
		Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja	
2015/2016	SSI	Języki Asemblerowe	4	1	
Imię:	Mateusz	Prowadzący: OA/JP/KT/GD/BSz/GB	JP		
Nazwisko:	Szostok				
<h2>Raport końcowy</h2>					
Temat projektu: <div style="text-align: center; margin-top: 20px;"> <h1>Ważone dodawanie dwóch plików graficznych w formacie .bmp</h1> </div>					
Data oddania: dd/mm/rrrr			22/03/2016		

Spis treści

1.	Założenia projektu.....	3
1.1.	Część główna programu.....	3
1.2.	Funkcje biblioteki	3
2.	Analiza zadania	4
3.	Schemat blokowy programu.....	6
4.	Opis programu.....	7
	<i>Zmienne prywatne</i>	8
5.	Funkcje biblioteki	9
5.1.	C#	9
5.2.	ASM.....	9
6.	Uruchamianie i testowanie programu	10
7.	Wyniki pomiarów oraz wykresy porównawcze	11
8.	Instrukcja obsługi programu	16
9.	Wnioski końcowe	17

1. Założenia projektu

Program powinien umożliwić ważone nakładanie obrazów w formacie *.bmp*, a tym samym podglądu wyniku działania algorytmu.

1.1. Część główna programu

Stworzenie aplikacji w języku C# umożliwiającej

1. załadowanie dwóch plików grafiki bitmapowej,
2. ustawienie określonej liczby wątków,
3. wyboru biblioteki dostarczającej funkcję do obróbki obrazu,
4. podział obrazu na odpowiednią liczbę części zależnej od zadanej liczby wątków oraz wywołanie dla każdej z nich funkcji łączenia.

1.2. Funkcje biblioteki

1. Napisanie biblioteki w języku C# udostępniająca funkcję umożliwiającą nakładanie obrazów przy wykorzystaniu algorytmu ważonego dodawania. Wynik działania będzie zapisany w bitmapie podanej jako *image1*.
2. Napisanie biblioteki w języku asemblera udostępniającą i realizującą tą samą funkcjonalność co w/w biblioteka.

Prototyp funkcji

```
void blendTwoImages(int** bitmaps, int* coords, int alpha);
```

2. Analiza zadania

Ważone nakładanie obrazów polega na dodaniu pikseli znajdujących się na tych samych współrzędnych mnożąc je wcześniej przez współczynnik określający „siłę” nałożenia danego obrazu. Poniżej znajduje się wzór, który został zaimplementowany

$$Image(x, y) = W * Image1(x, y) + (1 - W) * Image2(x, y) \text{ gdzie } 0 \leq W \leq 1$$

$W = 0,3$



$W = 0,7$



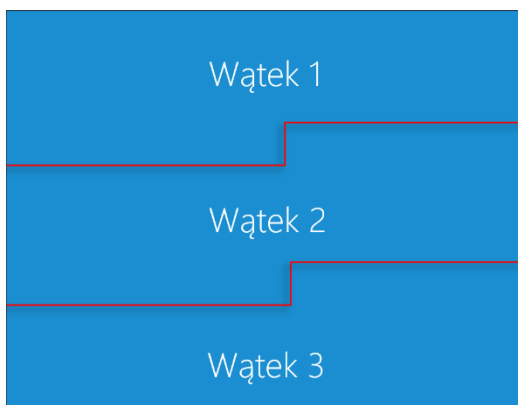
Efekt przejścia pomiędzy dwoma obrazami możemy uzyskać zmieniając współczynnik W od 0 do 1.

W programie aby ułatwić wprowadzanie wartości została przyjęta skala od 0 do 255. Aby zobrazować działanie programu potrzebne jest podanie min. dwóch obrazów, które powinien wskazać użytkownik. Tutaj ważny jest format podawanych grafik, ponieważ każdy z nich posiada swój własny standard dotyczący kodowania danych. Nie chcąc nazbyt komplikować zadania, postanowiłem aby obsługiwanym formatem w wersji 1.0 były pliki z 24-bitową grafiką bitmapową bez kompresji RLE.

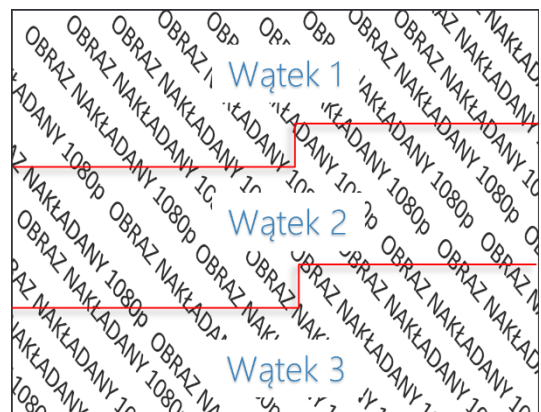
Program powinien być również wielowątkowy. Tutaj można było zrealizować to na dwa sposoby, pierwszy to umożliwienie wczytania tylu par obrazów ile użytkownik wybrał wątków, drugi natomiast bazuje na tym, że zastosowany algorytm można łatwo zrównoleglić, to właśnie ten sposób wybrałem, ponieważ jest to o wiele bardziej efektywne wykorzystanie wątków, a ponadto możliwym będzie przyspieszenie samej pracy programu.

Obrazy dzielone są na tyle części ile użytkownik wybierze wątków

Liczba wątków: 3



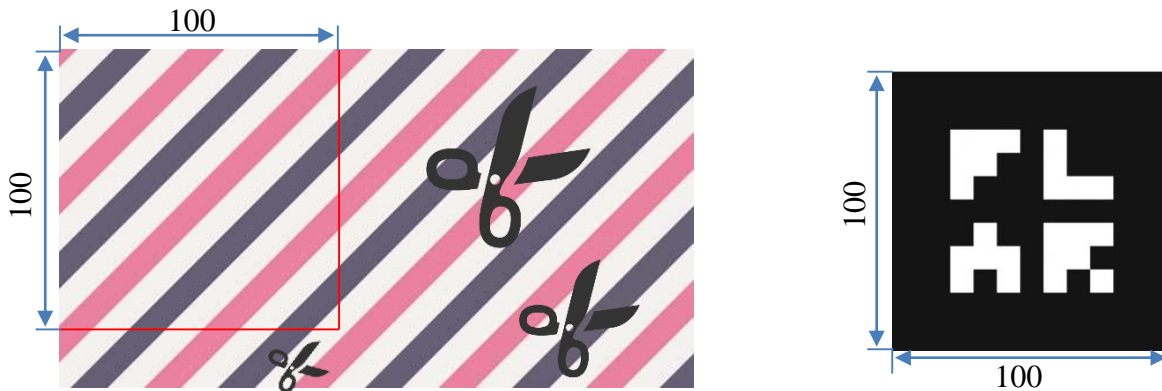
Obraz bazowy



Obraz nakładany

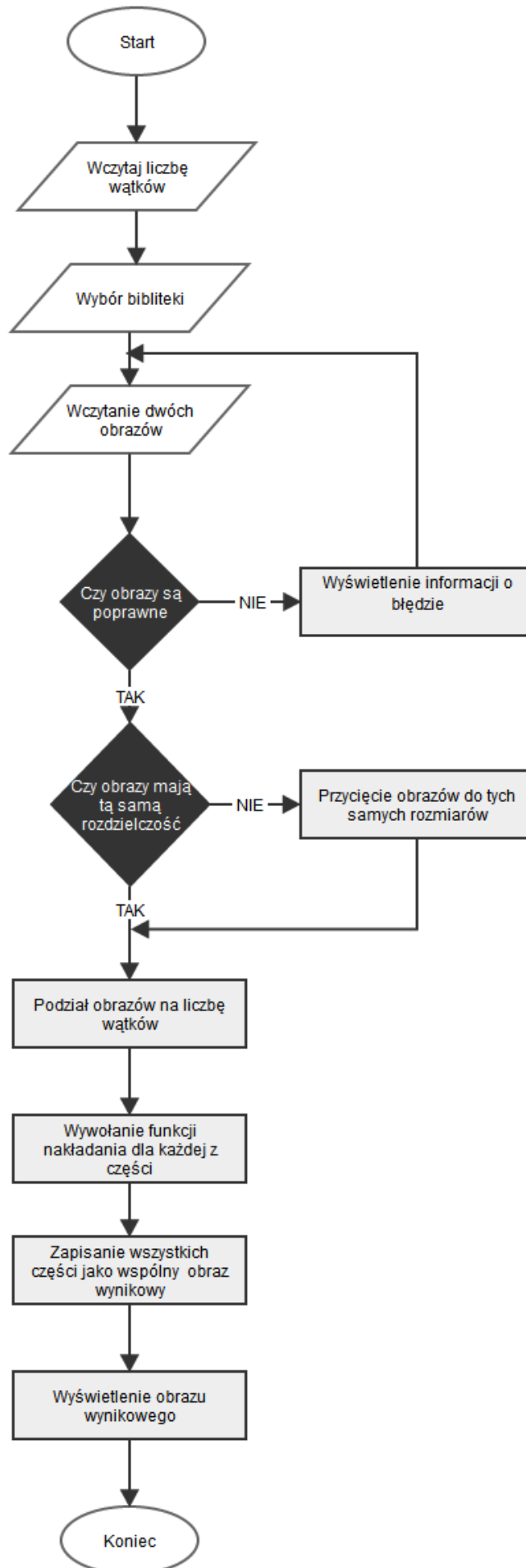
Wystąpić tutaj mogą dwa przypadki. Pierwszy najbardziej optymistyczny, gdzie następuje równy podział i można stworzyć n wątków o równym kroku tj. każdy ma taki sam rozmiar, gdzie n to liczba wątków zadana przez użytkownika. Problem natomiast pojawia się kiedy nie można dokonać równego podziału. W takim przypadku postanowiłem, że stworzonych zostanie $n-1$ wątków z regularnym krokiem, a ostatni z nich będzie dopełnieniem do pełnej liczby pikseli składających się na obraz.

Należało również rozwiązać problem, kiedy użytkownik poda obrazy o różnych rozdzielczościach. W moim odczuciu najbardziej logicznym podejściem jest przycięcie obrazu większego w taki sposób aby oba posiadały te same wymiary. Poniżej przedstawiono która część grafiki zostanie odrzucona.



Środowisko programistyczne, które zostało wybrane na zajęciach to **Visual Studio 2013**, ze względu na to postanowiłem, że program zostanie stworzony przy wykorzystaniu takich technologii jak C# oraz WPF. Język C# posiada przewagę nad C/C++ tym, że samo tworzenie kodu jest na wyższym poziomie abstrakcji (np. nie musimy się przejmować alokacją oraz zwalnianiem pamięci przydzielanej dynamicznie), a ponadto posiadamy więcej mechanizmów do tworzenia oraz nadzorowania pracy wątków. Z kolei silnik graficzny WPF jest nowszy od WinForms oraz pozwala na tworzenie skalowalnych, a przy tym wieloplatformowych aplikacji.

3. Schemat blokowy programu



4. Opis programu

Projekt w języku wysokiego poziomu został zrealizowany przy wykorzystaniu wzorca projektowego MVVM, który wymusza odpowiednią strukturę katalogów jak i dalsze użyte wzorce strukturalne. Poniżej zostanie opisana tylko główna klasa programu, reszta została opisana w kodzie źródłowym.

Klasa BlendImagesSystem

Metody publiczne

```
public unsafe void BlendImages()
```

- Metoda typu Coarse-grained, realizująca ważone nakładanie obrazów.

Metody prywatne

```
private void SaveDialog(WriteableBitmap resultBitmap)
```

- Wyświetlenie dialogu do zapisu bitmpay będącej wynikiem nałożenia przekazanych obrazów.

```
private void LoadImagesFromUserPaths()
```

- Załadowanie obrazu podanych przez użytkownika.

```
private void SetMaxArraySize()
```

- Ustalenia maksymalnej wielkości tablicy poprzez sprawdzenie znalezienie bitmapy onamniejszej szerokości oraz wysokości.

```
private void CropAllImagesToMaxArraySize()
```

- Przycięcie wszystkich bitmap do tej samej wielkości

```
private int* convertCoords(int[] coords)
```

- Konwersja tablicy współrzędnych opisujących podział obrazu na części.

```
private unsafe void createNewThread(int[] coords)
```

- Utworzenie nowego wątku nakładającego obrazy.
 - start - definiuje indeks początkowy od którego wątek będzie wykonywał obliczenia
 - stop - definiuje ineks końcowy na którym wątek zatrzyma obliczenia

```
private void createThreadList(int threadNumber)
```

- Utorzenie listy wątków które odpowiedzialne są za nałożenie obrazów na siebie.
 - threadNumber - ilość wątków jaka ma zostać utworzona

```
private unsafe int** createIntArrayFromByte()
```

- Metoda wykorzystywana do zainicjalizowania zmiennej `bitmapList` która wykorzystywana jest przez algorytm asemblera. Zwraca tablicę wskaźników na wskaźniki w stylu C/C++

```
private unsafe void copyAsmResultToImg1PixelsByte()
```

- Przekopiowanie obliczeń wykonanych przez bibliotekę asemblerową, do tablicy bajtów pixeli obrazu bazowego

Zmienne prywatne

Wartość	Nazwa	Opis
<code>AppSettings</code>	<code>appSettings</code>	Referencja do obiektu przechowującego ustawienia użytkownika
<code>int</code>	<code>maxWidth</code>	Maksymalna szerokość pliku graficznego
<code>int</code>	<code>maxHeight</code>	Maksymalna wysokość pliku graficznego
<code>int</code>	<code>threadPixelsStep</code>	Ilość pikseli przypadająca na jeden wątek
<code>byte[]</code>	<code>img1PixelsByte</code>	Tablica pikseli w postaci bajtów dla obrazu bazowego
<code>byte[]</code>	<code>img2PixelsByte</code>	Tablica pikseli w postaci bajtów dla obrazu nakładanego
<code>int[]</code>	<code>img1PixelsInt</code>	Tablica pikseli w postaci liczb całkowitych dla obrazu bazowego
<code>int[]</code>	<code>img2PixelsInt</code>	Tablica pikseli w postaci liczb całkowitych dla obrazu nakładanego
<code>int**</code>	<code>intBitmapsList</code>	Wskaźnik na listę tablic pikseli w postaci bajtów (dla asm)
<code>byte[][]</code>	<code>byteBitmapsList</code>	Wskaźnik na listę tablic pikseli w postaci bajtów (dla C#)
<code>List<Thread></code>	<code>threadList</code>	Lista utworzonych wątków podczas wykonywania obliczeń
<code>List<BitmapImage></code>	<code>bmpList</code>	Lista wczytanych plików graficznych podanych przez użytkownika
<code>List<BitmapImage></code>	<code>croppedBmpList</code>	Lista plików graficznych o tych samych rozmiarach (przycięte obrazy wejściowe do tych samych rozmiarów)
<code>List<int></code>	<code>threadResult</code>	Lista przechowująca wartości zwracane przez wyjątki (informację o powodzeniu/błędzie wykonania)

5. Funkcje biblioteki

5.1. C#

Implementacja algorytmu w języku wysokiego poziomu

```
public static void blendTwoImages(byte[][]bitmaps, int[]coords, int alpha)
```

- Metoda realizująca ważone nakładanie dwóch obrazów.

Parametry:

- imgBottom - referencja do obrazu na, który zostanie nałożony obraz przekazany jako imgTop,
- imgTop - obraz nakładany,
- alpha - waga obrazu nakładanego (od 0 do 255)
- start - definiuje indeks początkowy od którego wątek będzie wykonywał obliczenia,
- stop - definiuje indeks końcowy na którym wątek zatrzyma obliczenia

5.2. ASM

Implementacja algorytmu w języku asemblera

```
blendTwoImages PROC USES ebx edx ecx,  
    bitmaps: PTR PTR DWORD,  
    coords: PTR DWORD,  
    alpha: DWORD
```

- Procedura dokonująca ważonego nakładania obrazów. Wynik nałożenia obrazów będzie umieszczony w tablicy przekazanej jako obraz na który należy nałożyć drugi podany obraz.

Wejście:

- bitmaps - dwuwymiarowa tablica typu int** gdzie,
 - bitmapList[0] - obraz bazowy
 - bitmapList[1] - obraz nakładany
- coords- dwuwymiarowa tablica typu int* gdzie,
 - coords[0] - indeks początkowy od którego wykonywane będą obliczenia na tablicy bitmap.
- coords[1] - indeks na którym należy zakończyć obliczenia na tablicy bitmap.

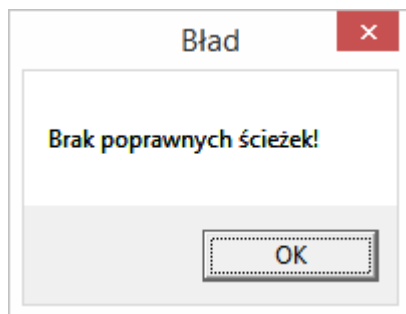
- alpha - waga nakładanego obrazu (przezroczystość) w zakresie od 0 do 255.

Wyjście:

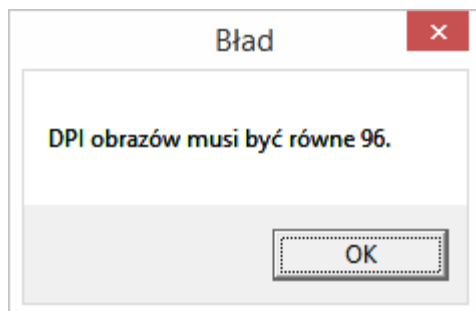
- 2 - zostały wykryte przekłamania w procedurze
- 1 - rozkaz cpuid nie jest wspierany
- 0 - instrukcje SSE2 są niedostępne
- 1 - procedura nałożenia obrazów przebiegła pomyślnie

6. Uruchamianie i testowanie programu

Program został przetestowany specjalnie przeze mnie przygotowanymi plikami o kilku najpopularniejszych rozdzielczościach ekranów – 1280x720, 1920x1080 oraz 3840x2160 oraz DPI równym 96. Został on również zabezpieczony przed błędami użytkownika. Poniżej znajdują się możliwe błędy jakie mogą wystąpić wraz z opisem jak należy postąpić w przypadku ich pojawienia.

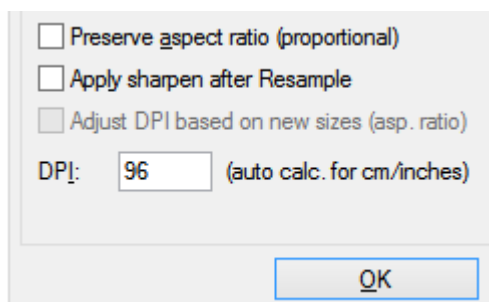


Jeśli wystąpi ten błąd, należy sprawdzić poprawność podanych ścieżek, czy plik znajduje się w podanej lokalizacji oraz czy możliwe jest jego odczytanie.



Problemem tutaj jest zła wartość DIP obrazu. Niestety obsługiwane są tylko te o DPI równym 96. Aby rozwiązać ten problem należy zmienić tą wartość w naszym obrazie. Możemy posłużyć się darmowym programem **IrfanView**, gdzie po otwarciu naszej grafiki należy otworzyć okno *Resize/Resample image (Image-> Resize/Resample)* gdzie można podać odpowiednią wartość.

Zrzut ekranu z programu IrfanView, przedstawiający miejsce wpisania nowej wartości DPI dla naszego obrazu



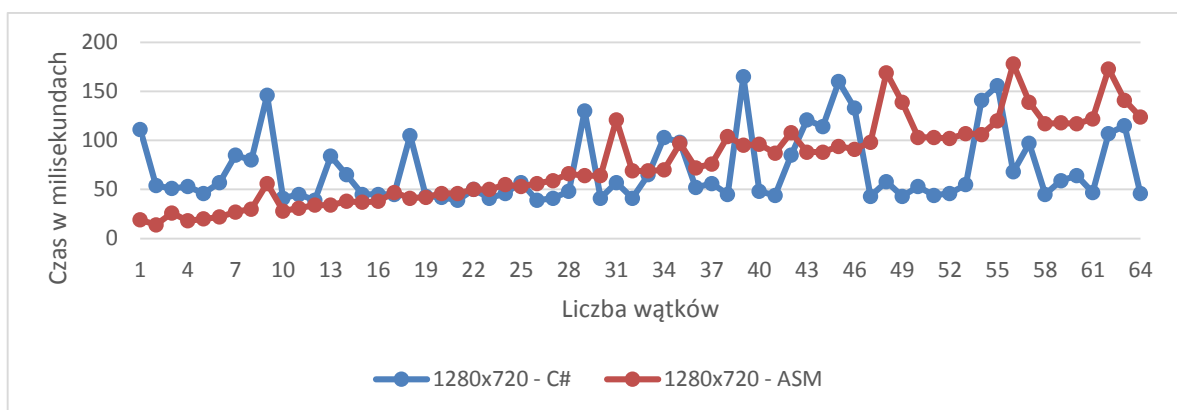
Istnieje również walidacja pól pobierających od użytkownika dane. Jeśli wpisujemy nieodpowiednią wartość to pole zostanie podświetlone na czerwono a przycisk „Połącz” pozostanie nieaktywny do czasu wprowadzenia poprawnych wartości.

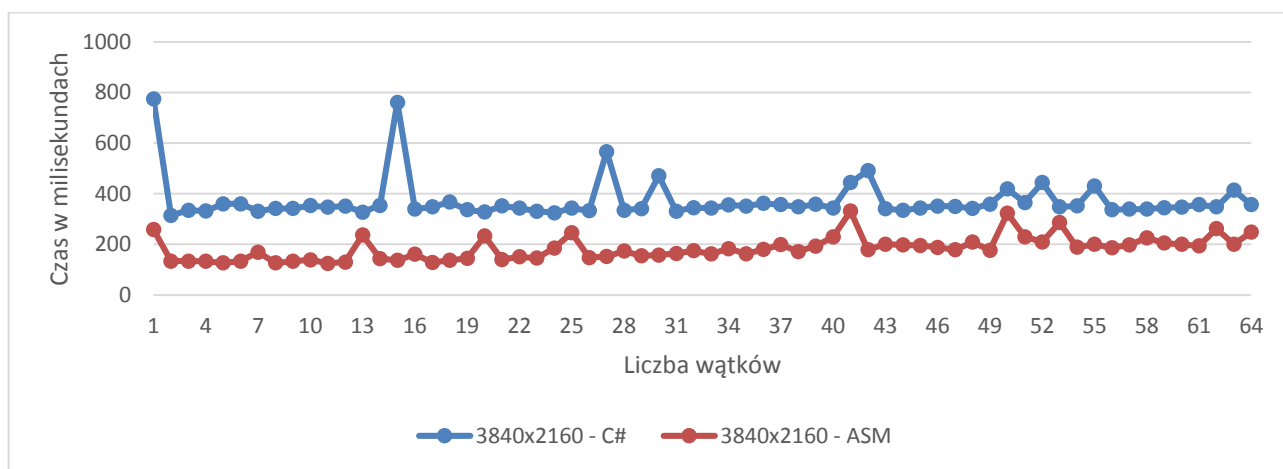
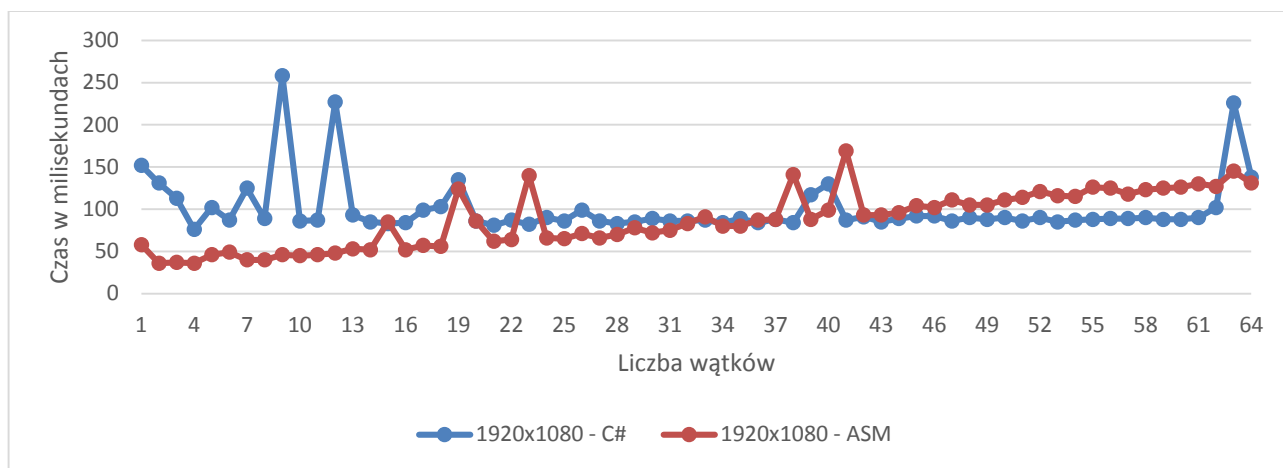
The screenshot shows a software window titled 'Kanał alpha'. At the top, there is a range indicator '0 < 500 < 255'. The number '500' is highlighted with a red border, indicating an invalid input. Below this, there is a section titled 'Wybierz obrazy' containing two input fields: 'Obraz bazowy' and 'Obraz nakładany'. Both fields have red borders, suggesting they are also part of the validation process. A 'Połącz' button is located at the bottom right of the interface.

7. Wyniki pomiarów oraz wykresy porównawcze

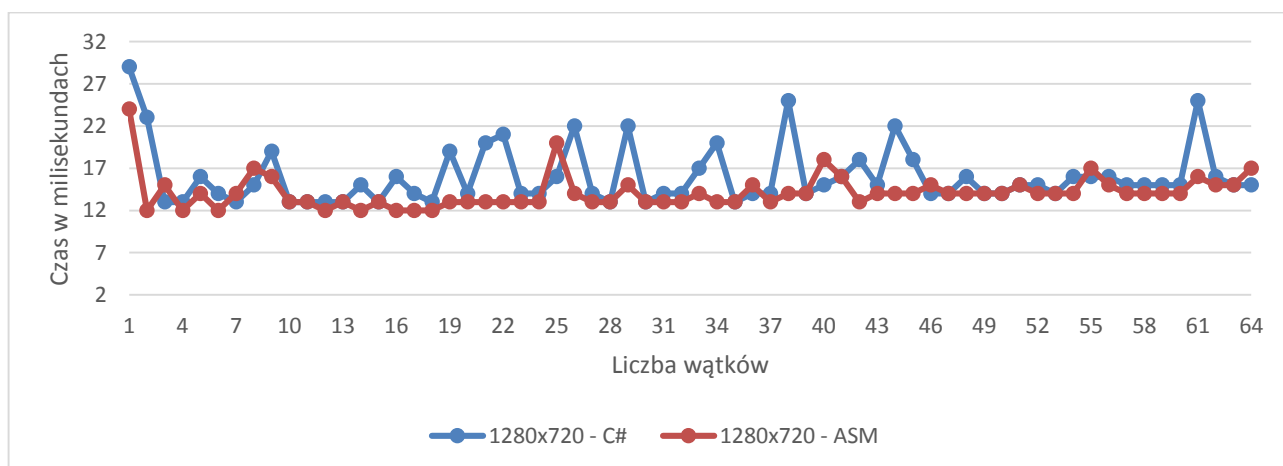
W celu wykonania pomiarów, napisałem metodę, która w pętli uruchamia obliczenia dla obu bibliotek DLL, dla wszystkich ilości wątków od 1 do 64. Wystarczy tylko podać obrazy na których wykonane zostaną obliczenia. Zabieg ten powtórzyłem do przygotowanych przeze mnie plików graficznych opisanych w rozdziale [Uruchamianie i testowanie programu](#). Poniżej znajdują się wyniki działania algorytmu przeprowadzone na dwóch różnych platformach sprzętowych.

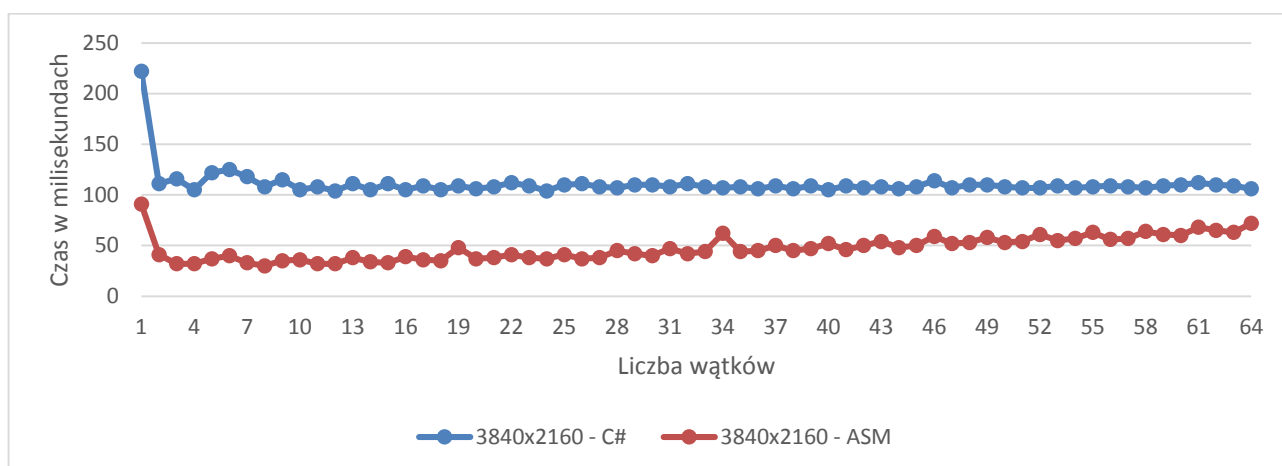
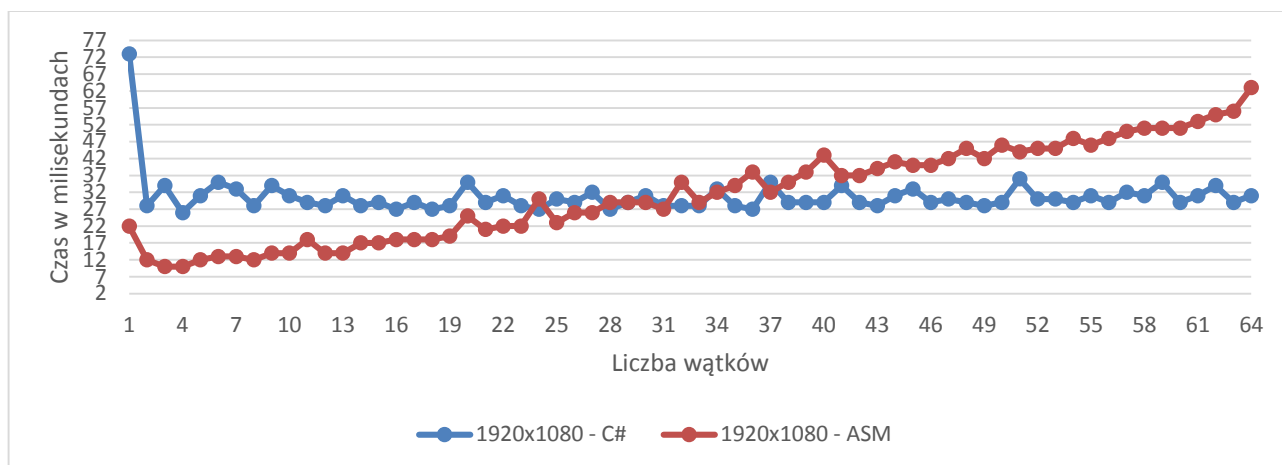
- Intel® Core™2 Duo Processor T5500
(2M Cache, 1.66 GHz, 667 MHz FSB)





- Intel® Core™ i5-5200U Processor
(3M Cache, up to 2.70 GHz)





Analizując przedstawione wykresy pierwsze co można zauważyć to to, że algorytm napisany w asemblerze wykonuje się szybciej na obu platformach sprzętowych. Powód dlaczego tak się dzieje, zostanie opisany nieco dalej. Godne uwagi jest również to, że zgodnie z założeniami najlepsze czasy uzyskujemy ustawiając liczbę wątków na ich rekomendowaną wartość tj. tyle ile posiadamy rdzeni w danym procesorze, kolejno dla procesor *T5500* najlepszy czas przypada na liczbę wątków równą 2, a dla procesora *i5-5200U* gdzie liczba dostępnych wątków wynosi 4, również osiągnięty został najlepszy wynik. Zwiększanie liczby wątków tym samym dzieląc obrazy na więcej części, niestety powoduje pogorszeniu czasów obliczeń, najprawdopodobniej ze względu na mechanizm kolejkowania wątków, oraz ciągłego przełączania kontekstów dla danych wątków, które w założeniach powinny uzyskiwać dostęp do procesora w równych odstępach czasu.

Powód uzyskiwania przez algorytm napisany w asemblerze czasów lepszy od algorytmu zaimplementowanego w języku wyższego poziomu związany jest najprawdopodobniej z dodatkowym zrównolegleniem już samego sposobu dodawania wartości pikseli do siebie. W C# dodawanie wartości RGB wykonuje się sekwencyjnie.

```
for (int j = coords[0] * 4; j < coords[1] * 4; )
{
    bitmaps[0][j] = (byte)((alphaBottom * (float)bitmaps[0][j]) +
        (alphaTop * (float)bitmaps[1][j]));
    ++j;
    bitmaps[0][j] = (byte)((alphaBottom * (float)bitmaps[0][j]) +
        (alphaTop * (float)bitmaps[1][j]));
    ++j;
    bitmaps[0][j] = (byte)((alphaBottom * (float)bitmaps[0][j]) +
        (alphaTop * (float)bitmaps[1][j]));
    ++j;
    bitmaps[0][j] = 255;
    ++j;
}
```

Z kolei w assemblerze, poprzez wykorzystanie rozkazów strumieniowych, możliwym było wykonanie dodawania oraz mnożenia tych wartości w tym samym czasie. Poniżej znajduje się „wizualizacja” tego procesu (niestety niemożliwym było dodanie animacji w wersji .pdf, a więc zostały przedstawione kolejne klatki z pokazu).

```

198 ;===== WAŻONE NAKŁADANIE OBRAZÓW =====
199 mov ecx, start ; zainicjalizowanie licznika pętli
200 blendLoop:
201 MOVDQU xmm1, [ebx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
202 MOVDQU xmm2, [edx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
203
204 CVTDQ2PS xmm1, xmm1 ; konwersja z double word na single-precision float
205 CVTDQ2PS xmm2, xmm2 ; konwersja z double word na single-precision float
206
207 MULPS xmm1, xmm6 ; mnożenie równoległe alphaBottom*bitmapBottom(x,y)
208 MULPS xmm2, xmm5 ; mnożenie równoległe alphaTop*bitmapTop(x,y)
209
210 ADDPS xmm1, xmm2 ; dodanie wartości rgb o odpowiednich wagach
211
212 CVTTPS2DQ xmm1, xmm1 ; konwersja z single-precision na double words z obcięciem
213 CVTTPS2DQ xmm2, xmm2 ; konwersja z single-precision na double words z obcięciem
214
215 MOVDQU [rgba], xmm1 ; pobranie wartości do struktury w celu wyłuskania parametru 'a'
216 mov [rgba.Pixel].a, 255 ; przywrócenie domyślnej wartości parametru 'a'
217 MOVDQU xmm1, [rgba] ; załadowanie poprawnych wartości do rejestru
218
219 MOVDQU [ebx + ecx*SIZEOF DWORD], xmm1 ; nadpisanie wartości w komórkach bitmapy bazowej
220
221 add ecx, bytesPerPixel ; zwiększenie licznika o jeden piksel który został już obliczony
222 cmp ecx, stop ; sprawdzenie czy należy już skończyć
223 jne blendLoop ; jeśli ecx != stop to wróć na początek pętli, w przeciwnym wypadku
224 ; zakończ obliczenia

```

XMM1

B	G	R	ALPHA
---	---	---	-------

XMM2

B	G	R	ALPHA
---	---	---	-------



```

198 ;===== WAŻONE NAKŁADANIE OBRAZÓW =====
199 mov ecx, start ; zainicjalizowanie licznika pętli
200 blendLoop:
201 MOVDQU xmm1, [ebx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
202 MOVDQU xmm2, [edx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
203
204 CVTDQ2PS xmm1, xmm1 ; konwersja z double word na single-precision float
205 CVTDQ2PS xmm2, xmm2 ; konwersja z double word na single-precision float
206
207 MULPS xmm1, xmm6 ; mnożenie równoległe alphaBottom*bitmapBottom(x,y)
208 MULPS xmm2, xmm5 ; mnożenie równoległe alphaTop*bitmapTop(x,y)
209
210 ADDPS xmm1, xmm2 ; dodanie wartości rgb o odpowiednich wagach
211
212 CVTTPS2DQ xmm1, xmm1 ; konwersja z single-precision na double words z obcięciem
213 CVTTPS2DQ xmm2, xmm2 ; konwersja z single-precision na double words z obcięciem
214
215 MOVDQU [rgba], xmm1 ; pobranie wartości do struktury w celu wyłuskania parametru 'a'
216 mov [rgba.Pixel].a, 255 ; przywrócenie domyślnej wartości parametru 'a'
217 MOVDQU xmm1, [rgba] ; załadowanie poprawnych wartości do rejestru
218
219 MOVDQU [ebx + ecx*SIZEOF DWORD], xmm1 ; nadpisanie wartości w komórkach bitmapy bazowej
220
221 add ecx, bytesPerPixel ; zwiększenie licznika o jeden piksel który został już obliczony
222 cmp ecx, stop ; sprawdzenie czy należy już skończyć
223 jne blendLoop ; jeśli ecx != stop to wróć na początek pętli, w przeciwnym wypadku
224 ; zakończ obliczenia

```

XMM1

B * alphaBottom	G * alphaBottom	R * alphaBottom	A * alphaBottom
-----------------	-----------------	-----------------	-----------------

XMM2

B * alphaBottom	G * alphaBottom	R * alphaBottom	A * alphaBottom
-----------------	-----------------	-----------------	-----------------





```
198 ;===== WAŻONE NAKŁADANIE OBRAZÓW =====
199 mov ecx, start ; zainicjalizowanie licznika pętli
200 blendLoop:
201 MOVDQU xmm1, [ebx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
202 MOVDQU xmm2, [edx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
203
204 CVTDP2PS xmm1, xmm1 ; konwersja z double word na single-precision float
205 CVTDP2PS xmm2, xmm2 ; konwersja z double word na single-precision float
206
207 MULPS xmm1, xmm6 ; mnożenie równoległe alphaBottom*bitmapBottom(x,y)
208 MULPS xmm2, xmm5 ; mnożenie równoległe alphaTop*bitmapTop(x,y)
209
210 ADDPS xmm1, xmm2 ; dodanie wartości rgb o odpowiednich wagach
211
212 CVTTPS2DQ xmm1, xmm1 ; konwersja z single-precision na double words z obcięciem
213 CVTTPS2DQ xmm2, xmm2 ; konwersja z single-precision na double words z obcięciem
214
215 MOVDQU [rgba], xmm1 ; pobranie wartości do struktury w celu wyłuskania parametru 'a'
216 mov [rgba.Pixel].a, 255 ; przywrócenie domyślnej wartości parametru 'a'
217 MOVDQU xmm1, [rgba] ; załadowanie poprawnych wartości do rejestru
218
219 MOVDQU [ebx + ecx*SIZEOF DWORD], xmm1 ; nadpisanie wartości w komórkach bitmapy bazowej
220
221 add ecx, bytesPerPixel ; zwiększenie licznika o jeden piksel który został już obliczony
222 cmp ecx, stop ; sprawdzenie czy należy już skończyć
223 jne blendLoop ; jeśli ecx != stop to wróć na początek pętli, w przeciwnym wypadku
224 ; zakończ obliczenia
```

XMM1

B_xmm1 + B_xmm2 | G_xmm1 + G_xmm1 | R_xmm1 + R_xmm2 | A_xmm1 + A_xmm2



```
198 ;===== WAŻONE NAKŁADANIE OBRAZÓW =====
199 mov ecx, start ; zainicjalizowanie licznika pętli
200 blendLoop:
201 MOVDQU xmm1, [ebx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
202 MOVDQU xmm2, [edx + ecx*SIZEOF DWORD] ; załadowanie 4 wartości (Move unaligned double quad words)
203
204 CVTDP2PS xmm1, xmm1 ; konwersja z double word na single-precision float
205 CVTDP2PS xmm2, xmm2 ; konwersja z double word na single-precision float
206
207 MULPS xmm1, xmm6 ; mnożenie równoległe alphaBottom*bitmapBottom(x,y)
208 MULPS xmm2, xmm5 ; mnożenie równoległe alphaTop*bitmapTop(x,y)
209
210 ADDPS xmm1, xmm2 ; dodanie wartości rgb o odpowiednich wagach
211
212 CVTTPS2DQ xmm1, xmm1 ; konwersja z single-precision na double words z obcięciem
213 CVTTPS2DQ xmm2, xmm2 ; konwersja z single-precision na double words z obcięciem
214
215 MOVDQU [rgba], xmm1 ; pobranie wartości do struktury w celu wyłuskania parametru 'a'
216 mov [rgba.Pixel].a, 255 ; przywrócenie domyślnej wartości parametru 'a'
217 MOVDQU xmm1, [rgba] ; załadowanie poprawnych wartości do rejestru
218
219 MOVDQU [ebx + ecx*SIZEOF DWORD], xmm1 ; nadpisanie wartości w komórkach bitmapy bazowej
220
221 add ecx, bytesPerPixel ; zwiększenie licznika o jeden piksel który został już obliczony
222 cmp ecx, stop ; sprawdzenie czy należy już skończyć
223 jne blendLoop ; jeśli ecx != stop to wróć na początek pętli, w przeciwnym wypadku
224 ; zakończ obliczenia
```

XMM1

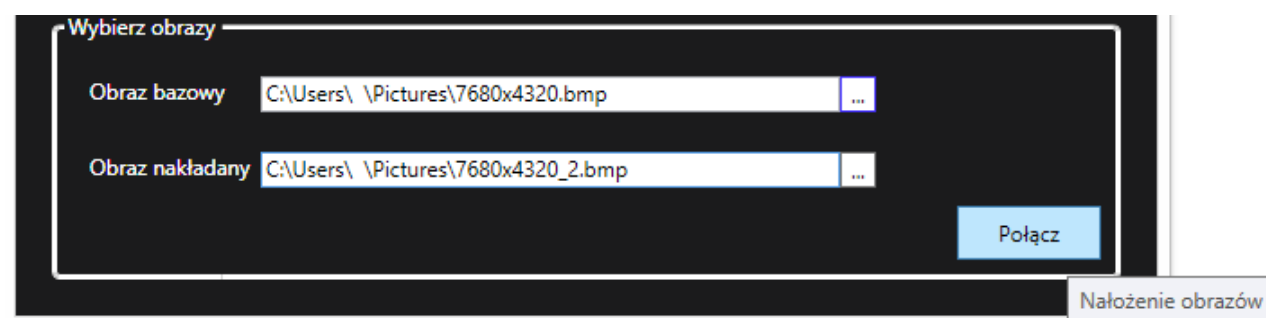
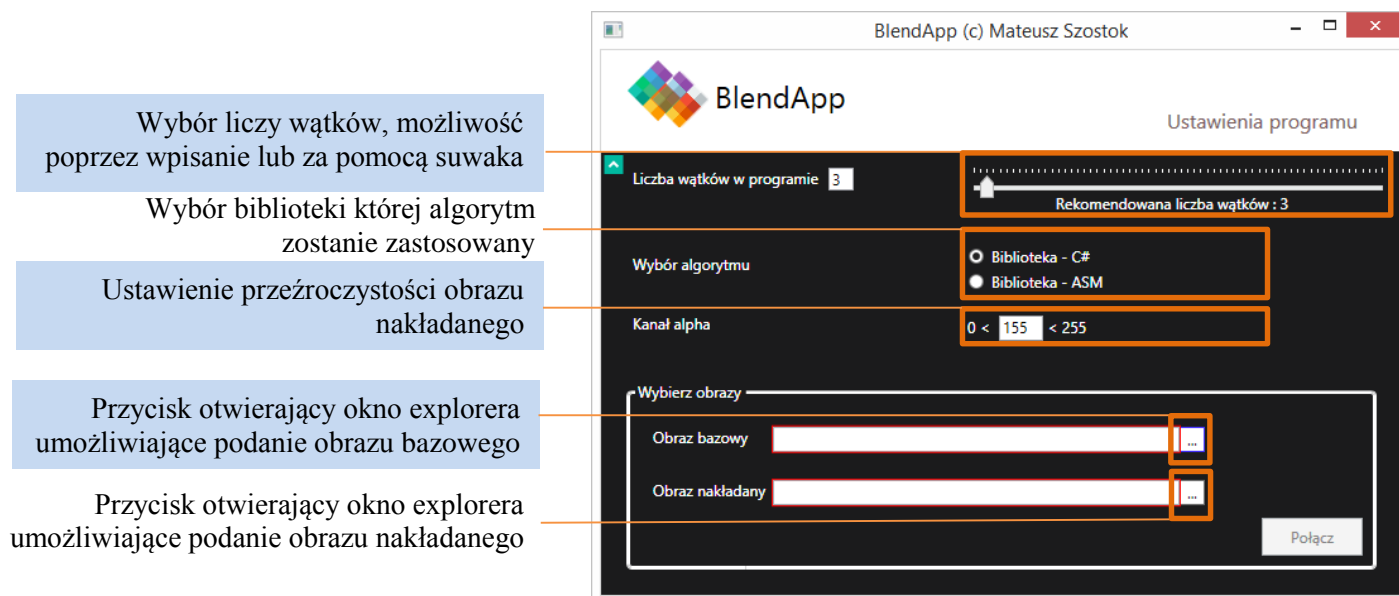
B

G

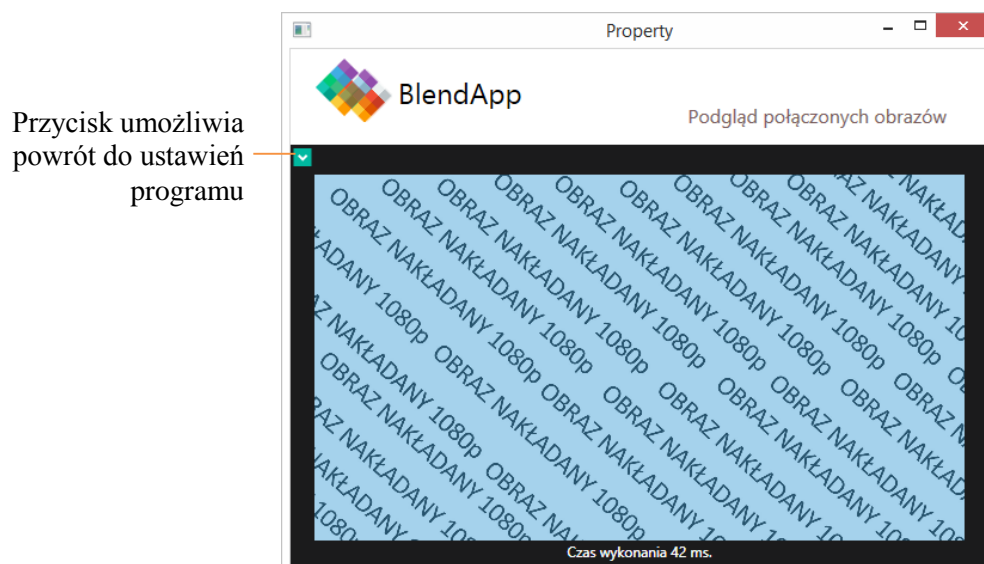
R

255

8. Instrukcja obsługi programu



Przycisk „Połącz” odblokowuje się dopiero kiedy podamy poprawne ścieżki do obrazów. Po naciśnięciu pojawi się okno explorera, w celu podania miejsca zapisania obrazu wynikowego. Następnie pojawi się podgląd zapisanego obrazu wraz z czasem wykonania algorytmu.



9. Wnioski końcowe

Finalnie jestem zadowolony z osiągniętych wyników oraz dokonanych wyborów zarówno dotyczących wykorzystania wielowątkowości jak i implementacji samego algorytmu, a w szczególności tej w języku niskiego poziomu gdzie mogłem wykorzystać już zdobytą wiedzę. Wcześniejsze laboratoria z języków assemblerowych spowodowały, że wiedziałem w jaki sposób poradzić sobie z napotkanymi problemami oraz posiadałem już umiejętność efektywnego debugowania programu.

Wyniki pomiarów, których dokonałem również są zadawalające, a co najważniejsze zgodne z założeniami teoretycznymi. Zauważyłem, że wykonywanie niektórych operacji przy wykorzystaniu języka assemblera mając nawet nowszy procesor, pozwala znacząco (ok 3,6 razy) zmniejszyć czas obliczeń, a więc nie tylko na słabszy ale również i na nowszych platformach warto tworzyć wstawki assemblerowe w kodzie wysokiego poziomu.