AKADEMIA TECHNICZNO-HUMANISTYCZNA
W BIELSKU - BIAŁEJ

WYDZIAŁ BUDOWY MASZYN I INFORMATYKI

PRACA DYPLOMOWA

Inżynierska Nr /KIiA

Mateusz Stępień
Nr albumu: 049831
Kierunek: Informatyka
Specjalność: Telekomunikacja i sieci komputerowe

Temat: Wizualizacja obiektów architektonicznych z zastosowaniem wirtualnej rzeczywistości

Zakres pracy:

1. Analiza wymagań funkcjonalnych i niefunkcjonalnych
2. Wybór technologii i narzędzi realizacji
3. Odwzorowanie obiektu architektonicznego w model 3D
4. Projekt aplikacji do prezentacji modelu
5. Edycja pisemna pracy

Symbol i kategoria pracy dyplomowej: I – program komputerowy

KATEDRA INFORMATYKI I AUTOMATYKI

Promotor: dr Marcin Bernaś

.....
Podpis i pieczątka
Kierownika Katedry

Bielsko-Biała, rok akademicki 17/18

Spis treści:

1. Wstęp

Informatyka to współcześnie jedna z najszybciej rozwijających się nauk, która posiada wiele rozbudowanych poddziedzin jak grafika komputerowa czy przetwarzanie danych. Pojawienie się komputerów umożliwiło przetwarzanie informacji w postaci cyfrowej na niespotykaną dotąd skalę. Dzięki temu informatyka wspiera nie tylko postęp naukowy w obszarach nauk ścisłych jak fizyka, chemia czy biologia, ale także przeniknęła do życia codziennego stając się jego nieodłączną częścią. Informatyka mocno wpłynęła na sposób przekazu informacji oraz proces prezentacji wyników analiz i symulacji w wielu branżach, takich jak architektura czy budownictwo. Wykorzystując coraz dokładniejsze i realistyczne metody bazujące na grafice komputerowej możliwe jest zaprezentowanie z dużą dokładnością obiektów, jeszcze przed ich skonstruowaniem. W głównej mierze przyczynił się do tego rozwój grafiki trójwymiarowej (3D) i narzędzi pozwalających na generowanie coraz to bardziej realistycznych i szczegółowych obrazów. Postęp technologiczny wymaga od twórców stosowania coraz to nowszych technik i narzędzi do tworzenia wizualizacji, aby opracowane modele generowały bardziej atrakcyjne i realistyczne obrazy. Graficzna prezentacja coraz częściej zostaje wzbogacona o technologie wirtualnej rzeczywistości, która pozwala w nowy sposób zaprezentować trójwymiarową przestrzeń, tworząc wrażenie immersji z wirtualnym środowiskiem. Nowe technologie i narzędzia wymagają od twórców nie tylko umiejętności graficzno artystycznych, ale także szerokiej wiedzy z zakresu informatyki pozwalającej stworzyć wizualizację, a także w odpowiedni sposób ją zaprezentować najczęściej w formie odpowiednio zbudowanej aplikacji wykorzystując zestaw VR.

Celem niniejszej pracy jest opracowanie aplikacji realizującej wizualizację obiektów architektonicznych przy zastosowaniu mobilnego systemu wirtualnej rzeczywistości. Aplikacja ma umożliwić lepszą prezentację stworzonej przestrzeni i pozwolić użytkownikowi na immersje z prezentowanym środowiskiem. Dodatkowym celem pracy jest przedstawienie kompletnego procesu tworzenia nowoczesnej wizualizacji z wykorzystaniem technologii wirtualnej rzeczywistości. Proces powinien zawierać studium nad sposobem testowania oraz optymalizacji aplikacji zarówno na etapie tworzenia samej aplikacji oraz graficznych zasobów wykorzystanych do budowy wirtualnej przestrzeni. Zaprezentowany model tworzenia wizualizacji obejmuje zarówno tworzenie geometrii trójwymiarowych modeli wraz z teksturami, które oddają realny wygląd i zachowanie danej powierzchni jak i implementację utworzonych zasobów do silnika gry (Unity), gdzie utworzono prezentowaną przestrzeń wraz z systemem poruszania się użytkownika. Docelowo utworzone środowisko ma zostać zaimplementowane w systemie wirtualnej rzeczywistości. Zadanie to wymaga połączenia odpowiedniej wiedzy i umiejętności praktycznych zarówno z zakresu grafiki 2D i 3D jak i tworzenia aplikacji wykorzystującej wirtualną rzeczywistość. Wymagana jest także znajomość odpowiedniego rozbudowanego oprogramowania, jakim jest program Blender i Unity 3D oraz procesów związanych z wizualizacją i renderowaniem trójwymiarowego środowiska.

Zbudowana aplikacja może zostać uruchomiona przy wykorzystaniu smartfonu i mobilnego zestawu VR, dzięki czemu nie będzie wymagany wyspecjalizowany sprzęt oraz skomplikowany proces jego konfiguracji.

W ramach pracy, na bazie zaprojektowanego modelu 3D, zostaną przedstawione dobre praktyki wykorzystywane przy projektowaniu aplikacji dla potrzeb opracowania środowisk wirtualnej rzeczywistości.

2. Grafika komputerowa

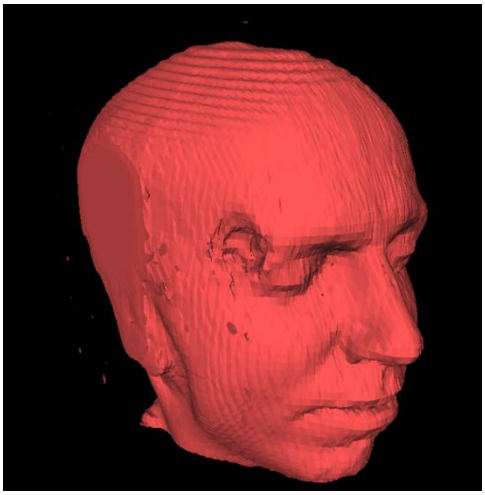
Grafika komputerowa to bardzo obszerny dział informatyki zajmujący się tworzeniem obrazów, obiektów czy wizualizacji zarówno artystycznych jak i rzeczywistych przy pomocy komputera.

Idea grafiki komputerowej narodziła się pod koniec lat pięćdziesiątych. Początkowo dziedzina ta realizowana była przy pomocy kosztownych i trudnych w obsłudze maszynach jednak zainteresowanie przeróżnych instytucji oraz szerokie możliwości wykorzystania grafiki komputerowej zarówno w dziedzinach zawodowych jak i rozrywkowych wpłynęło na jej znaczną popularyzację i rozwój. Dziś trudno nam sobie wyobrazić funkcjonowanie wielu dziedzin życia bez wykorzystania grafiki komputerowej. Stosujemy ją w większości programów użytkowych, jako interfejs użytkownika zawierający system okien oraz ikon umożliwiających zarządzanie równoczesnymi czynnościami a także możliwości wskazywania i wybieranie poszczególnych opcji [6]. Kolejnym obszarem wykorzystania grafiki jest nauka i biznes gdzie możemy tworzyć zaawansowane wykresy funkcji matematycznych i ekonomicznych oraz medycyna gdzie grafika coraz większą rolę odgrywa w diagnostyce medycznej [6]. W projektowaniu wspomaganym komputerowo (CAD) użytkownik korzysta z grafiki interakcyjnej do projektowania elementów i systemów mechanicznych [6]. Oczywiście branża rozrywkowa również bardzo mocno korzysta z grafiki komputerowej zaczynając od przemysłu filmowego gdzie tworzone są przeróżne efekty czy pełnometrażowe animowane filmy na grach komputerowych kończąc. I to właśnie wykorzystanie grafiki w kinematografii i grach przyczyniło się najbardziej do ogromnego rozwoju oprogramowania służącego do generowania realnych i foto-realistycznych obrazów wykorzystywanych powszechnie w wielu aspektach między innymi w celu wizualizacji obiektów architektonicznych, co pokazano w niniejszej pracy.

Głównym celem grafiki jest generowanie obrazów, dlatego głównym kryterium klasyfikacji tej dziedziny jest technika ich tworzenia. Wyróżniamy grafikę rastrową, w której obraz budowany jest przy pomocy dwuwymiarowej tablicy pikseli oraz grafikę wektorową, w której obraz rysowany jest przy pomocy odcinków i figur geometrycznych. Grafikę komputerową możemy podzielić tak że z uwagi na sposób przetwarzania danych a wyróżniamy tu grafikę 2D - grafika dwuwymiarowa, grafikę 2,5D - grafikę dwuwymiarową z użyciem warstw oraz grafikę 3D - grafikę trójwymiarową zajmującą się głównie wizualizacją obiektów trójwymiarowych [6]. Z uwagi na charakter pracy ten rodzaj grafiki opisany zostanie szczegółowo w kolejnych podrozdziałach w celu omówienia zastosowanych w niej technik i pojęć.

2.1 Grafika trójwymiarowa

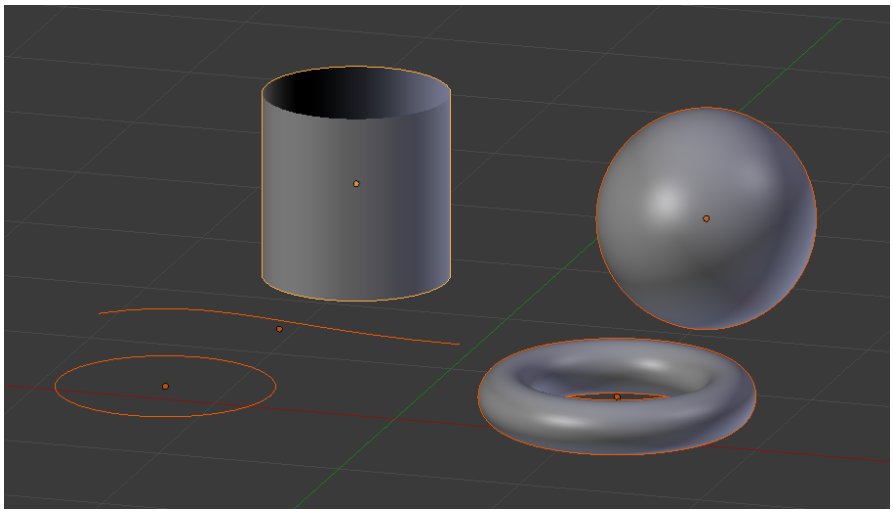
Grafika 3D opiera się na przedstawieniu obiektów w przestrzeni trójwymiarowej stosując zazwyczaj kartezjański układ współrzędnych składający się z trzech osi: X, Y oraz Z. Obiekty trójwymiarowe mogę być reprezentowane na kilka sposobów. Obiekty wokselowe (Woksel z ang. volumetric picture element) są to trójwymiarowe obiekty stworzone przy pomocy wokseli – elementarnych sześcianów, czyli najmniejszych elementów przestrzeni trójwymiarowej (rys. 1) [6].



Rysunek 1. Przedstawienie grafiki opartej na wokselach wielokątami, za pomocą algorytmu marching cubes (źródło: https://pl.wikipedia.org/wiki/Woksel#/media/File:Marchingcubes-head.png)

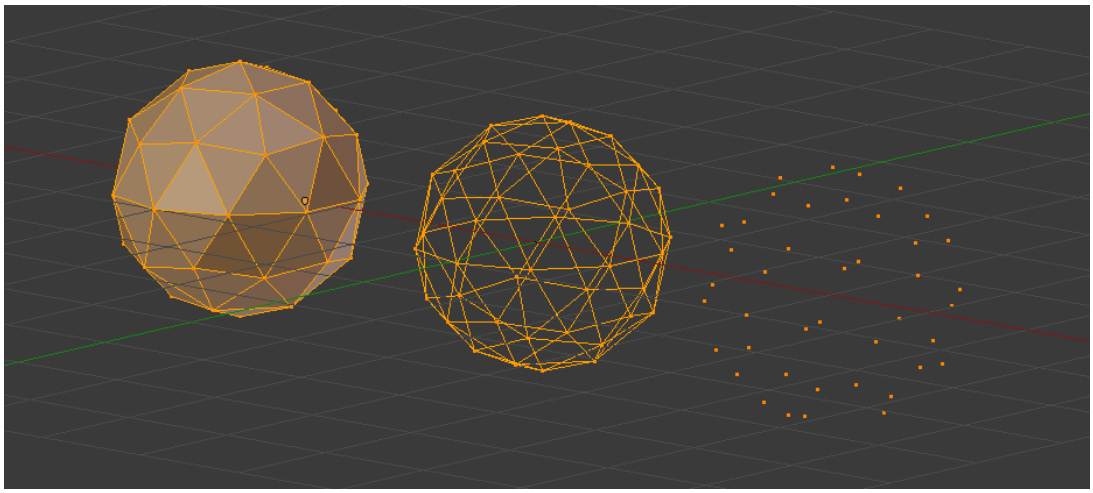
Metodę tą używa się głównie do przedstawienia i analizy danych medycznych lub naukowych gdzie ważne są elementy znajdujące się wewnątrz obiektów trójwymiarowych. Z tego sposobu reprezentacji obiektów korzystają najczęściej urządzenia tomograficzne [6].

Jednak grafika trójwymiarowa częściej opiera się na tworzeniu obiektów przy pomocy opisów matematycznych lub siatek wielokątów. Opis matematyczny polega na opisywaniu obiektów przy pomocy równań. Mogą to być przykładowo parametryczne powierzchnie Béziera czy krzywe NURBS (z ang. Non-Uniform Rational B-Spline), które często tworzą bazę elementów wykorzystywanych przy tworzeniu bardziej zaawansowanej geometrii obiektów 3D (rys. 2) [1][6][11].



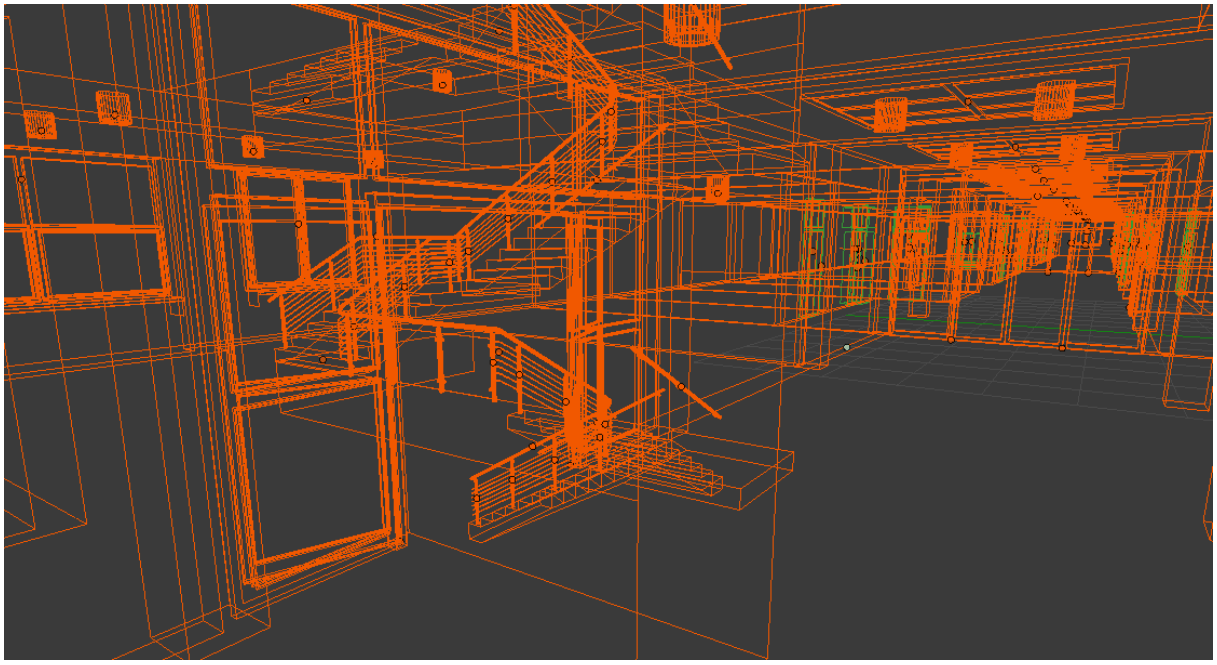
Rysunek 2. Obiekty stworzone za pomocą krzywych i powierzchni NURBS (opracowanie własne)

Najpopularniejszą reprezentacją i metodą tworzenia obiektów w grafice trójwymiarowej jest zastosowanie siatki wielokątów, która jest zbudowana z trójkątów lub czworokątów mających wspólne wierzchołki i krawędzie. W ten sposób możemy tworzyć proste bryły, a także skomplikowane obiekty zwiększając gęstość siatki wielokątów. Siatka obiektów jest reprezentowana przez wielokąty (ang. polygons), wierzchołki (ang. verticles) i krawędzie (ang. edges) (rys. 3)[1][11].



Rysunek 3. Siatka obiektów składająca się z wierzchołków, krawędzi i wielokątów (opracowanie własne)

Obiekty architektoniczne tworzone w celu realizacji tematu pracy zostały wykonane głównie przy pomocy wyżej opisanej metody siatek wielokątów, co prezentuje rysunek 4.



Rysunek 4. Siatka modeli 3D stworzonych na potrzeby realizacji wizualizacji (opracowanie własne)

Grafika trójwymiarowa ma bardzo szeroki wachlarz zastosowań. Przede wszystkim gry komputerowe i branża rozrywkowa opiera się w dużym stopniu na wykorzystaniu tej dziedziny przy tworzeniu treści. Współcześnie ciężko nam sobie wyobrazić wysokobudżetową grę komputerową bez zaawansowanych i realistycznych modeli i efektów 3D. Przemysł filmowy wykorzystuje techniki grafiki 3D do tworzenia efektów specjalnych czy całych pełnometrażowych animowanych filmów. Oczywiście wykorzystanie grafiki trójwymiarowej nie kończy się na celach rozrywkowych. Możemy jej używać do reprezentacji wykresów matematycznych, fizycznych czy symulowaniu zachodzących zjawisk. Medycyna wykorzystuję grafikę 3D w diagnostyce. Dzięki grafice trójwymiarowej możemy wykonywać przeróżne symulacje, wizualizacje i szkolenia [6][11]. Przykładami mogą być wizualizacje architektoniczne gdzie dzięki rozmieszczeniu obiektów w trójwymiarowej przestrzeni możemy pokazać wizję architekta i zaprezentować przestrzeń i wnętrze projektowanego obiektu.

Warto zaznaczyć, że każdy z wcześniejszych wymienionych przykładów zastosowania grafiki 3D może być rozszerzony o wirtualną rzeczywistość, aby jeszcze skuteczniej zasymulować, odwzorować i zaprezentować tworzone obiekty, oraz aby zwiększyć immersje dostarczając odbiorcom dodatkowych wrażeń i bodźców jak to ma miejsce chociażby w grach komputerowych wykorzystujących system wirtualnej rzeczywistości.

2.2 Wirtualna rzeczywistość

Celem pracy jest wykorzystanie wirtualnej rzeczywistości, dlatego podstawy tej technologii zostaną omówione w tym podrozdziale. Wirtualna rzeczywistość to interaktywne, generowane komputerowo doświadczenie, które odbywa się w symulowanym środowisku bazującym na odpowiednich efektach audio wizualnych [17]. Dziedzina ta jest prężnie

rozwijana od prawie 50 lat, dzięki czemu uzyskuje się coraz to nowsze sposoby do odtworzenia, prezentacji i interakcji z wirtualnym środowiskiem. Ostatnie kilka lat rozwoju tej technologii pozwoliło na sprawne jej wykorzystywanie. Zagadnienie wirtualnej rzeczywistości możemy rozpatrywać w dwóch formach: wirtualną rzeczywistość (ang. virtual reality – VR) gdzie tworzymy i symulujemy środowisko, w którym znajduje się odbiorca oraz rozszerzoną rzeczywistość (ang. Augmented Reality – AR) gdzie rzeczywiste środowisko jest łączone z elementami wygenerowanymi komputerowo, najczęściej ma to miejsce poprzez nakładanie na obraz z kamery modeli i elementów 3D. Technologia VR jest współcześnie głównie wykorzystywana w celu dostarczenia rozrywki jednak dzięki możliwości odwzorowania świata rzeczywistego technologia ta coraz częściej może być wykorzystywana w branży użytkowej gdzie tworzone są symulacje i szkolenia, w których wirtualne scenariusze przebiegają w bardzo trudnych, czy nawet ekstremalnych i nietypowych warunkach, ale są całkowicie bezpieczne dla zdrowia i życia. VR z sukcesem jest wykorzystywany także w handlu, turystyce i mediach społecznościowych oraz z powodzeniem może zostać stosowany do realizacji wizualizacji architektonicznych, co pokazuję w niniejszej pracy [17].

Współczesne systemy wirtualnej rzeczywistości bazują na wykorzystaniu gogli VR wyposażonych w odpowiednie soczewki oraz wyświetlacz (lub wyświetlacze) zakrywający w całości pole widzenia użytkownika, aby zmniejszyć wpływ bodźców środowiska zewnętrznego i wprowadzić wrażenie otaczającej przestrzeni. Dostosowana aplikacja renderuje stereoskopowy obraz odpowiednio z perspektywy prawego i lewego oka (rys. 5). Każdy z renderowanych obrazów jest zniekształcany (poddawany dystorsji beczkowej z ang. barrel distortion), aby przeciwdziałać zniekształceniu wynikającemu z użycia soczewek (dystorsji poduszkowej z ang. pincushion distortion). Rezultatem jest stereoskopowy obraz oddający wrażenie głębi wirtualnej przestrzeni. Celem wirtualnej rzeczywistości jest stymulacja wielu zmysłów użytkownika. Dlatego zestawy VR oprócz odpowiedniego wyświetlania obrazu są wyposażone w zestawy słuchawkowe oraz sensory i kontrolery, które śledzą ruchy użytkownika, angażując tym samym większą ilość zmysłów, co pogłębia odczucie immersji.

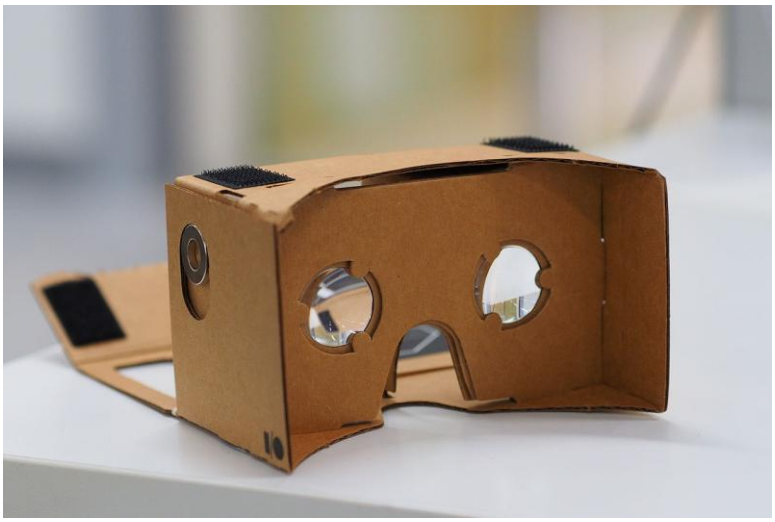
Rysunek 5. Renderowanie obrazu w systemie wirtualnej rzeczywistości (źródło: opracowanie własne)

Największy wzrost popularności technologii VR nastąpił w 2010 roku, gdy Palmer Luckey zaprojektował pierwszy prototyp gogli wirtualnej rzeczywistości Oculus Rift [17]. Zestaw Oculus Rift posiada dwa wyświetlacze OLED dla każdego oka, każdy o rozdzielczości 1080 × 1200. Panele te mają częstotliwość odświeżania 90 Hz i zapewniają widzenie stereoskopowe. Gogle posiaduję wbudowane regulowane soczewki zapewniające szerokie pole widzenia oraz zintegrowane słuchawki. Urządzenie posiada funkcję śledzenia położenia i rotacji za pomocą żyroskopu, akcelerometru i magnetometru a także kamery podłączanej do komputera, która działa z użyciem fal podczerwonych [17]. Innym popularnym zestawem wirtualnej rzeczywistości jest HTC Vive, w którego skład oprócz gogli wchodzą kontrolery i stacje bazowe (rys. 6). Zestaw wykorzystuje technologię śledzenia o nazwie Lighthouse, która za pomocą zamontowanych na ścianie stacji bazowych i promieni podczerwonych śledzi położenie gogli i kontrolerów.

Rysunek 6. Zestaw HTC Vive (źródło: https://en.wikipedia.org/wiki/HTC_Vive#/media/File:Vive_pre.jpeg)

Jednak wyżej opisane technologie posiadają pewne wady przede wszystkim jest to wysoka cena wyżej wymienionych zestawów VR oraz wymagana wysoka wydajność sprzętu. Musimy też pamiętać, że gogle te wymagają przewodowego połączenia z komputerem, co znacznie ogranicza mobilność użytkowników. W przypadku HTC Vive wymagane jest także przeprowadzenie dość skomplikowanej procedurę konfiguracji zestawu w tym mapowania obszaru, w którym będzie poruszał się użytkownik.

Istnieje też inne rozwiązanie VR, gdzie do wyświetlania wirtualnego środowiska używamy w pełni mobilnych gogli oraz smartfonu. W implementacji wirtualnej rzeczywistości na platformach mobilnych telefon pełni rolę wyświetlacza i śledzi zmiany położenia gogli przy pomocy akcelerometru i żyroskopu. Jednym z pierwszych popularnych zestawów mobilnych VR był Samsung Gear VR kompatybilny z flagowymi smartfonami firmy Samsung. Gogle Gear VR posiadają wbudowany kontroler i łączą się ze smartfonem przy pomocy kabla usb-c lub micro-USB. Inną popularną platformą do mobilnego zastosowania wirtualnej rzeczywistości jest Google Cardboard (rys. 7). Zestaw składa się z gogli wykonanych z prostych i tanich komponentów z wbudowanymi soczewkami i przyciskiem wykorzystującym magnes, który oddziałuje na magnetometr w telefonie, dzięki czemu korzystając z biblioteki udostępnionej przez Google możliwa jest obsługa magnetycznego przełącznika. Inne wersje Cardboard'a zawierają tekturową dźwignie, która po naciśnięciu dotyka ekranu.



Rysunek 7. Google Cardboard (źródło: https://en.wikipedia.org/wiki/Google_Cardboard#/media/File:Assembled_Google_Cardboard_VR_mount.jpg)

Na fali popularności platform Gear VR i Google Cardboard na rynku pojawiło się wiele mobilnych zestawów do zastosowań wirtualnej rzeczywistości przy użyciu telefonu, które są dostosowane do większości dostępnych na rynku smartfonów. Główną zaletą tego rozwiązania jest niewielki koszt zestawu, całkowita mobilność niewymagająca połączenia z komputerem i bardzo duża elastyczność. Mobilne zestawy wirtualnej rzeczywistości nie

wymagają skomplikowanej procedury konfiguracji wystarczy, że umieścimy smartfon z uruchomioną odpowiednią aplikacją wewnątrz gogli. Z uwagi na powyższe zalety w wizualizacji będącej wynikiem tej pracy zdecydowano się na implementację wirtualnej rzeczywistości dostosowanej na platformę mobilną przy użyciu gogli VR firmy Modecom wraz z bezprzewodowym kontrolerem łączącym się z telefonem przez Bluetooth (rys. 8), który umożliwia interakcje i poruszanie się w symulowanym środowisku.

Rysunek 8. Gogle Modecom Volcano Blaze (źródło: opracowanie własne)

Należy zwrócić uwagę, że używanie platform mobilnych do symulowania wirtualnego środowiska niesie za sobą pewne ograniczenia wynikające głównie z mniejszej wydajności telefonów w porównaniu z urządzeniami PC. Ograniczenia te zostały szerzej poruszone w dalszej części pracy.

2.3 Wizualizacja architektoniczna

Wizualizacja jest jednym z głównych elementów prezentacji obiektów architektonicznych zarówno w procesie projektowania jak i w celach marketingowych. Komputerowa wizualizacja architektoniczna jest sposobem przedstawienia obiektów architektonicznych poprzez wykorzystanie grafiki trójwymiarowej w celu uzyskania realistycznej wizji projektu [3].

Tworzenie wizualizacji składa się z kilku zasadniczych etapów. Pierwszym z nich jest stworzenie trójwymiarowego geometrycznego modelu docelowo pokazywanych obiektów i przestrzeni za pomocą programu do modelowania 3D. Modelowanie 3D ze względu na szerokie wykorzystanie i specjalistyczne programy obliczeniowe, których twórcy dążą do stworzenia jak najbardziej efektywnego systemu pozwala na definiowanie kształtu obiektu przy jak najmniejszej liczbie parametrów i umożliwia jego modyfikacje w wybranym

fragmencie. Jak już wspomniano we wcześniejszym rozdziale obiekty te mogą być tworzone na wiele sposobów, wykorzystując głównie krzywe Béziera i powierzchnie NURBS oraz siatki wielokątów.

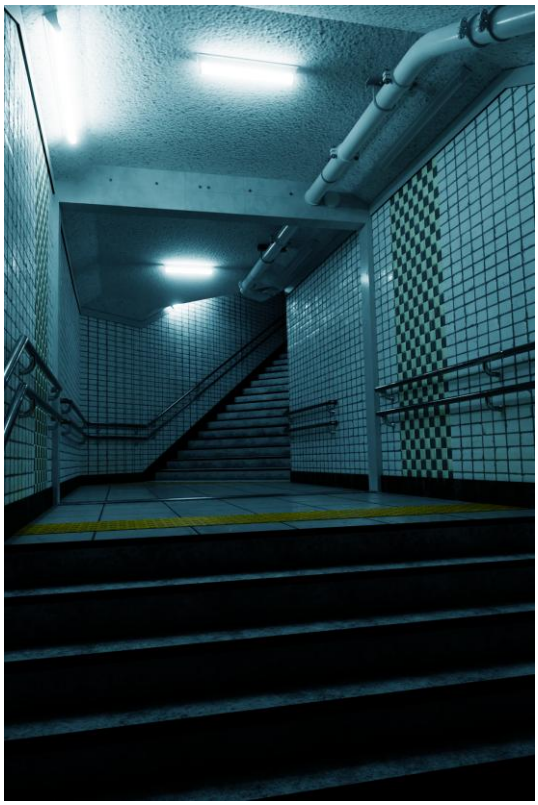
Kolejnym etapem tworzenia wizualizacji jest nakładanie na utworzone modele 3D tekstur ma to na celu głównie zwiększenie realizmu generowanych obrazów. Przypisując tekstury możemy nadać powierzchnią barwę i wzór, właściwości związane z obróbką i wykończeniem oraz cechy związane z odbiciem i przenikaniem światła [3]. Oczywiście zmiany właściwości danej powierzchni są pozorne i widoczne podczas renderingu różnice widzimy poprzez oddziaływanie światła na dany obiekt. Efekt padania promieni na powierzchnie z zastosowaną teksturą metaliczną (metallic map) lub teksturą chropowatości (glossiness) informuje nas o jej rodzaju. Dodatkowo możemy na obiekt nałożyć teksturę wypukłości z przypisaną mapą normalnych (z ang. normal map), która modyfikuje kierunek wektorów normalnych pojedynczych poligonów powierzchni. Poprzez interpretację zmienionych kierunków wektorów normalnych i padanie światła na daną powierzchnie uzyskujemy efekt pozornych wypukłości. Technika ta jest często stosowana, w procesie optymalizacji modeli 3D, aby ograniczyć gęstość siatki danego modelu. Podczas tworzenia modeli wykorzystanych w docelowej wizualizacji wiele detali i szczegółów jest dodane wykorzystując mapy normalnych, aby nadać obiektom bardziej realistycznych cech i zredukować ilość wyświetlanych poligonów.

Następnym kluczowym etapem jest oświetlenie, które jest bardzo istotne dla końcowego efektu wizualizacji. Światło na scenie umożliwia nam postrzeganie barwy, faktury i innych właściwości obiektów. Każda powierzchnia reaguje we właściwy sobie sposób na padające na nią promienie światła, jest to zależne od ustawień przypisanego jej materiału i nałożonych na nią tekstur. Przy padaniu światła na dany obiekt możemy wyróżnić następujące zjawiska: odbicie światła (kierunkowe i rozproszone), przenikanie światła zachodzące dla materiałów przeźroczystych oraz pochłanianie. Luminacja odpowiada jaskrawości dla obiektów emitujących światło i jasności dla obiektów odbijających. Luminacja oświetlonej powierzchni zależy od współczynnika odbicia światła danego materiału oraz ilości i jakości światła, które na nią pada. Luminacja może być modyfikowana przez kąt nachylenia powierzchni w stosunku do źródła światła oraz odległością i kierunkiem patrzenia obserwatora [3].

Ostatnim etapem jest ustawienie kadru i renderowania scen trójwymiarowych. Rendering trójwymiarowej sceny jest wykonywany przez specjalne oprogramowanie – silnik renderujący wykorzystujący algorytmy do obliczania drogi światła i przestrzeni. Dzięki metodzie śledzenia promieni (z ang. Raytracing) możemy uzyskać efekty mocno zbliżone do realnego działania światła. Raytracing polega na analizie przebiegu promienia miedzy obserwatorem a źródłem światła, promień ten nazywany jest promieniem pierwotnym i wywołuje on określone reakcje z obiektami spotkanymi na swojej drodze. W zależności od napotkanej powierzchni generowane są promienie wtórne (odbite i załamane). Innym popularnym modelem oświetlenia jest obliczanie oświetlenia globalnego Global Illumination który jest wykorzystywany w wielu silnikach 3D np. w Unity. Global Illumination bazuje na

oświetlaniu obiektów zarówno przez światło emitowane bezpośrednio ze źródła jak również przez światło odbite od innych obiektów na scenie. [3][9]

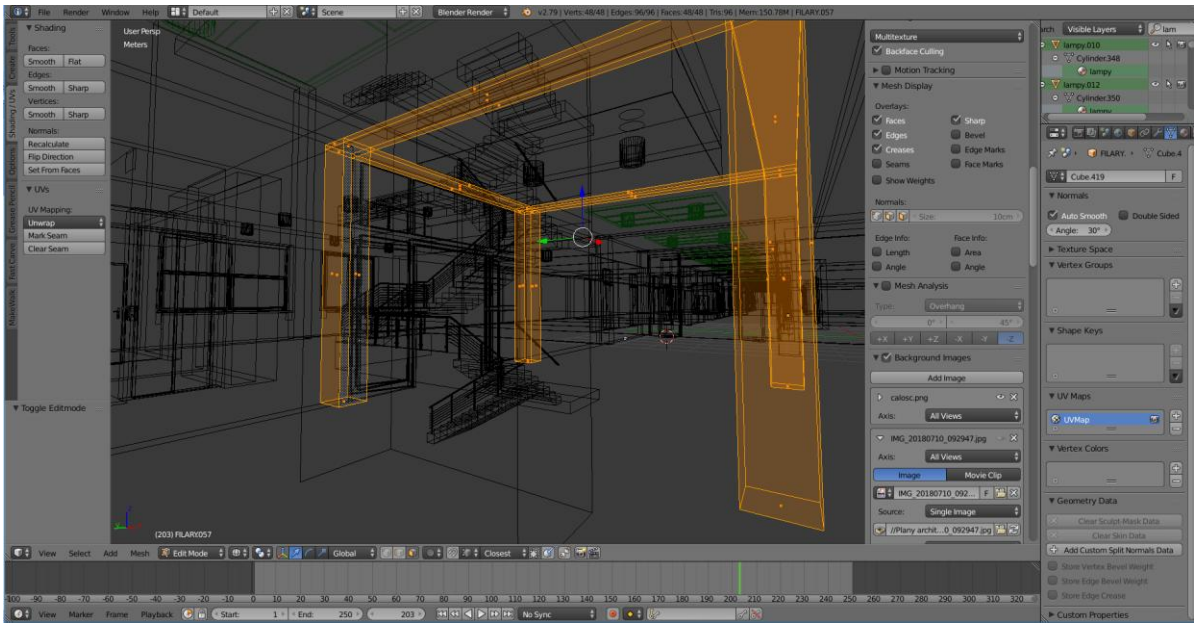
Przechodząc przez wszystkie powyższe kroki możemy przygotować kompletną sceną 3D. Finalną kwestią jest ustawienie kadru kamery, nałożenie dodatkowych efektów wizualnych takich jak np. balans kolorów czy korekcja kontrastu lub jasności oraz wykonanie renderowania, czyli przekształcenia sceny złożonej z trójwymiarowych modeli na wyjściowy obraz dwuwymiarowy. W zależności od zastosowanego oprogramowania scena może być renderowana w czasie rzeczywistym gdzie klatki obrazu generowane są przez silnik na bieżąco (przeważnie dla zapewnienia płynności obrazu przyjmuje się 60 klatek na sekundę), taki sposób renderowania jest stosowany między innymi w silnikach gier. Innym sposobem jest renderowanie ustawionego kadru, jako statyczny wyjściowy obraz (pojedynczą klatkę), co oferuje między innymi silnik renderujący Cycles. Renderując pojedynczy statyczny wyjściowy obraz możemy uzyskać znacznie lepszy wizualny efekt oraz większy realizm z uwagi na dokładniejsze obliczania zachowania światła. Poniżej przedstawiono wizualizację wykonaną w programie Blender, która została wyrenderowana z użyciem renderera Cycles (rys. 9).



Rysunek 9. Wizualizacja architektoniczna podziemnego korytarza metra (opracowanie własne)

3. Technologie i narzędzia

Do przygotowania wizualizacji z wykorzystaniem wirtualnej rzeczywistości będącej istotą tej pracy wykorzystano głównie oprogramowania z trzech kategorii: programu do modelowania 3D, oprogramowania do stworzenia tekstur oraz silnika gry (z ang. game engine), w którym wszystkie stworzone elementy zostały umieszczone i w którym zbudowana została docelowa aplikacja na platformę Android. Do modelowania obiektów trójwymiarowych wykorzystano program Blender 3D. W Blenderze stworzono geometrię wszystkich obiektów znajdujących się na scenie (rys. 10), dostosowując przy tym wektory normalnych oraz przygotowano modele do teksturowania rozwijając UV-mapy powierzchni, czyli mapy informujące jak tekstura ma się zachować na modelu 3D.



Rysunek 10. Modelowanie obiektów w programie Blender 3D (opracowanie własne)

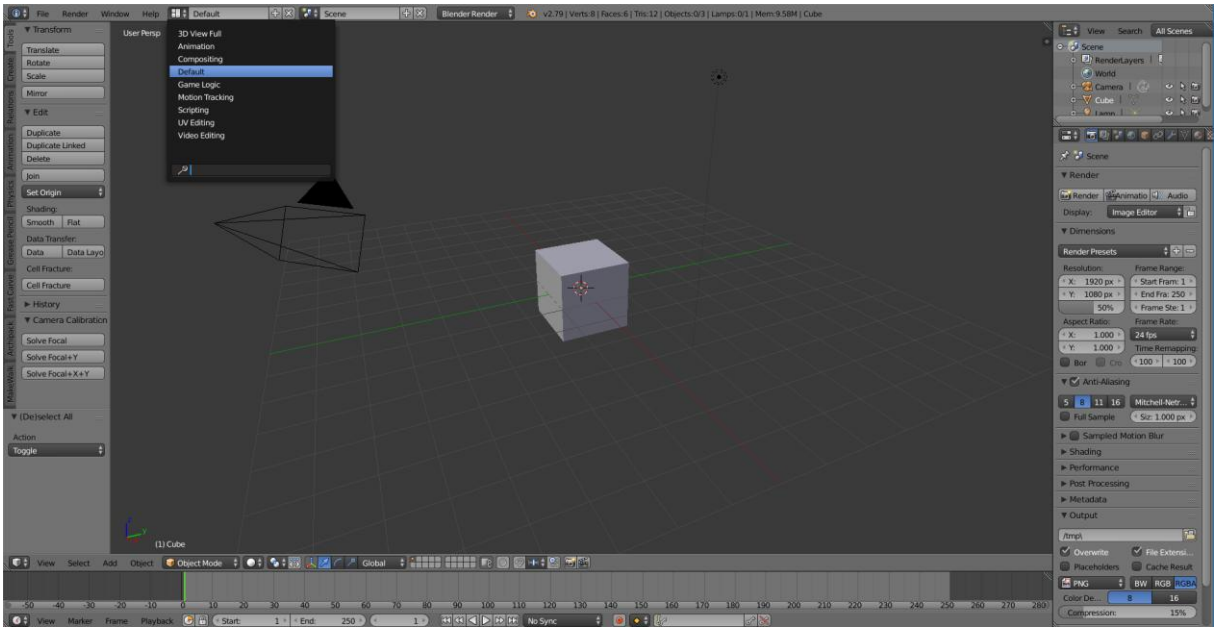
Do kolejnego etapu prac skorzystano z programu Substance Painter, w którym przygotowano podstawowe tekstury koloru (z ang. Base Color) dla modeli oraz dodatkowe tekstury dostosowane do modelu oświetlenia PBR, które odwzorowują właściwości danej powierzchni takie jak mapy normalnych (normal map), teksturę chropowatości (roughness) czy teksturę metaliczną (metallic). Finalnie modele wraz z teksturami umieszczono w silniku gry. Do tego celu wykorzystano silnik Unity 3D, w którym zbudowano scenę z trójwymiarowych modeli oraz ustawiono materiały i światła a także zaimplementowano systemu wirtualnej rzeczywistości wraz z systemem poruszania się użytkownika.

3.1 Blender 3D

Blender jest rozbudowanym multiplatformowym programem do pracy z grafiką 3D. Jest to w pełni darmowe i otwarte oprogramowanie pierwotnie stworzone przez firmę

NeoGeo. Od 2002 roku Blender jest rozwijany przez Blender Foundation a jego głównym programistą jest Ton Roosendaal [7].

Blender stanowi rozbudowane narzędzie i posiada bardzo wiele możliwości wykorzystania od modelowania, animacji i renderingu poprzez efekty specjalne, edycję wideo na silniku gier kończąc. Pomimo bardzo rozbudowanego interfejsu i mnogości funkcji i opcji interfejs Blendera bez zachodzących na siebie i blokujących się okien jest przejrzysty i czytelny. Pozwala na pełną konfiguracje układu okien a także możliwość wybrania jednego z wstępnie zdefiniowanych zestawów (presetów) (rys. 11), co pozwala w prosty sposób dostosować interfejs programu do aktualnie realizowanego zadania.



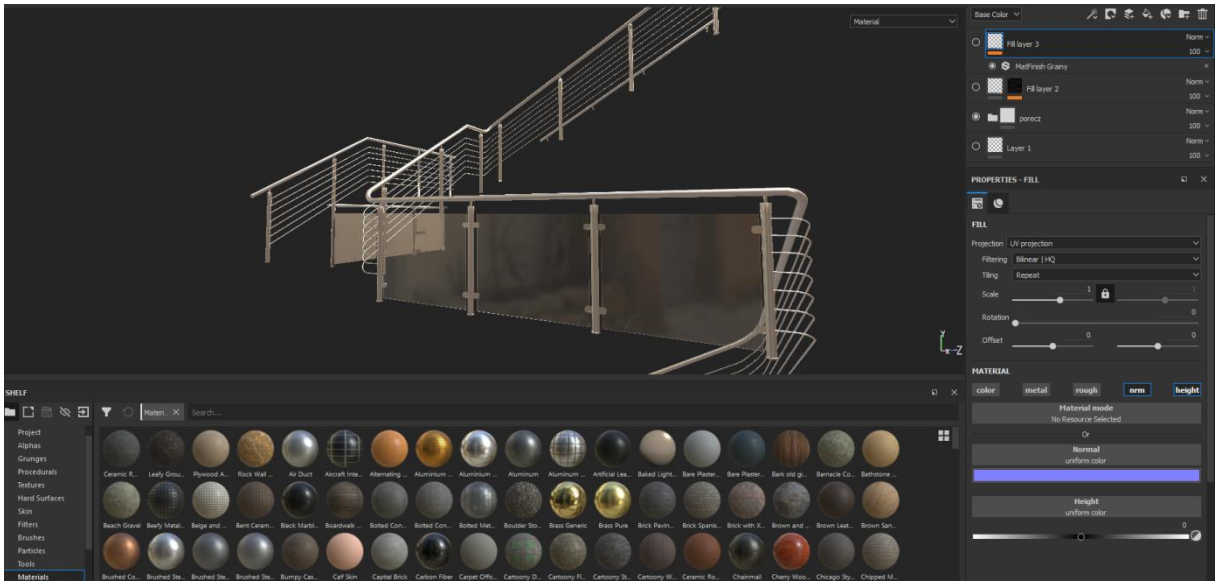
Rysunek 11. Okno programu Blender z domyślnie ustawionym interfejsem (źródło: opracowanie własne)

W celu wykonania modeli na potrzeby wizualizacji głównie korzystano z funkcji blendera przeznaczonych do tworzenia i modyfikacji geometrii a także rozwijania UV map, co zostało dokładnie opisane w dalszej części pracy.

3.2 Substance Painter

Substance Painter jest oprogramowaniem, które umożliwia teksturowanie modeli 3D. Program ten oferuje wiele narzędzi, dzięki którym możliwe jest tworzenie materiałów wykorzystujących model PBR z ang. Physically Based Rendering (rys. 13). Model oświetlenia PBR jest modelem bazującym na fizycznym zachowaniu się światła i materiałów i jest współcześnie wykorzystywany przez większość silników 3D. Substance painter umożliwia oprócz nakładania i malowania tekstur z wykorzystaniem zaawansowanych pędzli i masek także ich wypalenie, czyli przenoszenie informacji zawartych na siatce modelu na tekstury, które następnie są odczytywane przez model cieniujący, dzięki czemu możemy

zmniejszyć gęstość siatki zachowując jej detale. W programie można tworzyć proceduralne materiały, które są dostosowywane do siatki modelu. Substance Painter obsługuje stosowanie zaawansowanych ustawień warstw, które umożliwiają nakładać na siebie różne materiały np. umieszczając na jednej z warstw materiał plastiku natomiast przy pomocy kolejnej zmodyfikować jego barwę czy połysk dodatkowo na każdą z warstw lub jej maskę możliwe jest nakładania dodatkowych efektów, przykładowo generatorów zabrudzeń. Substance Painter obsługuje re-projekcje UV map, co pozwala nam załadować nowy model do projektu (o ile jego geometria jest zbliżona do poprzedniego) i automatycznie zastosować na nim wszystkie utworzone wcześniej materiały i efekty. Wartym uwagi narzędziem jest także zaznaczanie geometrii zaimportowanych modeli 3D, co pozwala tworzyć maski wybierając wielokąty, elementy siatki lub fragmenty UV. Interfejs programu oprócz wielu przyborników i narzędzi zawiera okno podglądu teksturowanego modelu renderowane w czasie rzeczywistym (rys. 12). Program oferuje możliwość ustawienia światła, które jest realizowane przez zastosowanie sferycznego obrazu o wysokiej rozpiętości tonalnej (HDRI), aby zasymulować wygląd obiektu w różnych warunkach oświetlenia.



Rysunek 12. Interfejs programu Substance Painter podczas wykonywania tekstur na potrzeby wizualizacji (źródło opracowanie własne)

Substance Painter posiada także wbudowany renderer Iray, jest to mechanizm renderowania bazującym na śledzeniu promieni opracowany przez firmę Nvidia umożliwiający wykonanie wysokiej jakości reneru wraz z ustawieniem post-procesów.

3.3 Unity

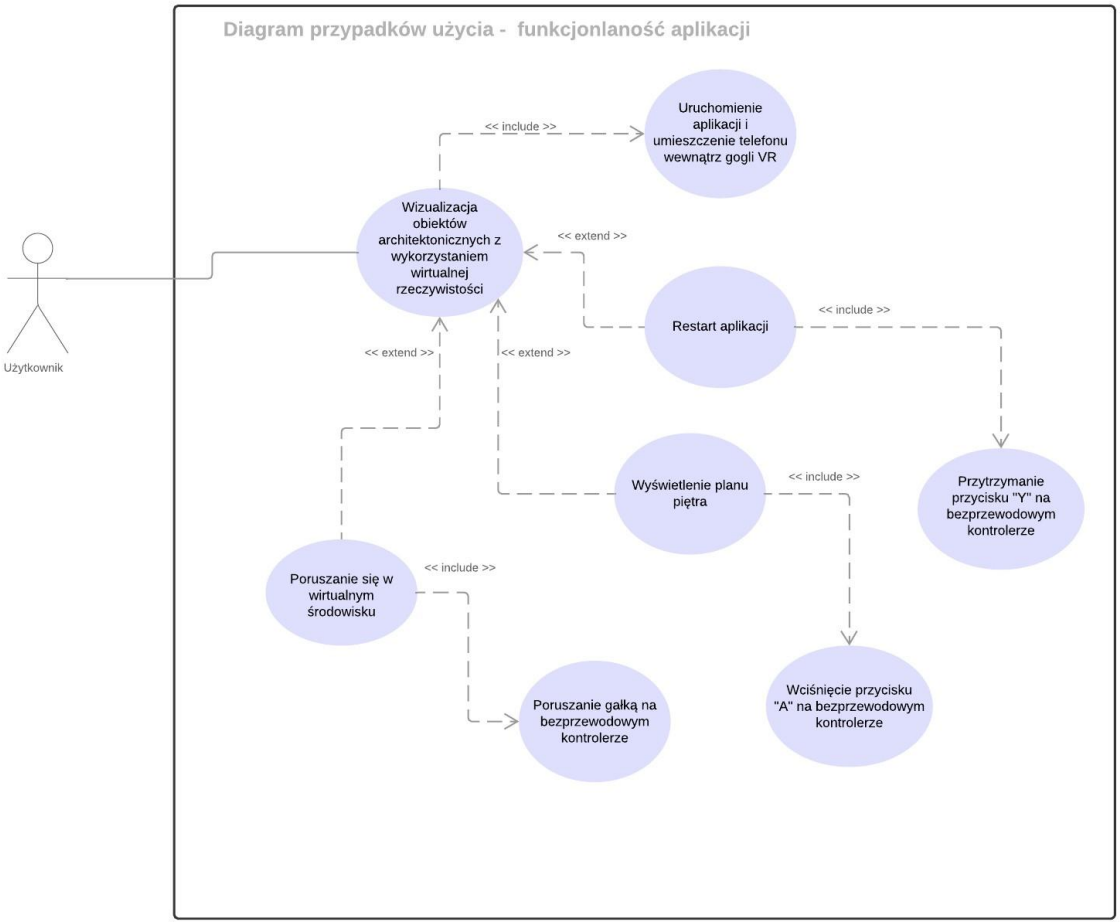
Unity jest wieloplatformowym silnikiem gier opracowanym przez Unity Technologies. Pierwsze wydanie miało miejsce w czerwcu 2005 roku, lecz oprogramowanie wciąż jest rozwijane i wypuszczane są kolejne wersje - silnik obsługuje coraz więcej platform, systemów operacyjnych, bibliotek i wtyczek oraz formatów plików [9]. Z każdą nową wersją poprawiana jest funkcjonalność a także jest powiększany wachlarz dostępnych

narzędzi. Podstawowym wykorzystaniem silnika Unity jest tworzenie gier komputerowych, które również stanowią pewnego rodzaju rozwiniętą formą wizualizacji i prezentacji. Mogą to być zarówno aplikacje bazujące na grafice dwuwymiarowej jak i trójwymiarowej. Silnik Unity wspiera kilka rodzajów API (ang. Application Programming Interface), czyli interfejsów programistycznych pozwalających na komunikację pomiędzy bibliotekami, różnego typu strukturami danych, klasami oraz systemami operacyjnymi [9]. Wspierane są między innymi komponenty Direct3D dla aplikacji z systemem windows, OpenGL ES dla aplikacji na systemy Android i iOS. Unity umożliwia budowę projektu na 27 różnych platform, wśród których znajdują się między innymi Windows, iOS, Mac, platformy konsolowe i mobilne jak Android, co stwarza możliwość szerokiego zastosowania Unity nie tylko do tworzenia gier, ale tak że symulacji, aplikacji funkcjonalnych czy wizualizacji. Unity pozwala między innymi na stosowanie kompresji tekstur, co jest przydatne przy wspomnianej multiplatformowości, ponieważ większość systemów posiada inny format kompresji obrazów zaprojektowanych do przechowywania tekstur. Oprogramowanie oferuje również możliwość pisanie skryptów w języku C#.

Z głównych funkcji silnika wykorzystano podczas tworzenia wizualizacji możliwość umieszczenia w nim wcześniej stworzonych modeli 3D, tworzenie materiałów bazujących na modelu oświetlenia PBR oraz zaawansowane ustawienia oświetlenia, które opisano w dalszej części pracy. W kolejnych etapach dzięki możliwości pisania skryptów zaimplementowano system poruszania się użytkownika, aby w jak najlepszym stopniu pozwolić na interakcję oraz wykorzystano fizykę do utworzenia kolizji z modelami znajdującymi się na scenie. Dodatkowo Unity wspiera technologie VR, co w połączeniu z możliwością budowy aplikacji na różne platformy pozwala zaimplementować i wykorzystać system wirtualnej rzeczywistości oraz zbudować aplikację na odpowiednie urządzenie w tym przypadku smartfon z systemem Android.

4. Funkcjonalność aplikacji

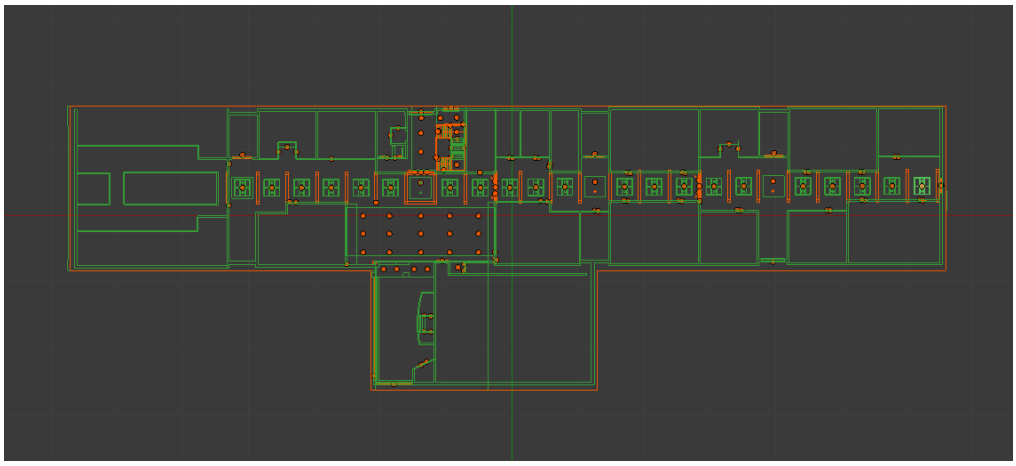
Działanie aplikacji w głównej mierze opiera się na wyświetlaniu trójwymiarowych modeli obiektów architektonicznych z wykorzystaniem systemu wirtualnej rzeczywistości, co pozwala stworzyć realistyczną wizualizację architektoniczną korzystającą z nowoczesnej technologii VR. Dodatkowo użytkownik może poruszać się w wirtualnym środowisku, dzięki czemu może dokładnie zapoznać się z prezentowaną przestrzenią. Poniżej zamieszczono diagram przedstawiający funkcjonalność aplikacji (rys. 13).



Rysunek 13. Diagram przypadków użycia tworzonej aplikacji (źródło: opracowanie własne)

4.1 Główne funkcje i założenia aplikacji

Celem stworzonej aplikacji jest wizualizacja architektoniczna pierwszego piętra budynku L Akademii Techniczno-Humanistycznej w Bielsku-Białej (rys.14). Modele architektoniczne użyte w tej wizualizacji utworzono na podstawie planów architektonicznych oraz wykonanych zdjęć i pomiarów.



Rysunek 14. Model 1 piętra Budynku L ATH (źródło: opracowanie własne)

Założeniem aplikacji jest także wykorzystanie wirtualnej rzeczywistości, aby lepiej zaprezentować stworzoną przestrzeń i umożliwić użytkownikom jak największą immersję. Aplikacja ta została stworzona z myślą o wykorzystaniu mobilnego zestawu VR, dlatego została zbudowana na platformę Android, co pozwala uruchomić ją na dowolnym smartfonie z systemem Android w wersji 5.0 lub nowszej. Do symulacji wirtualnej rzeczywistości z wykorzystaniem tej aplikacji można zastosować dowolny zestaw mobilny VR, co zapewnia dużą elastyczność w doborze sprzętu. Do testów aplikacji użyto gogli Google Card-Board oraz zestawu VR Volcano Blaze firmy MODECOME. Aplikacja obsługuje bezprzewodowy pad łączący się z telefonem przez bluetooth. Główna funkcjonalność aplikacji opiera się na wizualizacji obiektów architektonicznych oraz poruszaniu się użytkownika w wirtualnym środowisku zbudowanym z trójwymiarowych modeli z wykorzystaniem gogli VR i bezprzewodowego kontrolera. Dodatkowo funkcjonalność aplikacji została rozszerzona o funkcję restartu umożliwiającą w łatwy sposób zrestartować uruchomioną aplikację bez konieczności zdejmowania gogli VR oraz funkcję wyświetlania planu wizualizowanego piętra, co pozwala na lepsze zapoznanie się z prezentowanym obiektem. Podczas prac związanych z tworzeniem wyżej opisanej wizualizacji zastosowano także narzędzia i techniki optymalizacyjne, aby zapewnić płynne działanie aplikacji na urządzeniach mobilnych.

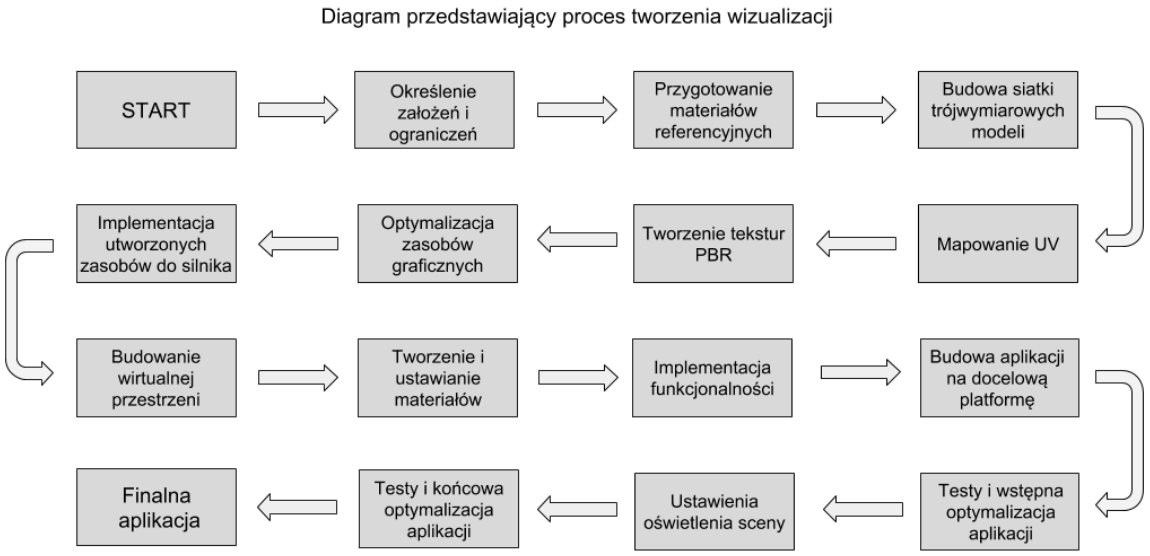
4.2 Ograniczenia

Zastosowanie systemu VR niesie za sobą pewne ograniczenia oraz wymaga zastosowania odpowiedniego procesu optymalizacji, który powinien być rozpatrywany od początku tworzenia aplikacji i zasobów, które się w niej znajdują. Sama technologia VR powoduje znaczny wzrost zapotrzebowania na moc obliczeniową procesora i karty graficznej, główną tego przyczyną jest potrzeba podwójnego renderowania obrazu po jednym na każde

oko odbiorcy. Biorąc pod uwagę, że aplikacja jest kierowana na urządzenia mobilne, które dysponują mniejszą mocą obliczeniową podczas procesu tworzenia aplikacji musimy stosować odpowiednie techniki budowy wirtualnego świata oraz jak najczęściej przeprowadzać testy tworzonej aplikacji. Przede wszystkim w odpowiedni sposób muszą zostać przygotowane zasoby, wyświetlane modele 3D muszą mieć odpowiednią nie zbyt gęstą geometrię a także zaleca się stosowanie atlasów tekstur, co pozwala na ograniczenie użytych w projekcie materiałów a co za tym idzie ograniczenie wywoływania procesu renderowania. Oświetlenie sceny powinno być realizowane w sposób statyczny, przez co możemy wypalić światła, czyli obliczyć zachowanie światła na scenie i przechowywać te informacje na mapach światła (ang. lightmaps) dzięki temu oświetlenie nie musi być przetwarzane na bieżąco co przy oświetleniu dynamicznym generuje spore obciążenie. Kolejnym ważną kwestią jest ograniczenie efektów odbicia czy refleksji materiałów, efektów cząsteczkowych (rozpryski wody, dym, ogień), skomplikowanych shader-ów oraz zastosowanie efektów związanych z fizyką (symulacja zderzeń i zniszczeń). Musimy też pamiętać, że efekty nakładane na kamerę, czyli finalny render obrazu takie jak wygładzanie krawędzi (ang. anti-aliasing), korekcja kolorów czy efekt rozmycia mogą znacznie obciążyć aplikację. Przy zastosowaniu systemu VR ważną kwestią jest zachowanie odpowiedniej częstotliwości wyświetlanych klatek (z ang. frame rate) w przypadku aplikacji wykorzystującej wirtualną rzeczywistość przyjmuje się że ilość klatek na sekundę nie powinna spadać poniżej 60 aby zapewnić jak najbardziej płynny obraz a co za tym idzie bardziej realistyczne odczucie wirtualnej rzeczywistości. Nagły spadek wyświetlanych klatek może być spowodowany przez wiele czynników i procesów, aby precyzyjnie określić co stanowi problem należy wykorzystać narzędzia analizujące działanie aplikacji, które dostarczą odpowiednich raportów i wskażą elementy wpływające na płynność działania aplikacji.

5. Wizualizacja obiektów architektonicznych

Wizualizacja będąca celem tej pracy, wraz z zasobami graficznymi (3D i 2D) została zbudowana według zaproponowanego schematu tworzenia aplikacji mobilnych 3D (rys. 15).



Rysunek 15. Proces tworzenia wizualizacji (źródło: opracowanie własne)

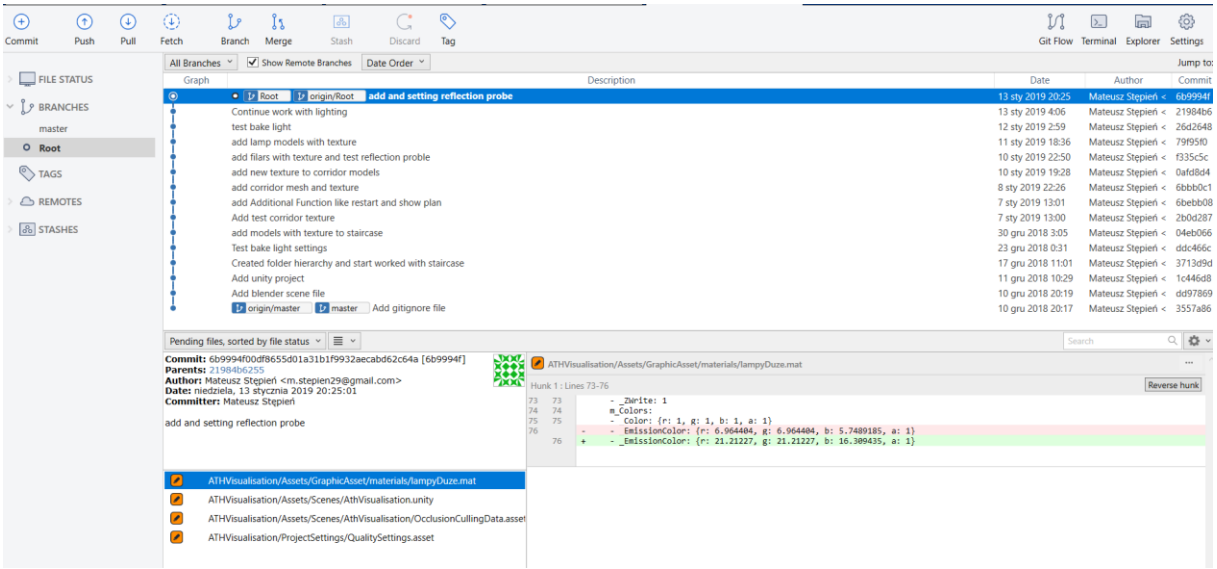
Przebieg prac rozpoczęto od analizy założeń i wymagań tworzonej aplikacji. Rozważania te pozwoliły określić rodzaj użytego oprogramowania w tym wybór silnika Unity 3D a także nakreśliły ograniczenia wydajnościowe i techniki optymalizacji, które zastosowano zarówno na etapie tworzenia materiałów graficznych użytych w aplikacji, jak i docelowej aplikacji kierowanej na platformy mobilne. Oceniono złożoność całego projektu i przygotowano materiały referencyjne. Następnie przystąpiono to utworzenia zasobów graficznych w tym geometrii trójwymiarowych modeli obiektów architektonicznych wraz z mapowaniem UV, oraz utworzenie odpowiednich tekstur. Modele wraz z teksturami zostały zaimportowane do odpowiednio skonfigurowanego projektu w silniku Unity, gdzie zostały wykorzystane do budowy wirtualnej przestrzeni. Następnie przystąpiono do tworzenia i ustawiania materiałów, wybierając odpowiedni moduł cieniujący i przypisując odpowiednie tekstury. Materiały zostały nałożone na trójwymiarowe obiekty 3D znajdujące się na scenie. Dalej zaimplementowano określoną funkcjonalność aplikacji z wykorzystaniem systemu wirtualnej rzeczywistości, a także przystąpiono do testów i wstępnych prac optymalizacyjnych. W finalnym kroku budowania wirtualnej przestrzeni opracowano oświetlenie całej sceny, wykorzystując odpowiednie techniki ustawiania świateł, dostosowane do wydajności docelowych urządzeń mobilnych. Gdy scena została zbudowana kontynuowano testy aplikacji i pracę optymalizacyjne dążąc do płynnego działania aplikacji, przy zachowaniu jak najlepszych efektów wizualnych.

W kolejnych rozdziałach i podrozdziałach ukazany proces tworzenia wizualizacji został szerzej przedstawiony.

5.1 Przygotowanie zasobów i środowiska pracy

Pierwszym etapem tworzenia wizualizacji architektonicznej było przygotowanie materiałów referencyjnych obiektów przedstawionych w aplikacji. Oprócz zapoznania się z planami architektonicznymi tworzonej przestrzeni wykonano zdjęcia i pomiary wizualizowanych obiektów, aby jak najlepiej odwzorować je w wirtualnej rzeczywistości. Kolejnym krokiem była instalacja niezbędnego oprogramowania (szczegółowo opisanego we wcześniejszej części pracy). Programu Blender (w wersji 2.79) w którym stworzono trójwymiarowe modele obiektów architektonicznych, programu Substance Painter (w wersji 2018.3.1), który posłużył do przygotowania tekstur oraz silnika Unity 3D (w wersji 2018.2.9) w którym zostały umieszczone wszystkie stworzone elementy i w którym aplikacja została zbudowana wraz z implementacją systemu VR.

Dodatkowo ważnym elementem każdego projektu jest wykorzystanie systemu kontroli wersji w celu jego zabezpieczenia poprzez stworzenie kopii zapasowej oraz możliwość śledzenia wszystkich zmian dokonywanych w projekcie. Na potrzeby tworzenia tej aplikacji utworzono repozytorium plików w serwisie GitHub wykorzystujące rozproszony system kontroli wersji GIT. Repozytorium zostało w odpowiedni sposób skonfigurowane do pracy z projektem tworzonym w silniku Unity. Przy pomocy pliku .gitignore ustalono odpowiednie reguły określające, które pliki mają być ignorowane np. pliki znajdujące się w folderze Builds czy pliki tymczasowe. Do obsługi repozytorium wykorzystano bezpłatne oprogramowanie SurceTree, które pełni rolą klienta systemu GIT i pozwala łączyć się ze zdalnym repozytorium wykorzystując protokół HTTPS. Oprogramowanie SourceTree poprzez przejrzysty interfejs graficzny pozwala w łatwy i szybki sposób zarządzać repozytorium, oferuje także wierz poleceń dla bardziej zaawansowanych i wymagających użytkowników.



Rysunek 16. Główna gałąź projektu w programie SourceTree

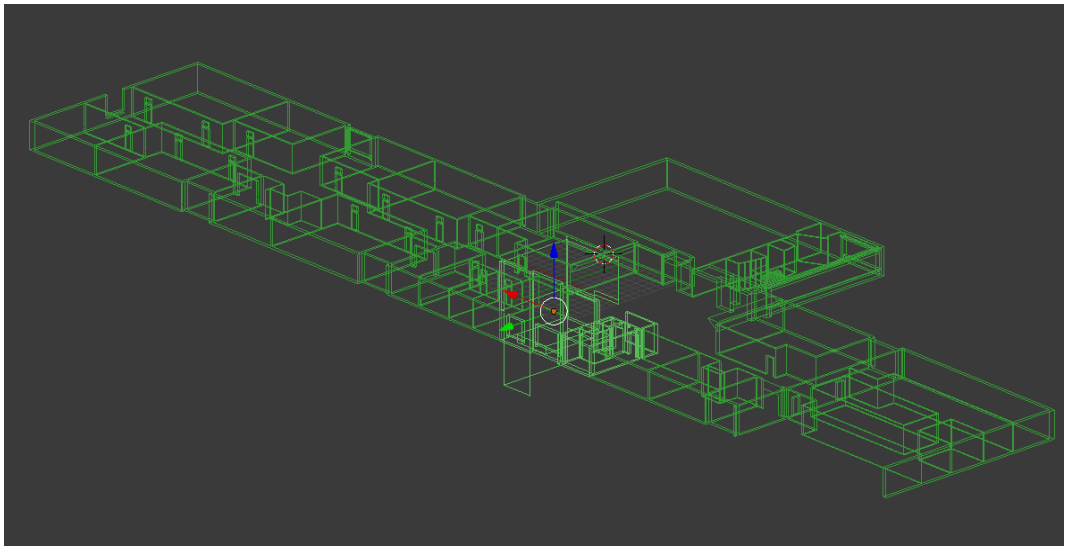
W przypadku tego projektu korzystano głównie z interfejsu graficznego SourceTree. Jako że prace nad projektem były wykonywane przez jedną osobę stworzono jeden branch – *Root* (rys., 16) na który na bieżąco wrzucano *commity*, czyli kolejne wersje zmian wprowadzanych w projekcie wraz z odpowiednim komentarzem. Zmiany wrzucano wykorzystując okno Working Copy gdzie dokonywano akceptacji zmian w poszczególnych plikach i wysyłano zmiany na serwer poprzez opcję *commit i push.* Source Tree jest także świetnym rozwiązaniem przy pracy zespołowej, ponieważ oferuje możliwość stworzenia osobnych odgałęzień dla każdego z członków zespołu. Dzięki temu praca może odbywać się równolegle a każda osoba może testować wprowadzone zmiany bez zakłócania pracy innych, poprzez wykorzystanie opcji *merge* gałęzie mogą być łączone w celu scalenia wszystkich plików. System GIT (a także Sourcetree) posiada pewne wady, główną z nich jest problem z przesyłaniem plików o dużych rozmiarach co w przypadku pracy z silnikiem Unity może powodować problemy i wymaga przesyłania dużych porcji plików w częściach.

5.2 Wykonanie modeli 3D

Głównym elementem wizualizacji jest wirtualne środowisko składające się z obiektów architektonicznych. W rozdziale tym opisano kolejne kroki tworzenia trójwymiarowych modeli, które są głównym elementem składowym sceny. Prace nad modelowaniem rozpoczęto od utworzenia nowego projektu w programie Blender i umieszczenia w nim wykonanych zdjęć referencyjnych i skanów planów architektonicznych (pierwszego pietra budynku L).

5.2.1 Modelowanie

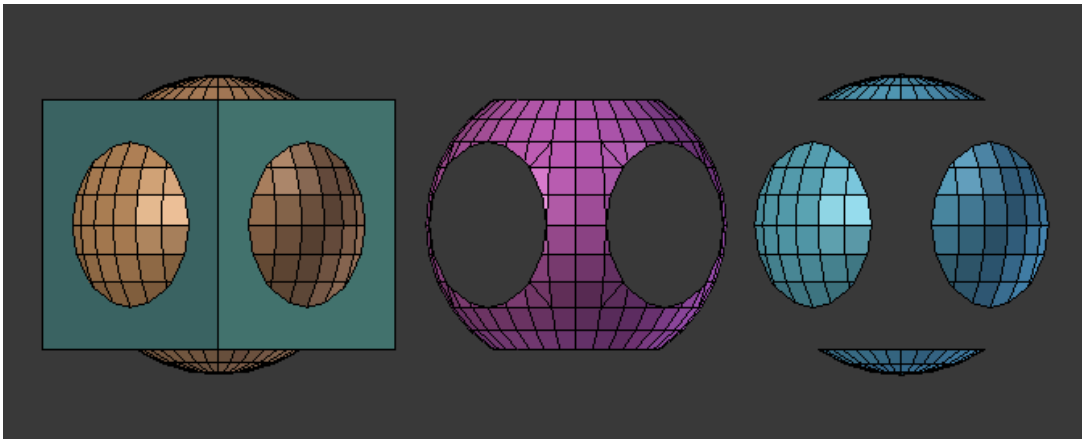
Bazując na wykonanych pomiarach i powyższych referencjach przystąpiono do modelowania od najbardziej bazowych elementów, jak ściany, sufit czy podłoga poprzez drzwi i schody na detalach jak na przykład poręcze kończąc. Tworzona przestrzeń została rozdzielona w programie na kilka warstw między innymi: elementy bazowe (rys. 17), drzwi, detale, co ułatwia prace pozwalając w szybki sposób zaznaczyć daną grupę elementów. Modelowane obiekty tworzone były poprzez dodawanie podstawowych brył dostępnych w programie Blender jak sześcian – *cube* czy walec – *cylinder* a następnie modyfikując ich siatkę wielokątów poprzez przenoszenie, skalowanie i obracanie a także powielanie i rozcięcia – *cut* zwiększano gęstość geometrii tworząc kolejne detale, tak aby jak najbardziej odwzorować rzeczywiste kształty.



Rysunek 17. Podstawowa siatka modelu 3D ścian (źródło: opracowanie własne)

Oczywiście gęstość siatki modelowanych obiektów musi być odpowiednio wyważona, tworząc modele 3D na potrzeby gier czy wizualizacji nie można dopuścić, aby liczba wierzchołków tworząca dany model była zbyt duża, ponieważ może to mieć krytyczny wpływ na wydajność aplikacji, należy o tym pamiętać już na etapie tworzenia zasobów graficznych.

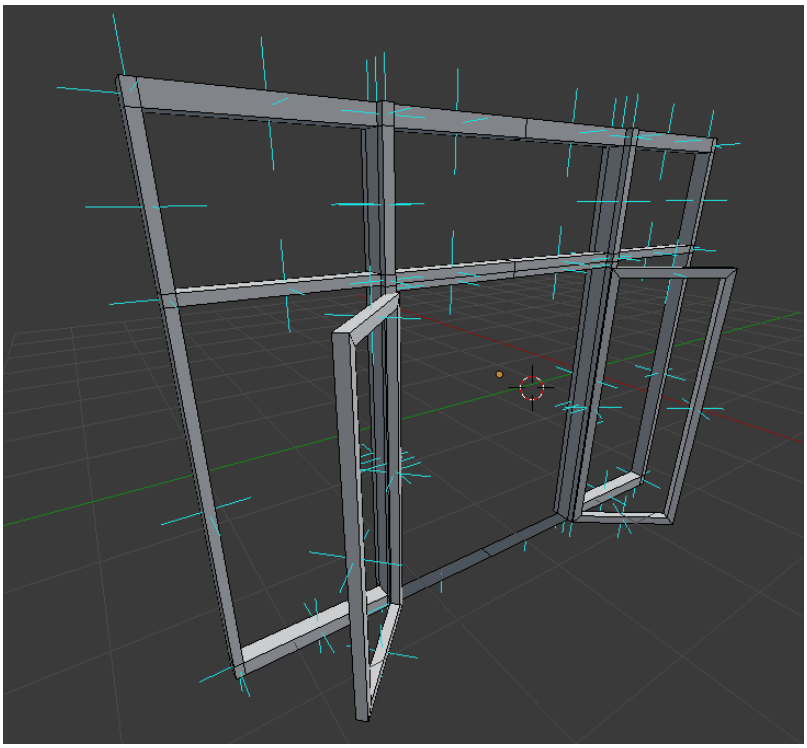
Podczas pracy z geometrią brył program Blender oprócz podstawowych opcji do zaznaczania wierzchołków, krawędzi i ścian, ich transformacji poprzez przenoszenia, skalowania i obracania oferuje wiele innych narzędzi ułatwiających pracę. Między innymi możliwość dociągania zaznaczonej geometrii – *snap* do innych elementów, tworzenie zeszlifowania danej krawędzi – *bevel*, czy edycja proporcjonalna – *Proportional Editing Mode* podczas, której transformacja zaznaczonych elementów wpływa na inne pobliskie elementy. Podczas pracy z siatką modeli 3D zbudowanych na cele w tej wizualizacji korzystano również z modyfikatorów oferowanych przez oprogramowanie Blender. Między innymi wykorzystywany był modyfikator operacji na siatkach - *Boolean* pozwalający na modyfikację siatki danego obiektu poprzez inny obiekt wykonując jedną z trzech operacji: *Union* - siatka docelowa jest dodawana do zmodyfikowanej siatki, *Difference* - zmodyfikowana siatka jest odejmowana od docelowej siatki i *Intersect* – gdzie siatka docelowa jest odejmowana od zmodyfikowanej siatki (rys. 18).



Rysunek 18. Zastosowanie operacji Union, Interseact i Difference (źródło: https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/booleans.html)

Innymi często stosowanym modyfikatorem był modyfikator *Array*, który tworzy tablicę kopii wybranego obiektu powielając dany model, przy czym każda kolejna jest przemieszczana w stosunku do poprzedniej w określony sposób, oraz modyfikator *Mirror* tworzący lustrzane odbicie siatki względem lokalnej osi (X,Y,Z).

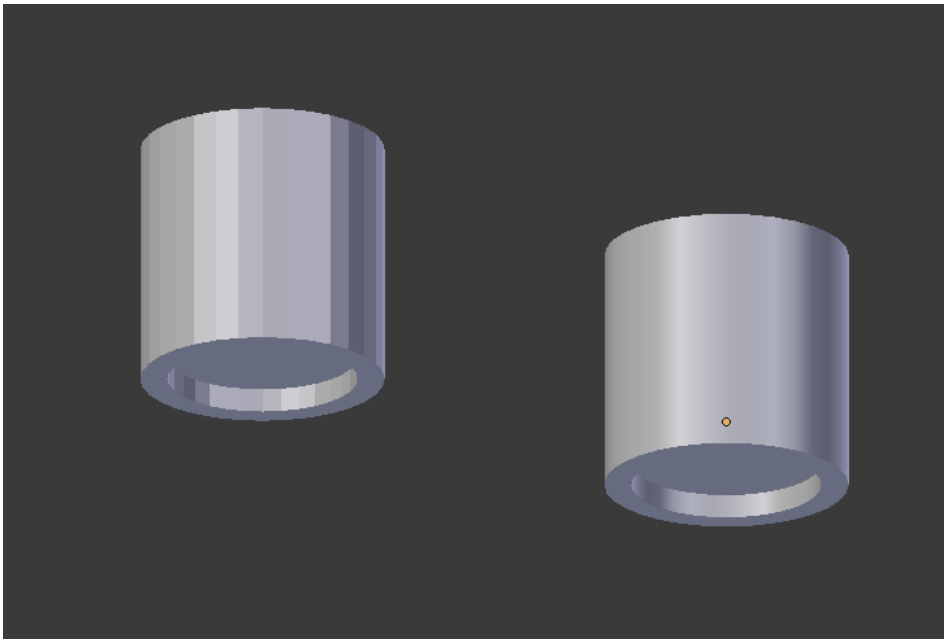
W końcowym etapie tworzenia trójwymiarowych modeli należy zwrócić uwagę na wektory normalnych powierzchni (z ang. normals). Standardowo są one prostopadłe do płaszczyzny i określają przód i tył danego wielokąta (rys. 19).



Rysunek 19. Wektor normalnych (źródło: opracowanie własne)

Blender wyświetla domyślnie wszystkie płaszczyzny w polu widzenia i te odwrócone przodem i tyłem natomiast silnik Unity 3D korzysta z metody *Backface Culling*, czyli wyświetla wielokąty zwrócone przodem w stronę pola widzenia co ma na celu ograniczenie renderowania niewidocznych powierzchni. Blender posiada narzędzia pozwalające na

odwracanie normalnych powierzchni a także opcję *Recalculate*, która pozwala przeliczyć normalne i automatycznie ustawić zwrot wektorów zamkniętej bryły na zewnątrz. Wektory normalnych są także wykorzystywane w procesie cieniowania obiektów. Za pomocą opcji *Shading Smooth* możemy ustawić wygładzanie wybranej powierzchni bez modyfikacji jej geometrii. Jest to bardzo korzystne pod kątem optymalizacji modelu 3D, ponieważ możemy otrzymać zaokrąglone krawędzie nie zwiększając gęstości siatki modelu (rys. 20). Oprogramowanie Blender realizuje cieniowanie obiektów wykorzystując Cieniowanie Phonga (z ang. Phong Shading) [7].

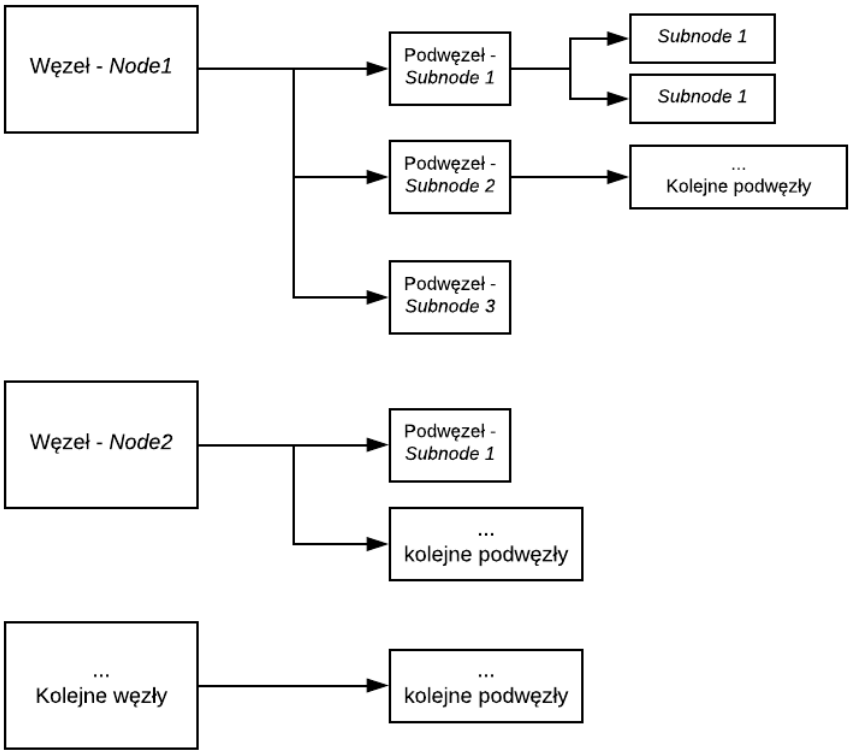


Rysunek 20. Obiekt z wyłączonym i włączonym wygładzaniem (źródło: opracowanie własne)

Podczas pracy z cieniowaniem obiektów korzystano z opcji *Auto Smooth* pozwalającej na ustawienie gładkich cieniowanych krawędzi w sposób automatyczny gdzie krawędzie, w których kąt pomiędzy powierzchniami jest mniejszy niż podany kąt zostaną wygładzone.

Każdy z utworzonych obiektów 3D został zapisany do pliku w formacie FBX. Format ten oprócz zapisywania siatki trójwymiarowego modelu umożliwia także zapis animacji, informacji o kamerze, oświetlenia czy modyfikatorów, oraz zapewnia wysoką kompatybilność z wieloma innymi narzędziami graficznymi w tym Unity.

Podstawowym elementem konstrukcyjnym formatu FBX są hierarchiczne węzły – *Node* wraz z pod węzłami - *Subnode*. Każdy wyeksportowany model jest zapisywany w postaci zagnieżdżonej listy węzłów i pod węzłów (rys. 21), które są analizowane i odczytywane przez oprogramowanie graficzne, w którym dany model jest umieszczony.



Rysunek 21. Diagram przedstawiający strukturę pliku FBX (źródło: opracowanie własne)

Każdy z węzłów i pod węzłów posiada odpowiedni identyfikator oraz właściwości określane przy pomocy typów danych między innymi: bool, short, int, long, float, double, bytes, czy string. Każdy węzeł ma następującą strukturę:

```
Nazwa węzła: {  <---- rozpoczęcie węzła
    Właściwość 1: wartość
    Właściwość 2: wartość

    Podwęzeł :  {   <---- rozpoczęcie podwęzła
        Właściwość 1: wartość
        […]
    }   <---- koniec podwęzła

    Właściwość 3: wartość
    […]
} <---- koniec węzła
```

Węzeł każdego z wyeksportowanych modeli zawiera podstawowe informacje na temat pliku fbx. Poniżej zamieszczono pierwszy węzeł z przykładowego modelu wyeksportowanego z programu Blender. [16]

```
FBXHeaderExtension:  {
	FBXHeaderVersion: 1003
	FBXVersion: 6100 // właściwość określająca wersję formatu FBX
	CreationTimeStamp:  { // podwęzeł zawierający informacje na temat daty
utworzenia pliku
		Version: 1000
		Year: 2019
		Month: 01
		Day: 12
		Hour: 13
		Minute: 37
		Second: 57
		Millisecond: 0
	}
	Creator: "FBX SDK/FBX Plugins build 20070228"
	OtherFlags:  {
		FlagPLE: 0
	}
}
```

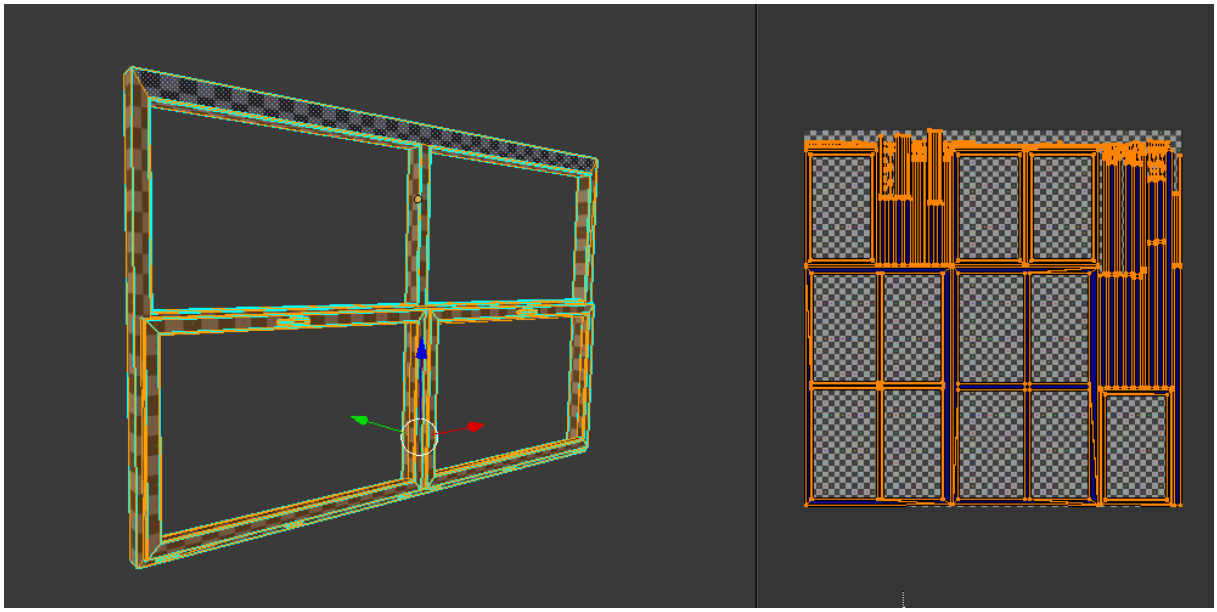
Informacje na temat trójwymiarowych modeli zapisanych w pliku FBX zawarte są w węźle *Objects*. Węzeł ten przechowuje informację na temat wierzchołków modeli 3D, indeksach, wektorach normalnych, współrzędne UV i materiałach przypisanych do modeli [17]. Węzeł *Objects* jest zbudowany w następujący sposób

```
Objects:  {
    Model: “nazwa modelu”, “rodzaj” { <---- węzeł konkretnego modelu
        […]
        Vertices: […]       <---- wierzchołki
        PolygonVertexIndex: […]  <---- indeksy
        LayerElementNormal: { }  <---- podwęzeł zawierający wektory normalnych
        LayerElementUV: { }  <---- współrzędne
    }

    Material: “nazwa materiału”, “” { } <---- węzeł materiałów
    […]
}
```

Pliki FBX (rozszerzenie .fbx) są aktualnie zwykle zapisywane w formacie binarnym, ale korzystając ze starszych wersji tego standardu mogą być również zapisywane w formacie ASCII. Na potrzeby tej wizualizacji modele zostały wyeksportowane wykorzystując wersję FBX 7.4, gdzie eksportowane pliki mają postać binarną.

5.2.2 Mapowanie UV

Po zakończeniu procesu modelowania obiektów 3D kolejnym etapem jest nałożenie na stworzone modele tekstur, które pozwolą dodać detale tworzonym obiektom oraz zwiększa realizm i odwzorowanie tworzonej przestrzeni. Tekstury zostały nałożone na powierzchnie modeli 3D za pomocą metody mapowania UV (z ang. UV Mapping). Proces ten polega na rozłożeniu trójwymiarowej siatki na postać dwuwymiarową z układem współrzędnych „U" i „V". Otrzymaną przestrzeń nazywamy Mapą UV (z ang. UV Map), która określa jak tekstura będzie nakładana na płaszczyznę modelu (rys. 22).



Rysunek 22. Model okna z rozwiniętą mapą UV (źródło: opracowanie własne)

Tworzenie map UV w programie Blender odbywa się poprzez wykorzystanie jednej z dostępnych metod rozwijania UV mapy w zależności od geometrii danego obiektu. Podczas pracy z modelami użytymi do realizacji tej wizualizacji korzystano z podstawowej funkcji *Unwrap*, gdzie siatka jest rozwijana na podstawie wyznaczonych szwów (z ang. mark seams), które określają miejsce rozcięcia UV mapy tworząc osobną płaszczyznę w dwuwymiarowej przestrzeni UV. Inną wykorzystywaną metodą jest inteligenta projekcja UV (z ang. *Smart UV, Project*). Metoda ta polega na automatycznym rozłożeniu mapy UV bazując na podanych przez użytkownika parametrach takich jak: próg kąta czy margines pomiędzy osobnymi powierzchniami na UV mapie. *Smart UV Project* daje dobre rezultaty przy nieskomplikowanej geometrii modelu 3D oraz pozwala w szybki sposób rozwinąć UV mapę. Przydatną metodą tworzenia map UV jest również projekcja bazująca na widoku kamery (z ang. Project from View) gdzie mapa UV danego modelu tworzona jest w zależności od jego ustawienia względem kamery jest to przydatne przy rozwijaniu map obiektów płaskich, które mają wszystkie ściany zwrócone w jednym kierunku, jak np. podłoga czy ściany. Blender oferuje również rozbudowany edytor, który pozwala dostosować stworzoną UV mapę do danego obiektu poprzez modyfikację skali, rotacji i położenia siatki UV, aby tekstura była w

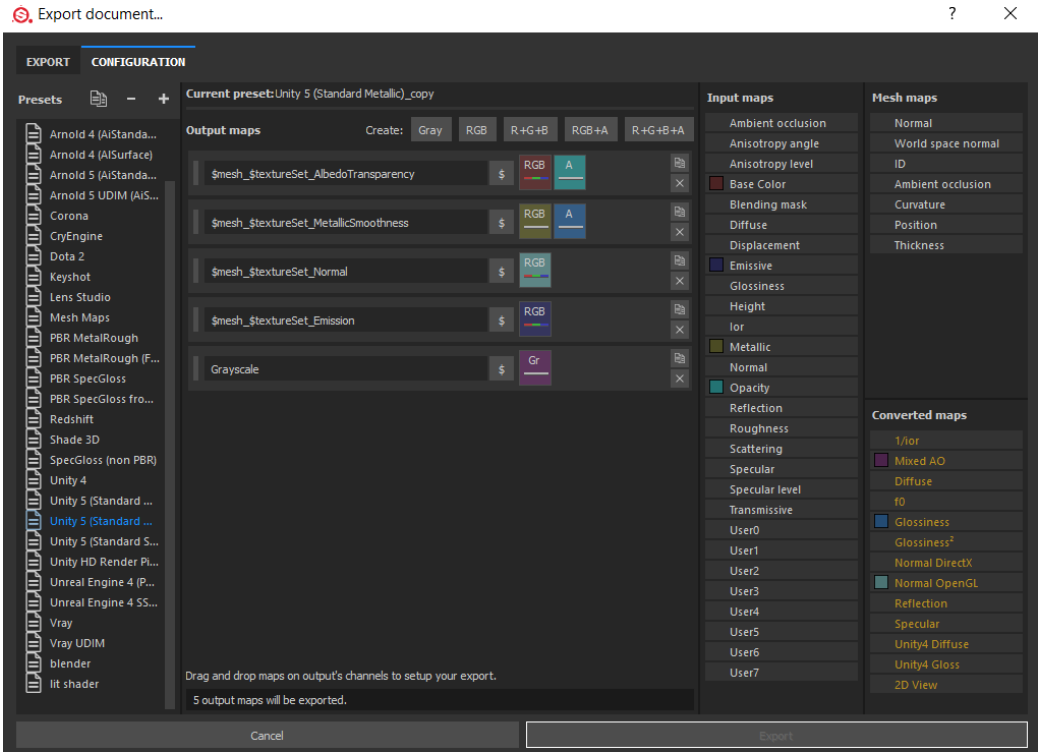
jak najlepszy sposób nakładana, bez widocznych deformacji czy miejsc złożenia. W procesie mapowania również rozważono aspekty optymalizacyjne aplikacji a mianowicie przyśpieszenie renderingu poprzez stworzenie atlasów tekstur, czyli mapowania wielu obiektów w taki sposób, aby modele mogły korzystać z jednej większej tekstury łączącej w sobie wiele mniejszych. Zabieg ten ma na celu ograniczenie ilość odwołań do karty graficznej (ang. draw calls).

5.2.3. Teksturowanie

Po utworzeniu modeli 3D wraz z rozwiniętymi mapami UV użyto programu Substance Painter (opisanego szerzej we wcześniejszej części pracy) do wykonania tekstur. Prace w tym programie rozpoczęto od utworzenia projektu gdzie oprócz nadania nazwy ustalana jest rozdzielczość tworzonych tekstur, oraz ustawiana jest ścieżka dostępu do zaimportowania wcześniej utworzonych modeli 3D. Kontynuując konfigurację projektu w programie Substance Painter wybrano odpowiednią mapę otoczenia (z ang. Environment Map), która odpowiada za oświetlenie sceny podczas pracy, oraz wybrano odpowiedni Shader (pbr-metal-rough-with-alpha-blending) umożliwiający tworzenie tekstur na potrzeby modelu PBR gdzie oprócz tektury koloru podstawowego wykorzystywana jest mapa normalnych oraz inne mapy takie jak metallic czy rughnes. Wybrany Shader oferuje także możliwość modyfikacji przeźroczystości, co było wykorzystywane przy tworzeniu tekstur szklanych obiektów jak na przykład szyby. Ostatnim krokiem przed przystąpieniem do tworzenia tekstur było wypalenie dodatkowych map obiektu między innymi mapy cieni (z ang. ambient occlusion), mapy pozycji czy World Space Map wykorzystywanych przy stosowaniu masek i efektów dostępnych w programie Substance Painter. Po odpowiednim przygotowaniu projektu przystąpiono do teksturowania, które opiera się na wybraniu z przybornika jednego z materiałów, który zawiera zestaw kanałów PBR z teksturami i dalej tworzeniu kolejnych warstw nakładanych na model. W programie Substance Painter możemy tworzyć zwykłe warstwy i malować po powierzchni modelu jednym z wybranych pędzli nakładając wybrany materiał, lecz częściej wykorzystywano warstwy pełne nakładające wybrany materiał na cały obiekt (z ang. fill layer). Podczas prac korzystano głównie z bogatej biblioteki predefiniowanych materiałów dostępnych w programie (rys. 23) dostosowując je w dalszych etapach do swoich potrzeb oraz tworzono własne materiały na bazie zaimportowanych tekstur.

Rysunek 23. Przybornik programu Substance Painter zawierający materiały dostępne w projekcie (źródło: opracowanie własne)

Po nałożeniu jednego z materiałów był on dostosowywany poprzez modyfikację poszczególnych tekstur wykorzystując narzędzia dostępne w programie Substance Painter, jak na przykład filtry czy generatory pozwalające dodać odpowiednie detale, aby nadać jak największy realizm tworzonym obiektom. A także korzystano z opcji maski warstwy, która określa, na jakie części modelu oddziałuje dana warstwa. Maski tworzone były malując po obiekcie pędzlem o odpowiednim kształcie lub poprzez wykorzystanie narzędzia zaznaczania - *Polygon Fill*, które pozwala w szybki sposób stworzyć maskę, wybierając wieloboki, elementy siatki lub fragmenty UV teksturowanego modelu 3D. Po zakończeniu procesu teksturowania w programie Substance Painter utworzone tekstury zostały wyeksportowane. Podczas eksportu ustalono format tekstur, rozdzielczość a także wybrano odpowiednią konfigurację (rys. 24), aby wyeksportowane tekstury były kompatybilne z shaderem stosowanym w silniku Unity 3D gdzie finalnie utworzone modele zostały umieszczone.



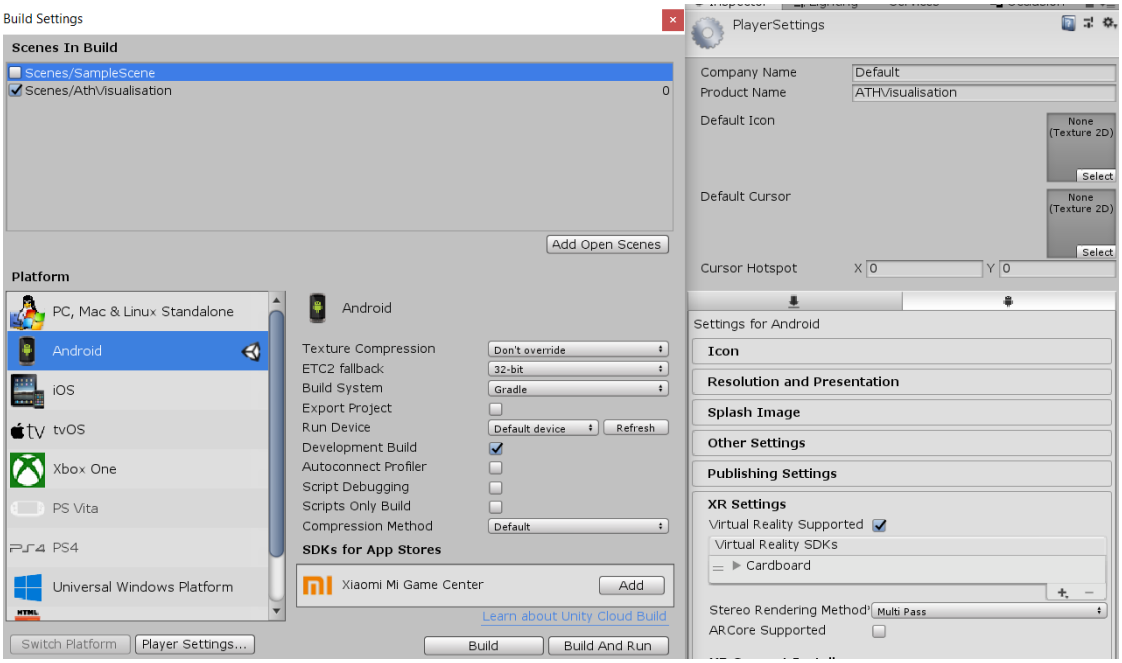
Rysunek 24. Konfiguracja eksportu tekstur w programie Substance Painter (źródło: opracowanie własne)

5.3 Wykonanie aplikacji

Po wykonaniu zasobów 3D – modeli wraz z teksturami, z których zbudowane jest wirtualna przestrzeń niniejszej wizualizacji przystąpiono do procesu tworzenia finalnej aplikacji działającej w systemie Android. Gdzie wcześniej utworzone elementy zostaną zaimplementowane dając możliwość pokazania użytkownikowi modeli architektonicznych wykorzystując do tego zestaw VR i system wirtualnej rzeczywistości.

5.3.1 Założenie projektu Unity

Pierwszym krokiem utworzenia finalnej wizualizacji było założenie projektu w silniku Unity i jego odpowiednie skonfigurowanie. W panelu *Build Settings* zmieniono docelową platformę, na którą zostanie zbudowana aplikacja na system Android (rys. 25) a za pomocą sekcji *Player Settings* dostosowane ustawienia budowanej aplikacji między innymi nazwę aplikacji, wymaganą minimalną wersję systemu Android zainstalowaną na urządzeniu a także uruchomiono wsparcie dla systemu wirtualnej rzeczywistości.

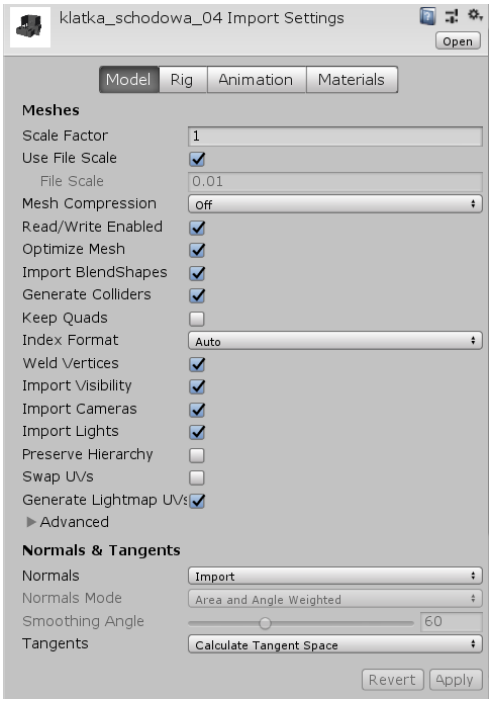


Rysunek 25. Unity 3D - konfiguracja projektu (źródło: opracowanie własne)

Wewnątrz projektu Unity utworzono kilka głównych folderów, aby zapewnić odpowiednią strukturę i organizację plików i katalogów w projekcie. Folder *Scenes*, w którym znajdują się utworzone w projekcie sceny zarówno te finalne jak *AthVisualisation* jak i testowe. Kolejnym folderem jest *GraphicAsset*, w którym znajdują się wszystkie użyte w projekcie zasoby graficzne posegregowane w odpowiednich podkatalogach takich jak: *models* – w którym przechowywane są modele, *texture* – zawierający tekstury, *materials* – w którym znajdują się materiały, *hdri* z teksturami otoczenia wykorzystywanymi przy oświetleniu sceny oraz folder *2D assets* zawierający dodatkowe elementy graficzne 2D. Innymi głównymi folderami są folder *Scripts* gdzie znajdują się skrypty odpowiedzialne za odpowiednią funkcjonalność aplikacji, *Prefabs* – zawierający prefabrykaty i *Plugins* –przechowujący biblioteki i wtyczki wykorzystywane pod daną platformę lub konkretną technologię. W hierarchii projektu znajdują się także foldery utworzone przez silnik Unity niezbędne do odpowiedniego działania projektu.

5.3.2 Implementacja modeli 3D

Utworzone wcześniej modele 3D obiektów architektonicznych należy w odpowiedni sposób eksportować z programu Blender i zaimplementować w projekcie Unity. W tym celu wszystkie modele zostały zapisane w formacie *FBX* co pozwala na ich elastyczne wykorzystanie nie tylko w silniku Unity, ale także innych silnikach 3D wspierających obsługę formatu plików FBX. Podczas eksportu w programie Blender należy pamiętać o odpowiednim ustawieniu skali i rotacji obiektu a także o ustawieniu punktu obrotu (z ang. Pivot Point), według którego dany model jest przenoszony i obracany w trójwymiarowej przestrzeni. Odpowiednio ustawiony Pivot point umożliwia łatwe ustawienie obiektów na finalnej scenie. Każdy z umieszczanych w silniku modeli 3D posiada własne ustawienia importu(rys. 26). Przy importowaniu modeli w celu wykorzystania w tworzonej wizualizacji korzystano z opcji *Generate Colliders* co powoduje stworzenie i przypisanie do obiektu komponentu kolizji tworząc *Collider* bazując na siatce danego modelu. Komponenty kolizji są szczególnie przydatne podczas realizacji poruszania się użytkownika na scenie i uniemożliwiają mu przenikanie przez obiekty ustawione na scenie. Innym ważnym parametrem jest *Generate Lightmaps UVs,* co powoduje wygenerowanie mapy UV dla map światła, mapy te są wymagane podczas wypalania oświetlenia co zostało szerzej opisane w dalszej części pracy.



Rysunek 26. Okno importu modelu 3D do silnika Unity

5.3.3 Tworzenie i ustawianie materiałów

Gdy modele 3D zostały odpowiednio zaimportowane do projektu i ułożone na scenie przystąpiono do stworzenia i nakładania materiałów, aby odwzorować realny wygląd tworzonej wirtualnej przestrzeni.

Umieszczone na scenie trójwymiarowe modele są z technicznego punktu widzenia zbiorem współrzędnych 3D – wierzchołków, które są ze sobą połączone tworząc trójkąty renderowane przez kartę graficzną. Każdy wierzchołek modelu może zawierać kilka informacji między innymi kierunek, w którym wskazuje (zwany normalnym), czy współrzędne odpowiadające za odpowiednie rozłożenie tekstury (UV mapy). Modele nie mogą zostać wyrenderowane bez materiału z przypisanym konkretnym modułem cieniującym (z ang. Shader).

Renderowanie w silniku Unity odbywa się w trzech głównych etapach. Pierwszym krokiem jest *Culling*, czyli proces oznaczenia obiektów, które mają zostać renderowane. W większości przypadków są to obiekty widoczne przez kamerę, obiekty zasłonięte prze inne modele, czyli te, które zostały ukryte w procesie *Occlusion Culling* (zagadnienie to jest dokładniej poruszone w rozdziale poświęconym optymalizacji) zostaną pominięte. Drugi etap to proces rysowania wcześniej wyznaczonych obiektów do buforów bazujących na pikselach (pixel buffers). W procesie tym uwzględnianie jest oświetlenie i indywidualne właściwości obiektów takie jak nałożone materiały. Na tym etapie następuje przekształcenie przestrzeni trójwymiarowej w dwuwymiarowy obraz. Finalnym krokiem procesu renderowania jest generowanie na podstawie buforów pikseli końcowej klatki obrazu, która następnie przesyłana jest do urządzenia wyświetlającego. Na tym etapie wykonywane są także operacje przetwarzania końcowego (*Post-processing*), które umożliwiają modyfikacje wyświetlanego obrazu (np. balans kolorów czy zmiana saturacji). Cały proces renderowania jest powtarzany zależnie od ilości wyświetlanych klatek na sekundę.

Materiały w silniku Unity definiują sposób renderowania powierzchni, w odniesieniu do użytych tekstur, informacji o skalowaniu tekstury, odcieniach kolorów i wielu innych parametrów zależnych od rodzaju użytego modułu cieniującego. Moduł cieniujący (Shader) jest to skrypt kierowany do karty graficznej zawierający matematyczne obliczenia i algorytmy obliczania koloru każdego piksela danego modelu na podstawie oświetlenia i konfiguracji materiału.

W Unity wyróżniamy dwa typy modułów cieniujących: cieniowanie powierzchni – *Surface shaders* przykładem może być standardowy moduł cieniujący w Unity (*Standard Shader*) oraz cieniowanie wierzchołków i fragmentów - *Vertex and fragment shaders*.

Cieniowanie powierzchni jest wykorzystywane dla materiałów symulujących realistyczne oddziaływanie z oświetleniem. W modułach cieniujących wartości materiału takie jak tekstury i parametry są podłączane do modelu oświetlenia, który wyprowadza końcowe wartości RGB dla każdego piksela. Poniżej zamieszczono przykładowy kod modułu cieniowania powierzchni w języku CG (C for graphics). [13]

```
CGPROGRAM

#pragma surface surf Lambert // wykorzystanie Lambertowskiego modelu oświetlenia

sampler2D _MainTex; // wprowadzenie tekstury 2D

struct Input {
```

```
    float2 uv_MainTex;
};
void surf (Input IN, inout SurfaceOutput o) {
    o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
}
ENDCG
```

Za pomocą linii sampler2D _MainTex; wprowadzono teksturę 2D. W funkcji *surf* tekstura ta jest ustawiana, jako właściwość Albedo materiału. Powyższy Shader korzysta z Lambertowskiego modelu oświetlenia. Lambertowski model oświetlenia opiera się na założeniu, że obiekty są idealnie matowe, a zatem światło odbite od powierzchni rozchodzi się we wszystkich kierunkach nad powierzchnią z taką samą intensywnością. [13]

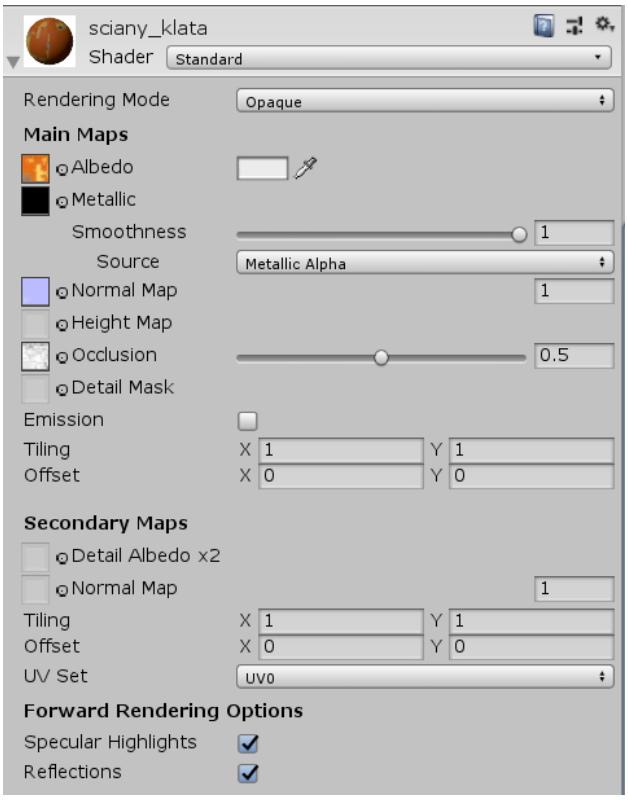
Natomiast moduł cieniujące typu *Vertex and fragment shaders* nie posiada wbudowanej obsługi modelu oświetlenia. Geometria modelu 3D w pierwszej kolejności jest przekazywana do funkcji *Vert*, która może wpłynąć na renderowane wierzchołki a następnie pojedyncze trójkąty przechodzą przez funkcję *Frag* która decyduje o ostatecznym kolorze RGB dla każdego piksela. Moduły cieniujące tego typu są wykorzystywane do efektów 2D, postprocessingu i specjalnych efektów 3D, które są zbyt skomplikowane, aby można je było zrealizować przy pomocy modułu cieniowania powierzchni. Poniżej ukazano kod *Shadera* typu *Vertex and fragment shaders*, który renderuje model w jednolitym zielonym kolorze. [13]

```
Pass {
    CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag

    struct vertInput {
        float4 pos : POSITION;
    };
    struct vertOutput {
        float4 pos : SV_POSITION;
    };
    vertOutput vert(vertInput input) {
        vertOutput o;
        o.pos = mul(UNITY_MATRIX_MVP, input.pos);
        return o;
```

```
    }
    half4 frag(vertOutput output) : COLOR {
        return half4(0.0, 1.0, 0.0, 1.0);
    }
    ENDCG
}
```

Funkcja *vert* konwertuje wierzchołki z ich macierzystej przestrzeni 3D do końcowej pozycji 2D na ekranie. Unity wprowadza *UNITY_MATRIX_MVP*, która wykonuje niezbędne obliczenia matematyczne. Finalnie w funkcji *frag* nadawany jest zielony kolor do każdego piksela. [13]

Większość materiałów utworzonych w projekcie bazuje na standardowym module cieniującym dostępnym w Unity (*Standard Shader*). Standard Shader oferuje obszerny zestaw funkcji i parametrów, dzięki czemu możemy przy jego pomocy odwzorować większość występujących w przyrodzie materiałów od powierzchni metalicznych jak aluminium czy stal przez materiały występujące naturalnie jak kamień, drewno na tworzywach sztucznych jak plastik, czy guma kończąc. Shader ten działa w oparciu o model oświetlenia PBR, przez co symuluje interakcje pomiędzy materiałami i światłem w sposób naśladujący rzeczywiste zachowanie środowiska [9]. Zaznaczając materiał, w oknie inspektora wyświetlane są jego właściwości i parametry gdzie dokonano zmian ustawień materiału i przypisano wcześniej utworzone i zaimportowane do projektu tekstury (rys. 27).



Rysunek 27. Parametry materiału w silniku Unity 3D (źródło opracowanie własne)

Opcja Tryb renderowania - *Rendering Mode* pozwala wybrać czy dany materiał obsługuje przeźroczystość a także określa tryb mieszania wartości przeźroczystych i nieprzeźroczystych, domyślnie to pole jest ustawione, jako nieprzeźroczysty - *Opaque* co oznacza, że materiał nie korzysta z przeźroczystości. Podczas tworzenia materiałów dla obiektów zawierających elementy przeźroczyste jak szyby tryb renderowania został ustawiony na *Transparent*, co powoduje ustawienie obszarów przeźroczystości na podstawie kanału alfa tekstury kolory (Albedo). W kolejnych parametrach przypisano do materiału odpowiednie wcześniej stworzone tekstury takie jak: teksturę koloru – *Albedo,* która odpowiada za kolory danej powierzchni oraz za pomocą kanału alfa określa jej przeźroczystość. Metaliczność – *Metallic* pozwala określić stopień metaliczności danego materiału a także jego połyskliwość, przypisując do tego pola teksturę Metallic Map możemy uzyskać różną metaliczność na poszczególnych częściach powierzchni a także dodać dodatkowe detale jak otarcia czy zabrudzenia przez modyfikacje stopnia gładkości. Kolejnym parametrem jest mapa wypukłości – *Normal Map* gdzie przypisujemy odpowiednią teksturę (mapę normalnych), która symuluje dodatkowe detale na powierzchni takie jak rysy czy wgłębienia. Odbywa się to bez modyfikacji geometrii obiektu, lecz poprzez wyznaczania w jaki sposób światło odbija się od danej powierzchni (modyfikując wektory normalnych). Korzystanie z map wypukłości jest szczególnie korzystne z punktu widzenia optymalizacji, ponieważ pozwala dodać wiele detali do obiektu bez zagęszczania siatki zmniejszając tym samym ilość renderowanych wierzchołków. Ostatnią teksturą przypisywaną do tworzonych materiałów jest tekstura cieni *(Ambient Occlusion)* przypisywana do materiału jako parametr *Occlusion* pozwala zasymulować zaciemnienie części powierzchni określając obszar i stopień wpływu oświetlenia pośredniego. *Standard Shader* umożliwia także stworzenia materiału

emitującego światło. Parametr *Emission* pozwala kontrolować kolor i intensywność światła emitowanego z powierzchni. Do parametru Emission podobnie jak do innych parametrów materiału w Unity możemy przypisać mapę emisji – teksturę określającą powierzchnie emitującą kolor oraz siłę i kolor emitowanego światła. Podczas prac materiały emitujące światło zostały wykorzystane do symulowania świecących lamp wewnątrz pomieszczeń.

5.3.4 Oświetlenie sceny

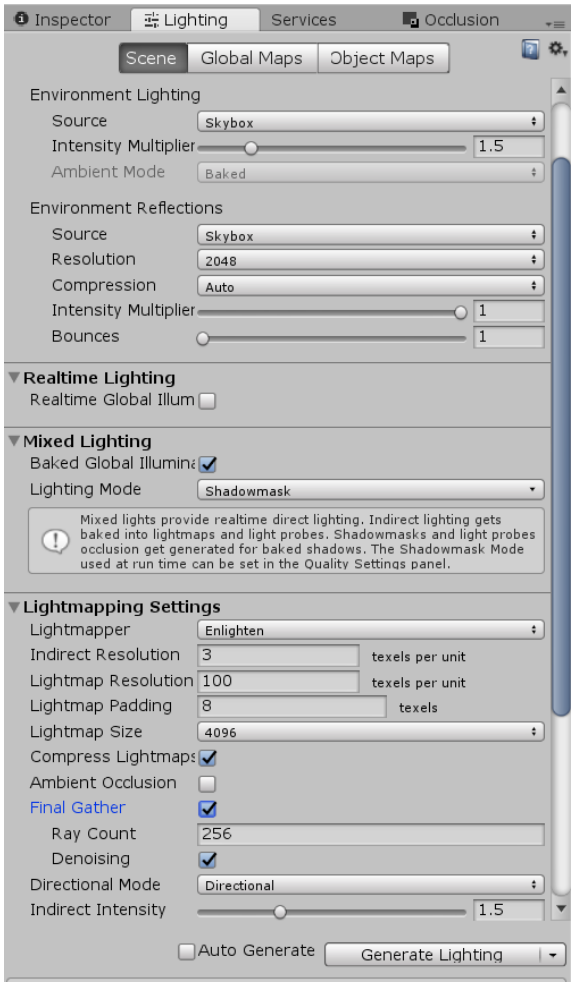
Gdy obiekty wraz z nałożonymi materiałami znajdowały się na scenie przystąpiono do ustawień oświetlenia, które pełni kluczową rolę w oddaniu realizmu tworzonej wirtualnej przestrzeni.

Przed rozpoczęciem właściwej konfiguracji oświetlenia, w oknie *GraphicsSettings* ustawiono odpowiednią ścieżkę renderowania – *Rendering Path*. Różne ścieżki renderowania mają różne charakterystyki wydajności, które w większości dotyczą świateł i cieni [9]. Jako że aplikacja będzie budowana na platformy mobilne ustawiono ścieżkę renderowania *Forward Rendering*, która jest obsługiwana na urządzeniach z systemem Android.

Ścieżka ta renderuje każdy obiekt w jednym lub więcej przebiegów, w zależności od świateł, które na niego wpływają. Dlatego każdy obiekt może być renderowany wiele razy w zależności od tego, ile świateł znajduje się w zasięgu. Zaletą tej ścieżki renderowania są niskie koszty obliczeniowe - co oznacza, że wymagania sprzętowe są niższe niż rozwiązania alternatywne. Dlatego używanie Forward Rendering jest zalecany na platformach mobilnych. Ponadto Forward Rendering bez problemu obsługuje przezroczystość obiektów, co jest wykorzystywane w tworzonej wizualizacji. Dodatkowym atutem jest również możliwość stosowanie technik sprzętowych, takich jak *multi-sample anti-aliasing* (MSAA), czyli wygładzanie krawędzi, które nie są dostępne w innych alternatywach potokach renderowania. Wygładzanie krawędzi poprawia wygląd krawędzi wielokątów, przez co nie są one widoczne jako "poszarpane", lecz wygładzone. W aplikacji korzystającej z wirtualnej rzeczywistości wygładzanie krawędzi mocno wpływa na immersje z wirtualnym środowiskiem, dlatego opcja ta powinna być uruchomiona.

Unity oferuje bogate możliwości dostosowania oświetlenia sceny oraz zastosowanie różnego typu świateł (rys. 28). Na potrzeby tworzenia wizualizacji wykorzystano światło kierunkowe – *Directional Light* symulujące zachowanie słońca oraz materiały emisyjne w celu odwzorowania lamp wewnątrz pomieszczeń. Dodatkowym źródłem oświetlenia całej sceny jest materiał otoczenia *Skybox Material,* który przy pomocy tekstury HDRi symuluje światło padające ze środowiska otaczającego scenę. Kolejnym krokiem był wybór techniki oświetleniowej, która określa czy światła na scenie będą obliczane w sposób rzeczywisty (z ang. realtime) i aktualizowane w każdej klatce działania aplikacji czy wstępnie obliczane wcześniej (wypalane) i zapisywane na mapach światła (z ang. lightmaps). Mając na uwadze ograniczenia narzucane przez docelową platformę android i wydajność urządzeń mobilnych, na którą kierowana jest tworzona wizualizacja a także dodatkowe obciążenie generowane przez zastosowanie systemu VR, zdecydowano się na wykorzystanie systemu wstępnego obliczenia świateł co pozwala zaoszczędzić wykonywania wielu obliczeń podczas działania

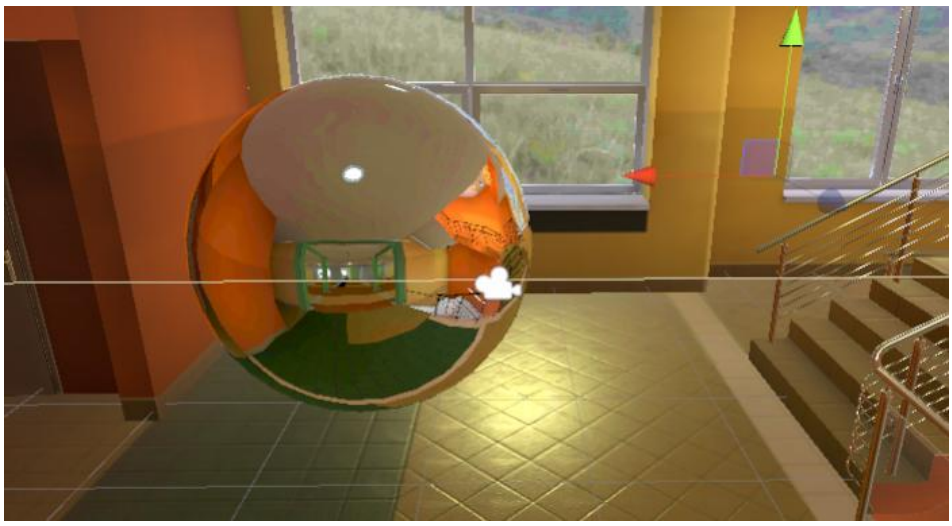
aplikacji i tym samym znacznie przyczynić się do zwiększenia wydajności. Dokładne ustawienia oświetlenia całej sceny w tym techniki oświetleniowej dokonywane są przy pomocy panelu *Lighting*.



Rysunek 28. Ustawienia świateł w silniku Unity 3D (źródło: opracowanie własne)

W sekcji ustawień środowiska - *Environment* ustawiono materiał otoczenia (*Skybox Material)* a także światło kierunkowe, które symuluje światło słoneczne. Ustawiono także źródło światła otoczenia na *Skybox*, aby światło to było symulowane przy pomocy wcześniej ustawionej tekstury otoczenia wraz z dostosowaniem intensywności. W dalszej części ustawień wybrano pieczony system globalnej iluminacji - *Baked Global Illumination* co umożliwia wstępne obliczenie światła i wypalenie go na mapach światła (*lightmaps*). Podczas tego procesu Unity uwzględni tylko obiekty oznaczone, jako *Lightmap Static* dlatego większość modeli znajdujących się na scenie została w oknie inspektora ustawiona jako obiekty statyczne- *Static* [9]. Aby wszystkie światła zostały odpowiednio wypalone i nie narzucały dodatkowych obliczeń w trakcie działania aplikacji oprócz wprowadzenia odpowiednich ustawień w zakładce *Lighting*, zmieniono także tryb działania wszystkich świateł znajdujących się na scenie na *Baked* co powoduje, że zarówno bezpośrednie i pośrednie oświetlenie z tych świateł jest wypiekane na mapach światła. Dodatkowo na scenie umieszczono i wypalono także sondy odbić - *Reflection Probe, które* rejestrują sferyczny widok otoczenia, który jest zapisywany jako mapa kubiczna (z ang. cube map) (rys. 29) [9].

Reflection Probe zostały użyte w celu odwzorowania odbicia innych obiektów znajdujących się na scenie na obiektach z połyskliwymi materiałami jak na przykład podłoga, poręcze schodów czy szyby.



Rysunek 29. Obiekt Reflection Probe odwzorowujący odbicia otoczenia (źródło: opracowanie własne)

5.3.5 System wirtualnej rzeczywistości i implementacja ruchu użytkownika.

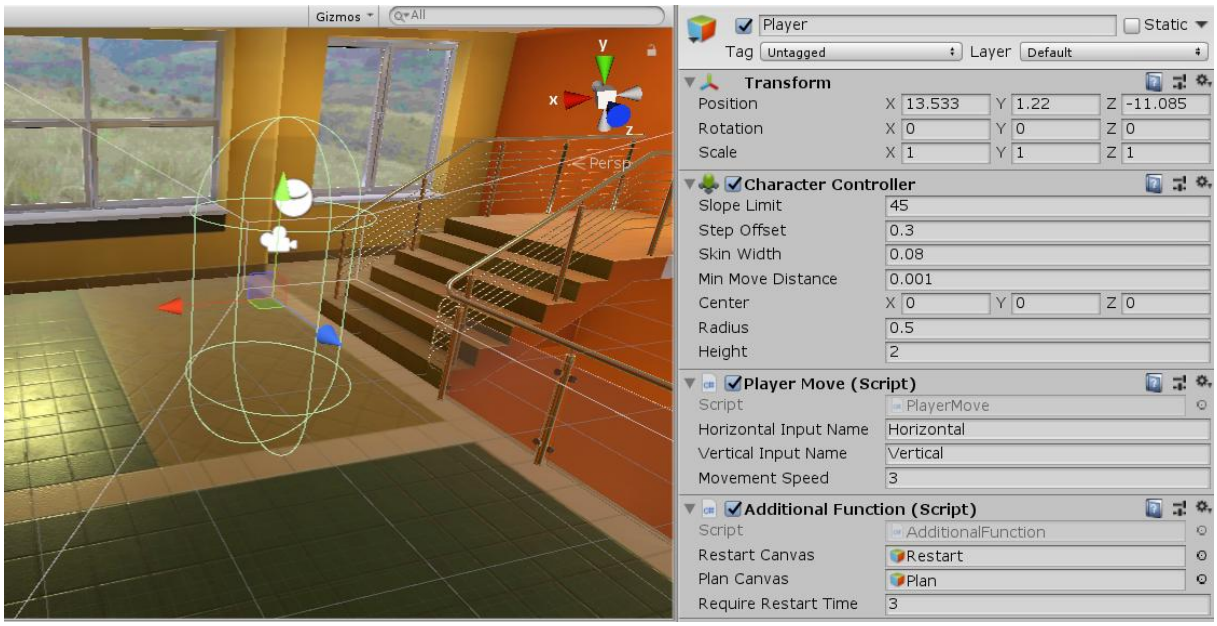
Kierując się założeniami aplikacji w kolejnych etapach przystąpiono do implementacji poruszania się użytkownika z wykorzystaniem systemu VR. Na scenie umieszczono obiekt o nazwa *Player,* do którego podpięto obiekt *Camera* zawierający komponent kamery, który odpowiada za dostosowanie i finalne renderowanie stworzonego środowiska. Obsługa systemu wirtualnej rzeczywistości w Unity jest uruchamiania przez zaznaczenie opcji wsparcie systemu VR w sekcji *Player Settings*. Warto tu zaznaczyć, że Silnik Unity pozwala korzystać z funkcji VR bez instalacji dodatkowego oprogramowania czy wtyczek.

Budując aplikację VR kierowaną na platformy mobilne, Unity dzieli ekran smart fonu na dwie części odpowiednio na prawe i lewe oko. W każdej części wyświetlany jest widok z odpowiednio dostosowaną perspektywą w celu zapewnienia stereoskopowego efektu 3D, który tworzy głębię dla użytkownika. W celu renderowania widoku dla każdego oka Unity tworzy dwie kamery i przebiega przez proces generowania stereofonicznych obrazów. Pętla renderowania (opisana we wcześniejszej części tej pracy) jest uruchamiana dwukrotnie. Każda kamera konfiguruje i uruchomia własną iterację pętli renderowania, w wyniku czego otrzymujemy dwa obrazy, które są przesyłane do telefonu. Operację te tworzą duże obciążenie dla karty graficznej i procesora, ponieważ w każdej klatce odbywa się podwójne przetwarzanie procesu renderowania. Unity w sekcji *Player Settings* umożliwia wybór jednej z metod podwójnego renderowania obrazu (*Stereo Rendering Method*) w celu optymalizacji procesu powstawania stereofonicznych obrazów. W budowanej aplikacji zastosowano metodę *Multi-Pass* gdyż jest ona wspierana na urządzeniach mobilnych z systemem Android. Ideą tego rozwiązania jest wyodrębnienie fragmentów pętli renderowania, które nie zmieniają się w zależności od kamery prawego i lewego oka. Dzięki temu proces wyznaczania obiektów,

które mają zostać wyrenderowane i obliczanie cieni odbywa się tylko raz dla widoku prawego i lewego oka. Gdyż te elementy nie są wyraźnie zależne od lokalizacji kamery, co pozwala zoptymalizować proces renderowania.

Do kamery zastosowana jest także funkcja śledzenia położenia i ruchów głowy użytkownika, co pozwala na określenie odpowiedniego położenia i orientacji kamery przed renderowanie obrazu [9]. Dzięki tym funkcją ruch obrotu kamery podczas ruchu głowy użytkownika odbywa się w sposób płynny, co zapewnia dobre wrażenie wirtualnej rzeczywistości.

Do obiektu *Player* przypisano kontroler postaci – *Character Cotroller* który jest widoczny w edytorze jako kapsuła odwzorowująca postać użytkownika poruszającego się po stworzonej scenie (rys. 30). Kontroler ten odpowiada za kolizję użytkownika z otoczeniem a także pozwala dokładnie skonfigurować sposób poruszania się w stworzonym środowisku. Konfigurację rozpoczęto od dostosowania promienia i wysokości kapsuły, aby odwzorować postać ludzką. Dodatkowo ustawiono maksymalny kąt podłoża, po którym może poruszać się użytkownik – *Slope Limit* oraz maksymalną wielkości obiektu, na który postać może wejść - *Step Offset* parametr ten jest używany do określenie maksymalnej wielkości stopnia na przykład przy poruszaniu się po schodach (rys. 30). Parametr *Skin Width* odpowiada za określenie jak bardzo użytkownik będzie mógł wnikać w inne obiekty [9].



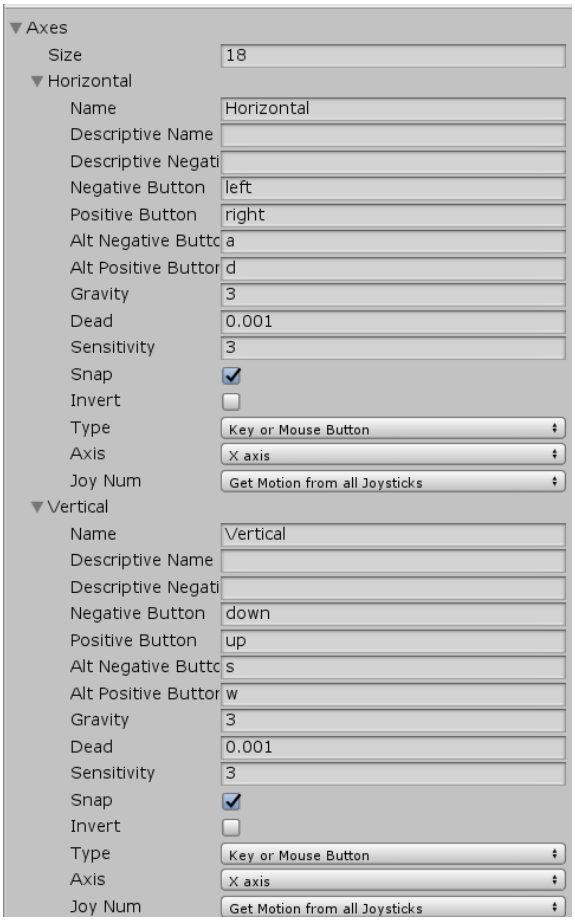
Rysunek 30. Umieszczony na scenie obiekt Player (źródło: opracowanie własne)

W kolejnym kroku utworzono i przypisano do obiektu *Player* skrypt w języku C# o nazwie *PlayerMove*, który realizuje poruszanie się postaci z wykorzystaniem bezprzewodowego kontrolera Bluetooth wchodzącego w skład używanego zestawu VR firmy Modecome. Kod skryptu zamieszczono poniżej.

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour {

    [SerializeField] private string horizontalInputName;
    [SerializeField] private string verticalInputName;
    [SerializeField] private float movementSpeed;
    private CharacterController charController;

    private void Awake()
    {
        charController = GetComponent<CharacterController>(); // odwołanie się do
komponentu CharacterController
    }
    private void Update()
    {
        PlayerMovement();
    }
    private void PlayerMovement()
    {
        float horizInput = Input.GetAxis(horizontalInputName) * movementSpeed; //
obliczenie wartości o jaką gracz ma zostać przemieszczony w przód

        float vertInput = Input.GetAxis(verticalInputName) * movementSpeed; //
obliczenie wartości o jaką gracz ma zostać przemieszczony w bok

        Vector3 forwardMovement = Camera.main.transform.forward * vertInput; //
Obliczenie wartości wektora w osi Z bazując na położeniu kamery

        Vector3 rightMovement = Camera.main.transform.right * horizInput; //
obliczenie wartości wektora w osi X bazując na położeniu kamery

        charController.SimpleMove(forwardMovement + rightMovement); // wywołanie
funkcji SimpleMove z wcześniej wyliczonymi wektorami.

    }

}
```

W skrypcie odwołano się do kontrolera postaci – *CharacterController* (który został wcześniej przypisany do obiektu *Player*) i wywołano funkcję *SimpleMove*, która realizacje poruszanie się obiektu *Player*. Zmienne *horizontalInputName* i *verticalInputName* określają przyciski odpowiedzialne za poruszanie się użytkownika na podstawie ich nazw przypisanych w oknie *InputMenager*. Z poziomu projektu w oknie *Inspector* po wybraniu obiektu *Player* określono nazwy przycisków „Horizontal” i „Vertical” nazwy te zostały dokładnie skonfigurowane w oknie *InputMenager* (rys. 31), aby umożliwić obsługę gałki na bezprzewodowym kontrolerze.



Rysunek 31. Ustawienia obsługi przycisków - InputManager (źródło: opracowanie własne)

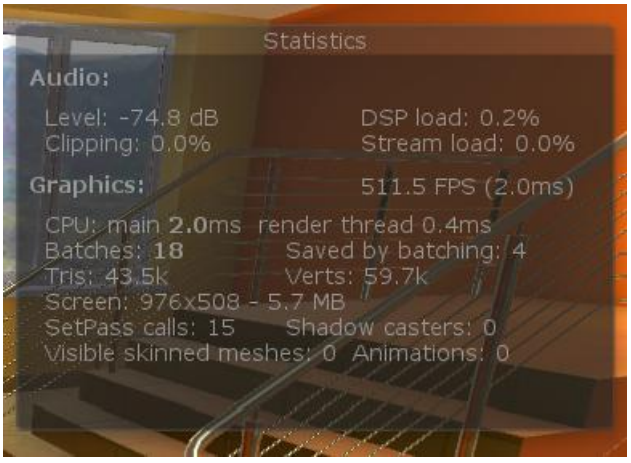
W funkcji *PlayerMovement* obliczamy wartość o jaką gracz ma zostać przemieszczony mnożąc wartość odczytaną z gałki na kontrolerze razy ustaloną szybkość poruszania się odpowiednio dla ruchu w kierunku horyzontalnym i wertykalnym - *horizInput* i *vertInput*. Wartość przycisku jest określana za pomocą klasy *Input*, które przyjmuje wartość przypisanego przycisku według ustawień dokonanych w oknie *InputManager* (rys. 31) w tym przypadku jest to wartość ujemna, gdy gałka jest przesunięta w lewo i dodatnia, jeśli gałka przesunięta jest w prawo (rys. 31). Podobnie odbywa się to dla ruchu w kierunku wertykalnym gdzie wartość jest dodatnia, gdy gałka przemieszczana jest w górę i ujemna, gdy w dół. W przypadku gdy gałka nie jest przesunięta w żadną stronę przyjmowana jest wartość 0. W kolejnym kroku do zmiennej typu *Vector3* (struktura wykorzystywana w Unity do przekazania pozycji w przestrzeni trójwymiarowej [9]) o nazwie *forwardMovement* obliczany jest wektor w osi Z czyli ruch użytkownika w przód i w tył. Podobnie odbywa się to dla ruchu horyzontalnego gdzie do zmiennej *rightMovement* obliczany jest wektor osi X, czyli ruch użytkownika w prawo i lewo. Obliczenia te są dokonywane odpowiednio dla ruchu w kierunku wertykalnym pobierając pozycję głównej aktywnej kamery przypisanej do obiektu *Player* w osi Z - Camera.*main.transform.forward* (biorąc pod uwagę także rotację obiektu) i mnożąc ją razy wcześniej obliczoną wartość o jaką użytkownik ma zostać przeniesiony. Podobnie dla ruchu w kierunku wertykalnym pobierana jest pozycja kamery przypisanej do obiektu *Player* w osi X - *Camera.main.transform.right* i mnożona razy wcześniej obliczoną wartość, o jaką użytkownik ma zostać przeniesiony. W ostatnim kroku za

pomocą wbudowanej do *CharacterController* funkcji *SimpleMove* odbywa się ruch użytkownika do wcześniej wyliczonej pozycji.

5.3.6 Testowanie i optymalizacja

Zagadnienie optymalizacji jest jednym z najważniejszych elementów każdego projektu a szczególnie aplikacji wykorzystujących VR gdzie płynność działania aplikacji i ilość klatek wyświetlanych na sekundę ma bardzo duży wpływ na wrażenie wirtualnej rzeczywistości. Proces optymalizacji rozpoczęto już na poziomie tworzenia zasobów 3D wykorzystanych w aplikacji między innymi podczas tworzenia geometrii trójwymiarowych obiektów budowano siatkę z jak najmniejszą liczbą wierzchołków unikając modelowania detali, które zostały dodane do obiektów przy pomocy tekstur i mapy normalnych czy podczas teksturowania tworząc atlasy tekstur pozwalające na wykorzystanie tej samej tekstury przez wiele obiektów. Oświetlenie ma także diametralny wpływ na wydajność projektu, dlatego wszystkie światła w projekcie są wypalane i zapisywane na mapach światła, co pozwala na wykonanie jak największej liczby obliczeń zwianych z oświetleniem wcześniej i tym samym zaoszczędzić mocy obliczeniowej urządzenia podczas działania aplikacji. Unity udostępnia szereg narzędzi pomocnych w procesie optymalizacji i testowania tworzonego projektu takich jak Pofiler czy Frame Debugger. Tworzoną aplikację testowano wielokrotnie w trakcie procesu produkcji, zarówno na docelowej platformie android z wykorzystanie VR jak i z poziomu edytora Unity, co pozwoliło na bieżąco monitorować wydajność a także wyeliminować ewentualne błędy.

Podczas testów w edytorze uruchomiono okno *Statistic* zawierające informację dotyczące wyświetlanej klatki generowane w czasie rzeczywistym działania aplikacji z poziomu edytora (rys. 32).

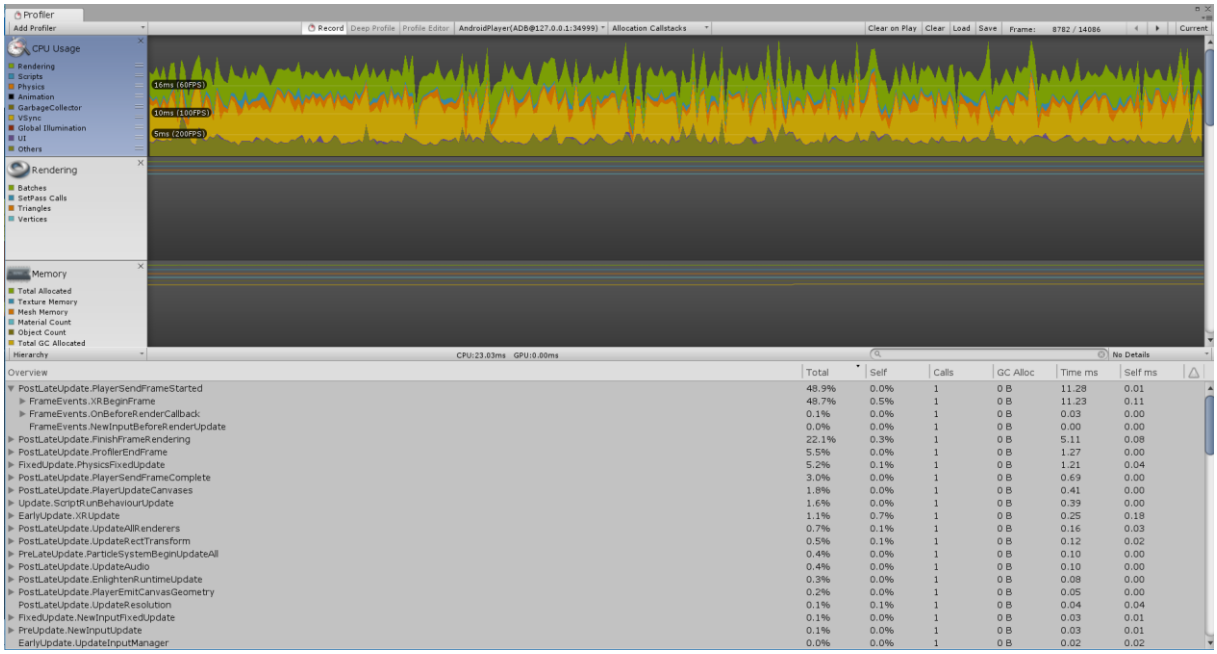


Rysunek 32. Okno statystyk uruchomionej aplikacji w silniku Unity (źródło: opracowanie własne)

W sekcji Graphic okna *Statistic* pokazywana jest między innymi ilość czasu potrzebna na przetworzenie i renderowanie jednej klatki oraz ilość wyświetlanych klatek na sekundę a także liczba rysowanych trójkątów i wierzchołków wyświetlanych modeli 3D (rys. 32). Parametr *Batches* określa ilość odwołań do karty graficznej. Aby narysować obiekt na ekranie, silnik musi odwołać się do interfejsu API karty graficznej. Pojedynczy Batch to

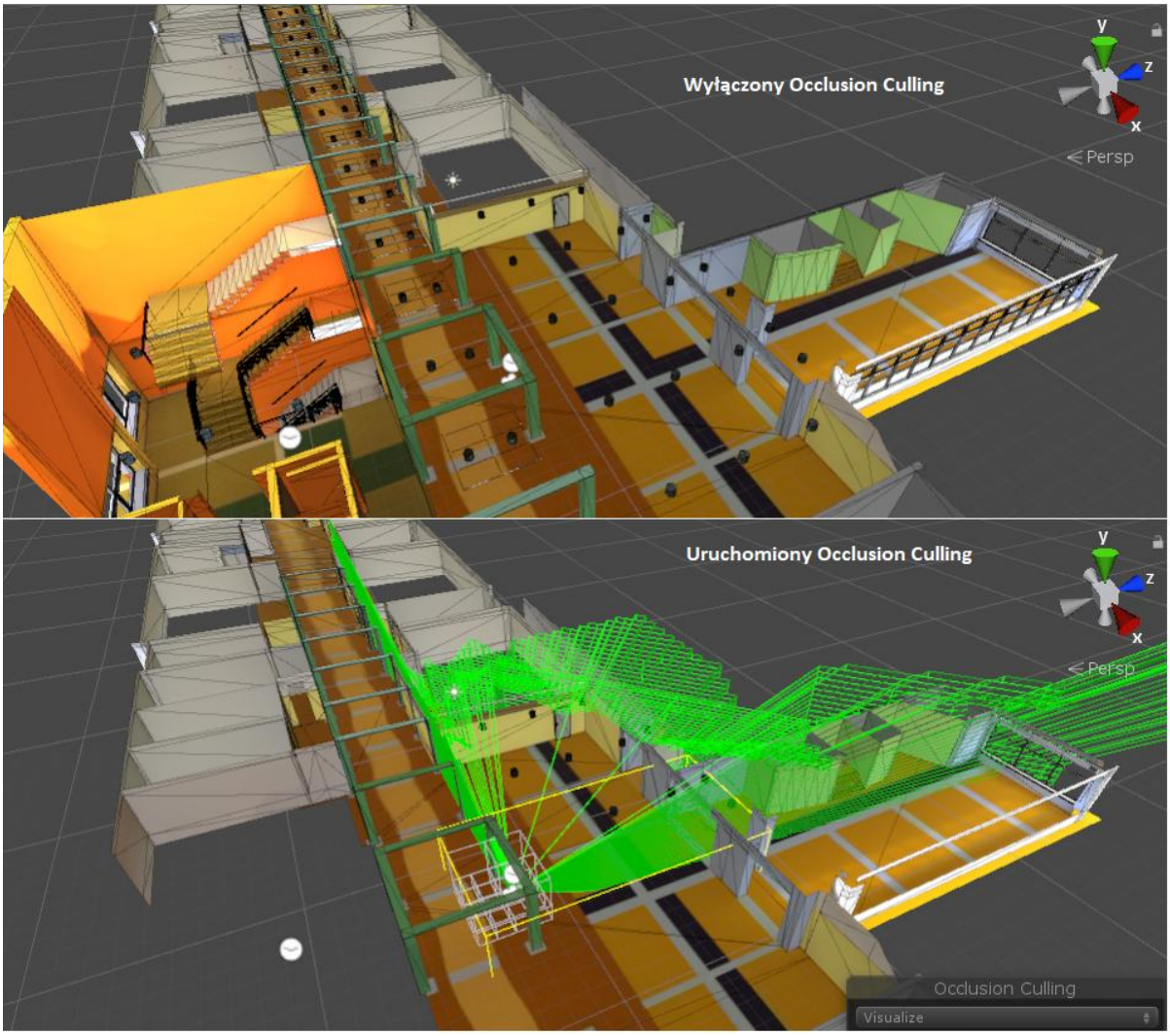
pakiet zawierający dane, który zostanie przesłany do GPU (graphics processing unit). Duża ilość tych odwołań - *Batches* bardzo mocno wpływa na wydajność, dlatego w procesie optymalizacji dąży się do jak największego zredukowania tej wartości. Aby zredukować ilość odwołań do karty graficznej Unity może łączyć ze sobą dość dużą liczbę wierzchołków geometrii w jeden pakiet (*Batching*), jednak należy pamiętać, że dany pakiet może mieć tylko raz ustawiany materiał. Oznacza to, że grupowanie działa tylko dla obiektów posiadających ten sam materiał. Grupowanie statyczne - *Static batching* polega na łączeniu statycznych nieruchomych obiektów w duże siatki, co pozwala przyśpieszyć renderowanie przesyłając je jako pojedynczy pakiet należy pamiętać, że odbywa się to tylko dla obiektów oznaczonych w oknie inspektora jako *Static*. Drugą techniką grupowania jest grupowanie dynamiczne - *Dynamic batching* grupowanie to próbuje zoptymalizować sposób renderowania obiektów niestatycznych, przekształcając ich wierzchołki w procesorze, grupując wiele podobnych wierzchołków ze sobą i rysując je za jednym razem. Grupowanie to ogranicza się do niedużych obiektów posiadających prostą geometrię. Parametr *Saved by Batching* w oknie Statistic (rys. 32) określa ilość pakietów, które udało się połączyć korzystając z powyżej opisanych technik grupowania. Aby poprawić wydajność działania aplikacji starano się jak najbardziej wykorzystać grupowanie obiektów poprzez ograniczenie liczby użytych materiałów, co było możliwe dzięki odpowiedniemu przygotowaniu modeli i tekstur między innymi dzięki łączeniu wielu tekstur w jedną (tworzenie atlasów tekstur), co zostało poruszone we wcześniejszych częściach pracy.

Dodatkowo podczas testów korzystano z rozbudowanego narzędzia *Profilier* które przechwytuje i analizuje pod kątem wydajności każdą klatkę w uruchomionej aplikacji, dostarczając szczegółowych raportów ukazujących ilość czasu pochłanianego przez każdy z obszarów działania aplikacji z podziałem na sekcje: wykorzystanie procesora, wykorzystania karty graficznej, wykorzystanie pamięci oraz audio i fizykę (rys. 33). Domyślnie *Profilier* analizuje informacje na podstawie uruchomienia aplikacji w edytorze. Jednak, aby w pełni wykorzystać potencjał tego narzędzia należy analizować wydajność bezpośrednio na uprzędzeniu docelowym, na którym aplikacja będzie finalnie uruchamiana. W tym celu podczas budowania aplikacji na telefon z poziomu edytora uruchomiono okno *Profilier* i zmieniono urządzenie, z którego są przechwytywane informacje na podłączony smartfon z systemem android.



Rysunek 33. Analiza działania aplikacji wykorzystując Unity Profiler (źródło: opracowanie własne)

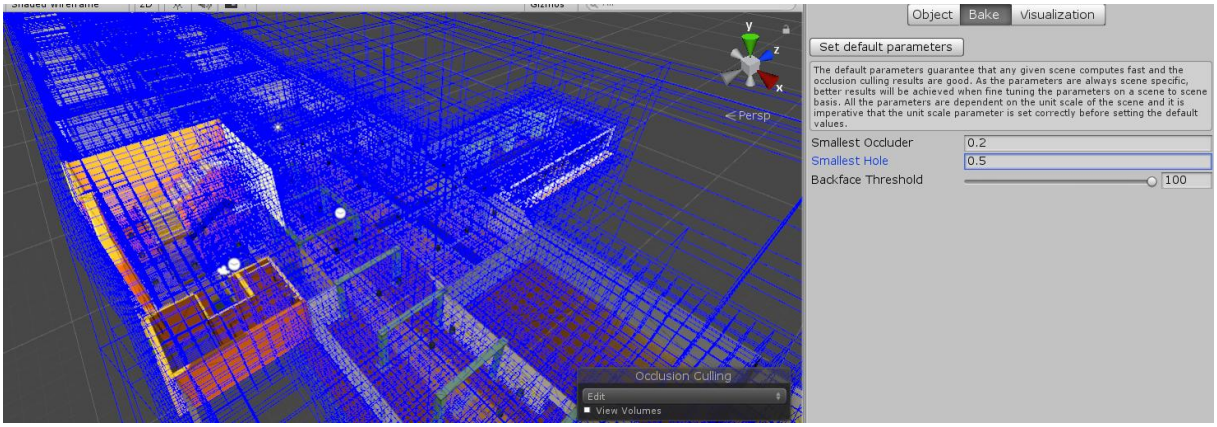
Oprócz testów i analizy aplikacji podczas procedury optymalizacji zastosowano szereg technik i narzędzi w celu uzyskania jak największej płynności tworzonej wizualizacji. Między innymi uruchomiono i skonfigurowano proces usuwania okluzji - *Occlusion Culling* który wyłącza renderowanie obiektów niewidocznych przez kamerę zarówno tych, które nie znajdują się w polu widzenia użytkownika jak i zapobiega renderowaniu geometrii zasłoniętej przez inne modele 3D znacznie ograniczając ilość rysowanych wierzchołków. Nie dzieje się to automatycznie, ponieważ w większości przypadków renderowania grafiki trójwymiarowej obiekty znajdujące się najdalej od kamery są rysowane w pierwszej kolejności, a bliższe obiekty są rysowane nad nimi (*Overdraw*). Dlatego *Occlusion Culling* różni się od funkcji *Occlusion Frustum*, która wyłącza renderowanie tylko obiektów poza obszarem wyświetlania kamery, ale nie wyłącza niczego, co jest zakryte innym obiektem. Poniższy zrzut ekranu przestawia działania funkcji *Occlusion Culling* zielonymi liniami oznaczone jest pole widzenia kamery (rys. 34).



Rysunek 34. Wizualizacja funkcji Occlusion Culling(źródło: opracowanie własne)

Proces usuwania okluzji przechodzi przez scenę za pomocą kamery wirtualnej, aby zbudować hierarchię potencjalnie widocznych zestawów obiektów. Te dane są wykorzystywane w czasie działania aplikacji przez każdą kamerę, aby określić, które modele są aktualnie widoczne dla użytkownika. Posiadając te informacje silnik przesyła do renderowania tylko aktualnie widziane obiekty, co zmniejsza liczbę odwołań do karty graficznej i znacznie zwiększa wydajność aplikacji.

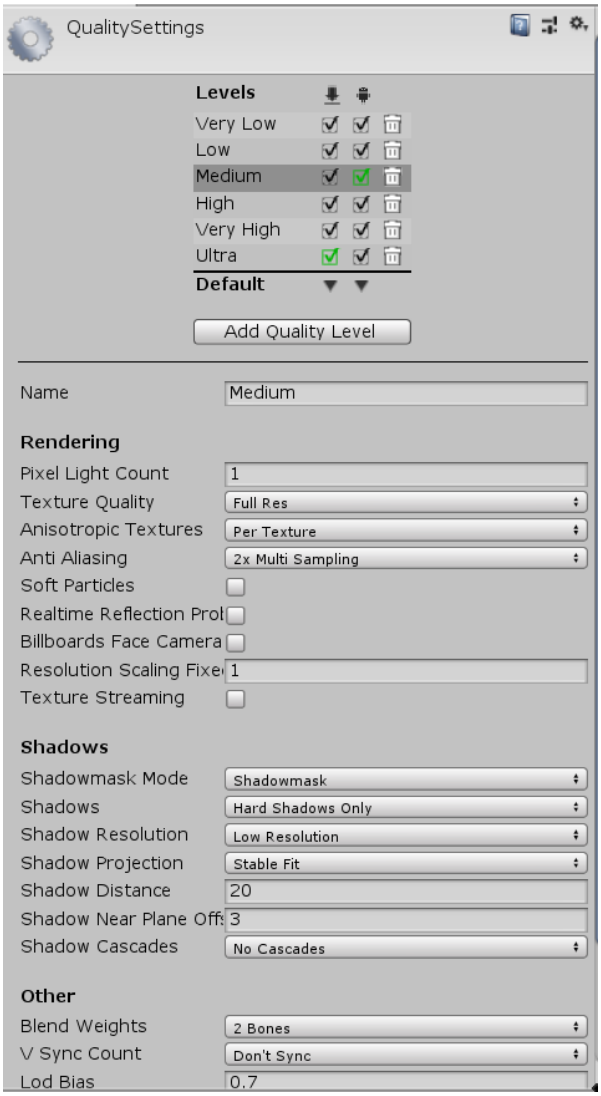
Dane dotyczące usuwania okluzji składają się z komórek, które dzielą obszar całej sceny na mniejsze obszary (rys. 35), komórki te tworzą binarne drzewo. *Occlusion Culling* używa dwóch drzew, dla obiektów statycznych - *View Cells* a dla obiektów dynamicznych - *Target Cells* [9]. Ważną kwestią jest dobra równowaga między rozmiarem obiektów a rozmiarem komórek. W zakładce *Bake* okna *Occlusion Culling* (rys. 35) dokonano odpowiednich ustawień podziału sceny na komórki dostosowując parametry *Smallest Oclluder* wyznaczający rozmiar najmniejszego obiektu, który będzie używany do ukrywania innych obiektów podczas procesu usuwania okluzji a także parametr *Smallest Hole* wyznaczający najmniejszą przerwę między geometrią, przez którą będzie brana pod uwagę podczas procesu usuwania okluzji.



Rysunek 35. Ustawienia funkcji Occlusion Culling (źródło: opracowanie własne)

Wpływ na wydajność aplikacji ma także zastosowana fizyka i obliczenia z nią związane. W celu zmniejszenia wykonywania obliczeń na procesorze ograniczono używanie kolizji bazujących na siatce modeli 3D. W tym celu dla części modeli zastąpiono komponent *mesh collider* komponentem *Box collider*, który generuje kolizje na podstawie sześcianu dopasowanego do rozmiaru obiektu.

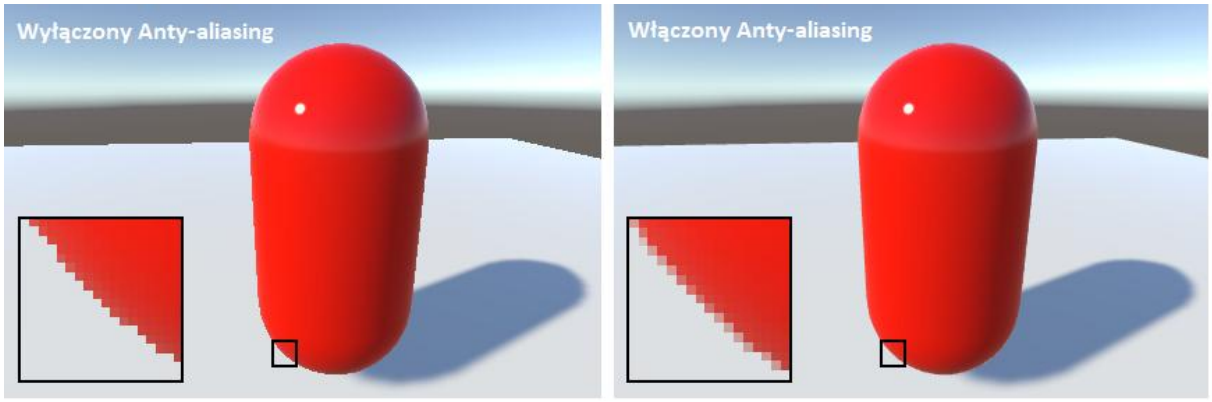
W całym procesie optymalizacji starano się jak najlepiej wyważyć jakość renderowanego obrazu a płynność działania aplikacji biorąc pod uwagę moc obliczeniową sprzętu mobilnego i dodatkowe narzuty spowodowane zastosowaniem VR. Korzystając z ustawień jakości w oknie *QualitySettings* oraz ustawień graficznych w oknie *Graphics* dostosowano ustawienia jakości finalnie renderowanej aplikacji, co ma kluczowy wpływ na wydajność. W oknie *QualitySettings* dostępne są ustawienia między innymi z zakresu renderowania obrazu czy ustawień cieni (rys. 36). Ustawienia jakości są grupowane w odpowiednich poziomach *Levels* wyznaczających jakość wyświetlanego obrazu.



Rysunek 36. Ustawienia jakości aplikacji w silniku Unity (źródło: opracowanie własne)

W procesie testowania aplikacji wybrano i przypisano do docelowej platformy (Android) poziom jakości *Medium* (rys. 37) który zapewnia płynne działanie aplikacji zachowując dostateczny poziom jakości renderowanego obrazu.

W kolejnym kroku prac nad optymalizacją dostosowano i uruchomiono funkcję wygładzania krawędzi – *Anti-Aliasing*. *Aliasing* to efekt, w którym linie wyglądają na postrzępione lub mają wygląd "schodów" (rys. 37). Najczęściej zjawisko to występuje, jeśli obraz nie jest renderowany w wystarczająco wysokiej rozdzielczości, aby dokładnie odwzorować krawędź modelu 3D. Zjawisko *Aliasingu* jest szczególnie niekorzystne w przypadku aplikacji VR, ponieważ mocno wpływa na immersje użytkownika z wirtualnym środowiskiem.



Rysunek 37. Działanie techniki wygładzania krawędzi w środowisku Unity (źródło: https://docs.unity3d.com/Manual/class-QualitySettings.html)

Proces wygładzania krawędzi polega na renderowaniu obrazu w wyższej rozdzielczości niż ten finalnie wyświetlany na ekranie smart fonu. Dzięki temu na jeden wyświetlany piksel przypada więcej pikseli, z których podczas procesu *Anti-Aliasing'u* pobierane są kolory, których wartość jest uśredniania. Dzięki temu zabiegowi przejścia między kolorami pikseli w finalnie renderowanym obrazie są stopniowe a krawędzie wyglądają przez to na mniej postrzępione (rys. 37). Dzięki zastosowaniu ścieżki renderowania *Forward Rendering* (co zostało opisane we wcześniejszej części pracy) możliwe jest wykorzystanie techniki MSAA (Multisample, anti-aliasing) która cechuje się dużo większą wydajnością stosując cieniowanie tylko do finalnie renderowanego piksela (pomijając piksele wygenerowane na potrzeby procesu wygładzania krawędzi). Należy mieć na uwadze, że pomimo stosowania techniki MSAA, wygładzanie krawędzi jest procesem wymagającym dodatkowej mocy obliczeniowej, dlatego w Oknie Ustawień jakości – *QualitySettings* (rys. 36) wybrano opcję *2x Multi Sampling* co oznacza, że rozdzielczość na bazie której działa wygładzanie krawędzi jest dwa razy większa niż rozdzielczość finalnie renderowanego obrazu. Taki poziom wygładzania krawędzi pozwala uzyskać zadowalający wizualny efekt bez dużych strat w wydajności działania aplikacji.

5.3.7 Dodatkowe funkcje i rozwój aplikacji

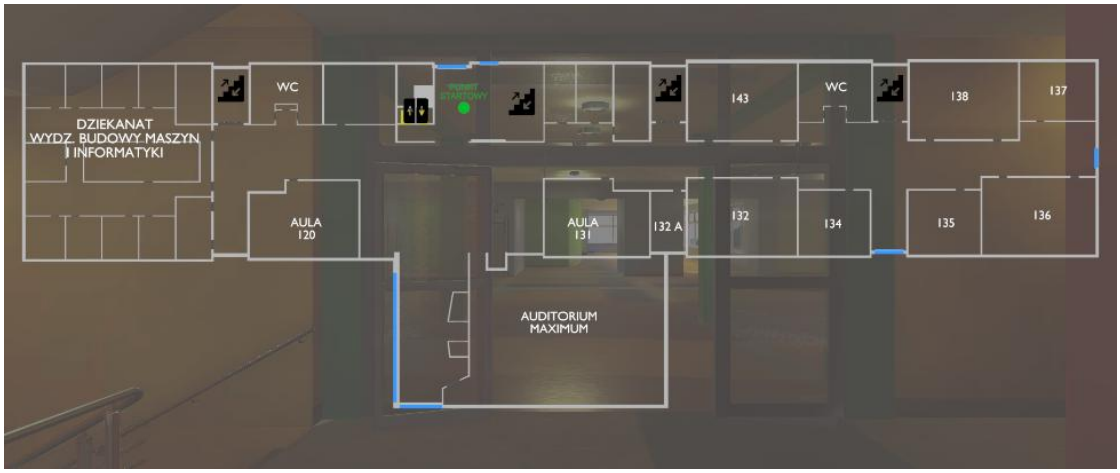
W aplikacji oprócz podstawowej funkcji poruszania się użytkownika wykorzystując bezprzewodowy kontroler zaimplementowano także dodatkowe funkcję, które pozwolą użytkownikowi na lepsze zapoznanie się z wizualizowanym obiektem oraz usprawniają korzystanie z aplikacji. Utworzono w tym celu skrypt *Additional Function*, który został przypisany do obiektu *Player*. Skrypt ten zawiera implementacje poniżej opisanych funkcji.

Funkcja restartu umożliwia w łatwy sposób zrestartowanie działającej aplikacji przy pomocy kontrolera, co pozwala na jej ponowne uruchomienie bez wyjmowania telefonu z gogli VR. Opcja ta jest uruchamiana poprzez przytrzymanie przycisku *Y* na bezprzewodowym kontrolerze. Restart aplikacji realizowany jest przez skrypt sprawdzający czy dany przycisk został wciśnięty, jeśli tak uruchamiany jest przypisany do kamery *Canvas* - obszar interfejsu użytkownika, który zwiera pole tekstowe z odpowiednim komunikatem. Gdy przycisk zostanie przytrzymany przez odpowiednią ilość czasu zdefiniowanego przy pomocy zmiennej *requireRestartTime* wywołana zostaje funkcja restartująca aplikację. Poniżej zamieszczono

funkcję *Update, w której* sprawdzane jest czy dany przycisk został wciśnięty oraz uruchamiane są funkcje wyświetlające komunikat i restartujące aplikację.

```
void Update () {

        if (Input.GetKeyDown("joystick button 3")) // sprawdzanie czy przycisk został
wciśnięty
        {
            ShowRestartCanvas(); // uruchomienie funkcji wyświetlającej komunikat
        }


          if (Input.GetKey("joystick button 3")) // sprawdzanie czy przycisk jest
trzymany
          {

            restartTimer += Time.deltaTime; // odliczanie przez jaki czas przycisk
jest wciśnięty

            if (restartTimer >= requireRestartTime)
            {
                RestartApp(); // wywołanie funkcji restartującej aplikację
            }
          }

        if (Input.GetKeyUp("joystick button 3"))
        {
            ShowRestartCanvas();
        }
    }
```

Kolejną funkcją rozszerzającą działanie aplikacji jest możliwość wyświetlenia planu wizualizowanego piętra wraz z zaznaczonymi numerami pomieszczeń (rys. 38). Funkcja ta także jest uruchamiania poprzez bezprzewodowy kontroler naciskając przycisk *A*. Po naciśnięciu przycisku wyświetlany jest odpowiedni *Canvas* z przypisanym obiektem *Image* zawierającym plan pietra budynku.



Rysunek 38. Funkcja wyświetlenia planu piętra (źródło: opracowanie własne)

Wykorzystane w procesie tworzenia wizualizacji narzędzia w tym silnik Unity 3D pozwalają na szerokie rozszerzenie działania aplikacji nie tylko poprzez umieszczenie dodatkowych obiektów architektonicznych powiększając tym samym zakres wizualizacji, ale także na utworzenie dodatkowych funkcji pozwalających na większą interakcję z prezentowanym środowiskiem.

W kolejnych wersjach aplikacji planuje się wzbogacić wirtualną przestrzeń o dodatkowe modele 3D w tym także wnętrza pomieszczeń oraz zaimplementować funkcję pozwalającą na otwieranie i zamykanie drzwi czy system wskazujący drogę do wyznaczonego pomieszczenia, co pozwoli użytkownikom na łatwe znalezienie konkretnego pomieszczenia. Dodatkowo w aplikacji można umieścić menu umożliwiające wybór piętra lub budynku, który będzie wizualizowany.

6. Wnioski

W pracy udało się zrealizować założone cele. Za pomocą narzędzia Blender opracowano model architektoniczny piętra tworząc odpowiednią geometrię trójwymiarowych obiektów dostosowaną do aplikacji mobilnych i systemu wirtualnej rzeczywistości. Następnie w programie Substance Painter utworzono realistyczne tekstury dostosowane do modelu oświetlenia PBR, które pozwalają odwzorować realistyczne zachowanie światła na powierzchni modeli i dodać do obiektów odpowiednie detale. Korzystając z możliwości silnika Unity 3D utworzono prezentowaną przestrzeń wraz z implementacją określonej funkcjonalności oraz stosując odpowiednie techniki optymalizacji zbudowano aplikację do wizualizacji obiektu w środowisku wirtualnym VR. Stworzona aplikacja z powodzeniem może zostać wykorzystana do prezentacji pierwszego piętra budynku L a dzięki umieszczeniu dodatkowych modeli (plakatów i banerów) może także posłużyć, jako narzędzie marketingowe wykorzystujące najnowsze technologie VR.

Wybrany silnik Unity jest narzędziem bardzo rozbudowanym posiadającym mnogość funkcji z zakresu grafiki i programowania, co w połączeniu z możliwością budowy aplikacji na różne platformy sprawia, że jest to środowisko bardzo elastyczne i powszechnie wykorzystywane w wielu branżach.

Wykorzystując opisane narzędzia zaproponowano schemat tworzenia trójwymiarowych modeli i tekstur oraz ich implementacje w silniku gier Unity gdzie zbudowano finalną aplikację. Ukazany schemat z powodzeniem może być wykorzystany podczas tworzenia zarówno wizualizacji, gry komputerowej, czy aplikacji instruktażowej prezentującej przykładowo działanie linii produkcyjnej.

Zbudowana aplikacja mobilna działa na smartfonie wykorzystując mobilny zestaw VR jak Samsung Gear VR, Google Card-Board, czy użyty do prezentacji efektów tej pracy zestaw Modecome Volcano Blaze. Pozwala w prosty sposób, w praktycznie każdych warunkach i niewielkim kosztem skorzystać z możliwości oferowanych przez wirtualną rzeczywistość. Dodatkowo takie rozwiązanie bazujące na wykorzystaniu telefonu nie wymaga wydajnego sprzętu i pozwala trafić do szerszego grona odbiorców. Pozwala w łatwy sposób uruchomić środowisko wirtualnej rzeczywistości unikając skomplikowanych procesów konfiguracji czy dużych wydatków powiązanych z zakupem stacjonarnych zestawów VR jak na przykład HTC Vive.

Tworząc aplikację mobilną wykorzystującą wirtualną rzeczywistość należy mieć na uwadze wysoki wzrost zapotrzebowania na moc obliczeniową wynikający z zastosowanie systemu VR a także stosunkowo niską wydajność urządzeń mobilnych, które pomimo wysokiego poziomu zaawansowania mocno odstają w porównaniu do komputerów stacjonarnych. Niesie to za sobą konsekwencje i ograniczenia w postaci niższej, jakości finalnej wizualizacji i mniejszego stopnia immersji a także mniej płynne działanie aplikacji. Mając to na uwadze przeprowadzano testy tworzonej aplikacji oraz podjęto konkretne kroki optymalizacji zarówno podczas tworzenia zasobów graficznych użytych do budowy wizualizacji między innymi poprzez uboższą siatką trójwymiarowych modeli czy tworzenie

atlasu tekstur, jak i w fazie budowy samej aplikacji stosując odpowiednie techniki optymalizacji w tym zastosowanie statycznych świateł, uruchomienie systemu Occlusion Culling czy ograniczenie efektów cząsteczkowego i post procesów. Działania te pozwalają zminimalizować niedogodności wynikające z wykorzystania urządzeń mobilnych, przy zachowaniu jak najlepszych efektów wizualnych i w jak największym stopniu wykorzystać potencjał, jaki niesie technologia wirtualnej rzeczywistości. Dzięki czemu ukazano finalny projekt w bardziej atrakcyjnej formie, zastępując klasyczną wizualizację w postaci renderowanych dwuwymiarowych obrazów interaktywnym środowiskiem.

Schemat działania wykorzystujący platformy mobilne i VR sprawia, że aplikacje w tym wizualizacje architektoniczne wykorzystujące wirtualną rzeczywistość mogą być powszechnie stosowane nie tylko przez wyspecjalizowane firmy zajmujące się wirtualną rzeczywistością, ale także niewielkie przedsiębiorstwa. Głównie z uwagi na niewielki koszt mobilnych gogli VR i niski próg wejścia, co sprawia że technologia ta coraz częściej jest z powodzeniem wykorzystywana w warunkach domowych i akademickich.

Należy także zwrócić uwagę na ubogą dostępność naukowych opracowań dotyczących wykorzystania i implementacji wirtualnej rzeczywistości w aplikacjach w tym wizualizacji architektonicznych, co może stanowić trudność w powszechnym tworzeniu oprogramowania wykorzystującym VR. Podczas prac często bazowano na własnym doświadczeniu płynącym z pracy w branży gier i materiałach internetowych. Brak wyspecjalizowanej bibliografii w głównej mierze może być spowodowany niedawnym unowocześnieniem i upowszechnieniem technologii wirtualnej rzeczywistości, przez co jest ona wciąż mało popularna głównie w Polsce. Jednak biorąc pod uwagę rosnący rozwój technologii informatycznych w tym technologii wirtualnej rzeczywistości (i rzeczywistości rozszerzonej) stosowanie i wykorzystanie systemów VR staje się coraz bardziej popularne i upowszechniane nie tylko w branży rozrywkowej i marketingowej, ale także przenika do kolejnych dziedzin i sfer użytkowych, zaspokajając rosnące wymagania końcowych odbiorców i społeczeństwa nastawionego w coraz większym stopniu na konsumpcję wirtualnych treści.

Bibliografia:

1. Chlipalski P.: Blender 2.69. Architektura i projektowanie, Helion, Gliwice, 2014
2. Geig M.: Unity. Przewodnik projektanta gier, Helion, 2015
3. Rogińska-Niesłuchowska M.: Wizualizacja- Element warsztatu współczesnego architekta. Przestrzeń i forma 11/2009, s. 183-190.
4. Simonds B.: Blender. Praktyczny przewodnik po modelowaniu, rzeźbieniu i renderowaniu, Helion 2014
5. Świt- Jankowska B.: Wizualizacja architektoniczna, jako współczesne narzędzie pracy architekta, Politechnika Poznańska, Poznań, 2011
6. Traczyk R.: Multimedia i grafika komputerowa. Podstawowe pojęcia z zakresu grafiki komputerowej, CKU Koszalin, 2010

Materiały internetowe:

7. Dokumentacja do programu Blender: https://docs.blender.org/manual/en/dev/index.html, data pobrania: 15.10.2018
8. Dokumentacja do programu Substance Painter: https://support.allegorithmic.com/documentation/spdoc/substance-painter-20316164.html, data pobrania: 15.10.2018
9. Dokumentacja do programy Unity: https://docs.unity3d.com/Manual/index.html, data pobrania: 15.10.2018
10. Draw Call Batching: https://docs.unity3d.com/Manual/DrawCallBatching.html, data pobrania: 15.10.2018
11. Grafika 3D: https://pl.wikipedia.org/wiki/Grafika_3D , data pobrania: 20.12.2018
12. HTC Vive: https://en.wikipedia.org/wiki/HTC_Vive , data pobrania: 20.12.2018
13. Moduły cieniujące w Unity: https://unity3d.com/learn/tutorials/topics/graphics/gentle-introduction-shaders, data pobrania: 03.01.2019
14. Optimisation for VR in Unity: https://unity3d.com/learn/tutorials/topics/virtual-reality/optimisation-vr-unity data pobrania: 15.10.2018
15. Optymalizacja mobile VR: https://unity3d.com/how-to/optimize-mobile-VR-games data pobrania: 15.10.2018
16. Struktura pliku FBX: https://banexdevblog.wordpress.com/2014/06/23/a-quick-tutorial-about-the-fbx-ascii-format , data pobrania: 03.01.2019
17. Wirtualna Rzeczywistość: https://en.wikipedia.org/wiki/Virtual_reality, data pobrania: 20.12.2018

Spis rysunków:

Zawartość płyty DVD:

- Praca dyplomowa – wersja źródłowa (.docx)
- Praca dyplomowa (.pdf)
- Skompilowana aplikacja – AthVisualization (.apk)
- Projekt źródłowy Unity