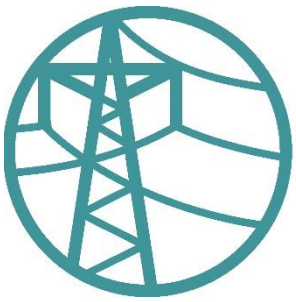


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Katedra Informatyki i Automatyki

Techniki multimedialne

Generowanie chmury punktów z modelu 3D - projekt

Przemysław Szpara

L3

Rzeszów 2018

1. Wprowadzenie

Celem projektu było stworzenie aplikacji, której zadaniem byłoby generowanie chmury punktów z modelu 3D tj. symulacja Kinect'a. Czas poświęcony na projekt miał wynieść około 24 godziny. Aplikacja została napisana w języku C# z pomocą silnika graficznego Unity, który posiada wiele użytecznych funkcji, dzięki czemu nie było konieczności pisania wszystkiego samemu. W aplikacji chodziło o wyświetlanie mapy głębokości punktów, które są w zasięgu kamery Kinecte'a oraz możliwości zapisywania tej mapy w postaci pliku PGM. W skład projektu wchodziły też takie rzeczy jak:

- przygotowanie przewidywanego wykresu Gantta, czyli wykresu w którym przewidujemy ile czasu w przybliżeniu zajmie nam każde zadanie;
- przygotowanie finalnego wykresu Gantta, czyli wykresu z rzeczywistym czasem jaki został poświęcony na wykonanie każdego zadania;
- przygotowanie repozytorium na serwisie GitHub wraz z licencją i plikiem README;
- wygenerowanie dokumentacji kodu przy pomocy narzędzia Doxygen;
- napisanie dokumentu Post-mortem, czyli co poszło dobrze a co nie;
- napisanie dokumentacji całego projektu;

2. Wykresy Gantta

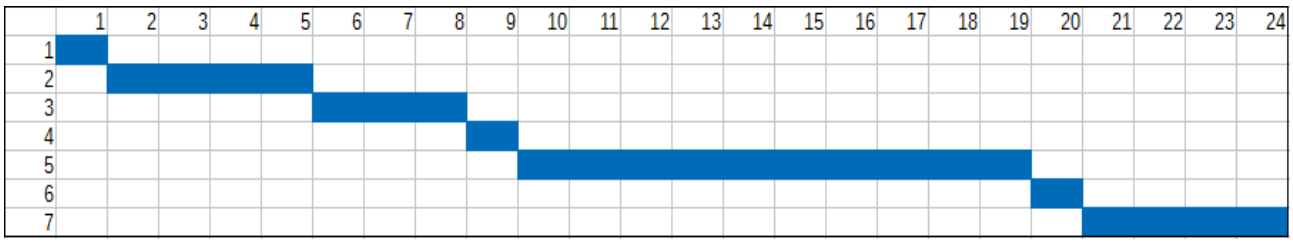
2.1 Przewidywany wykres Gantta

Projekt został podzielony na 7 zadań. Suma czasów potrzebnych na zrealizowanie każdego zadania wyniosła 24 godziny. Tabela z zadaniami została przedstawiona poniżej i przedstawia nazwę zadania, przewidywany czas realizacji oraz początek i zakończenie zadania.

	Zadanie	Czas [h]	Start	Koniec
1	Tworzenie wykresu Gantta	1	0	1
2	Szukanie przydatnych informacji	4	1	5
3	Instalowanie potrzebnego oprogramowania	3	5	8
4	Tworzenie projektu GitHub	1	8	9
5	Implementacja aplikacji	10	9	19
6	Generowanie dokumentacji aplikacji	1	19	20
7	Pisanie dokumentacji całego projektu	4	20	24

Rys 2.1. Tabelka przestawiająca przewidywane zadania

Na podstawie tabelki powstał wykres Gantta, czyli graficzne przedstawienie czasu poszczególnych zadań w stosunku do całego projektu. Wykres znajduje się poniżej.



Rys 2.2. Przewidywany wykres Gantta

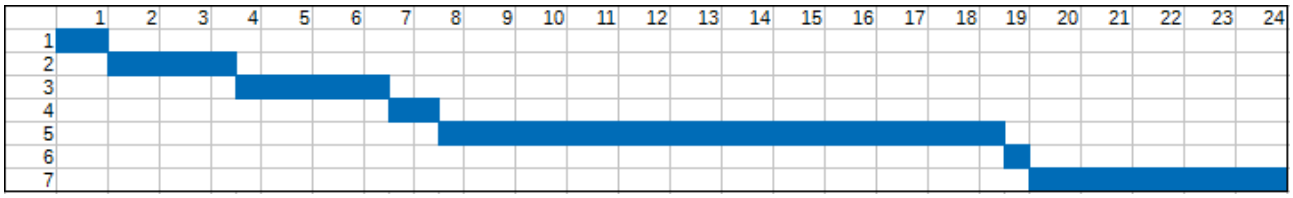
2.2 Dokładny wykres Gantta

Po zakończeniu projektu został przygotowany dodatkowy wykres Gantta który dokładnie obrazuje czas przeznaczony na poszczególne zadanie. Różnica pomiędzy zakładanym czasem potrzebnym na realizacje zadania, a dokładnym czasem jest zależna od zadnia. Niektóre zadania zostały dokładnie przewidziane, lecz zdarzyło się się, że różnica wyniosła 1,5 godziny. Jednak w większości przypadków różnica nie przekracza 30min. Tabelkę z zadaniami i dokładnymi czasami potrzebnymi czasami potrzebnymi na ich realizację można zaobserwować poniżej.

	Zadanie	Czas [h]	Start	Koniec
1	Tworzenie wykresu Gantta	1	0	1
2	Szukanie przydatnych informacji	2,5	1	3,5
3	Instalowanie potrzebnego oprogramowani	3	3,5	6,5
4	Tworzenie projektu GitHub	1	6,5	7,5
5	Implementacja aplikacji	11	7,5	18,5
6	Generowanie dokumentacji aplikacji	0,5	18,5	19
7	Pisanie dokumentacji całego projektu	5	19	24

Rys 2.3. Tabelka przedstawiająca zadania z dokładnymi czasami

Na podstawie tabelki powstał wykres Gantta, który można zaobserwować poniżej.



Rys 2.4. Dokładny wykres Gantta

3. Implementacja aplikacji

Implementacja aplikacji została podzielona na 3 części:

- szukanie punktów znajdujących się w pobliżu Kinecte'a;
- rysowanie w czasie rzeczywistym mapy głębi odległości punktów od Kinecte'a;
- zapis mapy głębi do pliku PGM;

3.1 Szukanie puntów modelu 3D

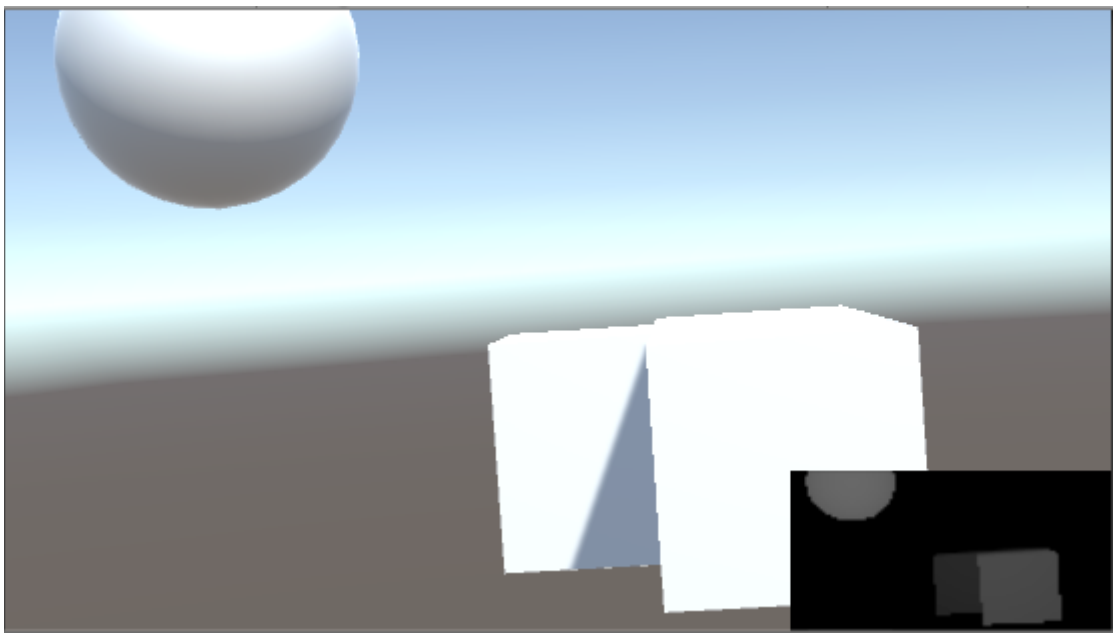
Podczas szukania punktów zastosowana została technika Raycastingu. Gdy promień wyprowadzony z kamery napotkał jakiś obiekt, do tablicy została zapisana odległość punktu do płaszczyzny kamery. Jeżeli promień nie napotkał żadnego obiektu na swojej drodze to do tablicy wstawiano maksymalną odległość jaka była w zasięgu kamery. Odległości normalizowane są do przedziału od 0 do 65535. Zasada działania zapisywania odległości została przedstawiona na poniższym rysunku.

Rys 3.1. Zasada działania pozyskiwania odległości

Algorytm polega na przeglądaniu widoku kamery co x-pikseli w pętlach „for”. Dla każdej iteracji wysyłany jest promień. Jeżeli promień zderzy się z obiektem to zapisywana jest jego odległość do płaszczyzny kamery Kinect w tablicy dwuwymiarowej.

3.2 Generowanie mapy głębi

Podczas działania aplikacji, w każdej klatce generowana jest mapa głębi wyświetlana w czasie rzeczywistym w dolnym prawym rogu okna. Mapa głębi jest monochromatyczna. Wartości od 0 do 65535 określające odległość do punku normalizowane są do przedziału od 0 do 1. Biały kolor oznacza, że punkt znajduje się bardzo blisko kamery a czarny, że obiekt jest daleko kamery. W kodzie wykorzystana do tego celu została funkcja „Lerp”. Na podstawie otrzymanych wartości generowana jest tekstura 2D, a później podstawiana jest ona do obrazu wyświetlanego w oknie widoku. Przykładowy widok z kamery Kinecte’a wraz z mapą głębi znajduję się na rysunku poniżej.



Rys 3.2. Przykładowy widok z okna mapy głębi

3.3 Zapis mapy głębi

Mapę głębi generowaną w każdej klatce użytkownik może zapisać do formatu PGM. Format PGM pozwala na zapisywanie obrazów monochromatycznych. Składa się on z nagłówka „P2”. Następnie określana jest szerokość oraz wysokość obrazu w pikselach. Kolejnym składnikiem jest maksymalna wartość koloru, która wynosi 65535. Następnie generowana jest tablica wartości pikseli. Każdy atrybut oddzielony jest od siebie znakiem spacji, tabulacji bądź nowej linii.

Użytkownik naciskając znak „I” na klawiaturze zapisuję bieżącą mapę głębi do katalogu, którego nazwą jest „dzien-miesiąc-rok”, do pliku o nazwie „PGM – godzina-minuta-sekunda-milisekunda”.

4. Podsumowanie

Podczas projektu stworzono aplikację, dzięki której możliwe jest symulowanie kamerą Kinect oraz zapis wyników symulacji. Aplikacja została napisana w języku C# wykorzystując silnik graficzny Unity. Dzięki projektowi możliwe było zapoznanie się z procesem tworzenia projektów, zarządzanie czasem i tworzeniem dokumentacji kodu.