Wstęp do grafiki komputerowej

Podstawy matematyczne, transformacje, rzutowanie

Tematyka wykładu

- Z lekka matematyczne podejście do
  - Potoku graficznego (wspomnianego pobieżnie na pierwszym wykładzie)
  - Przypomnienie przestrzeni wektorowej i afinicznej
  - Macierzowa reprezentacja transformacji geometrycznych i rzutowania.
- Próba wyjaśnienienia do czego jest nam to potrzebne.

Potok graficzny (Graphics Pipeline)

- Modelowanie. Matematyczny opis obiektów geometrycznych.

Co umożliwia nam Three.js?

- … w zakresie transformacji

- Przykłady spotykamy już na pierwszych zajęciach. Są one dość intuicyjne.
- Możemy wykonać cztery podstawowe operacje na obiekcie:

  własności

  - `position`

  funkcje

  - `rotation`
  - `scale`
  - `translateX()/translateY()/translateZ()`

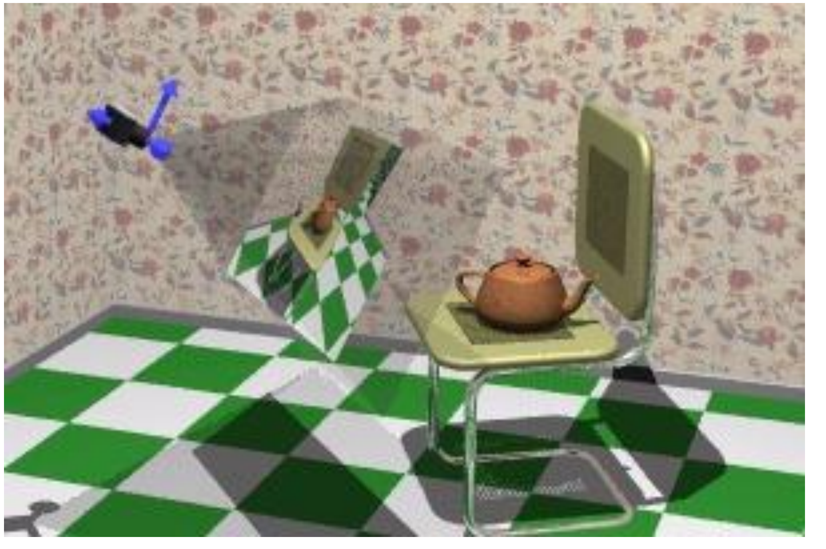
Transformacje na obiektach w Three.js

- Np.

```
cube.rotation.x = -0.5*Math.PI;
cube.position.x = 15;
cube.position.y = 0;
cube.position.z = 0;
cube.scale.x = 2;
cube.scale.y = 3;
cube.translateX(5);
```

Transformacje w Three.js

- Do tego możemy dodać funkcję `lookAt();` W różnych formach występuje ona we wszystkich wariantach OpenGL



- W Three,js mamy np. `camera.lookAt(new Vector3(x,y,z));` gdzie `camera` jest zwykle obiektem z klasy `PerspectiveCamera`

Geometria wektorowa

Po co nam potrzebny jest opis wektorowy?

- Istotna część potoku graficznego dotyczy liniowych przekształceń geometrycznych

Proste i płaszczyzny

- Wektorowa reprezentacja prostej przechodzącej przez punkty P_0 i P_1

$$P = P_0 + t\vec{v}$$

gdzie
$$\vec{v} = P_1 - P_0$$

- Jeśli $\vec{n}$ jest wektorem prostopadłym do prostej, to dla każdego punktu P na prostej zachodzi $\vec{n} \cdot (P - P_0) = 0$

- Można powiedzieć, że rozwiązanie powyższego równania wyznacza prostą

Wektorowa reprezentacja płaszczyzny.

- Płaszczyzna jest wyznaczona jednoznacznie przez trzy niewspółliniowe punkty, np. P_0, P_1, P_2 .

- Jeśli $\vec{v}_1 = P_1 - P_0$ i $\vec{v}_2 = P_2 - P_0$ to wektor normalny do płaszczyzny można wyznaczyć jako iloczyn wektorowy $\vec{n} = \vec{v}_1 \times \vec{v}_2$
- Równanie płaszczyzny: $\vec{n} \cdot (P - P_0) = 0$

Transformacje

Transformacje

- Najogólniej transformacja T jest funkcją, która odwzorowuje punkt A w inny punkt $T(A)$
- Transformacje dokonują się w przestrzeni wektorowej i stowarzyszonej z nią przestrzeni afinicznej punktów.

Rodzaje transformacji

- Transformacje odwracalne (invertible, reversible) i nieodwracalane (irreversible) – łatwo wymyślić przykłady
- Transformacje izometryczne (przesunięcia, obroty)
- Transformacje skalujące
- Transformacje zachowujące współliniowość punktów – czyli transformacje liniowe

Transformacje liniowe

- Jak wiemy takie transformacje spełniają następujące warunki:

$$T(\vec{u}+\vec{v}) = T(\vec{u}) + T(\vec{v})$$

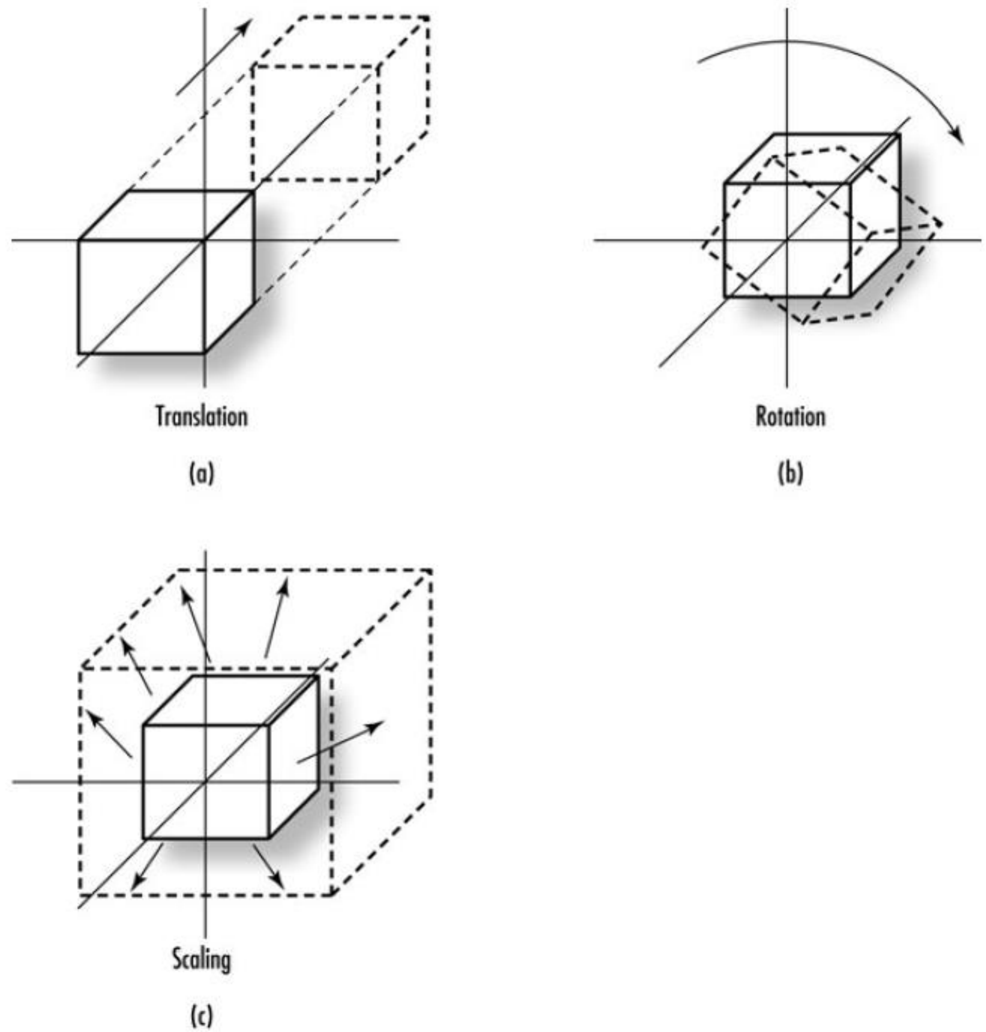
$$T(a\vec{v}) = aT(\vec{v})$$

gdzie a jest skalarem.

- Liniowa transformacja jest realizowana jako operacja macierzowa:

$$T(\vec{v}) = M\vec{v}$$

Przypomnijmy podstawowe transformacje obiektów



Transformacje…

- Teraz chcemy się im przyjrzeć dokładniej
- Zacznijmy od najprostszego… skalowania… w 2D i 3D

$$\begin{cases} x' = s_x \cdot x \\ y' = s_y \cdot y \end{cases}$$

$$\begin{cases} x' = s_x \cdot x \\ y' = s_y \cdot y \\ z' = s_z \cdot z \end{cases}$$

- W grafice do takich przekształceń używamy macierzy

Skalowanie w zapisie macierzowym

$$\begin{cases} x' = s_x \cdot x \\ y' = s_y \cdot y \end{cases} \rightarrow \qquad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x^{-1} & 0 \\ 0 & s_y^{-1} \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}$$

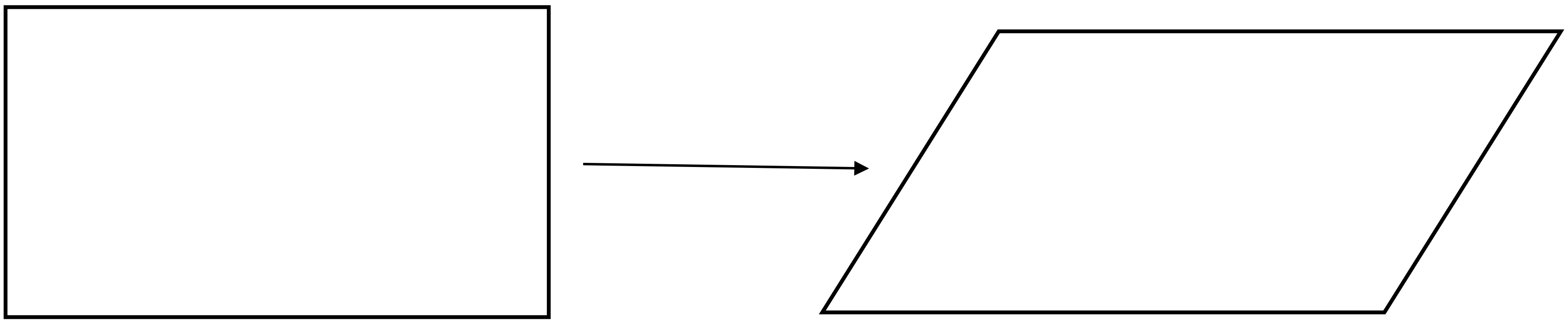
Pod warunkiem, że
$\det(S) \neq 0$

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Ścinanie w 2D

$$SH = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \qquad SH^{-1} = \begin{bmatrix} 1 & -a \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + ay \\ y \end{bmatrix}$$



Jak będzie wyglądać ścinanie i macierz ścinania w 3D?

Obroty

Obroty w 2D są proste, zwłaszcza wokół początku układu współrzędnych

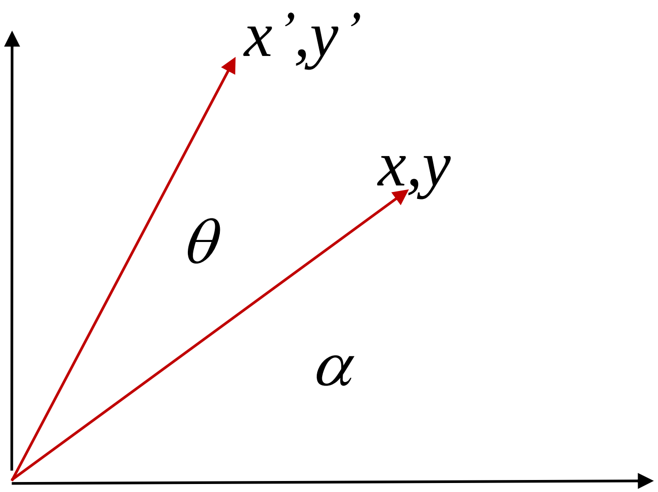
$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Można je szybko wyprowadzić

$$x = \vec{r}\cos\alpha, \quad y = \vec{r}\sin\alpha$$

$$x' = \vec{r}\cos(\alpha+\theta)$$

$$y' = \vec{r}\sin(\alpha+\theta)$$



Obroty w 3D są nieco bardziej złożone

Obroty w 3D

- Tradycyjnie definiuje się trzy macierze do trzech obrotów wokół osi *X*, *Y* i *Z*.

$$R_Z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Macierz dla dodatniego obrotu od osi X do Y

$$R_Y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Macierz dla obrotu od osi Z do X

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Macierz dla dodatniego obrotu od osi Y do Z

Translacja – problem do rozwiązania

- Translacja jest przekształceniem afinicznym, które łączy przestrzeń wektorową z punktami.

$$T(P) = MP + \vec{Q}$$

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z + T_z \end{aligned} \qquad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

- Chcemy się pozbyć się wyrazów wolnych i doprowadzić przekształcenie do postaci $T(P) = MP$

Bardziej ogólne przekształcenie

- Możemy wprowadzić najogólniejsze przekształcenie afiniczne w 3D… choć nie będziemy go efektywnie wykorzystywać

$$x' = a_x x + b_x y + c_x z + d_x$$
$$y' = a_y x + b_y y + c_y z + d_y$$
$$z' = a_z x + b_z y + c_z z + d_z$$

lub

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}$$

Współrzędne jednorodne

- Potrzebujemy dodać czwartą współrzędną ($w=1$). To dość powszechna metoda np. w algebrze liniowej do pozbywania się wolnych wyrazów.
- Macierze 4x4 w przestrzeni 3D – najpopularniejsze w grafice komputerowej

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- w może wyglądać jedynie jako zmienna wypełniająca – ale może też odgrywać dodatkową rolę

Rozszerzenie wymiarów macierzy

- Macierz 3x3 reprezentuje przestrzeń Euklidesa
- Macierz 4x4 reprezentuje przestrzeń rzutowania
- Układ równań reprezentujących proste równoległe nie ma rozwiązania w przestrzeni Euklidesa
- Może mieć natomiast rozwiązanie w przestrzeni rzutowania (dlaczego?)

Przejście między przestrzeniami

- Przejście między obiema przestrzeniami jest proste:

$$h(x, y, z, w) = v(x / w, y / w, z / w)$$

$$v(x, y, z) = h(x, y, z, 1)$$

Translacja modelu we współrzędnych jednorodnych w 3D

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$TP = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \\ z + T_z \\ w \end{bmatrix}$$

Skalowanie modelu we współrzędnych jednorodnych w 3D

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierz skalowania

Obroty modelu we współrzędnych jednorodnych w 3D

:

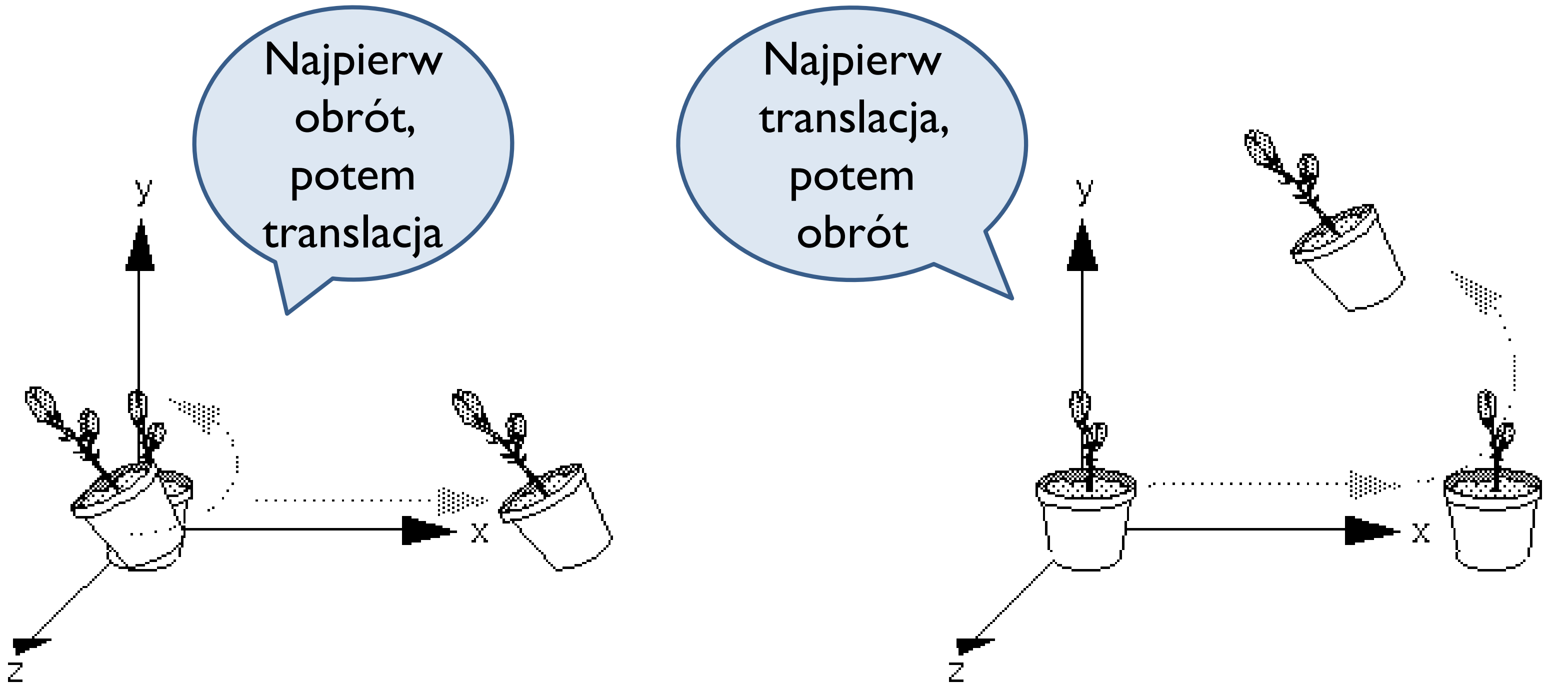
$$\mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_y} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_z} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierze obrotów

Uwaga na marginesie:
Jak myśleć o transformacjach?



Kolejność transformacji zwykle ma znaczenie! Bywa, że nie ma, np. obroty 2D wokół początku układu współrzędnych są przemienne. Ale obroty w 3D już nie.

Składanie transformacji

- Składamy tak jak funkcje

$$p' = S(p) \qquad p'' = R(p')$$

$$p''' = R(S(p)) = RS(p)$$

- W interpretacji macierzowej RS jest również macierzą
- $RS \neq SR$

Transformacje można składać…

- Złożenie transformacji przekłada się na mnożenie macierzy transformacji. Np. obrót o 90^{o} wokół osi x i translacja o 10 jednostek w głąb wzdłuż osi z:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90^\circ) & -\sin(90^\circ) & 0 \\ 0 & \sin(90^\circ) & \cos(90^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Spójrzmy jeszcze raz na translację…

- Wynik translacji:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x + 10w \\ y + 10w \\ z + 10w \\ w \end{bmatrix}$$

- w jest zazwyczaj ustawiane na 1, można jednak nadać mu inną wartość; wtedy skala translacji jest zmieniona.

Składanie transformacji

- Przykładem może być wykonanie obrotu wokół dowolnej prostej przechodzącej przez początek układu współrzędnych.
- W zestawie standardowych transformacji mamy macierze obrotów wokół każdej z osi.
- Przykładowe postępowanie jest opisane na następnym slajdzie:

Składanie transformacji c.d.

- Przykładowe postępowanie może być opisane formułą:

$$\boldsymbol{p}' = R_{y\theta}^{-1} R_{x\varphi}^{-1} R_{z\alpha} R_{x\varphi} R_{y\theta} \boldsymbol{p}$$

obracamy prostą $\boldsymbol{p}$ wokół osi y o kąt θ, tak żeby znalazła się w płaszczyźnie yz, następnie wokół osi x o kąt φ, tak żeby pokryła się z osią z, a następnie wykonujemy właściwy obrót już wokół osi z o kąt α. Na koniec wycofujemy się do układu początkowego wykonując dwie transformacje odwrotne.

- Jak widać powszechne jest wykonywanie transformacji odwrotnych do danych

Macierze odwrotne w składaniu przekształceń.

- Transformacje odwrotne są realizowane poprzez mnożenie przez macierze odwrotne – które trzeba policzyć.
- Typowa sekwencja obudowująca właściwą transformację (która poniżej jest pominięta), wymagająca odwrócenia trzech transformacji jest dość oczywista:

$$M = M_3 M_2 M_1$$

$$M^{-1} = M_1^{-1} M_2^{-1} M_3^{-1}$$

$$M^{-1} M = M_1^{-1}\left(M_2^{-1}\left(M_3^{-1} M_3\right) M_2\right) M_1$$

Tu wstawiamy właściwą transformację

Jak policzyć macierze odwrotne?

Odwracanie macierzy

- Jak je policzyć?

- Ręcznie według wzorów
- Z wykorzystaniem funkcji w dostępnych bibliotekach np. `inv()` lub jakiejś podobnej
- Wykorzystując fakt, że macierze transformacji są często ortogonalne (proszę przy okazji sprawdzić które). Prypomnienie na następnym slajdzie.

Macierz ortogonalna

Definicja

Macierz A jest ortogonalna wtedy i tylko wtedy gdy

$$A^T A = I$$

gdzie A^T jest macierzą transponowaną do A

Np. macierz rotacji jest ortogonalna

$$R^T R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} =$$

$$\begin{bmatrix} \cos^2\theta + \sin^2\theta & -\cos\theta\sin\theta + \sin\theta\cos\theta \\ -\sin\theta\cos\theta + \cos\theta\sin\theta & \sin^2\theta + \cos^2\theta \end{bmatrix}$$

Macierz ortogonalna

Z poprzedniego twierdzenia wynika ważna i pożyteczna własność:

$$A^{-1} = A^T$$

Znacznie szybsza jest transpozycja macierzy niż jej odwracanie - i to jest główny wniosek z tego slajdu.

Orientacja

Wprowadzenie

- Kluczowym elementem w grafice 3D jest
  - Scena z obiektami
  - Kamera umieszczona w określonym punkcie i patrząca na określony punkt na scenie

- Zadaniem do wykonania jest przejrzyście określić zmianę współrzędnych gdy poruszamy kamerą albo obiektami na scenie.

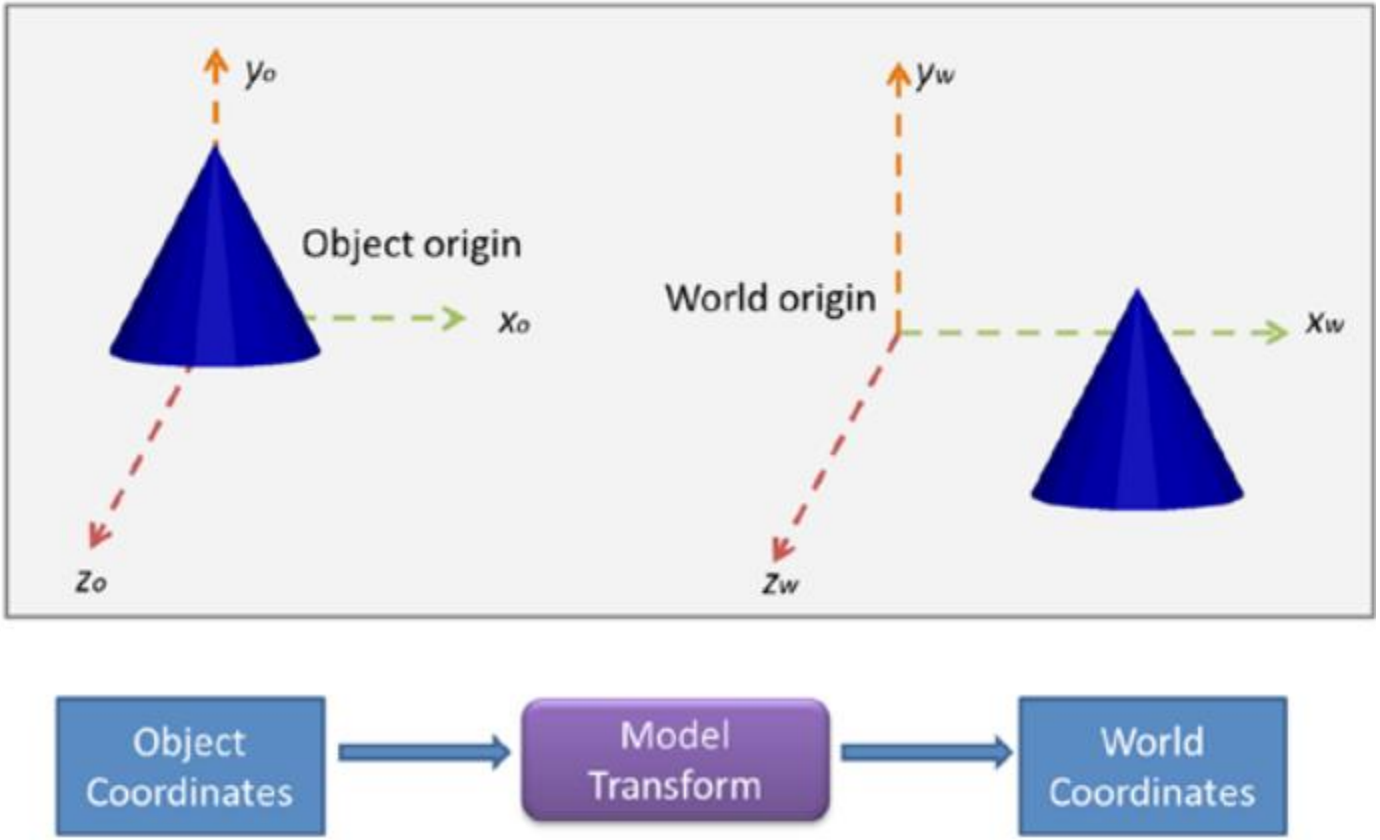
- Zwykle zmiany współrzędnych rozbijamy na etapy (niekoniecznie wszystkie etapy są niezbędne)

Potok transformacji (potok graficzny bez rasteryzacji i renderowania)

Tradycyjny podział:

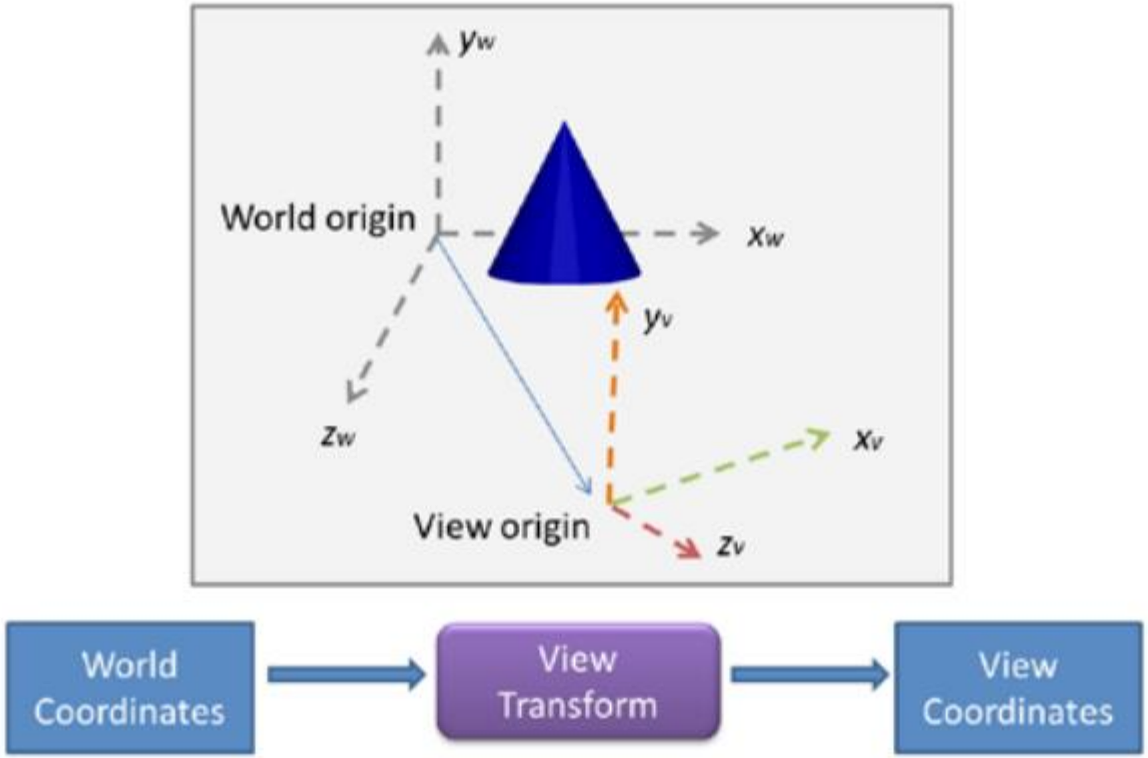
1. Object coordinates
2. World coordinates
3. View (or eye or camera) coordinates (współrzędne obserwatora)
4. Clip coordinates
5. Normalized device coordinates
6. Viewport (canvas or window or screen) coordinates

Elementy potoku transformacji (1)



Transformacja współrzędnych obiektu (modelu)

Elementy potoku transformacji (2)

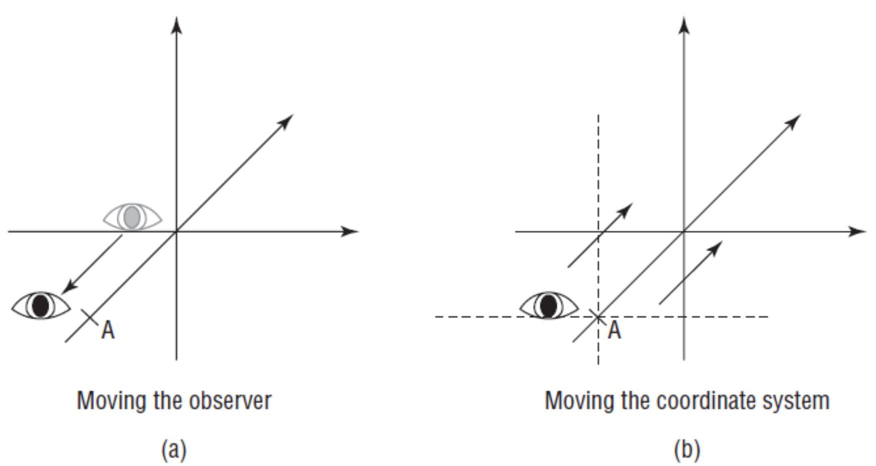


Transformacja widoku – przesuwamy/obracamy kamerę razem układem współrzędnych. Oczywiście wpółrzędne „nieruchomego" modelu też się zmieniają w nowym układzie współrzędnych.

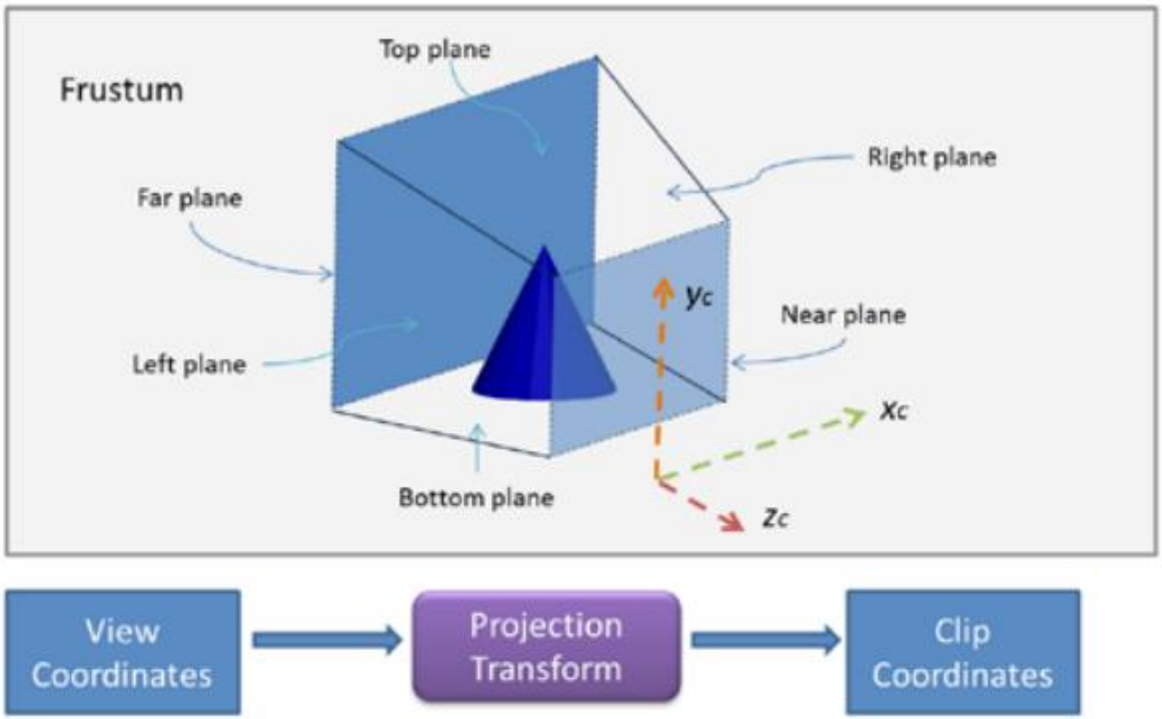
Dygresja o transformacji widoku i transformacji modelu

Transformowanie widoku (czyli położenia obserwatora w układzie współrzędnych) i transformowanie modelu dopełniają się.

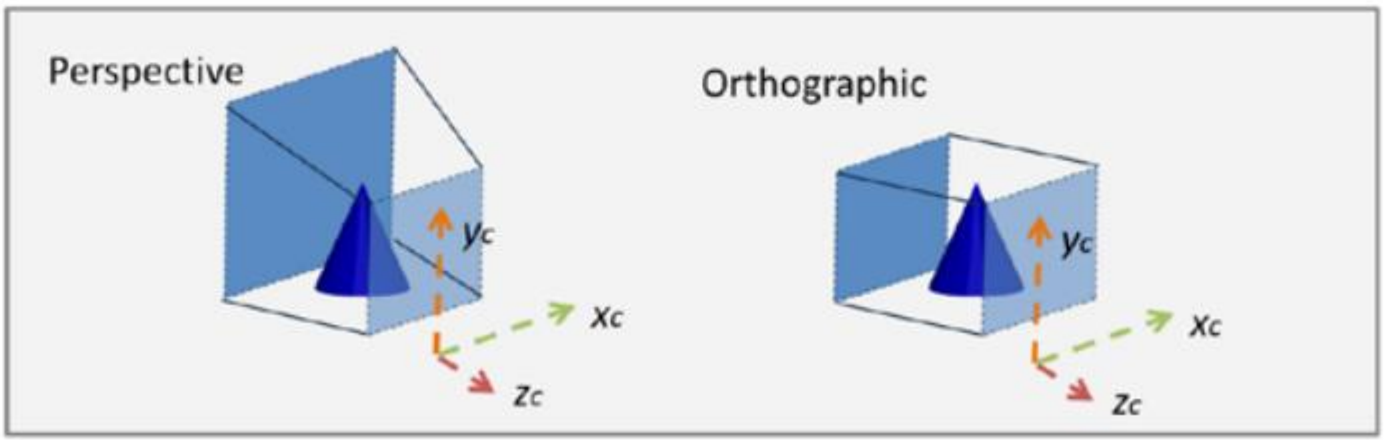
Możemy uzyskać identyczny widok sceny np. przesuwając obserwatora w prawo lub obiekt na scenie w lewo:



View space → Clip space



View space → Clip space



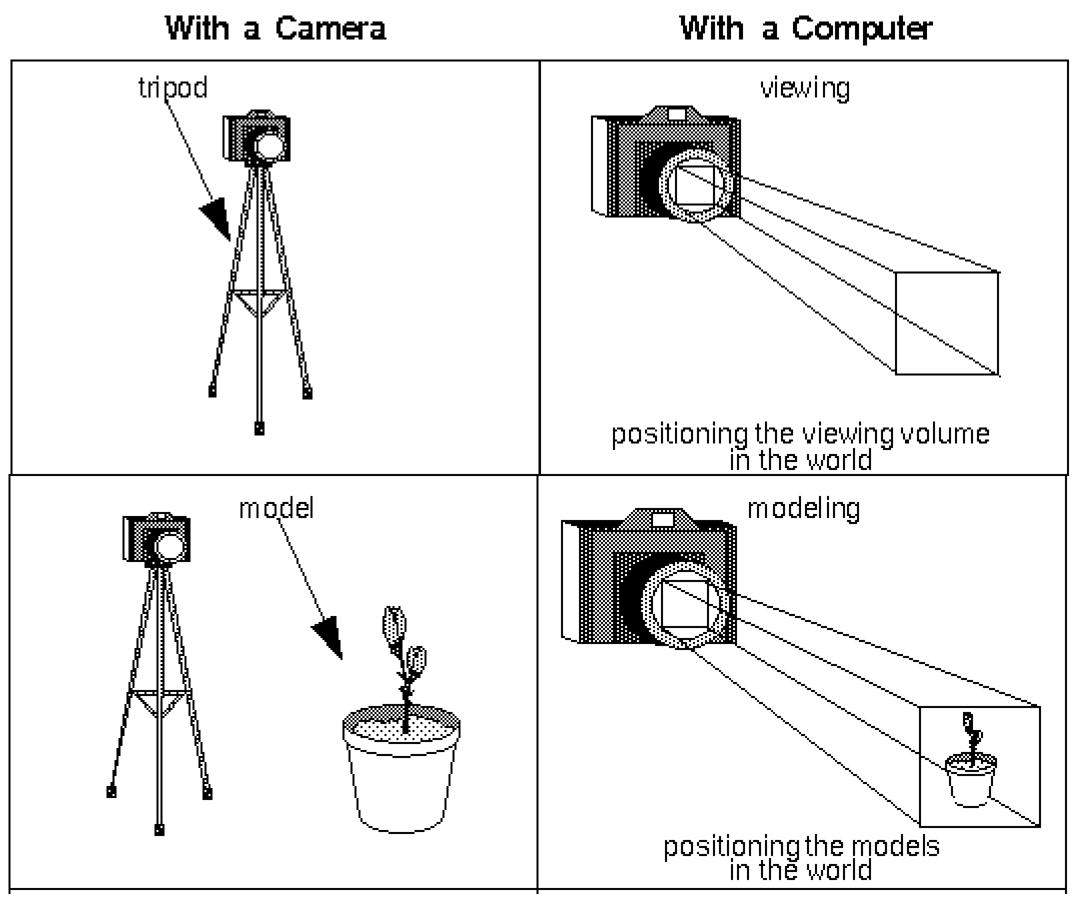
Transformacje geometryczne - podsumowanie

Całość przekształceń geometrycznych składa się z czterech elementów

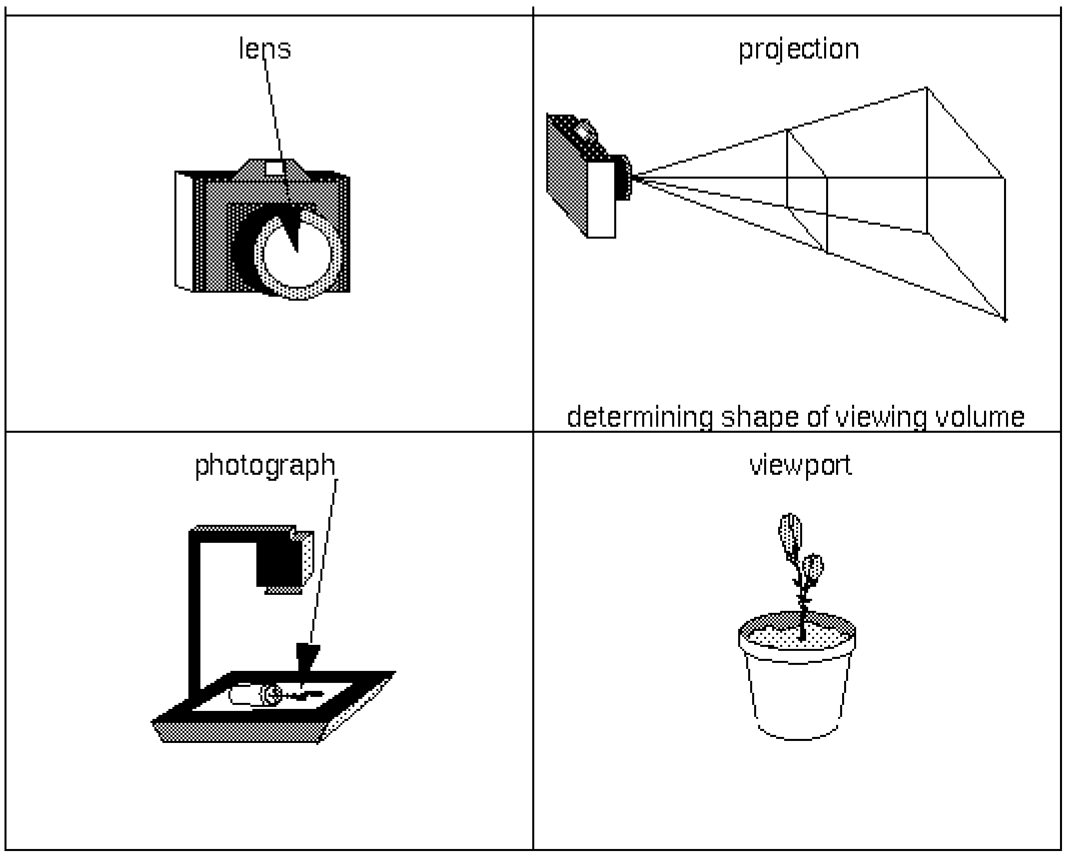
- **Viewing - Transformacje widoku** określają położenie kamery
- **Modeling - Transformacje modelowania** przemieszczają obiekty na scenie

  (Modelview – określa wzajemne relacje **Viewing** i **Modeling)**
- **Projection - Transformacje rzutowania** definiują bryłę widoku i płaszczyzny obcięcia
- **Viewport - Mapowanie na okno**

Analogia z aparatem fotograficznym



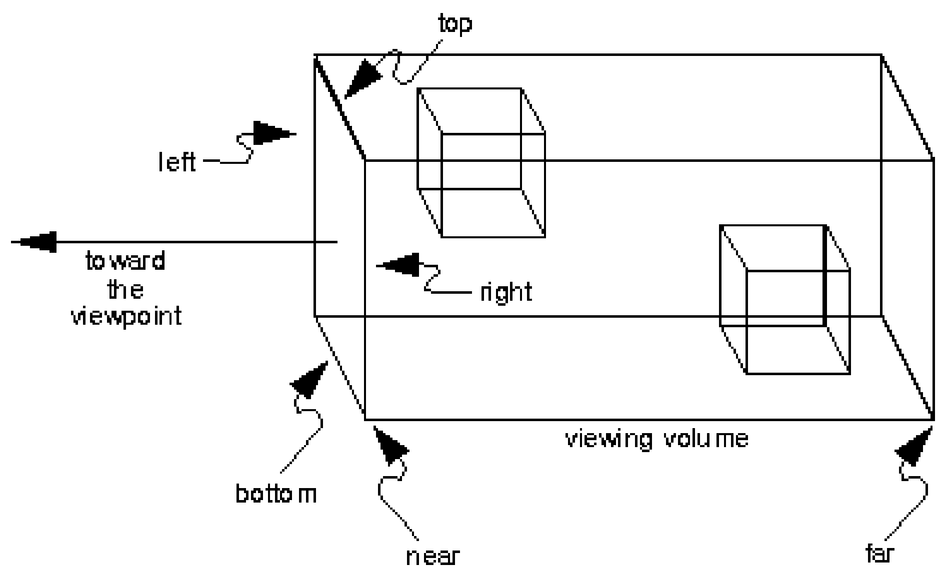
Analogia, c.d.


lens
projection
determining shape of viewing volume
photograph
viewport

Rzutowanie ortogonalne (ang. *orthographic*)

OrthographicCamera(*left*, *right*, *bottom*, *top*, *near*, *far*)

```
var camera = new THREE.OrthographicCamera
(left, right, bottom, top, near,  far);
scene.add( camera );
```



Jak patrzy obserwator?

Rzutowanie ortogonalne c.d

- Obszar rzutowania wzdłuż osi Z zawarty jest między wartościami –*near* i –*far*.
- Jeśli pominiemy wywołanie **`ortho`** to domyślna postać jest następująca:
  `ortho``(-1., 1.,-1. ,1.,-1.,1.);`
- Obserwator jest ustawiony w położeniu (0,0.+∞) i patrzy w kierunku ujemnych wartości osi *z*

Macierz rzutowania ortogonalnego

$$O = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{near - far} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

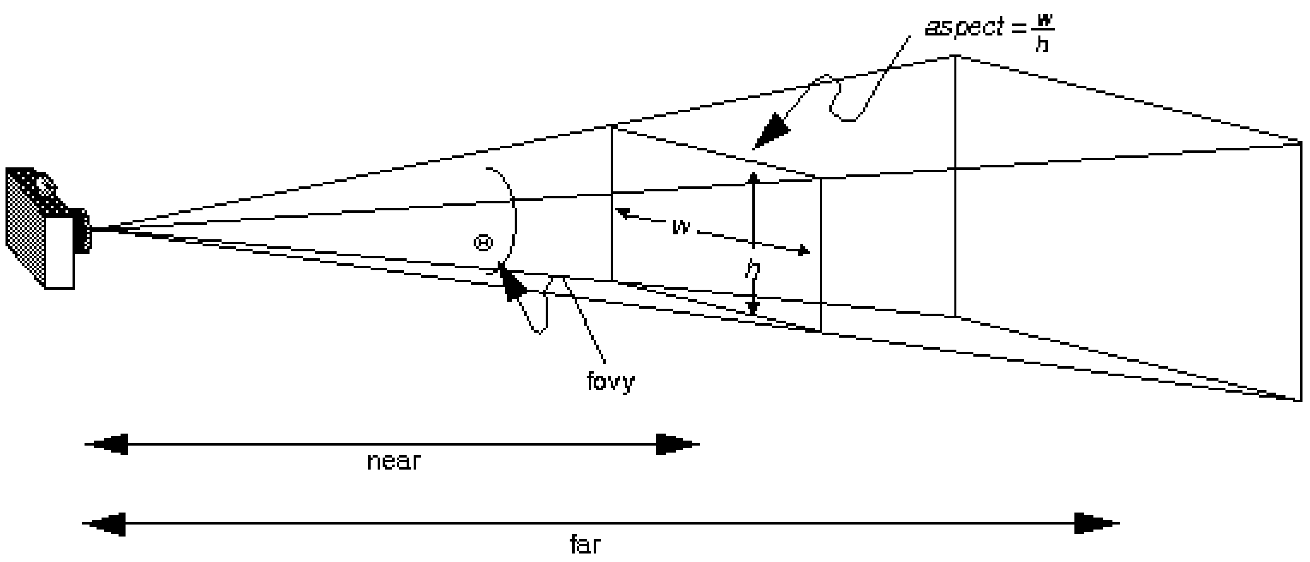
Rzut perspektywiczny

perspectiveCamera(*fovy*, *aspect*, *near*, *far*)

```
var camera = new THREE.PerspectiveCamera(
45, width / height, 1, 1000 );
scene.add( camera );
```

fovy – kąt widzenia w płaszczyźnie *yz*

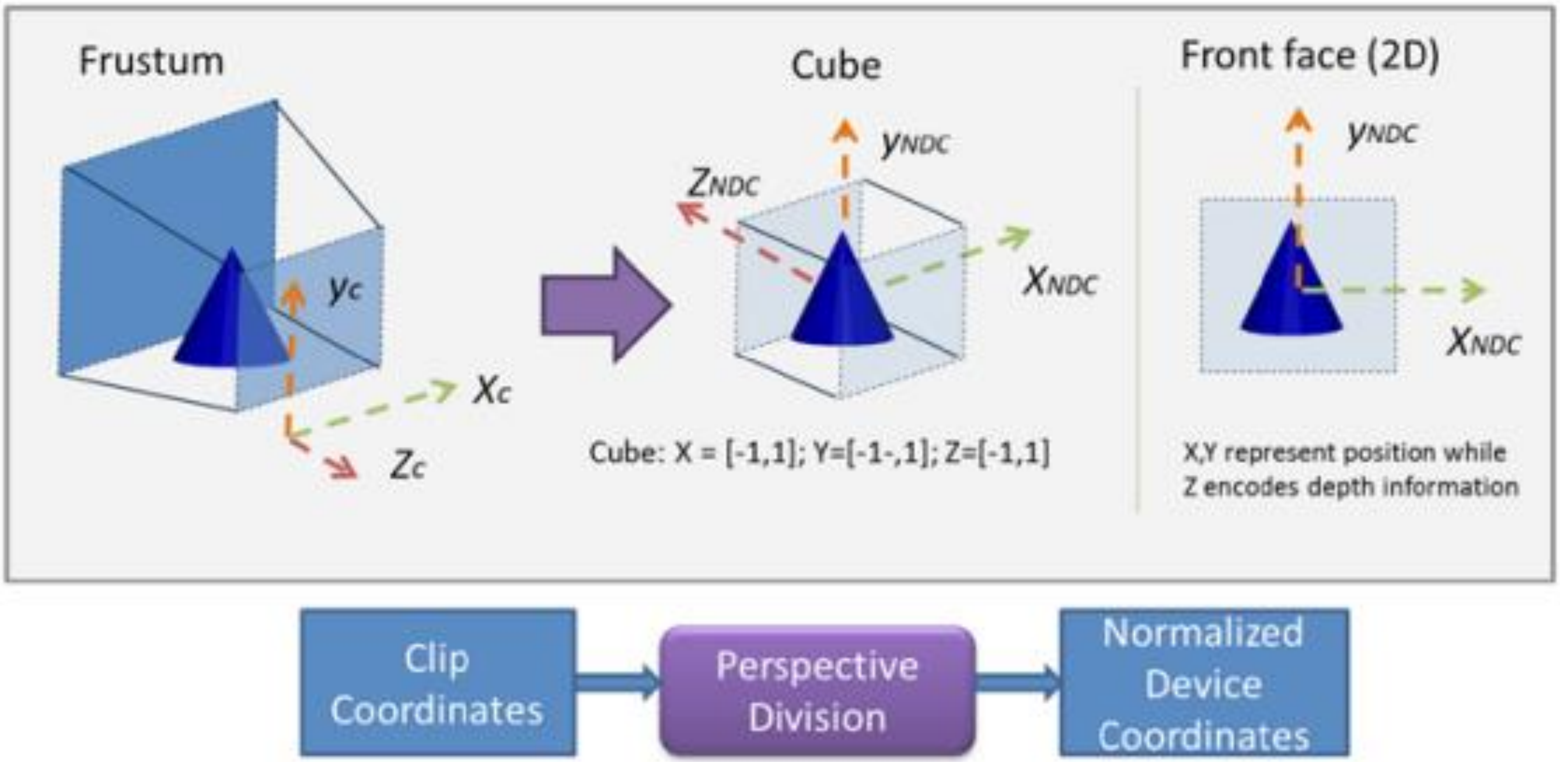
aspect – stosunek szerokości do wysokości pola widzenia

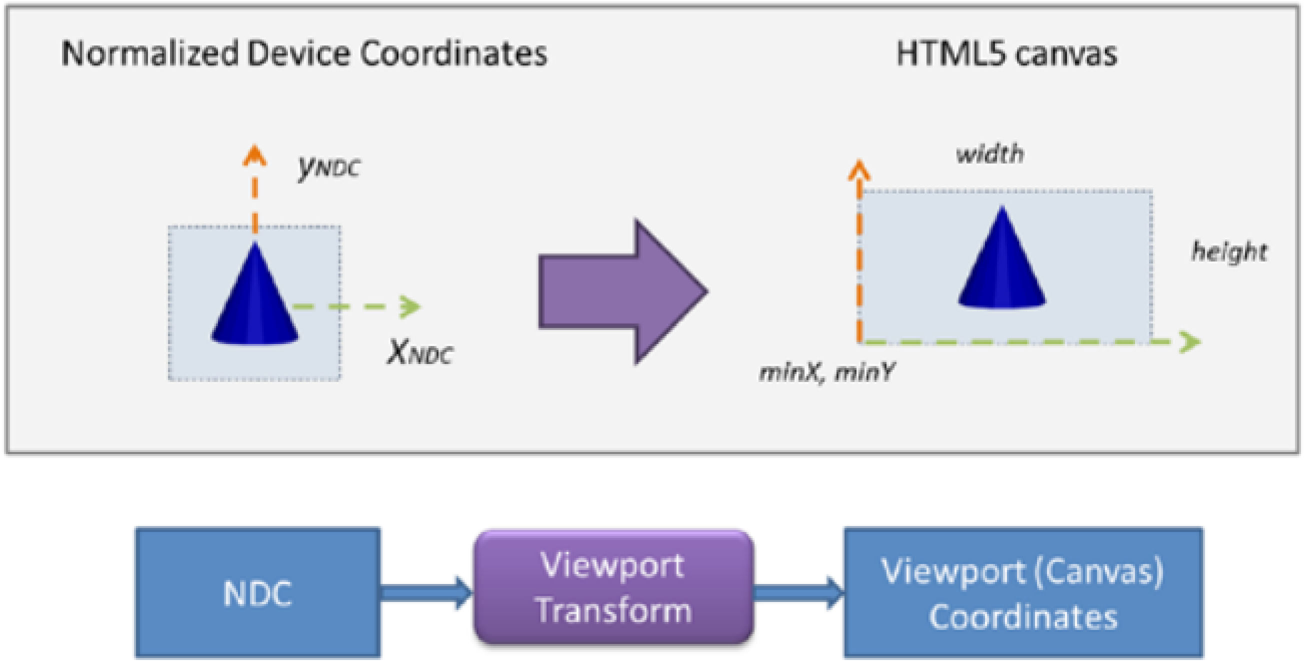


Macierz rzutowania perspektywicznego

$$P = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & \frac{2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Clip space – NDC space





Normalized Device Coordinates
HTML5 canvas
yNDC
XNDC
width
height
minX, minY
NDC
Viewport Transform
Viewport (Canvas) Coordinates

Dwa podejścia do rzutowania i transformacji

- Podejście pierwsze:
  w programie wykonujemy jawnie operacje na współrzędnych, korzystając zwykle z operacji macierzowych
- Podejście drugie:
  ukrywamy operacje macierzowe pod funkcjami. To podejście – choć ukrywa istotę obliczeń – jest jednak łatwiejsze i wygodniejsze w użyciu.
- Wydaje się, że ma sens używanie obu podejść.

A co z WebGL/Three.js?

- W Three.js - wiadomo
- WebGL core właściwie nie dostarcza własnych funkcji.
- W shaderach pisanych w GLSL można korzystać z operacji macierzowych i wektorowych oraz funkcji GLSL.
- Dla Javascriptu istnieją różne biblioteki wspomagające operacje macierzowe. Z najpopularniejszych można wymienić:
  - Sylvester (http://sylvester.jcoglan.com/)
  - WebGL-mjs (https://code.google.com/p/webgl-mjs/)
  - glMatrix (http://glmatrix.net/)

W uzupełnieniu:
Rzutowanie na fragment okna – `gl.viewport`

Standardowo rzutowanie odbywa się na całe okno.

Można powiedzieć, że domyślnie jest wywoływana funkcja
`gl.viewport(0,0,w,h);`
gdzie **`w`** i **`h`** są szerokością i wysokością okna

Można za jej pomocą ograniczyć obszar rzutowania jedynie na fragment okna

W OpenGL odpowiednikiem tej funkcji jest **`glViewport`**