PROJEKT NR 33
WYKRESY 3D FUNKCJI TRZECH ZMIENNYCH WE WSPÓŁRZĘDNYCH SFERYCZNYCH

Autorzy: Maciej Pieczonka, Łukasz Idzik, Łukasz Radzik

Opis projektu

Celem projektu było napisanie programu rysującego wykres różnych funkcji f(r,θ,φ). Aplikacja pozwala rysować funkcję w trzech różnych postaciach.

W pierwszym sposobie rysowana jest sfera składającą się z punktów. Kolor punktów zależy od wartości, jaką funkcja przyjmuje w danym punkcie (minimum - niebieski, maksimum - czerwony). Użytkownik sam wybiera promień sfery - r oraz ilość punktów podziału zakresu zmienności parametrów θ i φ.

W drugim trybie program rysuje punkty, lecz tym razem są one oddalone od środka układu współrzędnych tym dalej, im większa w danym punkcie jest wartość funkcji, dla ustalonego wcześniej r. W tym przypadku minimum funkcji pokrywa się ze środkiem układu współrzędnych (dla lepszego pokazania, że faktycznie im większa wartość w danym punkcie, tym jest on dalej, również i w tym przypadku kolor punktów zależy od wartości funkcji).

W trzecim trybie pracy programu użytkownik, oprócz liczby punktów podziału zakresu zmienności θ i φ podaje również wartości parametrów const, ε, Δr oraz r_{max}. Program, dla danego θ i φ, dla kolejnych r z przedziału [0, r_{max}] (z krokiem Δr) sprawdza, czy spełnione są nierówności: const-ε < f(r,θ,φ) < const+ε. Jeśli tak, to dany punkt jest rysowany. Wszystkie punkty są rysowane w kolorze niebieskim.

Program umożliwia także obrót układu współrzędnych o dowolny kąt. Jest także możliwość zapisu wykresu do pliku.

Założenia wstępne przyjęte w realizacji projektu

Na początku wartości wszystkich parametrów są ustawione na 0. Aby program poprawnie narysował dany wykres, wymagane jest podanie odpowiednich wartości parametrów: ilość punktów podziału zakresu zmienności θ i φ powinna być liczbą całkowitą dodatnią, r, ε, Δr oraz r_{max} to liczby rzeczywiste większe od zera, natomiast const jest dowolną liczbą rzeczywistą. Jeśli któryś z parametrów wymaganych do rysowania danym sposobem nie spełnia powyższych warunków, to wykres nie zostanie narysowany.

Początkowo kamera ustawiona jest w taki sposób, że znajduje się na dodatniej osi Z i jest zwrócona w kierunku środka układu współrzędnych. Wszystkie naniesione zmiany (obroty, zmiana któregoś z parametrów, zmiana funkcji lub sposobu rysowania) są generowane na wykresie w czasie rzeczywistym.

W zależności od sposobu rysowania, podziałki na osiach układu współrzędnych mają inne znaczenie. W sposobie pierwszym (rysowanie sfery), każda kolejna podziałka oznacza 5 jednostek więcej (czyli na każdej z osi mieszczą się wartości od -20 do 20 jednostek). W trybach drugim i trzecim rysowania podziałki służą jedynie jako punkty odniesienia i oznaczają kolejno 25%, 50%, 75%, 100% długości każdej z półosi (dodatkowo w drugiej metodzie rysowane punkty zostały przeskalowane tak, aby były oddalone od środka układu o nie więcej niż 80%; ma to na celu poprawienie czytelności wykresu).

Aplikacja została także wyposażona w znajdujące się w lewym dolny rogu pole tekstowe, w którym wyświetlane są informacje o aktualnym sposobie rysowania oraz o wybranej funkcji.

Analiza projektu

Specyfikacja danych wejściowych

Podana przez użytkownika wartość danego parametru w odpowiadającym mu polu (jest to zmienna typu wxString) zostaje przekonwertowana na typ double lub int, w zależności od typu zmiennej, do której przypisujemy daną wartość.

Rodzaj oczekiwanych danych wyjściowych

Od programu oczekuje się poprawnie narysowanego wykresu, zależnego od wartości parametrów, sposobu rysowania oraz wybranej funkcji.

Zdefiniowanie struktur danych

Do przechowywania współrzędnych rysowanych punktów oraz odcinków wykorzystaliśmy napisaną przez nas klasę Vector. Dodatkowo, w celu przypisania koloru do każdego z punktów, skorzystaliśmy z kontenera std::pair, w którym zostały umieszczone współrzędne punktu (typ Vector) oraz jego kolor (typ wxColour).

Do przeprowadzenia poszczególnych transformacji związanych m.in. z rotacją skorzystaliśmy z macierzy i operacji na nich. W tym celu stworzono klasę Matrix, która w łatwy sposób pozwoliła nam wykonać wymagane operacje,

Do zapisu wykresu na dysk wykorzystano typy wxImage oraz wxBitmap.

Specyfikacja interfejsu użytkownika

Użytkownik ma możliwość wybrania trybu, w jakim chce wyświetlić dany wykres oraz funkcji, która ma być rysowana. Wartości poszczególnych parametrów są wprowadzane poprzez wpisanie ich w odpowiednie pola. Za pomocą trzech suwaków znajdujących się w prawym dolnym rogu okna aplikacji, użytkownik ma możliwość dowolnie obracać wykres, a znajdujący się obok przycisk “Reset” pozwala szybko powrócić do widoku początkowego.

Istnieje również opcja zapisania wykresu do pliku, poprzez wciśnięcie przycisku “Zapisz”. Spowoduje to pokazanie się okna dialogowego, które umożliwia wybór lokalizacji oraz nazwę i typ tworzonego pliku. Dostępne formaty to .jpg, .png, .bmp.

Wyodrębnienie i zdefiniowanie zadań

Projekt można podzielić na kilka mniejszych modułów. Są to:

- Utworzenie okna aplikacji za pomocą programu wxFormBuilder.
- Utworzenie klas Vector i Matrix pozwalających na wygodny dostęp i obsługę danych.
- Napisanie funkcji umożliwiających wykonywanie poszczególnych transformacji.
- Obsługa podawanych przez użytkownika parametrów i wyznaczanie pozycji i koloru punktów na ich podstawie.
- Napisanie metody rysującej w poprawny sposób punkty i odcinki.
- Testowanie poprawności działania aplikacji i optymalizacja.

Decyzja o wyborze narzędzi programistycznych

Do naszego projektu wykorzystaliśmy bibliotekę wxWidgets, ponieważ posiada ona wiele funkcji wbudowanych znacznie przyspieszających proces tworzenia aplikacji. Do kompilacji użyliśmy kompilatora g++ w wersji 7.5.0. W projekcie staraliśmy się używać standardu C++11. Jako edytora używaliśmy Visual Studio Code oraz środowiska Visual Studio.

Podział pracy i analiza czasowa

Na początku stworzyliśmy okno aplikacji w wxFormBuilder. Przygotowaliśmy interfejs, tak by posiadał wszelkie niezbędne suwaki, przyciski i miejsca na przyjmowanie danych z zewnątrz. Kolejnym krokiem było zaimplementowanie osi i macierzy transformacji, która pozwala nam obracać układ względem wybranej osi. Następnie napisana została funkcja, która tworzy punkty posiadające odpowiednie współrzędne oraz kolor zależny od wartości funkcji. Mając taką funkcję możliwe zostało utworzenie funkcji rysującej punkty we współrzędnych kartezjańskich. Na koniec dodaliśmy implementację przycisku, który umożliwia zapisywanie do pliku.

Opracowanie niezbędnych algorytmów

W programie wykorzystaliśmy algorytm do obracania obrazu. Chcieliśmy uzyskać efekt obrotu wokół wybranej osi układu współrzędnych, a nie względem "wirtualnych" osi ekranu, dla których płaszczyzna ekranu to osie X i Y, a wgłąb jest oś Z. Algorytm przedstawiony na wykładzie (macierz obroty) nie dawał tego efektu, więc musieliśmy zaimplementować własny. Algorytm ten działa w następujący sposób. Najpierw wykonywana jest transformacja współrzędnych wszystkich punktów/odcinków do układu ekranu. Następnie nakładane są wymagane rotacje, a na koniec wykonywana jest transformacja z powrotem do układu osi współrzędnych. Wadą tego algorytmu jest to, że wykonuje on operacje nie względem położenia początkowego, a względem stanu poprzedniego. Aby nie utracić informacji o operacjach koniecznych do wykonania, aby przejść od razu ze stanu początkowego do aktualnego, stworzono macierz m_rotateHistory. Po przemnożeniu współrzędnych danego punktu (bez nałożonych transformacji) przez tę macierz, jesteśmy w stanie zsynchronizować położenia układu współrzędnych i poszczególnych punktów

Jako że w aplikacji wykorzystano normowanie położeń punktów/odcinków, ich współrzędne musiały się mieścić w przedziale [-1, 1]. Dlatego utworzono funkcję, która umożliwiała unormowanie wartości wpisanych przez użytkownika do właśnie tego przedziału (na przykład, gdy w pierwszym trybie rysowania użytkownik poda wartość r = 10, a na osiach umieszczono wartości na moduł nie większe niż 20, to odległość r zostanie zmieniona na wartość 0.5, tak by promień sfery sięgał do połowy długości półosi.

Kodowanie

Opis klas:

- MyApp - klasa niezbędna do uruchomienia programu
- MyFrame1 - klasa wygenerowana przez wxFormBuilder, są w niej wszystkie elementy interfejsu
- GUIMyFrame1 - klasa, która odpowiada za prawidłowe działanie programu. Są w niej zaimplementowane wszystkie niezbędne metody, które sterują programem.
- Vector - klasa tworząca wektor trójwymiarowy zapewniająca podstawowe operacje matematyczne na wektorach

- Matrix - klasa, która tworzy tworzy macierz oraz zapewnia podstawowe operacje matematyczne na macierzach

Opis zmiennych:

- std::vector<std::pair<Vector, wxColour>> points - zmienna przechowująca współrzędne punktów oraz przypisany kolor do danego punktu
- std::vector<Vector> m_axes - zmienna przechowująca początki i końce osi współrzędnych
- std::vector<Vector> m_arrowheads - zmienna przechowująca współrzędne strzałek na końcach osi współrzędnych
- std::vector<Vector> m_segments - zmienna przechowująca współrzędne miarek osi, wyświetlanych na ekranie.
- Matrix m_roration History - zmienna przechowująca iloczyn wszystkich rotacji, jakie zaszły od położenia początkowego do aktualnego
- wxBitmap picture_bm - zmienna typu bitmapa, wykorzystywana przy zapisie obrazu do pliku.
- wxImage save_img - zmienna przechowująca obraz, który zostanie zapisany do pliku
- double m_rotX, m_rotY, m_rotZ - zmienne przechowujące wartość obrotu układu względem osi
- int m_phi, m_theta - zmienne, które przyjmują wartość odpowiadającą za ilość podziałów zakresów θ oraz φ
- double m_r, m_const, m_epsilon, m_delta_r, m_r_max - zmienne przechowujące wartości poszczególnych parametrów
- int m_chartChoice - zmienna, która przechowuje numer aktualnie wybranej funkcji
- bool m_funChanged - flaga wskazująca czy użytkownik zmienił wykres, który ma zostać wyświetlony
- bool m_chartChanged - flaga wskazująca czy sposób rysowania wykresu się zmienił
- bool m_resetAxes - flaga wskazująca czy osie mają zostać przywrócone do pozycji domyślnej
- bool m_rChanged, m_thetaChanged, m_phiChanged, m_constChanged, m_epsilonChanged, m_delta_rChanged, m_r_maxChanged - flagi, wskazujące czy użytkownik zmienił wartości r, θ lub φ.

Opis funkcji:

- funNr - metoda zwracająca wartość 0, 1 , 2 w zależności, którą funkcję wybrał użytkownik do rysowania
- F1, F2, F3 - metody zwracające wartości poszczególnych funkcji dla danych wartości przekazanych argumentów
- getPoints - metoda tworząca punkty, które następnie obsługuje poprzez przypisanie do danego punktu koloru oraz przekonwertowanie go na współrzędne kartezjańskie.
- Reset - metoda przywracająca wszystkie wartości do stanu początkowego
- DrawAxes - metoda rysująca osie współrzędnych
- Repaint - główna metoda, odpowiedzialna za wywołanie wielu innych funkcji. Odpowiada ona za poprawne wyświetlanie danych na ekranie użytkownika
- GetRorationMatrix - metoda zwracająca macierz rotacji pozwalającą przejść do kolejnego stanu.

Testowanie

Działanie programu testowano manualnie, ustawiając różne wartości rotacji, ilości rysowanych punktów oraz innych wartości wprowadzanych przez użytkownika. Żaden z tych testów nie sprawił naszemu programowi problemu.

Jedyną operacją która potencjalnie może być problematyczna jest zmiana ilości rysowanych punktów – ustawienie zbyt dużej ich ilości może ‘zamrozić’ program. Jest to jednak w dużym stopniu zależne od wydajności sprzętu użytkownika, przez co postanowiliśmy nie implementować górnej granicy tych wielkości.

Wdrożenie, raport i wnioski

W implementacji udało nam się wykonać wszystkie podstawowe i rozszerzone założenia, za wyjątkiem rysowania siatki łączącej punkty wykresu, na które nie potrafiliśmy znaleźć rozwiązania.

Optymalizacji z pewnością mogłoby ulec szukanie minimalnej wartości funkcji, z pewnością jest możliwe znalezienie jej analitycznie zamiast przeszukiwania wszystkich punktów.

Temat projektu był bardzo ciekawy i pozwolił nam utrwalić wiedzę o przekształceniach w 3D, a sam końcowy efekt oraz możliwość inspekcji kształtów tych funkcji są dla nas bardzo zadowalające.