

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I
I N A U K I N F O R M A C Y J N Y C H



Praca dyplomowa inżynierska

na kierunku Informatyka

Aplikacja wirtualnej rzeczywistości wykorzystująca algorytm
maszerujących sześciianów

Anastasia Khlebous

Wojciech Trześniowski

promotor

dr inż. Paweł Kotowski

WARSZAWA 2019

.....

podpis promotora

.....

podpis autora

Streszczenie

Aplikacja wirtualnej rzeczywistości wykorzystująca algorytm maszerujących sześciianów

W pracy opisano technologię wirtualnej rzeczywistości, jej zastosowanie i przykładowe zestawy VR. Ponadto praca przedstawia zasadę działania algorytmu maszerujących sześciianów służącego do generowania siatek trójkątów, a także możliwe jego modyfikacje i alternatywne algorytmy.

Dokument zawiera opis projektu i implementacji aplikacji VR służącej do tworzenia scen 3D składających się z terenu i ustawionych na nim obiektów. Powierzchnie obiektów i terenu są generowane za pomocą algorytmu maszerujących sześciianów. Aplikacja jest oparta o silnik Unity i jest dedykowana platformie Oculus Rift. Zbadano także podobne oprogramowanie już istniejące na rynku, wyciągnięto wnioski z ukończonego projektu i zaproponowano wskazówki do jego dalszego rozwoju.

Słowa kluczowe: wirtualna rzeczywistość, VR, algorytm maszerujących sześciianów, Unity

Abstract

Marching Cubes Algorithm in Virtual Reality

The work describes virtual reality technology, its application and exemplary VR sets. In addition, the work presents the principle of the marching cubes algorithm used for generating triangular meshes, as well as possible modifications and alternatives algorithms. The document contains a description of the design and implementation of the VR application used for creating 3D scenes consisting of terrain and objects set on it. Terrain and object surfaces are generated using the marching cubes algorithm. The application is based on the Unity Engine and is dedicated to the Oculus Rift platform. Similar already existing on the market software was examined, completed project was concluded and outlines for its further development were suggested.

Keywords: virtual reality, VR, marching cubes algorithm, Unity

Warszawa, dnia

Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisany w pkt 2) pod tytułem „Aplikacja wirtualnej rzeczywistości wykorzystująca algorytm maszerujących sześciianów”, której promotorem jest dr inż. Paweł Kotowski, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

1.	Wstęp	11
2.	Podział pracy	12
2.1.	Dokument	12
2.2.	Aplikacja	12
3.	Wirtualna rzeczywistość	13
3.1.	Definicja	13
3.2.	Obszary zastosowań VR	13
3.3.	Dostępne rozwiązania technologiczne	17
4.	Algorytm maszerujących sześciianów	20
4.1.	Opis działania algorytmu	20
4.2.	Obliczenia równoległe	21
4.3.	Sposób obliczania wartości normalnych	22
4.4.	Algorytm maszerujących czworościanów	23
4.5.	Dual Contouring	23
5.	Aplikacja	24
5.1.	Opis	24
5.2.	Wymagania funkcjonalne	26
5.2.1.	Moduł menu głównego	26
5.2.2.	Moduł edycji sceny	27
5.2.3.	Moduł edycji terenu	29
5.2.4.	Moduł edycji modelu	31
5.3.	Wymagania niefunkcjonalne	32
6.	Dostępne rozwiązania	33
7.	Implementacja	39
7.1.	Technologie	39
7.2.	Architektura	40
7.2.1.	Klasy warstwy interfejsu	41

7.2.2.	Klasy warstwy logiki	42
7.3.	Opis wybranych aspektów	43
7.3.1.	Ikony scen i modeli	43
7.3.2.	Zapisywanie i wczytywanie	44
7.3.3.	Wykorzystanie GPU w obliczeniach	46
7.3.4.	Programowanie reaktywne	47
7.3.5.	Zaznaczanie obiektu	49
8.	Instrukcja użytkownika	50
8.1.	Zasady działania menu	50
8.1.1.	Typy pozycji menu	53
8.2.	Tryb główny	54
8.3.	Tryb edycji sceny	55
8.3.1.	Lewy kontroler	55
8.3.2.	Prawy kontroler	56
8.3.3.	Tryb edycji sceny przy zaznaczonym obiekcie	57
8.4.	Tryb edycji terenu	58
8.4.1.	Lewy kontroler	58
8.4.2.	Prawy kontroler	59
8.5.	Tryb edycji modelu	61
8.5.1.	Lewy kontroler	61
8.5.2.	Prawy kontroler	62
9.	Podsumowanie	65
9.1.	Wnioski	66
9.2.	Dalszy rozwój	67

1. Wstęp

Celem niniejszej pracy jest przedstawienie technologii wirtualnej rzeczywistości i działania algorytmu maszerujących sześciianów. Praca opisuje też projekt i implementację aplikacji opartą o oba omawiane zagadnienia, służącą do tworzenia prostych scen 3D.

Wirtualna rzeczywistość jest dynamicznie rozwijającą się technologią, która zyskuje coraz większą grupę użytkowników. Dzięki intuicyjności użycia i immersyjności doświadczenia platformy tego typu zaczynają być powszechnie stosowane w celach zarówno rozrywkowych jak i przemysłowych. Technologia ta jest wciąż nowatorska, a przez to bardzo zróżnicowana pod względem możliwości i potencjalnych zastosowań. Wciąż udoskonalane zestawy VR znajdują zastosowanie w różnych gałęziach branży informatycznej.

Algorytm maszerujących sześciianów pozwala na generowanie siatek trójkątów dla trójwymiarowych obiektów, które podlegają dynamicznym zmianom. Jest wykorzystywany m.in. do proceduralnego generowania terenu z użyciem GPU. W połączeniu ze sprzętem VR, mającym intuicyjny interfejs do manipulacji elementami w przestrzeni, daje możliwość zastosowania go w aplikacjach do projektowania modeli 3D.

2. Podział pracy

2.1. Dokument

Anastasia Khlebous:

- Wstęp
- Wirtualna rzeczywistość
- Aplikacja
- Instrukcja użytkownika
- Bibliografia
- Spis załączników

Wojciech Trześniowski:

- Algorytm maszerujących sześciianów
- Dostępne rozwiązania
- Implementacja
- Podsumowanie
- Wykaz stosowanych skrótów i oznaczeń
- Spis rysunków

2.2. Aplikacja

Anastasia Khlebous:

- Projekt interfejsu użytkownika
- Implementacja warstwy interfejsu
- Integracja z zestawem Oculus Rift

Wojciech Trześniowski:

- Implementacja warstwy logiki
- Przystosowanie algorytmów do uruchomienia na karcie graficznej

3. Wirtualna rzeczywistość

3.1. Definicja

Pojęcie wirtualnej rzeczywistości obejmuje wszystko co jest trójwymiarowym, generowanym komputerowo, interaktywnym środowiskiem. Użytkownik może wykonywać w nim określone operacje i jest integralną częścią wirtualnego świata[5]. Wirtualna rzeczywistość jest osiągana głównie przez użycie specjalnych gogli i kontrolerów, które mają zapewnić immersyjność stworzonego środowiska. Często świat wirtualny jest kojarzony z branżą gier komputerowych, ale w rzeczywistości ma dużo innych zastosowań.

3.2. Obszary zastosowań VR

Architektura i wizualizacja wnętrz

Architekci mogą wykorzystywać gogle VR do wizualizacji swoich projektów[6]. Takie rozwiązanie łączy dokładność rysunków i łatwość odbioru makiet. Istnieją symulacje komputerowe dla architektów, ale są to głównie narzędzia do projektowania, a nie prezentowania rezultatów. Takie oprogramowanie może co najwyżej pozwolić na wygenerowanie pokazowego filmu, ale w porównaniu do aplikacji VR, tracony jest aspekt interaktywności.

Z aplikacji VR zaczynają korzystać sprzedawcy mieszkań, którzy za pomocą wizualizacji VR przedstawiają klientom przyszłe wnętrza ich mieszkań. Lokal może być wciąż w budowie, ale posiadając projekt można pozwolić przyszłym mieszkańcom na wirtualny spacer. Symulacja może obejmować zmiany projektów wykończenia mieszkania, zmiany w umeblowaniu czy przedstawianie wnętrza i okolic w różnych porach dnia i roku. Dodatkowo wszystko może odbywać się zdalnie.

Medycyna

W medycynie technologia VR może być użyta do wizualizacji organów pacjenta[7]. Dzięki temu lekarze są w stanie stawiać trafniejsze diagnozy np. dotyczące pojawienia się nowotworów. Dodatkowo może to ułatwić zaplanowanie operacji. Początkujący lekarze i studenci medycyny z pomocą sprzętu VR mogą uczyć się poprzez oglądanie lub uczestniczenie w wirtualnych operacjach. Takie podejście pozwala na uczestniczenie w operacjach chirurgicznych do których uprawnieni są tylko doświadczeni lekarze. Ponadto daje to możliwość zdobycia doświadczenia bez narażania zdrowia i życia pacjentów.

Technologia VR jest wykorzystywana w terapiach, które mają na celu walkę z zaburzeniami lękowymi. Pacjent poddawany jest bodźcom, które powodują lek. Jednak ma on poczucie bezpieczeństwa, bo wszystko jest wykonywane w ścisłe kontrolowanych warunkach. Takim podejściem próbuje się leczyć m.in. lęk przed otwartą przestrzenią, zamknięciem czy lataniem. Podobnymi sposobami zwalcza się chroniczne bóle u przewlekłe chorych pacjentów. Przebywanie w wirtualnym świecie skupia uwagę pacjenta i odwraca ją od odczuwanego bólu.



Rysunek 3.1: Sala operacyjna w wirtualnej rzeczywistości - <https://www.3ders.org/articles/20170127-3d-systems-announces-vr-scenarios-for-surgical-training.html>

3.2. OBSZARY ZASTOSOWAŃ VR

Wojskowość

Wojsko korzysta z VR, by prowadzić szkolenia dla żołnierzy[8]. Chociaż jest to świat wirtualny to żołnierz uczy się jak prawidłowo zareagować w niebezpiecznych sytuacjach, a symulacje w wirtualnej rzeczywistości pozwalają na ćwiczenia bez ryzyka dla zdrowia i życia. Najbardziej popularne są symulatory dla pilotów i operatorów pojazdów opancerzonych, które posiadają realistyczne panele sterowania i są wyposażone w skomplikowane systemy (hydrauliczne, elektroniczne) reagujące na ruchy trenującej się osoby. Istnieją systemy dla operatorów na mostkach kapitańskich odwzorowujące działanie systemów sterujących i nawigacyjnych dla jednostek płynących. Takie systemy pozwalają żołnierzom na zdobycie doświadczenia w różnorodnych warunkach i sytuacjach poprzez zastosowanie dowolnie dostosowanych, ale również powtarzalnych scenariuszy szkoleniowych.

Podejmowane są też próby zdalnego sterowania sprzętem wojskowym z użyciem gogli VR. Dzięki takiemu podejściu żołnierz nie jest narażony na niebezpieczeństwo, gdyż zdalnie kontroluje robota, pojazd czy drona. Jednym z przykładów jest robot Taurus, który służy do zdalnego rozbrajania ładunków wybuchowych.



Rysunek 3.2: Żołnierz podczas ćwiczeń ze sprzętem VR - <https://thinkmobiles.com/blog/virtual-reality-military/>

Branża rozrywkowa

Dla branży gier komputerowych technologia wirtualnej rzeczywistości wiąże się z nowymi możliwościami rozwoju, ale i nowymi wyzwaniami[9].

Z jednej strony doświadczenie eksploracji wirtualnego świata jest bardziej efektowne przy wykorzystaniu gogli VR niż myszki i klawiatury. Wykonywane czynności są bardziej naturalne dla użytkownika. Ponadto gracz nie tylko ma wpływ na to co widzi, ale jest bliżej bycia częścią obserwowanego świata.

Ale z drugiej strony programiści musieliby zmierzyć się z problemem poruszania się w VR. Gracz ma ograniczone możliwości poruszania się przez pomieszczenie, w którym się znajduje. Problem lokomocji jest związany z oszukiwaniem ludzkich zmysłów. To co widzi użytkownik, nie zgadza się z tym, co dzieje się z jego ciałem. Takie sytuacje mogą prowadzić do bólu głowy i nudności. W wielu grach powyższe problemy są rozwiązywane przez zastosowanie teleportacji, ale są też bardziej zaawansowane i wymagające innego sprzętu rozwiązania. Jednym z nich jest bieżnia do gogli VR Virtuix Omni. Trzeba też pamiętać, że sprzęt VR jest kosztowny, przez co często też niedostępny dla wielu graczy. Z tego powodu powstaje coraz więcej salonów gier VR.



Rysunek 3.3: Bieżnia do gogli VR Virtuix Omni - <http://www.virtuix.com/virtuix-launches-omniverse-esports-competitive-vr-gaming-platform/>

Liczba dostępnych gier na urządzenia VR stale rośnie, ale przez zróżnicowaną jakość, wywołuje

3.3. DOSTĘPNE ROZWIĄZANIA TECHNOLOGICZNE

to mieszane uczucia wśród graczy. Wiele gatunków gier może być z powodzeniem przeniesiona do VR. Przykładem mogą być gry wyścigowe, chociaż wymaga to zastosowania dodatkowego sprzętu. Jednak coraz więcej gier zostaje przystosowanych pod zestawy VR tylko po to, by skorzystać na wzroście popularności wirtualnej rzeczywistości. Gry ze względu na gatunek czy rozgrywkę mogą się do tego nie nadawać, a VR może narzucić na nie ograniczenia. Przykładem jest gra "The Elder Scrolls V: Skyrim VR", która spotkała się z krytyką graczy przez uciążliwe poruszanie i manipulację przedmiotami oraz ograniczenia w jakości grafiki.



Rysunek 3.4: Zestaw do symulowania prowadzenia samochodu przystosowany do sprzętu VR - <https://www.roadtovr.com/r-craft-vr-motion-simulator-oculus-rift-dirt-rally-sim-racing/>

3.3. Dostępne rozwiązania technologiczne

Projekt Google Cardboard jest jednym z najłatwiej dostępnych zestawów. Pomyślem na gogle jest wykorzystanie możliwości telefonu i minimalistycznej konstrukcji, co pozwala na znacznie zmniejszenie kosztu do średnio kilkunastu dolarów. Niski koszt jest największą zaletą, ale prostota produktu wiąże się z ograniczeniami. Jakość wyświetlonego obrazu jest uzależniona od mocy obliczeniowej użytego telefonu, która jest mniejsza w porównaniu z komputerem. Używanie gogli może sprawiać problemy przez brak regulacji soczewek i konieczność trzymania urządzenia w ręku.



Rysunek 3.5: Zestaw Google Cardboard - https://lh3.googleusercontent.com/MAwTQ5qs2ogQg_NSQBQTh3nrG78sMRALh9MGWzvlB5-t63NLtQyI3HBJttOL9Owx5fcE

Samsung Gear VR, podobnie jak Google Cardboard, wykorzystuje możliwości telefonu, ale nie ogranicza się do nich. W przeciwieństwie do Google Cardboard korzystającego z akcelerometru i żyroskopu smartfona, Samsung Gear VR wykorzystuje czujniki wbudowane w gogle do śledzenia ruchów głowy. Dodatkowo gogle posiadają przyciski, touchpad i możliwość dotosowania odległości soczewek. Zestaw jest przystosowany do telefonów firmy Samsung i istnieje możliwość wybrania wersji zawierającej kontroler.



Rysunek 3.6: Zestaw Samsung Gear VR - <https://www.amazon.com/Samsung-Controller-SM-R325-International-Version/dp/B07GMC1C13>

Zestaw HTC Vive, w porównaniu do wyżej wymienionych rozwiązań, nie jest samodzielnym urządzeniem i wymaga wykorzystania komputera. Wysokie wymagania systemowe pozwalają na zapewnienie wysokiej jakości obrazu i płynnego działania. Urządzenie wykorzystuje różnorodne

3.3. DOSTĘPNE ROZWIĄZANIA TECHNOLOGICZNE

czujniki, w szczególności: żyroskop MEMS, akcelerometr i czujniki pozycjonowania laserowego. Do śledzenia pozycji gogli i zawartych w zestawie dwóch kontrolerów używane są stacje Lighthouses. Są one w stanie objąć przestrzeń o wymiarach 4,5m na 4,5m. Jednak wysoka jakość pociąga za sobą wysoką cenę zestawu.



Rysunek 3.7: Zestaw HTC Vive - <http://home.mehromah.ir/mehromah-learn/scientific-technology/4204-HTC-will-play.html>

Zestaw Oculus Rift, tak jak zestaw HTC Vive, wymaga wykorzystania komputera. Posiada podobną specyfikację i wymagania systemowe, ale zestaw zawiera wbudowane słuchawki, kontrolery są lżejsze, a ich użycie bardziej intuicyjne. Zestaw umożliwia łatwy dostęp do aplikacji z platformy Oculus i SteamVR. Z opini użytkowników wynika, że Oculus Rift jest prosty w instalacji, a gogle są wygodniejsze w porównaniu do innych dostępnych zestawów.



Rysunek 3.8: Zestaw Oculus Rift - <https://www.currys.co.uk/gbuk/tv-and-home-entertainment/gaming/virtual-reality/oculus-rift-touch-bundle-10168251-pdt.html>

4. Algorytm maszerujących sześciianów

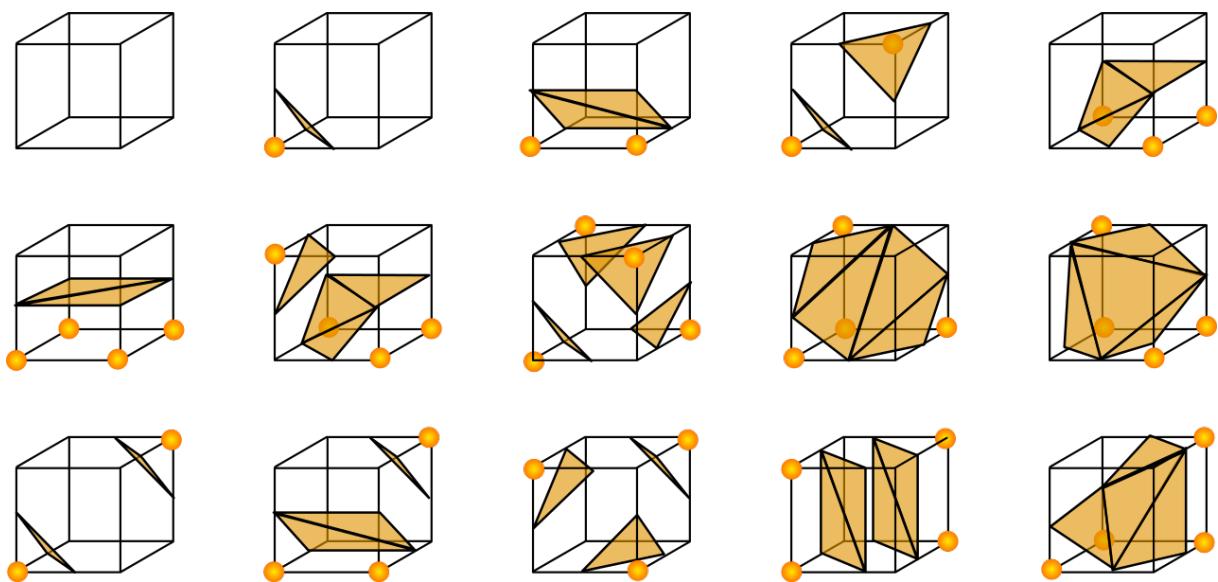
Algorytm maszerujących sześciianów służy do generowania siatki trójkątów[1]. Jest on wykorzystywany w sytuacji, gdy dostępne są tylko próbkoowe dane pożądanego obiektu lub gdy celowo zredukowano obiekt do próbek (np. gdy obiekt ma podlegać dynamicznym zmianom).

4.1. Opis działania algorytmu

Ograniczona przestrzeń zostaje podzielona na równej wielkości sześciiany tworzące regularną, trójwymiarową siatkę punktów. Każdy punkt ma przypisaną wartość, która mówi czy znajduje się on wewnątrz czy na zewnątrz obiektu i jak daleko znajduje się od jego powierzchni. Wartość większa od ustalonego progu oznacza, że punkt jest zawarty w obiekcie, a jeśli jest mniejsza to punkt jest poza obiektem. Z każdego sześciianu zostaje wyznaczony zbiór ścian, a suma zbiorów wszystkich ścian ze wszystkich sześciianów składa się na wynikowy model.

W celu wyznaczenia zbioru ścian rozpatrujemy każdy sześciian osobno. Wierzchołki sześciianu są ponumerowane. Za pomocą wartości w każdej parze wierzchołków znajdowane są wszystkie punkty, gdzie model powinien przecinać krawędź. Następnie konstruowane są ściany, korzystając z pomocniczej tablicy, która mówi jak wygląda generowany zbiór ścian dla danego zestawu przecięć (jaka jest kolejność tych wierzchołków). Każdy z 8 wierzchołków może być w obiekcie lub nie, dlatego istnieje 2^8 , czyli 256, możliwych rodzajów zbiorów ścian, z których możemy wyróżnić 15 kanonicznych orientacji (Rysunek 4.1), które powtarzają się jako odbicia symetryczne. Zbiór ścian zawiera co najwyżej 5 trójkątów składających się na co najwyżej 4 ściany.

4.2. OBLICZENIA RÓWNOLEGŁE



Rysunek 4.1: Kanoniczne orientacje generowanego zestawu ścian - https://en.wikipedia.org/wiki/Marching_cubes

4.2. Obliczenia równoległe

Aby uzyskać dokładniejsze odwzorowanie modelu, należy zwiększyć gęstość sześcianów w przestrzeni. Prowadzi to do zwiększenia rozmiaru danych, a zatem wymaga więcej mocy obliczeniowej. Jeśli nie ma możliwości jej zwiększenia, to aby poprawić dokładność bez zwiększenia czasu tworzenia modelu, można zrównoleglić algorytm.

Algorytm maszerujących sześcianów może być w przystosowany do obliczeń równoległych[2]. Operacje w ramach sześcianów są od siebie niezależne, dlatego każdy z nich może być obliczany przez inny wątek. Nakładają się jedynie odczyty wartości punktów dla sześcianów dzielących wspólne wierzchołki. Jednak nie ma to wpływu na poprawność, gdyż wartości te nie zmieniają się w czasie pojedynczego obliczenia. Wspólne odczyty nie mają znaczącego wpływu na wydajność, gdyż jeden punkt odczyta maksymalnie 8 sześcianów, czyli liczba odczytów jest ograniczona stałą i nie zależy od rozmiaru danych.

Do obliczeń równoległych można wykorzystać procesor graficzny. Takie podejście jest możliwe tylko przy zastosowaniu podstawowych typów danych o statycznym rozmiarze. Przy zapisie wyniku mogą pojawić się konflikty, które mogą być wykluczone przez kosztowne czasowo wzajemne blokowanie się wątków. Aby tego uniknąć należy użyć struktury danych, która pozwoli na jednoznaczne przypisanie obszaru pamięci do każdego z sześcianów, kosztem większego zużycia pamięci, tzn. w większości przypadków znaczna część pamięci zostanie niewykorzystana. Jest

to możliwe, gdyż wiadomo, że liczba ścian dla każdego z sześcielanów jest ograniczona, a zatem wymagany rozmiar pamięci dla wyniku jest ograniczony.

W aplikacji zastosowano zrównoległą wersję algorytmu, przystosowaną do uruchomienia na procesorze graficznym.

4.3. Sposób obliczania wartości normalnych

Konieczne jest przypisanie wierzchołkom wynikowych ścian wartości normalnych. Jeśli danymi wejściowymi są wartości w punktach siatki sześcielanów, to nie wyznaczają one jednoznacznie żadnego kształtu. Są jedynie uproszczeniem i odpowiadają pewnemu zbiorowi modeli. Dlatego nie ma jednoznacznie dobrej metody na wyznaczenie wartości normalnych.

Rozważano trzy metody wyznaczania normalnych. Ich wspólną cechą jest potrzeba zapisu wynikowej ściany w taki sposób, aby wiedzieć w której stronie jest ona zwrócona, tzn. po której stronie jest wnętrze modelowanego obiektu, a po której pusta przestrzeń.

1. Najprostszą metodą jest obliczanie normalnych niezależnie od innych ścian. Za pomocą wierzchołków ścian zostaje wyznaczona normalna do płaszczyzny, w której zawarta jest ściana. Następnie wyznaczona normalna jest przypisywana do każdego z wierzchołków trójkątów, wchodzących w skład ściany.
2. Istnieje możliwość uzyskania gładszego kształtu nie zwiększaając złożoności obliczeniowej (pomijając interpolację). Początkowo dla każdego punktu z wejściowej siatki algorytmu wyznaczono normalną. W tym celu obliczano średnią wartość sąsiednich punktów dla każdej współrzędnej. Otrzymany wektor jest normalną dla danego punktu. Każdy wierzchołek trójkąta wynikowej ściany znajduje się na krawędzi sześcielnu. Do wierzchołka zostaje przypisana normalna o wartości średniej normalnych z końców krawędzi. Popularnym rozwiązaniem jest użycie średniej ważonej, gdzie wagą jest odległość końca krawędzi od wierzchołka.
3. Do uzyskania gładzych kształtów, przy obliczaniu normalnej dla danego wierzchołka, można też wykorzystać dane o ścianach dzielących ten wierzchołek. Wtedy normalną jest średnia z normalnych tych ścian obliczanych pierwszą metodą. Popularnym rozwiązaniem jest użycie średniej ważonej, gdzie wagą jest pole powierzchni ściany, co pozwala uzależnić wpływ ściany od jej wielkości. Zastosowanie tej metody jest bardziej wymagające obliczeniowo, ponieważ wymaga wyszukiwania ścian dzielących dany wierzchołek.

4.4. ALGORYTM MASZERUJĄCYCH CZWOROŚCIANÓW

W metodzie drugiej i trzeciej normalne w wierzchołkach jednego trójkąta są różne, więc dla reszty punktów trójkąta normalne są wynikiem interpolacji wartości normalnych z wierzchołków.

W aplikacji do obliczania wartości normalnych zastosowano trzecią metodę.

4.4. Algorytm maszerujących czworościanów

Alternatywnym sposobem podziału przestrzeni jest podział na czworościany. Przestrzeń jest wstępnie dzielona na sześciany, po czym każdy z sześciianów jest podzielony na 6 nieregularnych czworościanów. Dalej algorytm przebiega na podobnej zasadzie.

Zastosowanie czworościanów pozwala na nieznaczne zwiększenie dokładności, a przez mniejszą ilość wierzchołków mniejsza jest też ilość możliwości wygenerowanych ścian, a tabele przypadków są mniejsze.

Jednak zmodyfikowany algorytm jest bardziej skomplikowany przez co mniej intuicyjny, trudniejszy w implementacji, podatny na błędy wynikające z konieczności ustalenia orientacji brył i znacznie zwiększa zużycie pamięci. Z wymienionych wyżej powodów nie zastosowano go w aplikacji.

4.5. Dual Contouring

Innym sposobem na renderowanie obiektu jest wykorzystanie algorytmu Dual Contouring[3]. Polega on na znalezieniu punktów wewnętrz sześciianu, a nie na jego krawędziach. Jest tu znacznie mniejsza liczba przypadków do rozpatrzenia.

Z drugiej strony algorytm jest trudniejszy do przystosowania do obliczeń równoległych, gdyż poszczególne sześciany korzystają nawzajem ze swoich pośrednich wyników obliczeń. Nie mamy też pewności, że obliczony punkt znajduje się wewnątrz danego sześciianu, przez co problematyczne staje się wyznaczenie ścian i nie mamy gwarancji, że powstałe ściany nie będą się przecinać. Dlatego algorytm nie został on użyty w aplikacji.

5. Aplikacja

5.1. Opis

Dostępnych jest coraz więcej aplikacji VR dedykowanych zarówno początkującym jak i profesjonalnym grafikom i projektantom. Jest to próba wykorzystania nowego rodzaju platformy do zwiększenia wydajności ich pracy.

Programy komputerowe korzystające z klawiatury i myszy muszą posiadać skomplikowane narzędzia, aby dać użytkownikowi możliwość modyfikacji w trzech wymiarach, czyli przekształcić ruch myszy na płaszczyźnie na zmiany w trójwymiarowej wirtualnej scenie. Ułatwić ma to zastosowanie kontrolerów VR, które poruszają się w trzech wymiarach eliminując potrzebę implementacji i użycia podobnych rozwiązań. Dzięki temu zmniejszony zostaje próg wejścia dla użytkownika, tzn. nie musi on poznawać zasad działania dostępnych narzędzi, ponieważ użycie kontrolerów jest intuicyjne i dużo bardziej naturalne. Dla profesjonalistów zmiana interfejsu może pozwolić na skrócenie czasu potrzebnego na przygotowanie modeli.

Wykonana aplikacja służy do tworzenia scen w wirtualnej rzeczywistości. Pozwala użytkownikowi na kształtowanie terenu, tworzenie modeli i ustawianie ich na scenie. Kształt modeli i terenu jest obliczany za pomocą algorytmu maszerujących sześciąników. Aplikacja może być wykorzystana m.in. do tworzenia wirtualnych makiet, planów ogrodów czy scen do gier komputerowych.

Ponadto zestawy VR są coraz bardziej popularne, dlatego stworzoną aplikację można wykorzystywać jako grę typu piaskownica (ang. sandbox), czyli grę komputerową, która nie ma określonego celu, ale pozwala na samodzielne jego określenie dostarczając różnych narzędzi i możliwych operacji do wykonywania.

W aplikacji zostały wyróżnione 4 moduły:

1. Moduł menu głównego pozwala na stworzenie nowej sceny lub wczytanie istniejącej.
2. Moduł edycji modelu umożliwia tworzenie i edycję modeli. Model jest bryłą o kształcie i kolorze nadanym przez użytkownika. Modyfikacje są przeprowadzane za pomocą trzymanego

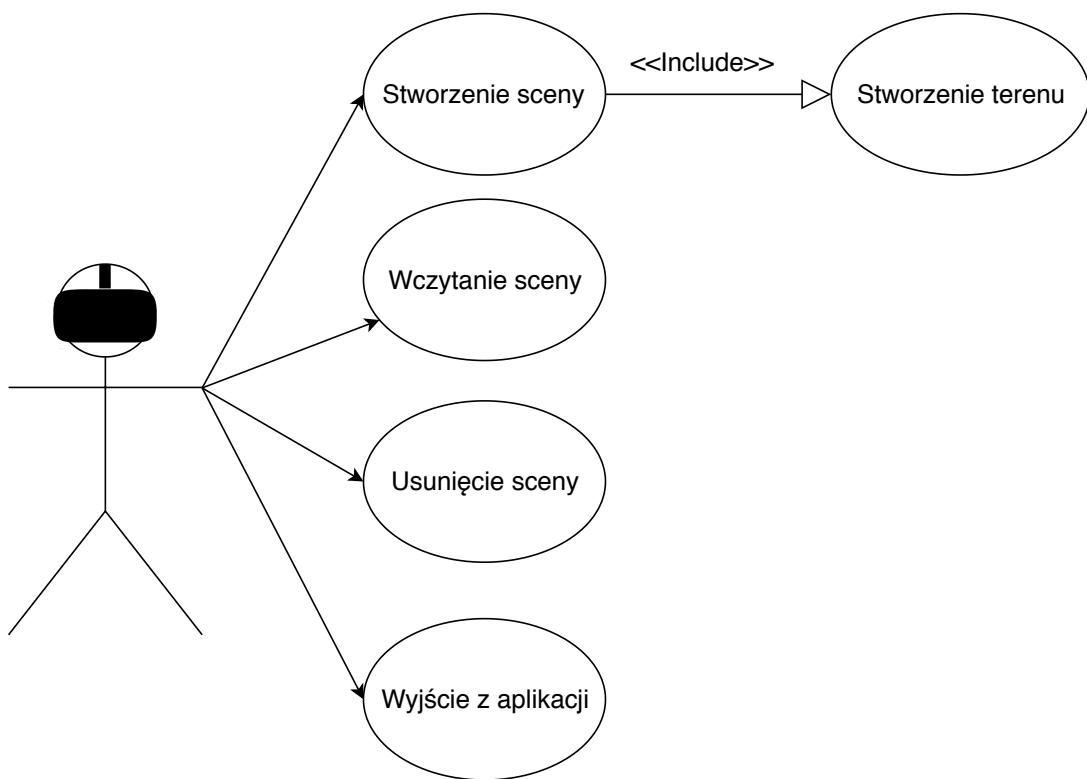
5.1. OPIS

przez użytkownika pędzla i polegają na wykonaniu zmian w jego zakresie na edytowanym obiekcie. Wynik modyfikacji jest uzależniony od wykonywanej przez użytkownika akcji, położenia pędzla i jego parametrów, do których zaliczają się m.in. rozmiar, kolor czy kształt. Gotowy model jest zapisywany i dostępny w module edycji sceny z listy modeli.

3. Moduł edycji terenu jest odpowiednikiem modułu edycji modelu dla terenu z różnicami wynikającymi ze specyfiki samego terenu. Scena zawiera dokładnie jeden teren, dlatego nie ma potrzeby posiadania listy terenów. Nie można zmieniać długości i szerokości, a tylko wysokość terenu co zmienia sposób działania pędzla, ale pozwala na wprowadzenie dodatkowego trybu jego działania.
4. Moduł edycji sceny pozwala na rozmieszczanie obiektów na przygotowanym wcześniej terenie. Obiekt jest kopią modelu, która ma na scenie określoną pozycję, rotację i wielkość. Ustawione obiekty można potem zaznaczać, aby zmienić wymienione parametry. Stworzony model można ponownie edytować, a wszystkie zmiany są aplikowane na powiązane z nim obiekty.

5.2. Wymagania funkcjonalne

5.2.1. Moduł menu głównego

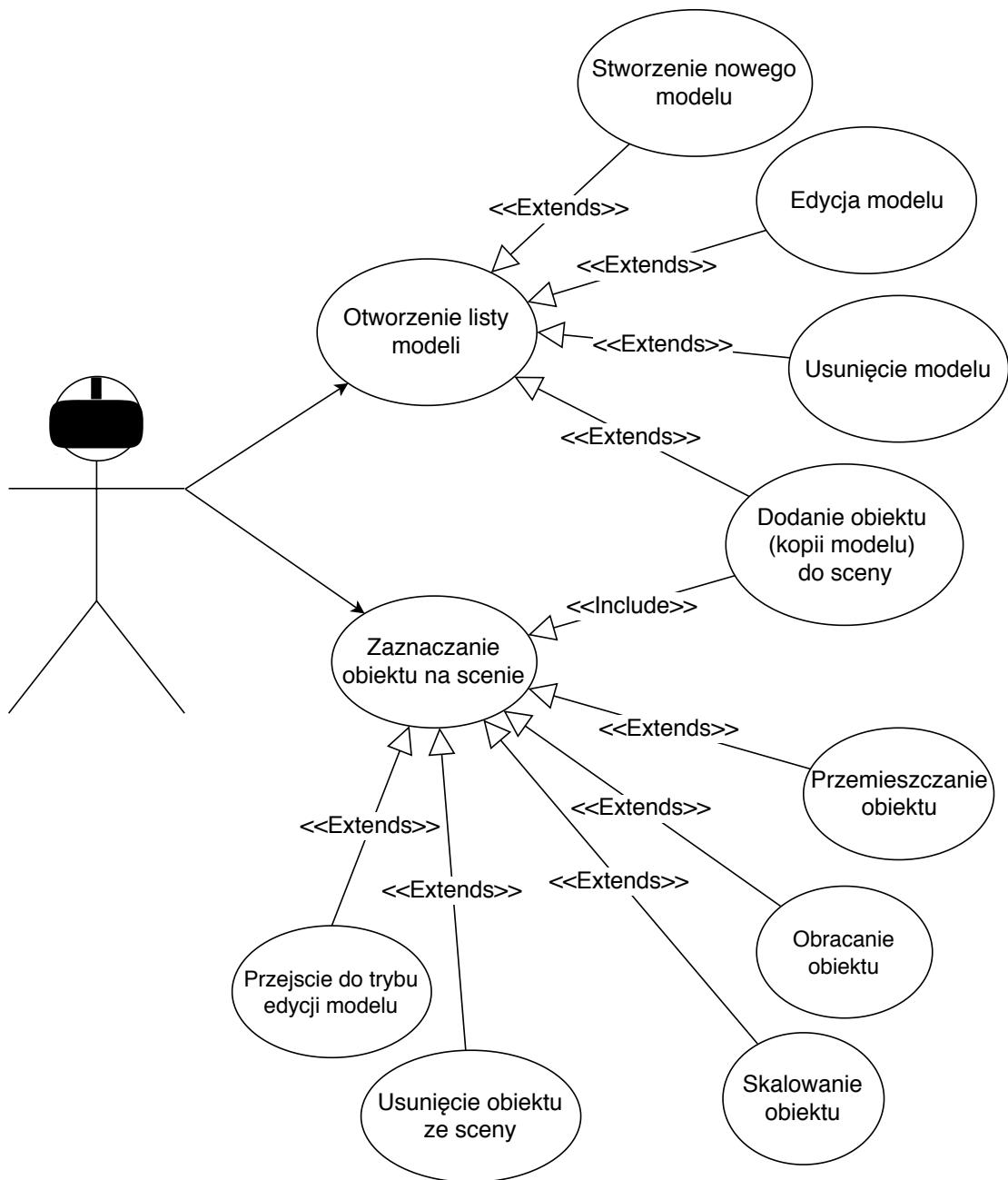


Rysunek 5.1: Diagram przypadków użycia modułu menu głównego

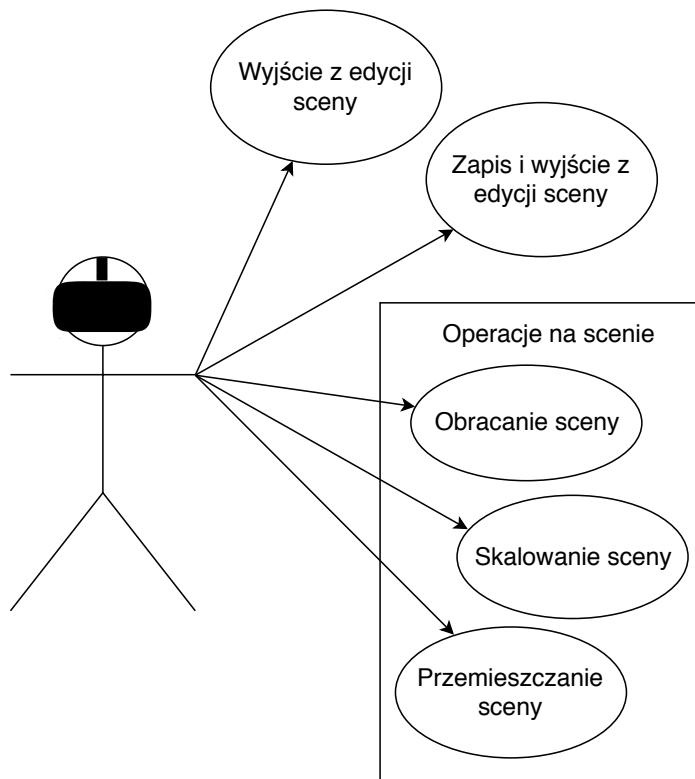
Wymagania funkcjonalne dla modułu menu głównego:

- Stworzenie sceny - przy tworzeniu sceny, tworzony jest również teren
- Wczytanie sceny
- Usunięcie sceny
- Wyjście z aplikacji

5.2.2. Moduł edycji sceny



Rysunek 5.2: Diagram przypadków użycia modułu edycji sceny, część 1



Rysunek 5.3: Diagram przypadków użycia modułu edycji sceny, część 2

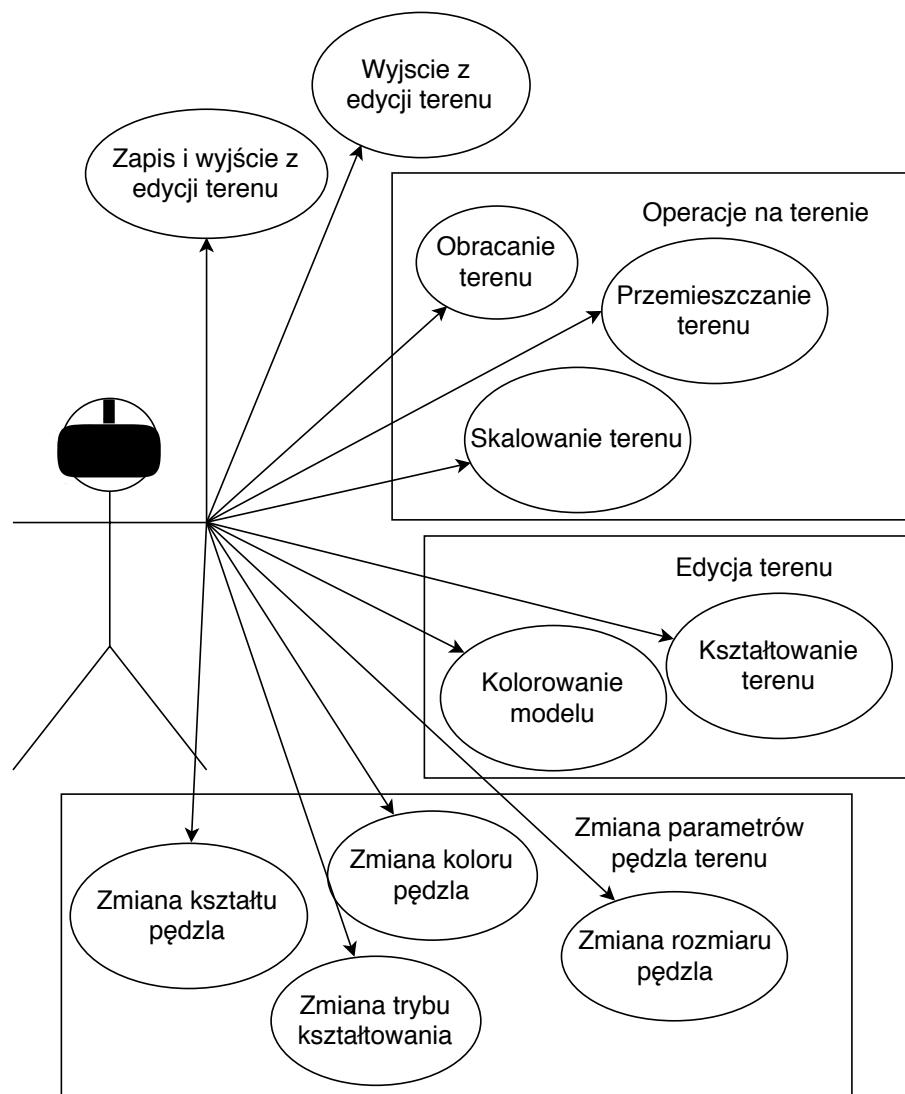
Wymagania funkcjonalne dla modułu edycji sceny:

- Wyjście z edycji sceny
- Zapis i wyjście z edycji sceny
- Operacje na scenie:
 - Obracanie sceny
 - Skalowanie sceny
 - Przemieszczanie sceny
- Stworzenie nowego modelu - powoduje również przejście do trybu edycji modelu
- Edycja modelu - powoduje również przejście do trybu edycji modelu
- Usunięcie modelu - powoduje również usunięcie obiektów tego modelu ze sceny
- Dodanie obiektu (kopii modelu) do sceny
- Zaznaczanie obiektu na scenie - zaznaczenie obiektu polega na wciśnięciu przycisku, gdy dany obiekt jest pierwszą przeszkodą na drodze promienia wychodzącego z kontrolera

5.2. WYMAGANIA FUNKCJONALNE

- Operacje na zaznaczonym obiekcie:
 - Obracanie obiektu
 - Skalowanie obiektu
 - Przemieszczanie obiektu
 - Przejście do trybu edycji modelu
 - Usunięcie obiektu ze sceny
 - Zakończenie edycji modelu

5.2.3. Moduł edycji terenu

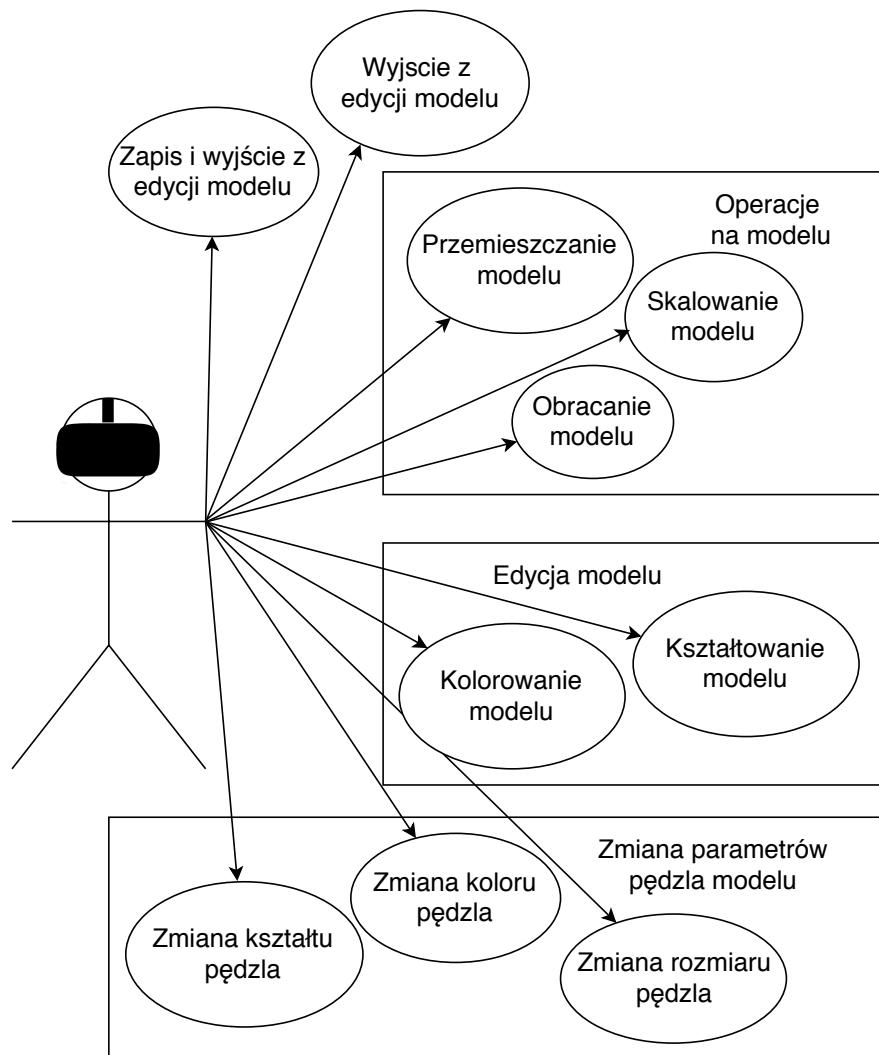


Rysunek 5.4: Diagram przypadków użycia modułu edycji terenu

Wymagania funkcjonalne dla modułu edycji terenu:

- Wyjście z edycji terenu
- Zapis i wyjście z edycji terenu
- Edycja terenu (wykonywane w granicach rzutu pędzla terenu):
 - Kolorowanie terenu
 - Kształtowanie terenu - kształtowanie terenu posiada dwa tryby:
 - * podnoszenie/obniżanie wszystkich punktów
 - * podnoszenie/obniżanie tylko punktów leżących najniżej/najwyżej
- Operacje na terenie:
 - Obracanie terenu
 - Skalowanie terenu
 - Przesuwanie terenu
- Zmiana parametrów pędzla terenu:
 - Zmiana koloru pędzla
 - Zmiana rozmiaru pędzla
 - Zmiana kształtu pędzla - dostępne są dwa kształty: koło i prostokąt

5.2.4. Moduł edycji modelu



Rysunek 5.5: Diagram przypadków użycia modułu edycji modelu

Wymagania funkcjonalne dla modułu edycji modelu:

- Wyjście z edycji modelu
- Zapis i wyjście z edycji modelu
- Wyjście z edycji modelu - powrót do trybu edycji sceny
- Operacje na modelu:
 - Obracanie, skalowanie, przemieszczanie modelu modelu
- Edycja modelu (wykonywane w granicach pędzla modelu):
 - Zmiana kształtu, koloru, rozmiaru i parametrów pędzla modelu

- Kolorowanie modelu
- Kształtowanie modelu
- Zmiana parametrów pędzla modelu:
 - Zmiana koloru pędzla
 - Zmiana rozmiaru pędzla
 - Zmiana kształtu pędzla - dostępne są dwa kształty: sfera i prostopadłościan

5.3. Wymagania niefunkcjonalne

Niniejszy rozdział zawiera listę wymagań niefunkcjonalnych przygotowanych zgodnie z metodą FURPS (ang. Functionality, Usability, Reliability, Performance, Supportability - Funkcjonalność, Używalność, Niezawodność, Wydajność, Wsparcie)

1. Używalność

- Aplikacja jest w języku angielskim.

2. Niezawodność

- dokładność bryły - bryła jest tworzona z dokładnością 1.5cm dla wymiarów nie przekraczających $1\text{m} \times 1\text{m} \times 1\text{m}$. Dokładność jest proporcjonalna dla większych brył.

3. Wydajność

- Wymagania sprzętowe
 - Karta graficzna: NVIDIA GTX 970 / AMD R9 290 lub lepsza
 - Procesor: Intel i5-4590 lub lepszy
 - Pamięć: 8GB+ RAM
 - Dysk twardy: wolne 2GB+
 - System operacyjny: Windows 7 SP1 64 bit lub nowszy
- Aplikacja działa w minimum 50 klatkach na sekundę.

4. Wsparcie

- Aplikacja jest przystosowana do zestawu Oculus Rift.

6. Dostępne rozwiązania

Na rynku istnieją aplikacje posiadające podobne funkcjonalności. Pojawia się coraz więcej narzędzi dedykowanych platformom VR dla projektantów i artystów. Wiele z nich to projekty na etapie prototypu lub udostępnione w modelu wczesnego dostępu. Jednak od czasu rozpoczęcia projektu, którego dotyczy niniejszy dokument, niektóre aplikacje znacznie się rozwinęły i przeszły z fazy eksperymentu do pełnoprawnego produktu.

Mniej rozbudowane aplikacje, rozwijane przez mniejsze zespoły, są często publikowane jako gry na platformie Steam. Jednak większe firmy takie jak Google czy Facebook także zaczęły rozpowszechniać swoje produkty przez różne platformy z tym że nie ograniczają się do Steam. Do najbardziej popularnych należą także Oculus Experiences, VIVEPORT i Windows Mixed Reality.

Istniejące produkty możemy podzielić na dwie kategorie. Aplikacje do tworzenia modeli i aplikacje do rysowania w 3D. Niektóre zawierają funkcjonalności z obu kategorii.

Dostępne są różne gogle i kontrolery VR, które nie są kompatybilne ze sobą. Niektóre programy mogą mieć więcej narzędzi, ograniczając się do niewielu zestawów, ale za to w pełni wykorzystując ich możliwości. Inne mają ograniczone funkcjonalności na rzecz dotarcia do jak największego grona użytkowników. Rynek urządzeń i aplikacji VR nie jest jeszcze dostatecznie ustandardyzowany, dlatego różne urządzenia mają różne funkcjonalności, a powstałe dla nich aplikacje dostarczają inne zestawy możliwych operacji i inne listy narzędzi dla użytkownika. Różnice widać na przykładzie możliwych modyfikacji kształtu obiektów. W zależności od aplikacji dostępne jest przemieszczanie i zmiana kształtu ścian modelu, zmiana położenia wierzchołków, dodawanie i usuwanie wirtualnej gliny, dodawanie i usuwanie figur płaskich lub podzbiór wymienionych operacji.

Oprócz zagadnień dotyczących wirtualnej rzeczywistości aplikacje są wyposażane w standardowe funkcjonalności dla programów do modelowania i renderowania obrazów czy tworzenia animacji. Zalicza się do nich m.in. dostęp do globalnej bazy modeli, integracja z innym oprogramowaniem, możliwość tworzenia wtyczek, zapisywanie wyników pracy do popularnych formatów.

Dzięki temu aplikacje tego typu zaczęły być często wykorzystywane do tworzenia modeli do drukarek 3D.

Medium

Medium jest aplikacją firmy Oculus i służy do tworzenia skomplikowanych modeli 3D. Ma służyć profesjonalnym artystom i zwiększyć wydajność ich pracy w stosunku do pracy z wykorzystaniem aplikacji desktopowych.



Rysunek 6.1: Medium - <https://vrproject.com.pl/oculus-medium/medim>

Blocks

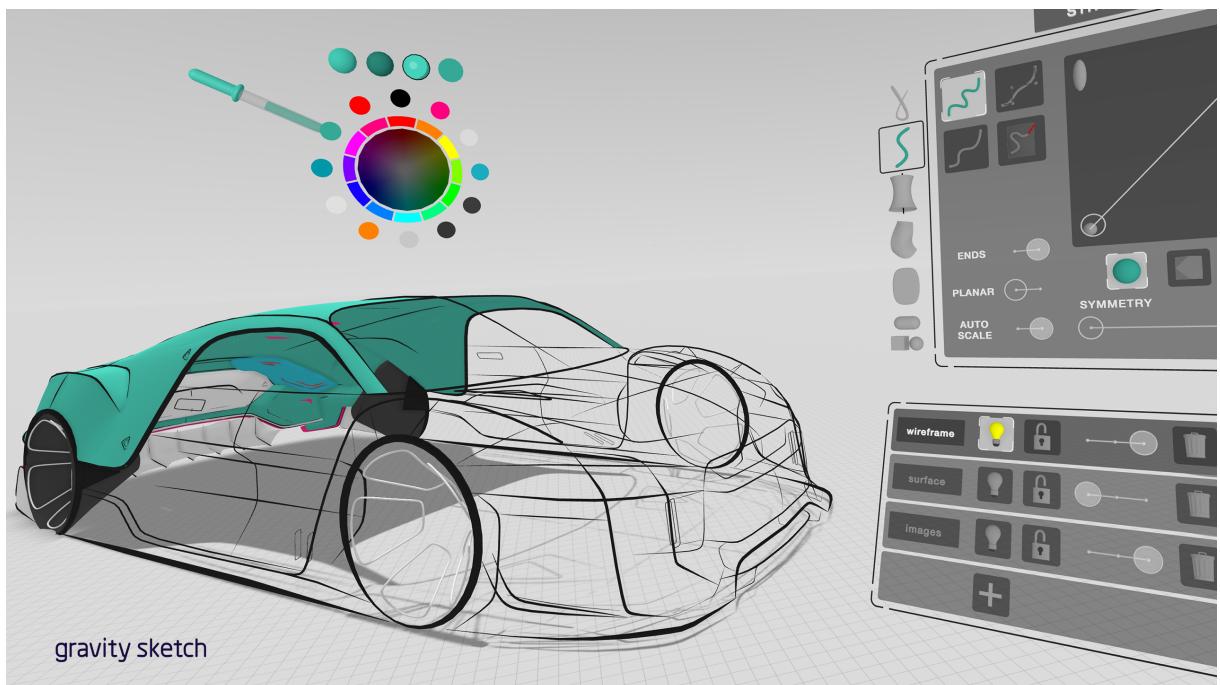
Blocks od Google jest aplikacją do tworzenia modeli 3D. Nacisk został położony na prosty, intuicyjny interfejs bez przytłaczającej liczby narzędzi. Dzięki temu aplikacja jest łatwa w użyciu nawet dla osób nie będących profesjonalnymi projektantami czy grafikami. Stworzone modele można udostępniać za pomocą platformy Poly.



Rysunek 6.2: Blocks - <https://vr.google.com/blocks/>

Gravity Sketch

Gravity Sketch jest rozwijany przez Gravity Sketch Limited. Aplikacja pozwala na stworzenie szkicu lub rysunku 3D obiektu, a następnie na wykonanie właściwego modelu. Z tego powodu posiada funkcjonalności z obu kategorii. Dodatkowo wspiera technologię Leap Motion, która pozwala na używanie aplikacji bez użycia kontrolerów, ale za pomocą gestów i ruchów rąk.



Rysunek 6.3: Gravity Sketch - <https://angel.co/gravity-sketch-1/jobs>

Tilt Brush

Tilt Brush jest aplikacją od Google służącą do tworzenia rysunków w 3D. Skupia się na zapewnieniu użytkownikowi możliwie największej liczby narzędzi. Dostępne są m.in. pędzle o różnej teksturze oraz pędzle tworzące animowane obiekty takie jak bąbelki czy poruszające się strumienie światła. Obiekty są kompatybilne z aplikacją Blocks przez platformę Poly.



Rysunek 6.4: Tilt Brush - <https://vrproject.com.pl/tilt-brush/tilt-brush>

Quill

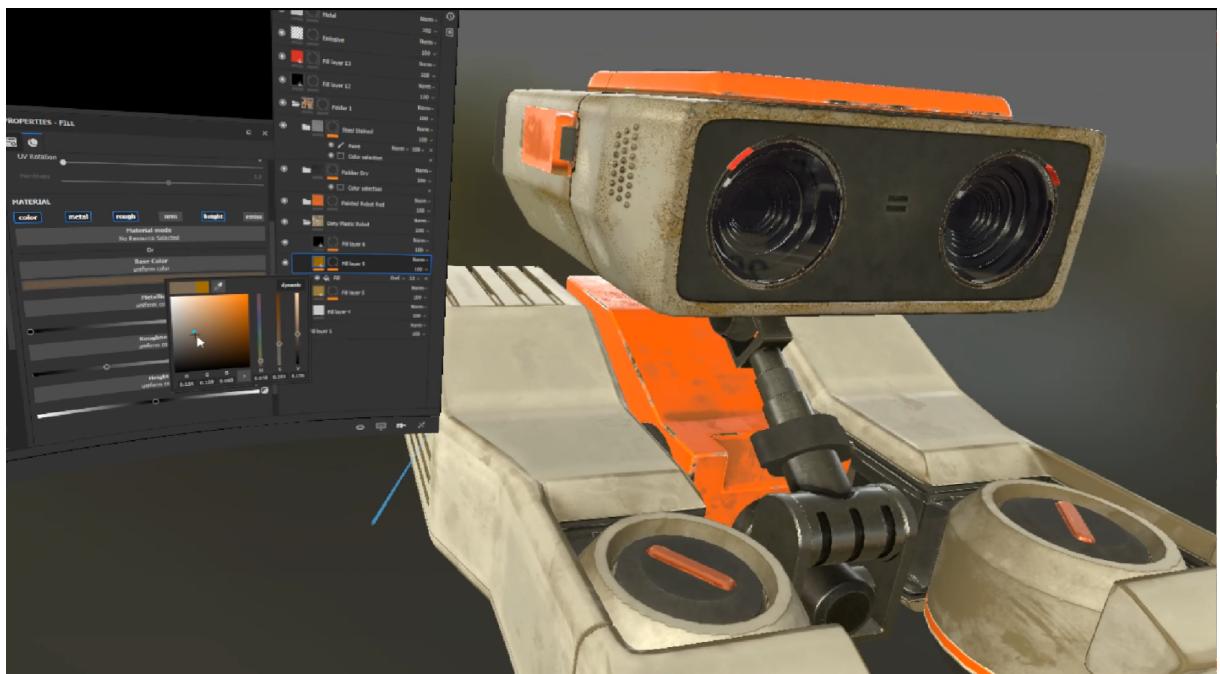
Quill jest odpowiednikiem Tilt Brush od firmy Oculus. Aplikację wyróżnia tworzenie animacji przez modyfikację trójwymiarowych rysunków.



Rysunek 6.5: Quill - <https://quill.fb.com>

Substance Painter

Substance jest profesjonalnym środowiskiem do tworzenia modeli i animacji. Jednym z modułów jest Substance Painter, który służy do tworzenia i nakładania tekstur na trójwymiarowe modele. Interfejs został przystosowany do zestawu Oculus Rift.



Rysunek 6.6: Substance Painter - <https://www.allegorithmic.com/blog/substance-painter-oculus-rift-prototype>

A-Painter

A-Painter jest aplikacją firmy Mozilla. Działa na podobnej zasadzie co Tilt Brush, ale spośród innych dostępnych aplikacji wyróżnia ją dostęp z przeglądarki. A-Painter został napisany w oparciu o framework A-Frame. Aplikacja jest bezpłatna, a kod źródłowy jest dostępny na platformie GitHub. Użyto technologii webowych w szczególności języka JavaScript i biblioteki webGL.



Rysunek 6.7: A-Painter - <https://github.com/immersive-web/webvrrocks/pull/87/files>

7. Implementacja

7.1. Technologie

Interfejs użytkownika

Interfejsem użytkownika są gogle i kontrolery wirtualnej rzeczywistości. Taki interfejs może stwarzać ograniczenia w porównaniu do zwykłej aplikacji desktopowej (np. mniejsza ilość klawiszy), ale po zaznajomieniu się z poruszaniem i nabyciu podstawowych nawyków korzystania z kontrolerów interakcja z aplikacją jest bardziej intuicyjna dla użytkownika.

Spośród dostępnych zestawów wybrano gogli **Oculus Rift** i kontrolerów **Oculus Touch Controllers**.

Silnik gry

Do stworzenia aplikacji wykorzystano silnik do tworzenia gier komputerowych, głównie z powodu zawartego w nim silnika graficznego, który służy do renderowania scen. Silnik do gier zawiera dodatkowe moduły, które wykorzystano do implementacji komunikacji z użytkownikiem, logiki systemu, komunikacji pomiędzy CPU i GPU, zapisywania i wczytywania scen.

Wybrano silnik **Unity** (od wersji 2017.4.4f1 i wzwyż) z następujących przyczyn:

- Działa na systemie operacyjnym **Microsoft Windows** i **Linux**
- **Oculus Rift** jest kompatybilny z **Unity**
- Istnieją gotowe biblioteki do odczytania informacji (pozycji, rotacji, stanu przycisków) z kontrolerów **Oculus Touch Controllers**
- Umożliwia pisanie skryptów w **C#**
- Umożliwia kompilowanie shaderów (ang. compute shader), które pozwalają na wykonywanie obliczeń ogólnego przeznaczenia na karcie graficznej.

- Zespół ma doświadczenie z pracą z **Unity**

Brano pod uwagę wykorzystanie silnika **Unreal Engine**, ponieważ dzięki użyciu języka programowania **C++** zapewnia wyższą jakość renderowanych scen przy wykorzystaniu tej samej mocy obliczeniowej. Jednak nie został wybrany głównie ze względu na brak doświadczenia zespołu w pracy z nim.

7.2. Architektura

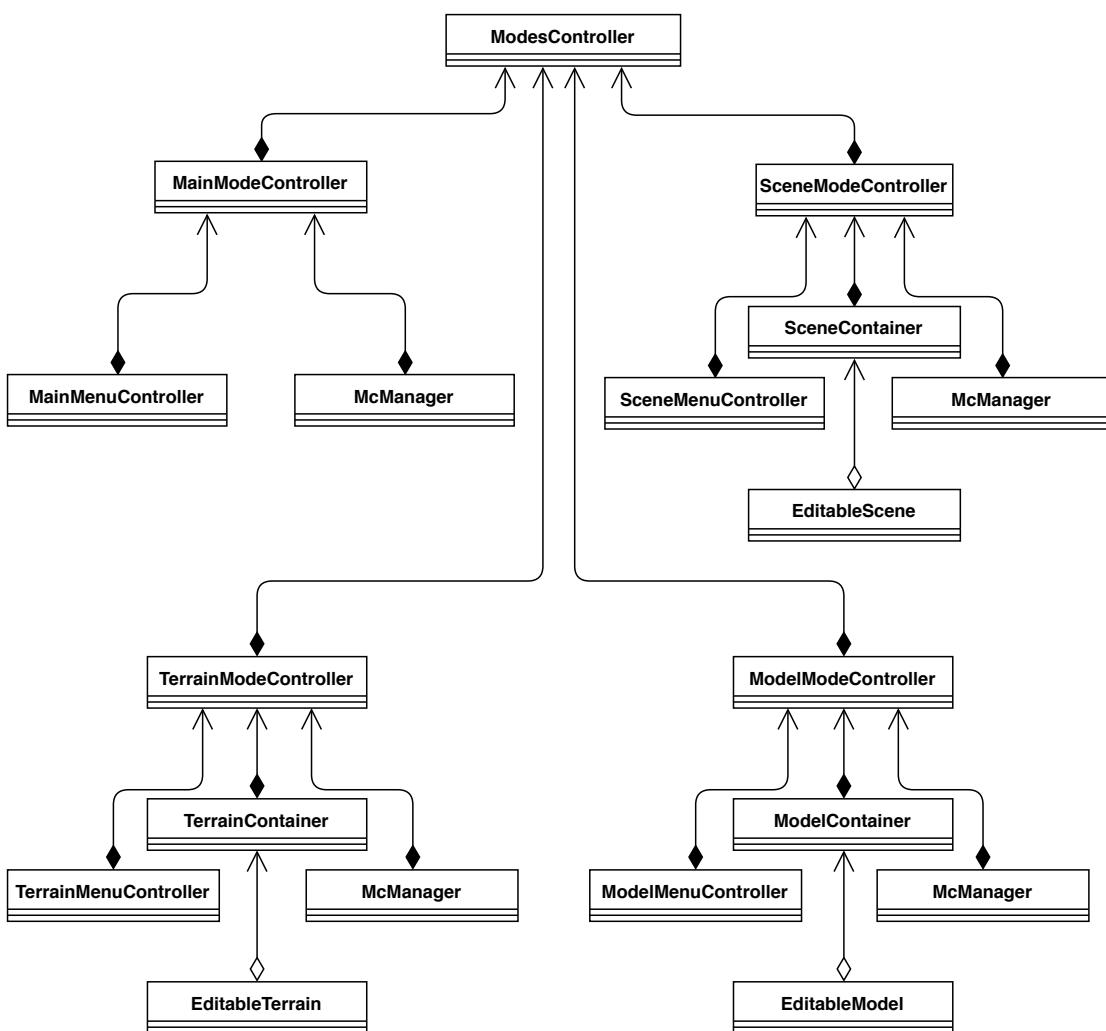
W aplikacji możemy wydzielić dwie warstwy:

- Warstwa interfejsu - Odpowiada za reagowanie na działania użytkownika na podstawie odebranych danych z gogli i kontrolerów VR. Ma na celu pokazać użytkownikowi do jakich operacji i narzędzi ma dostęp w danej chwili, zapewnić przepływ danych i informacji o podjęciu akcji do warstwy logiki, a po uzyskaniu wyników wyświetlić rezultat.
- Warstwa logiki - Odpowiada za modyfikację danych i przygotowanie edytowanych elementów do wyświetlenia. Na podstawie danych wejściowych wprowadza zmiany w edytowanej scenie, terenie lub modelu. Ma dostęp do systemu plików w celu zapisywania i wczytywania scen. Wykorzystuje GPU do przeprowadzania obliczeń wymagających dużej mocy obliczeniowej, w tym wyznaczania powierzchni terenu i modeli za pomocą algorytmu masyuerujących sześciianów. Moduł jest niezależny od interfejsu.

Do przedstawienia zależności między głównymi elementami wykorzystano diagramy klas UML.

7.2. ARCHITEKTURA

7.2.1. Klasy warstwy interfejsu



Rysunek 7.1: Diagram klas warstwy interfejsu

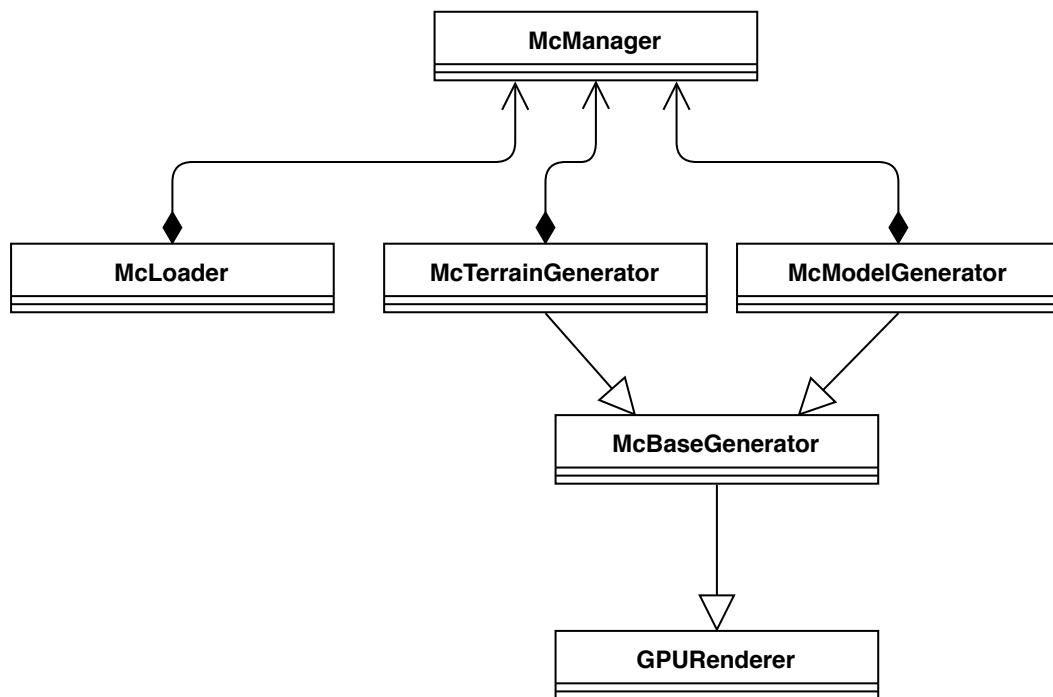
Klasy warstwy interfejsu:

- **ModesController** kontroluje przesył danych między pozostałymi kontrolerami.
- **MainModeController**, **SceneModeController**, **ModelModeController**, **TerrainModeController** - kontrolery odpowiedzialne za prawidłową aktywację i dezaktywację modułów w czasie przechodzenia między trybami. Korzystając z **ModesController** mogą wzajemnie wysyłać dane między sobą.
- **MainMenuController**, **SceneMenuController**, **ModelMenuController**, **TerrainMenuController** - kontrolery zapewniające właściwe działanie wszystkich menu. Odpowiadają

za wyświetlanie, nawigację i ukrywanie menu w ramach modułu do którego należą, a także zbierają i reagują na informacje z przycisków i gałek analogowych kontrolerów VR.

- **SceneContainer**, **ModelContainer**, **TerrainContainer** - kontenery na edytowaną scenę, model i teren. Dla uproszczenia operacje przemieszczania i obrotu są wykonywane na kontenerach a nie właściwych obiektach, z wyłączeniem obiektów ustawionych na scenie.

7.2.2. Klasy warstwy logiki



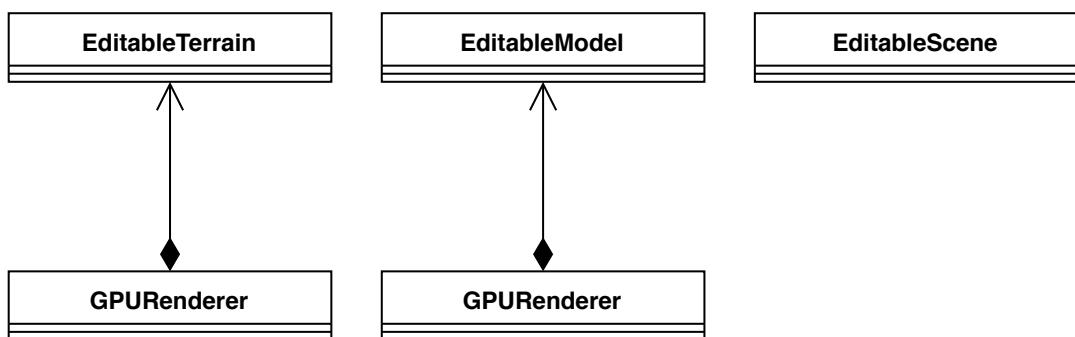
Rysunek 7.2: Diagram klas warstwy logiki, część 1

Klasy warstwy logiki:

- **GPURenderer** pozwala na generowanie siatek trójkątów z użyciem algorytmu maszerujących sześciianów. Obliczenia są wykonywane na karcie graficznej.
- **McTerrainGenerator** odpowiada za stworzenie powierzchni dla terenu.
- **McModelGenerator** odpowiada za stworzenie powierzchni dla modelu.
- **McBaseGenerator** jest klasą bazową dla **McTerrainGenerator** i **McModelGenerator**. Sufcowe wyniki obliczeń przetwarza do obiektów klas używanych w aplikacji, a także na ich podstawie tworzy odpowiednie obiekty Unity.

7.3. OPIS WYBRANYCH ASPEKTÓW

- **McLoader** odpowiada za zapisywanie i wczytywanie danych, a także serializację i deserializację sceny, terenu i modeli. Definiuje format i lokalizację przechowywanych danych.
- **McManager** wykorzystując powyższe moduły jest interfejsem dla warstwy wyżej do wygodnego tworzenia, usuwania, wczytywania i zapisywania obiektów sceny, terenu i modeli. Nie jest tylko implementacją wzorca projektowego fasady, ale ma na celu konfigurację tworzonych obiektów tak, aby były gotowe do wykorzystania bez dodatkowych modyfikacji czy inicjalizacji. Definiuje też strukturę zapisu danych i konwencje nazewnicze obiektów Unity.



Rysunek 7.3: Diagram klas warstwy logiki, część 2

- **EditableTerrain** przechowuje stan załadowanego terenu i pozwala na jego edycję. Obliczenia są wykonywane na karcie graficznej.
- **EditableModel** jest odpowiednikiem **EditableTerrain** dla modelu.
- **EditableScene** przechowuje stan załadowanej sceny i pozwala na jej edycję. W skład sceny wchodzą teren, obiekty na scenie i lista modeli. Scena nie potrzebuje dostępu do klas przeprowadzających obliczenia na karcie graficznej.

7.3. Opis wybranych aspektów

7.3.1. Ikony scen i modeli

Podczas wybierania sceny z listy scen lub modelu z listy modeli potrzebny jest sposób na identyfikację szukanego elementu. Zrezygnowano z wprowadzania nazw przez użytkownika, ponieważ jest to wystarczająco intuicyjne, a wymaga wprowadzenia dedykowanego interfejsu. Jako rozwiązanie wybrano robienie zdjęć i wyświetlanie dostępnych scen i modeli jako listy ikon.

Zdjęcie jest robione z widoku kamery użytkownika. Na ten czas kontrolery i menu zostają ukryte, aby nie znalazły się na wynikowej ikonie. Zostaną one aktywowane przed kolejną klatką, więc nie będzie to widoczne dla użytkownika. Z klasy `Camera` użyta została metoda `Render`, która pozwala wymusić wyrenderowanie obrazu kamery, a także instancja klasy `RenderTexture` dzięki której obraz kamery jest renderowany do obiektu tekstury, a nie na ekran. Wynik jest zapisywany w pliku z rozszerzeniem `.png`.

7.3.2. Zapisywanie i wczytywanie

W aplikacji wyróżniono trzy możliwości zapisu:

- zapis sceny
- zapis terenu
- zapis modelu

Stan modelu jest zapisywany w klasie `McData`. Przechowywane są dane o wierzchołkach i są to informacje wystarczające do odtworzenia modelu (do ponownego wygenerowanego modelu). Dzięki temu nie ma potrzeby przechowywania skomplikowanych obiektów silnika Unity opisujących edytowany model. Zapis terenu przebiega analogicznie do zapisu modelu.

Listing 7.1: McData.cs

```

1 public class McData
2 {
3     public Guid Guid { get; set; }
4     public float[] Values { get; set; }
5     public Vector4[] Colors { get; set; }
6 }
```

Scena nie definiuje struktury elementów, ale ułożenie tych elementów względem siebie. Dlatego klasa `McSceneData` nie zawiera danych o żadnych wierzchołkach, ale identyfikator terenu jaki należy wczytać i listę obiektów klasy `McSceneModelData` zawierającą dane o położeniu obiektów na scenie. Podczas wczytywania sceny wykorzystywane są wczytane i wygenerowane modele do nadania obiektom wymaganego kształtu.

7.3. OPIS WYBRANYCH ASPEKTÓW

Listing 7.2: McSceneData.cs

```
1 public class McSceneData
2 {
3     public Guid Guid { get; set; }
4     public Guid TerrainGuid { get; set; }
5     public List<McSceneModelData> Models { get; set; }
6 }
```

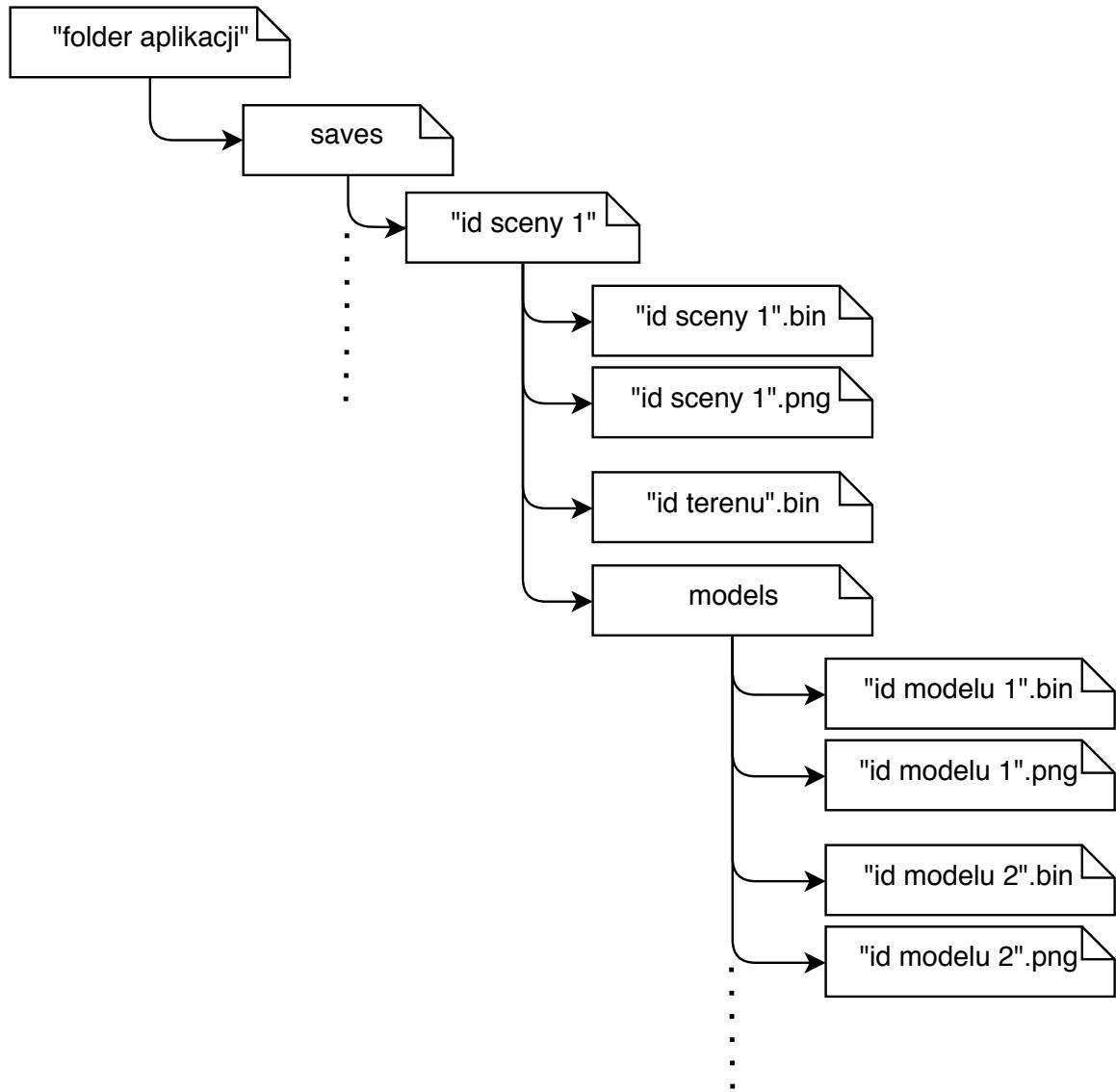
Listing 7.3: McSceneModelData.cs

```
1 public class McSceneModelData
2 {
3     public Guid Guid { get; set; }
4     public Vector3 Position { get; set; }
5     public Vector3 Rotation { get; set; }
6     public Vector3 Scale { get; set; }
7 }
```

Wczytywanie jest wykonywane tylko przy ładowaniu sceny. Załadowana zostaje scena z terenem i modelami. Klasy `EditScene`, `EditTerrain` i `EditModel` zawierają publiczne metody `GetData` i `SetData` do załadowania i pobrania przechowywanych danych.

Dane są zapisywane jako pliki binarne. Nie użyto popularnych formatów takich jak XML czy JSON, ponieważ serializowane dane są używane tylko wewnątrz aplikacji, tzn. nie przesyłano ich przez sieć, nie korzystają z nich inne systemy i nie ma potrzeby, by były modyfikowane ręcznie przez użytkownika. Wybrana metoda zapewnia dostatecznie dużą szybkość serializacji i deserializacji.

Wszystkie sceny są przechowywane w katalogu „saves” w głównym katalogu aplikacji. Każda scena mieści się w odrębnym katalogu nazwanym identyfikatorem sceny, w którym znajdują się dane sceny, dane terenu i dane modeli w podkatalogu „models”. Każdy plik jest nazwany identyfikatorem obiektu którego dotyczy i ma rozszerzenie `.bin`. Ikony scen i modeli są zapisywane w tych samych katalogach co pliki z danymi.



Rysunek 7.4: Struktura zapisywanych plików

7.3.3. Wykorzystanie GPU w obliczeniach

Pożądana dokładność tworzonych elementów wymusza użycie algorytmu maszerujących sześciianów wykorzystując gęstą siatkę próbek. Wymagane obliczenia, jeśli przeprowadzane są na CPU, stają się zbyt czasochłonne co nie pozwala zapewnić użytkownikowi możliwość modyfikowania elementów w czasie rzeczywistym. Dlatego wszystkie operacje modyfikacji na generowanych strukturach zostały przeniesione na GPU.

Wykorzystane zostały wbudowane narzędzia i funkcjonalności Unity. Algorytmy zaimplementowano w shaderach obliczeniowych (ang. compute shader) w języku HLSL. Silnik Unity pozwala na wygodne uruchamianie napisanych shaderów z konwersją typów wysyłanych argumentów. Po

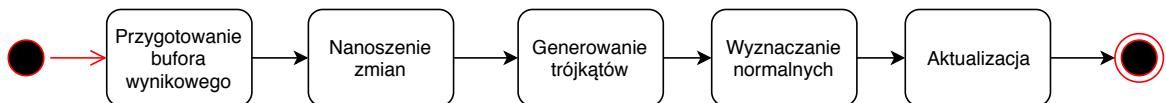
7.3. OPIS WYBRANYCH ASPEKTÓW

zakończeniu obliczeń dane mogą być pobrane lub też wykorzystane przez kolejny shader.

Przed uruchomieniem shaderów wykonywana jest część inicjalizacyjna. Następuje przygotowanie na GPU buforów dla argumentów i wyników dla wszystkich używanych shaderów, a także struktury danych po stronie CPU na ostateczny wynik. Po załadowaniu do buforów dostarczonych danych przeprowadzane są obliczenia.

Kolejne kroki obliczeń:

1. Przygotowanie bufora wynikowego - Ze względu na strukturę i sposób zapisu wynikowych danych, bufor należy wypełnić domyślnymi wartościami. Ta operacja jest wykonywana na GPU, aby uniknąć kosztownego przesyłu danych z CPU na GPU przy każdej modyfikacji. Ten etap jest niezależny od następnego, ale ze względu na niski koszt obliczeń, nie ma potrzeby zrównoleglania obu procesów.
2. Nanoszenie zmian - Na podstawie pozycji i parametrów pędzla zmieniane są dane w buforze danych próbkowych. Modyfikacje są uzależnione od rodzaju pędzla i wybranego trybu.
3. Generowanie trójkątów - Przeprowadzany jest proces generowania wierzchołków na podstawie dostarczonych próbek i tabel pomocniczych wykorzystując algorytm maszerujących sześciąników. Struktury wierzchołków z wyznaczonymi pozycjami i kolorami są zapisywane w buforze wynikowym.
4. Wyznaczanie normalnych - Dla wyznaczonych wierzchołków obliczane są wartości normalnych zgodnie z przyjętym wcześniej algorytmem.
5. Aktualizacja - Pobierane są dane z bufora wynikowego i wykorzystywane są do zaktualizowania edytowanego obiektu.



Rysunek 7.5: Diagram stanów dla obliczeń na GPU

7.3.4. Programowanie reaktywne

Programowanie reaktywne jest jednym z paradygmatów programowania, który jest oparty na zdarzeniach. Główną ideą programowania reaktywnego jest istnienie dwóch komponentów - obserwatora (ang. Subscriber) i obiektu obserwowanego (ang. Observable). Obiekt obserwowany

tworzy strumienie danych, w które umieszcza informacje o zmianach. Obserwator (może być więcej niż jeden) subskrybuje się na ten strumień i decyduje co robić z dostarczoną informacją.

W aplikacji użyto biblioteki UniRX (Reactive Extensions for Unity). Jest to reimplementacja biblioteki .NET Reactive Extensions specjalnie dla silnika Unity. UniRX pozwala na wygodne filtrowanie i operowanie na danych otrzymanych ze strumieni.

Programowanie reaktywne zostało użyte by mieć możliwość przesyłu danych pomiędzy komponentami. Takie podejście pozwala wyeliminować dwustronne zależności. Jedna ze stron jest odpowiedzialna za całą logikę, a druga jest ograniczona do udostępniania posiadanych informacji.

Niniejszy przykład przedstawia zablokowanie otwierania i zamykania menu w przypadku, gdy zostanie wybrana jedna z pozycji tego menu. Klasa `MenuRightModelController` odpowiada za sterowanie menu. Posiada ona strumień `ItemSelectedStream`, na który jest umieszczana informacja o tym że jedna z pozycji została wybrana (linijka 13 Listing 7.4).

Listing 7.4: MenuRightModelController.cs

```

1 public class MenuRightModelController : MonoBehaviour
2 {
3     private ISubject<Unit> itemSelectedSubject = new Subject<Unit>();
4     public IObservable<Unit> ItemSelectedStream
5             { get { return itemSelectedSubject; } }
6     ...
7     void Update()
8     {
9         ...
10        if (OVRInput.Get(selectItemButton))
11        {
12            ...
13            itemSelectedSubject.OnNext(Unit.Default);
14        }
15    }
16    ...
17 }
```

Klasa `MenuModelController` odpowiada za prawidłowe otwieranie i zamykanie wyżej wspomnianego menu. W funkcji `Start` (czyli funkcji wywoływanej jednorazowo przy włączeniu aplikacji)

7.3. OPIS WYBRANYCH ASPEKTÓW

kacji) `MenuModelController` subskrybuje się na strumień o nazwie `ItemSelectedStream`. Gdy pojawi się w nim dowolna informacja, zostanie wywołana metoda `StopListeningForRight`, czyli metoda która blokuje zamykanie i otwieranie menu.

Listing 7.5: MenuModelController.cs

```
1 public class MenuModelController : MonoBehaviour
2 {
3     [ SerializeField ] private MenuRightModelController menuRightController ;
4     ...
5     private void Start ()
6     {
7         ...
8         menuRightController . ItemSelectedStream . Subscribe ( _ =>
9             StopListeningForRight ( ) );
10        ...
11    }
12    ...
13 }
```

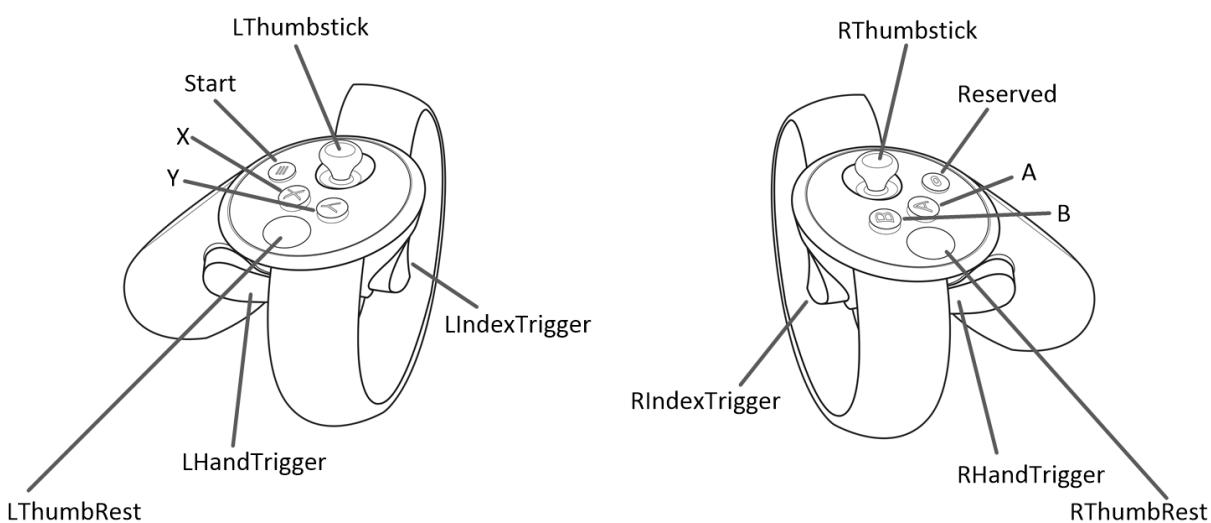
7.3.5. Zaznaczanie obiektu

Do manipulowania obiektem na scenie wymagane jest wcześniejsze jego zaznaczenie. Polega ono na skierowaniu promienia (by rzucić promień trzeba wcisnąć odpowiedni przycisk) wychodzącego z kontrolera tak, aby na jego drodze staną dany obiekt sceny i puścić wciśnięty wcześniej przycisk. Rozpatrywany jest tylko pierwszy obiekt przecinający promień. Jeśli będzie to teren lub nic nie stoi na drodze promienia to nie zostaje wykonana żadna akcja.

Do wizualizacji promienia wykorzystano klasę `LineRenderer`, która pozwala na renderowanie linii o określonym kolorze, rozmiarze, punkcie początkowym i końcowym. Do puszczenia promienia wykorzystano metodę `Physics.Raycast`, która zwraca informację czy promień napotkał jakiś obiekt. Dodatkowo otrzymujemy strukturę `RaycastHit` ze szczegółami napotkanego obiektu.

8. Instrukcja użytkownika

Nazwy przycisków użyte w dalszej części są zgodne z nazwami przedstawionymi na rysunku 8.1.



Rysunek 8.1: Oculus touch controllers - oznaczenia przycisków - <https://developer.oculus.com/documentation/unity/latest/concepts/unity-ovrinput/>

8.1. Zasady działania menu

Szczegółowe opisy menu znajdują się w dalszej części tego rozdziału. Wszystkie menu zostały stworzone zgodnie z tą samą konwencją. Poniżej zostały opisane wspólne zasady działania dla wszystkich menu.

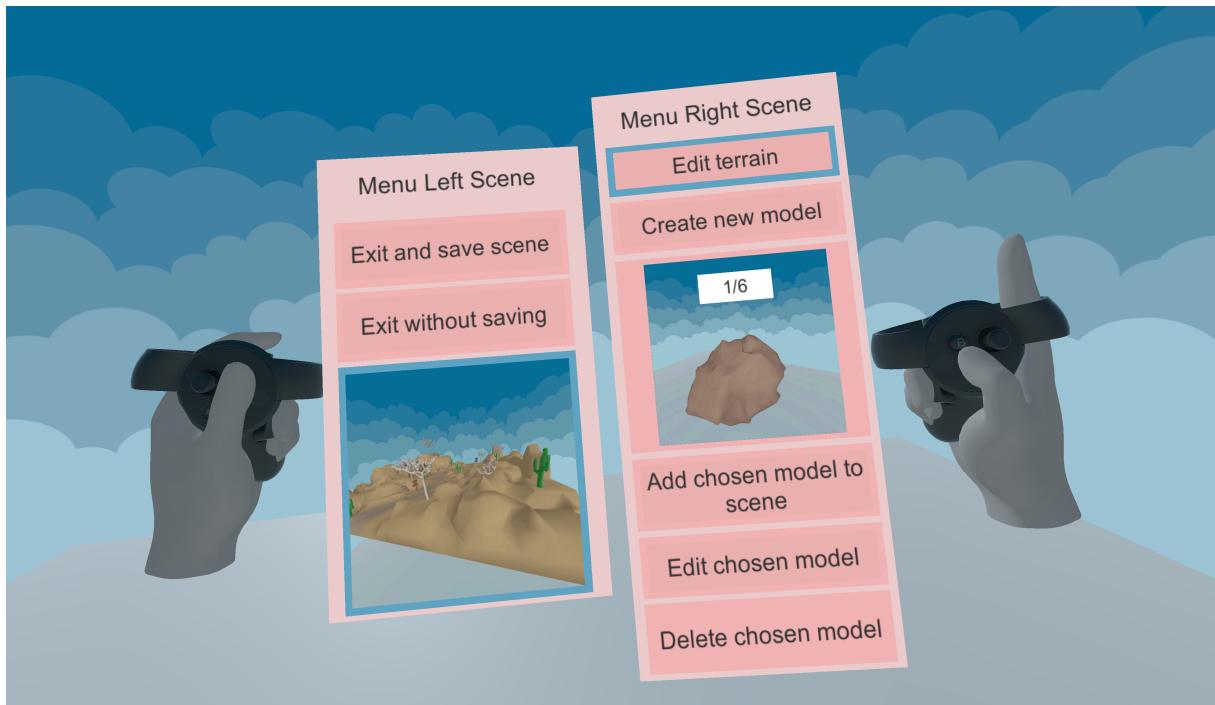
Menu są obiektami „przyczepionymi” do kontrolerów. Nawigacja odbywa się za pomocą gałek analogowych. Dla menu lewej ręki jest to LThumbstick:

- prawo - otwarcie menu
- lewo - zamknięcie menu

8.1. ZASADY DZIAŁANIA MENU

Dla menu prawej ręki jest to LThumbstick:

- lewo - otwarcie menu
- prawo - zamknięcie menu

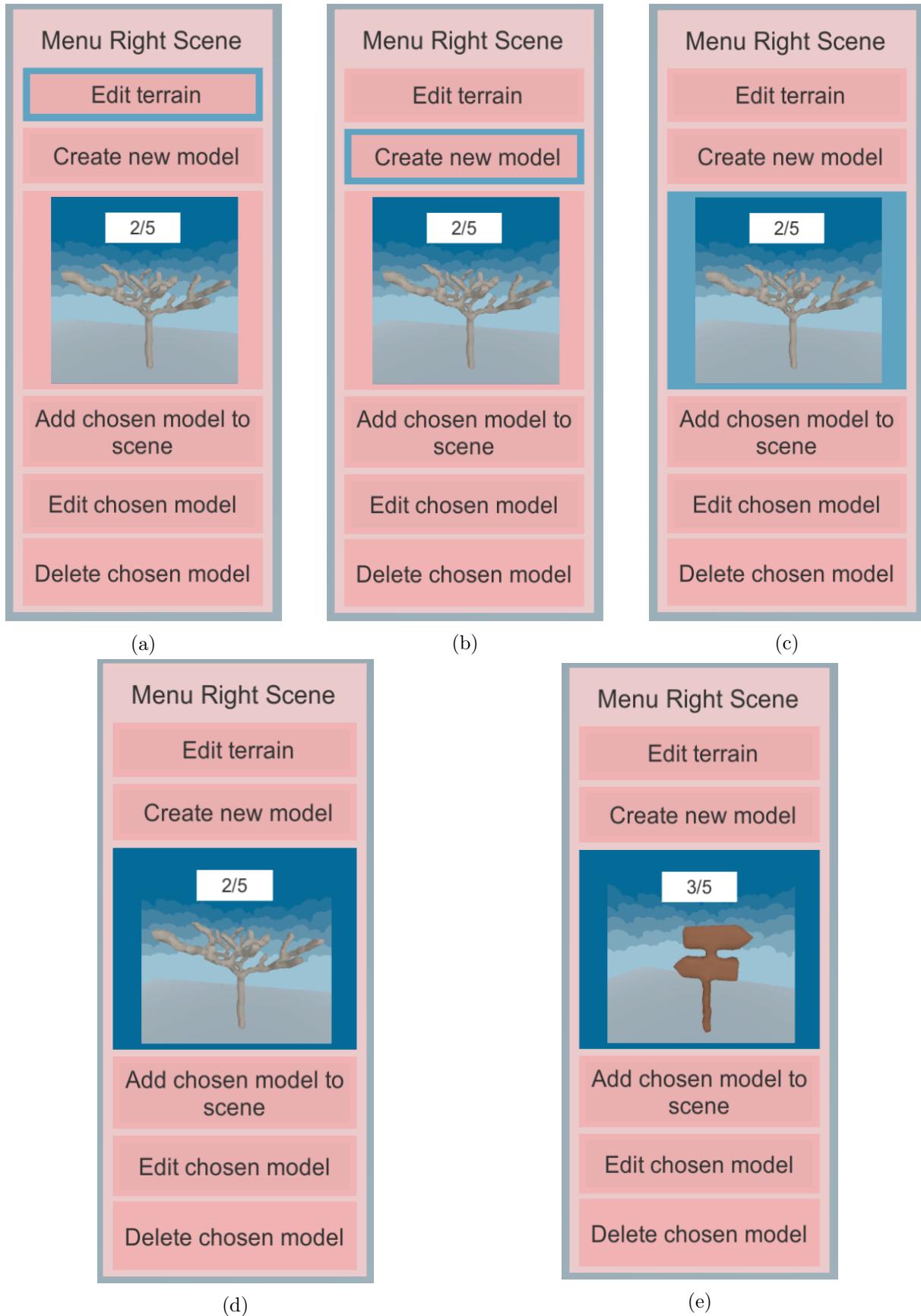


Rysunek 8.2: Przykładowy wygląd menu

Przy otwartym menu:

- dół, góra - zmiana aktywnej pozycji menu
- kliknięcie - wybór aktywnej pozycji

Na Rysunku 8.3 pokazane jest przykładowe działanie menu. Rysunek 8.3a przedstawia przykładowy stan początkowy po otwarciu menu. Aktywną pozycją jest **Edit terrain**. Rysunki 8.3b i 8.3c przedstawiają kolejne etapy po przesunięciu gałki analogowej dwa razy w dół. Aktywowane są kolejno pozycje **Create new model** i **Model preview**. Rysunek 8.3d przedstawia stan menu po kliknięciu gałki analogowej - pozycja **Model preview** jest wybrana. Jest to pozycja typu „horizontalne podmenu” (patrz Rozdział 8.1.1), więc możemy teraz zmieniać pogląd modeli za pomocą przesuwania gałki analogowej w lewo bądź w prawo. Na Rysunku 8.3e pokazane jest menu po przesunięciu gałki analogowej w prawo.



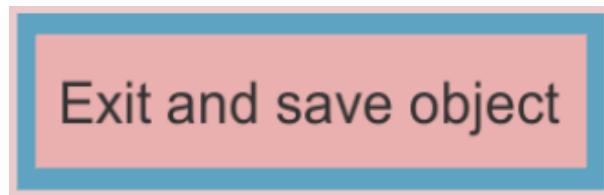
Rysunek 8.3: Kolejne etapy poruszania się w menu

8.1. ZASADY DZIAŁANIA MENU

8.1.1. Typy pozycji menu

Pozycje w menu dzielą się na kilka typów:

- Proste - po kliknięciu zostaje podjęta pewna akcja.



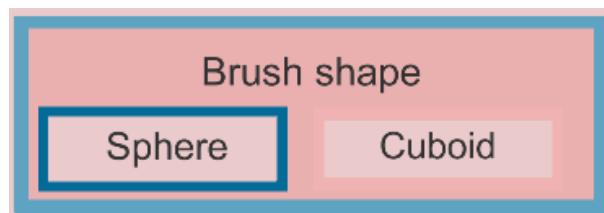
Rysunek 8.4: Przykład pozycji typu „prostego”

- Horyzontalne podmenu - po kliknięciu użytkownik może zmieniać aktywną pozycję podmenu (gałka analogowa w lewo bądź w prawo). Aby wyjść należy kliknąć gałkę analogową.



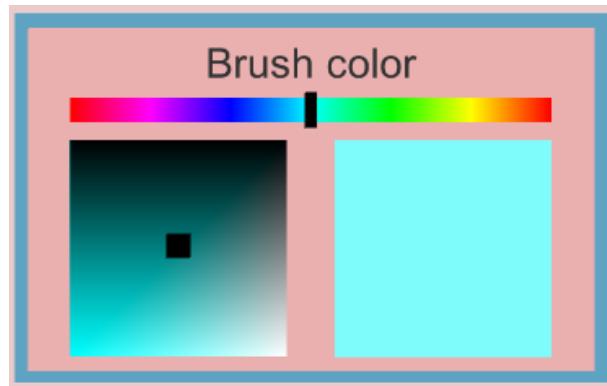
Rysunek 8.5: Przykład pozycji typu „horyzontalne podmenu”

- Suwak - po kliknięciu użytkownik może zmieniać wartość suwaka (gałka analogowa w lewo bądź w prawo). Aby wyjść należy kliknąć gałkę analogową.



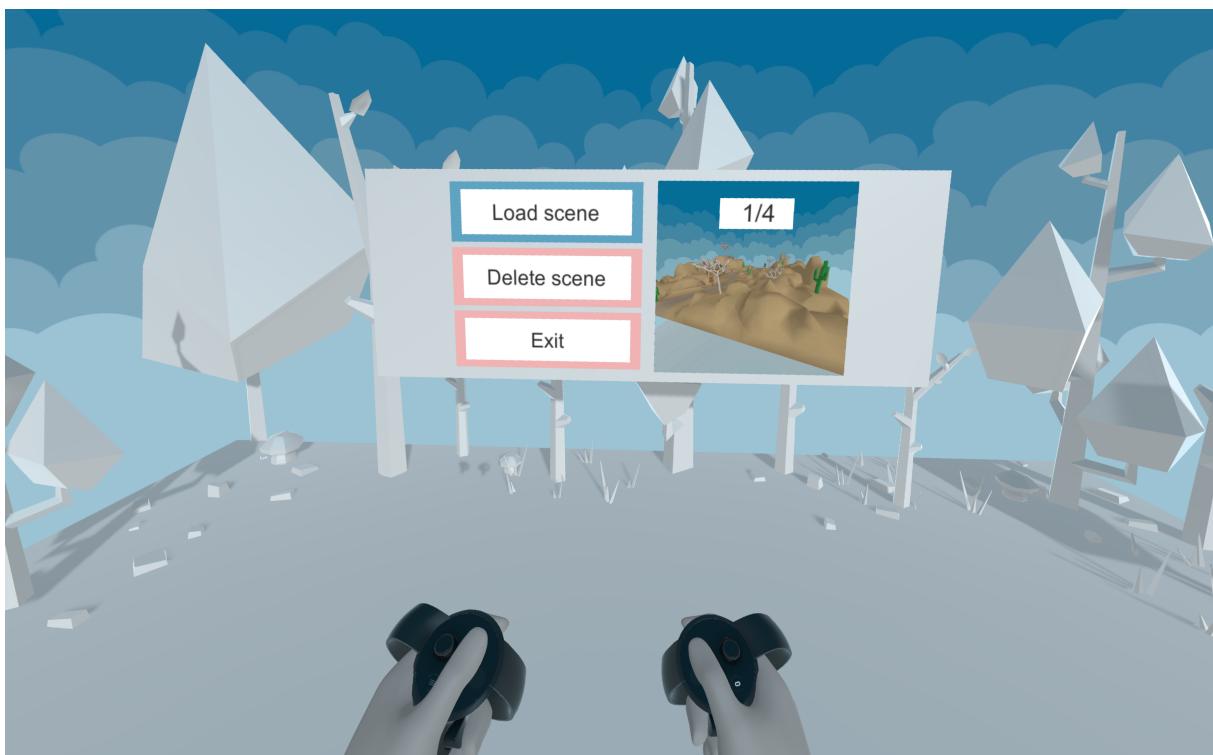
Rysunek 8.6: Przykład pozycji typu „suwak”

- Kolor - pozwala zmienić kolor. Zmiana nasycenia za pomocą gałki analogowej, zmiana odcienia - IndexTriger i HandTriger. Aby wyjść należy kliknąć gałkę analogową.



Rysunek 8.7: Przykład pozycji typu „kolor”

8.2. Tryb główny



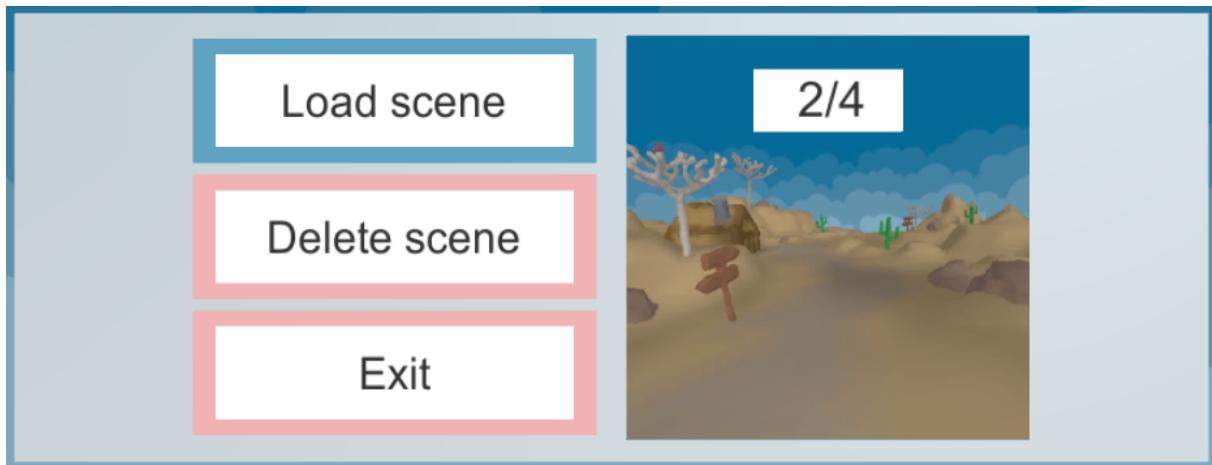
Rysunek 8.8: Widok gracza po włączeniu aplikacji

Przy włączeniu aplikacji użytkownik znajduje się w trybie głównym. Przed użytkownikiem znajduje się menu główne. Do nawigacji jest użyta gałka analogowa LThumbstick:

- lewo, prawo - zmiana wybranej sceny
- dół, góra - zmiana aktywnej pozycji menu

8.3. TRYB EDYCJI SCENY

- kliknięcie - wybór aktywnej pozycji
 - Load scene - przejście do trybu edycji wybranej sceny
 - Delete scene - usunięcie wybranej sceny
 - Exit - wyjście z aplikacji



Rysunek 8.9: Menu trybu głównego

8.3. Tryb edycji sceny

8.3.1. Lewy kontroler

Ze względu na problem lokomocji w VR, użytkownikowi zostały ograniczone możliwości poruszania się w VR. Zamiast tego zostały udostępnione narzędzia do przesuwania i rotacji wszystkich obiektów z którym użytkownik może mieć interakcje.

Przesuwanie sceny - przycisk Start:

- Wciśnięcie przycisku i przesunięcie kontrolera powoduje, że scena przesuwa się zgodnie z przesunięciem kontrolera.
- Puszczenie przycisku oznacza koniec przesuwania.

Rotacja sceny - przycisk X:

- Wciśnięcie przycisku i przesunięcie kontrolera w lewo bądź w prawo powoduje, że scena obraca się zgodnie z przesunięciem kontrolera .
- Puszczenie przycisku oznacza koniec rotacji.

Skalowanie sceny - przycisk Y:

- Wciśnięcie przycisku i przesunięcie kontrolera do sceny powoduje zmniejszenie sceny, od-
sunięcie kontrolera od sceny powoduje zwiększenie sceny.
- Puszczenie przycisku oznacza koniec skalowania.

Menu:

- `Exit and save scene` - zapis zmian na scenie i powrót do trybu głównego
- `Exit without saving` - powrót do trybu głównego bez zapisu zmian
- `Scene preview` - zrobienie nowego zdjęcia sceny. Zdjęcie jest zapisem aktualnego widoku
użytkownika z pominięciem kontrolerów.



Rysunek 8.10: Lewe menu w trybie edycji sceny

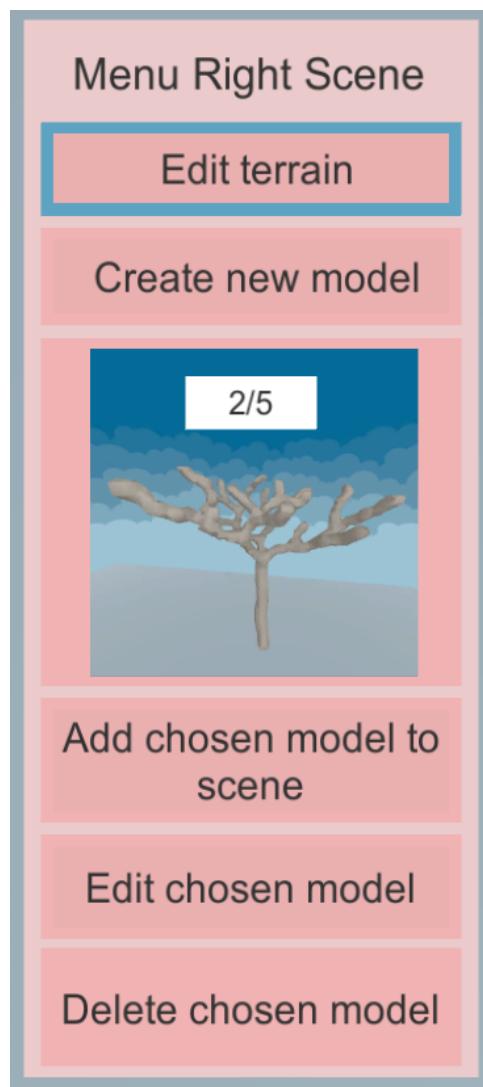
8.3.2. Prawy kontroler

Menu:

- `Edit terrain` - przejście do trybu edycji terenu
- `Create new model` - przejście do trybu edycji nowego modelu

8.3. TRYB EDYCJI SCENY

- **Model preview** - pozwala zmieniać wybrany model (lewo, prawo). Należy kliknąć RThumbstick aby wyjść.
- **Add chosen model to scene** - dodanie aktywnego modelu do sceny. Tryb edycji sceny znajduje się w trybie edycji obiektu na scenie.
- **Edit chosen model** - przejście do trybu edycji wybranego modelu
- **Delete chosen model** - usunięcie wybranego modelu



Rysunek 8.11: Prawe menu w trybie edycji sceny

8.3.3. Tryb edycji sceny przy zaznaczonym obiekcie

Lewy kontroler

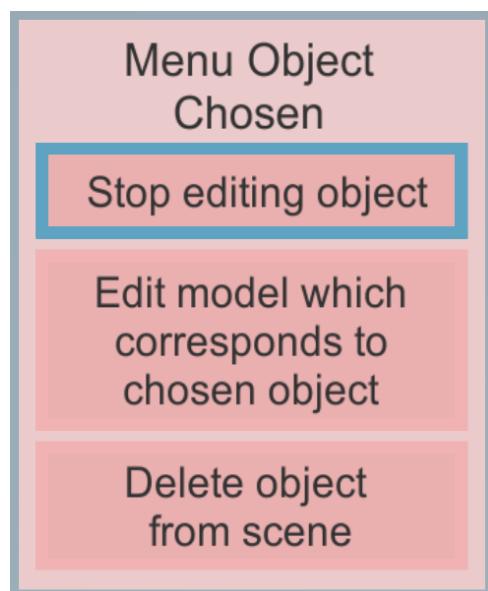
Przesuwanie i skalowanie obiektu działają analogicznie do przesuwania i skalowania sceny.

Rotacja obiektu - przycisk X:

- Wciśnięcie przycisku i obracanie kontrolera w dowolnej osi powoduje, że obiekt przyjmuje rotację kontrolera.
- Puszczenie przycisku oznacza koniec rotacji.

Menu prawego kontrolera:

- `Stop editing object` - powrót do trybu edycji sceny
- `Edit model which corresponds to chosen object` - przejście do trybu edycji modelu zaznaczonego obiektu
- `Delete object from scene` - usunięcie obiektu ze sceny i powrót do trybu edycji sceny



Rysunek 8.12: Prawe menu w trybie edycji sceny przy zaznaczonym obiekcie

8.4. Tryb edycji terenu

8.4.1. Lewy kontroler

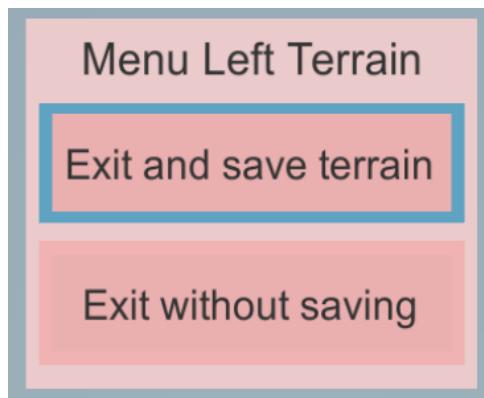
Przesuwanie, rotacja i skalowanie terenu działają analogicznie do przesuwania, rotacji i skalowania sceny.

Menu:

- `Exit and save scene` - zapis zmian na scenie i powrót do trybu edycji sceny

8.4. TRYB EDYCJI TERENU

- **Exit without saving** - powrót do trybu edycji sceny bez zapisu zmian



Rysunek 8.13: Lewe menu w trybie edycji terenu

8.4.2. Prawy kontroler

W trybie **ksztaltowania terenu**:

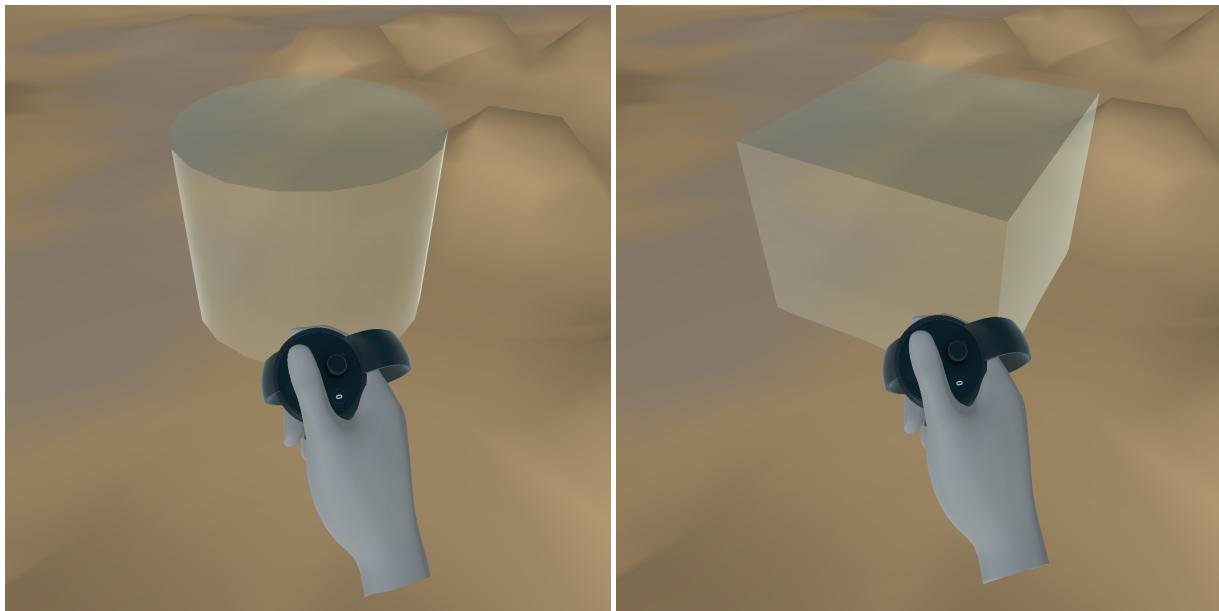
- przycisk B - Wciśnięcie i przytrzymanie przycisku powoduje podniesienie lub obniżenie terenu w zakresie pędzla. Zmiany zależą od kierunku przesuwania kontrolera.
- przycisk A - Wciśnięcie i przytrzymanie przycisku powoduje analogiczne zachowanie do przycisku B ale w trybie „extreme”. Podnoszone są najniższe, a obniżane najwyższe punkty terenu.

W trybie **kolorowania terenu**

- przycisk B - Wciśnięcie i przytrzymanie przycisku koloruje teren w zakresie pędzla zgodnie z wybranym wcześniej kolorem

Menu:

- **Modification mode** - pozycja typu „horizontalne menu”. Pozwala zmienić obecny typ modyfikacji pomiędzy ksztaltowaniem a kolorowaniem.
- **Brush shape** - pozycja typu „horizontalne menu”. pozwala zmienić obecny typ modyfikacji pomiędzy kołem i prostokątem.

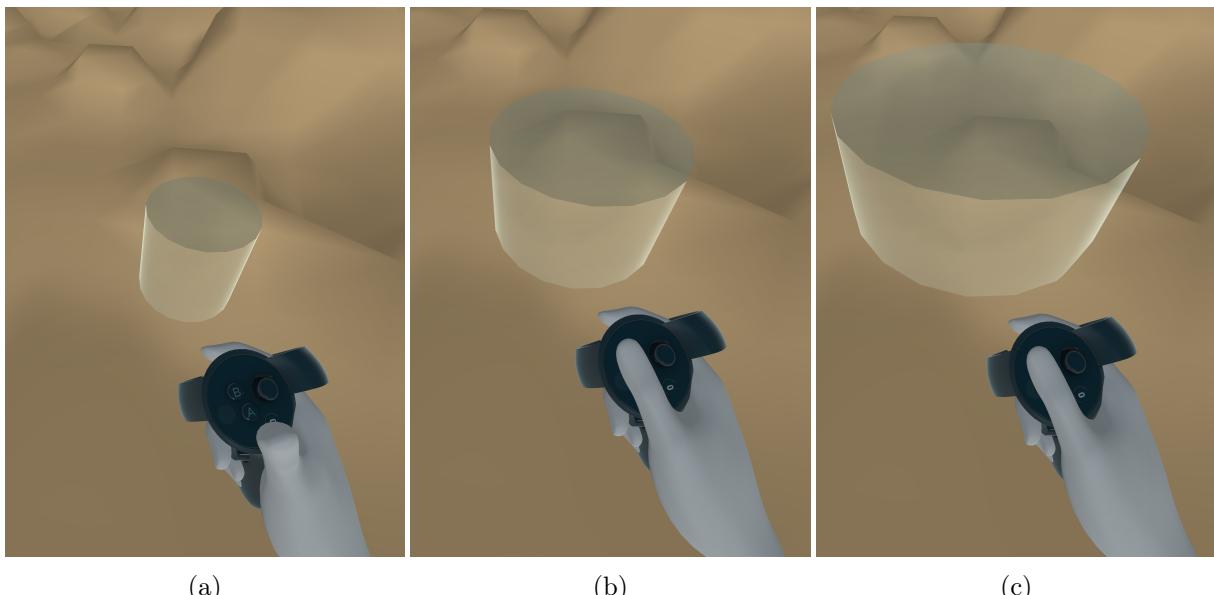


(a) Koło

(b) Prostokąt

Rysunek 8.14: Rodzaje pędzla w trybie edycji terenu

- **Brush size** - pozycja typu „suwak”. pozwala zmienić rozmiar pędzla.



(a)

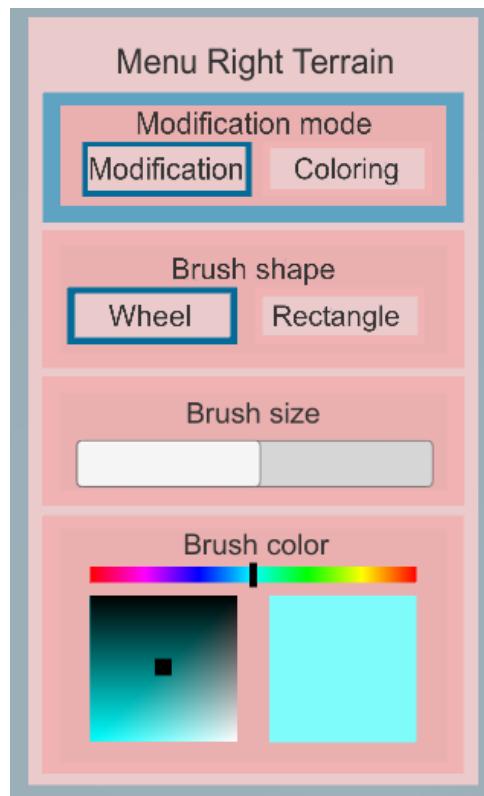
(b)

(c)

Rysunek 8.15: Przykładowe rozmiary pędzla w trybie edycji terenu

- **Brush Color** - pozycja typu „kolor”. Pozwala zmienić kolor pędzla dla trybu kolorowania.

8.5. TRYB EDYCJI MODELU



Rysunek 8.16: Menu prawej ręki w trybie edycji terenu

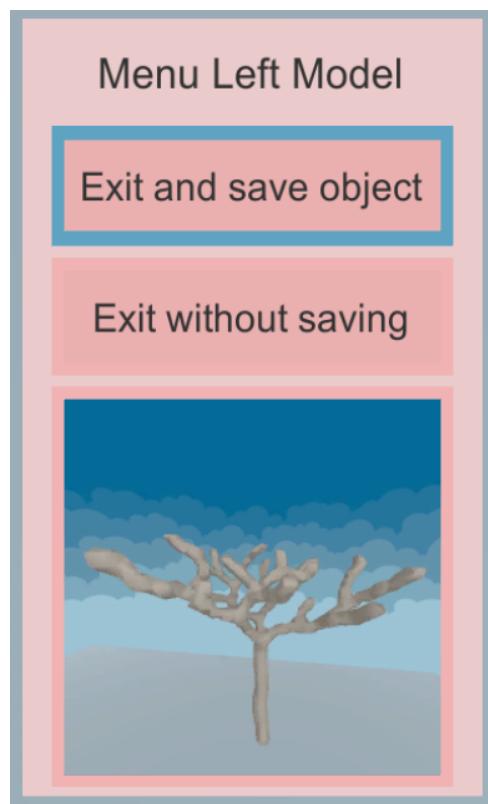
8.5. Tryb edycji modelu

8.5.1. Lewy kontroler

Przesuwanie, rotacja i skalowanie terenu działają analogicznie do przesuwania, rotacji i skalowania obiektu w trybie edycji sceny przy zaznaczonym obiekcie.

Menu:

- **Exit and save scene** - zapis zmian na scenie i powrót do trybu edycji sceny.
- **Exit without saving** - powrót do trybu edycji sceny bez zapisu zmian.
- **Model preview** - zrobienie nowego zdjęcia sceny. Zdjęcie się robi z widoku gracza. Na czas robienia zdjęcia kontrolery i menu jest chowane.



Rysunek 8.17: Lewe menu w trybie edycji modelu

8.5.2. Prawy kontroler

W trybie **ksztaltowania modelu**

- przycisk B - Wciśnięcie i przytrzymanie przycisku powoduje dodawanie modelu w zakresie pędzla.
- przycisk A - Wciśnięcie i przytrzymanie przycisku powoduje usuwanie modelu w zakresie pędzla

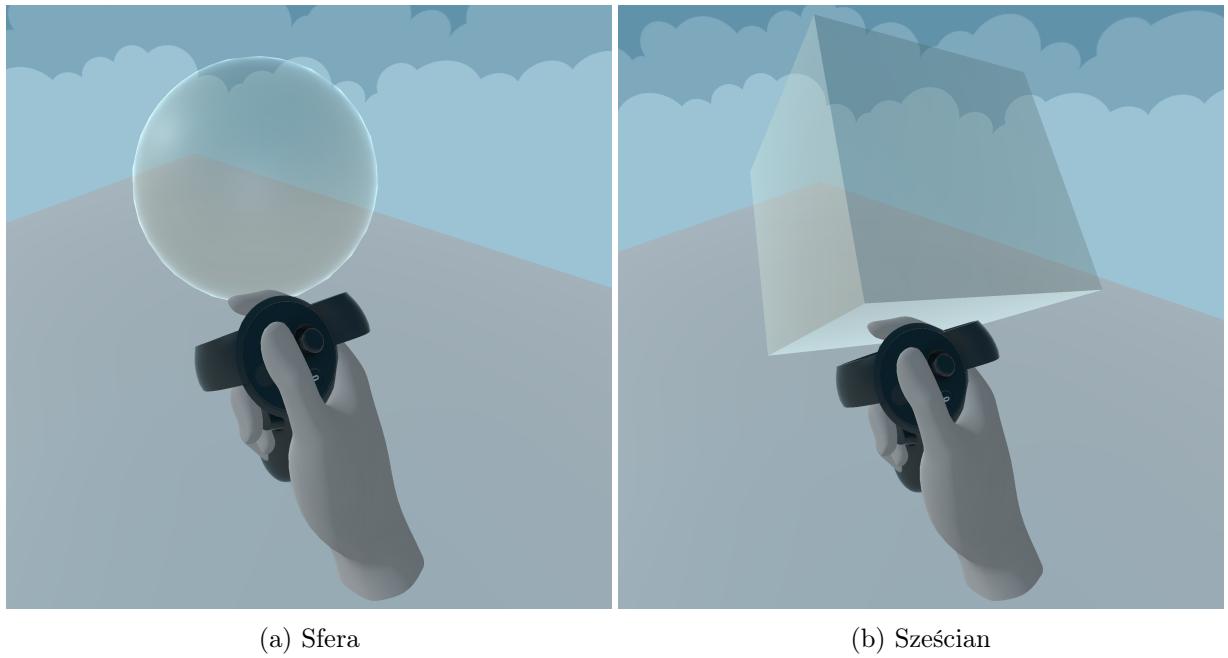
W trybie **kolorowania modelu**

- przycisk B - Wciśnięcie i przytrzymanie przycisku powoduje kolorowanie modelu w zakresie pędzla.

Menu:

- **Modification mode** - pozycja typu „horyzontalne menu”. Pozwala zmienić obecny typ modyfikacji pomiędzy kształtowaniem a kolorowaniem.
- **Brush shape** - pozycja typu „horyzontalne menu”. Pozwala zmienić obecny kształt pędzla pomiędzy kulą i sześciianem dla trybu modyfikacji

8.5. TRYB EDYCJI MODELU

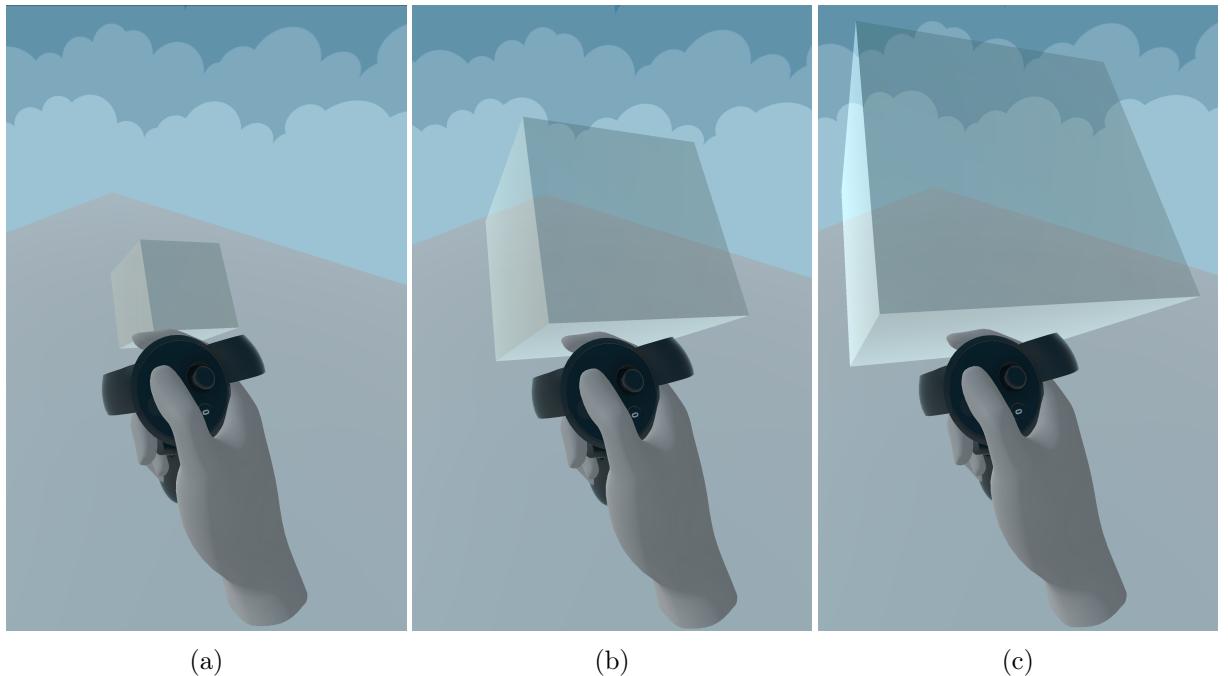


(a) Sfera

(b) Sześciian

Rysunek 8.18: Rodzaje pędzla w trybie edycji modelu

- **Brush size** - pozycja typu „suwak”. Pozwala zmienić rozmiar pędzla.



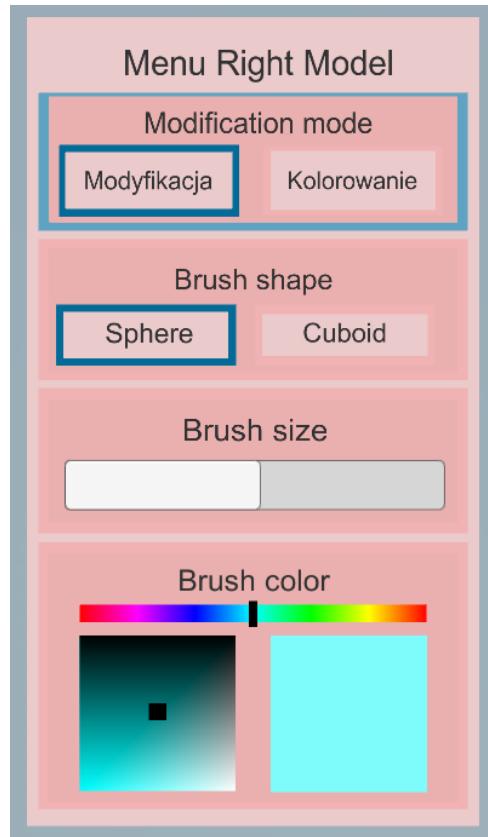
(a)

(b)

(c)

Rysunek 8.19: Rozmiary pędzla w trybie edycji modelu

- **Brush Color** - pozycja typu „kolor”. Pozwala zmienić kolor pędzla dla trybu kolorowania.



Rysunek 8.20: Menu prawej ręki w trybie edycji modelu

9. Podsumowanie

Wynikiem pracy jest aplikacja będąca narzędziem do projektowania scen 3D z użyciem zestawu Oculus Rift. Aplikacja pozwala na tworzenie trójwymiarowych modeli, które następnie można ustawić na przygotowanym wcześniej terenie. Kształty modeli i terenu są generowane za pomocą algorytmu maszerujących sześciianów. Użytkownik ma do wykorzystania paletę narzędzi, które pozwolą mu na otrzymanie pożądanego efektu.



Rysunek 9.1: Przykładowa scena

9.1. Wnioski

Najbardziej kosztownym elementem generowania obiektu okazało się przesyłanie danych między GPU, a CPU. Z tego powodu większość prób optymalizacji obliczeń nie powodowała znaczących różnic. Jedynym sposobem na zwiększenia wydajności byłaby rezygnacja z przesyłania wyniku na CPU i renderowanie siatek trójkątów polegając na danych już będących w buforze. Jednak takie podejście uniemożliwia wykorzystanie do tego narzędzi z silnika Unity i zmusza do samodzielnego implementacji shaderów znacznie zwiększając wymagany nakład pracy.

Aplikacja zapewnia płynne działanie dla siatki o wymiarach 40 na 40 na 40 próbek. Przy większych wymiarach jest znaczący spadek ilości klatek na sekundę. Uzyskano docelową dokładność, ale nie zawsze okazuje się ona być wystarczająca. Aby dorównać produktom dostępnym na rynku, należałoby zwiększyć gęstość siatki próbek lub wprowadzić siatkę dynamiczną[4], zapewniającą większą dokładność tylko tam, gdzie jest ona wymagana.

Zaprojektowanie interfejsu dla urządzeń VR jest problematyczne. Jeśli liczba możliwych akcji jest mała, to wystarczy przypisać je do konkretnych przycisków. W przeciwnym przypadku należy je pogrupować i udostępnić użytkownikowi w postaci otwieranych menu. Przeniesienie z aplikacji desktopowych rozbudowanego, stale widocznego panelu opcji nie sprawdza się, ponieważ przeszkadza on użytkownikowi blokując pole widzenia.

Jeśli dla danego narzędzia, jako miejsce przeprowadzanych modyfikacji, przypiszemy pozycję kontrolera, to nie będzie to intuicyjne dla użytkownika, bo ręka zasłania miejsce nanoszenia zmian. Z przeprowadzonych testów wynika, że użytkownik oczekuje też większego zasięgu wykonywania akcji niż długość ręki. Najlepiej sprawdzają się wskaźniki czy promienie działające na teoretycznie nieograniczoną odległość lub imitowanie trzymania narzędzi, które służą jako przedłużenie ręki i pozwalają na wykonywanie operacji z większą precyzją.

Pytaniem pozostaje czy powstała aplikacja i podobne do niej oprogramowanie faktycznie jest w stanie zwiększyć wydajność pracy czy nadaje się tylko do wykorzystania w celach rozrywkowych.

Mamy też świadomość, że projekt stworzony w ramach pracy inżynierskiej nie może konkurować z aplikacjami rozwijanymi przez zespoły profesjonalistów z takich firm jak Google czy Facebook. Chcieliśmy stworzyć prototyp, który będzie spełniał przyjęte założenia, a przy tym poszerzyć swoją wiedzę w obszarze grafiki 3D.

9.2. Dalszy rozwój

Wyznaczono priorytety dla dalszego rozwoju powstałej aplikacji:

- Zaimplementowanie opisanej wcześniej zmiany sposobu renderowania obiektu, co pozwoli na ograniczenie wymaganej mocy obliczeniowej, co prowadzi do możliwości zwiększenia szczegółowości generowanych modeli. Dopiero po wprowadzeniu tych zmian sensowne staną się dodatkowe optymalizacje związane z implementacją algorytmów odpowiedzialnych za modyfikację próbkowych danych i tworzenie na ich podstawie obiektów.
- Aby aplikacja mogła wyjść z fazy prototypu stać się użytecznym narzędziem, należy zapewnić możliwość używania modeli i scen poza samą aplikacją. Składają się na to eksportowanie i importowanie modeli do powszechnie używanych formatów, a także integracja z platformami to udostępniania efektów pracy.
- W procesie testowania uwagę zwrócono na brak możliwości cofnięcia ostatnich zmian. Brak takiej funkcjonalności stwarza zagrożenie utraty wykonanej pracy, dlatego taka opcja powinna zostać dodana zarówno w modułach generowania terenu i modeli jak i do rozmieszczania obiektów w module edycji sceny.

Bibliografia

- [1] William E. Lorensen and Harvey E. Cline, *Marching cubes: A high resolution 3D surface construction algorithm*, 1987.
- [2] Timothy S. Newman and Hong Yi, *A survey of the marching cubes algorithm*, 2006.
- [3] Scott Schaefer and Joe Warren, *Dual Contouring: "The Secret Sauce"*
- [4] Allamandri, Fabio, Paolo Cignoni, Claudio Montani and Roberto Scopigno, *Adaptively Adjusting Marching Cubes Output to Fit A Trilinear Reconstruction Filter*, 1998
- [5] *What is Virtual Reality?*, <https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html> (dostęp 05.01.2019)
- [6] *Aplikacje VR w Architekturze*, <https://epicvr.pl/pl/vr-architekturze-wizualizacje-vr-wnetrzavr/> (dostęp 16.01.2019)
- [7] *Virtual Reality in Medicine*, <https://thinkmobiles.com/blog/virtual-reality-medicine/> (dostęp 11.01.2019)
- [8] *Virtual Reality in Military*, <https://thinkmobiles.com/blog/virtual-reality-military/> (dostęp 11.01.2019)
- [9] *Virtual Reality in Gaming*, <https://thinkmobiles.com/blog/virtual-reality-gaming/> (dostęp 11.01.2019)

Wykaz stosowanych skrótów i oznaczeń

VR - (ang. virtual reality) wirtualna rzeczywistość

GPU - (ang. graphics processing unit) procesor graficzny

CPU - (ang. central processing unit) procesor

Bryła - dynamiczna bryła ograniczona prostopadłościanem, obliczana za pomocą algorytmu maszerujących sześciianów.

Model - bryła, służąca za wzór dla obiektów.

Pędzel modelu - ograniczona przestrzeń wokół kontrolera, o wybranym kształcie, w granicach której zostają wykonane modyfikacje modelu.

Obiekt - kopia wybranego modelu o określonej skali, pozycji i rotacji

Teren - bryła, z wyróżnioną jedną stroną, zwaną powierzchnią terenu, na której aplikowane są modyfikacje.

Układ współrzędnych terenu - dwuwymiarowy układ współrzędnych powierzchni terenu. Każdy punkt układu ma przypisaną wysokość i kolor.

Pędzel terenu - ograniczona powierzchnia (figura płaska) wokół kontrolera, zwrócona zawsze równolegle do układu współrzędnych terenu. W granicach jej rzutu na powierzchnię terenu są aplikowane modyfikacje terenu.

Scena - teren wraz z ustawnionymi na nim obiektyami. Każda scena posiada dokładnie jeden teren. Każda scena ma przypisaną listę modeli.

Spis rysunków

3.1	Sala operacyjna w wirtualnej rzeczywistości - https://www.3ders.org/articles/20170127-3d-systems-announces-updated-vr-scenarios-for-surgical-training.html .	14
3.2	Żołnierz podczas ćwiczeń ze sprzętem VR - https://thinkmobiles.com/blog/virtual-reality-military/	15
3.3	Bieżnia do gogli VR Virtuix Omni - http://www.virtuix.com/virtuix-launches-omniverse-esports-competitive-vr-gaming-platform/	16
3.4	Zestaw do symulowania prowadzenia samochodu przystosowany do sprzętu VR - https://www.roadtovr.com/r-craft-vr-motion-simulator-oculus-rift-dirt-rally-sim-racing/	17
3.5	Zestaw Google Cardboard - https://lh3.googleusercontent.com/MAwTQ5qs2ogQg_NSQBQTh3nrG78sMRALh9MGWzvlB5-t63NLtQyI3HBJttOL9Owx5fcE	18
3.6	Zestaw Samsung Gear VR - https://www.amazon.com/Samsung-Controller-SM-R325-International-Version/dp/B07GMC1C13	18
3.7	Zestaw HTC Vive - http://home.mehromah.ir/mehromah-learn/scientific-technology/4204-HTC-will-play.html	19
3.8	Zestaw Oculus Rift - https://www.currys.co.uk/gbuk/tv-and-home-entertainment/gaming/virtual-reality/oculus-rift-touch-bundle-10168251-pdt.html	19
4.1	Kanoniczne orientacje generowanego zestawu ścian - https://en.wikipedia.org/wiki/Marching_cubes	21
5.1	Diagram przypadków użycia modułu menu głównego	26
5.2	Diagram przypadków użycia modułu edycji sceny, część 1	27
5.3	Diagram przypadków użycia modułu edycji sceny, część 2	28
5.4	Diagram przypadków użycia modułu edycji terenu	29
5.5	Diagram przypadków użycia modułu edycji modelu	31
6.1	Medium - https://vrproject.com.pl/oculus-medium/medim	34

SPIS RYSUNKÓW

6.2	Blocks - https://vr.google.com/blocks/	35
6.3	Gravity Sketch - https://angel.co/gravity-sketch-1/jobs	35
6.4	Tilt Brush - https://vrproject.com.pl/tilt-brush/tilt-brush	36
6.5	Quill - https://quill.fb.com	37
6.6	Substance Painter - https://www.allegorithmic.com/blog/substance-painter-oculus-rift-prototype	37
6.7	A-Painter - https://github.com/immersive-web/webvrrocks/pull/87/files	38
7.1	Diagram klas warstwy interfejsu	41
7.2	Diagram klas warstwy logiki, część 1	42
7.3	Diagram klas warstwy logiki, część 2	43
7.4	Struktura zapisywanych plików	46
7.5	Diagram stanów dla obliczeń na GPU	47
8.1	Oculus touch controllers - oznaczenia przycisków - https://developer.oculus.com/documentation/unity/latest/concepts/unity-ovrinput/	50
8.2	Przykładowy wygląd menu	51
8.3	Kolejne etapy poruszania się w menu	52
8.4	Przykład pozycji typu „prostego”	53
8.5	Przykład pozycji typu „horizontalne podmenu”	53
8.6	Przykład pozycji typu „suwak”	53
8.7	Przykład pozycji typu „kolor”	54
8.8	Widok gracza po wyłączeniu aplikacji	54
8.9	Menu trybu głównego	55
8.10	Lewe menu w trybie edycji sceny	56
8.11	Prawe menu w trybie edycji sceny	57
8.12	Prawe menu w trybie edycji sceny przy zaznaczonym obiekcie	58
8.13	Lewe menu w trybie edycji terenu	59
8.14	Rodzaje pędzla w trybie edycji terenu	60
8.15	Przykładowe rozmiary pędzla w trybie edycji terenu	60
8.16	Menu prawej ręki w trybie edycji terenu	61
8.17	Lewe menu w trybie edycji modelu	62
8.18	Rodzaje pędzla w trybie edycji modelu	63
8.19	Rozmiary pędzla w trybie edycji modelu	63
8.20	Menu prawej ręki w trybie edycji modelu	64

SPIS RYSUNKÓW

9.1 Przykładowa scena	65
---------------------------------	----

Spis załączników

1. płyta CD