WPROWADZENIE

1. Czym jest shader + Pokaz gotowych shaderow

Program cieniujący - shader, to program komputerowy używany w grafice komputerowej, wykonywany na karcie graficznej. Programy cieniujące są wykorzystywane do manipulowania wyglądem i zachowaniem grafiki na poziomie pikseli lub wierzchołków, w zależności od rodzaju shadera.

Istnieją dwa podstawowe rodzaje programów cieniujących: vertex i fragment shader.

Vertex shadery są odpowiedzialne za przekształcenie wierzchołków, ich położenia, skalowanie, rotacji, a także obliczanie oświetlenia na podstawie źródeł światła. W efekcie, vertex shadery wpływają na położenie i wygląd obiektów na scenie.

Fragment shadery operują na poziomie pojedynczych pikseli na ekranie. Są odpowiedzialne za określanie koloru piksela na podstawie różnych czynników, takich jak tekstury, oświetlenie, przezroczystość i wiele innych efektów wizualnych.

Programy cieniujące są szeroko wykorzystywane w przemyśle gier komputerowych, animacji filmowych i grafice komputerowej. Umożliwiają twórcom grafiki osiągnięcie zaawansowanych efektów wizualnych i tworzenie realistycznych i złożonych scen. Dzięki elastycznemu charakterowi programów cieniujących, graficy mogą tworzyć własne efekty i dostosowywać wygląd grafiki do swoich potrzeb.

Gotowe shadery:

<https://www.shadertoy.com/view/XslGRr>

<https://www.shadertoy.com/view/MdX3zr>

<https://www.shadertoy.com/view/MdXSWn> <- odkomentować //#define żeby włączyć animacje

1. Trochę o fragment shaderze

Fragment shader (inaczej pixel shader) jest jednym z rodzajów programów wykorzystywanych w grafice komputerowej. Jest to program uruchamiany na jednostce cieniującej GPU (Graphics Processing Unit) w celu obliczenia wyglądu poszczególnych pikseli na ekranie.

Fragment shader działa na pikselach, które są generowane przez proces renderowania obiektów na scenie. Jego głównym zadaniem jest określanie ostatecznego koloru (oraz innych właściwości) każdego piksela na podstawie różnych czynników, takich jak oświetlenie, tekstury, cienie, przezroczystość i wiele innych.

Programista ma kontrolę nad tym, jak piksele są renderowane, definiując fragment shader. Ten program jest uruchamiany niezależnie dla każdego piksela na ekranie, co daje duże możliwości dostosowania wyglądu końcowego obrazu.

Fragment shader otrzymuje dane wejściowe takie jak pozycję piksela, kolory tekstur, dane oświetleniowe, a także dodatkowe wartości przekazywane z vertex shadera. Na podstawie tych danych programista może implementować różne operacje, takie jak miksowanie kolorów, generowanie efektów specjalnych, symulowanie oświetlenia i wiele innych.

Działanie fragment shadera jest częścią procesu programowalnego potoku renderowania (programmable rendering pipeline), który umożliwia deweloperom większą kontrolę nad wyglądem renderowanych scen w grach komputerowych, aplikacjach 3D i innych systemach graficznych.

1. Opis środowiska Shadertoy i wykorzystywanych funkcji + podstawy glsl

Shadertoy to platforma online pozwalająca na tworzenie własnych fragment shaderów w czasie rzeczywistym. W naszych przykładach korzystamy z wcześniej przygotowanych funkcji pozwalających rysować prymitywne kształty (koło, prostokąt, trójkąt). Większość zadań polega na zmianie, dodaniu kodu w funkcji mainImage – miejsca, które tego wymagają są opisane komentarzami.

Funkcja mainImage musi przypisać zmiennej fragColor kolor danego piksela.

Wyświetlany obraz i jego współrzędne - siatka reprezentuje krok co 0.1



Podstawy glsl (na potrzeby tego laboratorium)

1. Konstrukcja wektorów:

vec2 a = vec2(0.0) równoważne z vec2 a = vec2(0.0 , 0.0)

vec3 b = vec3(a,0.0) równoważne z vec3 b = vec3(a.x,a.y,0.0)

b.xy równoważne z vec2(b.x,b.y)

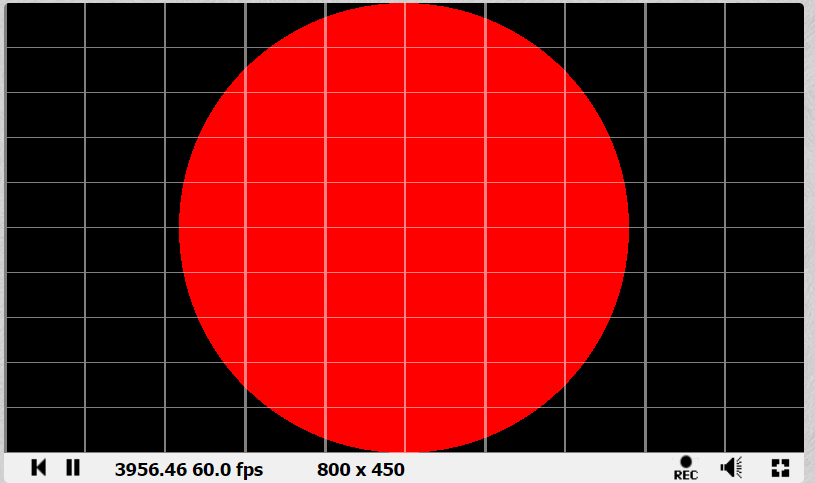
Można też np. B.zx równoważne z vec2(b.z,b.x)

vec2 c = b.xy równoważne z vec2 c = vec2(b.x,b.y)

Funkcja float length(wektor) - zwraca długość wektora

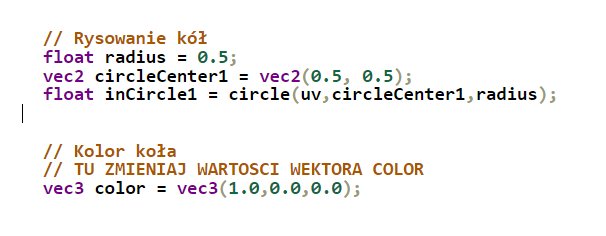
Zadania dla Podstawówki:

1.1 Zmiana koloru:



Zadanie polega na sprawdzeniu jak działa zmiania koloru w glsl. Przed wykonaniem zadania trzeba zapoznać uczniów ze sposobem definiowania kolorów w grafice komputerowej (RGBA, dane przechowywane w wektorze, zakres wartości <0.0,1.0> ).

Aby wykonać zadanie, należy zmieniać wartość wektora color w funkcji mainImage.

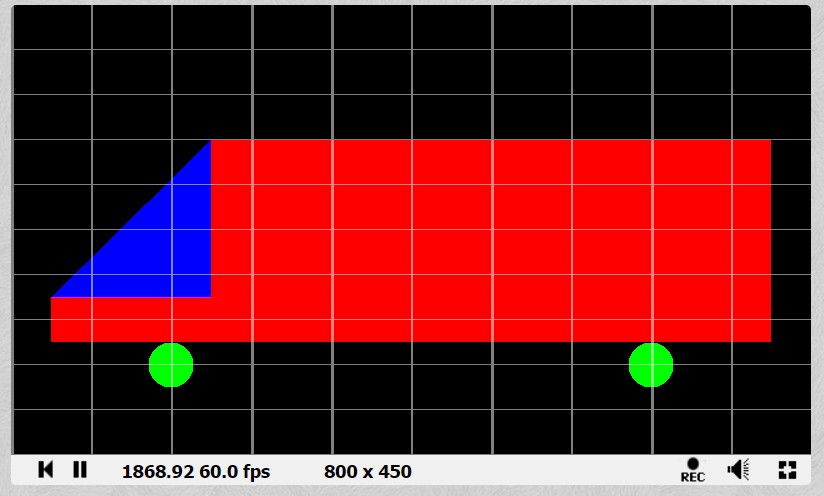


Następnie można zmienić wartość wektora color na pozycje myszki:

vec3 color = vec3(iMouse.xy/iResolution.xy,0.0);

Wtedy kolor będzie się zmieniał PO KILKNIĘCIU w dowolne miejsce na wyświetlaczu.

1.2 Rysowanie samochodu

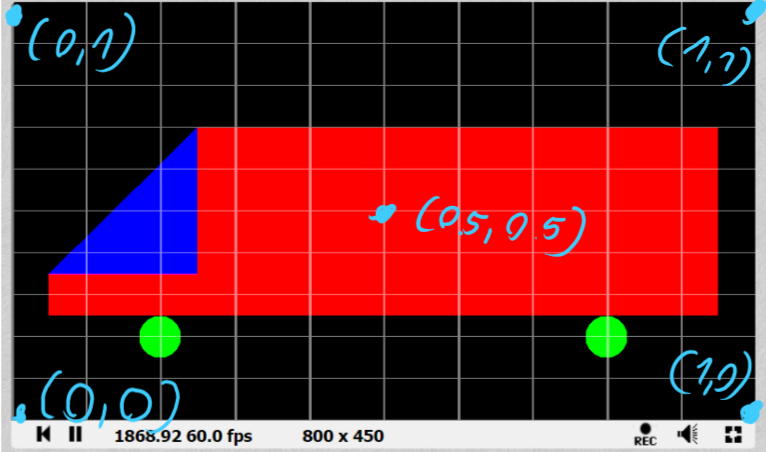


WARIANT 1:

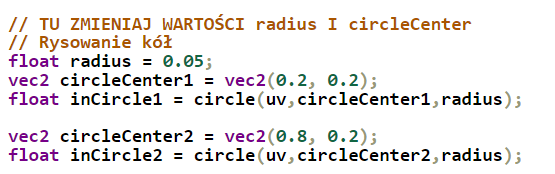
Zadanie polega na wykorzystaniu gotowych funkcji rysujących prymitywne kształty (koło, prostokąt, trójkąt),

w celu narysowania na ekranie obiektu przypominającego samochód.

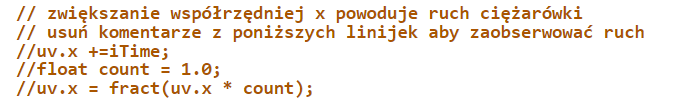
Przed wykonaniem zadania trzeba wytłumaczyć, że pomocnicza siatka przedstawia układ współrzędnych wyświetlacza, gdzie lewy dolny róg to punkt (0.0,0,0), a prawy górny - (1.0,1.0)



Aby tego dokonać należy ustawić w funkcji mainImage odpowiednie promienie i środki kół np.:

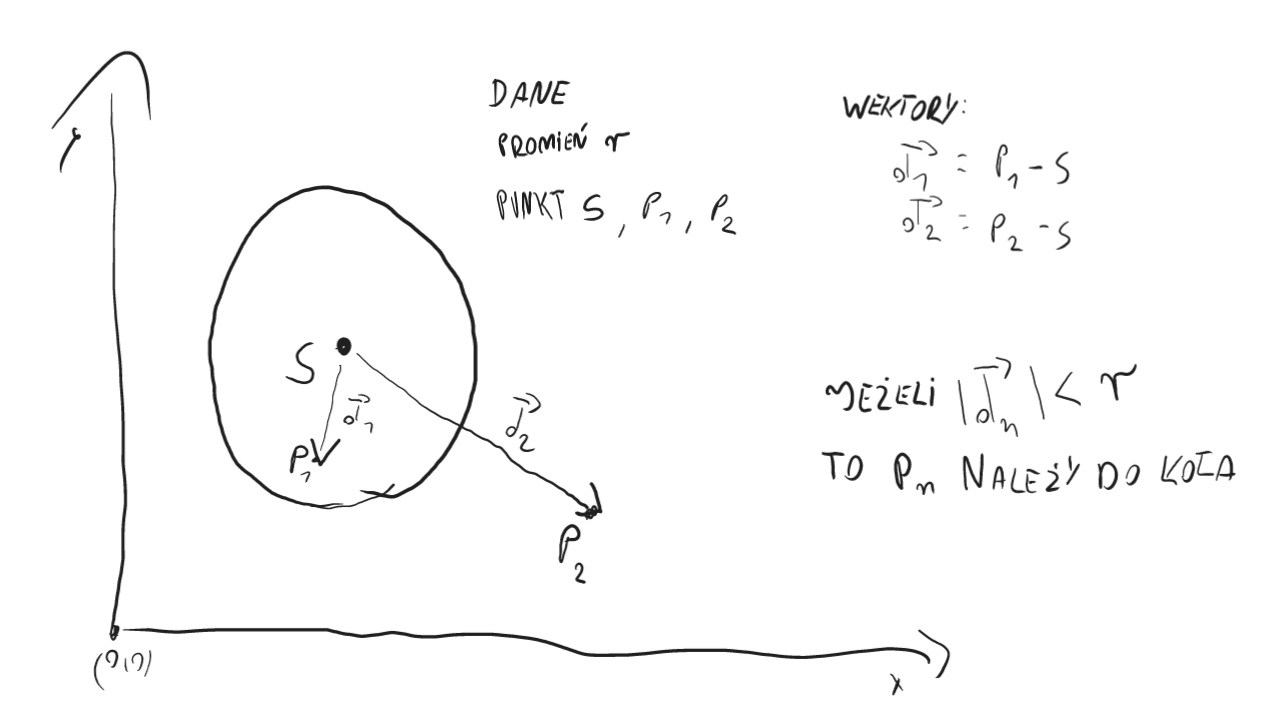


Po ustawieniu odpowiednich współrzędnych, można odkomentować te linie aby zaobserwować ruch samochodu (zwiększenie zmiennej count zwiększy liczbę samochodów)



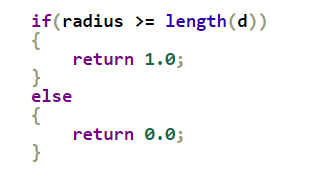
WARIANT 2:

Zadanie polega na uzupełnieniu funkcji rysującej koło. Należy uzupełnić warunek, który sprawdza czy testowany punkt należy do koła. Trzeba wytłumaczyć, jak działa to sprawdzenie:

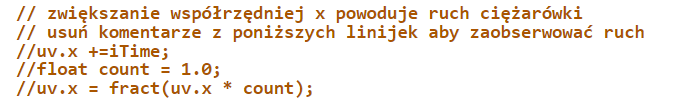


Funkcja zwraca 1.0 jeżeli punkt należy do koła, 0.0 wpp

Przykład rozwiązania:



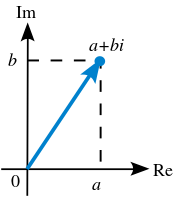
Później można odkomentować te linie aby zaobserwować ruch samochodu (zwiększenie zmiennej count zwiększy liczbę samochodów)



Zadanie liceum:

1. **Liczby zespolone**

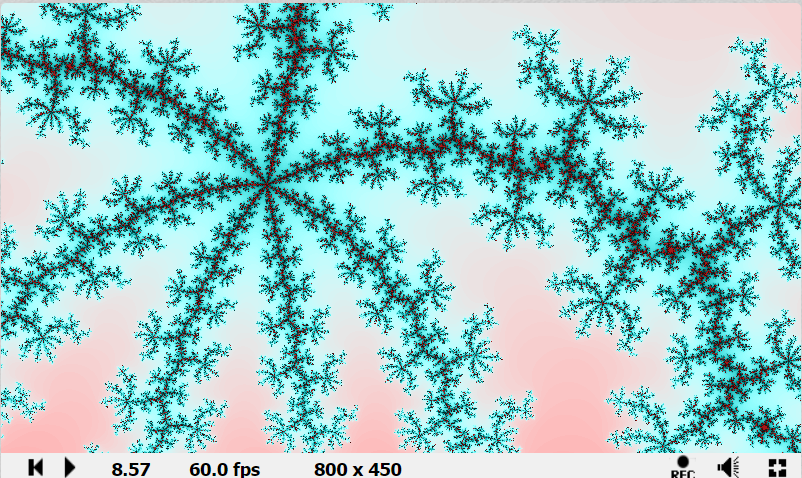
Liczby zespolone są rozszerzeniem zbioru liczb rzeczywistych o jednostkę urojoną i, gdzie i^2= -1. Można je zapisać w postaci a + bi. Liczb zespolonych nie da się zapisać na osi liczbowej, więc zamiast tego używa się dwuwymiarowej płaszczyzny zespolonej.



Dodawanie, odejmowanie i mnożenie liczb zespolonych przebiega tak samo jak dla liczb rzeczywistych, przy czym należy pamiętać o wartości i2dla mnożenia:

(a + bi)^2 = a^2+2abi+b2i2 = a^2+2abi-b^2

1. **Mandelbrot set**



Jest to zbiór punktów na płaszczyźnie zespolonej, dla których ciąg:

Rekurencyjnie:

z0=0

zn+1=zn2+p

Iteracyjnie:

z0=0

z1=z02+p

z2=z12+p

...

nie dąży do nieskończoności.

Dla przykładu dla p=1 mamy sekwencję 0,1,2,5,26... która dąży do nieskończoności więc ten punkt nie należy do zbioru.

Natomiast dla p= -1 mamy 0,-1,0,-1,0… więc ten punkt należy do zbioru.

Źródło: https://arukiap.github.io/fractals/2019/06/02/rendering-the-mandelbrot-set-with-shaders.html

Zadanie polega na uzupełnieniu funkcji Mandelbrot: w każdej iteracji należy zmienić wartość vec2 zespolona (według wzoru na zbiór Mandelbrota)

