Projekt nr 14
Deformacje geometryczne

Igor Sala
Szymon Konieczny
Damian Wędrocha

1. Tytuł projektu i autorzy

W ramach projektu na przedmiot "Podstawy Grafiki Komputerowej" przygotowaliśmy projekt nr 14 "Deformacje geometryczne", który pozwala na poddanie grafiki rastrowej przekształceniom geometrycznym. Autorzy projektu oraz ich podział pracy:

- Igor Sala - stworzenie GUI, nadzorowanie repozytorium GIT, podział pracy, wczytywanie obrazu, definicja struktur danych, algorytm obrotu w osi obrazu i w pozostałych osiach, operacja undo, pisanie dokumentacji, testy.
- Szymon Konieczny - algorytm pochylający obrazek, algorytm obcinania ramki (używany w innych algorytmach), zapisywanie obrazu, pisanie dokumentacji, testy.
- Damian Wędrocha - algorytm eliminacji zniekształceń wykorzystujący zaawansowany model uwzględniający parametry korekcyjne obiektywu, algorytm odbicia lustrzanego względem osi x i y. Pomoc w pisaniu pozostałych algorytmów, testy.

2. Opis projektu

Celem projektu jest utworzenie oprogramowania pozwalającego na modyfikację dowolnego pliku graficznego obejmującą jego przekształcenia geometryczne.

3. Założenia wstępne

Projekt polega na stworzeniu kompleksowej aplikacji do obróbki geometrycznej obrazów. Program ma zapewniać funkcjonalności opisane jako "wymagania podstawowe" takie jak:

a) obrót wokół osi prostopadłej do płaszczyzny rejestrującej;

b) obrót wokół osi prostopadłej i równoległej leżącej w płaszczyźnie rejestrującej;

c) przechylanie obrazu;

d) eliminacja zniekształceń typu "beczka" i "poduszka".

Ponadto program pozwala na wczytanie zdjęcia w kilku formatach (.bmp, .jpeg, .jpg, .png, .tiff, .ppm), zapis obrazu w wymienionych formatach oraz rozszerzenie powyższych "wymagań podstawowych" obejmujące zmianę osi wokół, której przeprowadzony jest obrót w płaszczyźnie rejestrującej, zmianę punktu obserwatora w osi Z, uwzględnienie zaawansowanego modelu obejmującego parametry korekty obiektywu.

4. Analiza projektu

4.1. Specyfikacja danych wejściowych

Program powinien wczytywać pliki graficzne o dowolnym rozszerzeniu oraz dokonywać ich modyfikacji.

4.2. Opis oczekiwanych danych wyjściowych

Jako oczekiwane dane wyjściowe opisujemy obrazy na których dokonane zostały odpowiednie przekształcenia - gotowe do zapisania przez użytkownika na dysku lub zapisane w takiej postaci aby można było dokonywać bardziej kompleksowych przekształceń obrazu, dane wyjściowe jednego algorytmu mogą zostać przekazane jako dane wejściowe do innego algorytmu.

4.3. Zdefiniowanie struktur danych

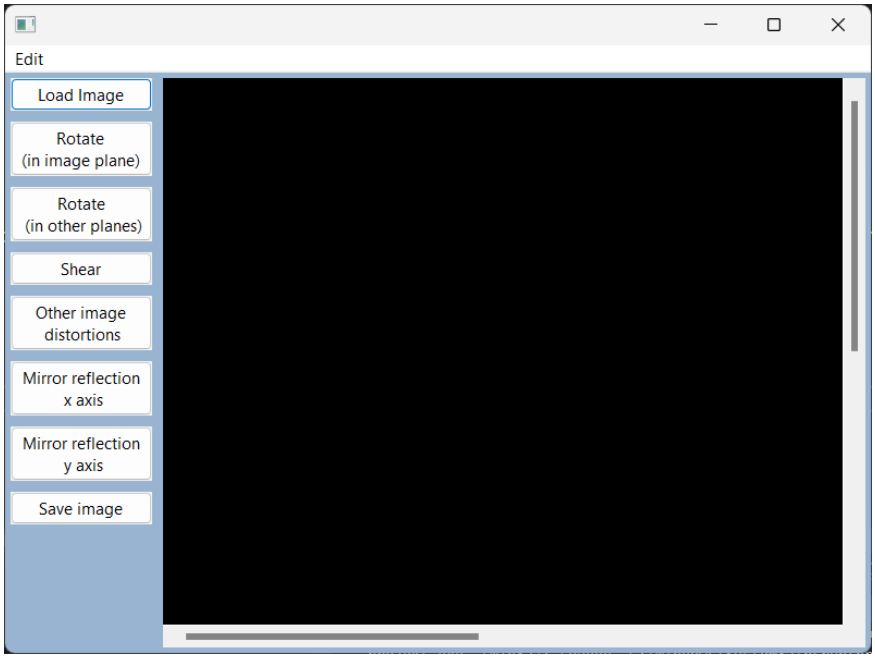
W projekcie wykorzystywaliśmy głównie struktury danych typowe dla języka C/C++ czyli tablice oraz std::vector. Tablice były wykorzystane chociażby do przetworzenia obiektów wxImage przechowujących dane dotyczące obrazu wczytywanych przez program, std::vector idealnie sprawdzało się w symulowaniu działania listy zawierającej historię przekształceń.

Przy pomocy wxFormBuildera wygenerowano klasę GUI, po której dziedziczy klasa GUIMyFrame1 która jest główną klasą programu . Klasa ta została zdefiniowana przez nas korzystając z możliwości języka C++ rozszerza działanie klasy bazowej wykorzystując polimorfizm i dziedziczenie.

W projekcie wykorzystaliśmy również klasy używane już przez nas wcześniej na zajęciach Vector4 i Matrix4, które wczytywały dane reprezentujące czterowymiarowy wektor oraz macierz 4x4. Dodatkowo Vector4 został przerobiony w taki sposób, aby przechowywał informację o kolorze punktu, który dany wektor reprezentuje na płaszczyźnie obrazu. Dla obu klas przeciążyliśmy odpowiednie operatory pozwalające na dokonywanie mnożenia macierzy z macierzami i macierzy z wektorem.

4.4. Specyfikacja interfejsu użytkownika

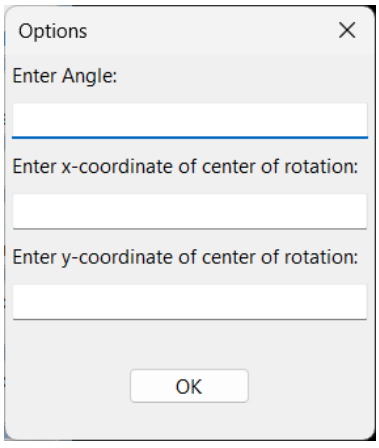
W przypadku interfejsu użytkownika naszego programu postanowiliśmy postawić na przejrzystość i prostotę. Okno naszego programu podzielone jest na dwie części: panel boczny i panel z obrazem. Na górze w górnym panelu stworzenie jest pole Edit->Undo.



Rysunek 1: Okno programu.

Jak można zauważyć panel w którym znajduję się zdjęcie jest przewijalny ponieważ pewne deformacje obrazu mogą prowadzić do zmiany jego wielkości - w celu przekazywania informacji do użytkownika jak aktualnie wygląda modyfikowany obraz zdecydowaliśmy się na takie rozwiązanie.

Panel boczny służy do wyboru przekształcenia jakie chcemy wykonać. Naciśnięcie dowolnego przycisku powoduje otworzenie się nowego okna dialogowego pozwalającego na wybór parametrów dotyczących danego przekształcenia.



Rysunek 2: Przykładowe okno wyświetlające się po kliknięciu przycisku odpowiadającego pewnej deformacji.

4.5. Wyodrębnienie i zdefiniowanie zadań

1. Analiza zadania. Przypomnienie sobie zagadnień z przedmiotu. Definicja zadań.

2. Wybór środowiska programistycznego i wykorzystanych narzędzi, utworzenie projektu oraz repozytorium GIT.
3. Zaprojektowanie interfejsu graficznego wykorzystując program wxFormBuilder.
4. Implementacja działania aplikacji, utworzenie wyskakujących okienek. Przygotowanie kodu pod implementację algorytmów.
5. Opracowanie algorytmów deformacji geometrycznych i ich implementacja.
6. Testowanie aplikacji, naprawa błędów oraz usuwanie rzucanych wyjątków.
7. Kontakt z prowadzącym w celu zweryfikowania poprawności działania algorytmów.
8. Poprawa błędów oraz ponowne testy, debugging etc.
9. Częściowa refaktoryzacja kodu.
10. Sporządzenie dokumentacji.

4.6. Decyzja o wyborze narzędzi programistycznych

W celu wykorzystania możliwości dobrego środowiska programistycznego takich jak; łatwy debugging oraz zarządzanie plikami projektu, linkowanie i kompilacja, postawiliśmy na Visual Studio. W celu efektywnej współpracy nad kodem użyliśmy systemu kontroli wersji GIT. Wykorzystaliśmy framework wxWidgets do tworzenia aplikacji okienkowej ze względu na jego użyteczność, znajomość z zajęć oraz wieloplatformowość (pomimo tworzenia naszej aplikacji w Visual Studio co wymusza system operacyjny Windows nasz projekt jest prosty do przeniesienia na inną platformę, gdzie do kompilacji i linkowania projektu można użyć narzędzia CMake). Ponadto wykorzystaliśmy wxFormBuilder do wygenerowania GUI w celu przyspieszenia procesu jego wykonywania.

5. Podział pracy

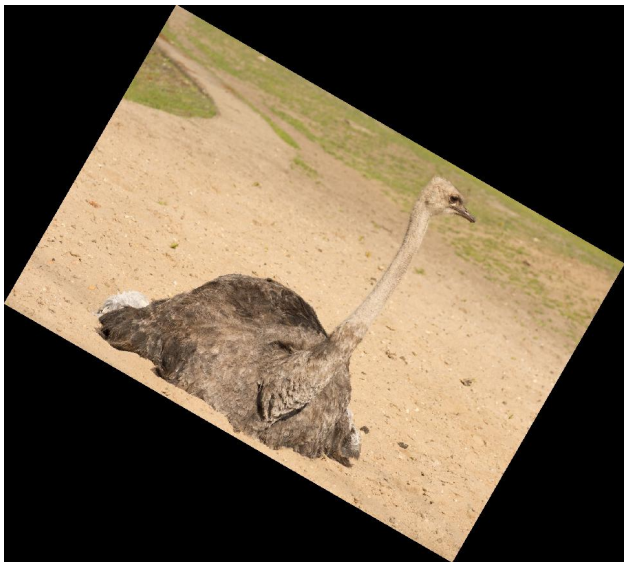
Podział pracy pod względem zadań został przedstawiony w punkcie pierwszym. Przybliżony czas poświęcony kluczowym etapom wykonania projektu:

1. Powtórzenie zagadnień, przygotowanie algorytmów, podział zadań, przygotowanie środowiska, utworzenie repozytorium - 10h
2. Wygenerowanie GUI (głównego okna oraz wyskakujących okienek) - 5h
3. Przygotowanie kodu pod implementację algorytmów - 6h
4. Implementacja odpowiednich algorytmów - 20h
5. Dyskusje na platformach komunikacyjnych odnośnie dokonywanych postępów - 6h
6. Naprawa błędów, testy - 12h
7. Poprawienie czytelności kodu, refaktoryzacja, drobne poprawki działania - 14h
8. Napisanie dokumentacji - 8h

6. Opracowanie i opis niezbędnych algorytmów

6.1. Algorytm obrotu w płaszczyźnie ekranu

Obrót w płaszczyźnie ekranu zaczynamy od wyznaczenia współrzędnych w jakich znajdą się narożniki ekranu w celu późniejszego odnoszenia się do nich w znajdowaniu się punktów obróconego obrazu, oraz w ustaleniu jego nowej wielkości. Najpierw więc przeprowadzamy translację o wektor o współrzędnych przeciwnych do współrzędnych punktu obrotu obrazu, obrót i translację powrotną dla punktów narożnych. Następnie znajdujemy punkty w których współrzędna x i y przyjmuję wartości maksymalne i minimalne. Uzyskane dane pozwalają nam na określenie wielkości wynikowego obrazu. Następnie przedstawiony już algorytm obrotu stosujemy dla pozostałych punktów w odwrotnej logice - sprawdzamy gdzie znajdował się dany punkt przed obrotem w celu ułatwienia procesu określenia koloru danego piksela.



Rysunek 3: Obrót o 30 stopni

6.2. Algorytm obrotu w płaszczyznach innych niż płaszczyzna obrazu

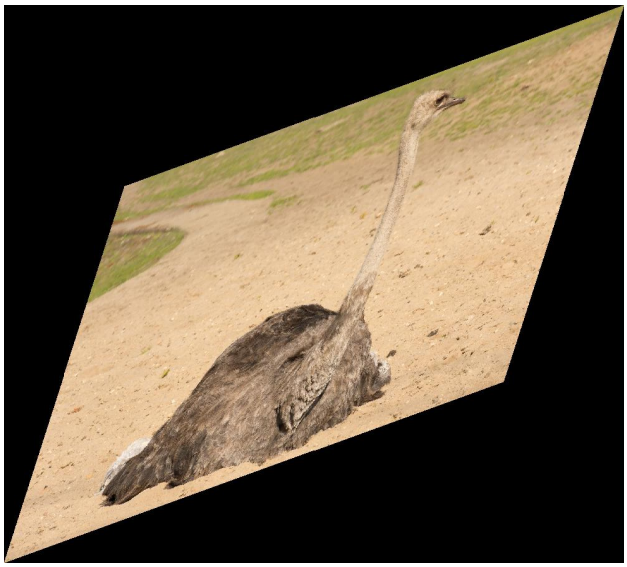
Istota tego algorytmu jest podobna jak tego opisanego powyżej. Również korzystając z macierzy przekształceń dokonujemy translacji przed dokonaniem obrotu. Dopuszczamy obrót w każdej osi obrazu. Następnie każdy punkt obrazu obracamy o odpowiedni kąt. Następnie dokonujemy rzutu perspektywicznego obrazu na ekran. Tutaj określamy odległość ekranu od obserwatora. Następnie wypełniamy potencjalnie powstałe puste pola algorytmem interpolującym.

Rysunek 4: Obrót wokół osi X

6.3. Algorytm przechylania obrazu

Pierwszym krokiem jest wyznaczenie rozmiaru wynikowego obrazu. W tym celu wykorzystujemy funkcję trygonometryczną tangens.

Następnie przepisujemy dane z przechylanego obrazu wierszami (przechylanie w poziomie) lub kolumnami (przechylanie w pionie) poprzedzone odpowiednim odstępem którego wielkość zmienia się liniowo wraz z numerem przepisywanego wiersza/kolumny, na przykład dla przechylenia o ujemny kąt w poziomie im wyższy numer wiersza tym większe przesunięcie względem lewej krawędzi obrazka.



Rysunek 5: przechył o 20 stopni w poziomie i 20 stopnie w pionie

6.4. Algorytm obcinania ramki

Algorytm obcina pustą ramkę wokół obrazu, gdzie "ramka" to ciąg kolejnych wierszy lub kolumn całkowicie wypełnionych kolorem czarnym znajdujących się bezpośrednio przy krawędzi obrazu.

Algorytm znajduje puste obszary z każdej strony obrazka, a następnie tworzy obraz pomniejszony o znalezioną ramkę.

6.5. Algorytm poprawy zniekształceń spowodowanych wadami obiektywu

W wersji podstawowej eliminacja zniekształceń polegałaby na znajdowaniu nowej pozycji dla danego piksela na podstawie podanego współczynnika deformacji odległości względem środka obrazu.

Wersja rozszerzona zakłada 4 parametry korekcji:

1. Parametr A - wpływający w zasadzie tylko na najdalsze piksele
2. Parametr B - będący bardziej uniwersalnym współczynnikiem korekcji, który to w większości wypadków jako jedyny jest potrzebny do naprawy obrazu
3. Parametr C - będący najbardziej uniwersalnym parametrem, który przy braku podania reszty działa w wersji podstawowej
4. Parametr D - odpowiedzialny za skalowanie obrazu lub jego brak

Korekcja w wersji rozszerzonej polega na znalezieniu niezniekształconej odległości od środka zdjęcia dla danego piksela jako wartości wielomianu stopnia 3, w którym to powyższe parametry stoją odpowiednio przy potędze od najwyższej do najniższej, a wartością "x"jest obecna odległość względem centrum obrazu. Na podstawie tak znalezionego promienia proporcjonalnie zmniejszane są współrzędne x oraz y dla danego piksela.

(a) Zniekształcony obraz

(b) Naprawiony obraz

Rysunek 6: Poprawa zniekształcenia typu beczka

(a) Zniekształcony obraz

(b) Naprawiony obraz

Rysunek 7: Poprawa zniekształcenia typu poduszka

6.6. Algorytm operacji undo

Operacja polega na cofnięciu wprowadzonej zmiany. Cofana jest tylko ostatnia modyfikacja. W praktyce aktualny obraz podmieniany jest na zapisaną wersję sprzed modyfikacji.

6.7. Algorytm lustrzanego odbicia

Prosty algorytm odbijający lustrzanie obraz względem osi X lub osi Y. Schemat działania zamienia kolory pikseli o tej samej odległości od wskazanej osi, leżące po różnych jej stronach.

7. Kodowanie

7.1. Algorytm obrotu w płaszczyźnie ekranu

Metoda obracająca przyjmuje jako parametry kąt obrotu w stopniach oraz współrzędne środka rotacji.

Następuję pobranie kopii obrazka do zmiennej wxImage oraz pobranie jest wysokości, kąt obrotu przeliczany jest na wartość w radianach. Program inicjalizuje tablicę corners w której znajdują się początkowe współrzędne narożników. Następnie przeprowadzamy obrót dla narożników wykorzystując odpowiednie przekształcenia i funkcje trygonometryczne. Określamy współrzędne krawędzi obróconego obrazu i na ich podstawie wyznaczamy wielkość obrazu oraz offset. Następnie wykorzystując dwie pętle for ponawiamy algorytm dla punktów należących do wynikowego obrazu, wykorzystaną wcześniej funkcje trygonometryczną sinus poprzedzimy minusem (znaku przed cosinusem ze względu na jego parzystość nie musimy zmieniać). Kolor piksela pobieramy z początkowego obrazu funkcjami Get i ustawiamy na wyznaczony kolor metodą SetRGB.

Otrzymany obraz zapisujemy do wskaźnika do obiektu klasy wxImage przechowującej dane obrazu, który rysujemy w panelu. Na koniec obcinamy pustą ramkę i rysujemy zmienione zdjęcie w panelu.

7.2. Algorytm obrotu w płaszczyznach innych niż płaszczyzna obrazu

Funkcja jako argumenty przyjmuje wartości całkowite, które odpowiadają pozycji suwaków w oknie dialogowym. Są to kąty obrotu w wokół prostopadłych do siebie nawzajem osi oraz parametry modyfikujące pozycję obserwatora oraz osi wokół których obraz się obraca. Pozostają one również normalizowane na etapie początkowym.

Wczytujemy dane z obrazu wejściowego i zapisujemy do tablicy unsigned char. Następnie wypełniamy tablicę vectors przechowującą współrzędne punktów obrazka oraz ich kolory - współrzędne są normalizowane. Następnie definiujemy macierze odpowiadające przekształceniom. Każdy punkt w tablicy mnożymy przez odpowiednie macierze. Gdy pomnożymy wektory przez macierz perspektywy dzielimy współrzędną x i y przez współrzędną w. Jednocześnie współrzędne mnożyliśmy przez macierz, która dostosowała wielkość zdjęcia wynikowego. Przez to, że wcześniej normalizowaliśmy do współrzędnych $\in$ (-1,1) teraz współrzędne należą do przedziału (-width/2, width/2), dla szerokości, dlatego do każdego punktu dodajemy width/2. Analogicznie postępujemy dla wysokości. Kolorujemy odpowiednie piksele obrazu metodą SetRGB - starając się zakrywać również te piksele, które nie mają swojego reprezentanta w tablicy wektorów.

Otrzymany obraz zapisujemy do wskaźnika do obiektu klasy wxImage przechowującej dane obrazu, który rysujemy w panelu. Na koniec obcinamy pustą ramkę i rysujemy zmienione zdjęcie w panelu. Nie pomijamy zwalniania zaalokowanej dynamicznie pamięci co robimy w osobnej funkcji.

7.3. Algorytm przechylania obrazu

Algorytm został rozdzielony pomiędzy dwie funkcje, ale obie działają analogicznie. Jedna obsługuje obrót w poziomie, a druga w pionie. Obie przyjmują jako parametr kąt obrotu w stopniach. Jeżeli użytkownik użyje jednocześnie przechyleń w obu osiach to najpierw wykonane zostanie przesunięcie w poziomie, a dopiero potem w pionie.

Algorytm przechylania w poziomie

1. Zamieniamy kąt na radiany. Pobieramy wymiary obrazka.
2. Wyliczamy zmianę szerokości jako tangens kąta pomnożony przez wysokość
3. Alokujemy pamięć na obraz wynikowy.
4. Wyznaczamy punkt referencyjny (połowa zmiany szerokości) względem którego będziemy przepisywać dane. Pozwoli on nam na obsługę kątów dodatnich jak i ujemnych za pomocą tej samej pętli.
5. Przepisujemy kolejne wiersze za każdym razem wyliczając przesunięcie względem punktu referencyjnego, jako $(\frac{wysokosc}{2} - nr\_wiersza) \cdot tan(\varphi)$.
6. Obcinamy ramkę i rysujemy nowy obraz

Algorytm przechylania w pionie
Działanie identyczne jak dla wersji poziomej. Różniące się punkty zostały wypisane poniżej:

2. Wyliczamy zmianę wysokości jako tangens kąta pomnożony przez szerokość
4. Tym razem punkt referencyjny to połowa zmiany wysokości
5. Przepisujemy kolejne kolumny za każdym razem wyliczając przesunięcie względem punktu referencyjnego jako $(\frac{szerokosc}{2} - nr\_kolumny) \cdot tan(\varphi)$.

7.4. Algorytm obcinania ramki

Algorytm został podzielony na dwie funkcje. Jedna wycina krawędź lewą i prawą, a druga górną i dolną.

Obie funkcje działają w analogiczny sposób. Najpierw szukają wielkości obu pustych obszarów poprzez zliczanie wierszy/kolumn w których znajdują się tylko punkty o kolorze RGB(0,0,0). Liczenie zaczyna się od krawędzi obrazu i kończy na pierwszym napotkanym kolorowym pikselu. Następnie tworzony jest nowy obraz o pomniejszonym rozmiarze i przepisywane są do niego kolejne wiersze z pominięciem wyznaczonej ramki. Funkcje zwracają wskaźnik do nowego obrazu.

7.5. Algorytm poprawy zniekształceń spowodowanych wadami obiektywu

Metoda odpowiedzialna za poprawe owych zniekształceń przyjmuje jako argumenty opisane wcześniej parametry jako liczby zmiennoprzecinkowe oraz flagę w postaci zera lub jeden z zależności czy zaznaczone zostało pole w oknie dialogowych, odpowiedzialne za automatyczne ustawienie parametru D jako różnicy 1 - parametr A - parametr B - parametr C w celu uniknięcia skalowania obrazu.

Następnie wczytywane są wysokość i szerokość oraz dane kolorów pikseli. Na tej podstawie tworzona jest tablica wektorów przechowująca owe kolory oraz pusta tablica wektorów, która później wypełniona zostanie poprawionymi danymi. Kolejnym krokiem jest ustawienie skali jako krótszej wielkości zdjęcia z celu normalizacji odległości aby podawane parametry nie musiały być zmieniane względem rozmiarów obrazu.

Następnie ustalane są współrzędne środka i dokonywane transformacje dla każdego piksela. Polegają one na wyliczeniu aktualnej odległości od środka, a następnie użyciu tej wartości jako zmiennej x w wielomianie $Ax^3 + Bx^2 + Cx + D$, którego wynikiem jest szukana odległość piksela od środka obrazu. Z tak otrzymanego wyniku przeskalowywane są współrzędne x oraz y każdego piksela. Mając te dane budowany jest nowy obraz na zasadzie przypisywania aktualnemu pikselowi koloru piksela, który powinien być na jego miejscu. Finalnie rysowany jest nowy obraz.

7.6. Algorytm operacji undo

Rozbudowujemy inne metody o zapisywanie aktualnego stanu obrazu do kontenera std::vector.

Po naciśnięciu przycisku w menu Edit->Undo usuwany jest aktualny obraz i ustawiany jako aktualny zapisany stan sprzed jednej modyfikacji. Zwalniamy pamięć w kontenerze w przypadku wczytania nowego programu i przy zamykaniu okna programu.

7.7. Algorytm lustrzanego odbicia

Funkcja odbijająca obiekt przyjmuje jako parametr flagę wskazującą na to, który rodzaj odbicia został wywołany.

Następnie wczytuje dane obrazka, a następnie wypełnia nimi tablice wektorów oraz tworzy pusty wektor pomocniczy do zamieniania właściwości wektorów z tablicy.

W kolejnym kroku metoda sprawdza wartości flagi i w zależności czy jest to 0 czy 1 (1 - odbicie względem osi Y, 0 - względem osi X) oblicza połowę długości lub wysokości i iterując po tablicy zamienia wartościami kolorów odpowiednie wektory wykorzystując wektor pomocniczy.

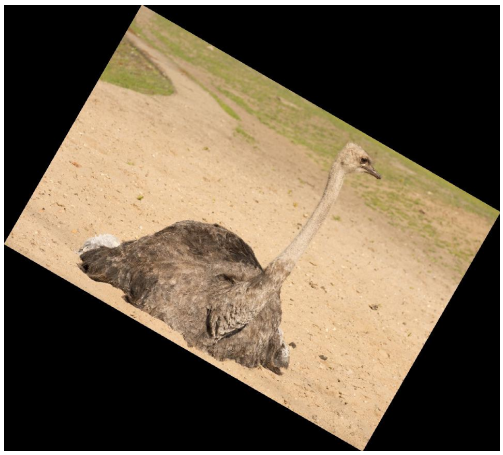
Po wprowadzonych zmianach metoda na nowo rysuje obrazek.

8. Testowanie

1. Testy niezależnych bloków przeprowadzaliśmy na bieżąco przy przy pisaniu oraz dodatkowo na koniec jeszcze raz sprawdziliśmy każdą ich funkcjonalność. W tej części wykryliśmy i naprawiliśmy liczne błędy, np. błędne wyznaczanie nowego rozmiaru przy obcinaniu ramki powodowało ucinanie jednego wiersza/kolumny z właściwego obrazu.

A) Obracanie obrazu w jego płaszczyźnie:

- Obrót o 30 stopni względem punktu o współrzędnych (0;0)



Rysunek 8: Obrót o 30 stopni

- Obrót obrazu o 90 dwukrotnie i o 180 stopni

(a) Obraz obrócony dwukrotnie o 90 stopni

(b) Obraz obrócony raz o 180 stopni

Rysunek 9: Test obrotu w płaszczyźnie obrazu

- Obrót o 30 stopni w prawo i następnie z powrotem o 30 stopni w lewo.

Rysunek 10: Obrót o 30 stopni i z powrotem

Wszystkie testy przebiegły zgodnie z oczekiwaniami.

B) Obracanie obrazu w innych płaszczyznach - W tym punkcie ze względu na sposób napisania kod nie zakładaliśmy, że obrót o 30 stopni w pewnej osi - zapisanie obrazu i obrót w tej samej osi o -30 stopni da ten sam efekt.

- Obrót o około 30 stopni w osi x z różnymi parametrami odległości od obiektu

(a) Obraz obrócony - maksymalna dostępna odległość do obrazu

(b) Obraz obrócony - minimalna dostępna odległość od obrazu

Rysunek 11: Test obrotu w płaszczyznach innych niż płaszczyzna obrazu

- Obrót o około 30 stopni w osi y z różnymi parametrami odległości od obiektu

(a) Obraz obrócony - maksymalna dostępna odległość do obrazu

(b) Obraz obrócony - minimalna dostępna odległość od obrazu

Rysunek 12: Test obrotu w płaszczyznach innych niż płaszczyzna obrazu

- Obrót w o 30 stopni w osi X przy różnej osi obrotu

Rysunek 13: Obrót o 30 stopni w osi równoległej osi x - wynik

Wszystkie testy przebiegły zgodnie z oczekiwaniami choć dla większych plików graficznych problemem była wydajność działania programu. Moglibyśmy to usprawnić wykorzystując wielowątkowość.

C) Pochylanie obrazu:

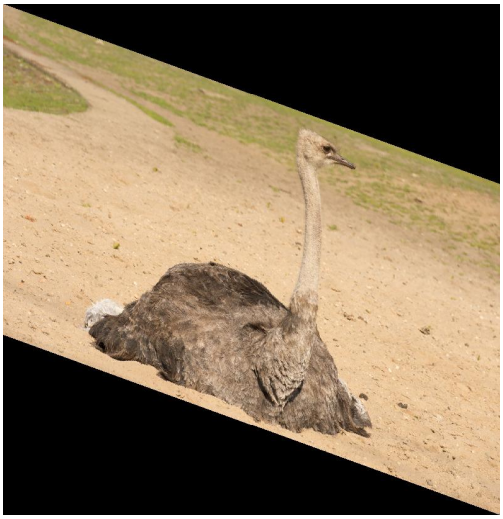
- Testy pochyłu w osi X o kąt 30^o oraz oddzielnie w osi Y o kąt -20^o.

(a) Początkowy obraz

(b) Przechylony obraz

Rysunek 14: Test przechylenia o 30^o w osi X

(a) Początkowy obraz



(b) Przechylony obraz

Rysunek 15: Test przechylenia o -20^o w osi Y

- Testy pochyłu już przechylonego obrazu przeprowadzono najpierw przechylając w osi X, a następnie w osi Y dla zestawu kątów: $X = -20^o$, $Y = 10^o$.

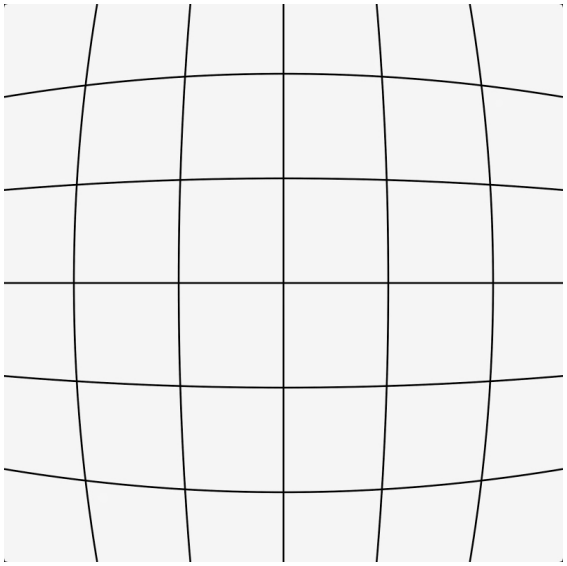
(a) Obraz przechylony o -20^o w osi X

(b) Obraz dodatkowo przechylony o 10^o w osi Y

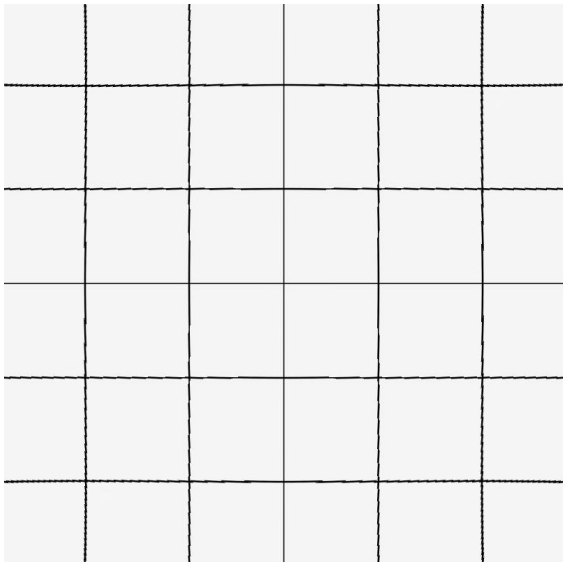
Rysunek 16: Test nakładania się przechyleń

- Sprawdzono czy przechylenie o kąt α, a następnie o kąt $-\alpha$ zwraca początkowy obraz. Test wyszedł pozytywnie dla osi X i dla osi Y.

D) Deformacje związane z wadami obiektywu



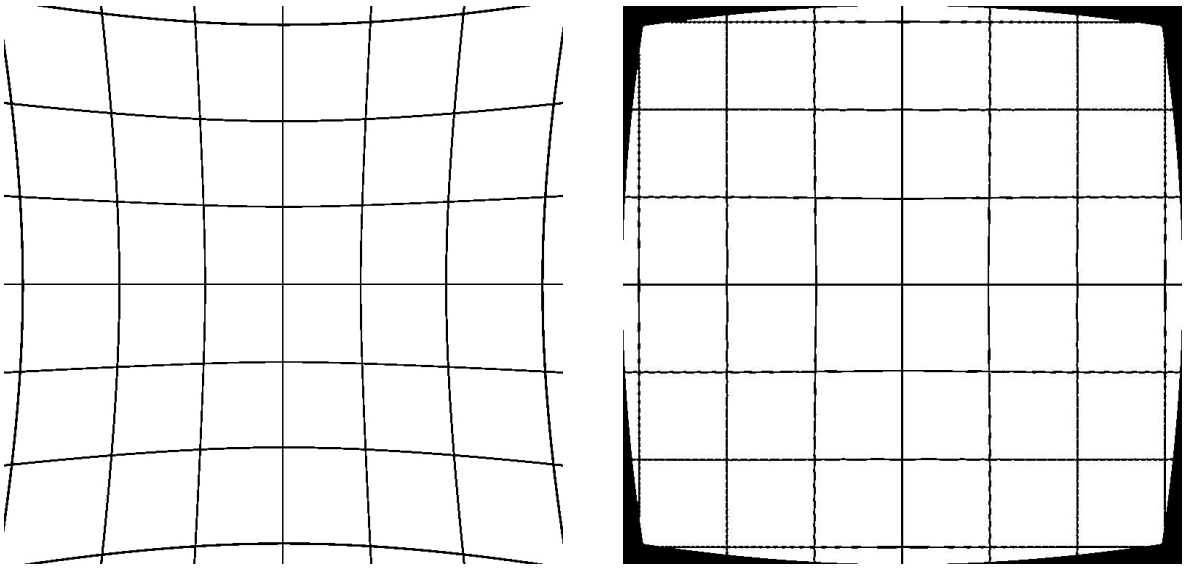
(a) Początkowy obraz



(b) Naprawiony obraz

Rysunek 17: Naprawa deformacji typu "beczka"

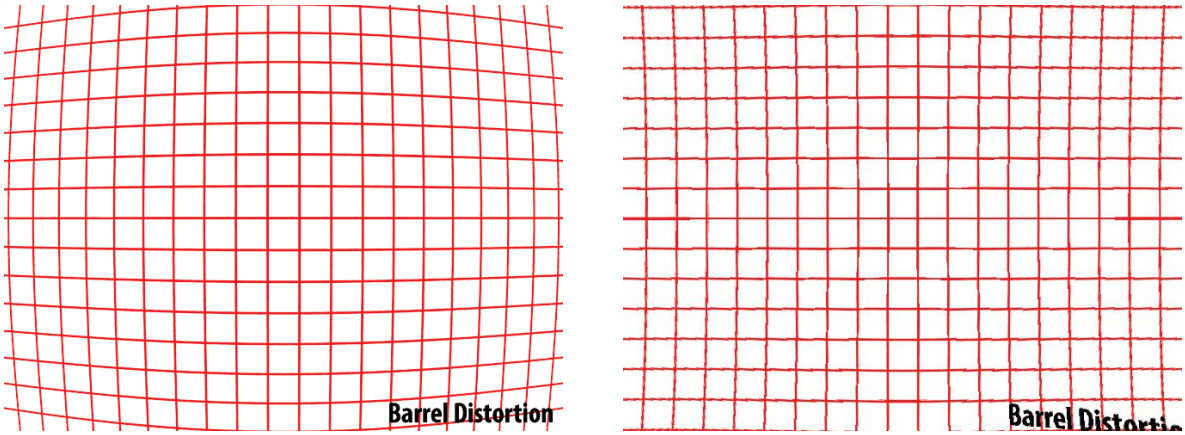
Test został wykonany dla parametrów: A = domyślnie (0), B = -0.07, C = -0.07, D = domyślnie (1 - A - B - C).



(a) Początkowy obraz (b) Naprawiony obraz

Rysunek 18: Naprawa deformacji typu "poduszka"

Test został wykonany dla parametrów: A = domyślnie (0), B = 0.11, C = domyślnie (0), D = domyślnie (1 - A - B - C).



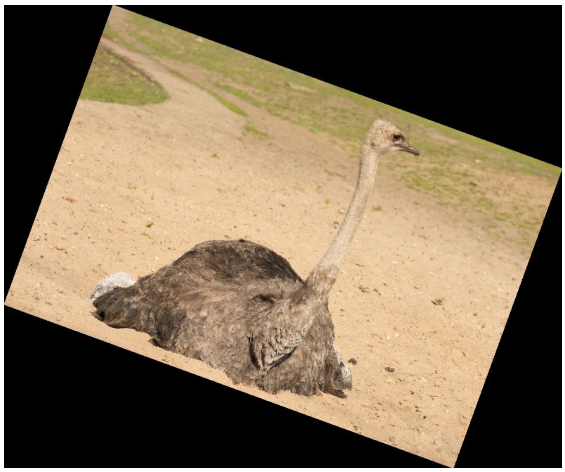
(a) Początkowy obraz (b) Naprawiony obraz

Rysunek 19: Naprawa deformacji typu "beczka"dla niekwadratowego obrazu

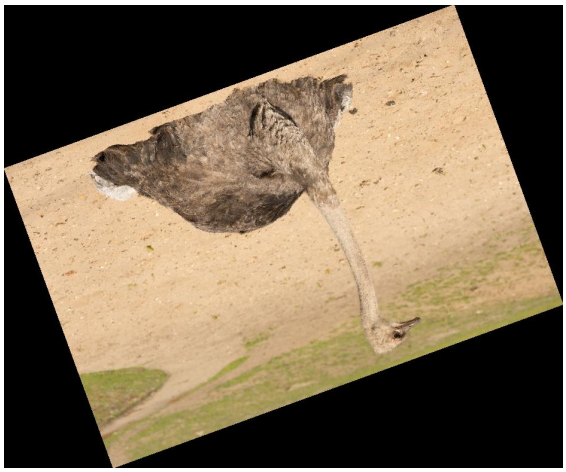
Test został wykonany dla parametrów: A = domyślnie (0), B = -0.036, C = -0.036, D = domyślnie (1 - A - B - C).

Ostatni test pokazuje lekko gorsze działanie metody dla obrazów niekwadratowych, co mogło być oczekiwane ze względu na dobór algorytmu. Jednak funkcja nadal działa poprawnie.

E) Odbicie lustrzane



(a) Początkowy obraz



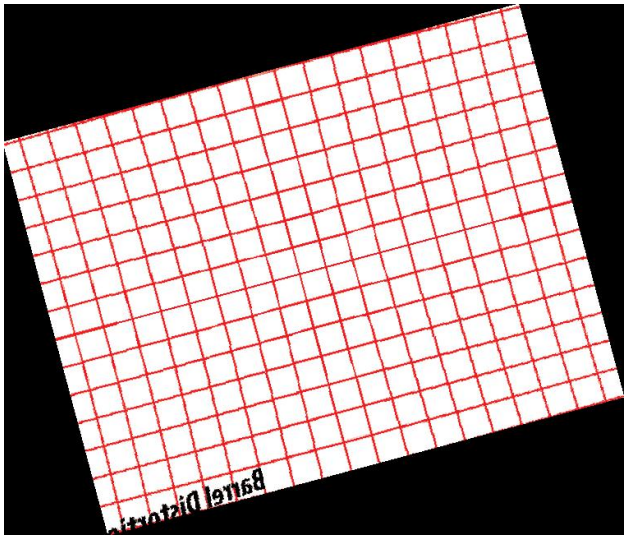
(b) Obraz odbity względem osi X

(c) Obraz odbity względem osi Y

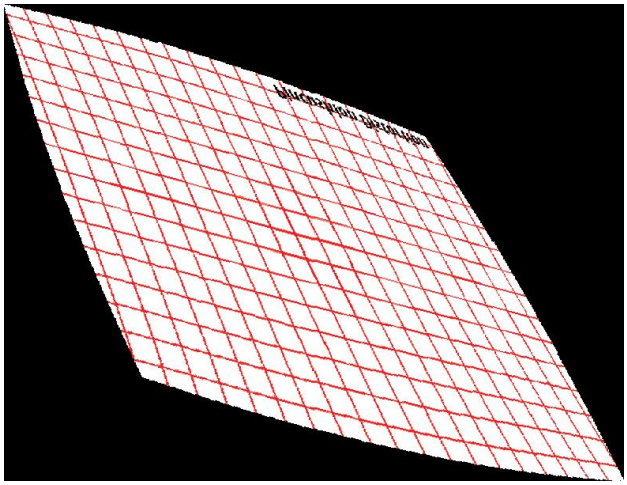
Rysunek 20: Test odbicia lustrzanego

F) Testy zapisywania i wczytywania różnych typów plików pozwoliły nam na wykrycie, że program nie obsługuje ".gif". Pozostałe rozszerzenia działają poprawnie.

2. Testy powiązań bloków przeprowadzaliśmy zawsze po implementowaniu każdego z bloków. Jednocześnie testowaliśmy działanie operacji undo. Sprawdzaliśmy czy nowy blok działa poprawnie na obrazach zmodyfikowanych przez inne bloki, np. test czy pochylony obraz dobrze się obraca i vice versa. W tym punkcie nie napotkaliśmy błędów.

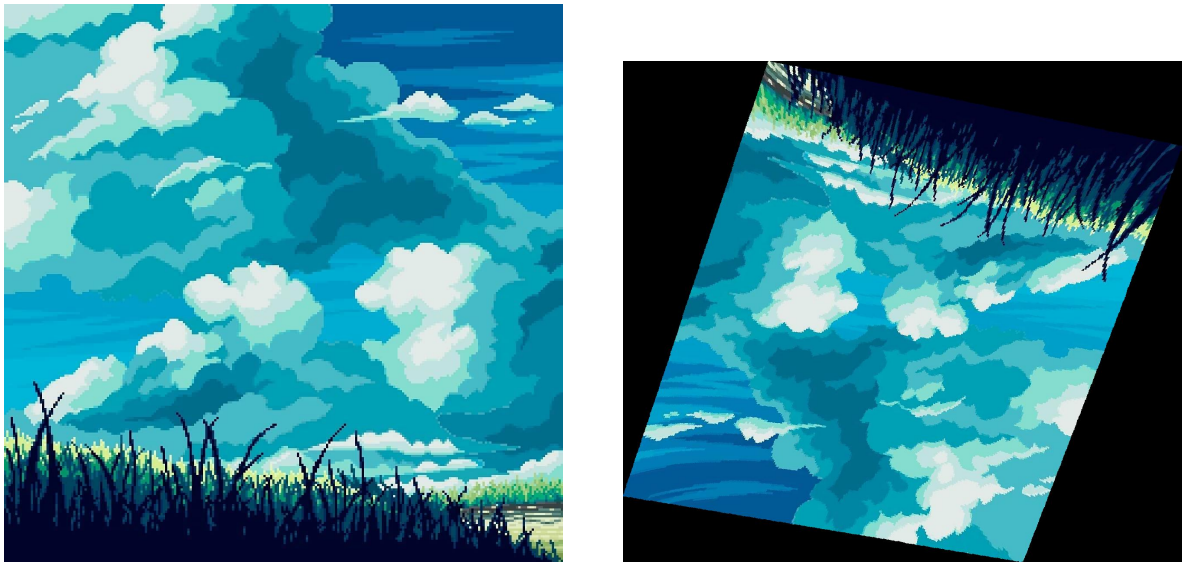


Rysunek 21: Przykładowy test nr 1: Naprawa zniekształcenia typu "beczka"→ obrót w płaszczyźnie obrazka → odbicie lustrzane względem osi Y



Rysunek 22: Przykładowy test nr 2: Naprawa zniekształcenia typu "poduszka"→ obrót w innych płaszczyznach z przesunięciem osi → pochylenie obrazu → odbicie lustrzane względem osi X

3. Testy całościowe miały miejsce na koniec pisania całego programu. W tej części zwróciliśmy szczególną uwagę na to czy w przypadku użycia różnych funkcji wielokrotnie w przeróżnych konfiguracjach zachowana jest poprawność działania programu oraz przetestowaliśmy działanie na różnych obrazach (np. bardzo dużych). Zauważyliśmy, że program działa znacząco wolniej dla większych obrazów (szczególnie obrót wokół osi w płaszczyźnie obrazu), ale pomimo tego program działa poprawnie.



(a) Początkowy obraz (b) Zmodyfikowany obraz

Rysunek 23: Test nałożenia wszystkich modyfikacji

9. Wdrożenie, raport i wnioski

Program spełnia wszystkie podstawowe założenia projektowe. Spośród rozszerzonych wymagań zabrakło siatki edycyjnej. W zamian za to dodaliśmy funkcję Undo i odbicie lustrzane.

Wszystkie przeprowadzone testy zakończyły się pomyślnie, jednak uważamy, że w przyszłości należałoby poprawić wydajność oraz unowocześnić interfejs graficzny - być może skorzystać z innych narzędzi niż wxWidgets.

W trakcie realizacji projektu zdobyliśmy doświadczenie pracy grupowej oraz mogliśmy zmierzyć się z bardziej złożonym problemem niż dotychczasowe. Wyciągnęliśmy wnioski, które w przyszłości pozwolą nam na sprawniejszą pracę nad tak rozbudowanymi projektami.