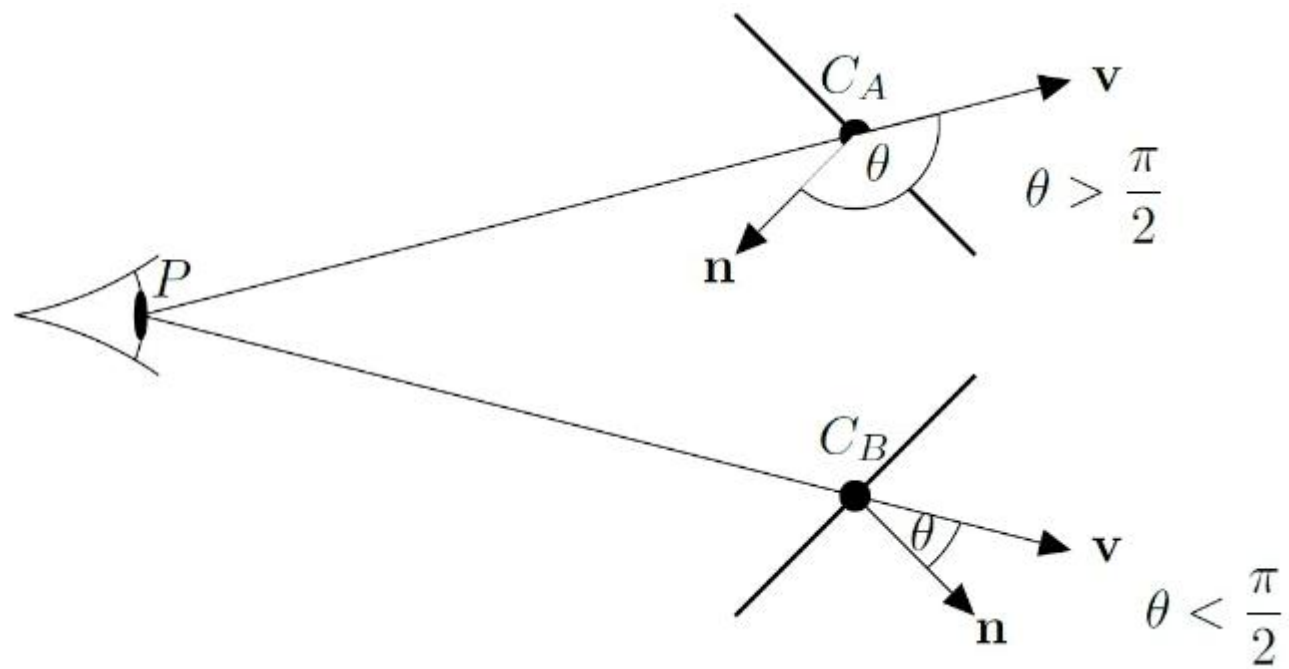


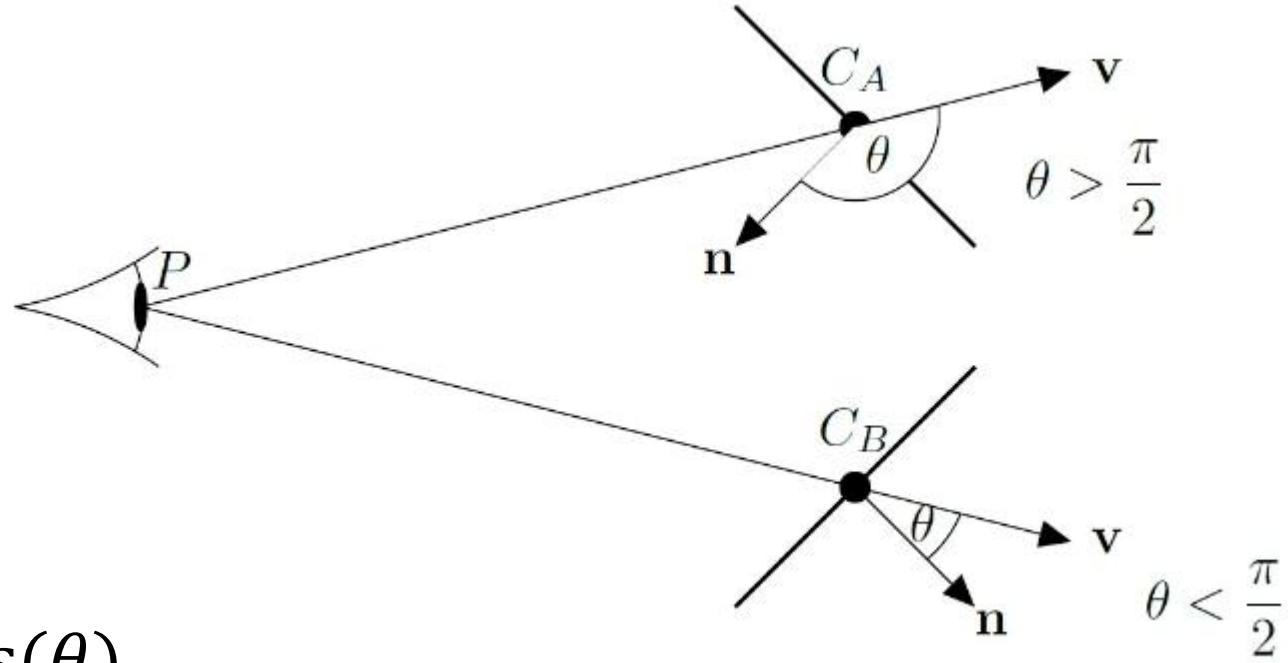




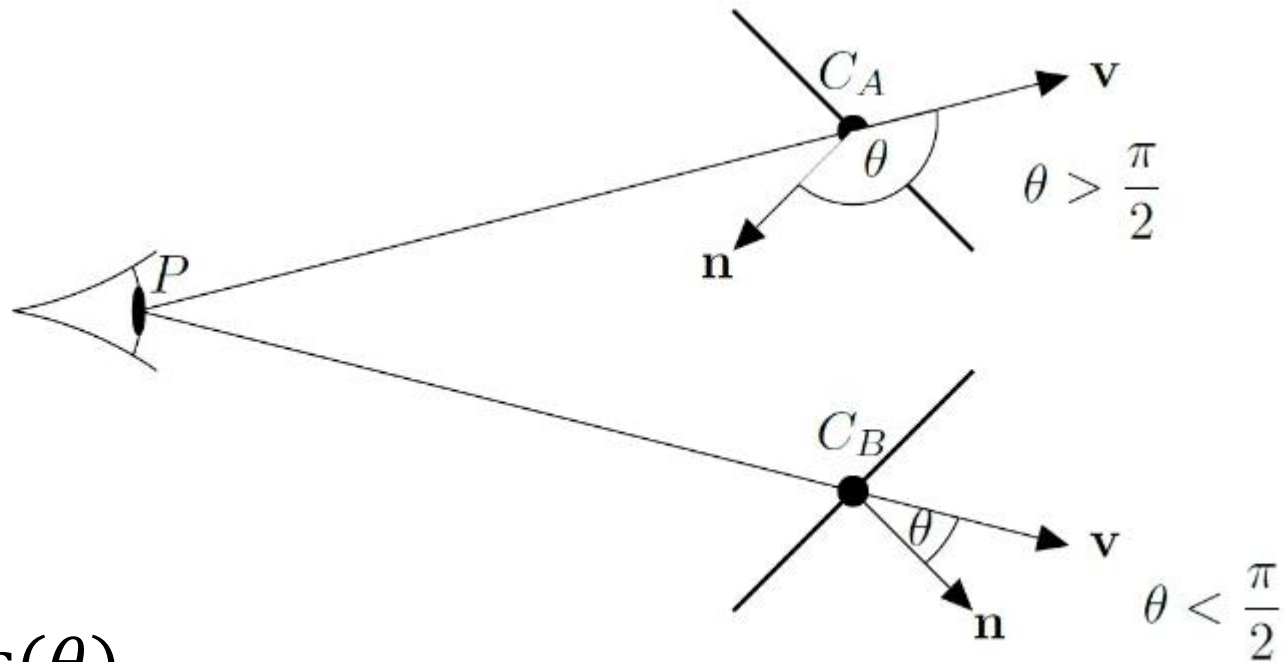








- $\mathbf{v} \cdot \mathbf{n} = |\mathbf{v}||\mathbf{n}|\cos(\theta)$



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

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

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

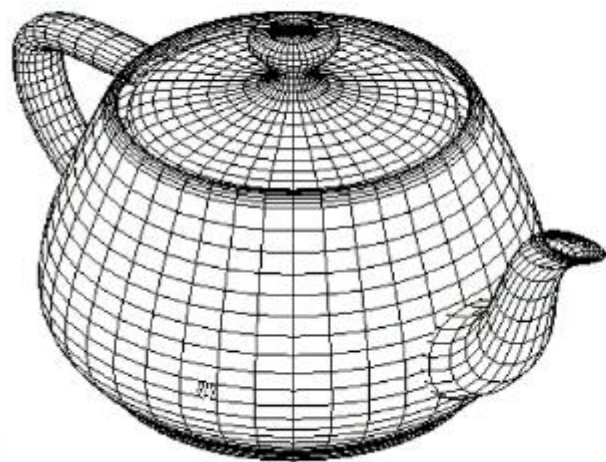
end for



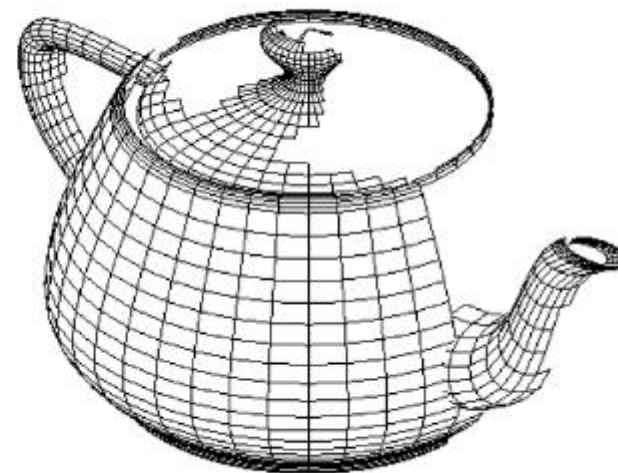
wszystkie wielokąty



backface culling

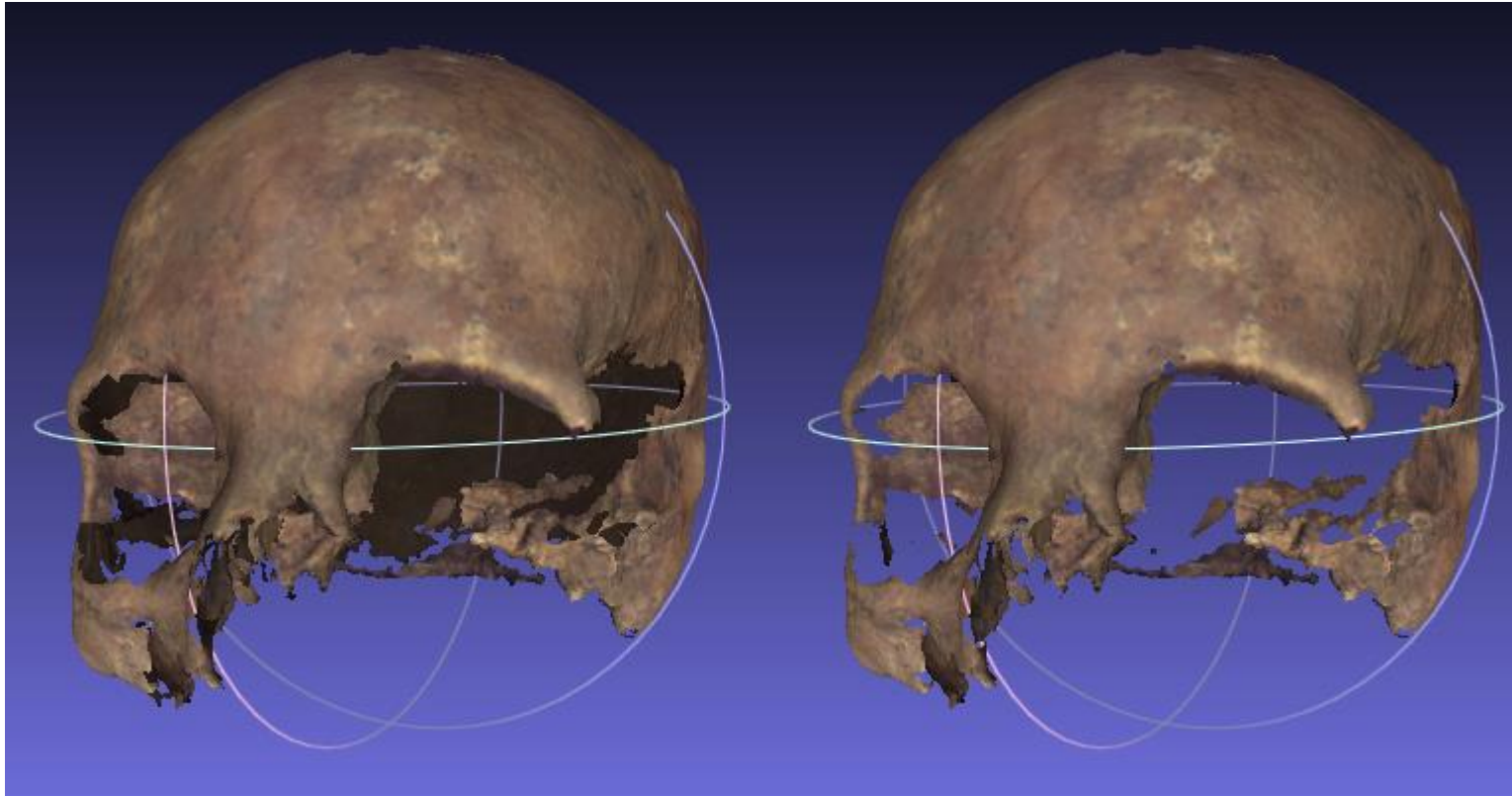


wszystkie wielokąty



backface culling

Przykład

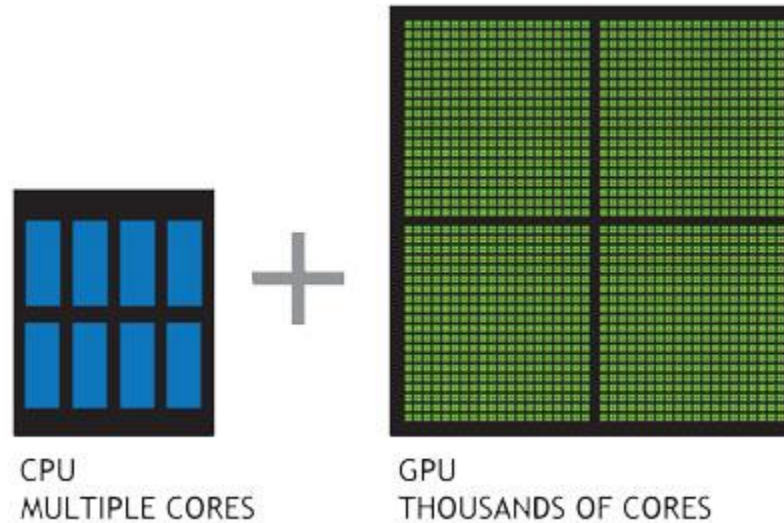


Bez backface culling

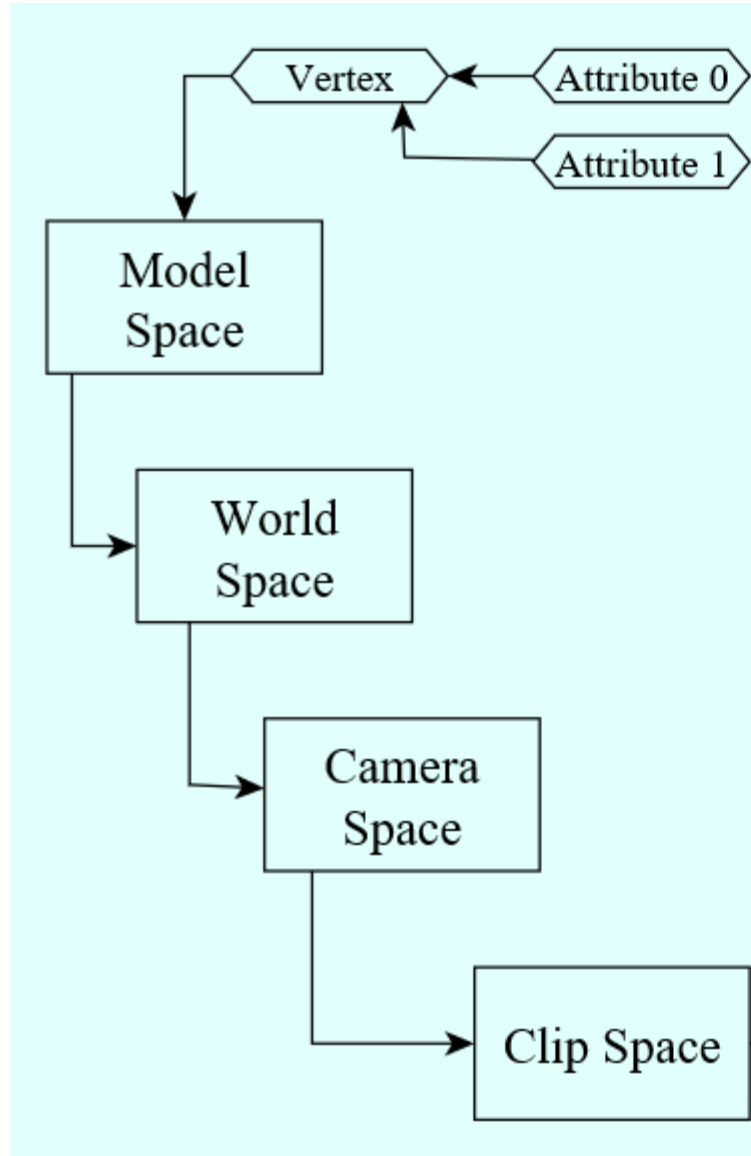
Backface culling

Shader

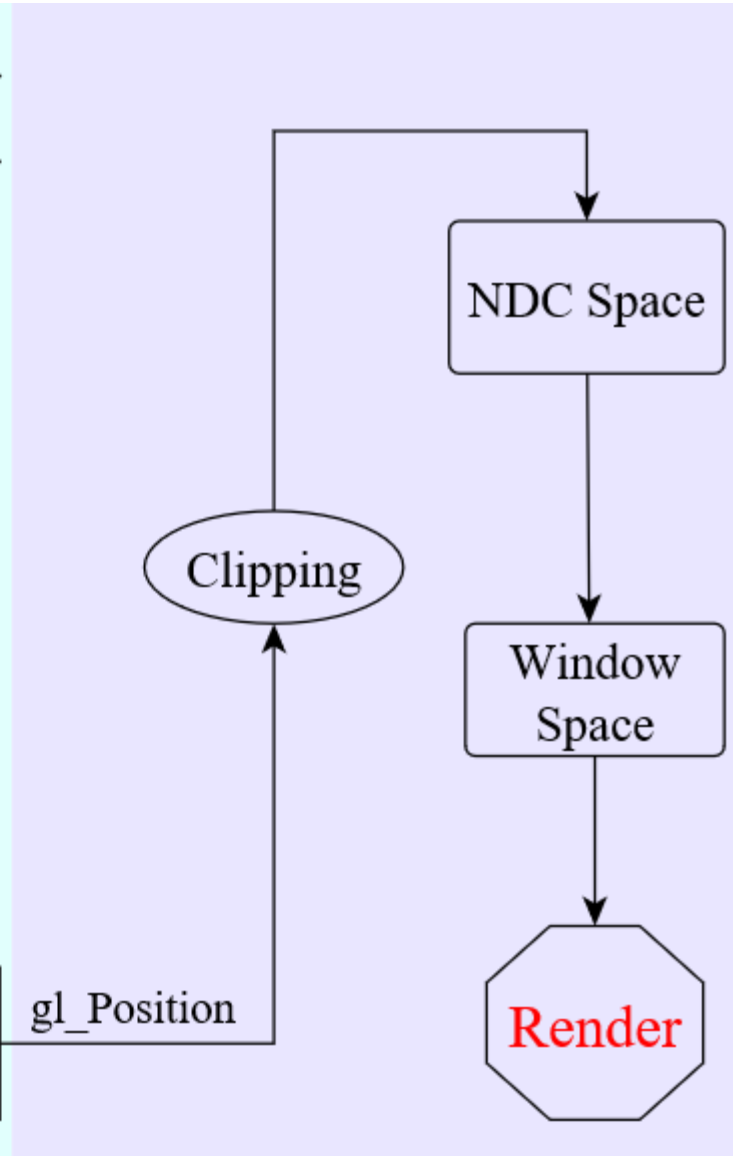
- Zestaw **instrukcji** uruchomiony na poziomie **karty graficznej**
- Nazwa bierze się od shading (cieniować)
- **Cg**, **HLSL** i **GLSL**



CPU



GPU



```
graph TD; A[Vertex] --> B[Model Space]; B --> C[World Space]; C --> D[Camera Space]; D --> E[Clip Space]; F[Attribute 0] --> A; G[Attribute 1] --> A;
```

The diagram illustrates the 3D pipeline. It starts with a 'Vertex' (hexagon) which is transformed into 'Model Space' (rectangle). 'Model Space' is then transformed into 'World Space' (rectangle), which is transformed into 'Camera Space' (rectangle), which is finally transformed into 'Clip Space' (rectangle). Additionally, 'Attribute 0' and 'Attribute 1' (hexagons) are shown as inputs to the 'Vertex'.

```
graph TD; gl_Position[gl_Position] --> Clipping([Clipping]); Clipping --> NDC_Space[NDC Space]; Clipping --> Window_Space[Window Space]; NDC_Space --> Render{{Render}}; Window_Space --> Render; Vertex_shader[Vertex shader] --> NDC_Space; Fragment_shader[Fragment shader] --> Window_Space;
```

Shadery jako pliki na twardym dysku

```
void init()  
{  
    glEnable(GL_DEPTH_TEST);  
    program = shaderLoader.CreateProgram("shaders/shader_2_1.vert", "shaders/shader_2_1.frag");  
}
```

Włączanie shaderów

```
glUseProgram(program);
```

```
glm::mat4 camera = createCameraMatrix();
```

```
glm::mat4 perspective = createPerspectiveMatrix();
```

```
// Oblicz odpowiednia transformacje.
```

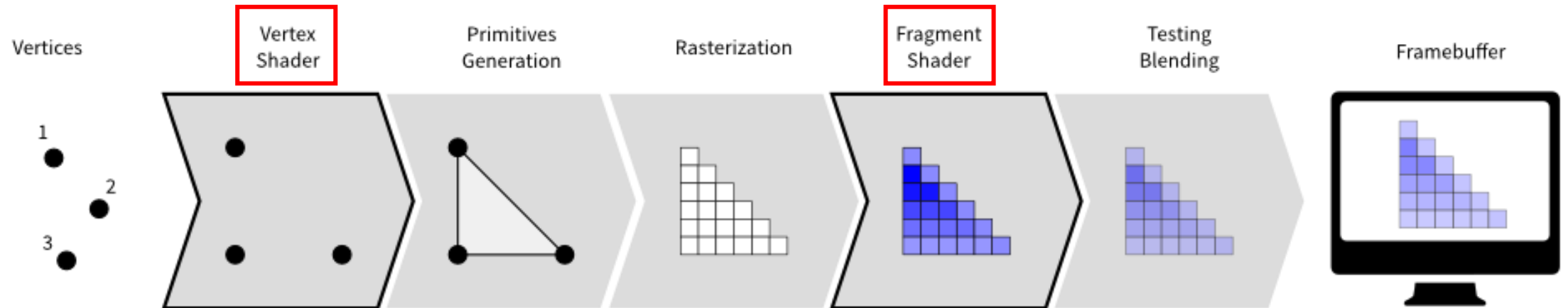
```
glm::mat4 transformation;
```

```
glUniformMatrix4fv(glGetUniformLocation(program, "transformation"), 1
```

```
// Narysuj pudełko w początku układu współrzędnych.
```

```
glUseProgram(0);
```

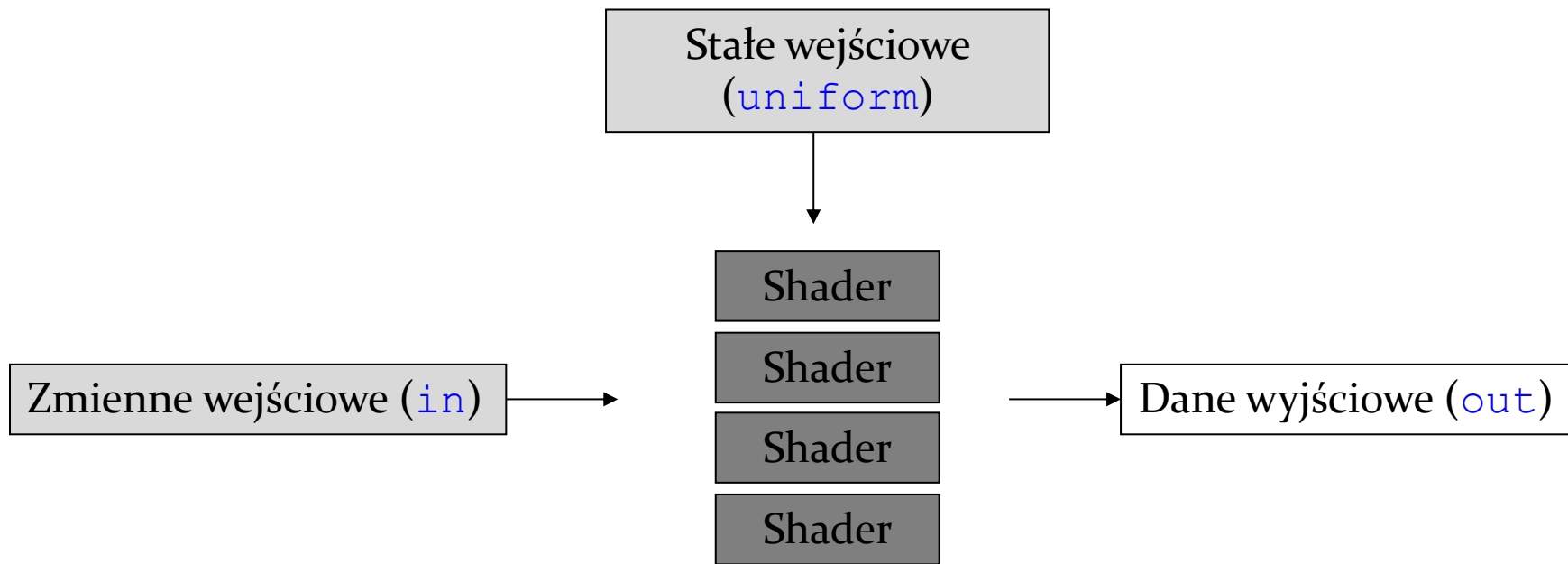
OpenGL



Język programistyczny: GLSL

- GLSL jest podobny do C **ale bez**
 - Wskaźników
 - Rekursji
 - Dynamicznej alokacji pamięci
- GLSL jest podobny do 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 shaderów

`out`: dane wyjściowe shadera

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 shaderów

out: dane wyjściowe shadera

`gl_Position` : pozycja wierzchołka – musi być podana!

GLSL syntaks

- GLSL ma preprocesora

```
#version 430

#ifdef FUNKCJA_1
    Funkcja1();
#else
    Funkcja2();
#endif
```

- Wszystkie shadery mają funkcję main()

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

GLSL: Typy

- Typy skalarne: `float`, `int`, `uint`, `bool`
- Wektory też są typy domyślne:
 - `vec2`, `vec3`, `vec4`
 - `ivec*`, `uvec*`, `bvec*`
- Elementy wektorów można na kilka sposobów indeksować:
 - `.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 od 0
```

GLSL: Macierze

- Macierze są typy domyślne:
 - kwadratowa: `mat2`, `mat3`, `mat4`
 - M x N: `matmxn` `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 - nie iloczyn skalarny!
```

```
mat3 m = // ...
```

```
mat3 v = // ...
```

```
mat3 mv = v * m; // macierz * macierz
```

```
vec3 xyz2 = mv * xyz; // macierz * wektor
```

```
vec3 xyz3 = xyz * mv; // wektor * macierz (wektor jest transponowany)
```

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 l1 = log(x);    // ln  
float l2 = log2(x);   // log2  
  
float s = sqrt(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 if (x == 0.0)  
{  
    f = 0.0;  
}  
else  
{  
    f = -2.0;  
}
```

`sign(x) * 2.0`

GLSL domyślne funkcje

- Wybrane funkcje geometryczne

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

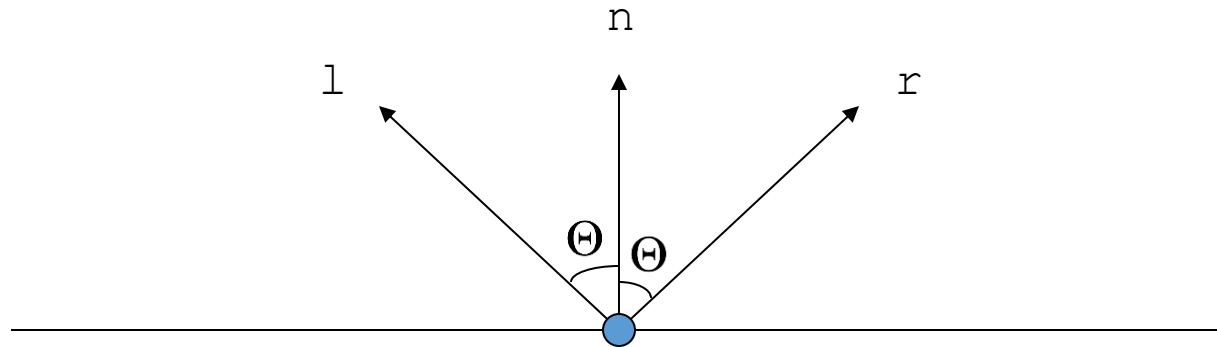
float f = length(l); // długość wektora
float d = distance(p, q); // dystans pomiędzy dwoma punktami

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

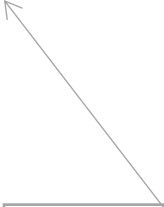
- `reflect(-l, n)`
 - Mając l i n , znajdź r .



GLSL domyślne funkcje

- Wszystkie funkcje zwracają wartości.

```
vec3 n = // ...  
normalize(n); // niepoprawnie
```



normalize() zwraca wartość
n = normalize(n);

GLSL domyślne funkcje

- Wybrane metody numeryczne

```
mat4 m = // ...
```

```
mat4 t = transpose(m);
```

```
float d = determinant(m);
```

```
mat4 d = inverse(m);
```

```
//transpozycja
```

```
//wskaźnik
```

```
//macierz odwrotna
```

GLSL domyślne funkcje

- Wybrane funkcje wektorowe

```
vec3 p = vec3(1.0, 2.0, 3.0);  
vec3 q = vec3(3.0, 2.0, 1.0);  
  
bvec3 b = equal(p, q);           // (false, true, false)  
bvec3 b2 = lessThan(p, q);       // (true, false, false)  
bvec3 b3 = greaterThan(p, q);    // (false, false, true)  
  
bool b4 = any(b);                // true  
bool b5 = all(b);                // false
```

GLSL domyślne funkcje

- Przykład

```
bool foo(vec3 p, vec3 q)
{
    if (p.x < q.x)
    {
        return true;
    }
    else if (p.y < q.y)
    {
        return true;
    }
    else if (p.z < q.z)
    {
        return true;
    }
    return false;
}
```

```
return any(lessThan(p, q));
```