Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

http://bazyluk.net/dydaktyka

Wydział Informatyki

Bartosz Bazyluk

Wprowadzenie

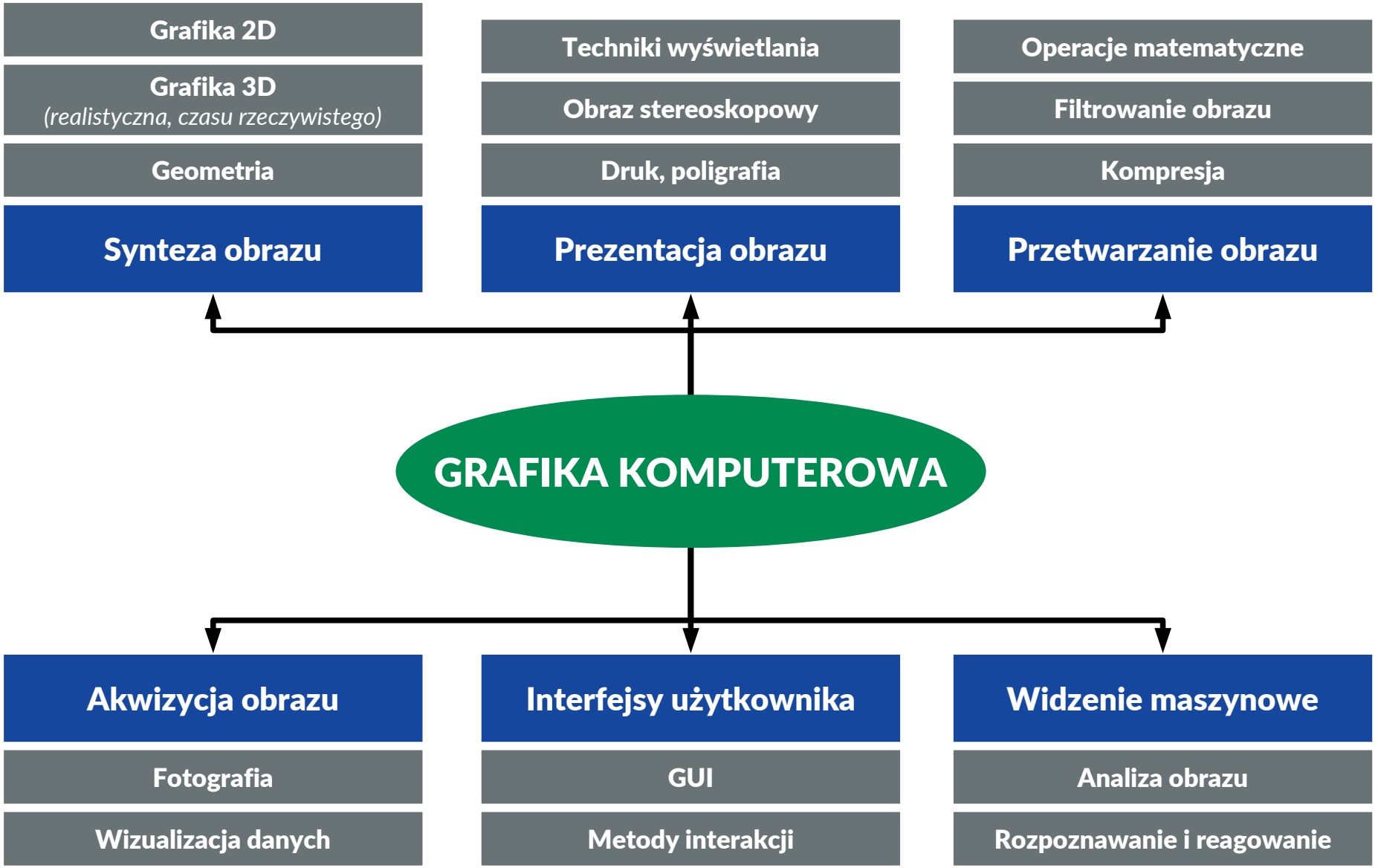
Organizacja i tematyka zajęć, warunki zaliczenia

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

O MNIE

- mgr inż. **Bartosz Bazyluk**
- Pokój **322/WI2** lub **316/WI2**
- bbazyluk@wi.zut.edu.pl
- http://bazyluk.net/dydaktyka
  - wykłady, zadania, materiały, informacje

- Czym się zajmuję:
  - Programowanie **grafiki czasu rzeczywistego** i gier komputerowych 3D
  - **Eye tracking** jako narzędzie badań i interakcji
  - **Fotografia HDR** i przetwarzanie **obrazów HDR**
  - **Percepcja głębi** w obrazach syntezowanych komputerowo


Grafika 2D
Grafika 3D
(realistyczna, czasu rzeczywistego)
Geometria
Synteza obrazu
Techniki wyświetlania
Obraz stereoskopowy
Druk, poligrafia
Prezentacja obrazu
Operacje matematyczne
Filtrowanie obrazu
Kompresja
Przetwarzanie obrazu
GRAFIKA KOMPUTEROWA
Akwizycja obrazu
Fotografia
Wizualizacja danych
Interfejsy użytkownika
GUI
Metody interakcji
Widzenie maszynowe
Analiza obrazu
Rozpoznawanie i reagowanie

TEMATYKA WYKŁADÓW

- **Synteza grafiki 3D** w czasie rzeczywistym
  - **Programowanie grafiki**
    - *OpenGL*
  - **Techniki** grafiki czasu rzeczywistego
    - **Transformacje** geometryczne
    - **Scena** i **kamera**
    - **Oświetlenie** i **cieniowanie**
    - Definicja **geometrii**
    - **Teksturowanie**
    - Programy **cieniujące**
- **Obraz**
  - **Przechowywanie** i **kompresja** obrazu
  - **Wyświetlanie** obrazu
  - **Fotografia** i obrazowanie ***HDR***

LITERATURA

- **Grafika komputerowa**
  - Peter Shirley, Stephen Robert Marschner. **"Fundamentals of Computer Graphics"**. AK Peters, 2009.

- **Grafika czasu rzeczywistego**
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman. ***"Real-Time Rendering, Third Edition"***. AK Peters, 2008.
  - Mike McShaffry, David Graham. ***"Game Coding Complete, Fourth Edition"***. Course Technology, 2012.
  - **"OpenGL Blue Book"**, czyli dokumentacja OpenGL. http://www.opengl.org/sdk/docs/
  - **"OpenGL Red Book"**, czyli wprowadzenie do OpenGL. http://www.glprogramming.com/red/

ZALICZENIE PRZEDMIOTU

- Uzyskanie oceny z **zajęć laboratoryjnych**
- Pozytywne zaliczenie **egzaminu końcowego**
  - Egzamin **pisemny**, forma **testowa**
  - Pytania mogą dotyczyć **wszystkich zagadnień** poruszanych na wykładach i laboratoriach
    - (bez spraw czysto programistycznych, nie programujemy na kartkach i bez dokumentacji)
  - Należy zdobyć **przynajmniej 50% punktów**, aby uzyskać ocenę pozytywną
  - **Dopuszczenie do egzaminu** warunkowane jest *wcześniejszym* uzyskaniem zaliczenia laboratoriów

KONKURS

- **Szczecin Gamedev Talents**

  - **Coroczny konkurs na najlepszą studencką grę** towarzyszący przedmiotowi Grafika Komputerowa i Wizualizacja
  - Już **9. edycja**!

  - Aby **wziąć udział** w konkursie, należy uzyskać ocenę bardzo dobrą z laboratoriów **do dnia 22 stycznia 2016 r.**
  - **Ogłoszenie wyników**: ostatni wykład, **25 stycznia 2016 r.**
  - Osoby biorące udział w konkursie są **zwolnione z egzaminu** z oceną bardzo dobrą


Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Grafika Komputerowa

Wstęp do syntezy trójwymiarowej grafiki komputerowej

Wydział
Informatyki

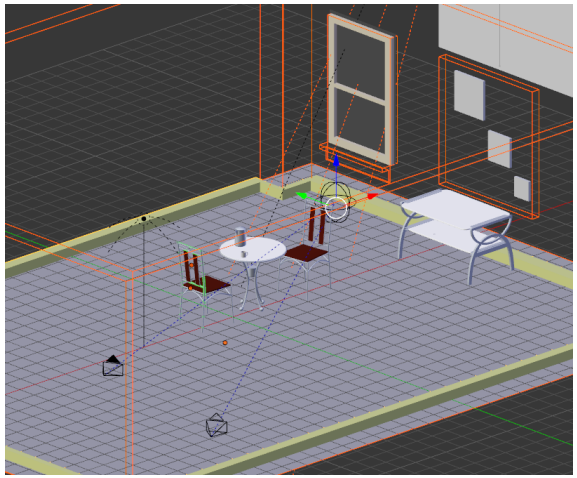
Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

Synteza grafiki 3D

- Pod pojęciem **syntezy grafiki** rozumiemy stworzenie grafiki na podstawie jej opisu
- **Komputerowa synteza grafiki** to procesy algorytmiczne prowadzące do uzyskania cyfrowego obrazu na podstawie opisu jego zawartości
- **Synteza grafiki 3D** bazuje na opisie przestrzennego świata, znajdujących się w nim obiektów, ich cech, środowiska, itp.
  - **Opis** wycinka świata na potrzeby syntezy grafiki 3D nazywamy **sceną**
  - **Celem** syntezy grafiki 3D jest uzyskanie **obrazu 2D** będącego wizualizacją sceny przy zadanych kryteriach

Synteza grafiki 3D

- **Scena**
  - Geometryczny **opis** rozmieszczenia obiektów, oświetlenia, kamery, ...



- **Algorytm graficzny**
  - Proces często nazywa się ***renderowaniem***

- **Obraz 2D**
  - **Wizualizacja**, "zdjęcie" sceny 3D widzianej w zadany sposób

Renderowanie

- Grafika **realistyczna**
  - Renderowanie ***off-line***
  - **Nieograniczony czas** na przygotowanie każdej klatki animacji
  - Celem jest uzyskanie **jak najlepszego obrazu**
  - Zastosowania:
    - Film
    - Wizualizacja produktów
    - Sztucznie generowane obrazy
    - Sztuka
- Grafika **czasu rzeczywistego**
  - Renderowanie ***on-line***
  - Konieczność **szybkiego wyświetlania** kolejnych klatek animacji
  - Celem jest uzyskanie **satysfakcjonującego obrazu** przy zachowaniu **płynności animacji**
  - Stosowane są liczne **uproszczenia**
  - Zastosowania:
    - Gry komputerowe
    - Interaktywne symulacje
    - Wirtualna rzeczywistość

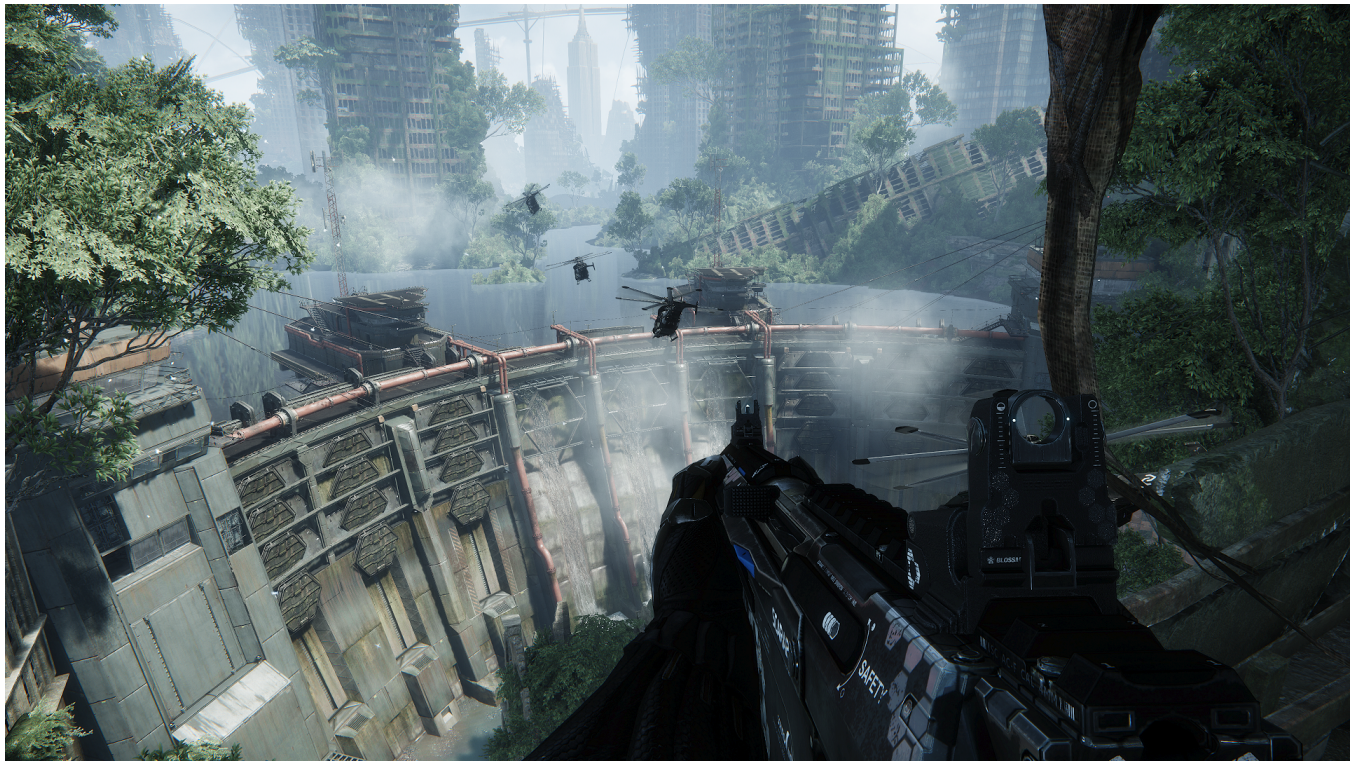
Grafika realistyczna

- **Nieograniczony czas** na renderowanie
  - Aby uzyskać najlepszy efekt, można poświęcić **od kilku sekund do nawet kilku dni** na wyrenderowanie jednej klatki animacji
- Algorytmy mające za zadanie naśladować **rzeczywiste zjawiska optyczne**
  - **Ogromna ilość obliczeń**, algorytmy takie jak np.:
    - *Raytracing*
    - *Photon mapping*

Źródło obrazu:
http://venturebeat.com

Grafika czasu rzeczywistego

- Konieczność uzyskania **płynności animacji**
  - **Często mniej, niż 1/60 sekundy** aby wyrenderować klatkę
- Generowanie grafiki poprzez **uproszczenia**
  - Redukcja **szczegółowości**
  - Uproszczony **model oświetlenia**
  - Ograniczona symulacja **zjawisk wizualnych**



Źródło obrazu:
Crysis 3, Crytek

PROGRAMOWANIE

GRAFIKI KOMPUTEROWEJ

CZASU RZECZYWISTEGO

Źródło obrazu:
Future Network USA

PROGRAMOWANIE

GRAFIKI KOMPUTEROWEJ

CZASU RZECZYWISTEGO

Game.exe

Źródło obrazu:
Electronic Arts

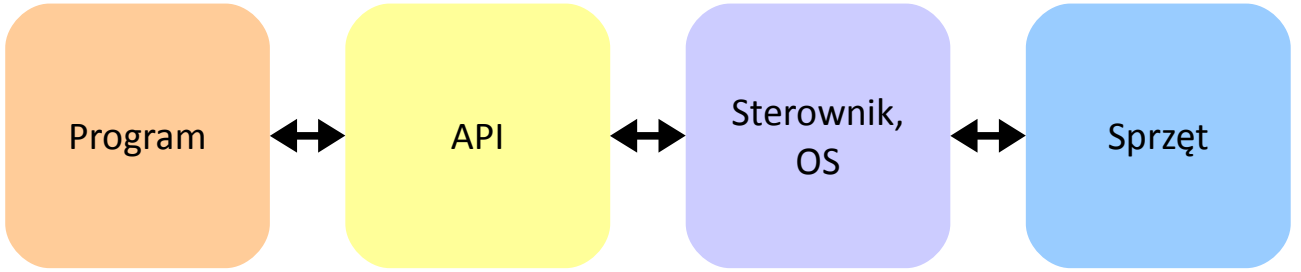
- Uproszczony **schemat działania** aplikacji grafiki czasu rzeczywistego
  - Problem: **zróżnicowanie platform** sprzętowo-programowych

Nasz program

System operacyjny

Karta graficzna

- Uproszczony **schemat działania** aplikacji grafiki czasu rzeczywistego
  - Ułatwienie: **ujednolicone *API*** graficzne



Nasz program

Ujednolicone API do programowania grafiki czasu rzeczywistego

System operacyjny

Karta graficzna

OpenGL

- **Open G**raphics **L**ibrary
- Jest to **API** pozwalające na renderowanie grafiki w czasie rzeczywistym, zorientowane na wykorzystanie **akceleracji sprzętowej**
- OpenGL **<u>nie</u> jest biblioteką**
  - Jest to **specyfikacja** interfejsu programowania (*API*)
  - Bibliotekami są **implementacje OpenGL** na konkretne platformy, dostarczane przez producentów sprzętu, systemów operacyjnych itp.
- Jednym z głównych celów OpenGL jest **wieloplatformowość**
- Konkurencją jest *Microsoft Direct3D* z ***DirectX***

OpenGL

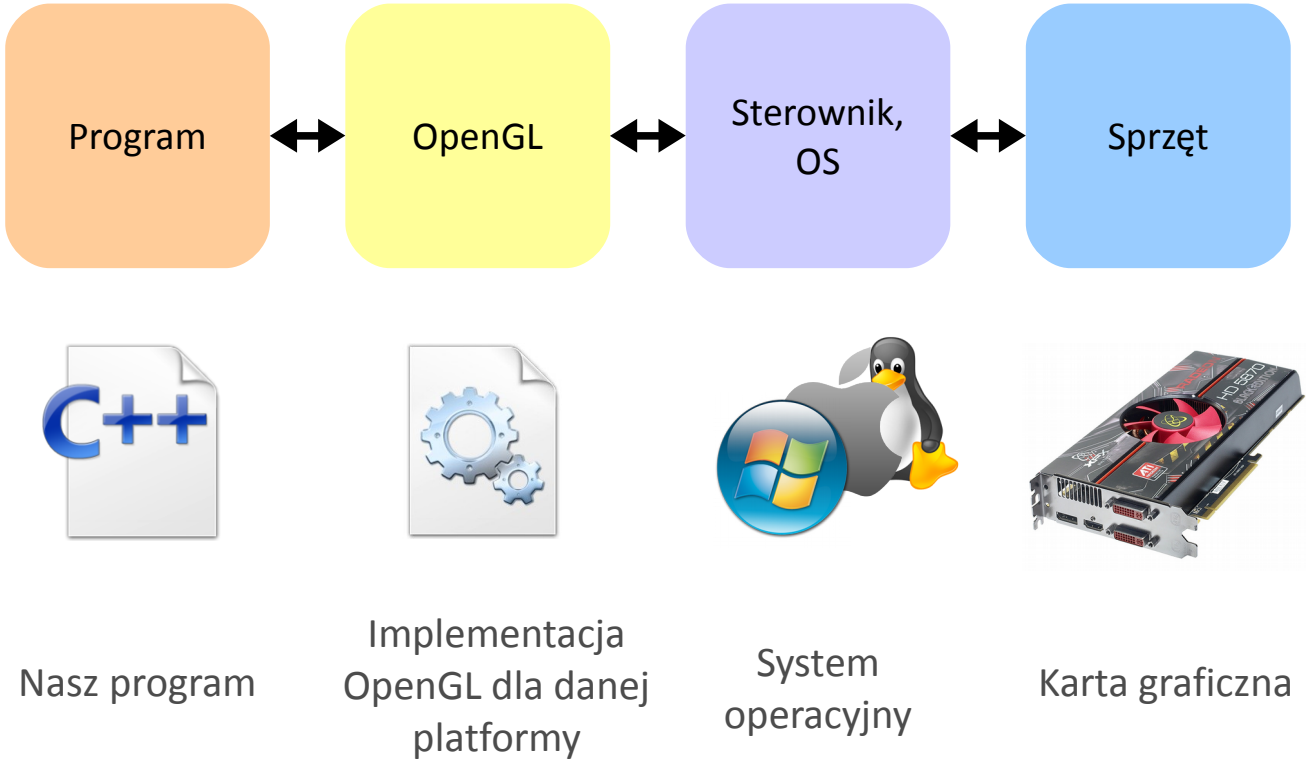
- OpenGL opisuje **zbiór funkcji i numerycznych stałych**, pozwalających na renderowanie grafiki
  - Nie obejmuje procedur **obsługi wejścia**, czy integracji z systemem operacyjnym w celu **utworzenia kontekstu**
    - Istnieją multiplatformowe **biblioteki** dedykowane dla OpenGL, które rozwiązują te problemy: np. *GLUT* (*freeglut*), *GLFW*
- Założeniem jest, że implementacja OpenGL opiera się głównie na procedurach **sprzętowych**
  - Operacje wykonywane bezpośrednio **przez kartę graficzną**, a nie software'owe rozwiązania oparte o *CPU*
  - Wiele z oferowanych możliwości, jeśli nie są zaimplementowane w procedurach sprzętowych, może być **emulowanych na *CPU***

OpenGL

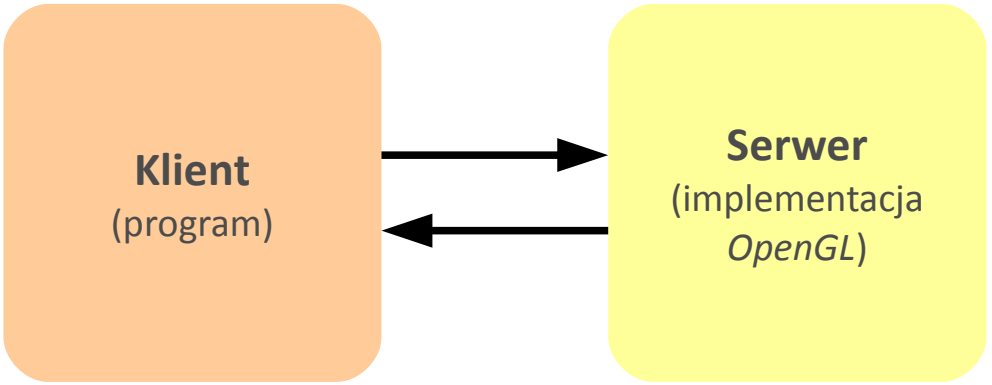
- **Historia**
  - 1992 – **OpenGL 1.0** wydany przez Sillicon Graphics (SGI)
  - 1997-2003 – rozszerzenia **OpenGL 1.x**
  - 2004 – **OpenGL 2.0**, zorientowany na programowalny potok renderowania (shadery, GLSL)
  - 2006 – OpenGL rozwijany jest przez non-profitową organizację ***Khronos Group***
  - 2008 – **OpenGL 3.0**, znaczne ograniczenie API, usunięcie archaicznych podejść
  - 2010 – **OpenGL 4.0**, odpowiednik DirectX 11 (np. teselacja)
  - 2014 – **OpenGL 4.5**, aktualna wersja
- **OpenGL ES**
  - Podzbiór standardu OpenGL, przeznaczony do implementacji na **urządzeniach mobilnych**
  - OpenGL ES 3.1 – wersja najbardziej aktualna (2014), kopmpatybilna z OpenGL 4.5

OpenGL

- OpenGL dostarcza procedur, za pomocą których możemy w **ujednolicony sposób** programować grafikę:
  - na różnych **kartach graficznych**
  - na różnych **systemach operacyjnych**



- Architektura wykorzystania *OpenGL* jako **warstwy pośredniej** pomiędzy aplikacją, a kartą graficzną opiera się o **model klient-serwer**
  - Specyfikacja OpenGL wcale nie wymaga, aby obie strony znajdowały się w jednej, fizycznej maszynie!



- Zlecenia renderowania
- Opis sceny
- Używa pamięci głównej
- Logika programu
- Użycie wynikowego obrazu

- Konteksty
- Renderowanie
- Komunikacja ze sterownikiem
- Dostęp do *GPU* i pamięci graficznej

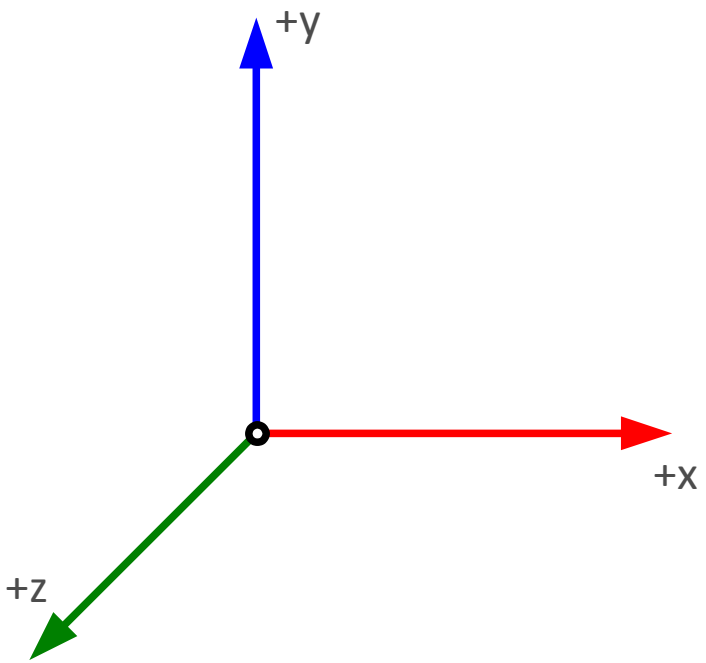
OpenGL

- Działanie OpenGL należy rozumieć jako **maszynę stanu**
  - **Stanem** nazywamy aktualne, chwilowe wartości wszystkich parametrów systemu
- **Jeśli zmienimy wartość jakiegoś parametru, pozostaje ona zmieniona do czasu kolejnej zmiany**
  - Przykład:
    Jeśli zaczniemy w jednej klatce rysować czerwony prostokąt, wszystkie następne obiekty będą czerwone jeśli nie dokonamy zmiany koloru. Nawet w następnej klatce!

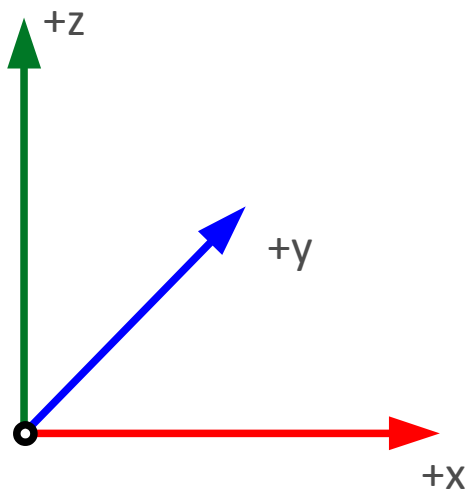
Współrzędne

- **Układ współrzędnych** w przestrzeni trójwymiarowej

OpenGL
„prawoskrętny”

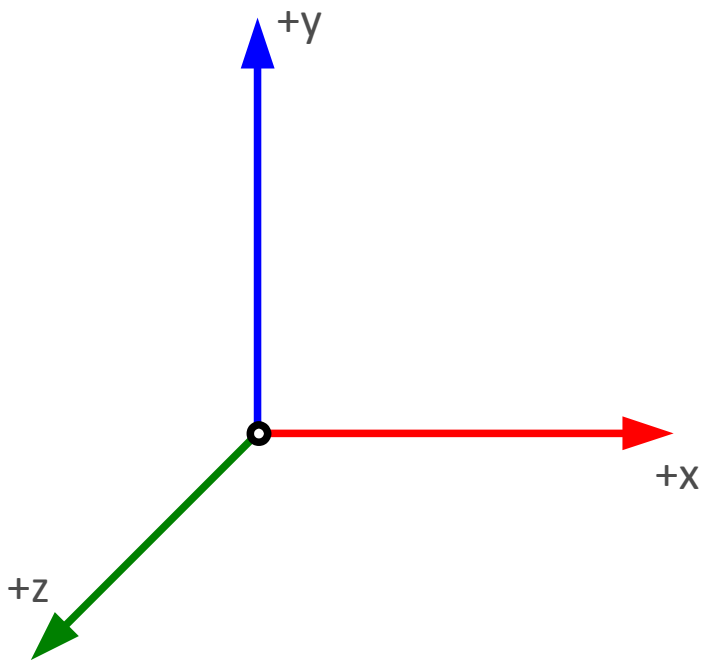


Direct3D
„lewoskrętny”



- **Jednostki są zupełnie arbitralne**, można przyjąć np. że jedna jednostka to 1 metr (albo 13 cm, albo 2 ft, albo...)

- **Układ współrzędnych** w OpenGL
  - **Trzeba** zapamiętać albo mieć pod ręką!



- **Jednostki są zupełnie arbitralne**, można przyjąć np. że jedna jednostka to 1 metr (albo 13 cm, albo 2 ft, albo...)

GLUT

- **GL Utility Toolkit**
  - Biblioteka narzędziowa do OpenGL
  - ***Freeglut*** jest open-source'owym klonem, z którego będziemy korzystać (w pełni kompatybilny z GLUT)
- Dostarcza m.in.:
  - **Międzyplatformową** obsługę **urządzeń wejścia** (mysz, klawiatura)
  - Łatwe **tworzenie okien** z kontekstem OpenGL
  - Procedury rysowania **prostych brył** (sześcian, kula itd.)
  - Implementację **pętli głównej** (*mainloop/gameloop*)
- Wykorzystuje ideę ***callbacków***, czyli funkcji obsługujących zdarzenia
  - W ten sposób programista ma możliwość reakcji na zdarzenia
  - Przypisanie *callbacka* odbywa się poprzez wywołanie specjalnej funkcji ustawiającej, z wskaźnikiem do funkcji obsługującej zdarzenie jako parametr

Przykładowy program

```
int main(int argc, char * argv[]) {
	glutInit(&argc, argv);

	glutInitWindowSize(640, 360);
	glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
	glutCreateWindow("OpenGL/GLUT Example");

	glutDisplayFunc(OnRender);

	glutMainLoop();

	return 0;
}
```

```
void OnRender() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt(0.0f, 0.0f, -6.0f,   0.0f, 0.0f, 0.0f,   0.0f, 1.0f, 0.0f);

	glColor3f(1.0f, 0.0f, 0.0f);
	glutSolidCube(1.0f);

	glutSwapBuffers();
	glFlush();
	glutPostRedisplay();
}
```

- Nazwy funkcji *OpenGL*:
  - **gl**NazwaFunkcji**param**(...)
    - **param** – jeśli dana funkcja może przyjmować różne rodzaje parametrów, w ten sposób określany jest rodzaj tych parametrów oraz ich liczba, np.:
      - glVertex**3f**(float x, float y, float z)
        glVertex**3fv**(float *xyz)
        glVertex**2f**(float x, float y)
        glVertex**3d**(double x, double y, double z)
        glVertex**2i**(int x, int y)
        ...
      - **f** – float, **d** – double, **i** – int, **b** – bool itd.
      - **v** – vector (tablica)
  - **gl**NazwaFunkcji**EXT**(...), **gl**NazwaFunkcji**ARB**(...)
    - Funkcjonalności pochodzące z rozszerzeń niebędących częścią standardu. Niektóre z nich w późniejszych wersjach OpenGL przestają posiadać sufiksy *EXT* i *ARB*.
      - Do obsługi rozszerzeń warto użyć biblioteki ***GLEW***
- Nazwy funkcji *GLU* oraz *GLUT*:
  - **glu**NazwaFunkcji(...), **glut**NazwaFunkcji(...)

VULKAN

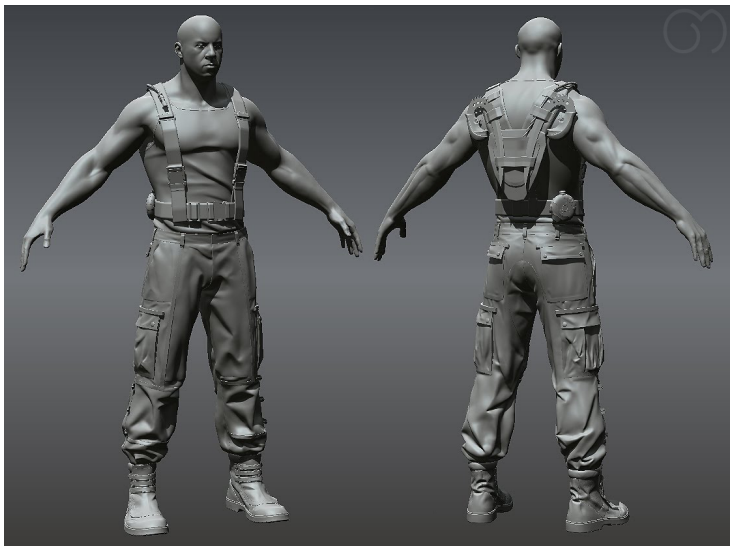
- Nowe API będące **spadkobiercą** OpenGL
  - Pierwszy raz zaprezentowane podczas ***Game Developer's Conference* (GDC) 2015**
  - Rozwijane przez **Khronos Group**
  - Wsparcie: m.in. ***Source 2 (DOTA2)***, przyszły ***Android***
  - Karty graficzne obsługujące min. *OpenGL 4.x*, *OpenGL ES 3.1*
  - Współczesne, obciążone małym narzutem (ang. ***lightweight***), wykorzystujące nowoczesne techniki, m.in.:
    - Binarna **postać pośrednia** programów cieniujących
    - Lepsze wsparcie dla sytuacji gdy **CPU jest wąskim gardłem**
    - Nastawione na **przetwarzanie wielowątkowe**

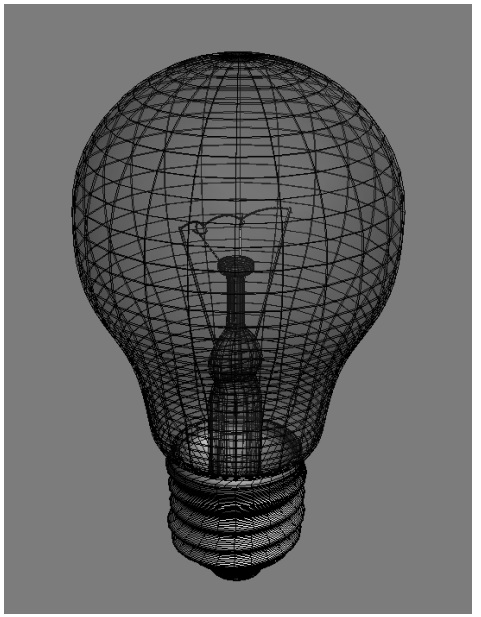
Scena 3D

- Elementy **sceny**:
  - **Obiekty geometryczne**
    - Bryła, położenie, kierunek obrotu, skala, ...
  - **Cechy obiektów** (tzw. *materiałów* z których są zbudowane)
    - Cechy powierzchni, kolor, tekstura, ...
  - **Kamera**
    - Położenie, kierunek widzenia, kąt widzenia, ...
  - **Źródła światła**
    - Położenie, rodzaj, kolor, intensywność, ...
  - **Cechy środowiska**
    - Mgła, dym, dodatkowe efekty wizualne, ...

Modele 3D

- Obiekty wirtualnego świata reprezentowane są na scenie najczęściej przez tzw. **modele**
- Modele to **geometryczna reprezentacja brył obiektów**
- Model nie jest grafiką, dopóki nie zostanie **wyrenderowany** – on tylko opisuje obiekt
  - Analogia do grafiki wektorowej

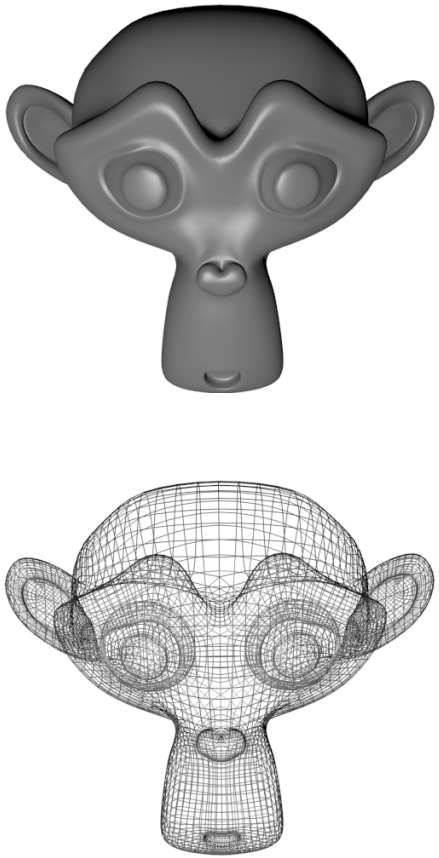




Źródła obrazów:
http://nzchristie.blogspot.com
http://maxattivo.blogspot.com

Modele 3D

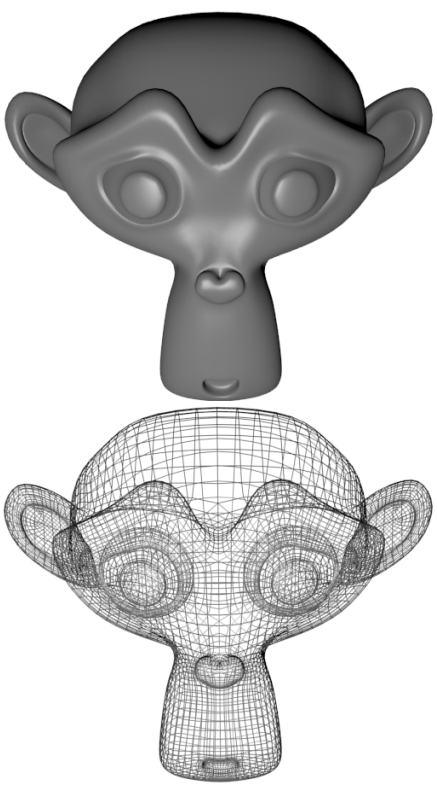
- Modele składają się z **wierzchołków**
- Wierzchołki tworzą **wielokąty** (ang. *Polygons*) które składają się na powierzchnię modelu



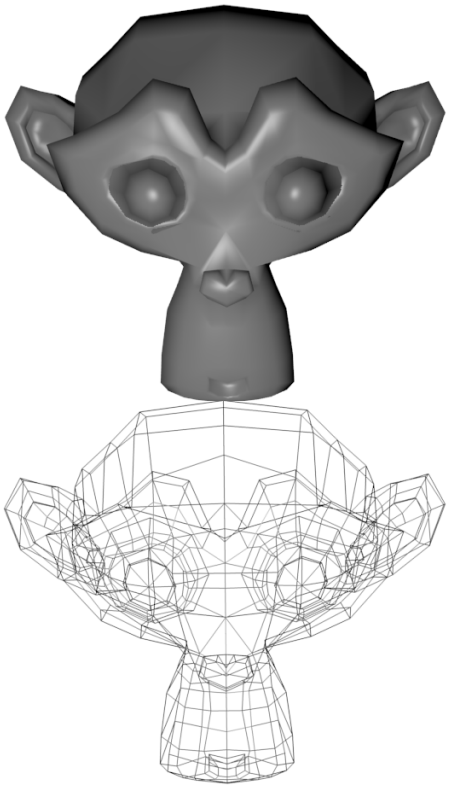
Modele 3D

- W grafice czasu rzeczywistego często stosuje się tzw. modele *low-poly*, czyli o **małej liczbie wielokątów**
- W ten sposób uzyskuje się **znacznie większą wydajność kosztem utraty jakości** (szczegółowości)

High-poly
(15488 trójkątów)

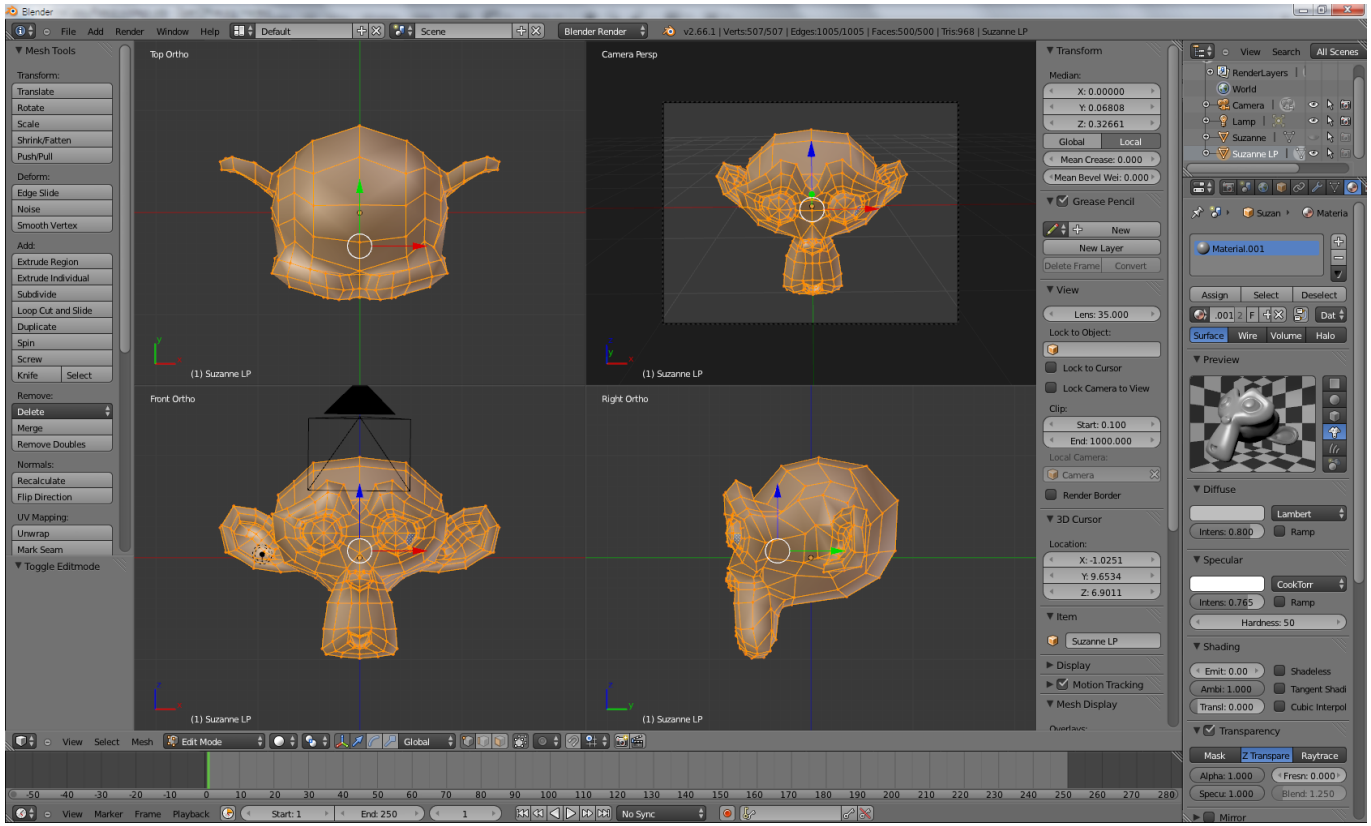


Low-poly
(968 trójkątów)



Modele 3D

- Modele tworzy się w wyspecjalizowanym oprogramowaniu do tworzenia grafiki 3D, np.:
  - **Blender** (darmowy)
  - **3ds Max**
  - **Maya**
  - **Google SketchUp** (darmowy)



Modele 3D

- Przykładowy **format**, w którym zapisuje się modele:
  - Wavefront **OBJ**
    - Plik **tekstowy**
    - Kolejne **linijki** odpowiadają m.in. pozycjom wierzchołków, wektorom normalnym, współrzędnym tekstur, ścianom

```
# Blender v2.66 (sub 1) OBJ File: ''
# www.blender.org
o Suzanne_LP_Suzanne.001
v 0.437500 0.164062 0.765625
v -0.437500 0.164062 0.765625
v 0.500000 0.093750 0.687500
v -0.500000 0.093750 0.687500
v 0.546875 0.054687 0.578125
v -0.546875 0.054687 0.578125
v 0.351562 -0.023438 0.617188
v -0.351562 -0.023438 0.617188
v 0.351562 0.031250 0.718750
v -0.351562 0.031250 0.718750
v 0.351562 0.132812 0.781250
v -0.351562 0.132812 0.781250
v 0.273438 0.164062 0.796875
v -0.273438 0.164062 0.796875
v 0.203125 0.093750 0.742188
v -0.203125 0.093750 0.742188
v 0.156250 0.054687 0.648438
v -0.156250 0.054687 0.648438
v 0.078125 0.242187 0.656250
v -0.078125 0.242187 0.656250
v 0.140625 0.242187 0.742188
v -0.140625 0.242187 0.742188
v 0.242188 0.242187 0.796875
v -0.242188 0.242187 0.796875
v 0.273438 0.328125 0.796875
v -0.273438 0.328125 0.796875
v 0.203125 0.390625 0.742188
v -0.203125 0.390625 0.742188
v 0.156250 0.437500 0.648438
v -0.156250 0.437500 0.648438
```

Przekształcenia geometryczne

Model

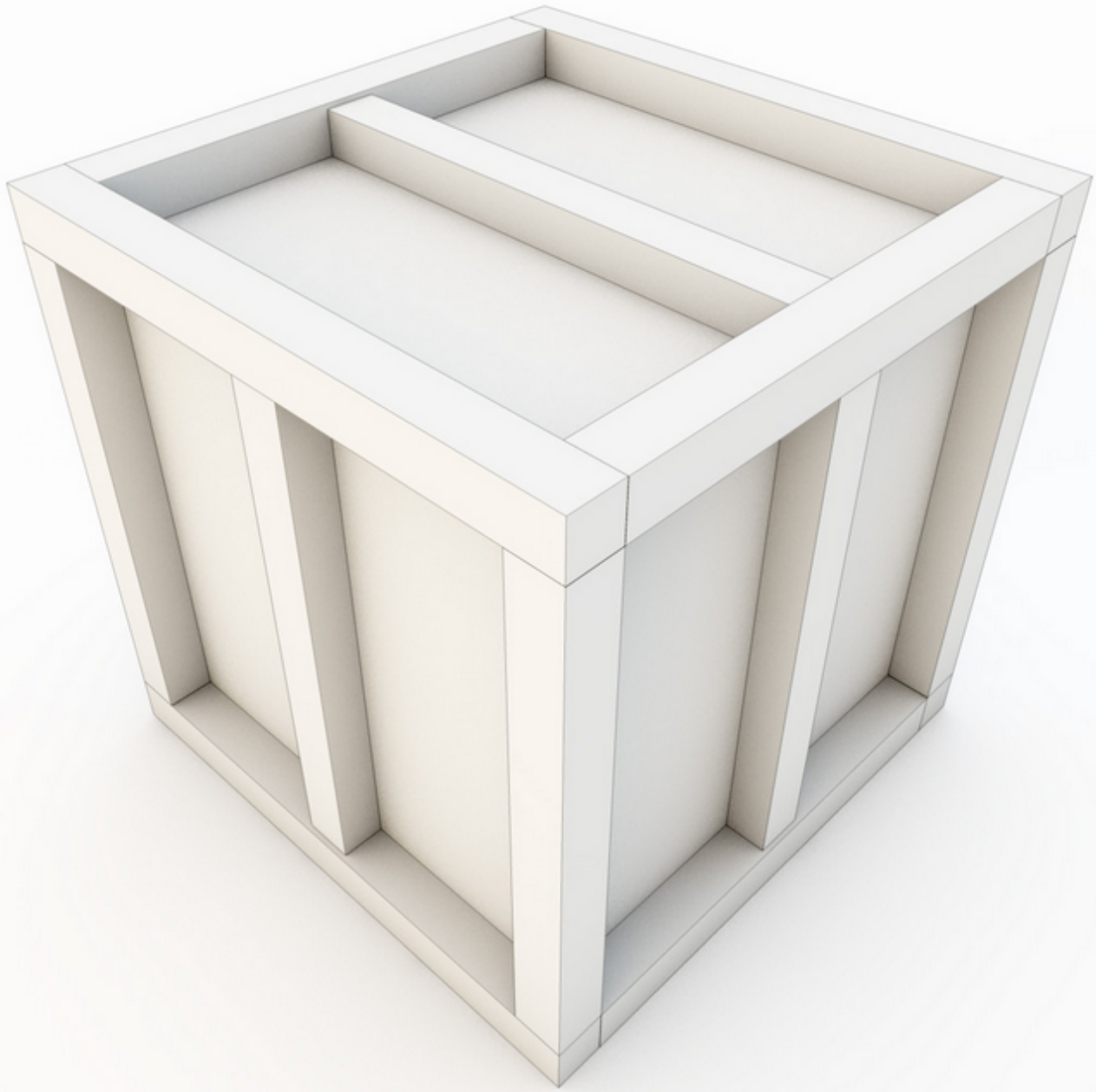
Posiadamy **model skrzyni**, którego **instancje** chcemy umieścić na naszej **scenie**.

Źródło obrazu:
Turbosquid / C4Dennis

Przekształcenia geometryczne

Model

Model zdefiniowany jest w charakterystycznym dla niego, **lokalnym układzie współrzędnych**.

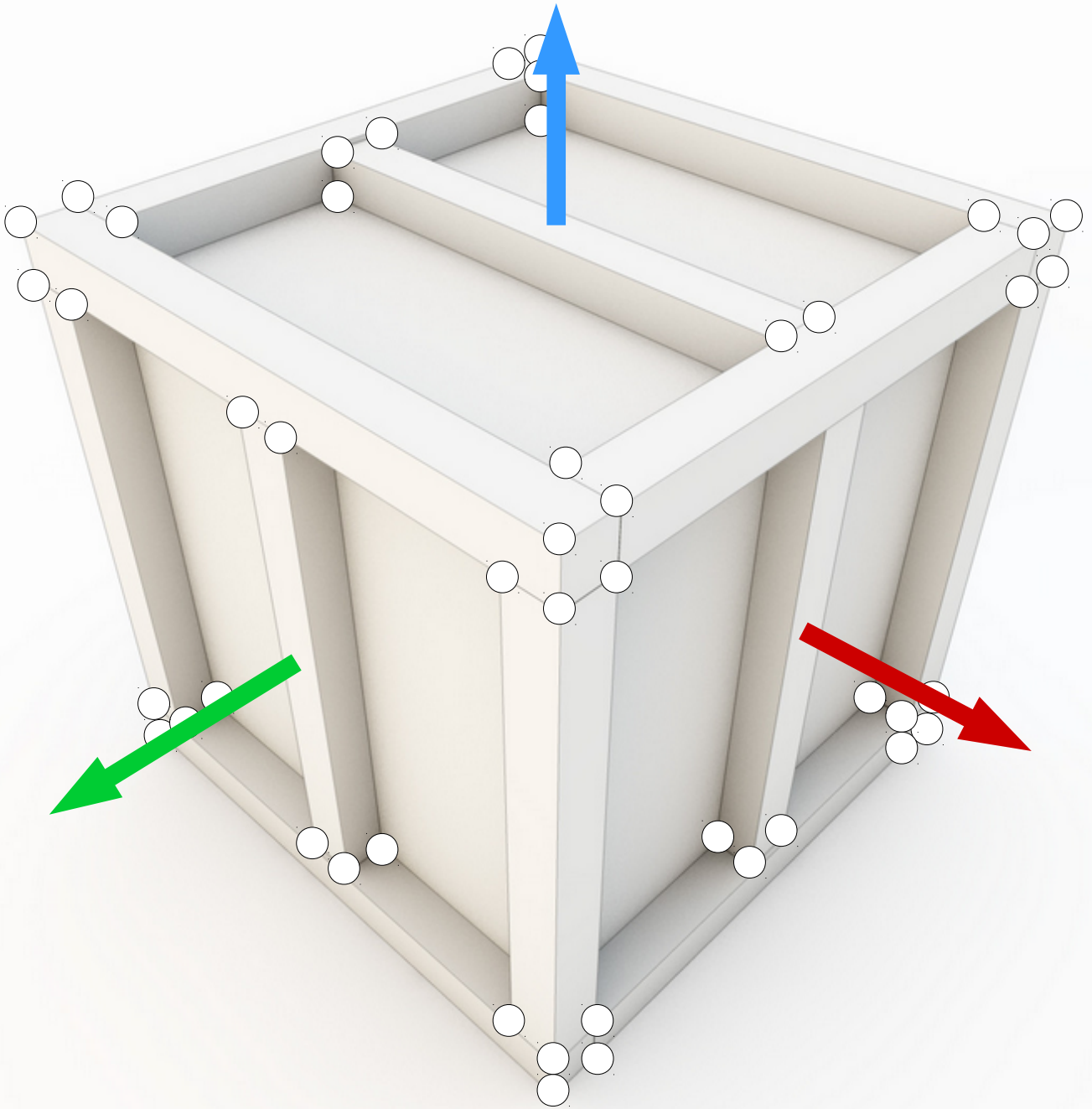


Źródło obrazu:
Turbosquid / C4Dennis

Przekształcenia geometryczne

Współrzędne lokalne

Każdy wierzchołek ma swoje własności określone w tym układzie.



Źródło obrazu:
Turbosquid / C4Dennis

Przekształcenia geometryczne

Wiele obiektów na scenie

Chcąc umieścić na scenie kilka skrzyń będących **instancjami naszego modelu**, musielibyśmy zdefiniować wszystkie ich wierzchołki we wspólnym **układzie współrzędnych świata**.

Wymagałoby to **osobnego zdefiniowania** każdej ze skrzyń, kolejno w każdym miejscu jej wystąpienia i z każdą orientacją.

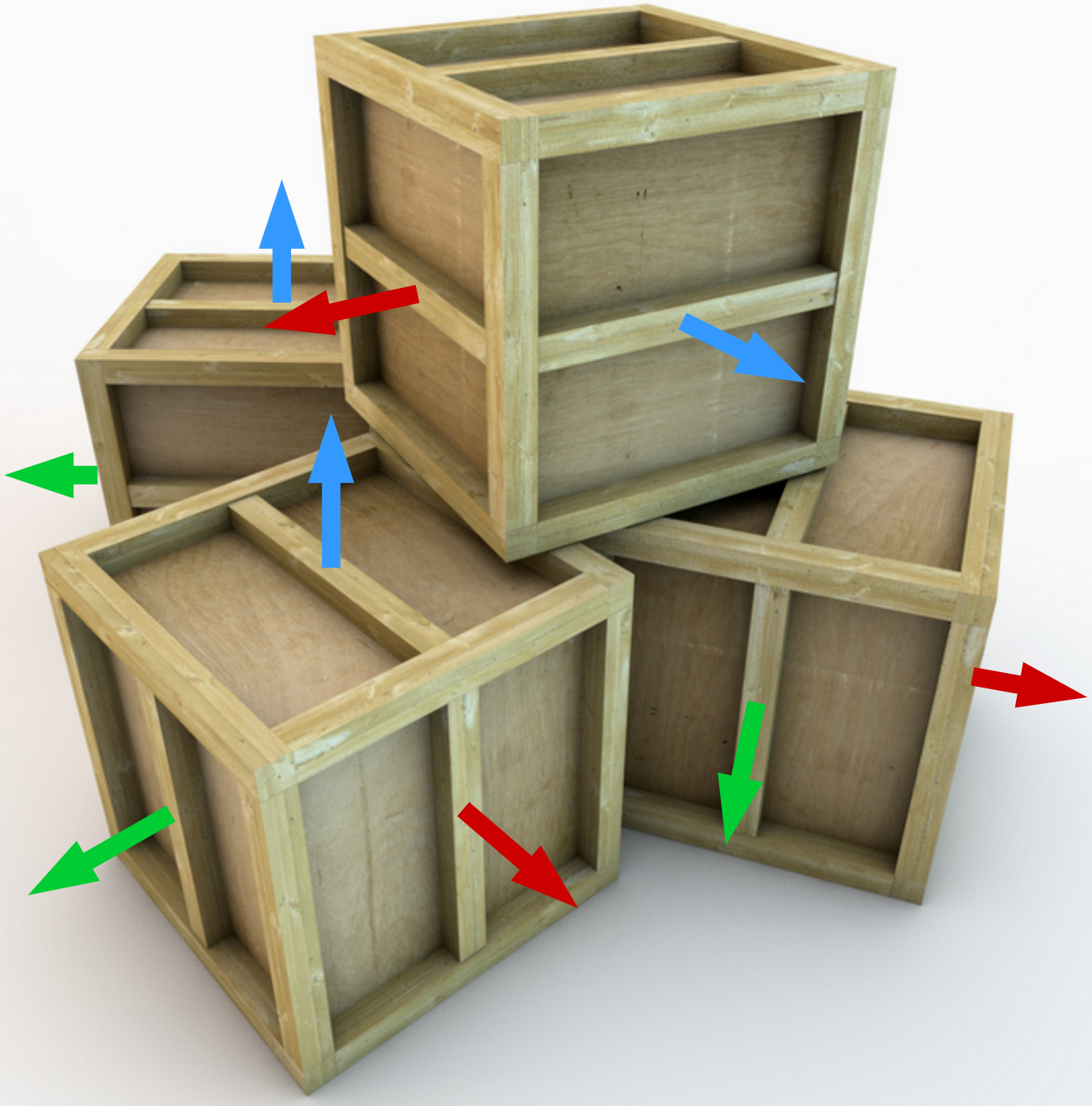
Źródło obrazu:
Turbosquid / C4Dennis

Przekształcenia geometryczne

Translacja, Rotacja, Skalowanie

Obiekty są wielokrotną instancją **tego samego** modelu, zdefiniowanego w tym samym **lokalnym układzie współrzędnych**.

Poprzez różną transformację (translację, rotację, skalowanie) w **przestrzeni świata** mają jednak różną **orientację**.



Źródło obrazu:
Turbosquid / C4Dennis

Przekształcenia geometryczne

- Przekształcenia geometryczne w przestrzeni 3D opisane są **macierzami 4x4**
  - Aby **poddać współrzędne** (x, y, z) **danemu przekształceniu** T, macierz T mnożona jest przez wektor (rozszerzony o dodatkową współrzędną $w = 1$), a cały wynik jest dzielony przez ostatnią współrzędną w'

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} T_{00} & T_{01} & T_{02} & T_{00} \\ T_{10} & T_{11} & T_{12} & T_{13} \\ T_{20} & T_{21} & T_{22} & T_{23} \\ T_{30} & T_{31} & T_{32} & T_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Przekształcenia geometryczne

- Podstawowe transformacje geometryczne to:
  - Translacja
  - Rotacja
  - Skalowanie

- Macierze, które je opisują:

X-Rotation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-Rotation in 3D

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale in 3D

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-Rotation in 3D

$$\begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Przekształcenia geometryczne: Przykład

- Translacja o wektor (4, -6, 2):

$$M_T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{xyz} = (4, -6, 2)$$

Przekształcenia geometryczne: Przykład

- Translacja o wektor (4, -6, 2):

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Niech *p* będzie punktem, który chcemy poddać translacji:

$p = (1, 2, 3)$ $\qquad$ $p' = ?$

(punkt po transformacji)

Przekształcenia geometryczne: Przykład

- Translacja o wektor (4, -6, 2):

Niech p będzie punktem, który chcemy poddać translacji:

$p = (1, 2, 3)$ $\qquad$ $p' = ?$

$$p' = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} = \ldots ?$$

Przekształcenia geometryczne: Przykład

- Translacja o wektor (4, -6, 2):

Niech p będzie punktem, który chcemy poddać translacji:

$p = (1, 2, 3)$ $\qquad$ $p' = ?$

$$p' = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 4 \cdot 1 \\ 0 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 - 6 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 2 + 1 \cdot 3 + 2 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 3 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 5 \\ -4 \\ 5 \\ 1 \end{bmatrix}$$

(aby **pozbyć się czwartej współrzędnej**, należy najpierw cały wektor **podzielić przez jej wartość**)

Przekształcenia geometryczne

- **Macierz modelu** jest złożeniem odpowiednich translacji, rotacji i skalowań
  - Czyli **iloczynem macierzy** odpowiadających takim transformacjom
  - Ważna jest **kolejność** mnożenia!
- Służy do **umieszczenia naszego modelu**, opisanego lokalnymi współrzędnymi, **w pożądanym miejscu** wirtualnego **świata**
- Dzięki takiemu podejściu możliwe jest **wielokrotne używanie** dokładnie tego samego modelu, opisanego **tymi samymi współrzędnymi wierzchołków**

Przekształcenia geometryczne

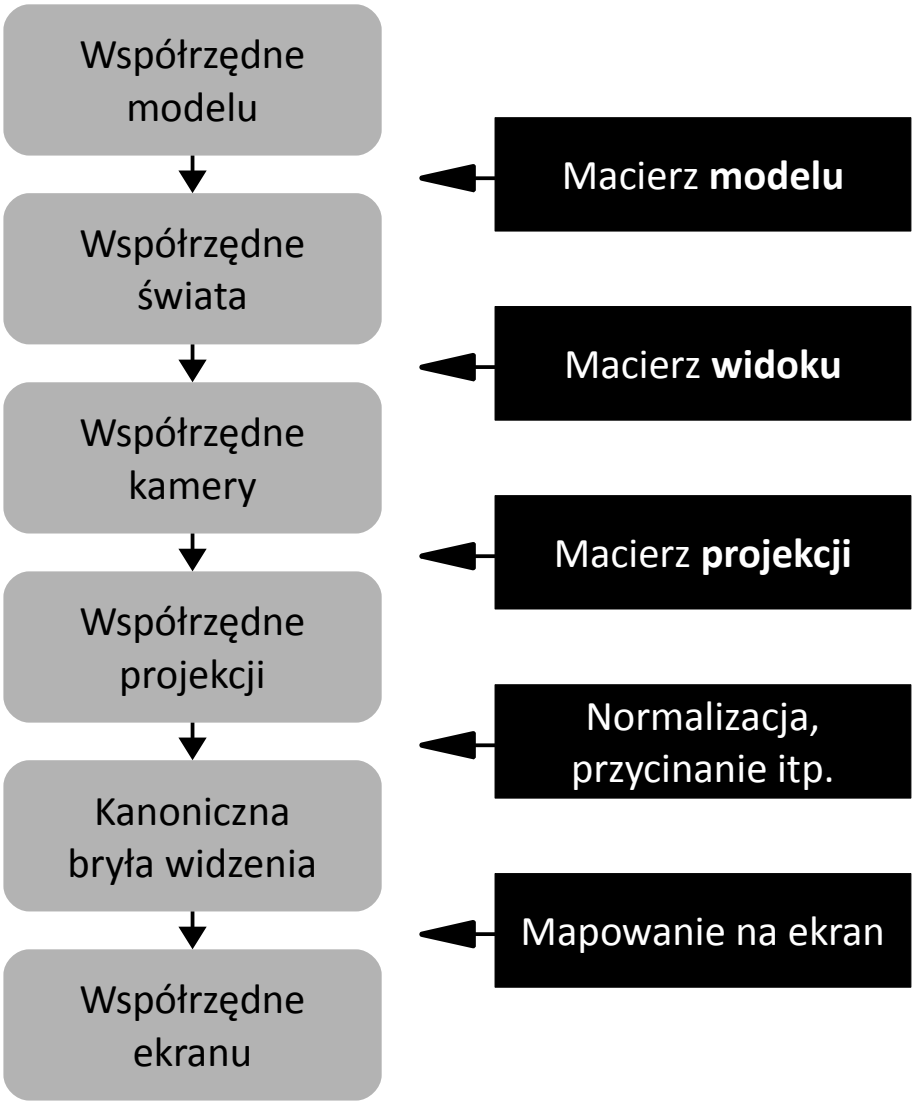
- **Macierz widoku**, a więc macierz związana z transformacją kamery, jest złożeniem odpowiednich translacji i rotacji
  - Czyli **iloczynem macierzy** odpowiadających takim transformacjom
- Macierz **projekcji perspektywicznej** wygląda następująco:

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2\times zFar\times zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

  - Gdzie:
    - $f = 1 / tan(fov / 2)$
    - $aspect$ = proporcje
    - $zNear$ = odległość bliskiej płaszczyzny przycinania
    - $zFar$ = odległość dalekiej płaszczyzny przycinania

Przekształcenia geometryczne

- Podczas procesu renderowania sceny dokonywane są **przejścia pomiędzy różnymi układami współrzędnych**



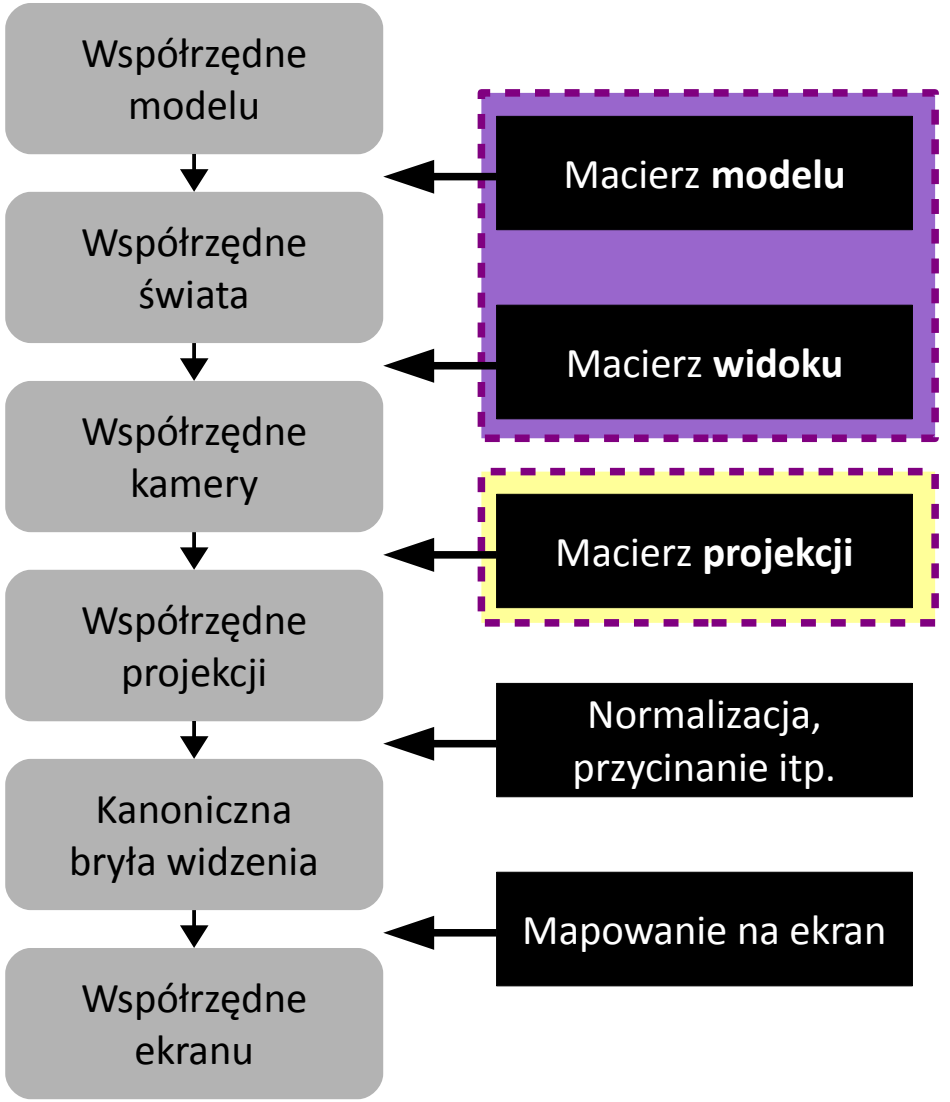
Transformacje w OpenGL

- Częścią stanu OpenGL są dwie podstawowe **macierze transformacji**:
  - Macierz **Modelu-Widoku** (**GL_MODELVIEW**)
    - Łączy w sobie przekształcenia związane zarówno z przejściem **ze współrzędnych modelu do współrzędnych świata**, jak i **ze współrzędnych świata do współrzędnych kamery**
      - Czyli uwzględnia zarówno **przekształcenie konkretnego obiektu na scenie**, jak i **położenie i obrót kamery**
  - Macierz **Projekcji** (**GL_PROJECTION**)
    - Uwzględnia **rodzaj projekcji** (perspektywiczna/prostokątna), **kąt widzenia**, **płaszczyzny przycinania**, **proporcje**.

- Tylko **jedna** z tych macierzy jest w danym momencie **edytowalna**
  - Wyboru aktualnie edytowanej macierzy dokonuje się za pomocą funkcji **glMatrixMode**(*matrix*)
    - *matrix* – **GL_MODELVIEW** lub **GL_PROJECTION**

Transformacje w OpenGL

- Umiejscowienie macierzy transformacji w potoku przetwarzania geometrii

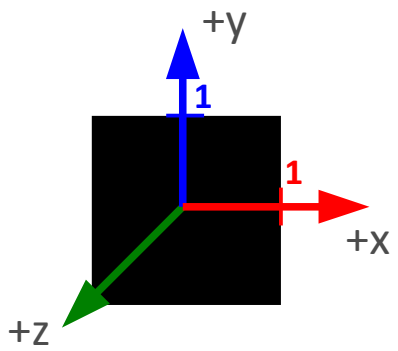


Transformacje w OpenGL

- Aktualnie wybraną do edycji macierz transformacji modyfikujemy za pomocą funkcji:
  - **glLoadIdentity**()
    - **Zastępuje** zawartość **macierzą jednostkową**
  - **glLoadMatrixf**(*float ***v***)
    - **Zastępuje** zawartość wartościami 16-elementowej tablicy
  - **glMultMatrixf**(*float ***v***)
    - **Mnoży** zawartość macierzy przez macierz ***v***
  - **glTranslatef**(*float **x**, float **y**, float **z***)
    - **Mnoży** zawartość macierzy przez macierz odpowiadającą translacji o wektor ***[x; y; z]***
  - **glRotatef**(*float **angle**, float **x**, float **y**, float **z***)
    - **Mnoży** zawartość macierzy przez macierz odpowiadającą rotacji o ***angle*** stopni wokół wektora ***[x; y; z]***
    - Warto stosować z wersorami, np. *[0; 1; 0]*
  - **glScalef**(*float **xs**, float **ys**, float **zs***)
    - **Mnoży** zawartość macierzy przez macierz odpowiadającą skalowaniu ze skalami ***[xs; ys; zs]***

Transformacje w OpenGL

- Przykłady
  - Zakładamy, że hipotetyczna funkcja **DrawBlackSquare()** rysuje czarny kwadrat będący częścią płaszczyzny *XY* i rozciągający się w zakresie *-1..1*:



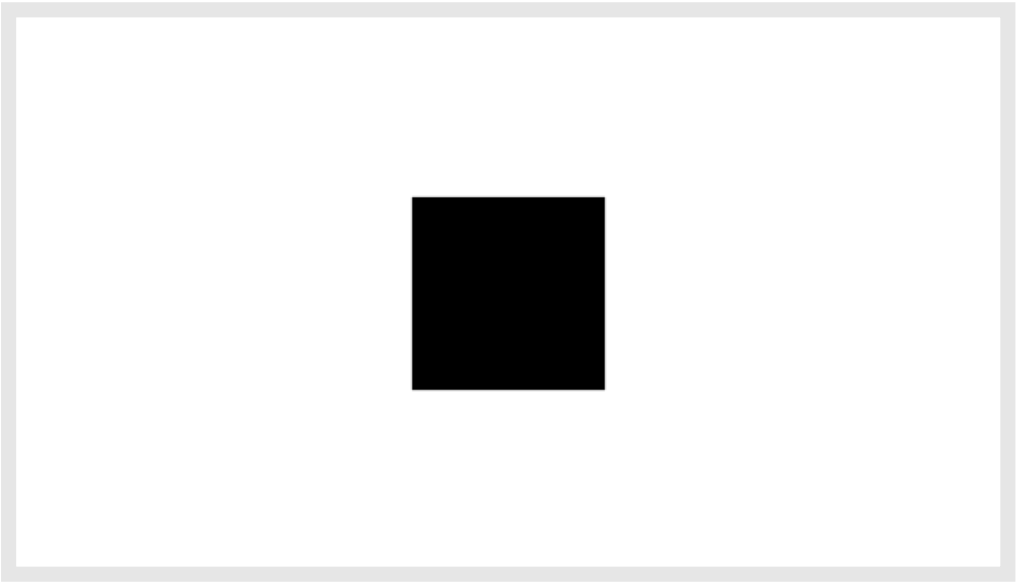
    - Na razie nie wnikamy, jak tego dokonuje

  - Scenę obserwujemy z punktu *[0; 0; 5]*, patrząc w kierunku *[0; 0; -1]*, wektor pionu *[0; 1; 0]*
  - Projekcja perspektywiczna

Transformacje w OpenGL

- Przykłady

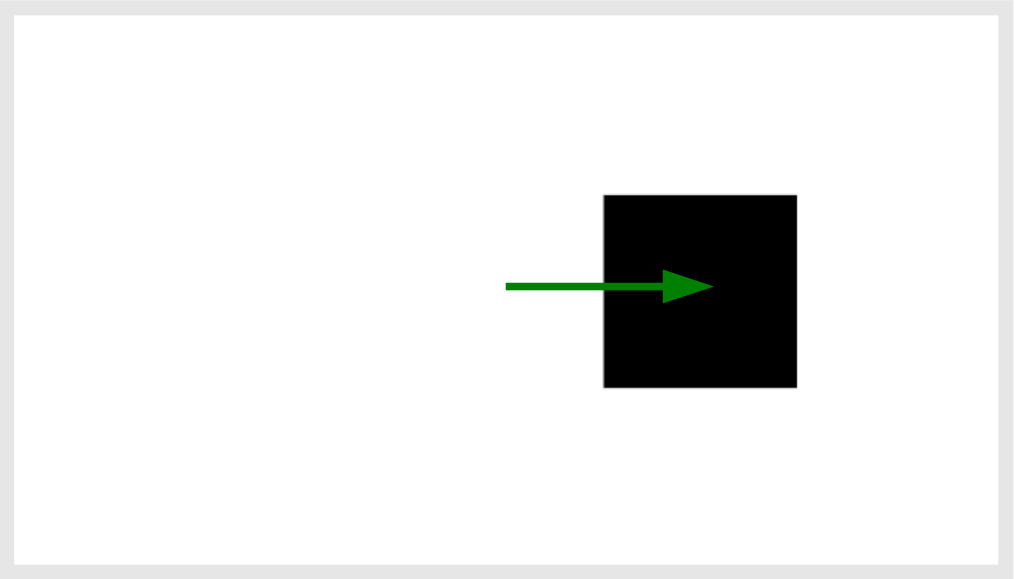
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

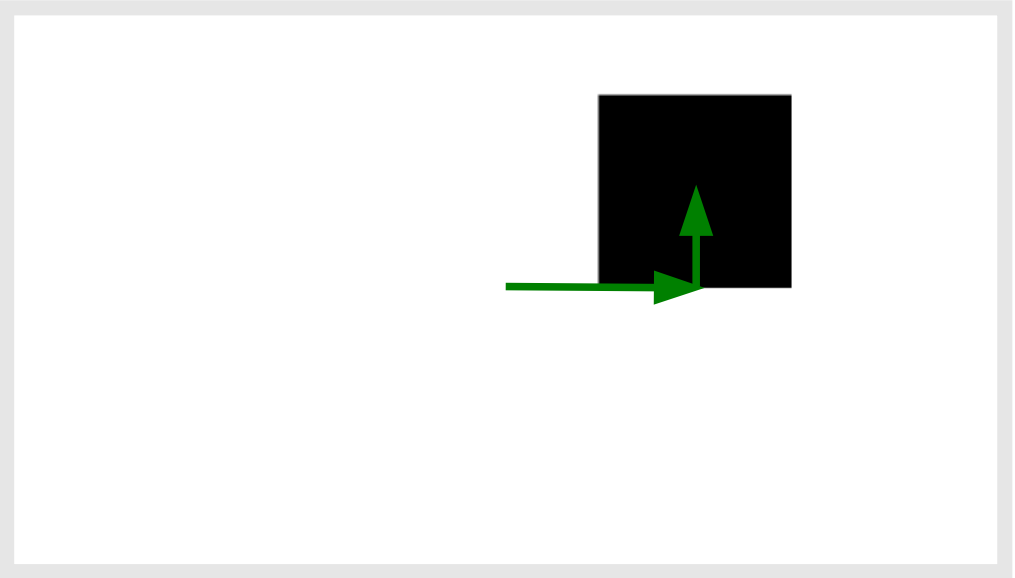
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(2.0f, 0.0f, 0.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

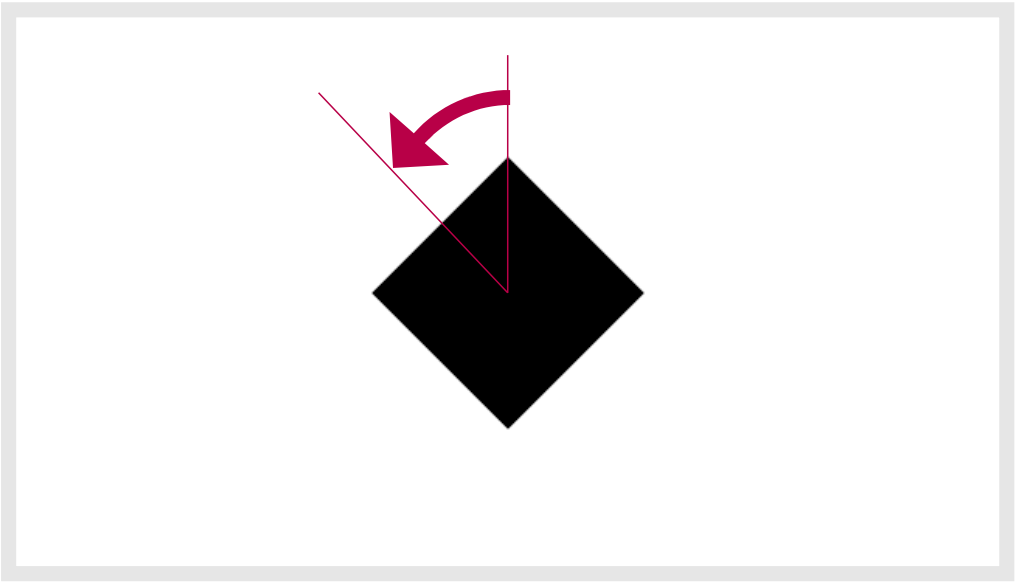
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(2.0f, 0.0f, 0.0f);
glTranslatef(0.0f, 1.0f, 0.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

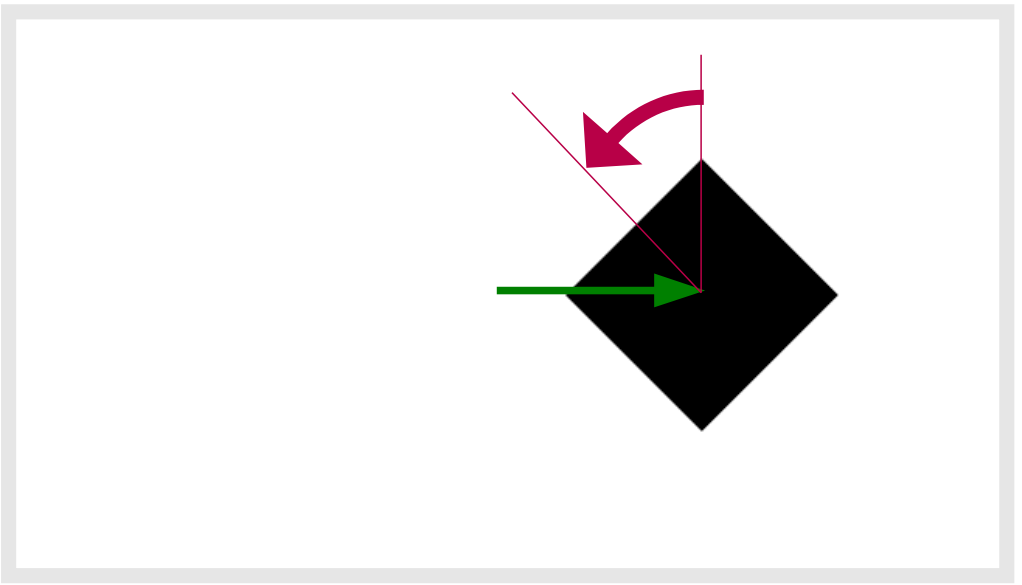
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

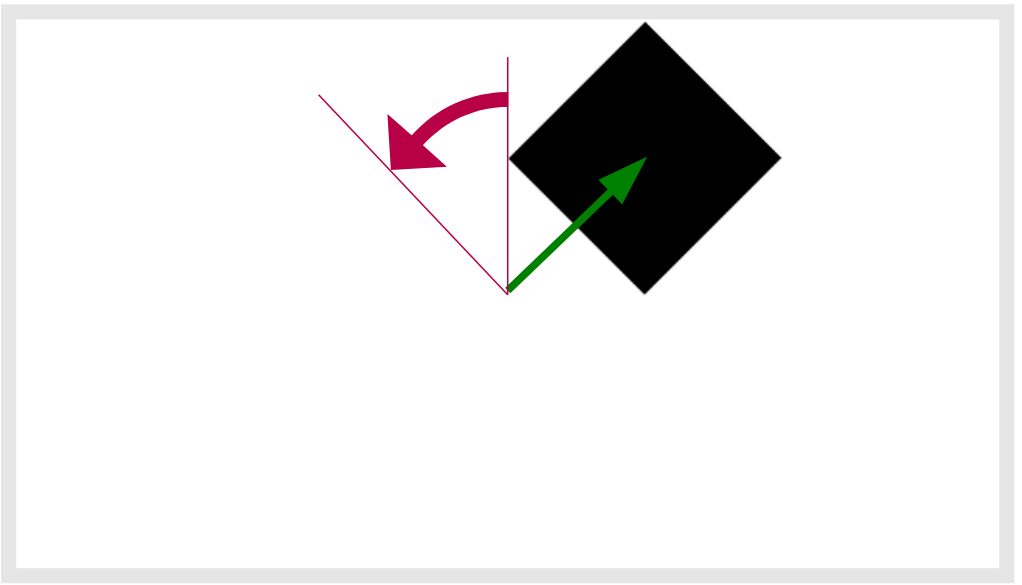
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(2.0f, 0.0f, 0.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

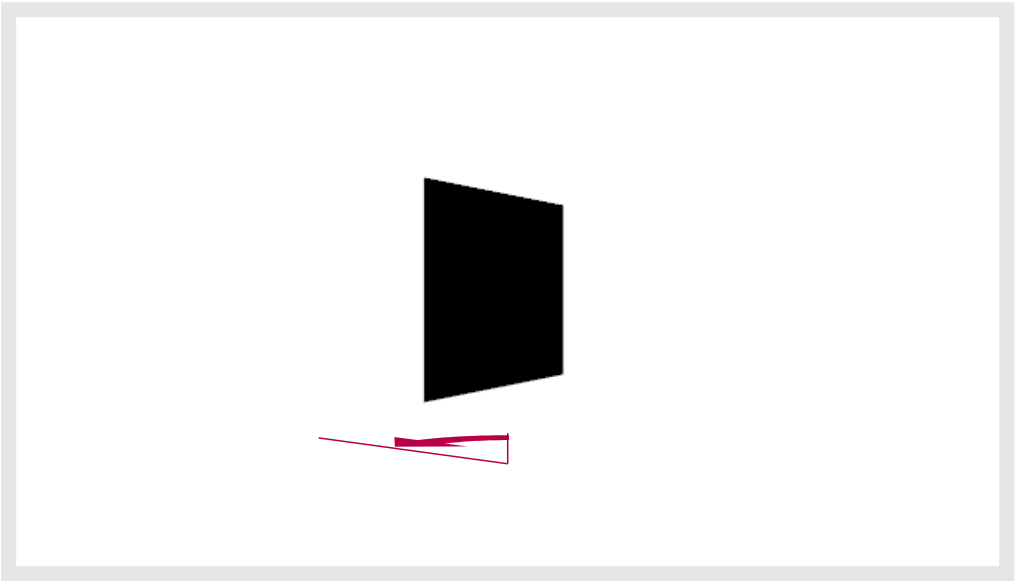
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glTranslatef(2.0f, 0.0f, 0.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

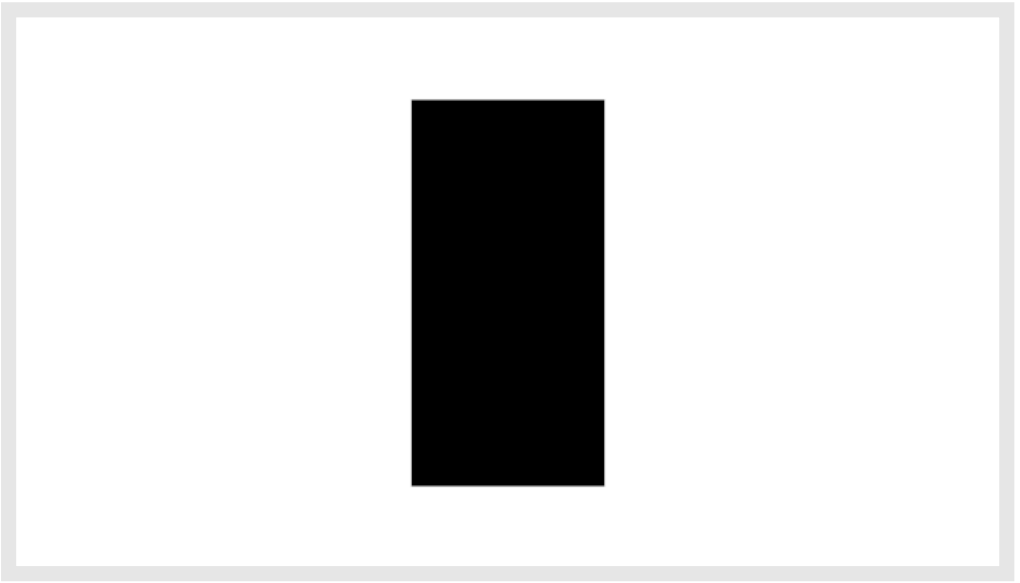
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0f, 0.0f, 1.0f, 0.0f);
DrawBlackSquare();
```



Transformacje w OpenGL

- Przykłady

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.0f, 2.0f, 1.0f);
DrawBlackSquare();
```

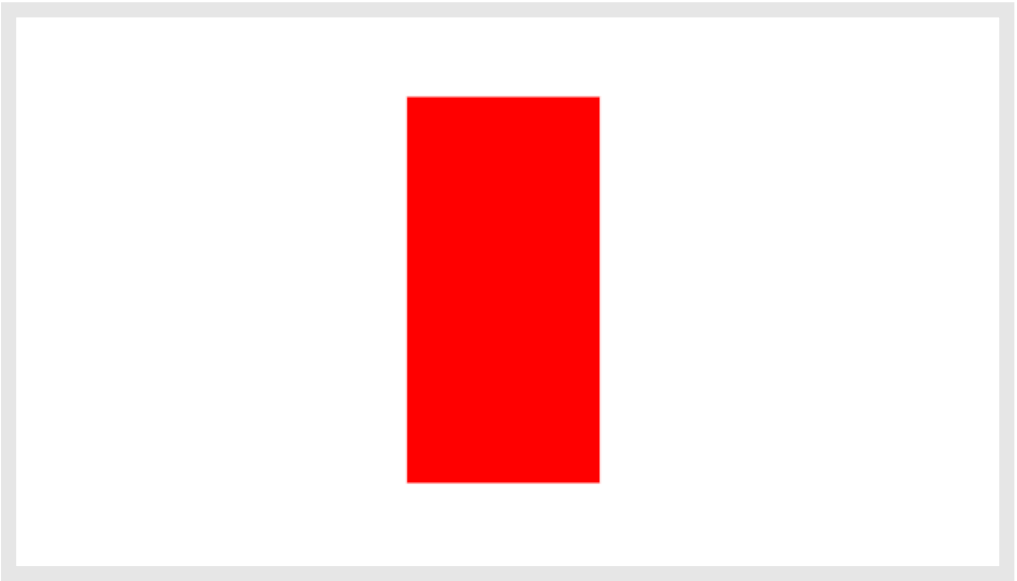


Transformacje w OpenGL

- Przykłady

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.0f, 2.0f, 1.0f);
DrawBlackSquare();
DrawRedSquare();
```

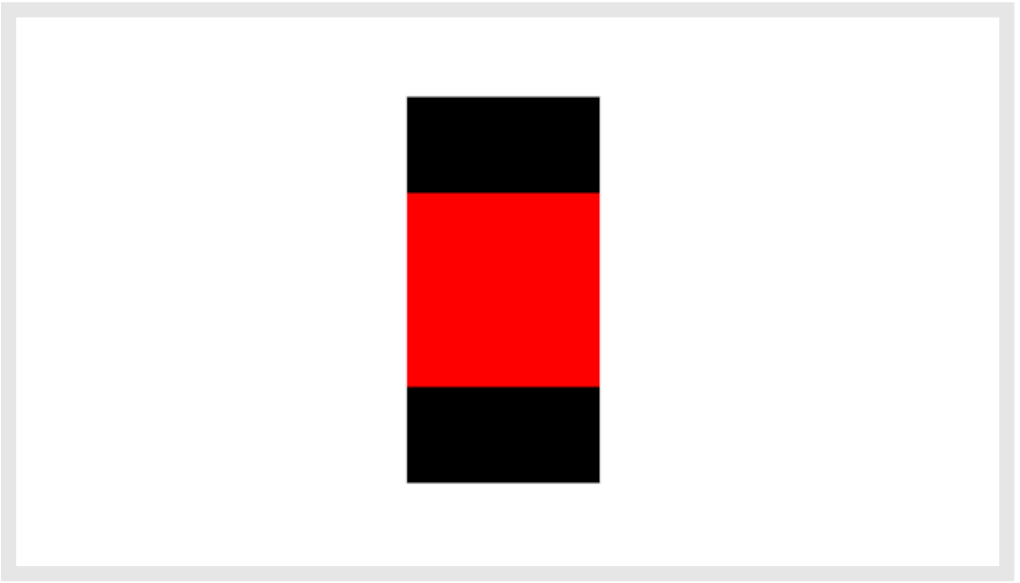
  - Stan macierzy pozostaje zmieniony!



Transformacje w OpenGL

- Przykłady

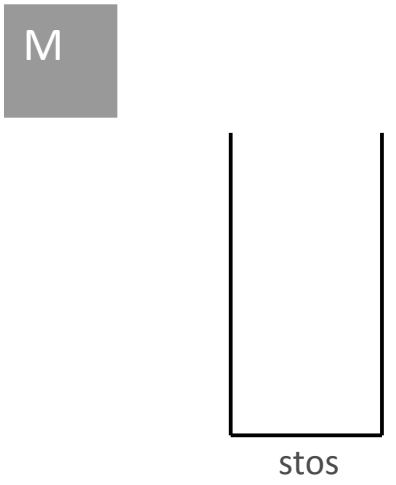
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix();
        glScalef(1.0f, 2.0f, 1.0f);
        DrawBlackSquare();
glPopMatrix();
DrawRedSquare();
```



Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

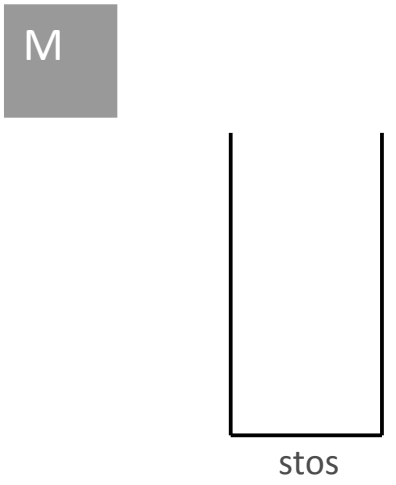


Podgląd rezultatu:

Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

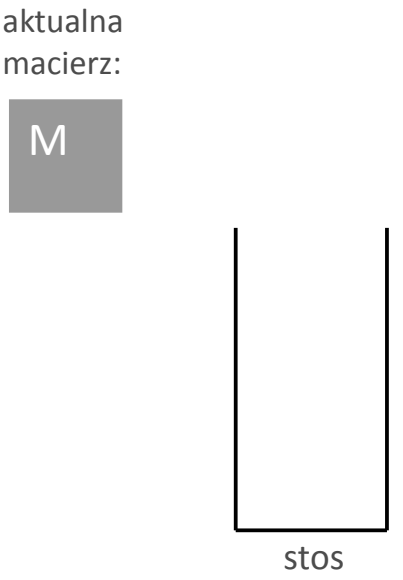


Podgląd rezultatu:

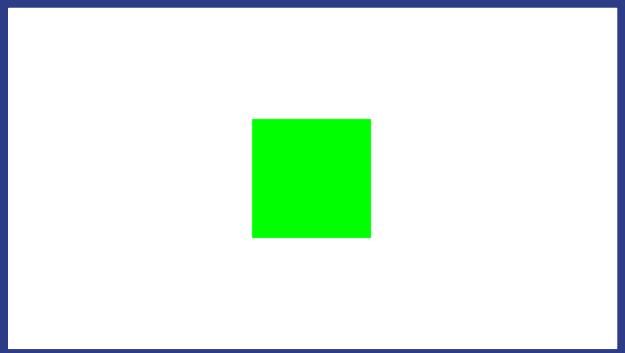
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```



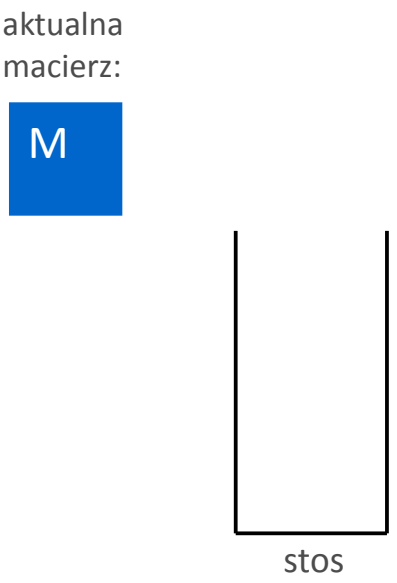
Podgląd rezultatu:



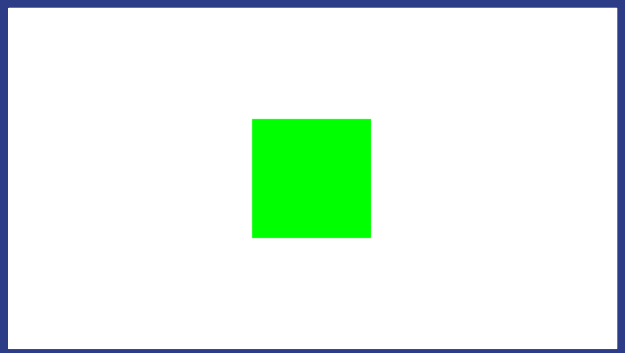
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```



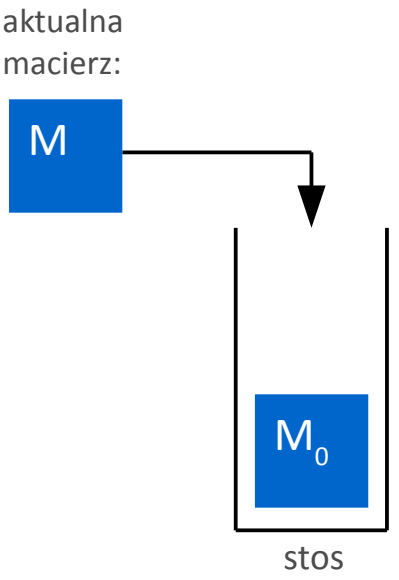
Podgląd rezultatu:



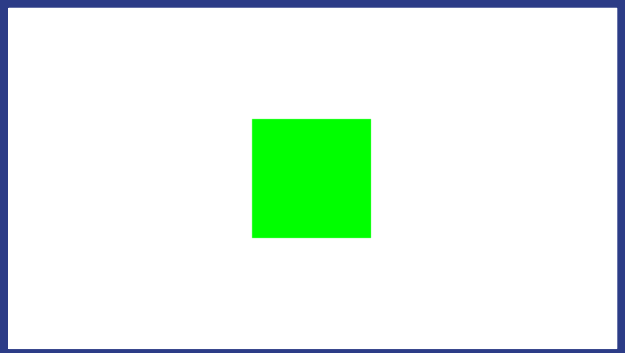
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```



Podgląd rezultatu:

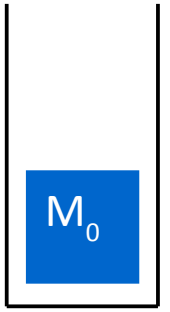


Transformacje w OpenGL

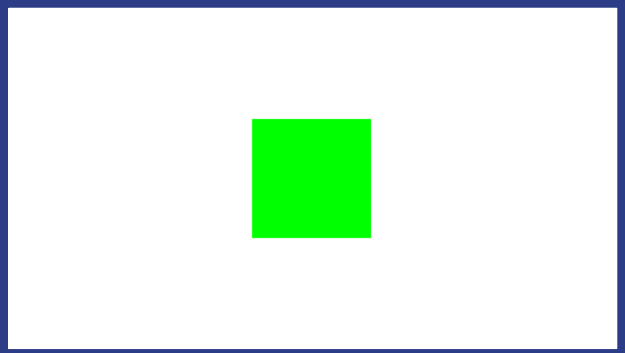
- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

aktualna macierz:



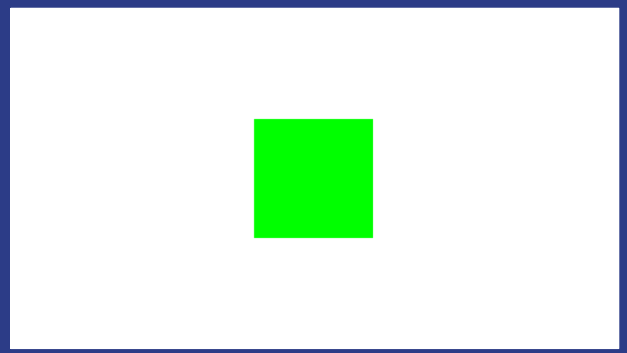
Podgląd rezultatu:



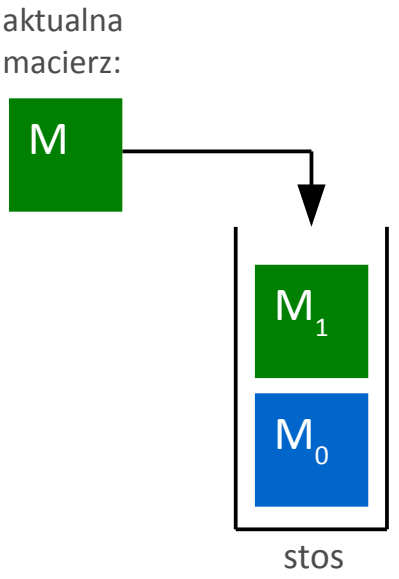
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

Podgląd rezultatu:



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

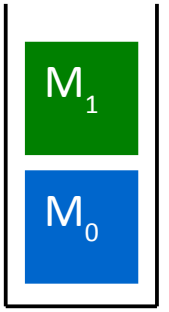


Transformacje w OpenGL

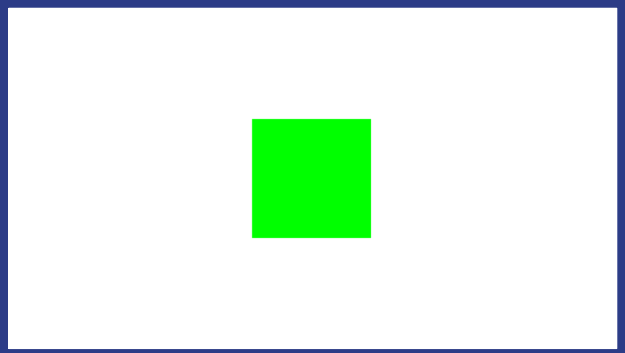
- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

aktualna macierz:



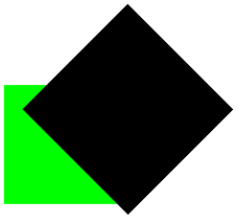
Podgląd rezultatu:



Transformacje w OpenGL

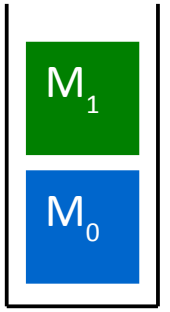
- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

Podgląd rezultatu:



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

aktualna macierz:

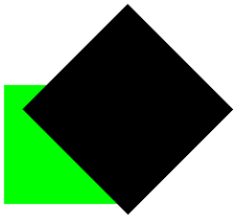


stos

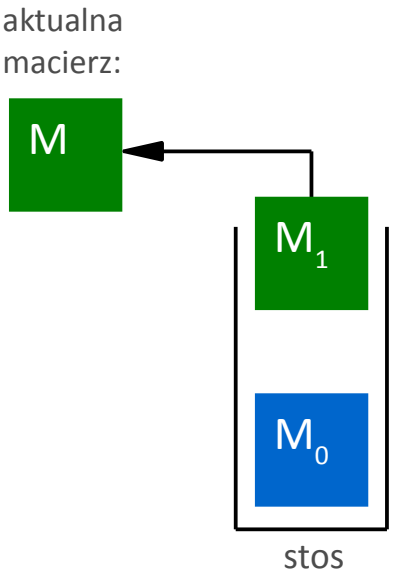
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

Podgląd rezultatu:



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

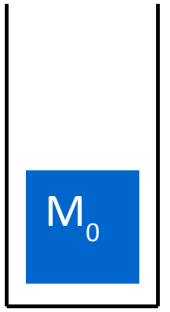


Transformacje w OpenGL

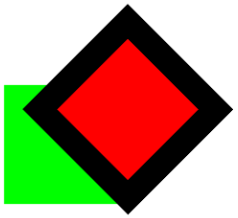
- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```

aktualna macierz:



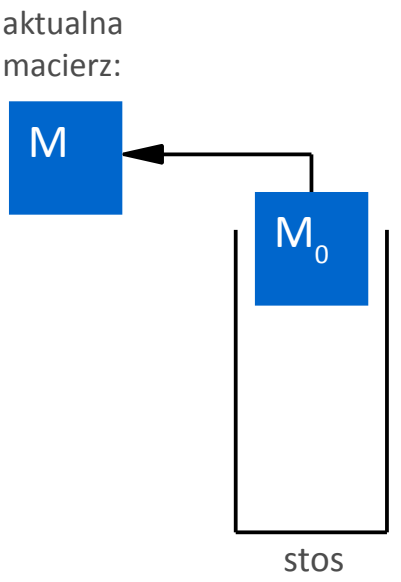
Podgląd rezultatu:



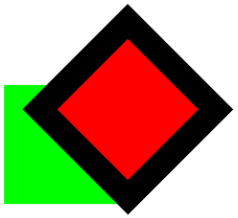
Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```



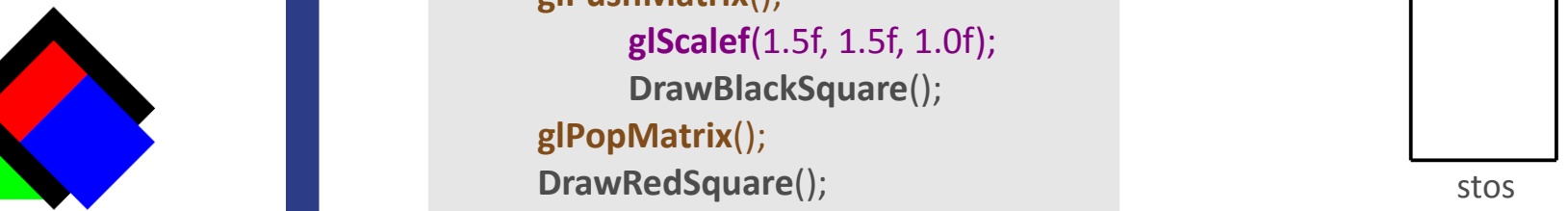
Podgląd rezultatu:



Transformacje w OpenGL

- **Zapamiętywanie** stanu macierzy **na stosie**
  - **glPushMatrix**() powoduje, że **aktualny stan** macierzy zostaje **zapamiętany na stosie**
  - **glPopMatrix**() **zdejmuje ze stosu** ostatnio zapamiętany stan i **przywraca go**
  - Możliwe jest wielokrotne **zagnieżdżanie**
    - Przy czym należy pamiętać, że zawsze przywracamy ostatni stan

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawGreenSquare();
glTranslatef(2.0f, 0.0f, -1.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glPushMatrix();
        glTranslatef(0.0f, 1.0f, 0.0f);
        glPushMatrix();
                glScalef(1.5f, 1.5f, 1.0f);
                DrawBlackSquare();
        glPopMatrix();
        DrawRedSquare();
glPopMatrix();
DrawBlueSquare();
```



Podgląd rezultatu:

Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Grafika Komputerowa

Wstęp do syntezy trójwymiarowej grafiki komputerowej

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok