

VoxelIt

DOKUMENTACJA TECHNICZNA

Krystian Dziwa | Rzeszów 2018

Spis treści

Opis aplikacji.....	4
Architektura biblioteki	4
Zależności zewnętrzne	4
Algorytm transformacji Siatki	4
Działanie aplikacji.....	6
Podsumowanie	8
Diagram Gantta.....	8
Oczekiwany czas trwania.....	8
Finalny czas trwania.....	9
Post mortem.....	9
Wykonane zadania.....	10
zadania niewykonane.....	10
dalszy rozwój aplikacji	10
Bibliografia	10

Opis aplikacji

VoxelIt jest biblioteką pozwalającą na konwersję obiektów 3d złożonych z trójkątów do ich odpowiedników złożonych z wokseli. Woksel jest trójwymiarowym odpowiednikiem piksela w grafice 2d. Pیکsel charakteryzuje pozycja w płaszczyźnie 2d oraz kolor, natomiast woksel opisuje pozycję w przestrzeni oraz kolor. Pozycja woksela w przestrzeni jest skwantowana – podobnie jak pozycja piksela na płaszczyźnie. Oznacza to, że pojedyncze woksle mogą znajdować się tylko w ściśle określonych punktach przestrzeni.

Tradycyjne podejście do tworzenia obiektów 3d zakłada wykorzystanie wielokątów złożonych z wierzchołków. Połączenia między wierzchołkami opisują tablice indeksów. Dodatkowo, każdy z wierzchołków może zawierać dodatkowe informacje, np. wektor normalny (wykorzystywany m. in. przy obliczaniu światła), kolor wierzchołka czy też różnego rodzaju wagi (potrzebne przy animacji szkieletowej). Algorytm transformacji tych wielokątów na woksle jest głównym elementem aplikacji. Zakłada on, wejściowe wielokąty są trójkątami, każdy z tych trójkątów jest osobno konwertowany na odpowiedni zestaw wokseli.

Architektura biblioteki

Aplikacja składa się z dwóch elementów – aplikacji konsolowej pozwalającej na konwersję obiektów oraz biblioteki, która odpowiada za właściwą konwersję oraz dostarcza odpowiednie typy danych. Całość została napisana wykorzystując najnowsze standardy języka C++. Biblioteka dostarcza możliwość linkowania statycznego oraz dynamicznego. Struktura projektu pozwala na kompilację z wykorzystaniem kompilatora Visual C++ v14.1.

ZALEŻNOŚCI ZEWNĘTRZNE

Biblioteki zewnętrzne, użyte do realizacji projektu odpowiadają za niektóre podstawowe funkcjonalności aplikacji. Elementami tymi są:

- Serializacja obiektów przy pomocy biblioteki Nlohmann Json (v3.1.1)
- Obiekty i funkcje matematyczne z biblioteki GLM (v0.9.9.0)
- Import zasobów za pomocą biblioteki Assimp (v4.1.0)

Automatyczna dokumentacja projektu może zostać wygenerowana przy pomocy oprogramowania doxygen. Dokumentacja ta zawiera opis wszystkich klas oraz ich metod.

ALGORYTM TRANSFORMACJI SIATKI

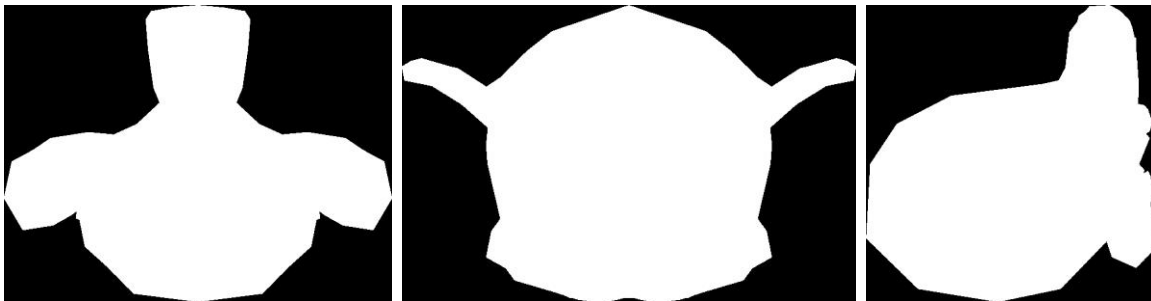
Obsługę transformacji siatki (Mesh) na bryłę złożoną z wokseli (Shape) zapewnia obiekt klasy MeshShapeConverter. Algorytm wykonywany jest dla każdego trójkąta osobno, dodatkowymi parametrami potrzebnymi podczas działania algorytmu są: ramka ograniczająca obiekt (bounding box) oraz rozmiar woksela. Dzięki ramce ograniczającej

można ustalić rozmiary trzech płaszczyzn rzutowania obiektu (List. 1), na których będzie rasteryzowany każdy z trójkątów zawartych w siatce obiektu.

```
auto bBox = mesh->getBoundingBox();  
...  
bBox.lower -= vec3(mWokselSize, mWokselSize, mWokselSize);  
bBox.upper += vec3(mWokselSize, mWokselSize, mWokselSize);  
tvec3<int32_t> planesLower = bBox.lower / mWokselSize;  
tvec3<int32_t> planesUpper = bBox.upper / mWokselSize;  
int32_t width = planesUpper.x - planesLower.x;  
int32_t height = planesUpper.y - planesLower.y;  
int32_t depth = planesUpper.z - planesLower.z;  
vector<uint8_t> xyPlane(width * height);  
vector<uint8_t> zyPlane(depth * height);  
vector<uint8_t> xzPlane(width * depth);
```

List. 1 generowanie trzech płaszczyzn rzutowania.

Pierwszym krokiem algorytmu jest wspomniana już rasteryzacja. Odbywa się ona trzy krotnie, rzutując dany trójkąt na każdą z płaszczyzn. Efekt jaki osiągniemy nie czyszcząc płaszczyzn pomiędzy poszczególnymi iteracjami algorytmu można zaobserwować na poniższym rysunku (Rys. 1)



Rys. 1 rzutowanie obiektu na trzy płaszczyzny

Następnie, gdy istnieją już 3 rzuty danego trójkąta – wybieramy z nich ten, w którym trójkąt zajmuje największą powierzchnię, po czym przeprowadzamy następny krok algorytmu. Wraz z wygenerowanym rzutem trójkąta, obliczana zostaje również ramka ograniczająca ten trójkąt, której koordynaty odpowiadają pozycjom wokseli. Iterując po wszystkich wokselach zawartych w danej ramce, wybieramy te, których koordynaty odpowiadają pikselom zawartym w rzucie trójkąta. Dla wybranych elementów obliczamy wartość ostatniej współrzędnej wokselu po czym dodajemy go do zestawu wszystkich wokseli. (List. 2)

```
for (int32_t j = zyBBox.y; j < zyBBox.w; ++j)  
{  
    for (int32_t i = zyBBox.x; i < zyBBox.z; ++i)
```

```

{
    if (zyPlane[(j - planesLower.y) * depth + (i - planesLower.z)] != 1)
        continue;

    auto newA = vec3(a.z, a.y, a.x);
    auto newB = vec3(b.z, b.y, b.x);
    auto newC = vec3(c.z, c.y, c.x);
    int32_t x = planeFunc(newA, newB, newC, i * mVoxelSize,
        j * mVoxelSize, mVoxelSize) / mVoxelSize;
    voxels.insert(Voxel({ x, j, i }));
}
}

```

List. 2 generacja wokseli.

Funkcja `planeFunc()` (List. 3) tworzy z podanych trzech wierzchołków równanie płaszczyzny postaci $a \cdot (x - x_p) + b \cdot (y - y_p) + c \cdot (z - z_p) = 0$, równanie to posłuży w obliczeniu trzeciej współrzędnej wokseli.

```

float MeshShapeConverter::planeFunc(vec3 p, vec3 q, vec3 r,
                                    float x, float y, float voxelSize)
{
    auto v1 = p - q;
    auto v2 = p - r;
    auto cross12 = cross(v1, v2);

    auto a = cross12.x;
    auto b = cross12.y;
    auto c = cross12.z;

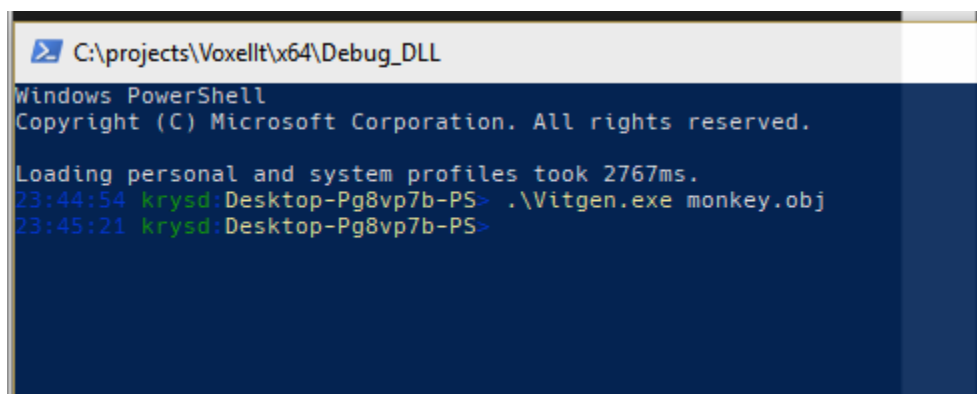
    return (-(a * (x - p.x) + b * (y - p.y)) / c) + p.z -
        (cross12.z < 0 ? voxelSize : 0.0);
}

```

List. 3 funkcja `planeFunc()`.

Działanie aplikacji

Aplikacja konsolowa Vitgen pozwala na konwersję plików `.obj` zawierających siatkę obiektów 3d na pliki `.obj` zawierające voxele zaprezentowane jako sześciiany złożone z trójkątów. Parametrem wejściowym aplikacji (Rys. 2) jest ścieżka do wejściowego pliku `.obj`, wyjście zapisywane jest do pliku „exported.obj”.

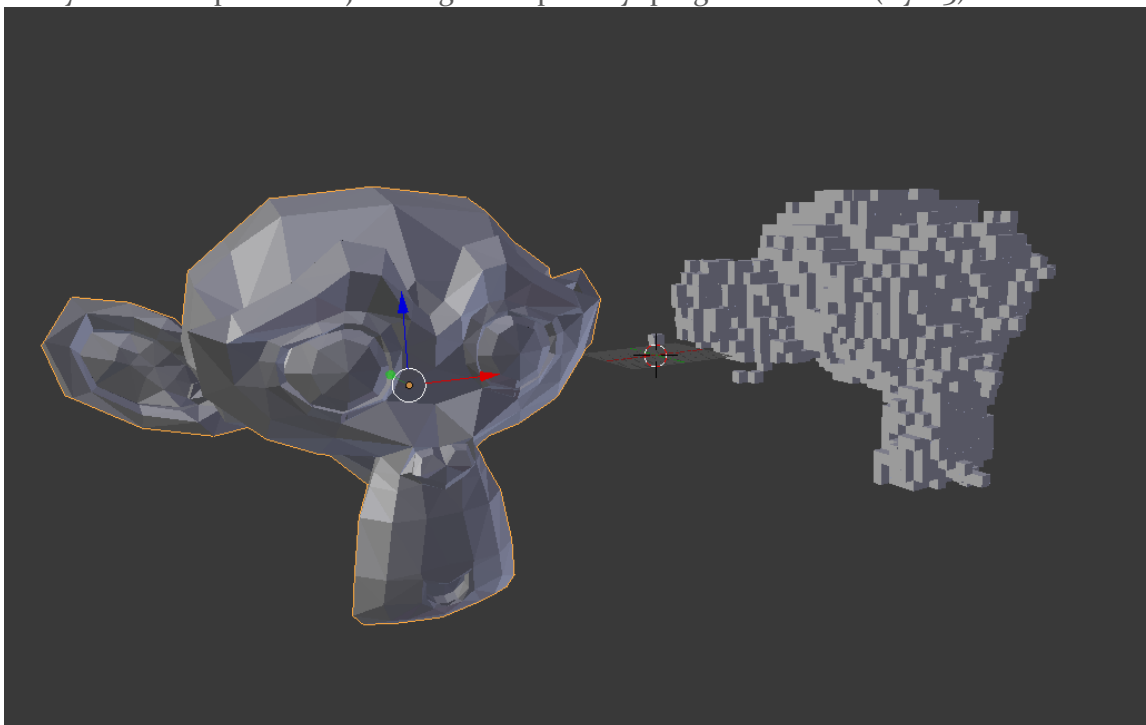


```
C:\projects\VoxelIt\x64\Debug_DLL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Loading personal and system profiles took 2767ms.
23:44:54 krysd:Desktop-Pg8vp7b-PS> .\Vitgen.exe monkey.obj
23:45:21 krysd:Desktop-Pg8vp7b-PS>
```

Rys. 2 uruchomienie aplikacji konsolowej.

Działanie aplikacji można zaobserwować używając oprogramowania pozwalającego na wyświetlanie plików .obj. Do tego celu posłużył program Blender (Rys. 3).



Rys. 3 wynik działania programu

Podsumowanie

Algorytm generowania brył złożonych z wokseli działa, jednak zabrakło czasu na przeprowadzenie stosownych optymalizacji oraz poprawy drobnych błędów przy generowaniu położenia poszczególnych wokseli.

Diagram Gantt

OCZEKIWANY CZAS TRWANIA

Zadanie	Czas trwania	Godz. rozpoczęcia	Godz. zakończenia
Zapoznanie z technikami wokselizacji.	3	0	3
Przygotowanie szkieletu aplikacji	3	3	6
Import plików z siatkami 3d	2	6	8
Zaprojektowanie struktury bryły złożonej z wokseli	3	8	11
Eksport obiektów do pliku	2	11	13
Implementacja algorytmu generowania wokseli	5	13	18
Przygotowanie szkieletu aplikacji graficznej	3	18	21
Realizacja wyświetlania brył złożonych z wokseli	3	21	24
Przygotowanie dokumentacji	2	24	26

FINALNY CZAS TRWANIA

Zadanie	Czas trwania	Godz. rozpoczęcia	Godz. zakończenia
Zapoznanie z technikami wokselizacji.	4:20	0	4:20
Przygotowanie szkieletu aplikacji	2:30	4:20	6:50
Import plików z siatkami 3d	3:00	6:50	9:50
Zaprojektowanie struktury bryły złożonej z wokseli	2:00	9:50	11:50
Eksport obiektów do pliku	1:30	11:50	13:20
Implementacja algorytmu generowania wokseli	8:30	13:20	21:50
Poprawa błędów algorytmu generowania wokseli	4:30	21:50	26:20
Przygotowanie dokumentacji	2:00	26:20	28:20
Przygotowanie szkieletu aplikacji graficznej	-	-	-
Realizacja wyświetlania brył złożonych z wokseli	-	-	-

Post mortem

Projekt mimo braku znacznej części funkcjonalności można uznać za sukces, ponieważ główne założenia zostały zrealizowane. Głównym powodem, przez który

funkcjonalności nie zostały dopięte była zbyt mała ilość czasu. W trakcie planowania projektu, czas przeznaczony na naprawę błędów został niedoszacowany.

WYKONANE ZADANIA

Do elementów wykonanych należy zaliczyć działającą bibliotekę dostarczającą struktury danych oraz algorytmy konwersji pomiędzy siatką 3d a bryłą złożoną z wokseli. Wykonano również prostą aplikację pozwalającą na demonstrację działania biblioteki.

ZADANIA NIEWYKONANE

Głównym elementem, na który zabrakło czasu była realizacja aplikacji wyświetlającej bryły złożone z wokseli przy pomocy biblioteki OpenGL oraz odpowiednich programów cieniujących, pozwalających na zaoszczędzenie ilości danych transmitowanych z pamięci procesora do pamięci karty graficznej.

Kolejnym elementem, który nie został wykonany była optymalizacja i poprawa działania algorytmu generowania wokseli. Aktualnie, algorytm działa sprawnie dla niewielkiej ilości wokseli dochodzącej do kilku tysięcy, niestety już przy większych ich ilościach znacząco zwalnia.

DALSZY ROZWÓJ APLIKACJI

Dalszy rozwój aplikacji może zostać oparty o elementy nie zrealizowane w trakcie prac nad projektem. Dużym krokiem w kierunku optymalizacji działania algorytmu zostało by jego zrównoleglenie lub nawet przygotowanie do wykonania przy pomocy karty graficznej. Bibliotekę można by rozwinąć o elementy pozwalające na szybkie wczytywanie gotowych brył złożonych z wokseli do pamięci oraz moduły pozwalające na szybki rendering obiektów przy pomocy bibliotek takich jak OpenGL czy Vulkan.

Bibliografia

- [1] http://jongarvin.com/up/MCV4U/slides/scalar_plane_handout.pdf, Dostęp 19.06.2018
- [2] <https://developer.nvidia.com/content/basics-gpu-voxelization>, Dostęp 19.06.2018