Politechnika Wrocławska

Architektura Komputerów 2 - laboratorium 5

sprawozdanie

Autor:
Jarosław Piszczała 209983

Prowadzący:
Prof. Janusz Biernat

Wydział Elektroniki
Informatyka
III rok

31 maja 2016

Spis treści

1 Wstęp teoretyczny

1.1 Format BMP

Format BMP został stworzony do grafiki bitmapowej[1]. W strukturę tego formatu wchodzi nagłówek oraz tabela pikseli. W nagłówku zapisane są wszelkie informacje na temat charakterystyki bitmapy z których najważniejsze to: wysokość, szerokość, oraz liczba bitów na piksel. Jest to istotne, gdyż te wartości wpływają na budowę tabeli pikseli [Rysunek 1]. Istotna jest także jej budowa, gdyż tabela pikseli jest zapisana od dołu do góry, od lewej do prawej. Oznacza to, iż dolny lewy róg jest jej początkiem, a prawy góry róg - końcem. Ważny jest też *padding*, gdyż w strukturze ilość bajtów w każdym wierszu musi być potęgą liczby 4.

Image Data PixelArray [x,y]					
Pixel[0,h-1]	Pixel[1,h-1]	Pixel[2,h-1]	...	Pixel[w-1,h-1]	Padding
Pixel[0,h-2]	Pixel[1,h-2]	Pixel[2,h-2]	...	Pixel[w-1,h-2]	Padding
			⋮		
Pixel[0,9]	Pixel[1,9]	Pixel[2,9]	...	Pixel[w-1,9]	Padding
Pixel[0,8]	Pixel[1,8]	Pixel[2,8]	...	Pixel[w-1,8]	Padding
Pixel[0,7]	Pixel[1,7]	Pixel[2,7]	...	Pixel[w-1,7]	Padding
Pixel[0,6]	Pixel[1,6]	Pixel[2,6]	...	Pixel[w-1,6]	Padding
Pixel[0,5]	Pixel[1,5]	Pixel[2,5]	...	Pixel[w-1,5]	Padding
Pixel[0,4]	Pixel[1,4]	Pixel[2,4]	...	Pixel[w-1,4]	Padding
Pixel[0,3]	Pixel[1,3]	Pixel[2,3]	...	Pixel[w-1,3]	Padding
Pixel[0,2]	Pixel[1,2]	Pixel[2,2]	...	Pixel[w-1,2]	Padding
Pixel[0,1]	Pixel[1,1]	Pixel[2,1]	...	Pixel[w-1,1]	Padding
Pixel[0,0]	Pixel[1,0]	Pixel[2,0]	...	Pixel[w-1,0]	Padding

Rysunek 1: tabela pikseli

1.2 Architektura MMX

Jest ona wykorzystywana głównie przetwarzania dużych ilości danych gdzie wykorzystywany jest jeden algorytm. Najważniejszymi cechami tejże architektury są[2]:

- 8 rejestrów 64 bitowych wykorzystujących rejestry FPU (MM0 - MM7).
- typ danych "packed". Jego działanie polega na traktowaniu rejestru 64 bitowego jako wektora pewnej liczby komórek o tej samej wielkości.

- format instrukcji. Instrukcje dla MMX budowane są w kolejności: **P**, skrót rozkazu/instrukcja (np. **ADD**), litery **S** dla wartości ze znakiem bądź **U** dla wartości bez znaku, **S** jeśli operacja jest wykonywana z nasyceniem, litery **L** lub **H** jeśli operacja wykonywana jest na mniej lub bardziej znaczących bitach oraz **B**, **W**, **D**, **Q** które odpowiadają za rozmiar komórki wektora. Przykładowa instrukcja MMX: **PADDUSB** (równoległe dodawanie, bez znaku z saturacją, bajtów).

2 Zakres prac

Zadaniem było stworzenie programu w C który wczytywał by plik graficzny (bmp, jpg) i wprowadzałby na nim filtry (odbicia poziome, pionowe oraz po przekątnej, saturacja, negatyw). Następnie należało wybrany filtr napisać w ASM, i porównać czas działania funkcji w C i ASM.

3 Rozwiązanie

Do stworzenia funkcji w C można było wykorzystać gotowy schemat ze strony *Zakładu Architektury Komputerów*[3] który potrafił wczytać oraz zapisać pliki graficzne. Przed przystąpieniem do tworzenia filtrów zostały do kodu programu dodane dwie zmienne (Listing 1): *rowSize* odpowiedzialne za długość wiersza w tablicy pikseli oraz *padding* będące wartością paddingu w aktualnie wczytanym pliku.

Listing 1: Dodatkowe zmienne

```
int rowSize =
        (( image->format->BitsPerPixel * image->w + 31) / 32) * 4;
int padding =
        4 - ( (image->w * image->format->BytesPerPixel) % 4);
```

3.1 Operacje na tabeli pikseli

Przestawianie pikseli w tablicy pikseli należy do trudniejszych zadań. Należy pamiętać o kolejności wierszy w pliku, nie można zapomnieć o występowaniu paddingu co utrudnia operacje przeprowadzane za pomocą prostych pętli for, poza tym działamy na macierzy jednowymiarowej interpretowanej później jako macierz dwuwymiarowa. Dla przykładu kod w Listing 2 przedstawia prostszą funkcję zmieniającą kolejność pikseli. W tym przypadku nie potrzebna nam była wartość *paddingu*. Jest to prosta suma pętli która zamienia piksele miejscami. W poziomie zakres jest od 0 do wartości *rowSize*. Dużo trudniejsze było napisanie funkcji która odbijała by obraz po przekątnej (Listing 3). Przydała się tam wartość *paddingu* dzięki której można było w szybki sposób ograniczać działanie na tablicy. Najistotniejsze jednakże jest to, aby przenosić wartości pełnych pikseli, a nie wyłącznie kolejne bajty, gdyż wtedy możemy zakłócić kolory poszczególnych pikseli.

Listing 2: Funkcja odbijająca obraz w pionie

```
void verticalFilter(unsigned char * buf, int width,
                int height,int size,char bpp, int rowSize)
{
  int i, j, k;
  char temp;
    for(i=0; i < (height/2) ; i++)
    {
      for(j=0; j < (width*bpp); j++)
      {
        temp = buf[(i *rowSize) +j];
        buf[ (i *rowSize) +j] = buf[ ((height-i-1) *rowSize) +j ];
        buf[ ((height-i-1) *rowSize) +j ] = temp;
      }
    }
}
```

Listing 3: Funkcja odbijająca obraz po przekątnej

```
void diagonalFilter(unsigned char * buf, int width,
        int height,int size,char bpp, int RowSize, int Padding)
{
  int i, j, k;
  char temp;
  for(i=0; i < height/2; i++)
  {
    for(j=0; j < (width); j++)
    {
      for(k=0; k<bpp; k++)
      {
        temp = buf[ ( i *(RowSize) ) + ( j *bpp ) +k ];

        buf[ ( i * RowSize) ) +( j *bpp ) +k ] = buf[ ( (height-
        i) * (RowSize) - Padding ) - ( j * bpp ) + k ];

        buf[ ( (height-i) *(RowSize) -Padding ) -
        ( j *bpp ) +k ] = temp;
      }
    }
  }
}
```

3.2 Operacje na wartościach pikseli

Dużo łatwiejsze jest przeprowadzanie tej samej operacji na wartościach pikseli. W przypadku negatywu (Listing 4) sprawa wygląda dość prosto. Wystarczy zanegować wartości każdego bajta, i w ten sposób otrzymujemy obraz w negatywie. Operacje na poszczególnych pikselach są o tyle łatwiejsze, że uruchamiamy odpowiednio przygotowany algorytm

na każdym kolejnym wektorze bajtów, bez potrzeby zastanawiania się jak są one rozmieszczone w tabeli pikseli.

Listing 4: Funkcja odbijająca obraz w pionie

```
void negativeFilter(unsigned char * buf, int width,
                int height,int size,char bpp, int RowSize)
{
  int i, j, k;
  char temp;
  for(i=0; i < height ; i++)
  {
    for(j=0; j < (width*bpp); j++)
    {
      buf[ (i*RowSize) + j] = (~buf[ (i*RowSize) + j]);
    }
  }
}
```

4 Wnioski

Laboratorium nauczyło mnie jak zbudowane są pliki graficzne. Pozwoliło to także zrozumieć sposób w jaki działają programy graficzne, jak działają filtry czy rysowanie. Zapewne po utworzeniu funkcji która konwertowałaby wektor jednowymiarowy w dwuwymiarowy, praca nad filtrami była by dużo prostsza a i kod byłby dużo prostszy w czytaniu. Niestety ze względu na trudność w początkowym zrozumieniu pracy nad tablicą trójwymiarową zapisaną jako wektor jednowymiarowy nie udało się podczas zajęć laboratoryjnych stworzyć kodu w ASM a także porównać czas działania. Mogę wyłącznie przewidywać, że czas działania będzie przynajmniej kilka razy większy ze względu na możliwość działania na kilku wartościach jednocześnie, co podobne w działaniu jest do pracy na wątkach.

Spis listingów

Literatura

[1] Strona internetowa: https://en.wikipedia.org/wiki/BMP_file_format [dostęp 31 maja 2016].

[2] *IA-32 Intel Architecture Software Developer's Manual* [dostępny w wersji elektronicznej: http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium/Dokumentacja/Intel%20Penium%20IV/].

[3] Strona internetowa: http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium/MMX/ [dostęp 31 maja 2016].