



Politechnika  
Wrocławska

# Grafika komputerowa i komunikacja człowiek-komputer

Laboratorium nr 4

Interakcja z użytkownikiem, transformacje wierzchołków

Szymon Datko

[szymon.datko@pwr.edu.pl](mailto:szymon.datko@pwr.edu.pl)

Wydział Informatyki i Telekomunikacji,  
Politechnika Wrocławska

semestr zimowy 2022/2023



# Cel ćwiczenia

1. Zapoznanie się z mechanizmem obsługi urządzeń peryferyjnych.
2. Zrozumienie zasady działania przekształceń wierzchołków.
3. Implementacja transformacji w reakcji na działania użytkownika.

# Obsługa zdarzeń z klawiatury/myszy

- ▶ Zagadnienie to **nie jest** częścią specyfikacji OpenGL.
- ▶ W naszym programie całą obsługą zdarzeń zajmuje się biblioteka **GLFW**.
- ▶ Zasadniczo chodzi o to, aby:
  - zdefiniować funkcję do wykonania w reakcji na zdarzenie,
    - musi ona przyjmować odpowiednią liczbę i rodzaj argumentów,
  - uaktywnić ją – podpiąć do określonego rodzaju zdarzenia,
  - stosowany jest tu mechanizm wywołań zwrotnych (ang. *callback*).
- ▶ Funkcja bezpośredniej reakcji na zdarzenie powinna być możliwie szybka,
  - często rejestruje się same akcje, a obsługę realizuje osobny proces.
- ▶ Więcej informacji: [https://www.glfw.org/docs/latest/input\\_guide.html](https://www.glfw.org/docs/latest/input_guide.html).

# Podstawowe rodzaje przekształceń

- ▶ Afiniczne<sup>1</sup>:
  - ▶ identyczność,
  - ▶ skalowanie,
  - ▶ translacja,
  - ▶ obrót.
  
- ▶ Nieafiniczne:
  - ▶ rzutowanie ortogonalne,
  - ▶ rzutowanie perspektywiczne.

Przekształcenie obiektu = przekształcenie wszystkich jego wierzchołków.

---

<sup>1</sup> czyli takie, które zachowują (w ogólności) proporcje odległości między punktami na tej samej linii przed i po wykonaniu transformacji, ale nie muszą zachować położenia punktu początkowego; może być zareprezentowane jako iloczyn macierzy  $n \times n$  i wektora  $n \times 1$  z dodatkowym przesunięciem o wektor;  $n$  - wymiar przestrzeni.

# Współrzędne jednorodne

- ▶ Podstawowym pojęciem w grafice komputerowej **wierzchołek**. Jest to punkt z określonym położeniem w przestrzeni 3D.
- ▶ W **reprezentacji jednorodnej** do zapisu położenia stosuje się wektory o liczbie elementów większej niż wymiar przestrzeni,

$$\begin{bmatrix} x & y & z & w \end{bmatrix} \leftrightarrow \left( \frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right).$$

- ▶ Typowo ostatnia składowa  $w$  dla położenia ma wartość 1.
- ▶ Zastosowanie takiej reprezentacji pozwala na bardzo łatwą realizację podstawowych przekształceń geometrycznych.

# Macierz przekształceń – identyczność

Macierz  $\mathbf{I}$  jest także nazywana macierzą jednostkową.

$$\mathbf{I} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Z reguły jest to domyślna macierz, od której rozpoczynamy dalsze obliczenia.

► Przykład obliczeń:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x & y & z & 1.0 \end{bmatrix}$$

# Macierz przekształceń – skalowanie

$$\mathbf{S} = \begin{bmatrix} S_x & 0.0 & 0.0 & 0.0 \\ 0.0 & S_y & 0.0 & 0.0 \\ 0.0 & 0.0 & S_z & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- ▶ Gdy  $S_x = S_y = S_z$  to mówimy o skalowaniu jednorodnym.
- ▶ Przykład obliczeń:

$$\begin{bmatrix} S_x & 0.0 & 0.0 & 0.0 \\ 0.0 & S_y & 0.0 & 0.0 \\ 0.0 & 0.0 & S_z & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} S_x \cdot x & S_y \cdot y & S_z \cdot z & 1.0 \end{bmatrix}$$

# Macierz przekształceń – translacja

$$\mathbf{T} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & T_x \\ 0.0 & 1.0 & 0.0 & T_y \\ 0.0 & 0.0 & 1.0 & T_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- ▶ Reprezentacja jednorodna punktu pozwala wyrazić operację przez macierz.
- ▶ Przykład obliczeń:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & T_x \\ 0.0 & 1.0 & 0.0 & T_y \\ 0.0 & 0.0 & 1.0 & T_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x + T_x & y + T_y & z + T_z & 1.0 \end{bmatrix}$$



## Macierz przekształceń – obrót wokół osi x

$$\mathbf{R}_{x,\gamma} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & \cos \gamma & -\sin \gamma & 0.0 \\ 0.0 & \sin \gamma & \cos \gamma & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

- ▶ Współrzędne w osi wokół której następuje obrót nie ulegają zmianie!
- ▶ Przykład obliczeń:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & \cos \frac{\pi}{2} & -\sin \frac{\pi}{2} & 0.0 \\ 0.0 & \sin \frac{\pi}{2} & \cos \frac{\pi}{2} & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x & -z & y & 1.0 \end{bmatrix}$$

# Macierz przekształceń – obrót wokół osi y i z

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos \theta & 0.0 & \sin \theta & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ -\sin \theta & 0.0 & \cos \theta & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\mathbf{R}_{z,\phi} = \begin{bmatrix} \cos \phi & -\sin \phi & 0.0 & 0.0 \\ \sin \phi & \cos \phi & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

## Uwaga!

W zależności od przyjętej definicji układu współrzędnych i kierunku obrotu, wyrazy w przedstawionej macierzy mogą się różnić co do znaku  $+/-$ .

# Macierz przekształceń – obrót wokół wektora

Obrót o kąt skierowany  $\phi$  (reguła prawej dłoni) wokół wektora  $\mathbf{v}$ .

$$\mathbf{v}^T = [v_x \quad v_y \quad v_z \quad 0.0], \quad \|\mathbf{v}\| = 1$$

$$\begin{bmatrix} v_x^2(1 - \cos \phi) + \cos \phi & v_x v_y(1 - \cos \phi) - v_z \sin \phi & v_x v_z(1 - \cos \phi) + v_y \sin \phi & 0.0 \\ v_x v_y(1 - \cos \phi) + v_z \sin \phi & v_y^2(1 - \cos \phi) + \cos \phi & v_y v_z(1 - \cos \phi) - v_x \sin \phi & 0.0 \\ v_x v_z(1 - \cos \phi) - v_y \sin \phi & v_y v_z(1 - \cos \phi) + v_x \sin \phi & v_z^2(1 - \cos \phi) + \cos \phi & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Lewa górna część (3x3) powyższej macierzy może zostać wyrażona w prostszy sposób:

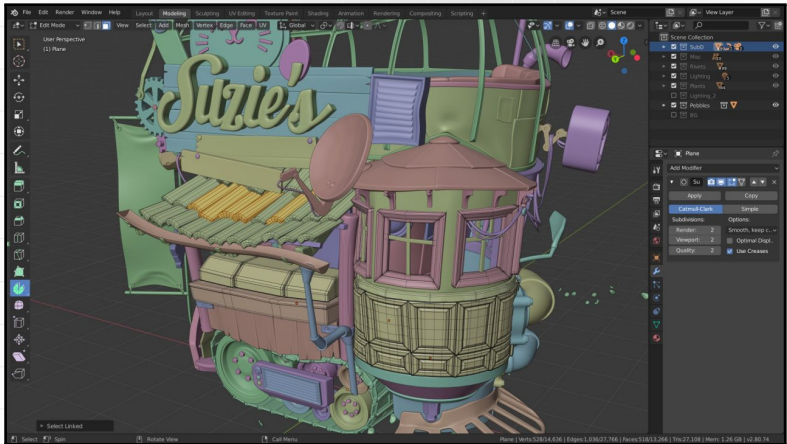
$$\mathbf{v} \mathbf{v}^T (1 - \cos \phi) + \mathbf{I} \cos \phi + \begin{bmatrix} 0.0 & -v_z & v_y \\ v_z & 0.0 & -v_x \\ -v_y & v_x & 0.0 \end{bmatrix} \sin \phi$$

## Uwaga!

Wektor  $\mathbf{v}$  określa oś obrotu, przechodzącą przez początek układu współrzędnych!

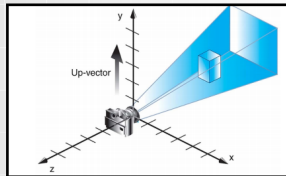
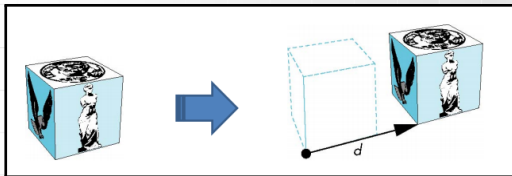
# Kompozycja wirtualnego świata

- ▶ Przedstawione transformacje pozwalają nam na skomponowanie sceny.
  - Można dzięki nim określić rozmieszczenie wszystkich obiektów.
  - Pozostaje określić na co na tej scenie patrzymy – opisać kamerę...



# Kamera jest kłamstwem

- ▶ W kontekście karty graficznej nie istnieje coś takiego jak **kamera!**
- ▶ Karta graficzna renderuje zawsze dla tego samego fragmentu przestrzeni.
  - Obszar rysowania obejmuje zakres  $[-1.0; 1.0]$  dla każdej z osi.
  - Jest to tak zwana **bryła widzenia**.
- ▶ Abstrakcja z kamerą na scenie stanowi intuicyjnie wygodny dla nas opis...
  - Łatwiej nam wyobrazić sobie przesunięcie obserwatora, niż świata.
  - Wykorzystujemy **względność ruchu**, aby tę reprezentację osiągnąć!



# Macierz patrzenia

Chcemy opisać położenie kamery jednoznacznie w przestrzeni:

- ▶ współrzędne położenia kamery  $\mathbf{e} = (e_x, e_y, e_z)$ ,
- ▶ współrzędne punktu zainteresowania  $\mathbf{p} = (p_x, p_y, p_z)$ ,
- ▶ wektor orientacji, wskazujący górę dla kamery  $\mathbf{u} = (u_x, u_y, u_z)$ .

Kierunek patrzenia można opisać wektorem  $\mathbf{f} = \frac{\mathbf{p} - \mathbf{e}}{\|\mathbf{p} - \mathbf{e}\|}$ ;  $\mathbf{s} = (\mathbf{f} \times \mathbf{u})$ .

Rzut wektora  $\mathbf{u}$  na płaszczyznę prostopadłą do  $\mathbf{f}$  to  $\mathbf{u}' = (\mathbf{f} \times \mathbf{u}) \times \mathbf{f}$ .

$$\mathbf{V}_{\text{lookAt}} = \begin{bmatrix} s_x & u'_x & f_x & -e_x \\ s_y & u'_y & f_y & -e_y \\ s_z & u'_z & f_z & -e_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

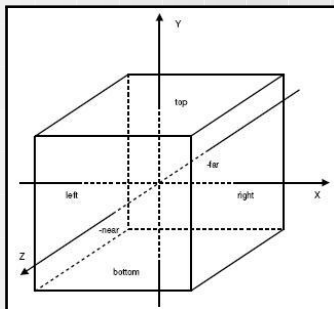
Oznaczenia:  $s$  – side vector,  $u$  – up vector,  $f$  – forward vector,  $e$  – eye position.

**Efektywnie:** przekształcamy cały świat przeciwnie do ruchu, jaki wykonałaby kamera.

# Macierz rzutowania prostopadłego

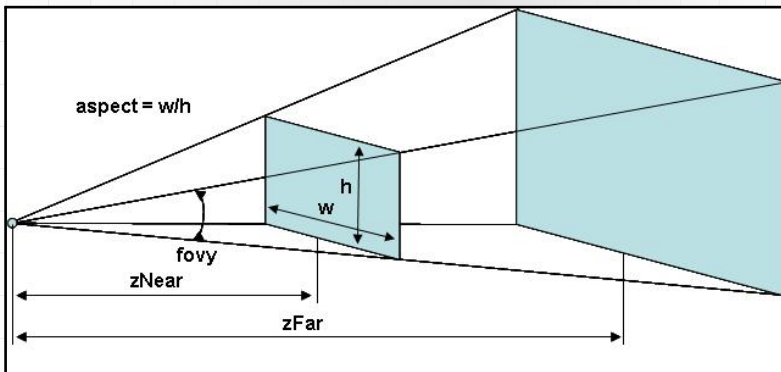
$$P_{ortho} = \begin{bmatrix} \frac{2}{right-left} & 0.0 & 0.0 & -\frac{right+left}{right-left} \\ 0.0 & \frac{2}{top-bottom} & 0.0 & -\frac{top+bottom}{top-bottom} \\ 0.0 & 0.0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Macierz nie opisuje wprost przekształcenia nieafinicznego, a bryłę widzenia!



**Efektywnie:** skalujemy fragment świata, który chcemy zobaczyć, do obszaru rysowania.

# Rzutowanie perspektywiczne



Obszar pomiędzy  $z_{Near}$  a  $z_{Far}$  to bryła widzenia.

Przy rzutowaniu perspektywicznym to tak zwany stożek ścięty (ang. *frustum*).

Pozwala nam zaobserwować głębię (odległość od kamery) na obrazie.

Obiekty bliżej płaszczyzny bliskiej są mniej zmniejszone, niż te przy p. dalekiej.



# Macierz rzutowania perspektywicznego

$$P_{\text{frustum}} = \begin{bmatrix} \frac{2 \cdot \text{near}}{\text{right} - \text{left}} & 0.0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & 0.0 \\ 0.0 & \frac{2 \cdot \text{near}}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0.0 \\ 0.0 & 0.0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -\frac{2 \cdot \text{far} \cdot \text{near}}{\text{far} - \text{near}} \\ 0.0 & 0.0 & -1.0 & 0.0 \end{bmatrix}$$

W szczególności to przekształcenie może być zadane przez:

- ▶ pole widzenia jako kąt  $\text{fovy}$ ,
- ▶ współczynnik proporcji obrazu  $\text{aspect} = \frac{\text{width}}{\text{height}}$ ,
- ▶ odległość bliższej  $\text{near}$  i dalszej  $\text{far}$  przestrzeni przycięcia;
- ▶ wtedy obliczamy:
  - ▶  $\text{top} = \text{near} * \text{tg}(\text{fovy} \cdot \frac{\pi}{360})$ ,
  - ▶  $\text{bottom} = -\text{top}$ ,
  - ▶  $\text{right} = \text{top} * \text{aspect}$ ,
  - ▶  $\text{left} = -\text{right}$ .

# Składanie przekształceń i praca w Legacy OpenGL

- ▶ Chcąc przekształcić wektor, mnożymy go lewostronnie przez macierz.
- ▶ Wynikiem takiej operacji jest nowy wektor, który też można przekształcić.

$$\vec{p}_{\text{wynik}} = \dots \cdot \mathbf{M}_{\text{czwarta}} \cdot \mathbf{M}_{\text{trzecia}} \cdot \mathbf{M}_{\text{druga}} \cdot \mathbf{M}_{\text{pierwsza}} \cdot \vec{p}_{\text{wierzchołka}}$$

- ▶ Macierze można wymnożyć najpierw, aby przyspieszyć obliczenia.
- ▶ Mnożenie macierzy nie jest przemienne, a więc **kolejność ma znaczenie**.
  - Przesunięcie po obrocie to co innego niż obrót po przesunięciu.
- ▶ W Legacy OpenGL zasadniczo prawdziwe jest następujące wyrażenie:

$$\vec{p}_{\text{wyjciowe}} = M_{\text{rzutowania}} \cdot M_{\text{modelu}} \cdot \vec{p}_{\text{wejciowe}}$$

- ▶ Współrzędne wejściowe stanowią argumenty funkcji `glVertex()`.
- ▶ Współrzędne wyjściowe to faktycznie rysowane położenie wierzchołka.
- ▶ Zmianę macierzy transformacji uzyskuje się za pomocą szeregu funkcji...

## Realizacja w Legacy OpenGL (1/2)

- ▶ Identyczność / cofnięcie wszystkich przekształceń:  
`glLoadIdentity()`
- ▶ Skalowanie:  
`glScalef(S_x, S_y, S_z)`
- ▶ Translacja / przesunięcie o wektor:  
`glTranslatef(T_x, T_y, T_z)`
- ▶ Obrót wokół osi X:  
`glRotatef(angle, 1.0, 0.0, 0.0)`
- ▶ Obrót wokół osi Y:  
`glRotatef(angle, 0.0, 1.0, 0.0)`
- ▶ Obrót wokół osi Z:  
`glRotatef(angle, 0.0, 0.0, 1.0)`

## Realizacja w Legacy OpenGL (2/2)

- ▶ Obrót wokół osi wyznaczonej przez wektor  $[V_x, V_y, V_z]$  i punkt  $[0, 0, 0]$ :  
`glRotatef(angle, V_x, V_y, V_z)`
- ▶ Przekształcenie patrzenia / przemieszczenie kamery na scenie:  
`gluLookAt(eyeX, eyeY, eyeZ,  
          centerX, centerY, centerZ,  
          upX, upY, upZ)`
- ▶ Rzutowanie ortogonalne:  
`glOrtho(left, right, bottom, top, zNear, zFar)`
- ▶ Rzutowanie perspektywiczne:  
`gluPerspective(fovy, aspect, zNear, zFar)`
- ▶ Wybór macierzy transformacji ( $M_{\text{rzutowania}}$  lub  $M_{\text{modelu}}$ ) do zmian:  
`glMatrixMode(GL_PROJECTION)` lub `glMatrixMode(GL_MODELVIEW)`
- ▶ Określenie rozmiaru rzutni w pikselach:  
`glViewport(x, y, width, height)`

## Nowości w przykładowym programie (1/3)

- ▶ Wprowadzono szereg zmiennych pomocniczych.

```
1| viewer = [0.0, 0.0, 10.0]
2|
3| theta = 0.0
4| pix2angle = 1.0
5|
6| left_mouse_button_pressed = 0
7| mouse_x_pos_old = 0
8| delta_x = 0
```

- Zmienna `viewer` przechowuje informacje o położeniu obserwatora.
- Zmienna `theta` zawiera wartość kąta obrotu.
- Zmienna `pix2angle` to czynnik skalujący na potrzeby obliczeń,
  - żeby maksymalny ruch myszą odpowiadał obrotowi o  $360^\circ$ ,
  - jej wartość jest definiowana w funkcji `update_viewport()`.
- Zmienna `left_mouse_button_pressed` zawiera stan przycisku myszy.
- Zmienna `mouse_x_pos_old` przechowuje ostatnie położenie w poziomie.
- Zmienna `delta_x` zawiera informację o różnicy położenia myszy.

## Nowości w przykładowym programie (2/3)

- Dodano funkcje związane z obsługą zdarzeń klawiatury i myszy.

```
1| def keyboard_key_callback(window, key, scancode, action, mods):
2|     if key == GLFW_KEY_ESCAPE and action == GLFW_PRESS:
3|         glfwSetWindowShouldClose(window, GLFW_TRUE)
4|
5| def mouse_motion_callback(window, x_pos, y_pos):
6|     global delta_x
7|     global mouse_x_pos_old
8|
9|     delta_x = x_pos - mouse_x_pos_old
10|    mouse_x_pos_old = x_pos
11|
12| def mouse_button_callback(window, button, action, mods):
13|     global left_mouse_button_pressed
14|
15|     if button == GLFW_MOUSE_BUTTON_LEFT and action == GLFW_PRESS:
16|         left_mouse_button_pressed = 1
17|     else:
18|         left_mouse_button_pressed = 0
```

- W funkcji `main()` dodano wywołania do obsługi zdarzeń.

```
1| def main():
2|     # (...)
3|     glfwSetKeyCallback(window, keyboard_key_callback)
4|     glfwSetCursorPosCallback(window, mouse_motion_callback)
5|     glfwSetMouseButtonCallback(window, mouse_button_callback)
6|     # (...)
```

## Nowości w przykładowym programie (3/3)

- ▶ Zmieniono parametry rzutowania – perspektywiczne w zakresie [0.1; 300].
- ▶ Dodano funkcję `example_object()`, rysującą przykładowy model.
- ▶ W funkcji `render()` wykonano transformacje wierzchołków.

```
1| def render(time):
2|     global theta
3|
4|     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
5|     glLoadIdentity()
6|
7|     gluLookAt(viewer[0], viewer[1], viewer[2],
8|              0.0, 0.0, 0.0, 0.0, 1.0, 0.0)
9|
10|    if left_mouse_button_pressed:
11|        theta += delta_x * pix2angle
12|
13|    glRotatef(theta, 0.0, 1.0, 0.0)
14|
15|    axes()
16|    example_object()
17|
18|    glFlush()
```

## Poruszanie kamerą dookoła obiektu

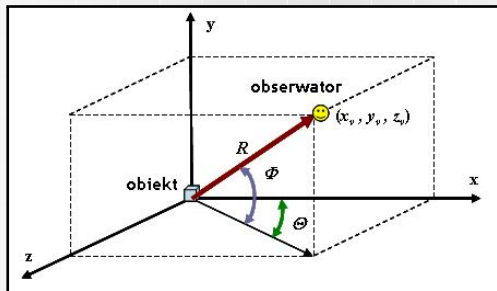
- ▶ Współrzędne kamery można określić za pomocą następujących równań,

$$x_{eye}(R, \theta, \phi) = R \cdot \cos(\theta) \cdot \cos(\phi),$$

$$y_{eye}(R, \theta, \phi) = R \cdot \sin(\phi),$$

$$z_{eye}(R, \theta, \phi) = R \cdot \sin(\theta) \cdot \cos(\phi).$$

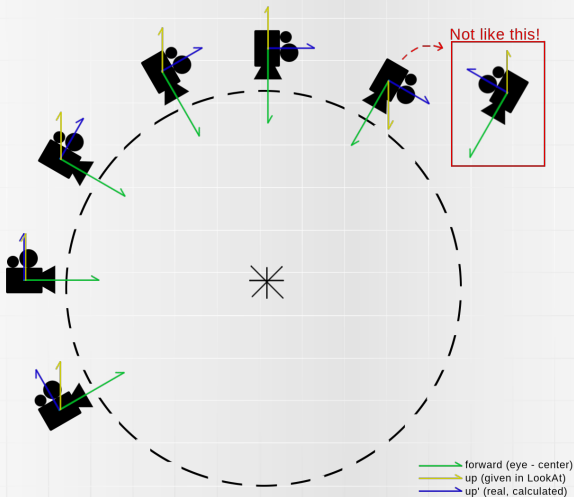
- ▶ Zakresy wartości kątów  $\theta$  i  $\phi$  to przedziały  $0 \leq \theta \leq 2\pi$  oraz  $0 \leq \phi \leq 2\pi$ .
- ▶ Parametr  $\theta$  to tak zwany kąt azymutu, zaś  $\phi$  to tak zwany kąt elewacji.





# Uwaga na wektor wskazujący górę

- ▶ Dla pewnego zakresu kąta elewacji, góra kamery powinna być odwrócona!
- ▶ W przeciwnym wypadku dojdzie do nieintuicyjnego obrotu kamery!



Koniec wprowadzenia.

# Zadania do wykonania...

# Zadania do wykonania (1)

Na ocenę **3.0** należy wprowadzić obracanie obiektu wokół osi X.

Wskazówki:

- przestudiować w jaki sposób zrealizowano obrót wokół osi Y o kąt `theta`,
- dodać zmienną pomocniczą `phi` i obsłużyć ruch myszą w pionie,
  - ▶ nadal bazować tylko na wciśniętym lewym przycisku myszy,
- intuicyjnie: obrót wokół osi X to obrót “góra-dół”,
- wymagane będzie zmodyfikowanie funkcji:
  - ▶ `mouse_motion_callback()`,
  - ▶ `render()`.

## Zadania do wykonania (2)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **3.5** należy wprowadzić obsługę drugiego przycisku myszy.

Wskazówki:

- obsługa nowego przycisku – w funkcji `mouse_button_callback()`,
- postać funkcji `mouse_motion_callback()` nie ulegnie zmianie,
- wprowadzić zmienną pomocniczą `scale`,
- w funkcji `render()`:
  - ▶ wykonać zmianę wartości zmiennej `scale`,
  - ▶ przy ruchu z wciśniętym prawym przyciskiem myszy,
- użyć wartości `scale` do przeskalowania obiektu – `glScalef()`.

## Zadania do wykonania (3)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **4.0** należy zrealizować poruszanie kamerą wokół modelu.

Wskazówki:

- nie powinno być konieczności modyfikacji funkcji obsługujących zdarzenia,
- zakomentować funkcje `glRotatef()` i `glScalef()` w funkcji `render()`,
- wykorzystać wartości zmiennych `theta` i `phi` do obliczenia  $x_{eye}$ ,  $y_{eye}$  i  $z_{eye}$ ,
- zdarzenia prawego przycisku myszy użyć do zmiany wartości parametru `R`,
- użyć wartości  $x_{eye}$ ,  $y_{eye}$  i  $z_{eye}$  jako argumentów funkcji `gluLookAt()`,
- wartości `theta` i `phi` przeskalować w funkcjach `sin()` i `cos()` przez  $\frac{\pi}{180}$ .

## Zadania do wykonania (4)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **4.5** należy usprawnić poruszanie kamerą wokół modelu.

Wskazówki:

- wprowadzić ograniczenia w zakresie przybliżania/oddalania kamery,
- zapewnić poprawność przejść kamery wokół modelu,
  - ▶ w szczególności zwrócić uwagę co dzieje się “nad” i “pod” obiektem,
  - ▶ pomoce może być użycie modułu (funkcja `fmod()` lub operator `%`) w celu ograniczenia wartości kątów do przedziału  $[0; 2\pi]$ ,
- wprowadzić możliwość przełączania między trybem obracania obiektu i trybem poruszania kamerą – na przykład za pomocą klawiatury.

# Zadania do wykonania (5)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **5.0** należy zrealizować zadanie dodatkowe.

Wskazówki:

- wybrać jeden z przykładów zaproponowanych jako "zadania domowe",
  - ▶ dokument znajduje się na stronie prowadzącego,
  - ▶ uściślenie odnośnie Zadania 4.5 (Trójkąt Sierpińskiego):
    - ruch ma działać analogicznie, jak w opisie poniżej,
- alternatywnie można wykonać swobodny pierwszoosobowy ruch kamery,
  - ▶ cel: ruch podobny do znanego z gier pierwszoosobowych,
  - ▶ wprowadzić zmienną pomocniczą dla kierunku patrzenia,
  - ▶ klawisze [w][s][a][d] posłużą do zmiany położenia obserwatora,
  - ▶ ruch myszką powinien wpływać na kierunek patrzenia,
  - ▶ punkt zainteresowania / kierunek patrzenia można wyznaczyć analogicznie do przypadku z kamerą poruszaną dookoła obiektu.