



Podstawy Grafiki Komputerowej
Wave Interference - dokumentacja projektu

Arkadiusz Kasprzak
Jarosław Cierpich
Wydział Fizyki i Informatyki Stosowanej
Informatyka Stosowana

25 sierpnia 2018

1 Wave Interference - wstęp

Do wykonania w ramach projektu z przedmiotu Podstawy Grafiki Komputerowej otrzymaliśmy projekt numer **009 - Interferencja Fal**. Projekt ten polega na napisaniu programu prezentującego interferencję fal na powierzchni w przestrzeni 3D. Podczas realizacji projektu zaimplementowane zostały wszystkie założenia wynikające z wymagań podstawowych i rozszerzonych projektu, jak również zostały przez nas wprowadzone pewne udoskonalenia.

2 Opis projektu

Celem projektu była wizualizacja na płaszczyźnie zjawiska fizycznego jakim jest interferencja fal. Wspomniana płaszczyzna zawieszona jest w przestrzeni 3D, można ją interpretować jako powierzchnię wody lub membranę. Program ma prezentować rozchodzenie się wspomnianych fal oraz ich nakładanie. Parametry fal są podawane przez użytkownika. Animacja prezentowana jest na ekranie komputera - zarówno w postaci płynnej, jak i poklatkowej. Istnieją też pomniejsze funkcjonalności, jak możliwość zapisania klatki animacji do pliku lub schowka lub wydrukowanie jej. Opis projektu nie narzuca narzędzi (za wyjątkiem języka) - zostało to pozostawione osobom wykonującym projekt - wybór padł na framework Qt.

3 Założenia wstępne

Podczas wykonywania projektu przyjęliśmy kilka założeń. Jednym z nich był sposób reprezentacji płaszczyzny w programie. Przyjęliśmy, że będzie ona reprezentowana jako zbiór równomiernie rozłożonych punktów, połączonych liniami. Założyliśmy również nieograniczoną ilość możliwych do wstawienia źródeł fal - wynika to z prostoty rozszerzalności rozwiązania z dwoma źródłami. Kolejnym założeniem była możliwość obrotu płaszczyzny wokół każdej z osi - pozwala to lepiej zaobserwować wizualizowany przez nas efekt. Jednym z ważniejszych kryteriów, jakim kierowaliśmy się, tworząc projekt była jak najlepsza optymalizacja - wykorzystywany przez nas framework Qt dostarcza odpowiednich środków do realizacji tego zadania (m.in. wbudowana obsługa wątków). Rozmiar płaszczyzny (w punktach) został przez nas odgórnie ustalony - wynosi on 80×80 punktów. Przenosząc to na jednostki w układzie płaszczyzny odpowiada to 1000×1000 jednostek - tzn

środek płaszczyzny usytuowany jest w punkcie (500, 500). Rozwiązanie takie pozwala uzyskać w miarę płynną animację (testowane na naszych komputerach). Założyliśmy również, że wszelkie transformacje (takie jak translacja, rotacja czy perspektywa) zostaną zaimplementowane przez nas od zera (bez użycia transformacji dostępnych w wykorzystywanym środowisku) - w celu utrwalenia wiedzy nabytej na przedmiocie. Wszystkie wymagania, zarówno podstawowe, jak i rozszerzone zostały spełnione. Poniżej lista zaimplementowanych funkcjonalności:

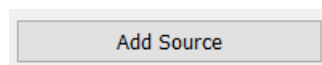
- **(podstawowe)** wyświetlenie płaszczyzny w przestrzeni 3D na ekranie monitora za pomocą odpowiednich transformacji
- **(podstawowe)** wybór przez użytkownika parametrów fali
- **(podstawowe)** wyświetlanie animacji w sposób poklatkowy - klatka po klatce - użytkownik może *przewijać* animację zarówno do przodu, jak i do tyłu klatka po klatce.
- **(podstawowe)** rzucenie pojedynczej klatki do pliku - jest to ta klatka, która aktualnie wyświetlana jest na ekranie, bez zastosowanego skalowania
- **(podstawowe)** rzucenie pojedynczej klatki do schowka systemu Windows - działa podobnie jak powyżej, klatka może następnie zostać wklejona np. do programu Microsoft Paint
- **(rozszerzone)** wyświetlanie animacji w sposób płynny - po wciśnięciu odpowiedniego przycisku animacja zostaje w sposób zapętłony odtwarzana na ekranie
- **(rozszerzone)** obracanie wyświetlanej powierzchni względem osi x, y oraz z (gdy nie trwa animacja)
- **(rozszerzone)** drukowanie pojedynczej klatki
- **(ponadprogramowe)** możliwość dodawania dowolnej ilości źródeł
- **(ponadprogramowe)** realizacja obliczeń za pomocą wielowątkowości.

4 Analiza projektu

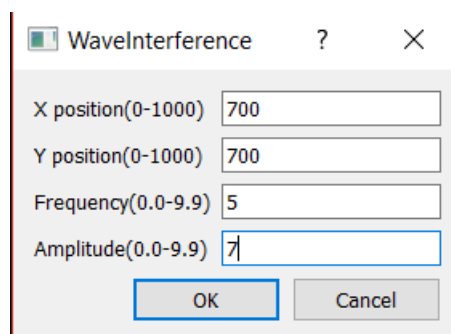
Poniżej przedstawiamy szczegółową analizę naszego projektu

4.1 Dane wejściowe

Użytkownik może w sposób bezpośredni manipulować dwoma rodzajami danych wejściowych. Pierwszy rodzaj to parametry dodawanego źródła fal. Ustawia się je po wciśnięciu przycisku **Add Source**:

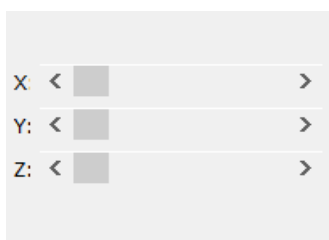


Po wciśnięciu tego przycisku pojawia się następujące menu:

A dialog box titled "WaveInterference" with a question mark icon and a close button (X). It contains four input fields: "X position(0-1000)" with value 700, "Y position(0-1000)" with value 700, "Frequency(0.0-9.9)" with value 5, and "Amplitude(0.0-9.9)" with value 7. At the bottom are "OK" and "Cancel" buttons.

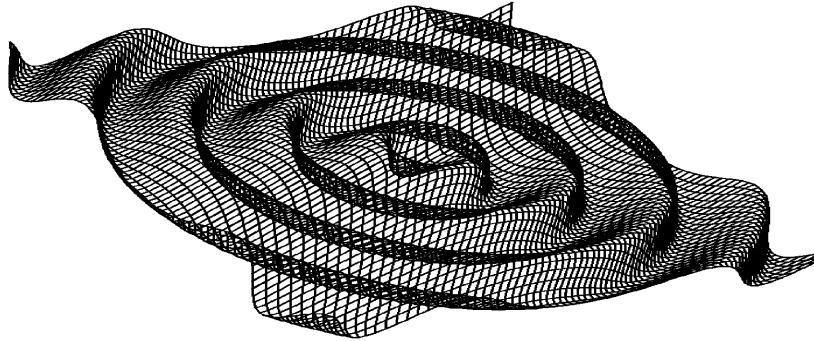
Parameter	Value
X position(0-1000)	700
Y position(0-1000)	700
Frequency(0.0-9.9)	5
Amplitude(0.0-9.9)	7

W menu tym użytkownik wprowadza położenie dodawanego źródła - współrzędne X i Y (w zakresie 0-1000, przy czym program został zabezpieczony przez niepoprawnymi danymi), częstotliwość wyświetlanej fali oraz jej amplitudę (obie w zakresie od 0.0 – 9.9). Drugim rodzajem danych wejściowych jest stopień obrotu rysowanej płaszczyzny wokół jednej z trzech osi - X, Y oraz Z (w zakresie 0 – 360°). Dane te podaje się za pomocą trzech suwaków, przedstawionych poniżej:



4.2 Dane wyjściowe

Dane wyjściowe w naszej aplikacji stanowi animacja przedstawiająca interferencję fal. Przykładowa klatka animacji wygląda następująco:



Istnieje możliwość zapisania pojedynczej klatki do pliku (w dowolnym formacie), do schowka (w systemie Windows) oraz jej wydrukowania - co również stanowi dane wyjściowe programu. Poszczególne opcje dostępne są pod odpowiednimi przyciskami naszej aplikacji.

4.3 Struktury danych

Podczas pisania programu zastosowaliśmy następujące struktury danych:

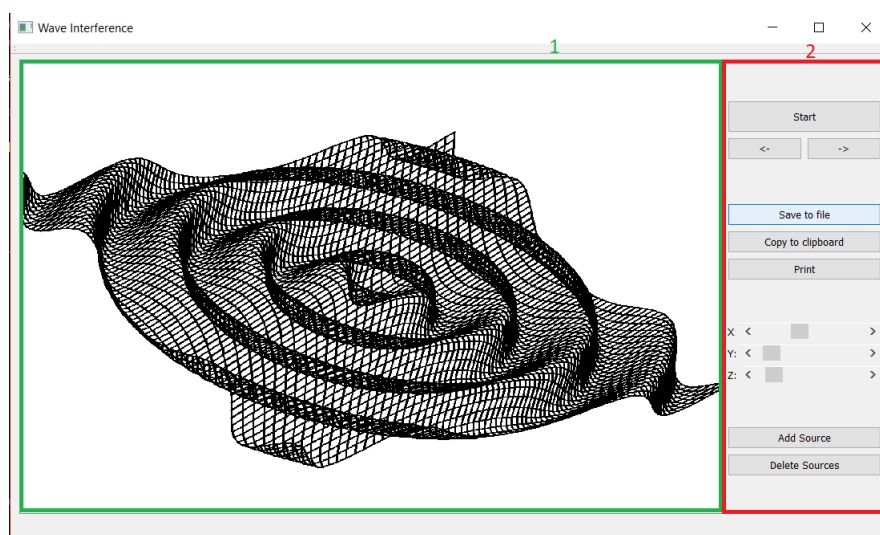
- **Matrix** - macierz 4×4 do przechowywania danych o transformacjach, jakim poddawane są punkty tworzące płaszczyznę
- **Vector4** - reprezentuje punkt 3D we współrzędnych jednorodnych
- **Point** - klasa reprezentująca punkt w przestrzeni trójwymiarowej - zawiera wektor (`std::vector<double>`) odległości od źródeł
- **point_vector** - alias na `std::vector<std::vector<Point>` - przechowuje punkty tworzące płaszczyznę
- **m_points2D** - `std::vector<std::vector<QPoint>` - przechowuje punkty wyświetlane na ekranie (po transformacjach)
- **m_bitmaps** - `std::vector<QPixmap>` - przechowuje kolejne klatki animacji
- **vector_vector** - alias na `std::vector<std::vector<Vector4>` - przechowuje wektory do transformacji (punkty we współrzędnych jednorodnych)
- **Model** - zawiera wszystkie dane i metody potrzebne do stworzenia animacji
- **MainWindow** - służy do komunikacji między użytkownikiem a modelem oraz do wyświetlania animacji (komunikacja odbywa się za pomocą mechanizmu słów i sygnałów)

W programie użyte zostały m.in. następujące klasy wbudowane biblioteki Qt:

- QPoint** - punkt 2D
- QSize** - przechowuje wielkość obiektu klasy QPixmap
- QPixmap** - bitmapa przechowująca pojedynczą klatkę animacji
- QTimer** - do realizacji animacji
- QThread** - realizacja wielowątkowości
- QGraphicsView** do którego został podłączony **QGraphicsScene** - wyświetlanie animacji
- QPainter** - służy do rysowania linii na QPixmap

4.4 Specyfikacja interfejsu użytkownika

Nasza aplikacja składa się z jednego głównego okna. Prezentuje się ono następująco:



Okno podzielone przez nas zostało na dwie główne części:

- 1 - scena wyświetlająca animację fali
- 2 - menu, służące do interakcji z użytkownikiem

W menu znajduje się szereg przycisków:

- Start** - rozpoczyna odtwarzanie animacji
- >** - przechodzi do kolejnej klatki animacji

<- - przechodzi do poprzedniej klatki animacji

Save to file - zapisuje aktualną klatkę animacji do pliku - po wciśnięciu tego przycisku pojawia się standardowe okno dialogowe do wyboru pliku wyjściowego. Uwaga: plik nie zostanie zapisany, jeśli użytkownik nie poda porawnego rozszerzenia (np. png).

Copy to clipboard - zapisuje aktualną klatkę animacji do schowka systemu Windows

Print - drukuje klatkę animacji

suwaki X, Y, Z - sterują rotacją płaszczyzny

Add Source - wyświetla menu dodawania źródła fali

Delete Sources - usuwa wszystkie wstawione na płaszczyznę źródła fali

4.5 Wyodrębnienie i zdefiniowanie zadań

Wykonany przez nas projekt został podzielony na następujące moduły:

- **graficzny interfejs użytkownika i kontroler** - składają się na niego pliki `mainwindow.h` i `mainwindow.cpp` oraz `customdialog.h` i `customdialog.cpp`
 - **struktury danych** - składają się na niego pliki: `matrix.h`, `matrix.cpp`, `vector4.h` oraz `vector4.cpp`
 - **operacje matematyczne** - pliki `transformations.h` oraz `transformations.cpp`
 - **jednostka zarządzająca** - pliki `model.h` oraz `model.cpp`
- Główna funkcja aplikacji zawarta została w pliku `main.cpp`.

4.6 Narzędzia programistyczne

Wykonując projekt wykorzystaliśmy następujące narzędzia:

- Framework Qt w wersji 5.11.0 do stworzenia GUI aplikacji
- Qt Creator w wersji 4.6.1
- Standardowa Biblioteka języka C++
- Git - system kontroli wersji
- serwis Github

Decyzja o wyborze Qt została podjęta głównie z powodu multiplatformowości tego narzędzia - dawało to możliwość pisania zarówno na systemie Windows, jak i Linux. Ponadto członkowie zespołu byli już zaznajomieni z tym frameworkiem, co bardzo ułatwiło pracę. Na decyzję miała również wpływ obszerna dokumentacja

dostępnych w Qt narzędzi, oraz prostota ich używania.

Pod systemem Windows (gdzie odbywało się większość prac) używaliśmy kompilatora Microsoft Visual C++ Compiler 15.0 (AMD 64).

5 Opracowanie i opis niezbędnych algorytmów

W programie wyodrębnić można dwa istotne algorytmy - obliczający położenie punktów w przestrzeni 3D na podstawie funkcji sinus (reprezentującej falę) oraz implementujący transformacje macierzowe z przestrzeni 3D na przestrzeń 2D.

- pierwszy z wymienionych algorytmów wykorzystuje fakt, iż wartość funkcji sinus w przestrzeni zależy jedynie od odległości punktu od źródła. Odległość jest obliczana w przestrzeni Euklidesowej. Algorytm ten modyfikuje z-tową składową każdego z punktów osobno. Wykorzystując przy tym mechanizm wielowątkowości (plansza została podzielona na cztery równe części).

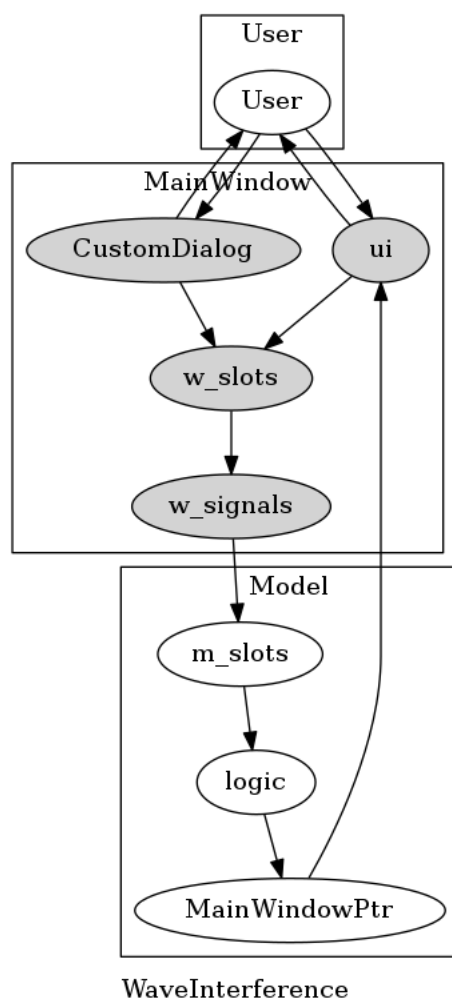
- drugi algorytm odpowiada za wszystkie transformacje macierzowe, którym poddawane są punkty, tzn.: rotacja względem wszystkich trzech osi, skalowanie oraz rzut perspektywiczny. Transformacje te są przeprowadzane na każdym punkcie osobno (reprezentowane jako `vector4`) z wykorzystaniem mechanizmu wielowątkowości w analogiczny sposób jak powyżej. Przykładowa macierz, tu odpowiadająca za rotację wokół osi z, przedstawiona została poniżej:

$$\begin{pmatrix} \cos(x) & -\sin(x) & 0 & 0 \\ \sin(x) & \cos(x) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6 Kodowanie

6.1 Wykres przepływu danych

Poniżej zamieszczony został schematyczny wykres obrazujący przepływ danych w opisywanej aplikacji:



6.2 Opis klas, funkcji oraz zmiennych

W programie wykorzystujemy zarówno klasy służące jako struktury danych oraz klasy realizujące zamierzone działanie. Do pierwszej grupy należą:

Matrix Klasa służąca do reprezentacji macierzy 4x4. Dostęp do poszczególnych pól macierzy realizowany jest za pomocą operatora []. Zaimplementowane są również operatory *, które służą do mnożenia macierz x macierz oraz macierz x wektor.

Vector4 Klasa służąca do reprezentacji wektora 4 wymiarowego. Ustawianie pierwszych trzech wartości odbywa się za pomocą metody `set`. Dostęp do poszczególnych wartości jest realizowany za pomocą operatora `[]`. Zaimplementowane są również operacje dodawania wektor do wektora (operator `+`) oraz odejmowania wektor od wektora (operator `-`).

Point Klasa służąca do reprezentacji punktu w przestrzeni 3D. Przechowuje ona trzy wartości reprezentujące położenia `x`, `y` oraz `z`. Oprócz tego przechowuje ona wektor odległości od aktualnie dodanych źródeł. Dodawanie kolejnych źródeł odbywa się za pomocą metody `add_source_dist`, która przyjmuje `x`-ową oraz `y`-ową współrzędną źródła. Dodatkowo klasa posiada metodę służącą do czyszczenia zawartości wcześniej wspomnianego wektora, czyli `clear_sources`. Konwersja punktu na `Vector4` odbywa się za pomocą metody `return_as_vector`. Oprócz tego dostęp do składowych punktu odbywa się za pomocą przeładowanego operatora `[]`.

Natomiast do drugiej grupy należą klasy:

MainWindow Klasa służąca do definicji interfejsu użytkownika oraz do komunikacji między użytkownikiem, a modelem. Definicja zawartości oraz wyglądu głównego okna jest stworzona za pomocą narzędzia Design zawartego w programie QtCreator. Komunikacja pomiędzy `MainWindow`, a `Model` odbywa się za pomocą mechanizmu 'slot and signals'. Wszystkie akcje użytkownika są przechwytywane przez sloty zamieszczone w klasie `MainWindow`:

```
void z_rotated(int rcv);  
void y_rotated(int rcv);  
void x_rotated(int rcv);  
void start_clicked();  
void next_clicked();  
void previous_clicked();  
void add_clicked();  
void delete_clicked();  
void copy_clicked();
```

```
void save_clicked();
void print_clicked();
Z tych metod emitowane są sygnały:
void window_resized(QSize new_size);
void next();
void previous();
void start_animation();
void stop_animation();
void redraw();
void source_added(int x_pos,int y_pos, double amplitude, double frequency);
void sources_deleted();
void model_clipboard();
void model_save();
void print_frame();
void calculate_matrices();
Które następnie są przechwytywane przez sloty umieszczone w klasie Model.
```

Model Klasa implementująca jednostkę zarządzającą. Przechowuje wszystkie najważniejsze struktury danych. Posiada konstruktor przyjmujący wymiar wyświetlanej płaszczyzny (w punktach). Posiada funkcję repaint służącą do rysowania pojedynczej klatki animacji. Klasa ta posiada następujące sloty służące do przyjmowania informacji do MainWindow:

```
void set_draw_size(QSize draw_size);
void start_animation();
void stop_animation();
void next();
void previous();
void redraw();
void sine_calc(int calc = 1);
void source_added(int x_pos, int y_pos, double amplitude, double frequency);
void sources_deleted();
void model_clipboard();
void model_save();
void print_frame();
void calculate_matrices();
```

CustomDialog Klasa służąca do pobrania wartości charakteryzujących nowe źródło, tj.: położenie x , y , częstotliwość oraz amplitudę. Wartości te mogą zostać pobrane po poprawnym zamknięciu okna, za pomocą metod `get_x`, `get_y`, `get_frequency`, `get_amplitude`.

7 Testowanie

Opisywana aplikacja testowana była pod kątem poprawności implementacji. Testy polegały na podawaniu do programu zróżnicowanych danych wejściowych, w tym również przypadków skrajnych, takich jak bardzo duża ilość źródeł. Parametrami zmienianymi podczas poszczególnych testów były: położenie źródeł i ich ilość, amplituda fali oraz jej częstotliwość. Wykonane testy pozwoliły na poprawę jakości implementacji aplikacji. Osobną gałąź testów stanowiły testy wydajnościowe, pomagające w dążeniu do jak największej optymalizacji projektu. W tym celu aplikacja uruchamiana była przy użyciu różnych danych od użytkownika, a następnie monitorowane było, w jakim stopniu zużywa dostępne zasoby (procesor, pamięć RAM). W miarę powstawanie programu testowane były również niezależne bloki (klasy, funkcje) programu. Sposób testowania był tutaj różny (często sprowadzał się do wypisania wyniku danej operacji w konsoli). W ten sposób testowany były m.in. operacje macierzowe. W celu testowania, czy wprowadzone w kodzie zmiany nie zmieniły w sposób nieakceptowalny i nieprzewidziany działania programu używano często prostych danych wejściowych, takich jak jedno źródło położone na środku płaszczyzny (punkt (500, 500)), o amplitudzie i częstotliwości równym wartości 5.

8 Wdrożenie, raport i wniośki

Projekt został doprowadzony do końca w wyznaczonym terminie. Udało się zrealizować wszystkie założenia, zarówno podstawowe jak i rozszerzone. Znalazło się również miejsce na ulepszenia, takie jak wielowątkowość. Projekt mógłby być rozszerzany w dalszym stopniu - jednym z pomysłów mogłaby być obsługa kolorów (np. kolor linii byłby związany z wartością funkcji falowej w danym miejscu) lub wprowadzenie wizualizacji za pomocą poligonów. Uważamy, że w realizacji

projektu istotny był wybór narzędzia (framework Qt). Dzięki dobrej organizacji zespołu projekt został zakończony w terminie.