Michał Szczygieł	Aleksander Śmierciak	25/05/2014

Zadanie 6 - Rozmycie Gaussa GPU

Celem zadania numer 6 było wdrożenie złożonego programu umożliwiającego zastosowanie filtru - rozmycie Gaussa na materiał video. Głównym założeniem tej aplikacji było zaimplementowanie własnego rozwiązania rozmycia z wykorzystaniem mocy obliczeniowej kart graficznych firmy Nvidia, za pomocą architektury CUDA. Do wykonania należało także wykorzystać dostępną bibliotekę OpenCV, która umożliwa manipulację strumieniem wejściowym video, w postaci macierzy zawierających informację o pikselach (obiektu typu *Mat*).

Poniżej została przedstawiona lista kroków do realizacji zadania:

- pobranie argumentów
- obliczenie gridu, bloku i pobranie liczby wątków
- otworzyć materiał filmowy, klatka po klatce
- zastosować filtr Gaussa poprzez dokonanie modyfikacji informacji o pikselach w każdej z klatek - obliczenia zrównoleglić z pomocą CUDA
- zapisać kolejno zmodyfikowane klatki
- zapisać całość po zastosowaniu odpowiedniego filtra video

Do implementacja rozmycia Guassa została użyta tradycyjna metoda bazująca na przekształceniu piskeli przez macierz charakterystyczną dla rozmycia gaussa.

W przypadku zastosowania archtektury CUDA do obliczeń równoległych, było odpowiednie przygotowanie gridu i rozdzielenie zadania na wątki.

Część programu odpowiedzialna za podział:

```
1  /**
2   * This method based on run parameters. Calculates the amount of threads/
        blocks for the application use.
3   *
4   * @param threadCount The amount of threads.
5   */
6  void prepareGrid(unsigned int threadCount)
7  {
8     // Round the number of threads
9     if ((threadCount % 2) == 1)
10    {
11       ++threadCount;
12    }
13
14    // Divide into blocks
15    for (int i = 512; i > 0; i--)
16    {
17       double blocks = (double) threadCount / i;
18
19       if (blocks == (int) (blocks))
20       {
21          double divThreads = sqrt(i);
22          if (divThreads == int(divThreads))
```

```
            {
                threadsOnX = int(divThreads);
                threadsOnY = int(divThreads);
            }
            else
            {
                threadsOnX = i;
                threadsOnY = 1;
            }

            break;
        }
    }
}
```

Znając specyfikację karty graficznej, na której przeprowadzone były testy programu, należało paramter wejściowy, oznaczający liczbę wątków na której działa aplikacja odpowiednio zinterpretować. Powyższy kod, dzieli na bloki (obliczanie wymiarów tzw. *grid'a*) w zależności od liczby wątków.

Implementacja Kernela CUDA:

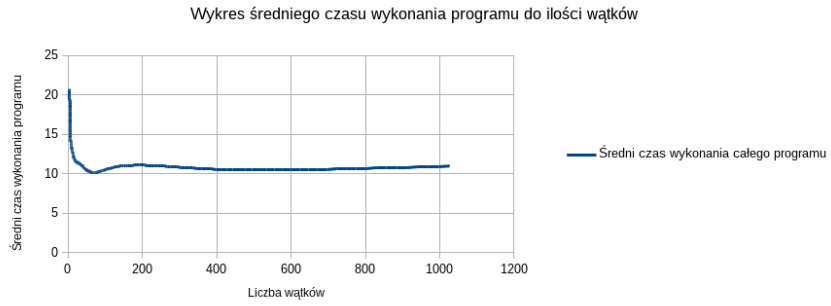
```
/**
 * CUDA implementation for Gaussian blur.
 *
 * @param imageIn       The pixel to perform.
 * @param imageOut      The result of pixel conversion.
 * @param width         The width of frame.
 * @param height     The height of frame.
 * @param channels      The channels color for pixel.
 * @param kernel     The kernel for gaussian blur.
 * @param kernelSize The size of kernel.
 * @param adjustedSize  The adjusted size of kernel (usually the half of
    kernel size rounded down).
 * @param sum        The sum of all kernel values.
 */
__global__ void gaussBlur(unsigned char *imageIn, unsigned char *imageOut,
      int width, int height, int channels, float *kernel, int kernelSize,
      int adjustedSize, int sum)
{
    const int index = getThreadId();

    if (index < width * height)
    {
        if (!(index % width < adjustedSize
              || index % width >= width - adjustedSize
              || index / width < adjustedSize
              || index / width >= height - adjustedSize))

        {
            float x = 0.0f;
            float y = 0.0f;
            float z = 0.0f;

            for (int j = 0; j < kernelSize * kernelSize; ++j)
            {
                // Compute index shift to neighboring cords.
                int shift = (j / kernelSize - adjustedSize) * width
```

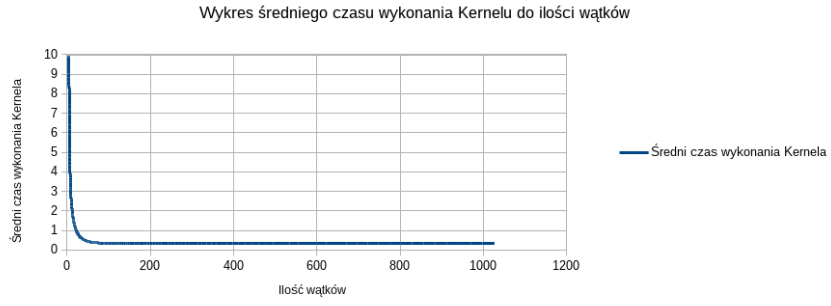
```
                    + j % kernelSize - adjustedSize;

                x += imageIn[(index + shift) * channels] * kernel[j];
                y += imageIn[(index + shift) * channels + 1] * kernel[j];
                z += imageIn[(index + shift) * channels + 2] * kernel[j];
            }

            // Apply to output image and save result.
            imageOut[index * channels] = (unsigned char) (x / sum);
            imageOut[index * channels + 1] = (unsigned char) (y / sum);
            imageOut[index * channels + 2] = (unsigned char) (z / sum);
        }
    }
}
```

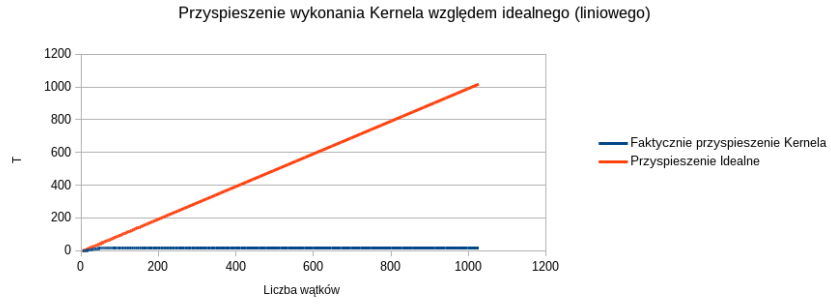
Przebieg

Obliczenia zostały wykonane na serwerze CUDA.

Poniżej wykresy przedstawiający rezultat przeprowadzonych operacji na wątkach.







Wnioski

Najlepsze przyspieszenie udało się osiągnąć dla 128 wątków (dla filmu Helicopter DivXHT ASP.divx). Jak widać nie da się osiągnąć większego przyspieszenia, bowiem rozmiar macierzy (klatki filmu) jest ograniczony i w pewnym momencie nie ma już możliwości dalszego dzielenia obliczen pomiedzy wątki. Dodatkowym obciążeniem mogą być konieczności przełączania kontekstów, jednak nie udało nam się znaleźć jednoznacznego potwierdzenia istnienia takiego problemu dla procesorów GPU w literaturze przedmiotu.