Politechnika Wrocławska

Grafika komputerowa i komunikacja człowiek-komputer

Laboratorium nr 2

Podstawy OpenGL, grafika 2D

Szymon Datko

szymon.datko@pwr.edu.pl

Wydział Informatyki i Telekomunikacji,
Politechnika Wrocławska

semestr zimowy 2022/2023

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Cel ćwiczenia

1. Zapoznanie się z podstawowymi elementami grafiki komputerowej.
2. Zgrubne zrozumienie procesu powstawania obrazu w komputerze.
3. Oswojenie się z interfejsem OpenGL, na przykładach 2-wymiarowych.

Politechnika Wrocławska

Dawno, dawno temu w naszej galaktyce

- Lata '70 – '90 ubiegłego wieku:
  - ręczne rysowanie obrazów bezpośrednio w pamięci,
  - brak jednolitego standardu definiowania grafiki,
  - kłopoty ze złożonością i utrzymaniem kodu, a jednocześnie wzrost zapotrzebowania na bardziej skomplikowane sceny.
- OpenGL zapoczątkowany przez firmę Silicon Graphics Inc.:
  - producenta wysokowydajnych stacji graficznych,
  - początkowo jako ich własnościowy system IRIS GL,
    - *Integrated Raster Imaging System Graphics Library*,
  - ostatecznie udostępniony jako **otwarty standard** (1992), po usunięciu elementów własnościowych z pierwotnego tworu.

Politechnika Wrocławska

Czym jest OpenGL?

- Formalnie, wbrew nazwie, nie jest to biblioteka:
  - Interfejs Programowania Aplikacji (ang. *API*),
  - **specyfikacja** w jaki sposób mają być realizowane instrukcje,
  - odpowiednią implementację dostarczają **sterowniki** sprzętu.
- Standard otwarty i niezależny od platformy sprzętowej.
- Wspierany przez wiodące systemy operacyjne.
- Obecnie nad rozwojem standardu czuwa Khronos group,
  - konkretnie **opengl architectural review board** (arb).
- Dotyczy wyłącznie generowania obrazów, niczego ponadto.

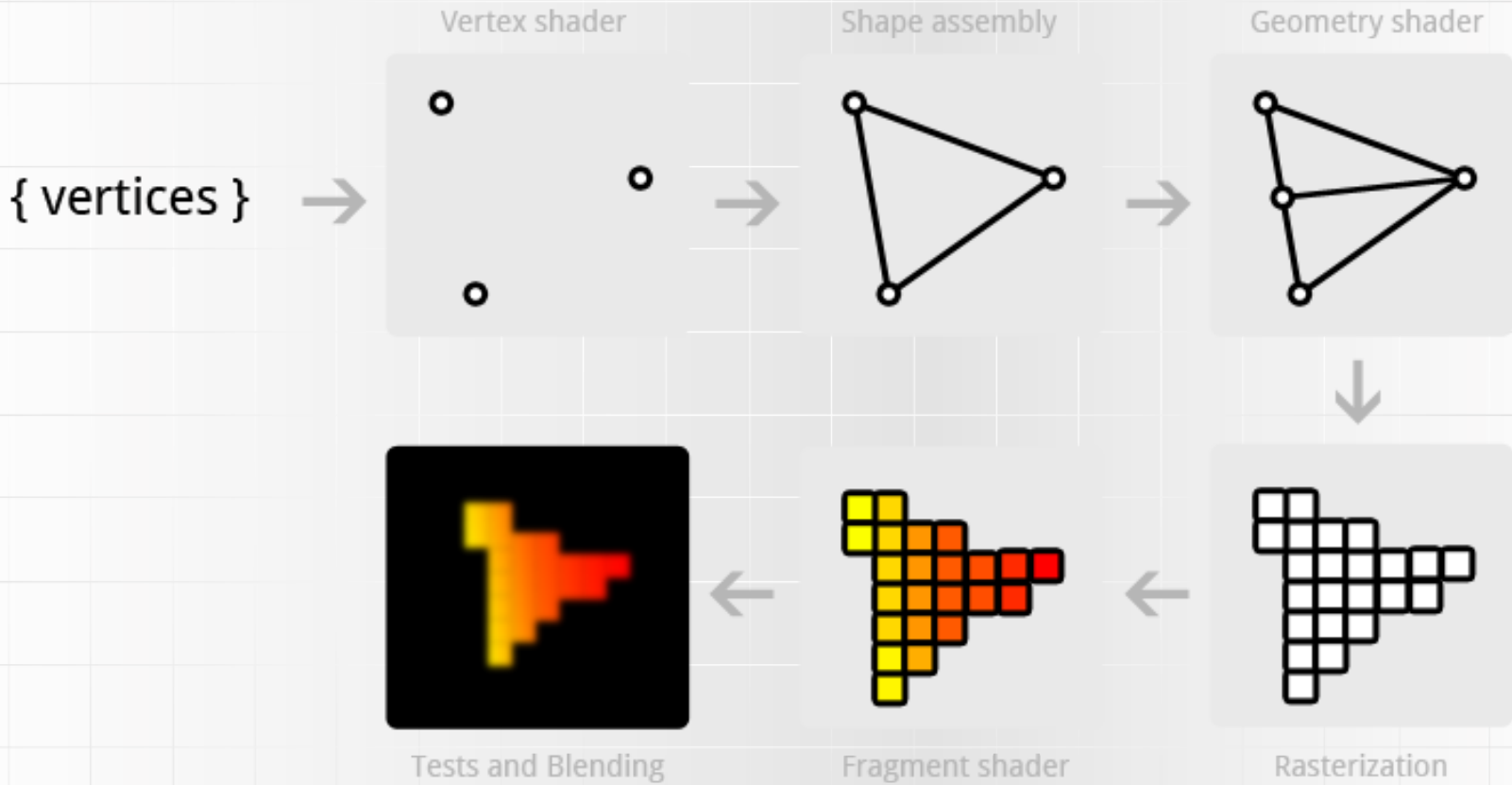
Politechnika Wrocławska

Podstawowe pojęcia

- **Wierzchołek** – punkt w przestrzeni, używany w prymitywach.
- **Prymityw** – podstawowa jednostka renderingu w OpenGL,
  - najważniejsze to między innymi: punkt, linia i trójkąt;
  - również warianty paskowe, pętlowe i wiatrakowe powyższych.
- **Renderowanie** – proces budowania obrazu w komputerze,
  - w ogólności: proces przekształcania czegoś do innej formy;
  - ang. *przekazać*, *oddać*, *sprawić*, *przedstawić*.
- **Rasteryzacja** – przekształcenie prymitywów w zbiór pikseli.
- **Piksel** – najmniejsza jednostka wizualna na wyświetlaczu.
- **Ramka** – zbiór pikseli, pojedynczy pełen obraz.

Powstawanie grafiki w komputerze

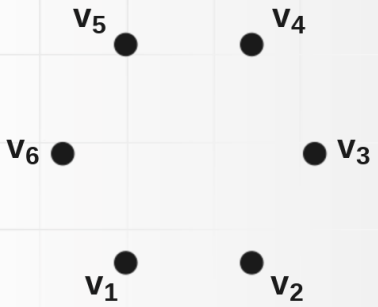
To wszystko to po prostu matematyka! ;-)



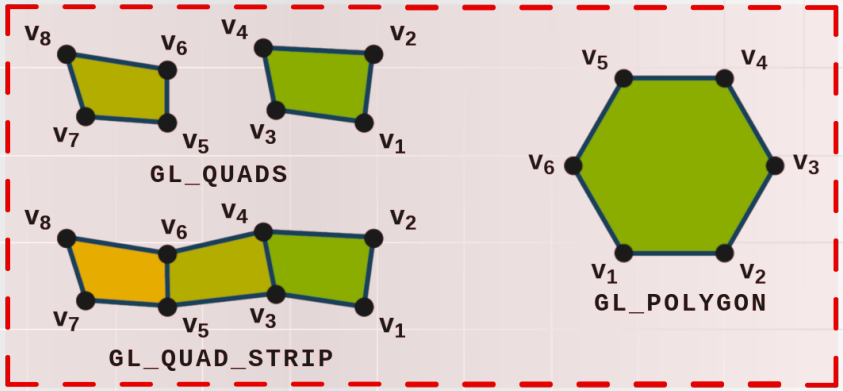
Uwaga! Grafika przedstawia elementy współczesnego potoku graficznego, nieobecne w omawianym tu *Legacy OpenGL*!

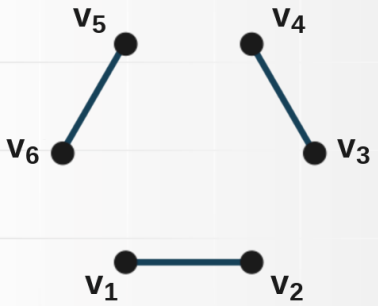
Źródło obrazka: https://open.gl/drawing

Rodzaje prymitywów

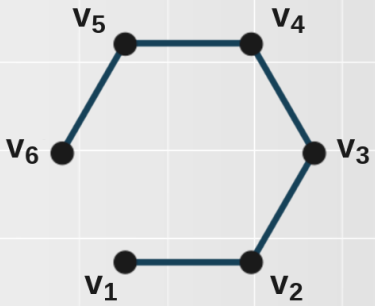


GL_POINTS

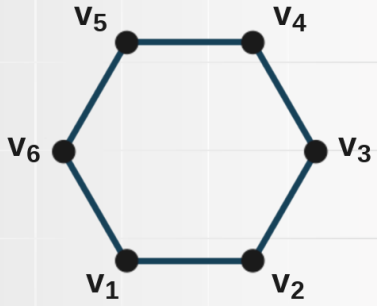




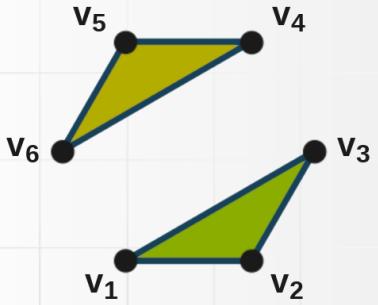
GL_LINES



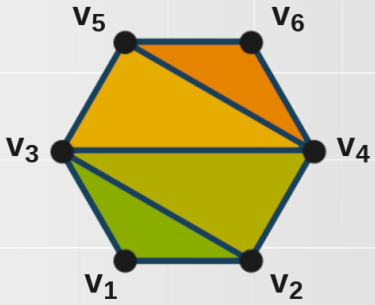
GL_LINE_STRIP



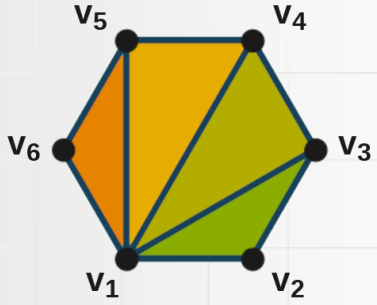
GL_LINE_LOOP



GL_TRIANGLES



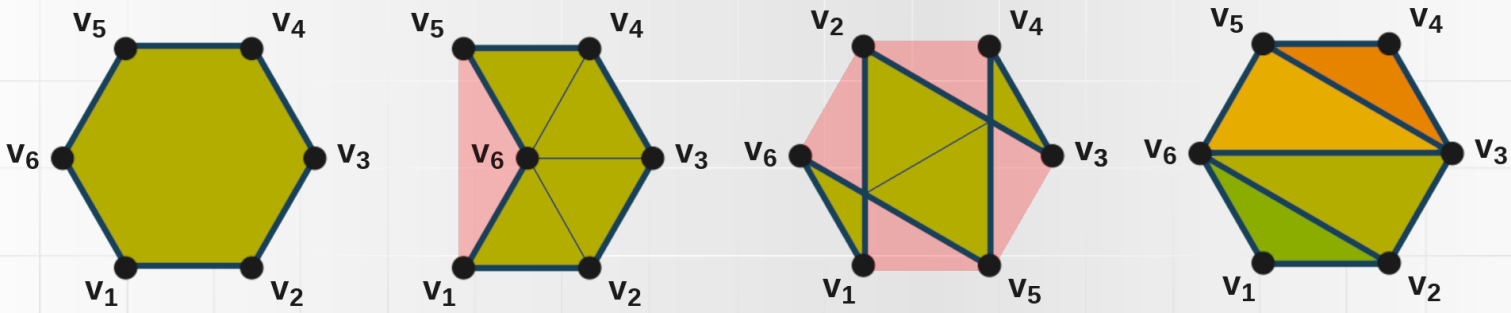
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

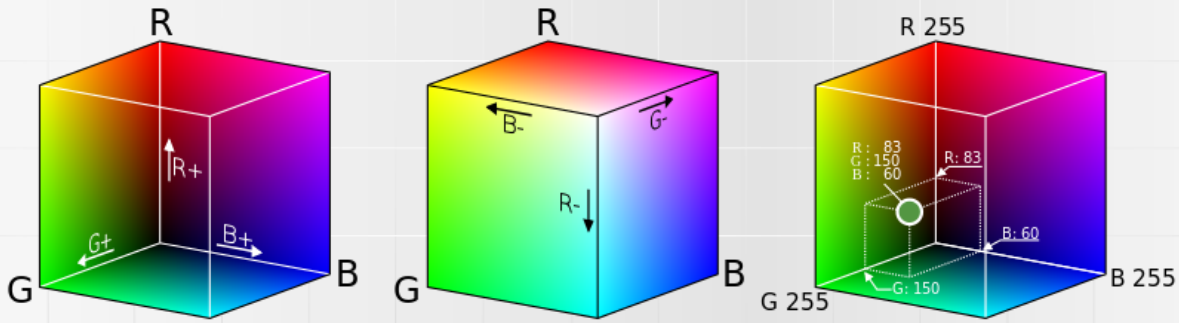
Wypukłość figur

- **Problem**: w jaki sposób pokolorować zaznaczone niżej fragmenty?
- Obecnie (OpenGL $\geq$ 3.1) nie korzysta się już ze złożonych prymitywów,
  - takich jak **GL_QUADS**, **GL_QUAD_STRIP** i **GL_POLYGON**.
- Zamiast tego stosujemy **siatki trójkątów** – czyli figur zawsze wypukłych.
- Efekty tej zmiany:
  - jednoznaczność opisu i optymalizacja działania kart graficznych,
  - nieco większy nakład pracy do wykonania dla programisty.



Modele kolorów

- **Kolor** jest pewnym subiektywnym wrażeniem, każdy postrzega go inaczej.
- Zwykle ludzie posługują się mało precyzyjną reprezentacją barw:
  - czerwony, khaki, ciepły piasek, błękit narodów zjednoczonych.
- Przy pracy z kolorem lepiej jest posłużyć się jednoznacznymi zapisami,
  - jednym z takich modeli jest na przykład model **RGB**,
  - barwę opisujemy przy pomocy trzech ustalonych kolorów $\rightarrow$ wektor,
  - zapisujemy nasycenie koloru **czerwonego**, **zielonego** i **niebieskiego**.
- 3 wektory ortogonalne modelu rozpinają przestrzeń możliwych odcieni,
  - głębia kolorów określa liczbę przejść pomiędzy skrajnymi kolorami,
  - wykonalne jest płynne przejście między dwoma odcieniami!



Źródło obrazka: https://commons.wikimedia.org/wiki/File:RGB_color_cube.svg

Słowo na temat nazewnictwa funkcji

- Nazwy funkcji biblioteki OpenGL rozpoczynają się od **gl**.
  - Ta sama konwencja dotyczy wielu bibliotek pomocniczych.
  - **glu** – dla funkcji z zestawu OpenGL Utility Library.
  - **glfw** – dla funkcji z biblioteki Graphics Library Framework.
- Dalej następuje **określenie**, mówiące co dana funkcja robi.
  - Zapis według konwencji tak zwanego *Camel Case*.
- Niektóre funkcje mają kilka możliwych wariantów wywołania.
  - Wtedy na końcu nazwy dodaje się specyficzną adnotację.
  - Określa ona **liczbę** i **typy** przyjmowanych argumentów.
    - **b**, **s**, **i**, **ub**, **us**, **ui** – liczby całkowite (ze znakiem lub bez),
    - **f**, **d** – liczby zmiennoprzecinkowe (różnej precyzji).
    - **v** – tablica (pisane na końcu = funkcja przyjmuje 1 argument).
- Przykłady dla funkcji ustawiającej kolor wierzchołka:
  - `glColor3ub()` – trzy liczby typu *unsigned byte* (wartości: 0 – 255),
  - `glColor3fv()` – tablica trzech liczb *float* (wartości dla koloru: 0.0 – 1.0).
  - Więcej: https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glColor.xml.

Politechnika Wrocławska

Dodatkowe narzędzia i często stosowane biblioteki

- Tworzenie kontekstu OpenGL.
  - Struktura, przechowująca wszystkie dane o stanie i pamięci OpenGL.
  - Okno, w którym wygenerowane ramki obrazu będą wyświetlane.
    - W naszym wypadku odpowiada za to biblioteka **GLFW**.
- Wczytywanie rozszerzeń OpenGL.
  - Pozwala skorzystać w elementów współczesnego OpenGL.
- Zestawy funkcji matematycznych.
  - Przydatne przy definiowaniu transformacji we współczesnym OpenGL.
- Ładowanie i przetwarzanie tekstur.
- Odczytywanie gotowych modeli 3D z plików.
- Więcej informacji:
  - https://www.khronos.org/opengl/wiki/Related_toolkits_and_APIs

Politechnika Wrocławska

Przygotowanie do pracy w systemie macOS

- Większość potrzebnych rzeczy powinna być domyślnie zainstalowana.
- Jeśli w systemie brakuje interpretera języka Python w wersji 3:
  - `brew install python3`
  - Narzędzie **brew**: `https://brew.sh`.
- Zainstalowanie potrzebnych dowiązań w języku Python:
  - `pip3 install --user PyOpenGL glfw`
- Uruchomienie z poziomu konsoli:
  - `python3 nazwa-pliku-ze-skryptem.py`

Przygotowanie do pracy w systemie Linux

- Niezbędne pakiety to biblioteka **GLFW** oraz dowiązania **PyOpenGL**.
- Aby zainstalować je w systemie **Ubuntu Linux** i podobnych:
  - `sudo apt update`
  - `sudo apt install python3 python3-pip libglfw3`
  - `pip3 install PyOpenGL glfw`
- Aby zainstalować je w systemie **Arch Linux** i podobnych:
  - `sudo pacman -Syu python python-opengl python-glfw`
- Uruchomienie z poziomu konsoli:
  - `python3 nazwa-pliku-ze-skryptem.py`

Politechnika Wrocławska

Przygotowanie do pracy w systemie Windows

- Pobranie i zainstalowanie interpretera języka Python.
  - Strona: https://www.python.org/downloads/windows/.
  - W instalatorze zaznaczyć opcję *Add Python to PATH*.
- Zainstalowanie potrzebnych dowiązań w języku Python:
  - W programie **cmd**: `pip install PyOpenGL glfw`
- Uruchomienie z poziomu wiersza poleceń:
  - `python nazwa-pliku-ze-skryptem.py`
- Visual Studio Code z wtyczką dla języka Python dobrze się sprawdza jako edytor i narzędzie do uruchamiania przygotowywanych skryptów.

Przykładowy program (1/2)

```
1  #!/usr/bin/env python3
2  import sys
3
4  from glfw.GLFW import *
5
6  from OpenGL.GL import *
7  from OpenGL.GLU import *
8
9
10 def startup():
11     glClearColor(0.5, 0.5, 0.5, 1.0)
12
13
14 def shutdown():
15     pass
16
17
18 def render(time):
19     glClear(GL_COLOR_BUFFER_BIT)
20     glFlush()
21
22
23 def main():
24     # patrz: kolejny slajd
```

Przykładowy program (1/2) – omówienie

- Linia **1** to tak zwany *shebang* – ścieżka do interpretera.
  - Więcej informacji na ten temat – na kursie Systemów Operacyjnych 2 :-)
- Linie **2**..**7** obejmują załadowanie niezbędnych bibliotek.
  - Wyrażenia `from ... import *` służą ułatwieniu na potrzeby zajęć.
  - Przez to kod może wyglądać prawie identycznie, jak przykłady w języku **C**.
  - Bardzo przepraszam wszystkich znawców Pythona za tę profanację ;-)
- Linie **10**..**15** to funkcje pomocnicze, docelowo wykonywane jednorazowo.
  - Wprowadzone zostały dla przejrzystości kodu na kolejnych zajęciach.
  - W linii **11** ustawiamy wartość koloru, do jakiego będzie czyszczony bufor.
  - Linia **15** zawiera instrukcję, która nic nie robi – tak zwany *placeholder*.
- Linie **18**..**20** definiują funkcję, która rysuje pojedynczą klatkę obrazu.
  - W tym przykładzie jest to wyczyszczenie ramki w pamięci – `glClear()`.
  - Następnie zawartość pamięci jest przesyłana do wyświetlenia – `glFlush()`.
  - **To głównie tę funkcję będziemy modyfikowali w ramach naszych zajęć!**
  - Funkcja ta powinna być możliwe szybka – unikamy zbędnych obliczeń, itd.

Przykładowy program (2/2)

```
    ⋮
def main():
    if not glfwInit():
        sys.exit(-1)

    window = glfwCreateWindow(400, 400, __file__, None, None)
    if not window:
        glfwTerminate()
        sys.exit(-1)

    glfwMakeContextCurrent(window)
    glfwSwapInterval(1)

    startup()
    while not glfwWindowShouldClose(window):
        render(glfwGetTime())
        glfwSwapBuffers(window)
        glfwWaitEvents()
    shutdown()

    glfwTerminate()


if __name__ == '__main__':
    main()
```

Przykładowy program – omówienie (2/2)

- Linie **23**..**42** to najistotniejsza funkcja programu.
  - Najpierw następuje przygotowanie biblioteki GLFW – `glfwInit()`.
    - Jeśli to z jakiegoś powodu się nie powiedzie, kończymy program.
  - Następnie utworzone zostaje okno, w którym wyświetlany będzie obraz.
    - Pierwsze dwa argumenty to początkowy rozmiar okna,
    - trzeci argument to tytuł okna – tutaj będzie to nazwa naszego pliku,
    - dwa ostatnie argumenty nie są dla nas interesujące na etapie zajęć.
    - W przypadku niepowodzenia, tu również kończymy program.
  - Kolejne wywołanie określa miejsce aktywnego obecnie kontekstu OpenGL,
    - czyli w którym miejscu generowany będzie przez nas obraz,
    - biblioteka GLFW pozwala stworzyć kilka okien na raz i je przełączać.
  - `glfwSwapInterval()` włącza tak zwaną synchronizację pionową.
    - Wpływa na funkcję `glfwSwapBuffers()`, ogranicza szybkość.
  - Dalej następuje główna część programu, powtarzana do zamknięcia okna.
    - W pętli wykonujemy funkcję `render()` i podmieniamy ramki obrazu.
    - Dodatkowo przetworzone zostaną zaistniałe zdarzenia okien i wejść.
- Linie **45**..**46** odpowiadają za uruchomienie głównej funkcji.
  - Wywołanie to następuję tylko przy bezpośrednim odpaleniu skryptu.
  - Przydatne jest to na wyższym etapie wtajemniczenia w mowie węży :-)

Przygotowanie rzutni

- Domyślna przestrzeń rysowania w OpenGL obejmuje zakres [-1.0; 1.0] każdej osi.
- Poniższy kod przekształca ten przedział do [-100.0; 100.0] dla osi X i Y.
- Aby kod zadziałał, zaraz za `glfwMakeContextCurrent(window)` należy dodać:
  - `glfwSetFramebufferSizeCallback(window, update_viewport)`
  - Dodatkowo `update_viewport(None, 400, 400)` w funkcji `startup()`.

```
1| def update_viewport(window, width, height):
2|     if height == 0:
3|         height = 1
4|     if width == 0:
5|         width = 1
6|     aspectRatio = width / height
7|
8|     glMatrixMode(GL_PROJECTION)
9|     glViewport(0, 0, width, height)
10|    glLoadIdentity()
11|
12|    if width <= height:
13|        glOrtho(-100.0, 100.0, -100.0 / aspectRatio, 100.0 / aspectRatio,
14|                1.0, -1.0)
15|    else:
16|        glOrtho(-100.0 * aspectRatio, 100.0 * aspectRatio, -100.0, 100.0,
17|                1.0, -1.0)
18|
19|    glMatrixMode(GL_MODELVIEW)
20|    glLoadIdentity()
```

Rysowanie za pomocą wierzchołków

- Jako argument funkcji `glBegin()` wskazuje się prymityw do rysowania.
- Wywołanie `glVertex()` umieszcza wierzchołek w pamięci.
  - Prymityw jest rysowany po podaniu mu odpowiedniej liczby wierzchołków.
  - Niekompletne kształty są ignorowane i znikają z pamięci po `glEnd()`!
- Wywołanie `glColor()` może się znaleźć przed każdym `glVertex()`.
  - Kolor każdej części prymitywu będzie interpolowany z przestrzeni barw.

```
1  def render(time):
2      glClear(GL_COLOR_BUFFER_BIT)
3
4      glColor3f(0.0, 1.0, 0.0)
5      glBegin(GL_TRIANGLES)
6      glVertex2f(0.0, 0.0)
7      glVertex2f(0.0, 50.0)
8      glVertex2f(50.0, 0.0)
9      glEnd()
10
11     glColor3f(1.0, 0.0, 0.0)
12     glBegin(GL_TRIANGLES)
13     glVertex2f(0.0, 0.0)
14     glVertex2f(0.0, 50.0)
15     glVertex2f(-50.0, 0.0)
16     glEnd()
17
18     glFlush()
```

Politechnika Wrocławska

Koniec wprowadzenia.

Zadania do wykonania...

Politechnika Wrocławska

Zadania do wykonania (1)

Na ocenę **3.0** należy przygotować podstawowe środowisko pracy.

Wskazówki:

- zainstalować niezbędne narzędzia i uruchomić przykładowy kod,
  - kod lepiej pobrać ze strony prowadzącego, niż kopiować ze slajdów,
- przerobić kod – narysować trójkąt z każdym wierzchołkiem innego koloru,
- proszę pamiętać o ustawionych parametrach rzutni:
  - zakres na osi X: od -100.0 (lewa strona) do 100.0 (prawa strona),
  - zakres na osi Y: od -100.0 (dół okna) do 100.0 (góra okna),
  - punkt o współrzędnych $(X = 0, Y = 0)$ – w środku okna.

Zadania do wykonania (2)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **3.5** należy napisać funkcję rysującą prostokąt w podanym miejscu.

Wskazówki:

- nowa funkcja powinna przyjmować 4 argumenty:
  - położenie w osi X – x,
  - położenie w osi Y – y,
  - rozmiar pierwszego boku – a,
  - rozmiar drugiego boku – b;
- położenie (x, y) może wskazywać środek prostokąta lub jego wierzchołek,
  - punkt (x, y) określa się wtedy fachowo jako punkt początkowy,
    - ang. *origin point*,
  - na tej podstawie należy wyznaczyć współrzędne reszty wierzchołków bryły,
  - do narysowania prostokąta należy wykorzystać dokładnie dwa trójkąty,
- funkcję należy przykładowo wywołać w ramach `render()`.

Zadania do wykonania (3)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **4.0** należy wprowadzić losowość kolorów i deformacje w prostokącie.

Wskazówki:

- proszę rozbudować funkcję z poprzedniego zadania, na przykład:
  - dodać kolejny argument do funkcji – d – z domyślną wartością 0.0,
  - nowy argument powinien sterować stopniem deformacji,
  - można na przykład przeskalować rozmiary boków a i b;
- uzyskanie losowej wartości w języku Python:
  - załadowanie biblioteki: `import random`;
  - przykładowe wywołanie: `random.random()`;
  - przydatne może być także użycie: `random.seed(...);`
  - oficjalna dokumentacja: https://docs.python.org/3/library/random.html;
  - proszę pamiętać o zakresach wybranego wariantu funkcji `glColor()`;
- w funkcji `render()` umieścić przykładowe wywołanie.

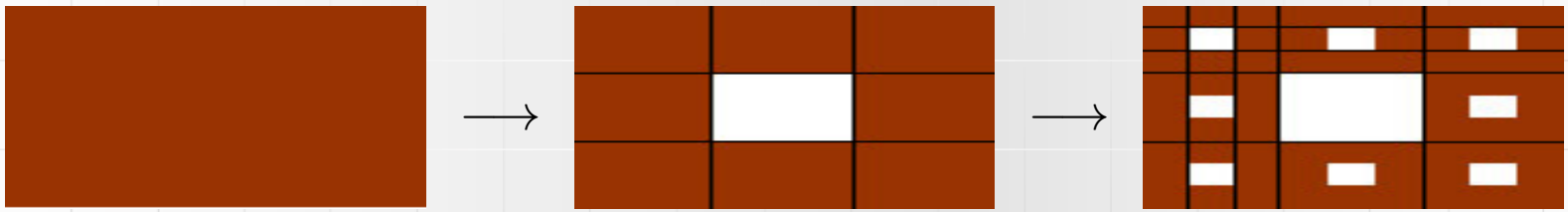
Zadania do wykonania (4)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **4.5** należy narysować fraktal – prostokątny dywan Sierpińskiego.

Wskazówki:

- zasadniczo istnieją dwa podejścia do narysowania tego fraktalu:
  - rysować poszczególne małe prostokąty w wyznaczonych miejscach, lub
  - narysować duży prostokąt i pomniejsze w miejscach "wycięć";
- wykorzystać funkcje z poprzednich przykładów:
  - najpierw narysować zarys fraktalu z ręcznie rozmieszczonych brył,
    - tak wyznaczymy interesujące nas współrzędne prostokątów,
  - następnie ubrać całość w funkcję rekurencyjną,
    - powtórzyć rysowanie w wyznaczonych współrzędnych.
    - z każdym stopniem rekurencji pomniejszać rozmiary boków;
- stopień samopodobieństwa powinien być parametrem programu.



Zadania do wykonania (5)

(po zrealizowaniu zadania poprzedniego)

Na ocenę **5.0** należy narysować drugi fraktal.

Wskazówki:

- wybrać jeden z przykładów zaproponowanych jako "zadania domowe",
  - dokument znajduje się na stronie prowadzącego,
  - interesują nas po prostu opisane tam pomysły na generację fraktali,
  - nie trzeba implementować różnych wariantów tego samego fraktala,
- alternatywnie można wykonać poprzedni fraktal w wariancie iteracyjnym,
  - warto zastanowić się nad dziedziną dozwolonych współrzędnych.