Zbiór Mandelbrota na GPU

Autorstwa Michała Wendt

# Wstęp historyczny

Benoît B. Mandelbrot był francusko-amerykańskim [matematyk](https://pl.wikipedia.org/wiki/Matematyka)iem [polsko-żydowskiego](https://pl.wikipedia.org/wiki/%C5%BBydzi_w_Polsce) pochodzenia. Urodził się [20 listopada](https://pl.wikipedia.org/wiki/20_listopada) [1924](https://pl.wikipedia.org/wiki/1924) roku w [Warszawie](https://pl.wikipedia.org/wiki/Warszawa). Zmarł [14 października](https://pl.wikipedia.org/wiki/14_pa%C5%BAdziernika) [2010](https://pl.wikipedia.org/wiki/2010) roku w [Cambridge (Massachusetts)](https://pl.wikipedia.org/wiki/Cambridge_(Massachusetts)). Praktykował wiele działów matematyki. Zainspirował się pracami francuskich matematyków: Gastona Julii i Pierre’a Fatou. Poprzez badanie iteracji funkcji zespolonych za pomocą nowoczesnych maszyn dostępnych dzięki pracy w IBM stworzył wykresy nazwane fraktalami. Dzisiaj jest przede wszystkim rozpoznawany jako „ojciec geometrii fraktalnej”. Jest to spowodowane odkryciem i opisaniem „[zbioru Mandelbrota](https://pl.wikipedia.org/wiki/Zbi%C3%B3r_Mandelbrota)” oraz wymyślenia samego słowa „[fraktal](https://pl.wikipedia.org/wiki/Fraktal)”.

# Czym jest Fraktal?

Samo pojęcie pojawiło się w świecie matematyki w latach 70 XX wieku.

Fraktal to pewien obiekt, który w sensie geometrycznym jest samopodobny, czyli jakaś jego część jest podobna do całości. Spotykamy się także z określeniem jako nieskończenie złożony. Czyli taki, który możemy teoretycznie nieskończenie powiększać.

Matematycy uogólniając opisują fraktale jako spełniające większość lub wszystkie z poniższych warunków:

* ma nietrywialną strukturę w każdej skali,
* struktura ta nie daje się łatwo opisać w języku tradycyjnej geometrii euklidesowej,
* jest samopodobny, jeśli nie w sensie dokładnym, to przybliżonym lub stochastycznym
* jego wymiar Hausdorffa jest większy niż jego wymiar topologiczny
* ma względnie prostą definicję rekurencyjną,
* ma *naturalny* („poszarpany”, „kłębiasty” itp.) wygląd.

Ważnym fraktalem jakim będziemy zajmowali się w tej pracy jest brzeg Zbioru Mandelbrota . Jest to specyficzny fraktal który istnieje tylko w świecie matematycznym. Sam brzeg zbioru Mandelbrota, nie spełnia wszystkich tych warunków tzn. ma wymiar Hausdorffa równy 2, czyli taki sam jak jego wymiar topologiczny. Sam Zbiór Mandelbrota jest podzbiorem [płaszczyzny zespolonej](https://pl.wikipedia.org/wiki/P%C5%82aszczyzna_zespolona) i nie jest fraktalem, ponieważ nie jest samopodobny, co zostało dowiedzione przez chińską matematyczkę [Tan Lei](https://pl.wikipedia.org/w/index.php?title=Tan_Lei&action=edit&redlink=1).

# Wzór na Zbiór Mandelbrota

Nasz [zbiór](https://pl.wikipedia.org/wiki/Zbi%C3%B3r) możemy opisać jako te punkty p(zespolone) p ∈ C , {\displaystyle p\in \mathbb {C} ,} dla których [ciąg](https://pl.wikipedia.org/wiki/Ci%C4%85g_(matematyka))(Zi)infi=0 ( z i ) i = 0 ∞ {\displaystyle (z\_{i})\_{i=0}^{\infty }} zdefiniowany [równaniem rekurencyjnym](https://pl.wikipedia.org/wiki/R%C3%B3wnanie_rekurencyjne)



nie dąży do [nieskończoności](https://pl.wikipedia.org/wiki/Niesko%C5%84czono%C5%9B%C4%87)



Można wykazać, że jest to równoważne z

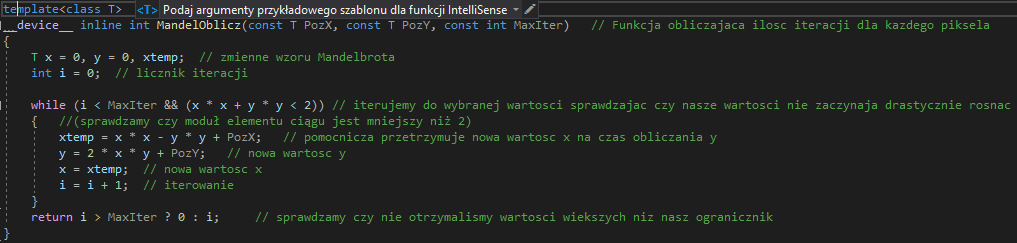


Podsumowując jednym zdaniem:



# Omówienie kodu programu

Używany przez mnie wzór będzie wydawał się o wiele łatwiejszy do zrozumienia niż ta definicja. Od tego wzoru właśnie chciałbym zacząć tę pracę.



Ogólnie mówiąc, aby określić czy punkt należy do zbioru (tutaj czy chcemy go pokolorować) musimy sprawdzić czy wszystkie z wybranych wyrazów początkowych ciągu zn spełniają warunek |Zn| < 2 (W samym programie jestem zmuszony używać 4 przez niedokładność wartości zmiennoprzecinkowych). Inaczej mówiąc wartość podciągu nie zaczyna drastycznie rosnąć. Takim wartością ustawiamy konkretny kolor, a punktom pobliskim kolor podobny i wymieszany z innymi pobliskimi. Jeżeli nasz iterator przekroczył ustaloną maksymalną wartość iteracji punkt będzie zamalowany na czarno.

Opiszmy co dzieje się na zdjęciu.

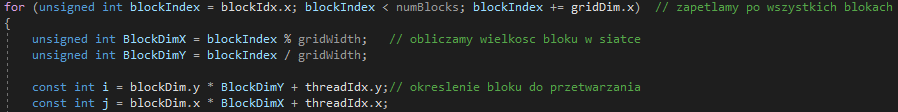
Na początku musimy zadeklarować nasze zmienne x oraz y równe 0. Tutaj tworze także zmienną pomocniczą.

Użyjemy także iteratora ‘i’ obliczającego ilość iteracji potrzebnych do wcześniej wspomnianego „drastycznego wzrostu”. Wnętrze pętli while jest dosłownym zaimplementowaniem wzoru wspomnianego wyżej. Jeżeli którykolwiek z warunków (koniec iteracji lub drastyczny wzrost ciągu) nastąpi możemy zakończyć pętle i zwrócić naszą ilość iteracji.

Kolejnymi ważnymi do omówienia funkcjami są „MandelbrotOn\*\*\*”

Są to dwie funkcje. Jedna w pliku .cu druga w .cpp. Funkcje są bardzo podobne ale z drobnymi różnicami, więc postanowiłem je omówić razem.

Ważna częścią funkcji działającej na GPU jest poruszanie się po siatce bloków. Przypisuje ona kolejnym watkom część obrazu, którą muszą się zająć

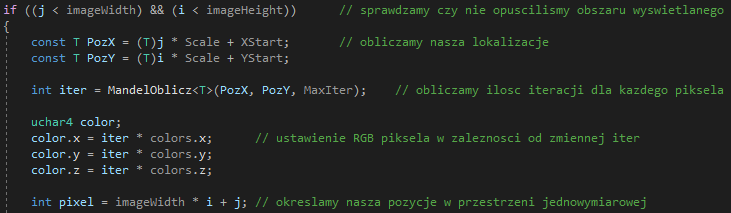


W dalszej części musimy oczywiście przejść po całym obrazie. W przypadku CPU są to zwykłe dwie pętle przechodzące pierw wszerz, a następnie po kolei w pionie do kolejnych poziomów.

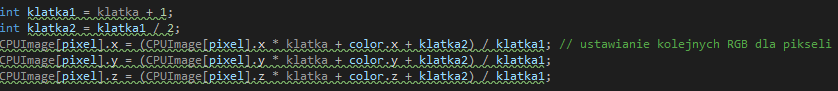


W funkcji GPU jest to tylko pojedyncza kontrola sprawdzająca czy nie wyszliśmy poza obszar malowany, ponieważ bloki wątków odpowiadają kolejnym pikselom.

Później w obu wersjach w identyczny sposób wywołujemy funkcje liczącą ilość iteracji (pokazaną wcześniej) oraz przemnażamy barwy RGB przez wynikową liczbę.

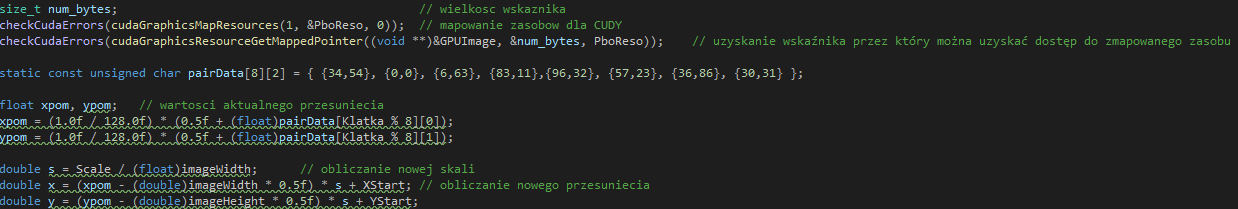


Teraz możemy przypisać kolor do konkretnego punktu naszego obrazu.

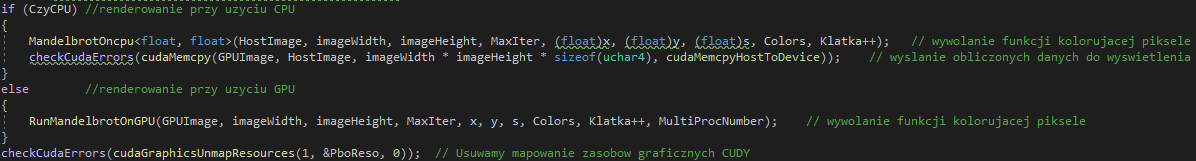


Ostatnią z funkcji związanych z Mandelbrotem będzie funkcja renderująca „render”. Jej głównym przeznaczeniem jest obliczanie nowego obszaru wyświetlanego i wywołanie odpowiedniej funkcji MandelbrotOn\*\*\*.

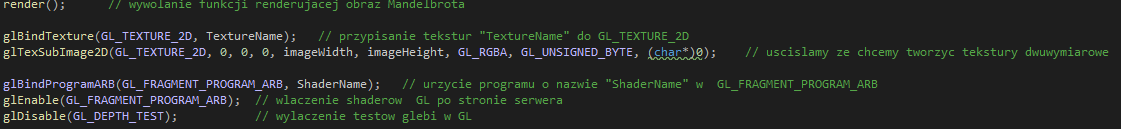
Musimy oczywiście zacząć od deklaracji zmiennych, określenia miejsca zasobów dla CUDY oraz obliczenie nowej wyświetlanej lokalizacji.



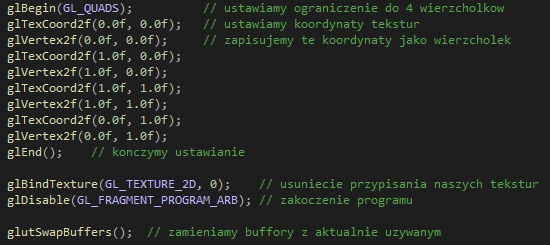
Gdy dane są już gotowe możemy wywołać funkcje zależnie od ustawienia urządzenia na CPU lub GPU. Trzeba pamiętać, że w przypadku CPU otrzymane dane trzeba wysłać do GPU w celu ich wyświetlenia na ekranie. Na koniec sprzątamy po sobie już niepotrzebne zasoby CUDY.



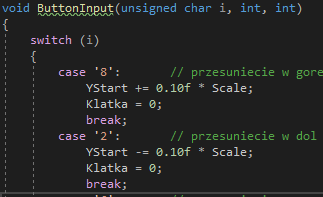
Powyższe polecenia są wywoływane przez display, która odpowiada za wyświetlenie nam obrazu w naszej ramce, tworzenie ramki pojawi się za chwile. Do wyświetlania obrazu używam OpenGl (dokładny opis technologii pozostawiam na koniec). Teraz tylko wspomnę, że jest to biblioteka umożliwiająca tworzenie grafiki. OpenGl musi oczywiście wiedzieć trochę o tym co chcemy wyświetlić, więc informujemy go w której zmiennej powinien szukać potrzebnych tekstur, o wymiarowości obrazu oraz typie shaderów.



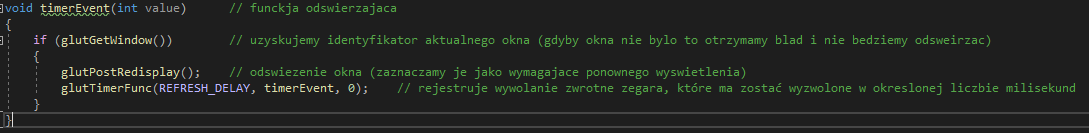
Tutaj musimy utworzyć prostokąt o 4 wierzchołkach w którym będziemy mogli wyświetlić obraz. Gdy wszystkie informacje są gotowe, zamykamy program GL-a i podmieniamy otrzymany wynik z poprzednio wyświetlanym.



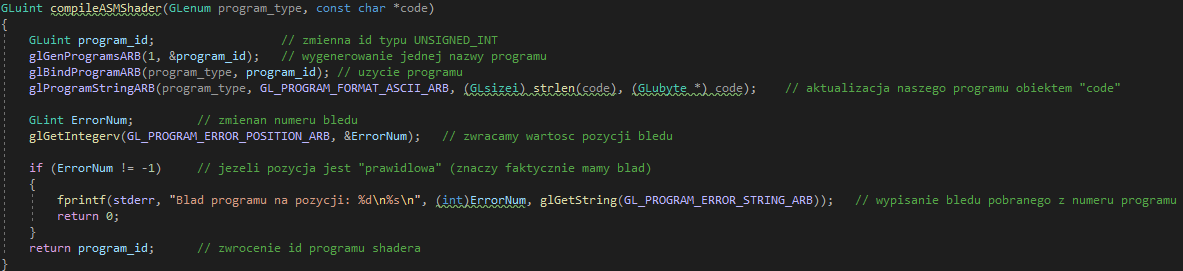
Kolejną funkcję omówię krótko. Jest to prosty switch do wykonywania konkretnych poleceń przy wejściu z klawiatury. Nasłuch na wejście pojawi się później. Możemy tu zmienić skale, przesuwać się, edytować zmienne lub zakończyć działanie programu.



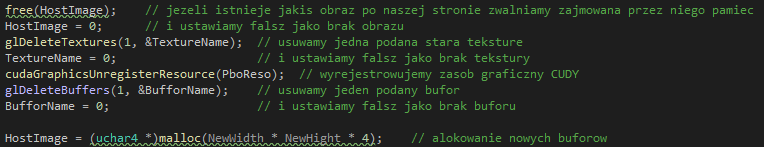
TimeEvent to krótka funkcja odświeżająca nasze okno w określonych w REFRESH\_DELAY odstępach czasu.



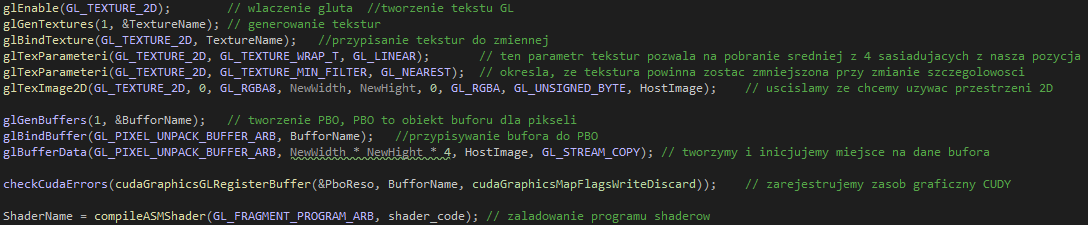
Tutaj już w dużym stopniu gotowa funkcja shadera OpenGL generująca program z danego codu i sprawdzająca ewentualne błędy zaistniałe podczas tego procesu.



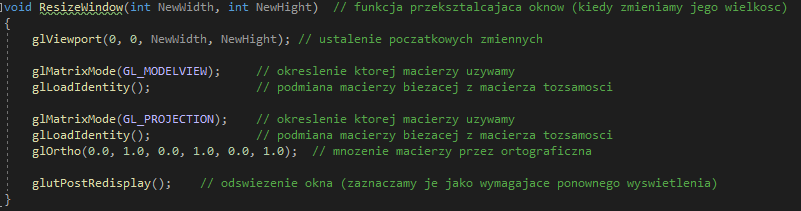
W funkcji initOpenGLBuffers jak sama nazwa mówi stworzymy miejsce na dane OpenGL-a. W tym celu musimy usunąć ewentualne już istniejące zapisy. I zaalokować miejsce na nowy obraz.



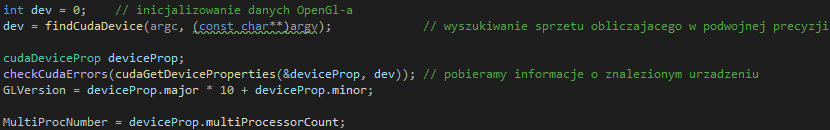
Teraz za pomocą funkcji GL-a możemy wygenerować wszystkie potrzebne nam dane w celu generowania tekstur.



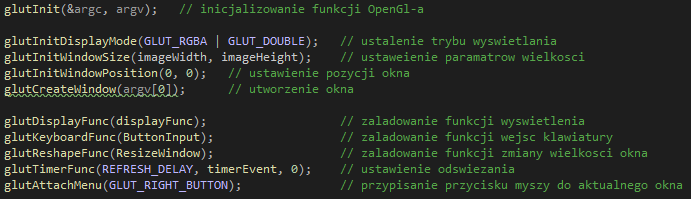
Edycja okna wyświetlanego może odbywać się na dwa sposoby, użytkownik może zmienić domyślną rozdzielczość obrazu na początku pliku .cpp lub podczas jego trwania przytrzymać przycisk myszy na ramce obrazu i rozciągnąć lub skurczyć w dowolnie wybrany sposób. Obraz wyświetlany będzie dostosowywał się natychmiastowo.



Zostało nam jeszcze kilka ważnych linii kodu w funkcji main. Pierwszy fragment odpowiada za inicjalizacje danych potrzebnych dla poprawnego działania CUDY. Czyli informacje o urządzeniach zdolnych do wykonywania wymaganych obliczeń.

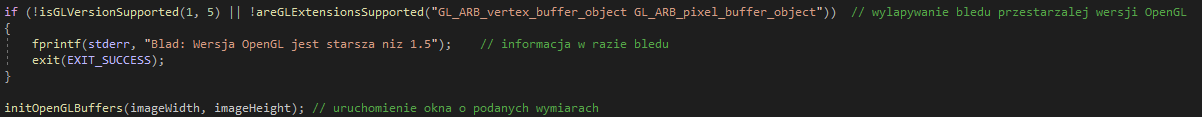


Drugi zestaw to początkowo zbierane dane dla GL-a (wraz z aktywacją funkcji), takie jak: w jakim formacie i jakiej rozdzielczości ma działać, gdzie ma zacząć, że ma nasłuchiwać klawiaturę czy jak często ma odświeżać swój proces.



Jeszcze sprawdzamy czy użytkownik ma dostatecznie nową wersję GL ( dodałem już bibliotekę więc nie powinno być z tym problemu) i informuje o ewentualnym błędzie.

I przechodzimy do funkcji teksturującej już wcześniej wytłumaczonej.



Na koniec informujemy OpenGL, żeby zapętlił swój program w celu nasłuchu na dalsze polecenia.

# Opis użytych technologii:

Program został napisany przy użyciu technologii Nvidia CUDA w języku C++. Korzysta z gotowych bibliotek CUDY oraz OpenGL-a. Powstał w środowisku VisualStudio.

Podstawowym aspektem przemawiającym za przewagą GPU nad CPU w tym typie problemu jest fakt wyspecjalizowania do intensywnych obliczeniowo, wysoce równoległych obliczeń bycia zaprojektowanym w taki sposób, by więcej tranzystorów było przeznaczonych do przetwarzania danych, a nie do buforowania danych i kontroli przepływu.

**Visual Studio** – środowisko programistyczne od firmy Microsoft

Wystarczy pobrać ze strony producenta:

[**https://visualstudio.microsoft.com/pl/free-developer-offers/**](https://visualstudio.microsoft.com/pl/free-developer-offers/)

I przejść przez prosty proces instalacji pamiętając o wybraniu opcji „Desktop development with C++” w szczegółach instalacji.

**CUDA** – technologia stworzona i rozwijana przez firmę [Nvidia](https://pl.wikipedia.org/wiki/Nvidia). Jej głównym założeniem jest użycie procesorów wielordzeniowych (najczęściej [kart graficznych](https://pl.wikipedia.org/wiki/Karta_graficzna)) w celu użycia mocy obliczeniowej w sposób wydajniejszy niż tradycyjny tj. sekwencyjny. Wykonywanie wielu obliczeń w tym samym czasie znacznie przyspieszyło obliczenia problemów numerycznych. Tym samym umożliwiając wykonywanie bardzo skomplikowanych działań ogromną ilość razy w bardzo krótkim czasie.

Cuda zawiera w swoim środowisku programistycznym własny kompilator (nvcc), debugger (cuda-gdb), [profiler](https://pl.wikipedia.org/wiki/Profilowanie_(in%C5%BCynieria_oprogramowania)), [interfejs programowania aplikacji](https://pl.wikipedia.org/wiki/Interfejs_programowania_aplikacji) oraz wiele przydatnych bibliotek, które można wykorzystać w językach Python, Fortran, Java, C# oraz Matlab.

Z Cudą możemy spotkać się na wielu systemach operacyjnych ponieważ aktualnie wspiera zarówno [Windows](https://pl.wikipedia.org/wiki/Microsoft_Windows)a, [Linux](https://pl.wikipedia.org/wiki/Linux)a jak i [Mac OS X](https://pl.wikipedia.org/wiki/MacOS).

**Kilka zalet pokazujących, że warto używać cudy:**

* Model pamięci procesora ściśle odpowiadający architekturze sprzętowej, co umożliwia świadome, efektywne wykorzystywanie dostępnych zasobów GPU, w tym [pamięci współdzielonej](https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_dzielona). Pamięć ta jest współdzielona przez wszystkie wątki w tzw. bloku (zwykle 128-512 wątków). Można jej używać jako programowalnej pamięci typu [cache](https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_podr%C4%99czna).
* Pełna kompatybilność wsteczna programów.
* Dostępność na wszystkich kartach firmy NVIDIA począwszy od serii [GeForce 8](https://pl.wikipedia.org/wiki/GeForce_8)

**Aby zacząć używać cudy musimy upewnić się, że nasza karta graficzna jest do tego zdolna. W tym celu sprawdzamy czy znajduje się na liście GPU na stronie Nvidii:** [**https://developer.nvidia.com/cuda-gpus**](https://developer.nvidia.com/cuda-gpus)

**Jeżeli nasza karta graficzna jest na liście możemy pobrać zestaw narzędzi CUDY:**

[**https://developer.nvidia.com/cuda-downloads**](https://developer.nvidia.com/cuda-downloads)

Proces instalacji jest dość prosty, wystarczy wybrać ścieżkę instalacji i typ instalacji.

**OpenGL**(Open Graphics Library) –uniwersalne i open-sourcowe API służące do renderowania grafiki wektorowej 2D i 3D. Jego głównym przeznaczeniem jest interakcja z GPU w celu uzyskania renderowania przyspieszanego sprzętowo.

API jest zdefiniowane jako zestaw funkcji, które mogą być wywoływane przez program klienta, wraz z zestawem nazwanych stałych całkowitych.

Instalacja opiera się na pobraniu odpowiednich paczek do projektu. W moim projekcie te paczki są już zaimplementowane.

**W razie problemów z oprogramowaniem Cudy lub VS polecam film, którego sam używałem:**

**https://www.youtube.com/watch?v=cuCWbztXk4Y**