Spis treści

Wstęp

Niniejsza dokumentacja opisuje działanie i wykonanie programu, który umożliwia oglądanie trójwymiarowego modelu głowy człowieka. Docelową grupą użytkowników są lekarze. Program miałby ułatwić diagnozę schorzeń związanych z głową. Model trójwymiarowy generowany jest na podstawie zdjęć tomografii komputerowej.

Pierwsze rozdziały dokumentacji są skierowane do użytkownika. Opisują sposób obsługi i możliwości programu. Opisany jest również sposób działania programu. Ostatnie rozdziały dotyczą szczegółów implementacyjnych. Przeznaczone są dla osób których zadaniem będzie konserwacja, lub rozwój tego oprogramowania. Należy podkreślić, że program działa w oparciu o wielowątkową architekturę NVIDIA CUDA i poprawne działanie programu zagwarantowane jest tylko na komputerze wyposażonym w taki moduł.

Sposób użytkowania

Instalacja

Program nie wymaga specjalnej instalacji. Całość zawiera się w jednym folderze.

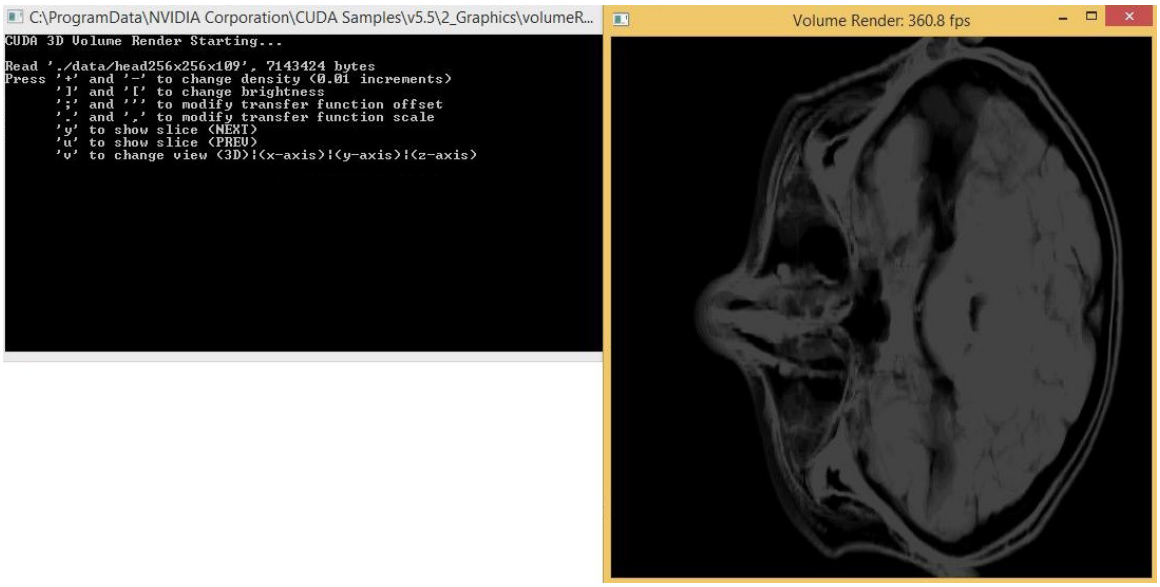
Ilustracja 1: Zawartość folderu z programem

Aby program działał poprawnie wymagane jest aby został uruchomiony na komputerze wyposażonym w kartę graficzną z technologią NVIDIA CUDA. Program uruchamia ikona aplikacji o nazwie *BrainModel3D*. Poprawnie uruchomiony program uruchomi dwa okna. Jedno z menu programu, a drugi z trójwymiarowym modelem głowy. Folder *data* zawiera dane które zostaną wczytane przez program. Znajduje się tam plika *head.raw*. Zawiera on zbiór zdjęć wykonanych tomografią komputerową. Pozostałe pliki są niezbędne do uruchomienia aplikacji, są to tak zwane biblioteki. Umożliwiają one min. wyświetlanie zdjęć na ekranie.

Zainstalowanie programu polega na skopiowaniu całego folderu wraz z zawartością na komputer. Jeśli wszystkie pliki będą umieszczone w taki sposób ja ka rysunku to program będzie poprawnie działał.

Obsługa programu

Uruchomiony program posiada dwa okna. Okno z menu (Ilustracja 2) wykonuje się na procesorze i odpowiada za obsługę przycisków klawiatury. Natomiast drugie okno wykonywane jest przez kartę graficzna i technologię CUDA.



Ilustracja 2: Okna aplikacji

Posługując się wskazanymi przyciskami na klawiaturze można manipulować modelem trójwymiarowym. Program najpierw wczytuje obrazy CT i każdy punkt obrazu przekłada na punkt w modelu. Wyświetlać można albo cały model 3D, albo przekroje. Każdy przekrój jest złożeniem kilku warstw punktów. Im więcej warstw tym więcej widocznych elementów, ale również jasność punktu zależy od nałożonych warstw. Następuje wtedy ich wzmocnienie.

Dostępne funkcjonalności:

- +/- - można zmieniać liczbę nałożonych warstw
-]/[- można zmieniać jasność punktów
- ;/ ' - zmienia offset
- . /, - przybliża i oddala obraz
- y – kolejny przekrój
- u – poprzedni przekrój
- v – przełącza widok. Model 3D | przekroje wzdłuż osi X | Y | Z

Dodatkowo obsługa jest ułatwia posługiwanie się myszką komputerową. Przytrzymując lewy klawisz myszy i przesuwając można obracać widok (model, jak i przekrój) w dowolnym kierunku. Osie obrotu przechodzą przez środek układu współrzędnych. Przytrzymanie prawego klawiszu pozwala przybliżać i oddalać widok. Należy być ostrożnym, bo jest to bardzo wrażliwy mechanizm.

Część techniczna

Czym jest CUDA

Technologia CUDA pozwala na wykorzystanie mocy obliczeniowej procesorów kart graficznych do rozwiązywania problemów numerycznych w sposób wydajniejszy niż w tradycyjnych, sekwencyjnych procesorach ogólnego zastosowania. Wąskim gardłem przy korzystaniu z tej technologii może być przesyłanie dużych ilości danych poprzez magistralę PCI-Express. Dlatego w naszym projekcie postanowiliśmy dane przechowywać w pamięci operacyjnej karty graficznej. Przesłanie danych składa się z dwóch etapów. Najpierw należy zadeklarować ile pamięci będzie potrzebne, a następnie kopiuje się dane w ten obszar. W ten sposób każdy z wątków uruchomionych na karcie graficznej ma szybki dostęp do danych, jednocześnie nie obciążając przy tym magistrali.

Renderowanie obrazu odbywa się na karcie graficznej przy pomocy kilkuset wątków. Każdy wątek dostaje ten sam zestaw danych i tylko na podstawie swojego id i id grupy wątków dowiaduje się który fragment obrazu ma opracować. W naszym programie każdy wątek wylicza kolor pojedynczego teksela, więc należało opracować w jaki sposób mają być pogrupowane, aby wygenerować cały obraz i żadne dwa wątki nie pracowały nad tym samym fragmentem. Zrealizowane zostało to poprzez podzielenie obrazu na 256 obszarów, każdy z nich odpowiada blokowi wątków, a w każdym bloku znajduje się liczba wątków odpowiadająca liczbie punktów w obszarze.

Pojedynczy wątek realizuje obliczenia mające na celu ustalenie koloru jednego teksela. Pierwszym krokiem jest ustalenie, którą część obrazu dany wątek generuje. W tym celu wyliczane są na podstawie id wątku i id grupy wątków, współrzędne generowanego obrazu, które później zamieniane są na współrzędne określające miejsce zapisu wyniku obliczeń. Kolejnym krokiem jest zdefiniowanie parametrów promienia pierwotnego, czyli półprostej biegnącej od obserwatora przechodzącej przez badany piksel i w kierunku sceny. Następnie sprawdzamy, w którym miejscu ten promień przecina naszą chmurę punktów. Kolor piksela uzyskujemy poprzez zsumowanie wartości napotkanych przez promień tekseli. Aby zbyt szybko nie uzyskać wartości maksymalnej dodajemy tylko mały procent koloru, a na końcu mnożymy przez jasność. Gdy wszystkie wątki skończą swoje obliczenia uzyskujemy kompletny obraz gotowy do wyświetlenia.

Źródła danych

Do projektu wykorzystaliśmy dane ze strony:

www.codeproject.com/KB/openGL/352270/head256x256x109.zip

Można z niej pobrać plik w formacie RAW. Zdjęcia w tym formacie są nazywane cyfrowymi negatywami. Nie bez przyczyny – możliwości, jakie daje RAW – stawiają go na równi z kliszą. To po prostu zapis danych z matrycy światłoczułej, niepoddany żadnej obróbce. W związku z tym każdy producent ma własny format RAW, np. CANON ma CRW.

W naszym projekcie są to surowe zdjęcia otrzymane po badaniu CT. Jeden plik to nie tylko jedno zdjęcie. Zapisane są wszystkie otrzymane zdjęcia ułożone w logicznej kolejności. Format ten nie jest tak popularny jak DCM, ale szybciej zaimplementowaliśmy wczytywanie obrazu z formatu RAW. DCM są specjalnym formatem do zapisu danych medycznych zgodnych ze standardem DICOM. Standard medyczny pozwolił na ujednolicenie wymiany i interpretacji danych medycznych reprezentujących lub związanych

z obrazami diagnostycznymi w medycynie.

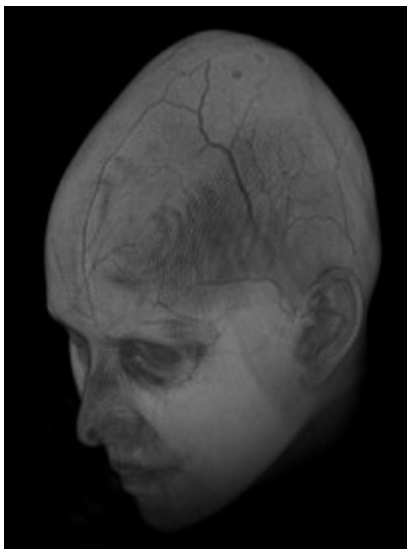
Istniejące rozwiązania

Przed przystąpieniem do opracowywania własnego projektu poszukiwaliśmy już wykorzystywanych rozwiązań. Żadne z nich nie wykorzystuje technologii CUDA, choć oczywiście korzystają z wielowątkowości.

- OsiriX – Open-Source projekt służący do obsługi obrazów medycznych. W ramach projektu powstało wiele rozwiązań, min. 2D Viewer, OsiriX for iPhone. Żaden z programów nie umożliwia wczytania zdjęć do modelu 3D.
- BrainVoyager Brain Viewer – komercyjne rozwiązanie. Wczytuje dane z CT. Umożliwia przeglądanie zdjęć, i zaznaczanie fragmentów mózgu. Potrafi nawet stworzyć model 3D mózgu na podstawie zdjęć. Jednak jest to model generowany, w związku z tym nie odwzorowuje całkowicie struktury głowy.
- Brain Browser – Open-Source projekt który pozwala renderować obraz 3D mózgu na podstawie danych MINC. Projekt wykorzystuje do tego tylko JavaScript i HTML 5.

Wczytywanie danych

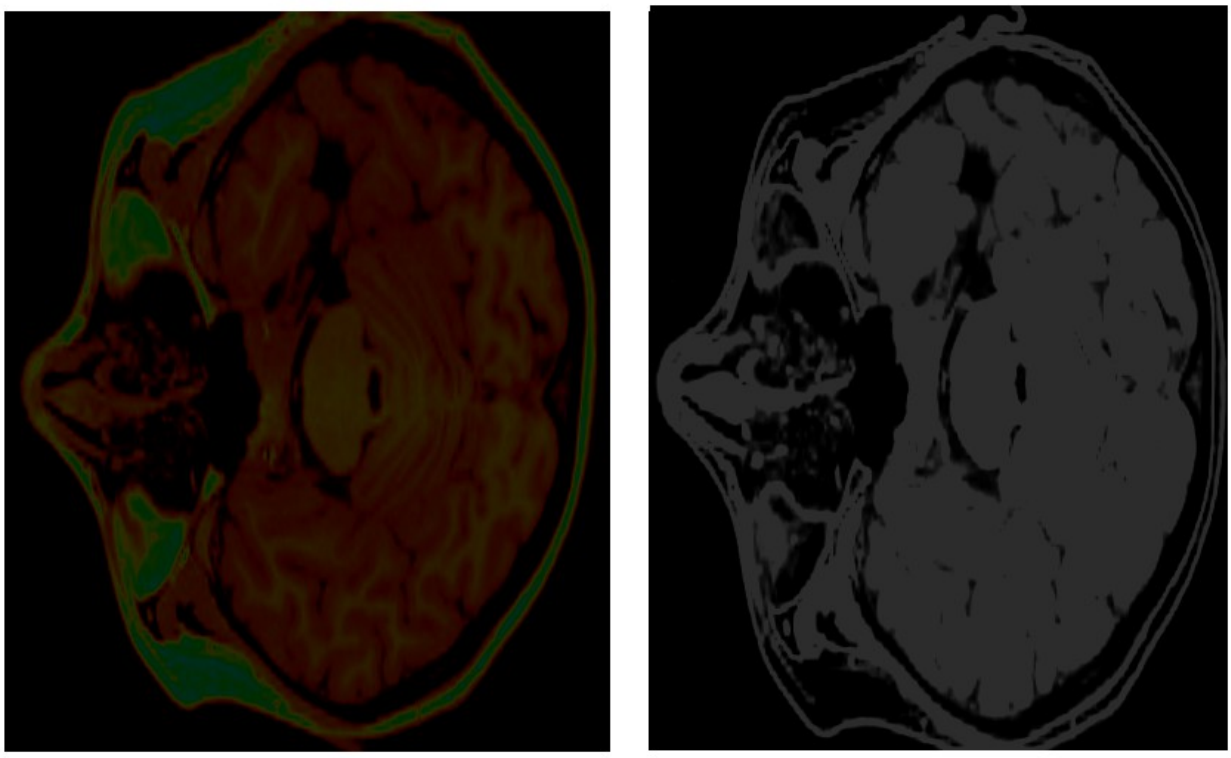
Pliki do programu wczytywane są w formacie RAW. Dane medyczne głównie przechowywane są w formacie DICOM (*.dcm), lecz formacie RAW są podobnie łatwo dostępne w bazach medycznych. Po głębszym rozpoznaniu metod i funkcji do wyświetlania oraz renderowania grafiki korzystniejsze i bardziej przystępne jest wykorzystanie danych medycznych z plików RAW, ponieważ jego struktura pozwala na sklejenie w jednym pliku nawet całej serii obrazów DICOM, a przez to możliwe jest przechowanie całego obiektu trójwymiarowego.



Ilustracja 3: Widok modelu 3D

Podczas wczytywania obiektu 3D do programu napotkany został problem z reprezentacją kolorów z danych binarnych do funkcji renderującej obraz. Powodem takiego zachowania było to, że obrazy medyczne są przechowywane i zapisywane w skali szarości (8bitów). Z kolei dane wczytywane przez aplikacje traktowane były jako dane o kolorze piksela w formacie RGBA. Rozwiązaniem problemu jest przekonwertowanie wartości skali szarości na format RGBA. Pozwala na to przedstawiona niżej funkcja.

```
__device__ uint rgbaFloatToGrayScale(float4 rgba)
{
	rgba.x = __saturatef( rgba.x);
	rgba.w = __saturatef( rgba.w);
	float x = rgba.x*255.0f;
	float w = rgba.w*255.0f;
	float color = uint(x);
	return (uint(w)<<24) + (uint(color)<<16) +
		(uint(color)<<8) + uint(color);
}
```



Ilustracja 4: Po lewej: Błędna reprezentacja skali szarości. Po prawejOdpowiednie odwzorowanie.

Przesłanie danych do pamięci karty graficznej

Gdy mamy załadowany plik z danymi w naszej aplikacji i wybrane urządzenie wspierające obsługę CUDA, następuje inicjalizacja za pomocą poniższej funkcji:

```
void initCuda(void *h_volume, cudaExtent volumeSize)
```

gdzie `h_volume` jest wskaźnikiem na dane, a `volumeSize` rozmiarem tych danych. Poniższe fragmenty funkcji `initCuda` przedstawiają w jaki sposób jest to realizowane. Pierwszym krokiem jest alokacja pamięci na urządzeniu:

```
checkCudaErrors(cudaMalloc3DArray(&d_volumeArray, &channelDesc, volumeSize));
```

- `d_volumeArray` jest globalną tablicą na dane wczytanego obrazu,
- `channelDesc` to struktura opisująca format danych,
- `volumeSize` jest rozmiarem pamięci, którą chcemy zaalokować.

Następnie kopiujemy wczytane dane do tablicy trójwymiarowej poprzez zdefiniowanie odpowiedniej struktury opisującej tą operację i wywołanie funkcji `cudaMemcpy3D`, która przekopiuje naszą chmurę punktów na pamięć karty graficznej. Tablica ta jest od tej pory dostępna przez wskaźnik `d_volumeArray`.

```
cudaMemcpy3DParms copyParams = {0};
copyParams.srcPtr   = make_cudaPitchedPtr(h_volume,
volumeSize.width*sizeof(VolumeType), volumeSize.width, volumeSize.height);
copyParams.dstArray = d_volumeArray;
copyParams.extent   = volumeSize;
copyParams.kind     = cudaMemcpyHostToDevice;
checkCudaErrors(cudaMemcpy3D(&copyParams));
```

Kolejnym krokiem jest ustawienie parametrów globalnej tekstury `tex` i powiązanie jej z tablicą trójwymiarową.

```
tex.normalized = true;
tex.filterMode = cudaFilterModeLinear;
tex.addressMode[0] = cudaAddressModeClamp;
tex.addressMode[1] = cudaAddressModeClamp;

checkCudaErrors(cudaBindTextureToArray(tex, d_volumeArray, channelDesc));
```

Poprzez tą teksturę będziemy mieli dostęp do danych w każdym wątku uruchomionym na karcie graficznej.

Tworzenie obrazu do wyświetlenia

Funkcja renderująca wywoływana jest na karcie graficznej w kilkuset wątkach równocześnie. Każdy wątek dostaje taki sam zestaw danych i tylko na podstawie swojego id i id grupy wątków dowiaduje się który fragment obrazu ma opracować. W naszym programie każdy wątek wylicza kolor pojedynczego piksela, więc należało opracować w jaki sposób mają być pogrupowane, aby wygenerować cały obraz i żadne dwa wątki nie pracowały nad tym samym fragmentem.

Zrealizowane zostało to poprzez podzielenie całego obrazu na 256 obszarów.

```
dim3 blockSize(16, 16);
```

Następnie policzono ile pikseli znajduje się w każdym segmencie:

```
gridSize = dim3(iDivUp(width, blockSize.x), iDivUp(height, blockSize.y));
```

Można teraz przejść do renderowania obrazu wywołując funkcję `d_render` na wyliczonej wcześnie liczbie wątków podzielonych na odpowiednie bloki:

```
d_render <<<gridSize, blockSize>>> (d_output, imageW, imageH, density,
                brightness, transferOffset, transferScale, ySliceShow, viewType);
```

- `d_output` – wskaźnik na tablicę wyjściową
- `imageW` – szerokość obrazu w pikselach
- `imageH` – wysokość obrazu w pikselach
- `density` – nasycenie obrazu
- `brightness` – jasność obrazu
- `ySliceShow` – grubość przekrojów
- `viewType` – rodzaj widoku

W samej funkcji d_render najpierw wyliczamy współrzędne piksela:

```
uint x = blockIdx.x*blockDim.x + threadIdx.x;
uint y = blockIdx.y*blockDim.y + threadIdx.y;
if ((x >= imageW) || (y >= imageH)) return;
```

Następnie wyliczamy współrzędne tekstury na której wykonujemy wszystkie operacje:

```
float u = (x / (float) imageW)*2.0f-1.0f;
float v = (y / (float) imageH)*2.0f-1.0f;
```

Później definiujemy promień, biegnący od piksela przez trójwymiarową teksturę:

```
Ray eyeRay;
eyeRay.o = make_float3(mul(c_invViewMatrix,
                    make_float4(0.0f, 0.0f, 0.0f, 1.0f)));
eyeRay.d = normalize(make_float3(u, v, -2.0f));
eyeRay.d = mul(c_invViewMatrix, eyeRay.d);
```

Szukamy miejsca pierwszego i ostatniego przecięcia z teksturą:

```
intersectBox(eyeRay, boxMin, boxMax, &tnear, &tfar);
```

Ustawiamy zmienną `pos` na pierwsze przecięcie z teksturą:

```
pos = eyeRay.o + eyeRay.d*tnear;
```

Definiujemy przesunięcie wzdłuż promienia:

```
step = eyeRay.d*tstep;
```

Wchodzimy do pętli, która kończy się w momencie dojścia do ostatniego punktu przecinającego teksturę, osiągnięciu przez piksel maksymalnej jasności, lub po osiągnięciu maksymalnej liczby powtórzeń.
Pobranie wartości teksela z pozycji `pos`:

```
sample = tex3D(tex, pos.x*0.5f+0.5f, pos.y*0.5f+0.5f, pos.z*0.5f+0.5f);
```

Sumowanie jasności piksela:

```
sum = sum + col*(1.0f - sum.w);
```

Ustawienie następnego punktu na linii promienia:

```
pos += step;
```

Po wyjściu z pętli przemnażamy przez współczynnik jasności, aby można było ustawić czytelny widok:

```
sum *= brightness;
```

Na koniec zapisujemy wynik w odpowiedniej komórce tablicy wyjściowej

```
d_output[y*(imageW) + x] = rgbaFloatToGrayScale(sum);
```

Podsumowanie projektu

W ramach projektu udało nam się wykonać aplikację wykorzystującą technologię NVIDIA CUDA do wyświetlania modelu mózgu. Rozwiązanie może ułatwić proces diagnostyczny lekarzy zajmującymi się schorzeniami głowy. Żadne ze znanych nam rozwiązań nie wykorzystuje do pracy technologię CUDA. Ich rozwiązania nie potrafią renderować obrazu głowy od razu na podstawie obrazów CT, ale muszą analizować obraz i generują przybliżony model mózgu. Rozwiązanie wykorzystujące technologię NVIDII działa dużo szybciej i pozwala na szybką obróbkę obrazu.

Technologia CUDA nie jest ani prosta w działaniu i wymaga czasochłonnej nauki. Jest to zupełnie inne podejście do programowania Zupełnie nowe dla osób programujących na co dzień obiektowo. Kiedy jednak pozna się już sposób działania można zrozumieć dlaczego programista musi godzić się na niektóre niewygodne rozwiązania. Kart graficzne posiadających te mechanizmy nie są standardowym wyposażeniem komputerów. Z tego powodu praca z CUDą może nie być dostępna dla każdego.

Stworzona aplikacja pozwala na wczytywanie modelu ze zdjęć zapisanych bezpośrednio przez tomograf, tzw. plików RAW. Użytkownik może obejrzeć powstałe przez tomograf zdjęcia w wygodnej wormie sekwencji zdjęć. Może również wyświetlić trójwymiarowy model, który może obejrzeć z dowolnej strony oraz dowolnie przybliżać. Możliwe jest również obejrzenie przekroju głowy pacjenta pod dowolnym kątem.

Problem generowania modelu 3D mózgu jest wciąż rozwiązywany. Technologia CUDA może znacznie ułatwić pracę rozwiązaniami. Znacznie przyspiesza pracę aplikacji. Niestety wymaga solidnego przygotowania i gruntownej wiedzy z zakresu kart graficznych NVIDIA.

Źródła

[1] http://www.daimi.au.dk/~trier/?page_id=98

[2] http://brainweb.bic.mni.mcgill.ca/brainweb/selection_ms.html

[3] http://www.brainvoyager.com/products/brainviewer.html

[4] http://www.osirix-viewer.com/Snapshots.html

[5] https://brainbrowser.cbrain.mcgill.ca

[6] http://www.codeproject.com/Articles/352270/Getting-started-with-Volume-Rendering