Grafika komputerowa

Symulacja głębi ostrości

Projekt 37

Akademia Górniczo – Hutnicza
im. Stanisława Staszica w
Krakowie

Autorzy:
Paweł Kępka
Mikołaj Domański

1. Tytuł projektu i jego autorzy

Celem projektu „Symulacja głębi ostrości” jest stworzenie aplikacji, która będzie dokonywała odpowiednich przekształceń na obrazie, za pomocą mapy głębokości danego obrazu.

Projekt został zrealizowany przez 2 osobowy zespół w składzie:

- Paweł Kępka – odpowiedzialny za komunikacje z użytkownikiem, GUI, tworzenie dokumentacji oraz efekt końcowy.
- Mikołaj Domański – odpowiedzialny za stworzenie algorytmów, które dokonują zadane przekształcenia oraz testowanie końcowego programu.

2. Opis projektu

Głębia ostrości to parametr stosowny w optyce i fotografii, który określa zakres odległości, w którym obiekty obserwowane przez dane urządzenie optyczne sprawiają wrażenie ostrych.

Rzeczywiste obiektywy tworzą nam takie obrazy, których głębia ostrości jest skończona, czyli wyostrzają nam się obiekty, na które nastawimy ostrość, a pozostałe będą rozmazane.

Obrazy wygenerowane za pomocą programów 3D charakteryzują się nieskończoną głębią ostrości, czyli są tak samo ostre w każdym punkcie. Aby zasymulować tę głębie trzeba stworzyć mapę głębokości dla danego obrazu, z której korzysta się aby wyostrzyć, to co jest blisko pożądanej odległości, a rozmyć resztę.

3. Założenia wstępne przyjęte w realizacji projektu

Program powinien się cechować następującymi funkcjami:

- wczytanie obrazu oraz jego mapy głębokości i możliwość zapisu obrazu, który wygenerowaliśmy
- możliwość zasymulowania głębi ostrości dla podanej odległości
- możliwość zmiany skali rozmycia
- ciągła możliwość podglądania obrazu, który nam się generuje

4. Analiza projektu

4.1 *Specyfikacja danych wejściowych*

Aby aplikacja zaczęła pracować potrzebujemy:

- Obrazu, na którym chcemy dokonać odpowiednich przekształceń, obraz ten ma mieć format grafiki bitmapowej – .bmp lub .jpg.
- Mapy głębokości dla podanego wcześniej obrazu formatu .bmp lub .jpg, mapa ta musi być już stworzona, np. przy użyciu odczytu buforu Z z obrazu lub przy użyciu programów do obróbki fotografii np. GIMP.
- Odległości ostrej, podawanej przez użytkownika, poprzez wpisanie w odpowiednie pole. Oczekujemy tutaj wartości pomiędzy 0 (bardzo daleko), a 255 (bardzo blisko).
- Skali rozmycia podawanej przez użytkownika za pomocą suwaka ostrości, możliwe są tutaj wartości od 1 (słabe rozmycie) do 5 (bardzo mocne rozmycie)

4.2 Oczekiwane dane wyjściowe

Podczas programu mamy możliwość zapisania obrazu z wszystkimi przekształceniami, które na nim dokonywaliśmy.

Obraz ten zapisywany jest w formacie .jpg.

4.3 Zdefiniowane struktury danych

Z racji tego, iż w programie używaliśmy biblioteki wxWidgets, klasy zdefiniowane są tak, aby mogły spełniać założenia definicji klas przy użyciu tej biblioteki.

Pierwszą strukturą jest klasa MyApp odpowiadająca za widok głównego okna. Odpowiada ona także za pracę z plikami .bmp i .jpg.
Drugą strukturą jest klasa MyFrame1, która z kolei odpowiada za cały interfejs oraz za funkcje, które wykonują się przy różnych operacjach. Posiada ona wszystkie zmienne, które nie są lokalne, a które są użyte w programie.

4.4 Specyfikacja interfejsu użytkownika

Interfejs jest zbudowana jako aplikacja jednookienkowa, na której tworzymy wszystko co chcemy.
Aplikacja posiada:

- Menu główne – okno domyślnie rozmiarów 900x600, które dzieli się na panel z wszystkimi ustawieniami oraz okno gdzie widzimy zmieniany obraz. Rozmiar głównego okna można zmieniać.

- Panel główny, który posiada obiekty takie jak:
  - przycisk wczytywania obrazu,
  - przycisk wczytania mapy,
  - przycisk zapisu nowopowstałego obrazu,
  - przycisk powrotu do oryginalnego zdjęcia,
  - pole wpisywania odległości ostrej,
  - suwak służący do zmiany skali rozmycia.

Oczywiście przy wyborze obrazu wczytywanego do programu mamy możliwość przeszukiwać katalogi na komputerze.

4.5 Wyodrębnienie i zdefiniowanie zadań

Cały projekt można podzielić na moduły:

- zrozumienie głównego zagadnienia, zapoznanie się z teoretycznymi zagadnieniami użytych w projekcie,
- stworzenie interfejsu komunikującego się z użytkownikiem, stworzenie na nim odpowiednich pól,
- stworzenie algorytmu rozmazywania odpowiednich obiektów na obrazie przy użyciu rozmycia gaussowskiego,
- dodanie do algorytmu rozmycia możliwości zmiany skali rozmycia,
- prezentacja przekształcanego obrazu na ekranie i odpowiednie generowanie danych wyjściowych (obrazu .jpg).

4.6 Decyzje o wyborze narzędzi programistycznych

Przy tworzeniu naszej aplikacji wykorzystano bibliotekę wxWidgets, ponieważ była omawiana szczegółowo na zajęciach z Podstaw Grafiki Komputerowej, a więc mamy doświadczenie przy tworzeniu aplikacji korzystających z tej biblioteki. Do tego ma bardzo dobrą dokumentację i większość problemów byliśmy w stanie rozwiązać sami.
Język, którego używaliśmy to C++ zgodnie z zaleceniami prowadzącego. Sam program tworzyliśmy w Microsoft Visual Studio 2017, ponieważ jest znakomitym środowiskiem programistycznym, które pomaga programiście w wiele możliwych sposobach.

5. Podział pracy i analiza czasowa

Część projektu	Czas wykonania	Kto wykonał
Stworzenie głównego Menu oraz panelu z obiektami	1 tydzień	Paweł Kępka
Część programu odpowiedzialna za odczytywanie obrazów	1 tydzień	Paweł Kępka
Stworzenie kilka obrazów i ich map głębokości	1 tydzień	Mikołaj Domański
Stworzenie algorytmu pozwalającego na rozmycie gaussowskie	1 tydzień	Mikołaj Domański
Stworzenie algorytmu dopasowywującego skalę rozmycia	1 tydzień	Mikołaj Domański
Usprawnienie algorytmów, tak aby generowanie obrazu wykonywało się szybciej	1 tydzień	Paweł Kępka
Stworzenie dokumentacji	2 dni	Paweł Kępka
Testy końcowe	3 dni	Mikołaj Domański

Łącznie projekt zajął nam około jeden miesiąc.

6. Opracowanie i opis niezbędnych algorytmów

Najważniejszym punktem przy pisaniu aplikacji było napisanie algorytmu rozmywania gaussowskiego w zależności od tego jak obiekt, który chcemy rozmyć jest daleko od obiektu o żądanej odległości.

Na początku algorytmu ustalamy skalę rozmycia *kernelRadius*.
Głównym punktem algorytmu jest stworzenie macierzy *kernel* o rozmiarze *kernelRadius x kernelRadius*, czyli macierzy pikseli „sąsiadujących" z pikselem, z którym aktualnie pracujemy. Im większe *kernelRadius* czyli skala rozmycia, tym większa jest macierz kernel.
Macierz ta tworzy się mając podaną zmienną *sigma*, która reprezentuje odległość od piksela od odległości ostrej.

```
Sigma = fabs(value - (Img_Map.GetRed(x, y)));          (1)
```

Jest to wartość bezwzględna z różnicy wartości odległości ostrej i wartości składowej R koloru piksela przy którym jesteśmy. Z racji tego, że jest to mapa szarości to składowe R G i B są równe.
Składowe macierzy kernel są liczone w następujący sposób

$$k[x][y]=\frac{1}{2*\pi*\sigma^2}e^{\frac{-(x-\text{średnia})^2+(y-\text{średnia})^2}{2*\sigma^2}} \qquad (2)$$

Następnie każda składowa przedzielona jest przez sumę wszystkich elementów macierzy.
W pętli przechodzącej przez wszystkie elementy macierzy kernel dla danego piksela zliczamy wszystkie wartości *kernel* sąsiadujących pikseli pomnożonych przez wartość koloru tego piksela.
Na końcu ustawiamy kolor piksela na policzony wcześniej i przechodzimy do kolejnego.

7. Kodowanie

Program koncentruje się na klasie MyFrame1, która nie tylko reprezentuje interfejs graficzny, ale także i funkcje rozmycia gaussowskiego.

Klasa ta posiada metody pozwalające na odczyt wszystkich danych wejściowych, zapisanie obrazu i przede wszystkim na odczyt odległości ostrej wpisywanej przez użytkownika oraz skali rozmycia.

Najważniejszą jednak metodą w tej klasie jest
```
Matrix void MyFrame1::applyFilter(int value, int radius);
```
Value to wartość odległości ostrej a radius to skala rozmycia.
Funkcja ta korzysta z drugiej funkcji:
```
getGaussian(double sigma,int kernelRadius);
```
Zwraca nam ona macierz kernelowską dla danego piksela o danej sigmie.
W klasie jest zdefiniowana także tablica_sigm[10].
Sigma jest liczona z wzoru (1) jednak jest ona normalizowana do jednej z 10 wartości (od 1 do 10). Wprowadza to pewną niedokładność w wyostrzaniu.
Operacje na obrazie to tak naprawdę operacje na kopii obrazu, który jest ciągle wyświetlany. Oryginalny obraz jest ciągle przechowany, aby móc się do niego odwoływać i móc do niego wrócić.

8. Testowanie

Testowanie programu polegało na wczytaniu obrazu i jego mapy i symulowanie głębi ostrości dla wielu wartości odległości ostrej, przy różnych wartościach skali rozmycia.
Otrzymane obrazy z symulowaną głębią ostrością są bardzo zadowalające. Z racji tego, iż trudno uzyskać jest dobrą mapę szarości i nie jesteśmy biegli w ich tworzeniu program testowaliśmy dla 5 obrazów z 5 pięcioma mapami.

Zmienianie skali rozmycia tylko na chwilę zawiesza zdjęcie, ale w konsekwencji wykonuje się bardzo dobrze. Dla pierwszej skali obraz jest dosyć ostry nawet w miejscach, gdzie odległość jest duża. Dla najwyższej skali dla dużej różnicy odległości rozmycie jest bardzo mocny.

9. Wnioski, raport wnioski

Testy dla każdego obrazu wypadły dobrze, nie zauważyliśmy żadnych błędów ani bugów.
Program mógłby być rozszerzony o odczytywanie i możliwość zapisania obrazu do więcej formatów pliku. Można by dołożyć także kolejne możliwości formatowania obrazu, np. zmienianie kontrastu lub jasności.
Temat symulacji głębi ostrości jest jednak bardziej obszerny i można by tutaj mnożyć usprawnienia naszego programu.