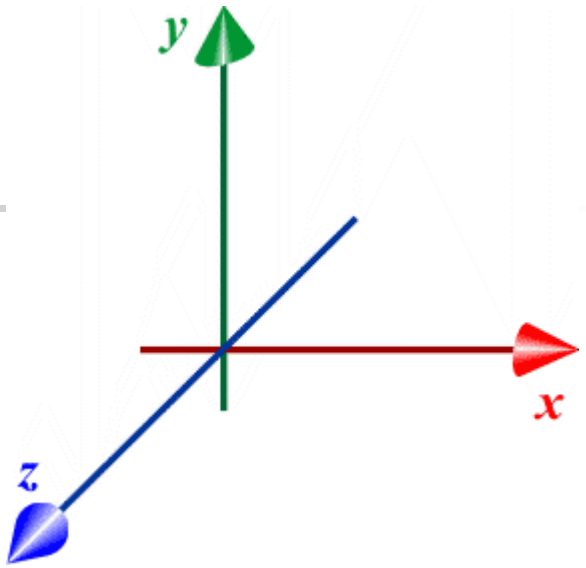
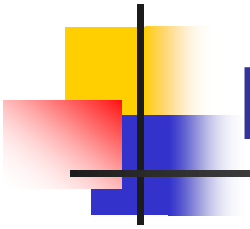
Wykład 1:




# Geometria grafiki 2D i 3D

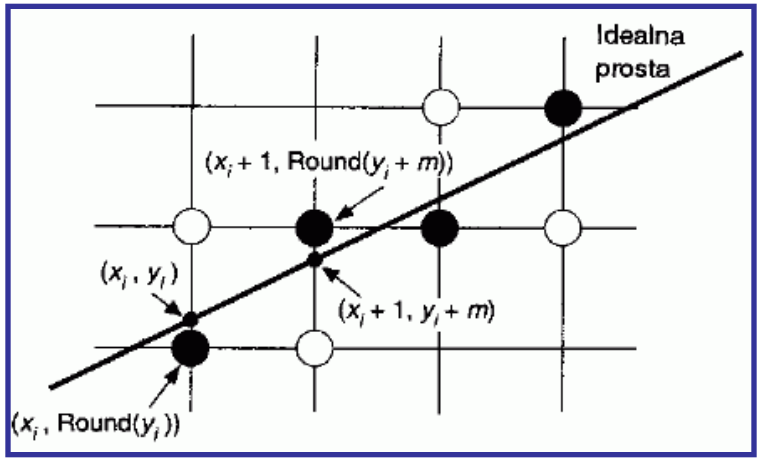


# podstawowe operacje grafiki 2D:

Tworzenie obrazu w rastrowej grafice 2D (np. na monitorze) wymaga wypełnienia wybranego obszaru pikseli określonymi barwami w taki sposób, aby powstał zamierzony rysunek.
Podstawowe operacje grafiki 2D to:
- rysowanie odcinka
- rysowanie podstawowych figur płaskich, np. łuku, okręgu, prostokąta
- wypełnianie obszaru
- obcinanie obrazu do widocznego obszaru

Podstawowe elementy obrazu 2D (np. odcinek, okrąg) określa się mianem prymitywów.
Z prymitywów składane są obrazy złożone.

# rysowanie odcinka:




**Algorytm przyrostowy**:

- ustalamy punkt początkowy i końcowy odcinka: $(x_0,y_0)$, $(x_k,y_k)$
- obliczamy nachylenie odcinka

$$m = (y_k-y_0)/(x_k-x_0)$$

- zwiększamy x o 1 i obliczamy y

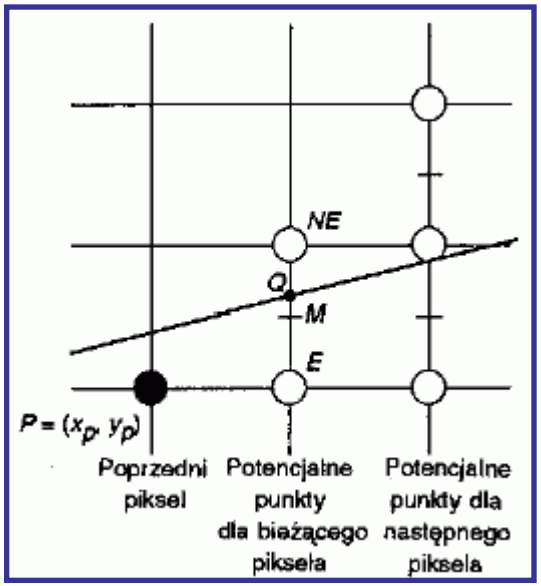
$$y_{i+1} = y_i + m \quad \text{dla} \quad i = 0,1,\ldots,k-1$$

zaokrąglając kolejną obliczoną wartość y do liczby całkowitej

Rysowane są piksele leżące najbliżej idealnego odcinka.

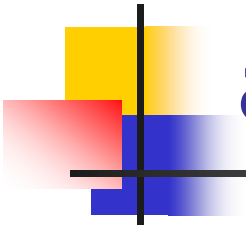
Algorytm wymaga operacji zmiennoprzecinkowych.

# rysowanie odcinka:




**Algorytm z punktem środkowym** (algorytm Bresenhama):

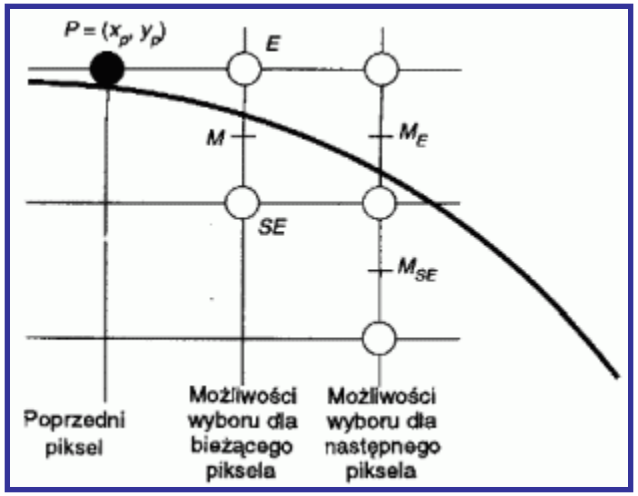
- algorytm operuje na liczbach całkowitych
- w każdym kroku iteracji rozpatrywane są dwa piksele leżące najbliżej idealnej linii
- wprowadzana jest zmienna decyzyjna – odległość linii od tych pikseli
- wybierany jest punkt leżący bliżej linii na podstawie znaku zmiennej decyzyjnej
- zmienna decyzyjna jest uaktualniana
- obliczenia są kontynuowane iteracyjnie dla kolejnych pikseli



# algorytm Bresenhama:

```
procedure Bresline (x1,x2,y1,y2)
begin
   dx := x2-x1; dy := y2-y1;
   d := 2*dy-dx; p1 := 2*dy; p2 := 2*(dy-dx);
   x := x1; y := y1;
   xend := x2;
   set_pixel(x,y);
   while (x<xend) do
     begin
       x := x+1;
       if (d<0) then d := d+p1
                else begin
                       d := d+p2;
                       y := y+1;
                     end;
       set_pixel(x,y);
     end
end
```

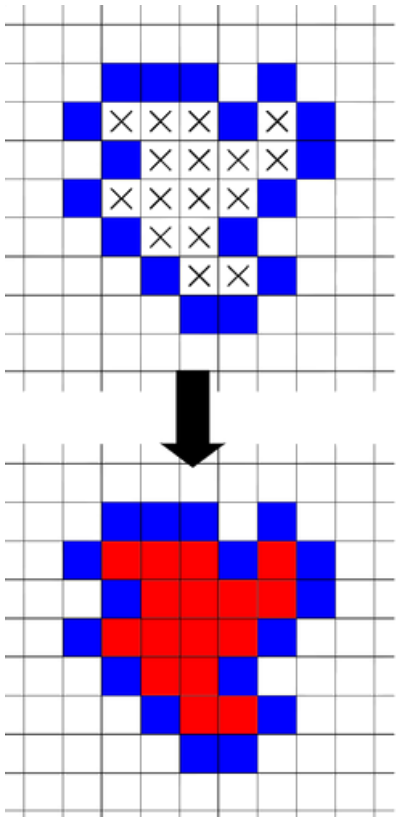
# rysowanie łuku okręgu:




**Algorytm z punktem środkowym**:
- podobny do algorytmu rysowania odcinka
- zmienna decyzyjna obliczana jest iteracyjnie
- wybór jednego z dwóch pikseli na podstawie znaku zmiennej decyzyjnej
- uaktualnienie zmiennej decyzyjnej
- obliczanie zmiennej decyzyjnej trudniejsze niż dla odcinka

Uwaga! Parametr charakteryzujący proporcje boków rastra pikseli w pionie i w poziomie, czyli aspekt ma wpływ na kształt rysowanego okręgu (elipsa)

# wypełnianie obszaru:

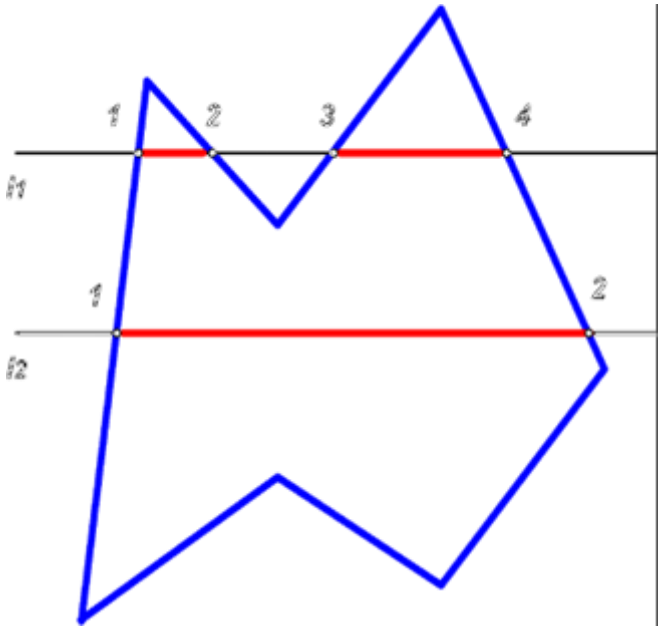


**Wypełnianie przez spójność**

- zakłada znajomość punktu startowego (tzw. „ziarna”) wewnątrz obszaru
- po wypełnieniu ziarna, startując z tego punktu wypełniamy punkty sąsiednie (o ile istnieją, nie są już wypełnione, ani nie są punktami granicznymi obszaru)
- punkty sąsiednie stają się wyjściowymi dla wypełniania w następnym kroku
- procedura jest powtarzana dopóki można wskazać punkty wyjściowe (niewypełnione) wewnątrz obszaru

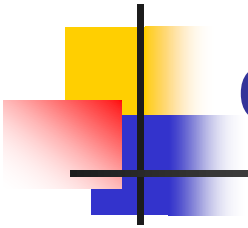
Algorytm wypełniania przez spójność jest algorytmem rekurencyjnym.

# wypełnianie obszaru:




**Wypełnianie przez kontrolę parzystości**

- jeśli punkty przecięcia ponumerujemy kolejnymi liczbami naturalnymi zgodnie z orientacją prostej (na rysunku od lewej do prawej dla prostej poziomej) i jeśli będziemy poruszać się po prostej zgodnie z jej orientacją, to każde nieparzyste przecięcie będzie „wejściem” do wnętrza obszaru, natomiast każde parzyste będzie „wyjściem” na zewnątrz
- aby wypełnić obszar należy go przecięć prostymi odpowiadającymi kolejnym rzędom pikseli, a następnie wypełnić odcinkami pomiędzy każdym nieparzystym przecięciem, a najbliższym parzystym

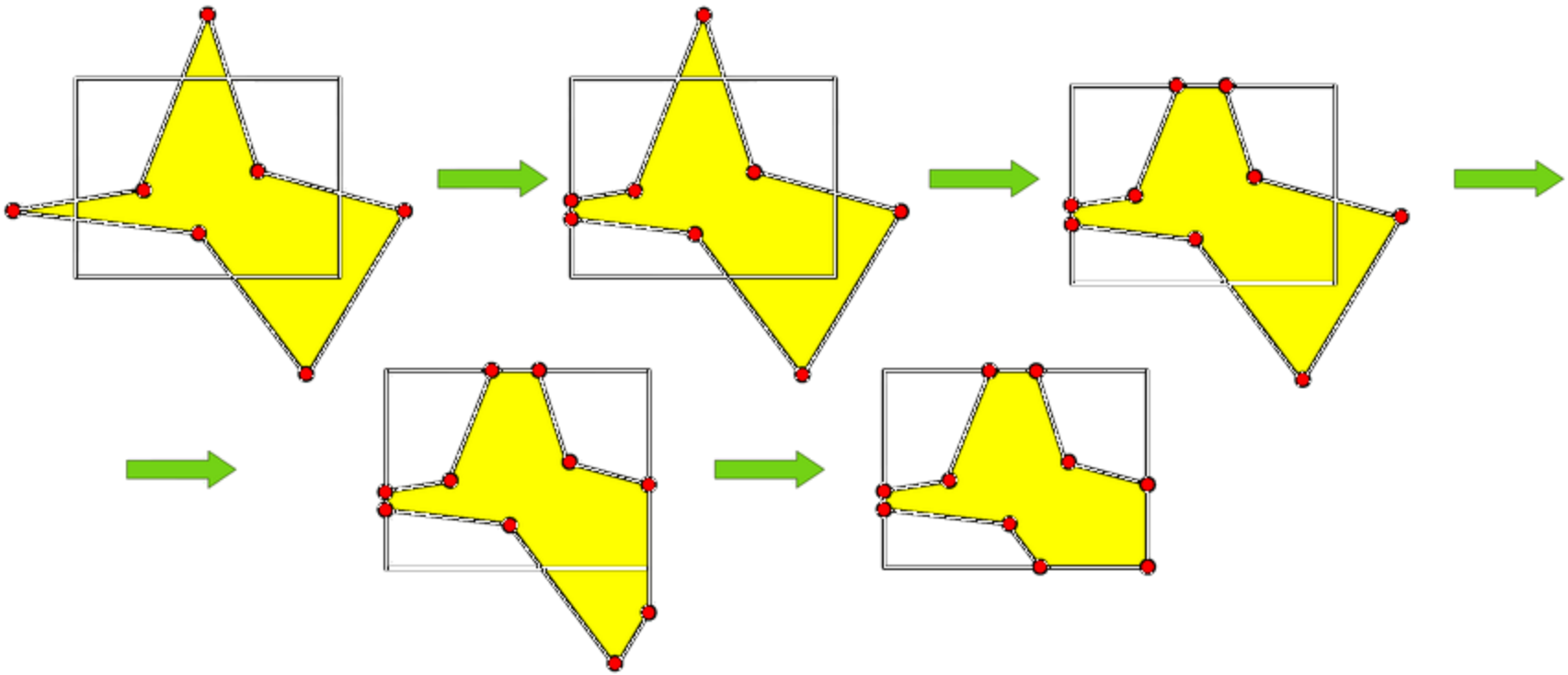


# obcinanie obrazu:

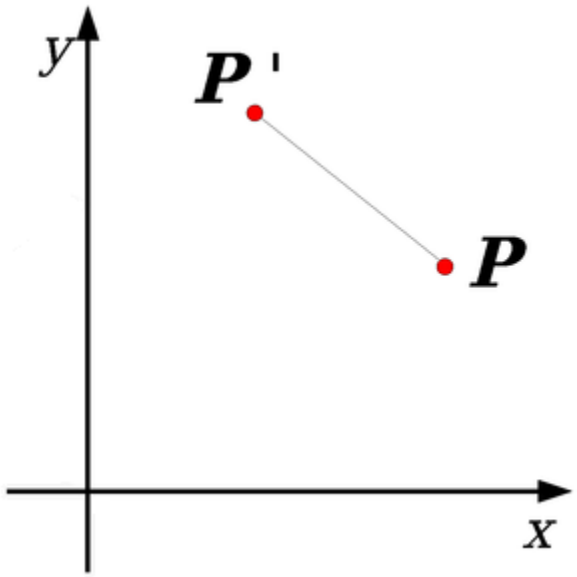
W praktyce część rysowanego prymitywu tworzącego obraz może znaleźć się poza obszarem wyświetlania (np. poza ekranem).
Stosuje się obcinanie (clipping) prymitywów poprzez:
- obcinanie przed rysowaniem
- obliczenie całego prymitywu, wyświetlanie widocznej części (wycinanie) występuje konieczność obliczeń wycinanej części
- rysowanie całych prymitywów w pamięci, wyświetlenie obciętego obszaru (nieefektywne)

Algorytm **Cohena-Sutherlanda** służy do obcinania odcinków do prostokątnego okna.
Algorytm **Sutherlanda-Hodgmana** umożliwia obcinanie wielokąta.
Algorytm **Cyrusa-Becka** oferuje efektywny zestaw operacji parametrycznych do analizy całego wielokąta i realizacji jego obcinania.

# algorytm Sutherlanda-Hodgmana :



# przekształcenia geometryczne 2D:

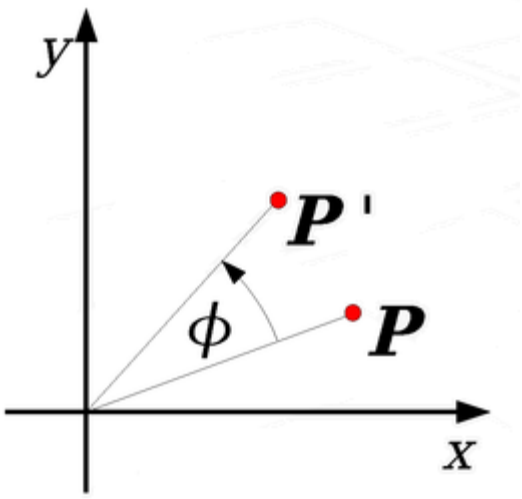



Podstawowe przekształcenia geometryczne obrazu na płaszczyźnie 2D to:
- obrót
- skalowanie
- przesunięcie (translacja)

Opis operacji geometrycznych musi być efektywny, prosty i ujednolicony. Takie warunki spełnia **opis macierzowy**.

Niech punkt P ma współrzędne (x,y). Wówczas wektor $P = [x\ y]^T$ opisuje położenie punktu P na płaszczyźnie.

# obrót:




Niech punkt P’ ma współrzędne (x’,y’).
Wówczas:

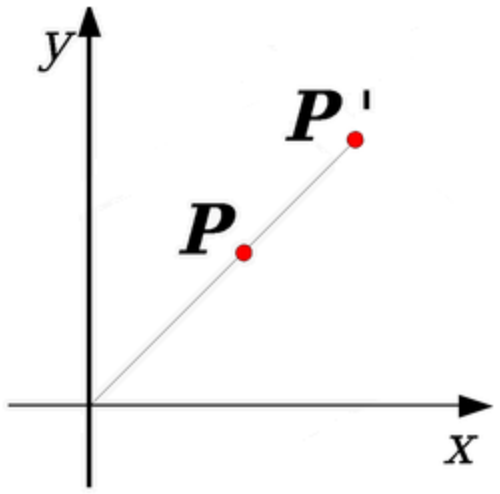
x’ = x*cos(ϕ) – y*sin(ϕ)
y’ = x*sin(ϕ) +y*cos(ϕ)

Jeśli przyjąć macierz $M_O$ postaci:

$$M_O = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}$$

to możemy napisać, że P’ = $M_O$*P

# skalowanie:




Niech punkt P’ ma współrzędne (x’,y’).
Wówczas:

$$x' = s*x$$
$$y' = s*y$$

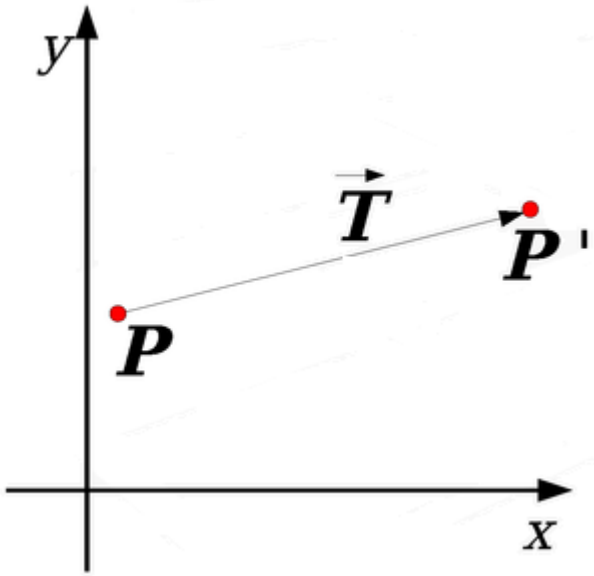
gdzie s jest współczynnikiem skalowania.

Jeśli przyjąć macierz $M_S$ postaci:

$$M_S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

to możemy napisać, że $P' = M_S * P$

# przesunięcie (translacja):




Niech wektor $T=[T_X\ T_Y]$ opisuje przesunięcie punktu P do P'. Wówczas:

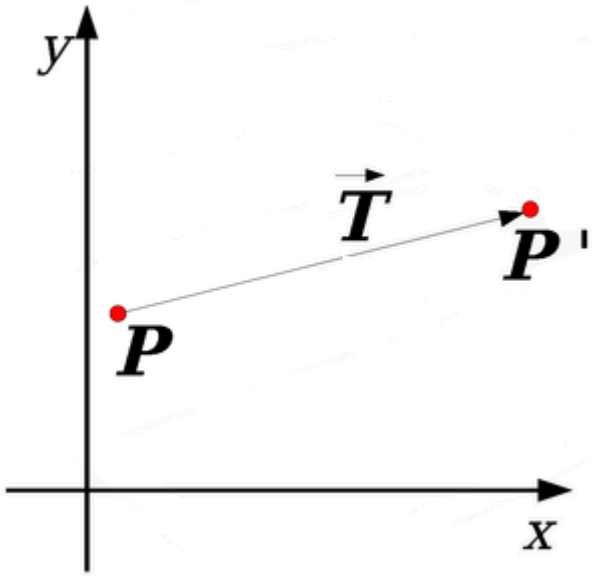
$$x' = x + T_X$$
$$y' = y + T_Y$$

Jeśli przyjąć macierz $M_T$ postaci:

$$M_T = \begin{bmatrix} T_X \\ T_Y \end{bmatrix}$$

to możemy napisać, że $P' = P + M_T$

# przesunięcie (translacja):




Zapis $P' = P + M_T$ nie jest jednak iloczynem macierzy i wektora współrzędnych punktu. Jeśli teraz przyjąć macierz $M_T$ postaci:

$$M_T = \begin{bmatrix} 1 & 0 & T_X \\ 0 & 1 & T_Y \\ 0 & 0 & 1 \end{bmatrix}$$

to możemy napisać, że

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_X \\ 0 & 1 & T_Y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# współrzędne jednorodne:

Zauważmy, że zapis operacji przesunięcia punktu w postaci iloczynu macierzy i wektora współrzędnych tego punktu stał się możliwy po wprowadzeniu tzw. **współrzędnych jednorodnych** punktu.

Punkt o współrzędnych $\begin{bmatrix} x \\ y \end{bmatrix}$ w 2D ma współrzędne jednorodne znormalizowane w postaci $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Analogicznie punkt o współrzędnych $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ w 3D ma współrzędne jednorodne znormalizowane w postaci $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$.

# macierze przekształceń w 2D:

Punkt o współrzędnych $\begin{bmatrix} x \\ y \end{bmatrix}$ w 2D ma współrzędne jednorodne znormalizowane w postaci $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Przekształcenie punktu P w P' można opisać w postaci P' = M * P, gdzie M jest macierzą 3x3 i ma postać:

- **symetria względem osi OX**: $M_{SOX} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- **symetria względem osi OY**: $M_{SOY} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- **symetria środkowa**: $M_{SS} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

# macierze przekształceń w 2D:

- **obrót**:

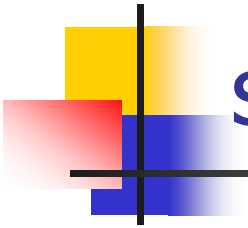
$$M_O = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **skalowanie**:
  zwykle $S_X = S_Y = s$

$$M_S = \begin{bmatrix} S_X & 0 & 0 \\ 0 & S_Y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **przesunięcie**:
  wektor przesunięcia $[T_X\ T_Y]$

$$M_T = \begin{bmatrix} 1 & 0 & T_X \\ 0 & 1 & T_Y \\ 0 & 0 & 1 \end{bmatrix}$$



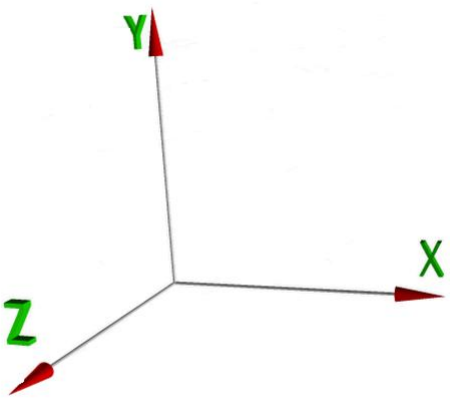
# składanie przekształceń:

Składanie przekształceń to wykonywanie mnożeń przez odpowiednie macierze. Jeśli macierz M opisuje całkowite przekształcenie będące wynikiem złożenia przekształceń elementarnych $M_1$, $M_2$, $M_3$, ..., $M_N$ (w takiej kolejności) to:

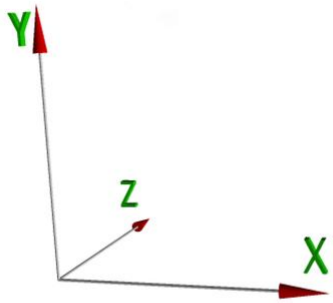
$$\mathbf{M = M_N * \ldots * M_3 * M_2 * M_1}$$

Mnożenie macierzy (składanie przekształceń geometrycznych) jest operacją łączną, ale nie jest operacją przemienną – złożenie najpierw translacji, a następnie obrotu jest innym przekształceniem, niż złożenie najpierw obrotu, a następnie translacji.

# śruba
# i dwa układy współrzędnych 3D:




Układ prawoskrętny




Układ lewoskrętny

Znane są dwa ustawienia osi trójwymiarowego układu współrzędnych:

- **układ prawoskrętny** – opis położenia obiektów na scenie (w przestrzeni obiektu)
- **układ lewoskrętny** – opis położenia obiektów związany z rzutowaniem; jeśli osie OX i OY definiują układ współrzędnych na **rzutni** (utożsamianej z płaszczyzną XOY), to kierunek wzrostu odległości od obserwatora wskaże oś OZ

Do rozpoznania rodzaju układu możemy użyć reguły śruby (prawo lub lewoskrętnej)

# macierze przekształceń w 3D :

Punkt o współrzędnych $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ w 3D ma współrzędne jednorodne znormalizowane w postaci $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$.

Przekształcenie punktu P w P’ można opisać w postaci P’ = M * P, gdzie M jest macierzą 4x4 i ma postać:

- **przekształcenie tożsamościowe**:
  P’ = $M_I$ * P = P

$$M_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# macierze przekształceń w 3D :

- **symetrie osiowe**:
  (tu symetria względem osi OX)
  macierze opisujące symetrie osiowe względem osi OY i OZ mają analogiczną postać ze zmienionymi znakami w odpowiednich kolumnach

$$M_{OX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **symetrie płaszczyznowe**:
  (tu symetria względem płaszczyzny YOZ)
  macierze opisujące symetrie płaszczyznowe względem płaszczyzn XOY i XOZ mają analogiczną postać ze zmienionym znakiem w odpowiedniej kolumnie

$$M_{YOZ} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# macierze przekształceń w 3D :

**- symetria środkowa**:

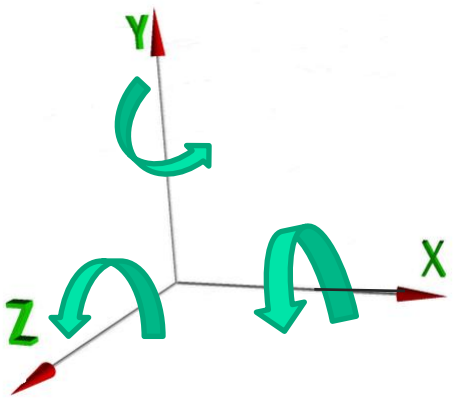
$$M_{SS} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**- zmiana skrętności układu współrzędnych**:
zmianę skrętności zapewnia macierz symetrii płaszczyznowej względem płaszczyzny XOY

$$M_{XOY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# macierze przekształceń w 3D :

- **obroty**:
  dodatnie obroty w prawoskrętnym układzie współrzędnych



Układ prawoskrętny

- **obrót o kąt $\phi_X$ wokół osi OX**:

$$M_{OX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi_X) & -\sin(\phi_X) & 0 \\ 0 & \sin(\phi_X) & \cos(\phi_X) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# macierze przekształceń w 3D :

**- obrót o kąt $\phi_Y$ wokół osi OY**:

$$M_{OY} = \begin{bmatrix} \cos(\phi_Y) & 0 & \sin(\phi_Y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi_Y) & 0 & \cos(\phi_Y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**- obrót o kąt $\phi_Z$ wokół osi OZ**:

$$M_{OZ} = \begin{bmatrix} \cos(\phi_Z) & -\sin(\phi_Z) & 0 & 0 \\ \sin(\phi_Z) & \cos(\phi_Z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# macierze przekształceń w 3D :

**- skalowanie**:
zwykle $S_X = S_Y = S_Z = s$

$$M_S = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**- przesunięcie**:
wektor przesunięcia $[T_X\ T_Y\ T_Z]$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & T_X \\ 0 & 1 & 0 & T_Y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

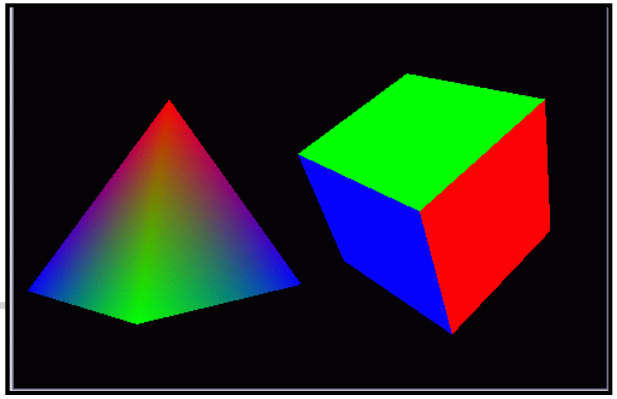
# przekształcenie odwrotne:

Jeśli macierz  opisuje pewne przekształcenie geometryczne, przekształcenie współrzędnych jest opisane równaniem $P' = M * P$  i jeśli istnieje przekształcenie odwrotne opisane macierzą $M^{-1}$, to $P = M^{-1} * P'$.

Gdy M będzie wynikiem złożenia odwracalnych przekształceń elementarnych $M_1$, $M_2$, $M_3$, ..., $M_N$ (w takiej kolejności), czyli $M = M_N * ... * M_3 * M_2 * M_1$, to wówczas: $M^{-1} = M_1^{-1} * M_2^{-1} * M_3^{-1} * ... * M_N^{-1}$

Ponieważ symetrie, obroty, skalowanie i przesunięcie opisane odpowiednimi macierzami są operacjami odwracalnymi, to w bardzo prosty sposób można pokazać, że dowolna operacja będąca złożeniem dowolnego zestawu tych operacji, jest też operacją odwracalną.

# rzutowanie:



**Rzutowanie** to operacja pozwalająca przedstawić obiekty trójwymiarowe na płaszczyźnie (tzw. **rzutni**).

Rzutowanie jest przekształceniem przestrzeni trójwymiarowej na przestrzeń dwuwymiarową. Rzutowanie polega na poprowadzeniu prostej przez dany punkt obiektu i znalezieniu punktu wspólnego tej prostej z rzutnią. Wyznaczony punkt nazywany jest **rzutem**, a prosta - promieniem rzutującym.

W szczególnych przypadkach powierzchnia rzutowania nie musi być płaszczyzną, rozpatruje się np. rzuty na powierzchnię walca lub na wycinek sfery.

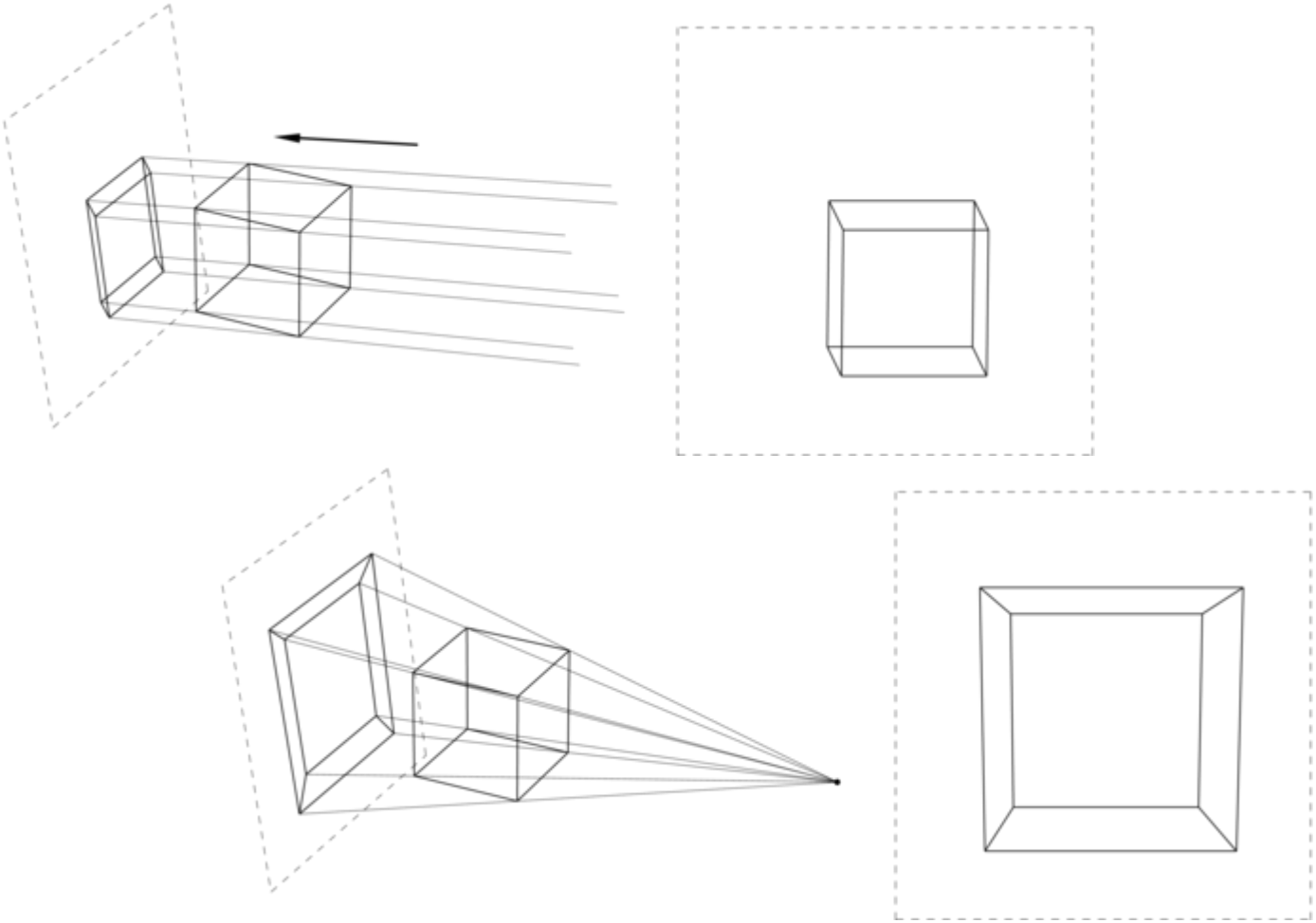
# rodzaje rzutowania:

**Rzutowanie równoległe** - promienie rzutujące są prostymi równoległymi:
- **rzutowanie równoległe prostokątne** - rzutnia jest prostopadła do kierunku rzutowania
- **rzutowanie równoległe ukośne** - w każdym innym przypadku.

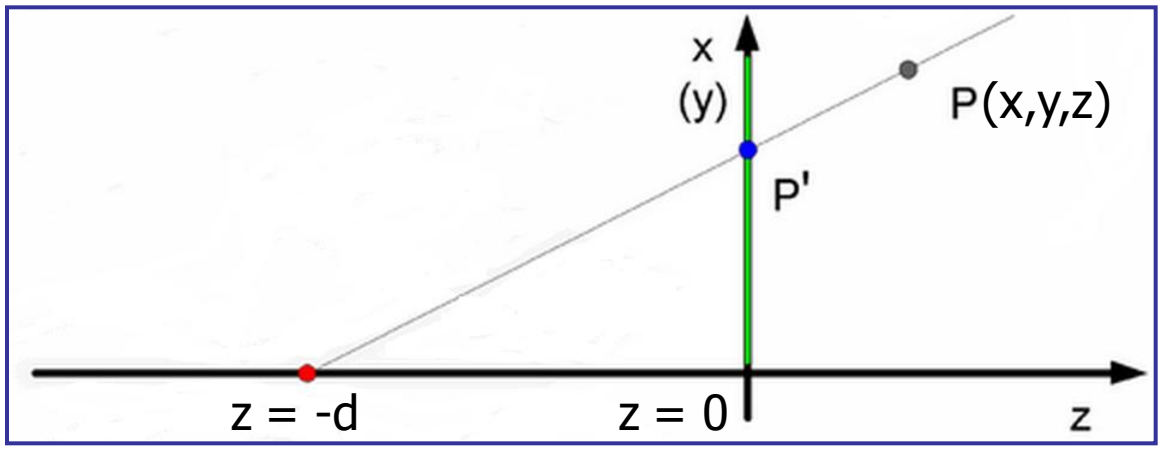
Rzutowanie równoległe:
- nie pozwala przedstawić obiektu zgodnie z widzeniem człowieka
- pozwala zdefiniować wymiary obiektu i jego parametry technologiczne, ponieważ zachowuje równoległość prostych oraz proporcje długości odcinków równoległych

**Rzutowanie perspektywiczne (środkowe lub centralne)** - promienie rzutujące tworzą pęk prostych:
- pozwala uzyskać obraz zbliżony do postrzeganego przez człowieka.

# rzutowanie równoległe i perspektywiczne :



# rzut perspektywiczny (I):



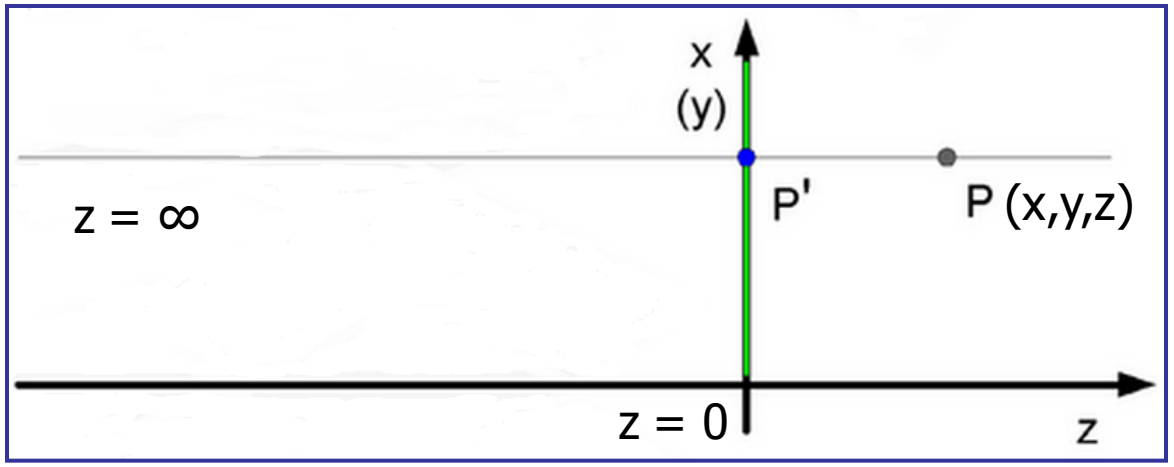

lewoskrętny układ OXYZ współrzędnych obserwatora
obserwator (środek rzutowania) znajduje się w punkcie (0,0,-d)
płaszczyzna rzutni ma równanie z = 0

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ \frac{z}{d}+1 \end{bmatrix}$$

*po normalizacji*

$$\begin{bmatrix} \frac{xd}{z+d} \\ \frac{yd}{z+d} \\ 0 \\ 1 \end{bmatrix}$$

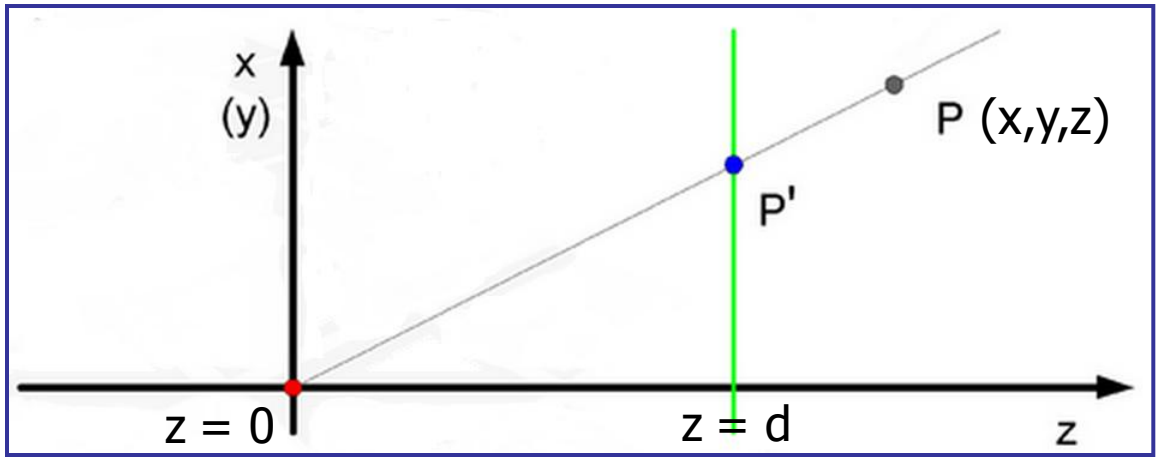
# rzut równoległy, prostokątny:




lewoskrętny układ OXYZ współrzędnych obserwatora
obserwator (środek rzutowania) znajduje się w punkcie (0,0,-∞)
płaszczyzna rzutni ma równanie z = 0

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

# rzut perspektywiczny (II):




lewoskrętny układ OXYZ współrzędnych obserwatora
obserwator (środek rzutowania) znajduje się w punkcie (0,0,0)
płaszczyzna rzutni ma równanie z = d

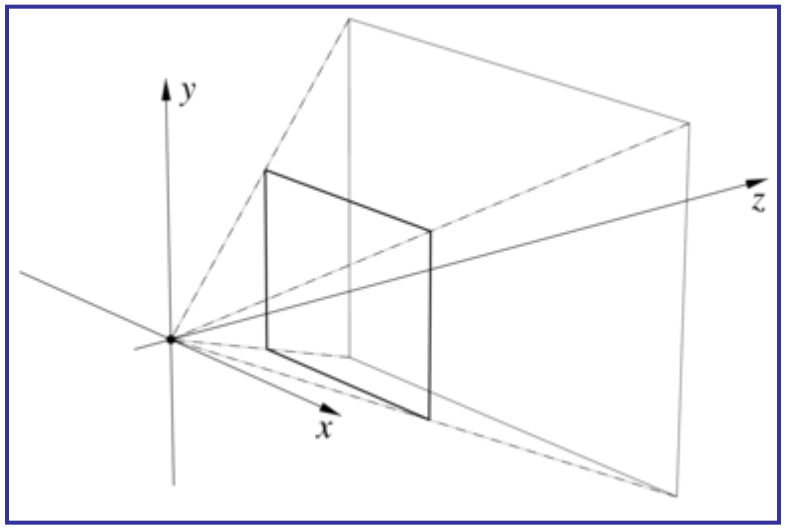
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

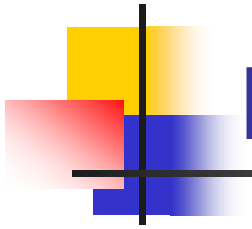
*po normalizacji*

$$\begin{bmatrix} \frac{xd}{z} \\ \frac{yd}{z} \\ d \\ 1 \end{bmatrix}$$

# ostrosłup widzenia:

Gdy dokonujemy rzutowania perspektywicznego na płaszczyznę, interesuje nas zwykle tylko prostokąt obrazu będący część rzutni. Wówczas środek rzutowania i prostokąt obrazu wyznaczą pewien fragment przestrzeni, który ma podlegać rzutowaniu. Jeżeli do tego dodamy dwie płaszczyzny równoległe do rzutni, które ograniczą wybrany fragment z przodu i z tyłu, to powstanie figura będąca ostrosłupem ściętym o podstawie prostokątnej nazywana **ostrosłupem widzenia** lub **piramidą widzenia**.





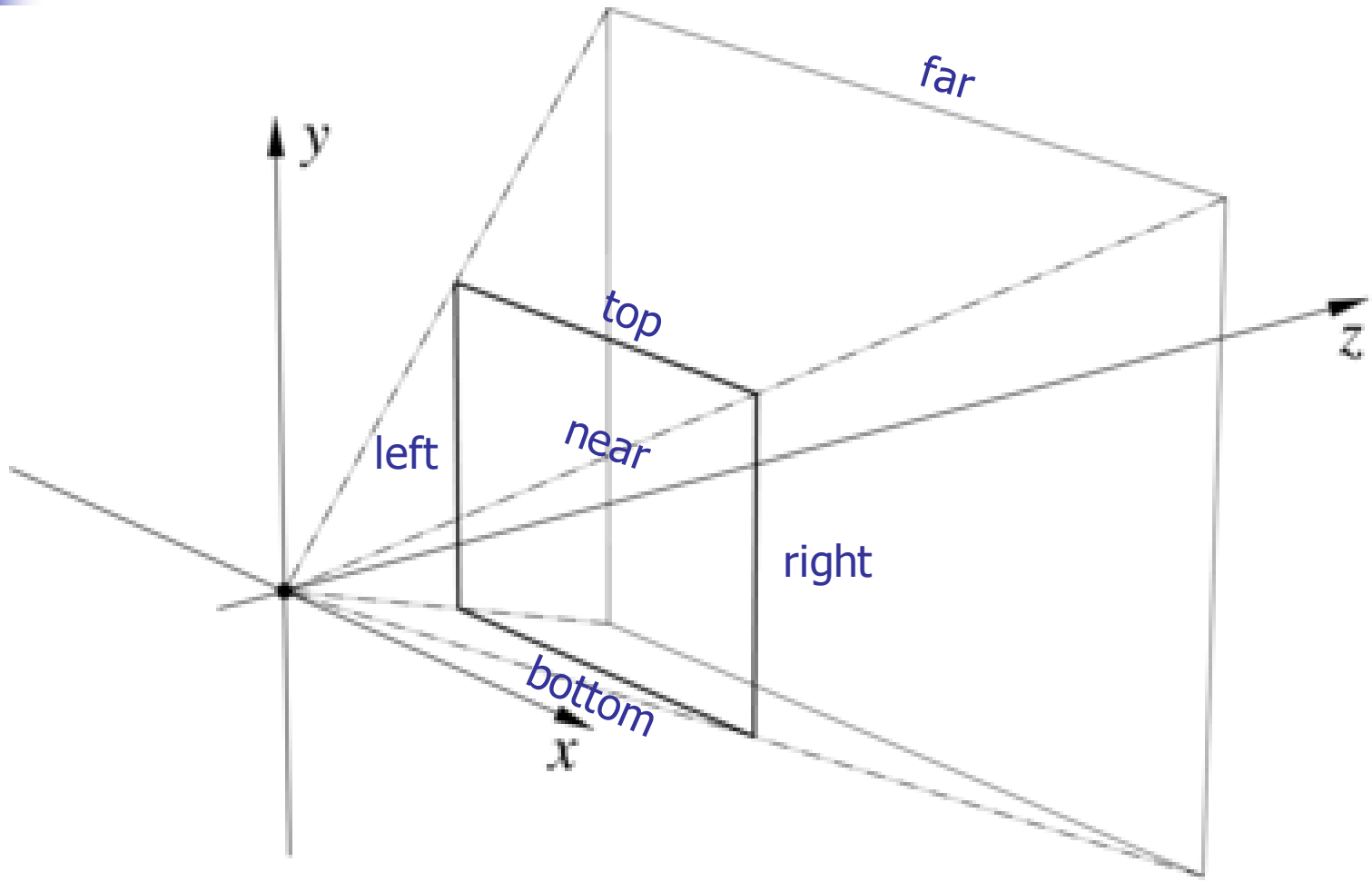
# przekształcenie perspektywiczne:

Niech środek rzutowania ma współrzędne (0,0,0), a prostokąt obrazu określają proste x=l (left), x=r (right), y=b (bottom), y=t (top).
Niech płaszczyzny podstaw ostrosłupa widzenia mają równania z=n (near) i z=f (far) (gdzie n<f), a rzutnia należy do płaszczyzny z=n. Wówczas przekształcenie perspektywiczne opisane macierzą 4x4, postaci:

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

przekształca ostrosłup widzenia w kostkę [-1,1] x [-1,1] x [0,1] zachowując informację o głębokości, tj. odległości punktu od środka rzutowania.

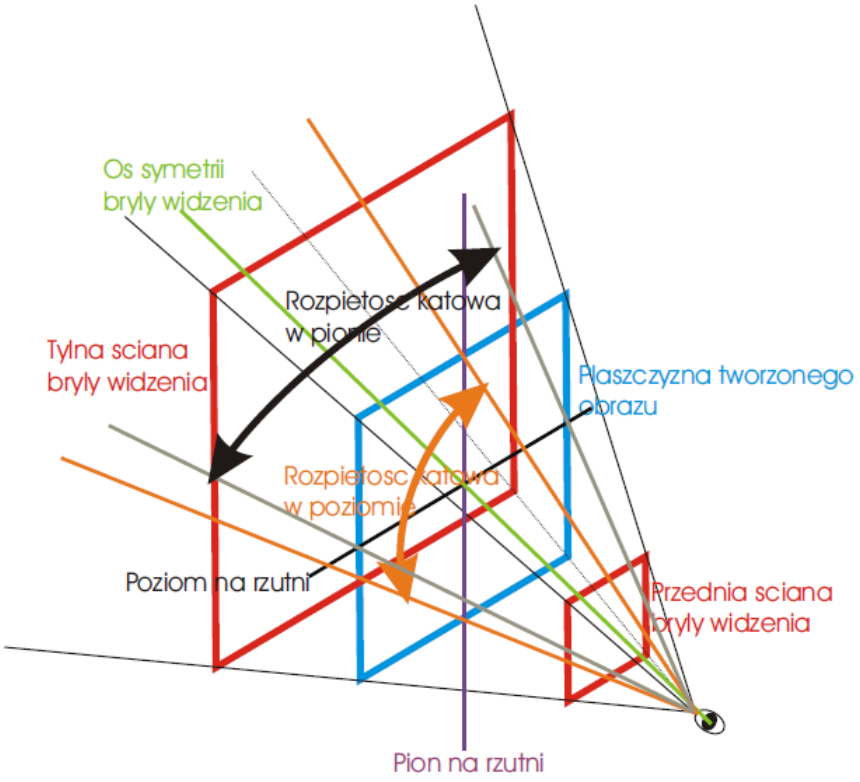
# ostrosłup widzenia (1):



# ostrosłup widzenia (2):

W ogólności ostrosłup widzenia możemy określić podając:
- rozpiętość kątową w pionie
- stosunek rozpiętości kątowej w poziomie do rozpiętości kątowej w pionie
- odległość obserwatora od przedniej ściany bryły widzenia
- odległość obserwatora od tylnej ściany bryły widzenia

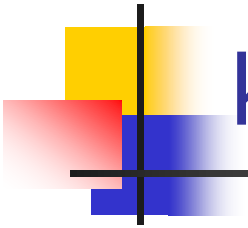
Oś symetrii ostrosłupa widzenia wyznaczona jest przez główny kierunek patrzenia definiowany dla obserwatora.



# wyznaczanie powierzchni widocznych:

Przy wyświetlaniu sceny 3D złożonym problemem jest rendering powierzchni.
W celu zminimalizowania czasu i ilości obliczeń potrzebnych do prawidłowego zaprezentowania sceny, przed tym etapem dokonuje się analizy sceny w celu wyznaczenia, które powierzchnie są dla obserwatora widoczne **(zostaną umieszczone na rzutni), a które pozostają ukryte (znajdują się „za obiektem”** lub są **„przysłonięte”** przez obiekty znajdujące się bliżej) i nie wymagają przetworzenia (tzw. problem eliminacji elementów zasłoniętych).
Dodatkowo, ukrycie przed obserwatorem niektórych powierzchni daje bardziej realistyczny efekt trójwymiarowości.

Jest wiele algorytmów rozwiązujących sformułowany wyżej problem, np. algorytm skaningowy lub algorytm drzewa podziału binarnego.

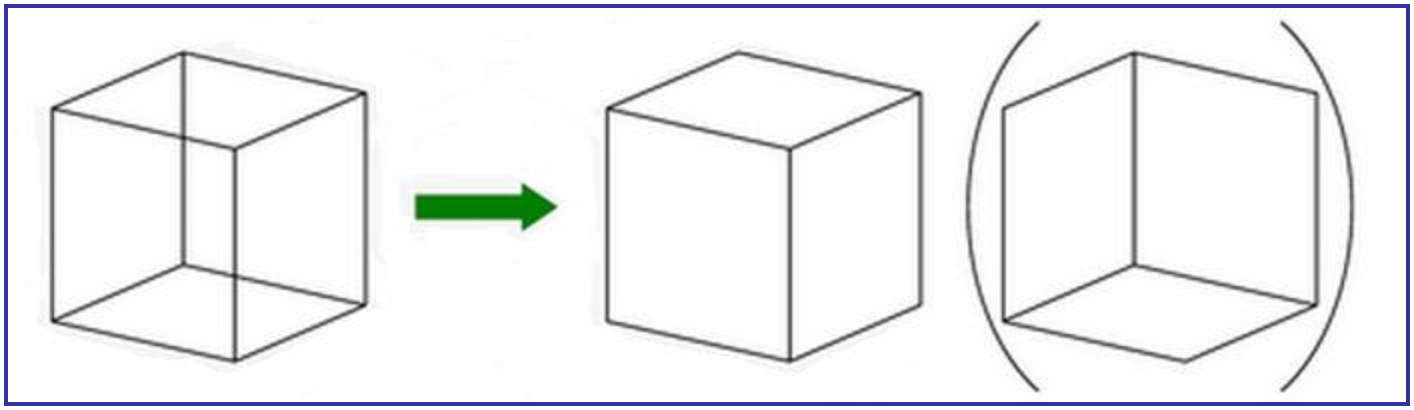


# klasy algorytmów:

Algorytmy rozwiązania problemu widoczności mogą należeć do jednej z trzech klas:
- algorytmy wykorzystujące informacje pochodzące z przestrzeni obserwatora obiektu (sceny)
- algorytmy wykorzystujące informacje pochodzące z przestrzeni rzutu
- algorytmy wykorzystujące informacje pochodzące z obu tych przestrzeni.

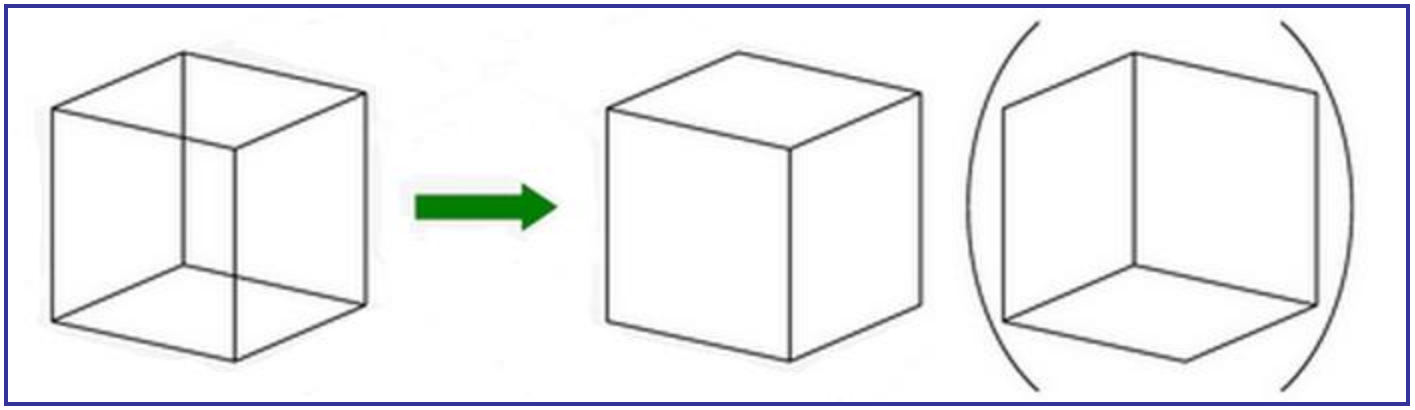
Z drugiej strony często mówi się o precyzji obrazowej lub obiektowej danego algorytmu:
- algorytmy pracujące z precyzją obrazową wykonują obliczenia z dokładnością urządzenia wyświetlającego (wynikającą z wyświetlania pojedynczego piksela)
- algorytmy pracujące z precyzją obiektową wykonują obliczenia z dokładnością związaną z definicją obiektów, a nie pikseli.

# prosty przykład:



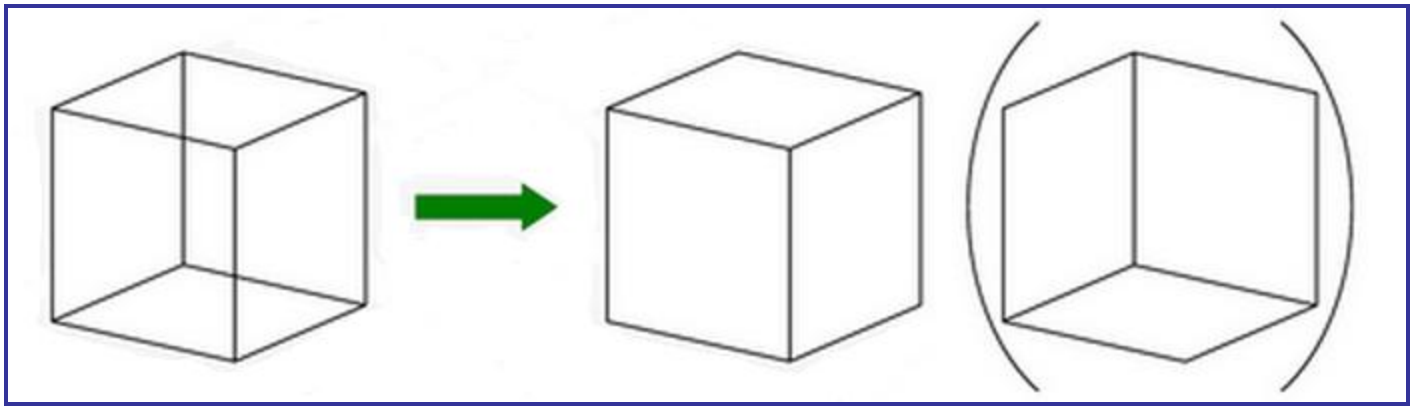
Dokonujemy analizy położenia obiektów w przestrzeni. Definiowane są wektory normalne do powierzchni obiektu, a następnie jest wyznaczany iloczyn skalarny tych wektorów. Znak tego iloczynu określa przynależność do określonej grupy.

# prosty przykład:

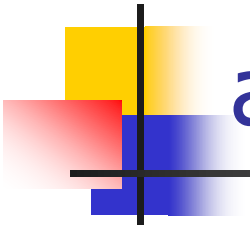


Opieramy się tylko na informacji z przestrzeni rzutu. Dzielimy wierzchołki rzutu na dwa typy. Pierwszy typ charakteryzuje się tym, że węzeł oraz zewnętrzne krawędzie są widoczne. Drugi typ charakteryzuje się tym , że co prawda nie można stwierdzić czy węzeł jest widoczny czy nie, ale za to wszystkie elementy mają taki sam atrybut. A więc jeśli jakaś jedna krawędź jest widoczna, to cały węzeł i wszystkie jego krawędzie są widoczne.

# prosty przykład:

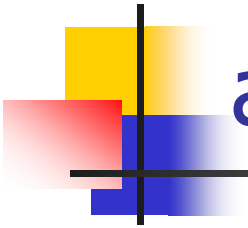


Korzystamy z informacji zarówno pochodzących z przestrzeni obiektu (sceny), jak i z informacji jakie dostarcza rzut. Porównywane jest zorientowanie ściany obiektu (w przestrzeni obiektu) i zorientowanie rzutu ściany (w przestrzeni rzutu). Jeśli oba zorientowania są jednakowe, to ściana jest widoczna (przednia).



# algorytm malarski:

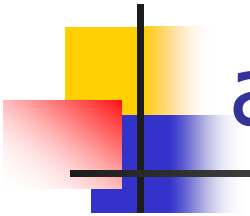
**Algorytm malarski (algorytm sortowania ścian)** (1972):
- wykorzystuje informacje zarówno z przestrzeni obiektu jak i z przestrzeni rzutu (łączy operacje z precyzją obiektową i operacje z precyzją obrazową)
- polega na posortowaniu elementarnych fragmentów sceny (wielokątów) zależnie od odległości od obserwatora: od najdalszego do najbliższego, koszt sortowania wynosi O(n*logn), gdzie n to ilość fragmentów obrazu
- fragmenty są następnie rysowane w uzyskanej z posortowania kolejności; zakłada się przy tym, że rysunek powstaje analogicznie do malowania obrazu olejnego (stąd nazwa algorytmu), tzn. fragment później malowany zasłania (zamalowuje) wszystko to, co było dotychczas namalowane w tym miejscu (tę cechę ma każde urządzenie wyświetlające, np. monitor)



# algorytm malarski:

Jak sortować elementarne fragmenty sceny (wielokąty)?
Jeżeli rozpatrzymy dwa wielokąty dowolnie ustawione w przestrzeni, to podstawowym problemem jest ustalenie, który z nich jest bliżej, a który dalej od obserwatora (różne kryteria porównywania położenia):

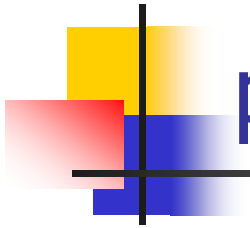
- porównujemy współrzędne **z** wielokątów; jeśli maksymalna współrzędna **z** jednego wielokąta jest mniejsza niż minimalna współrzędna **z** drugiego, to pierwszy jest bliżej obserwatora (najczęściej zakresy współrzędnych wielokątów zazębiają się nie dając możliwości prawidłowego posortowania)
- porównujemy położenia środków ciężkości wielokątów – bliżej obserwatora znajduje się ten, którego środek ciężkości znajduje się bliżej
- analizujemy położenie jednego wielokąta względem płaszczyzny wyznaczonej przez drugi wielokąt; jeżeli pierwszy z nich jest po tej samej stronie płaszczyzny co obserwator, to jest bliżej obserwatora i nie może być zasłonięty przez drugi wielokąt.



# algorytm bufora głębokości:

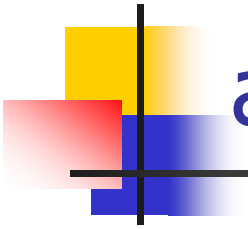
**Algorytm bufora głębokości (algorytm Z-bufora)** (1974):
- zakłada się istnienie bufora o rozmiarze całego wyświetlanego obszaru (ekranu) – występuje w tym przypadku odpowiedniość pozycji piksela ekranu i odpowiadającej mu pozycji w buforze
- dla każdego piksela w buforze zapamiętywana jest odpowiadająca mu głębokość, czyli współrzędna **z**
- na początku pracy algorytmu Z-bufor jest wypełniany maksymalną wartością współrzędnej **z**, jaka może wystąpić w analizowanym obszarze, jednocześnie wszystkie piksele obrazu przyjmują barwę tła
- następnie rysowane są wielokąty (w dowolnej kolejności), tzn. ich rzuty wypełniane są odpowiednią barwą (piksel jest wypełniony tylko wtedy, gdy jego współrzędna **z** jest mniejsza, niż wartość zapisana w buforze

Podczas wypełniania kolejnych wielokątów szukany jest zatem piksel, którego współrzędna **z** jest najmniejsza, tzn. szukany jest punkt leżący najbliżej obserwatora, czyli punkt rzeczywiście widoczny.



# pseudokod algorytmu Z-bufora:

```
void Z-Buffer {
  int pz; // bieżąca wartość głębokości z dla piksela o współrzędnych (x, y)
  for (y = 0; y < YMAX; y++) {
    for (x = 0; x < XMAX; x++) {
      WritePixel (x, y, BACKGROUND_VALUE);
      WriteZ (x, y, ∞);
    }
  }
  for (każdy wielokąt ) {
    for (każdy piksel w rzucie wielokąta ) {
      pz = wartość głębokości z danego wielokąta dla piksela o współrzędnych (x,y);
      if (pz <= ReadZ (x, y)) { // nowy punkt nie jest dalej
        WriteZ (x, y, pz);
        WritePixel (x, y, barwa danego wielokąta dla piksela o współrzędnych (x,y);
      }
    }
  }
}
```



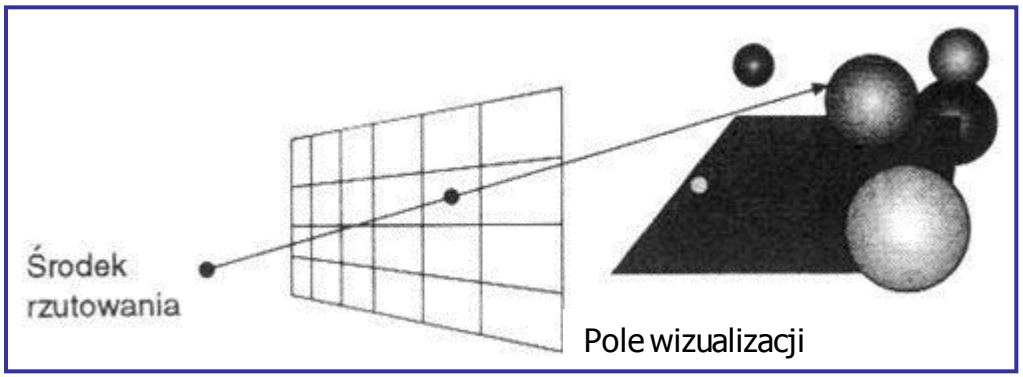
# algorytm bufora głębokości:

Algorytm bufora głębokości jest niewrażliwy na zasłanianie cykliczne ani przecinanie obiektów. Nie wymaga żadnych dodatkowych struktur danych ani operacji wstępnych.

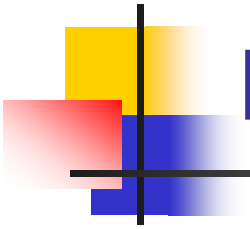
Przyjmuje się, że jest to algorytm o złożoności O(n) ze względu na liczbę wielokątów.

Algorytm ma tylko jedną wadą – potrzebuje pamięci o rozmiarze obrazu pozwalającej zapisać głębokość dla każdego piksela.
Prostota algorytmu sprzyja również implementacji sprzętowej. Stacje graficzne i większość dobrych kart graficznych ma dzisiaj możliwość sprzętowej realizacji bufora głębokości.

# metoda śledzenia promieni:




**Metoda śledzenia promieni** określa widoczność powierzchni na zasadzie śledzenia umownych promieni światła od oka obserwatora do obiektów sceny. Jest wybierany środek rzutowania (oko obserwatora) oraz pole wizualizacji na dowolnej rzutni. Można sobie wyobrazić, że pole wizualizacji jest podzielone na regularną siatkę (o wymaganej rozdzielczości), której elementy odpowiadają pikselom. Następnie dla każdego piksela w polu wizualizacji, ze środka rzutowania jest wypuszczany promień przez środek piksela w kierunku sceny. Barwa piksela jest ustawiana zgodnie z barwą obiektu dla najbliższego punktu przecięcia z powierzchnią obiektu na scenie.



# Literatura pomocnicza:

http://wazniak.mimuw.edu.pl/index.php?title=Grafika_komputerowa_i_wizualizacja (kurs autorstwa Dariusza Sawickiego):
- Moduł 3: Podstawowe operacje rastrowe
- Moduł 4: Przekształcenia geometryczne
- Moduł 5: Reprezentacja przestrzeni trójwymiarowej na płaszczyźnie
- Moduł 7: Eliminacja powierzchni zasłoniętych

Foley J., Van Dam A., Feiner S., Hughes J., Philips R.: Wprowadzenie do grafiki komputerowej. WNT 1995

Berg M., Kreveld M., Overmars M., Schwarzkop O.: Geometria obliczeniowa, algorytmy i zastosowania, WNT 2007

c. d. n.