POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH

PRACA DYPLOMOWA INŻYNIERSKA

NA KIERUNKU INFORMATYKA

Wirtualna Przymierzalnia 3D

3D Virtual Fitting Room

Autor:

Marta Kornaszewska

Współautor:

Monika Kogut

Promotor:

dr inż. Paweł Kotowski

Warszawa, styczeń 2015

....

podpis promotora podpisy autora

Streszczenie

Celem pracy dyplomowej było stworzenie aplikacji wykorzystującej rozszerzoną rzeczywistość, która imituje sklepową przymierzalnię. W stworzonej aplikacji dostępny jest szeroki wybór części garderoby w różnych wzorach i kolorach. Dodatkowo podzielone są one na kategorie: damską i męską. Program umożliwia użytkownikowi przymierzenie ubrań i akcesoriów przez nałożenie modelu 3D z wybranym elementem garderoby na obraz uzyskany z kamery sensora Kinect.

Model automatycznie dopasowuje się do sylwetki użytkownika, jednak możliwa jest manualna zmiana rozmiaru i pozycji według potrzeb użytkownika. Części garderoby dostosowują są do sylwetki nawet w trakcie ruchów i obrotów zmieniając odpowiednio swoje położenie i wielkość. Dostępna jest również funkcja wykonywania pamiątkowych zdjęć zapisywanych na komputerze użytkownika.

Aplikacja wykorzystuje czujnik ruchu Kinect. Urządzenie nie tylko dostarcza obraz z kamery, lecz także pozwala na interakcję z użytkownikiem dzięki wbudowanym mechanizmom rozpoznawania szkieletu. Użytkownik steruje aplikacją za pomocą gestów, dzięki czemu obsługa programu jest prosta i intuicyjna. Przy implementacji programu nie został bezpośrednio wykorzystany strumień głębokości. Uzyskiwany jest on przez odczytywanie fal podczerwonych wysyłanych przez emiter. Przetworzone dane ze strumienia mogą zostać użyte przy dalszym rozszerzaniu aplikacji i przyczynić się do dokładniejszego dopasowania modeli 3D do sylwetki użytkownika.

Niniejsza praca przedstawia teoretyczne podstawy, technologie oraz algorytmy, które zostały wykorzystane przy tworzeniu programu *Wirtualna Przymierzalnia 3D*. Wyjaśniono mechanizmy komunikacji z sensorem oraz opisano fragmenty kodu odpowiedzialne za najbardziej kluczowe funkcje aplikacji takie jak rotacja, translacja lub skalowanie modelu 3D. Przedstawiono również jej wygląd i przybliżono metody odpowiedzialne za rozpoznawanie gestów, które zostało wykorzystane przy obsłudze przycisków oraz sterowaniu menu aplikacji.

W pracy zawarto informacje na temat zaimplementowanych konceptów, wzorców projektowych, a także wykorzystanych bibliotek zewnętrznych. Dodatkowo scharakteryzowano komercyjne rozwiązania, przedstawiono kolejne fazy implementacji oraz problemy napotkane podczas tworzenia programu.

Abstract

The purpose of the thesis was to create an application that uses the augmented reality to create a virtual fitting room. The program provides a great variety of clothes and accessories available in numerous patterns and colours. Additionally they are divided into categories: for women and men. The program enables the user to try on clothes and accessories by applying a 3D model with a chosen item onto the image obtained from the Kinect sensor camera.

The model automatically fits the user's posture, however it is also a possible to resize or change the position of the model manually according to the needs of the person. Clothing items adjust to the silhouette even during movement and rotations changing their position and size. There is also the possibility of taking a photo which is saved on user's computer.

The application uses Kinect sensor. Not only does the device provide the application with an image from the camera, but it also enables the interaction with the user thanks to built-in skeleton tracking mechanisms. The user controls the program with gestures, which makes the application intuitive and easy to handle. During the implementation of the program the depth stream wasn't used directly. The stream is obtained by interpretation of the infrared waves sent from the emitter. Processed stream data can be used during the further development of the application and contribute in more precise adjustment of the 3D models to user's posture.

The following thesis presents theoretical aspects, technologies and algorithms used during the implementation of *3D Virtual Fitting Room*. The mechanisms of communication with the sensor as well as the most important parts of the code, responsible for essential parts of the application, such as rotation, translation and scaling the 3D model were briefly described. The construction of the application and the methods responsible for gesture recognitions, which were used for button handling and overall menu control, were also introduced.

The thesis contains information about implemented concepts, design patterns and outsourced libraries that have been used. Additionally commercial solutions were described, as well as phases of implementation and problems faced during the implementation.

Spis treści

1. Wstęp

Rozszerzona rzeczywistość to zagadnienie nowe, lecz dynamicznie rozwijane w ostatnich latach. Wykorzystywana jest zarówno w aplikacjach mobilnych, jak i desktopowych. Pozwala na łączenie elementów generowanych komputerowo z obrazem ze świata rzeczywistego. Obraz przetwarzany jest za pomocą kamery urządzenia. Może to być wbudowana kamera, jak w przypadku telefonów czy tabletów, lub też zewnętrzne urządzenie. Aplikacje wykorzystujące rozszerzoną rzeczywistość najczęściej sterowane są za pomocą gestów użytkownika. Dzięki temu programy takie mają szerokie zastosowania. Mogą służyć nie tylko rozrywce i reklamie, lecz także nauce.

W połączeniu z rozszerzoną rzeczywistością wykorzystywana jest najczęściej grafika 3D, która daje znacznie bardziej realistyczne efekty niż grafika 2D. Dzięki dostępności coraz lepszych kart graficznych i procesorów, możliwe jest generowanie modeli niemal identycznych z rzeczywistymi. Przez zrównoleglenie odświeżania grafiki i prowadzenia obliczeń, modele wyświetlane są w czasie rzeczywistym. Sprawia to, że program płynnie reaguje na zmiany otoczenia oraz interakcję użytkownika.

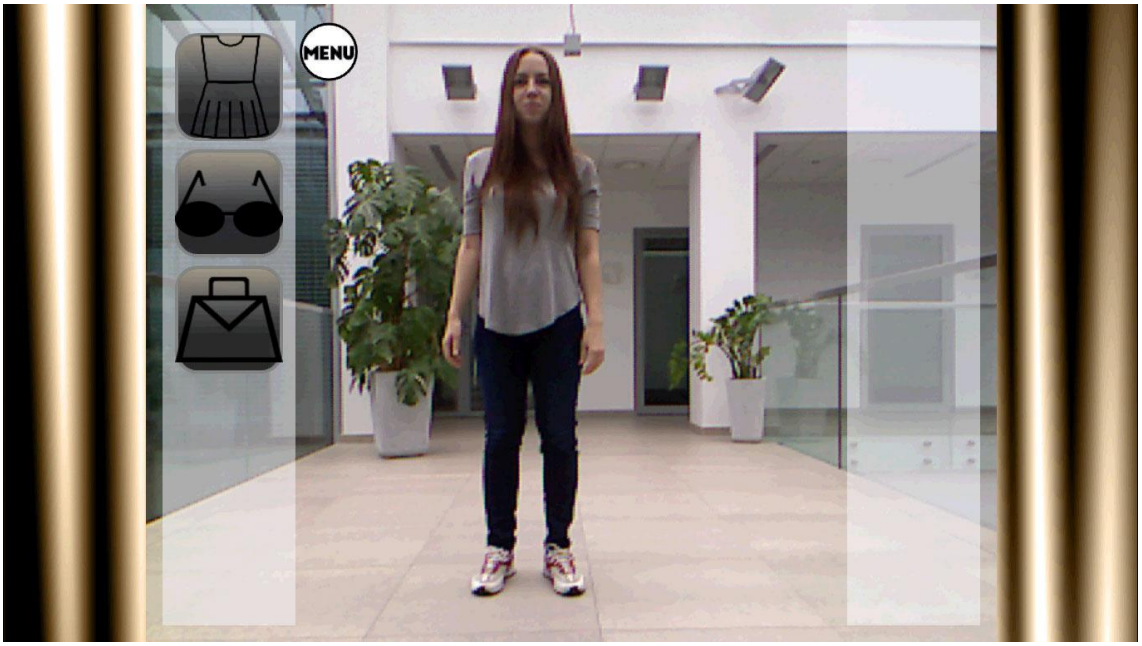
1.1. Opis programu

Aplikacja umożliwia użytkownikowi przymierzanie ubrań i akcesoriów. Zapewniony jest szeroki wybór części garderoby w różnych wzorach i kolorach. Dodatkowo podzielone są one na kategorię damską i męską.

Nakładane modele ubrań przylegają do użytkownika nawet w trakcie jego ruchów i obrotów. Są one automatycznie dopasowywane do szkieletu, jednak gdy rozmiar nie satysfakcjonuje użytkownika, może on ręcznie zmienić szerokość i wysokość ostatnio dodanego elementu poprzez użycie przycisków w górnym menu. Istnieje również możliwość zmiany pozycji modelu. Dzięki dedykowanym przyciskom w górnym menu, użytkownik może przesunąć ubranie odpowiednio w górę lub w dół.

Gdy użytkownik uzna wybrany zestaw za godny zapamiętania, może skorzystać z funkcji robienia zdjęć. Pliki ze zdjęciami nazywane są zgodnie z datą wykonania, aby zapewnić łatwość w znajdowaniu fotografii i utrzymać porządek w folderze.

Jeśli wybrany zestaw nie odpowiada użytkownikowi, ma on możliwość usunięcia ostatnio dodanego elementu lub wszystkich wybranych ubrań i zacząć zabawę z programem od początku.



Rys. 1 Główne okno aplikacji *Wirtualna Przymierzalnia 3D*

1.2. Aplikacje komercyjne

Istnieje kilka komercyjnych projektów, które umożliwiają użytkownikowi wirtualne przymierzanie rożnych części garderoby. Są to na przykład: *Dressing Room* firmy Fitnect, *Fitting Room* stworzone przez NICE oraz *EON Interactive Mirror* napisane przez EON Reality. Część z nich wykorzystano w kampaniach promocyjnych zagranicznych sklepów odzieżowych. Dzięki temu użytkownicy mogli przetestować programy oraz ocenić ich atrakcyjność i użyteczność. Aplikacje wzbudzały duże zainteresowanie, zapewniając przy tym świetną zabawę.

Programy komercyjne wiernie imitują ubrania. Dopasowują się do sylwetki użytkownika i zmieniają położenie zgodnie z ruchem szkieletu. Podczas obracania i przesuwania się użytkownika, części garderoby zachowują się zgodnie z rzeczywistością: zmieniają swoje położenie, obracają się, falują. Nawet przy robieniu gwałtownych ruchów kształt ubrania dostosowuje się do aktualnej pozycji ciała.

Rys. 2 Okno aplikacji *Dressing Room* firmy Fitnect

1.3. Charakterystyka pracy pisemnej

W skład pracy opisowej wchodzą wstęp, zakończenie oraz cztery rozdziały.

Pierwszy z nich poświęcono opisowi urządzenia Kinect. Uwzględniono konstrukcję sensora, strumienie danych oraz minimalne wymagania sprzętowe konieczne do pisania programów wykorzystujących to urządzenie.

W rozdziale drugim przybliżono narzędzia umożliwiające pracę z grafiką trójwymiarową. Przedstawiono również strukturę plików z modelami 3d.

Kolejny rozdział zawiera szczegółowe informacje dotyczące aplikacji *Wirtualna Przymierzalnia 3D* wykonanej w ramach projektu inżynierskiego. Opisano zastosowany wzorzec projektowy, metody rozpoznające gesty oraz przekształcenia modeli 3D. Przedstawiono także wygląd aplikacji.

Czwarty rozdział ukazuje proces tworzenia programu. Scharakteryzowano napotkane problemy w trakcie pracy nad aplikacją.

1.4. Podział zadań

Rozdziały pracy opisowej poświęcone sensorowi Kinect, trójwymiarowej grafice, architekturze aplikacji oraz metodom śledzenia szkieletu zostały opracowane przez Monikę Kogut. Wstęp pracy, proces tworzenia aplikacji oraz zakończenie zostały opisane przez Martę Kornaszewską. Autorka przedstawiła także wygląd i funkcjonalności programu komputerowego.

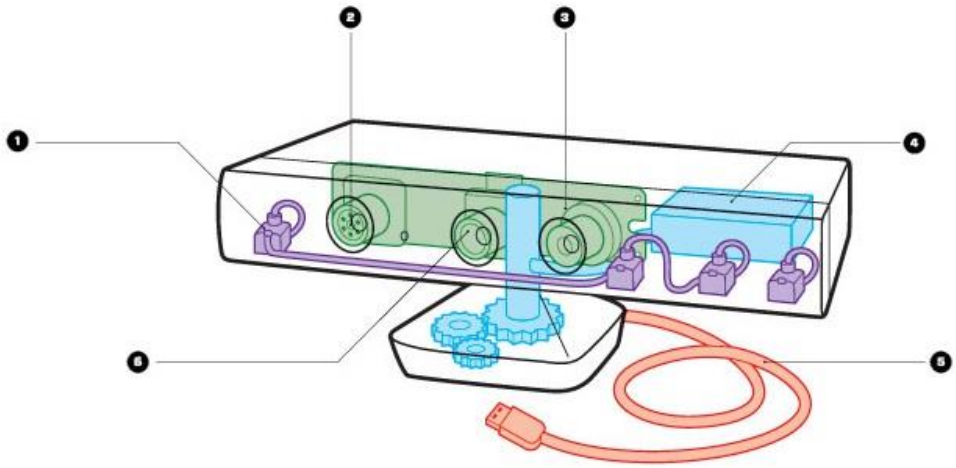
Moduły aplikacji odpowiedzialne za komunikację z Kinectem oraz grafikę trójwymiarową zostały zaimplementowane przez Monikę Kogut. Interfejs użytkownika, szata graficzna, a także przygotowanie i obróbka modeli 3D zostały zapewnione przez Martę Kornaszewską.

2. Sensor Kinect

Rys. 3 Sensor Kinect

Kinect jest urządzeniem wejściowym wyprodukowanym przez firmę Microsoft. Po raz pierwszy zaprezentowany został w 2009 roku na targach E3 w Los Angeles. Jest czujnikiem ruchu dla konsoli Xbox 360, lecz dzięki specjalnej przejściówce możliwe jest także korzystanie z sensora na komputerze. Urządzenie komunikuje się z komputerem za pomocą portu USB 2.0. Niestety niemożliwe jest korzystanie z sensora podłączonego do gniazda USB 3.0. Kinect pozwala na interaktywną rozrywkę bez użycia dodatkowych kontrolerów.

2.1. Budowa sensora



Rys. 4 Budowa sensora Kinect

1. Zestaw czterech mikrofonów. Posiadają one funkcję filtrującą zakłócenia, pozwalającą uzyskać dźwięk w bardzo wysokiej jakości. Dzięki mikrofonom możliwe jest rozpoznawanie mowy.
2. Emiter wysyłający wiązki fal podczerwonych. Wiązki te odbijają się od napotkanych przedmiotów i wracają do urządzenia.
3. Kamera głębokości odczytująca i analizująca fale podczerwone wysyłane przez emiter. Umożliwia to odtworzenie przestrzeni i obiektów znajdujących się w zasięgu sensora.
4. Automatyczny kontroler nachylenia.
5. Kabel ze złączem USB 2.0. Służy do podłączenia sensora z konsolą. Aby korzystać z urządzenia na komputerze niezbędny jest dodatkowy kabel pozwalający na podpięcie Kinecta do źródła zasilania.
6. Kamera RGB o maksymalnej rozdzielczości 1280x960 px. Przesyła do 30 klatek na sekundę w zależności od ustawionego trybu pracy.

2.2. Wymagania sprzętowe

Pisanie aplikacji z wykorzystaniem sensora Kinect wymaga instalacji Kinect SDK. Należy zaznaczyć, że przed instalacją Kinect SDK oraz innego niezbędnego oprogramowania nie powinno się podłączać sensora.

Minimalne wymagania sprzętowe niezbędne do instalacji sensora to:

- Windows 7 (x86 lub x64),
- dwurdzeniowy procesor 2,66 GHz,
- karta graficzna kompatybilna z DirectX 9.0c,
- 2GB pamięci RAM.

Oprogramowanie niezbędne do tworzenia aplikacji z wykorzystaniem sensora Kinect to:

- Microsoft Visual Studio 2010,
- Microsoft .NET Framework 4.0

- Microsoft Kinect SDK.

Przy tworzeniu aplikacji *Wirtualna Przymierzalnia 3D* korzystano z SDK w wersji 1.8, Microsoft Visual Studio 2013 oraz Microsoft .NET Framework 4.5.1.

2.3. Użycie sensora w aplikacji

Aby korzystać z urządzenia Kinect w aplikacji, należy dodać do referencji projektu bibliotekę *Microsoft.Kinect*. Poniższy kod przedstawia sposób korzystania z sensora. Program *Wirtualna Przymierzalnia 3D* korzysta jedynie z jednego urządzenia, co uwzględnione jest w poniższym przykładzie.

```
private KinectSensor _kinect;

private void DiscoverKinectSensors()
{
    KinectSensor.KinectSensors.StatusChanged +=
                                            KinectSensor_StatusChanged;
    _kinect = KinectSensor.KinectSensors.FirstOrDefault(
                        x => x.Status == KinectStatus.Connected);

    // Kod umożliwiający korzystanie ze strumieni

    if (_kinect != null)
        _kinect.Start();
}
```

Powyższy kod powoduje zapisanie się na zdarzenie *StatusChanged*, które wywołane zostaje w momencie, kiedy jeden z podłączonych sensorów zmieni swój status (zostanie podłączony, odłączony itp.). Zmiana zostanie obsłużona w metodzie *KinectSensor_StatusChanged*, co pominięto w dokumencie ze względu na przejrzystość kodu. W kolejnej linii do właściwości *Kinect* przypisany zostanie pierwszy z sensorów, który jest podłączony i gotowy do działania. Następnie powinna znaleźć się sekcja umożliwiająca korzystanie ze strumieni, co opisano w dalszych rozdziałach. Ostatnim etapem jest włączenie sensora przez wywołanie na nim metody *Start*.

2.4. Strumienie danych

Kinect SDK umożliwia pobieranie strumieni danych przetworzonych przez sensor. W wersji Kinect SDK 1.8 dostępne są strumienie:

- Color (obrazu z kamery),
- Skeleton (szkieletowy),
- Depth (głębokości).

Możliwe są dwa mechanizmy pobierania strumieni: *Polling model* oraz *Event model*. Polling model jest najprostszym mechanizmem odczytu danych. Polega on na cyklicznym odpytywaniu urządzenia o dostępność nowych informacji. Ustalony zostaje odstęp czasu, co który sprawdzane jest, czy ramka z nowymi danymi została już udostępniona przez sensor. Używany jest najczęściej przy korzystaniu z technologii XNA, oraz innych opartych o mechanizm pollingu. Model zdarzeń jest znacznie bardziej kompleksowy. W momencie nadejścia nowej ramki, program zostaje poinformowany o dostępności uaktualnionych danych.

W programie *Wirtualna Przymierzalnia 3D* wykorzystany został mechanizm zdarzeń. Zastosowano go do strumieni obrazu z kamery oraz szkieletowego. Strumień głębokości nie został bezpośrednio wykorzystany w aplikacji. Przedstawiony w dalszych rozdziałach sposób pobierania danych z sensora jest podstawą do obsługi strumieni danych. Przy operowaniu ramkami należy uwzględnić obsługę wyjątków oraz sytuacje szczególne, które uwzględniono w kodzie aplikacji, jednak ze względu na przejrzystość dokumentu zostały pominięte w opisie.

2.5. Strumień obrazu z kamery

Podstawowym strumieniem danych przetwarzanym przez Kinect jest obraz z kamery RGB. Sensor przetwarza obraz o rozdzielczości do 1280x960 px z szybkością do 30 klatek na sekundę. Dostępne są trzy formaty koloru, w których dane mogą być przesyłane z sensora:

- RGB - dostępny w rozdzielczości 640x480 px (30 FPS lub 15 FPS) oraz 1280x960 px i 12 FPS,
- YUV - potrzebujący mniej pamięci do trzymania obrazu. Dostępny w rozdzielczości 640x480 px i 15 FPS,
- Bayer - dostępny w rozdzielczości 1280x960 px i 12 FPS lub 640x480 px i 30 FPS.

Możliwy jest wybór i obsługa tylko jednego formatu jednocześnie. W programie *Wirtualna Przymierzalnia 3D* wykorzystany został format RGB o rozdzielczości 640x480 px oraz szybkością 30 klatek na sekundę.

Poniższy kod przedstawia sposób utworzenia bitmapy do przechowywania obrazu z kamery oraz sposób zapisania się na zdarzenie otrzymania nowej ramki danych. Pokazano także jak powinien być używany strumień danych.

```
private WriteableBitmap _kinectCameraImage;
private Int32Rect _cameraSourceBounds;
private int _colorStride;

private void InitializeKinectSensor(KinectSensor sensor)
{
    // Obsługa strumienia szkieletowego

    WriteableBitmap colorStream = sensor.ColorStream;
    colorStream.Enable();

    _kinectCameraImage = new WriteableBitmap(
                colorStream.FrameWidth, colorStream.FrameHeight
                , 96, 96, PixelFormats.Bgr32, null);

    _cameraSourceBounds = new Int32Rect(0, 0
                , colorStream.FrameWidth, colorStream.FrameHeight);
    _colorStride = colorStream.FrameWidth
                *    colorStream.FrameBytesPerPixel;
```

```
        sensor.ColorFrameReady +=
                        KinectSensor_ColorFrameReady;
    }

    private void KinectSensor_ColorFrameReady(object sender
                                        , ColorImageFrameReadyEventArgs e)
    {
        using (var frame = e.OpenColorImageFrame())
        {
            if (frame == null)      return;
            var pixels = new byte[frame.PixelDataLength];
            frame.CopyPixelDataTo(pixels);
            _kinectCameraImage.WritePixels(_cameraSourceBounds
                        , pixels, _colorStride, 0);
        }
    }
```

Metoda *InitializeKinectSensor* umożliwia korzystanie ze strumienia obrazu z kamery za pomocą metody *Enable*. Następnie inicjalizowana jest bitmapa *_kinectCameraImage*, do której kopiowane będą piksele ze strumienia. Bitmapa jest tworzona w rozmiarze obrazu z kamery, czyli 640x480 px, 96 dpi w formacie BGR32. Zmienna *_cameraSourceBounds* określa wielkość i położenie bitmapy, ustawiając jej pozycję na punkt (0, 0) oraz wielkość na rozmiar obrazu z kamery. Liczba *_colorStride* określa ilość bajtów przypadającą na jeden rząd obrazu. Jest ona iloczynem szerokości bitmapy i ilości bajtów opisujących pojedynczy piksel. Ostatnie wywołanie w metodzie *InitializeKinectSensor* powoduje zapisanie na zdarzenie otrzymania nowej ramki z danymi, które obsłużone będzie przez metodę *KinectSensor_ColorFrameReady*.

Metoda *KinectSensor_ColorFrameReady* otwiera ramkę obrazu z kamery przez wywołanie *OpenColorImageFrame*. Dane zostają przypisane do zmiennej lokalnej *frame*. Następnie następuje sprawdzenie, czy pobrane dane nie są puste. Zmienna *pixels* zostaje zainicjalizowana nową tablicą typu *byte* o długości odpowiadającej danym otrzymanym z ramki. W kolejnym kroku dane zostają skopiowane do tablicy przez wywołanie *CopyPixelDataTo*. Bity z tablicy *pixels* zapisywane są następnie do bitmapy *_kinectCameraImage.*

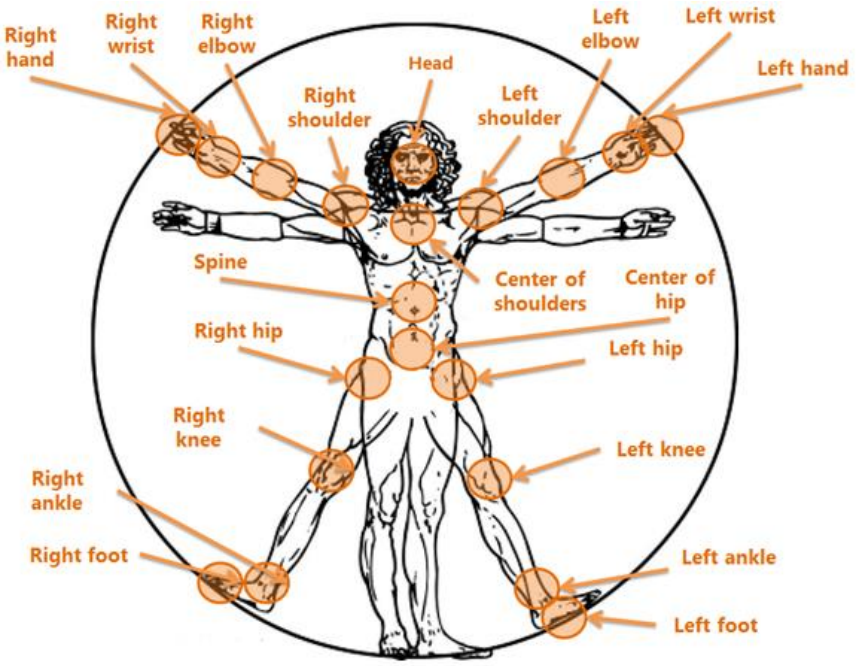
2.6. Strumień szkieletowy

Kinect SDK umożliwia dostęp do danych szkieletowych uchwyconych przez sensor. Dzięki temu możliwe jest rozpoznanie sześciu szkieletów oraz śledzenie aktywne maksymalnie dwóch. Szkielet znajdować się może w jednym z trzech stanów:

- Tracked (śledzony aktywnie),
- NotTracked (niesledzony),
- PositionOnly (śledzenie pasywne).

Tryb pasywny dostarcza jedynie danych o położeniu szkieletu w przestrzeni. Aktywne śledzenie szkieletu udostępnia dodatkowo położenie i dane szczegółowe na temat poszczególnych punktów sylwetki (Joint). Takie dane przesyłane są w tablicy indeksowanej poszczególnymi punktami szkieletowymi. Każdy z punktów znajdować się może w jednym z trzech stanów:

- Tracked (śledzony),
- NotTracked (niesledzony),
- Inferred (położenie punktu jest estymowane).



Rys. 5 Punkty szkieletowe

Szkielet w SDK Kinect opisany jest dwudziestoma punktami, tzw. *Joints*. Dzięki

informacjom o poszczególnych punktach programista jest w stanie rozpoznać nie tylko położenie człowieka względem sensora, lecz także pozę, w której się znajduje. Umożliwia to implementację rozpoznawania gestów i efektywną interakcję użytkownika z aplikacją.

Ramki danych szkieletowych otrzymywane są za pomocą mechanizmu zdarzeń. Poniższy kod przedstawia sposób zapisania się na zdarzenie nadejścia nowej ramki szkieletowej.

```
private Skeleton[] _skeletons;

private void InitializeKinectSensor(KinectSensor sensor)
{
    sensor.SkeletonStream.Enable();
    _skeletons = new Skeleton[
                sensor.SkeletonStream.FrameSkeletonArrayLength];
    sensor.SkeletonFrameReady +=
                KinectSensor_SkeletonFrameReady;
    // Obsługa strumienia obrazu z kamery
}
private void KinectSensor_SkeletonFrameReady(object sender
                                    , SkeletonFrameReadyEventArgs e)
{
    using (var frame = e.OpenSkeletonFrame())
    {
        if (frame == null || frame.SkeletonArrayLength == 0)
            return;
        frame.CopySkeletonDataTo(_skeletons);
    }
}
```

Funkcja *InitializeKinectSensor* umożliwia korzystanie ze strumienia szkieletowego za pomocą funkcji *Enable*. Następnie inicjalizuje zmienną *_skeletons*, która będzie trzymała dane szkieletowe. Jest to sześcioelementowa tablica, w której przechowywane będą informacje o wszystkich szkieletach wykrytych przez sensor. Na koniec następuje zapisanie się na zdarzenie *SkeletonFrameReady*. Zostaje ono wywołane, gdy nowa ramka danych dostępna jest do pobrania.

Metoda *KinectSensor_SkeletonFrameReady* odpowiada za obsługę nowej ramki danych szkieletowych. Ramka zostaje otwarta za pomocą wywołania *OpenSkeletonFrame* i przypisana do zmiennej lokalnej *frame*. Następnie wykonywane jest sprawdzenie, czy obiekt nie jest pusty, lub tablica danych szkieletowych nie jest długości 0. W ostatnim kroku uzyskane dane zostają skopiowane do zmiennej *_skeletons* za pomocą wywołania *CopySkeletonDataTo*.

3. Grafika 3D

Grafika trójwymiarowa jest jedną z dziedzin grafiki komputerowej, powszechnie stosowaną w dzisiejszych czasach. Modele wyświetlane są w bardzo wysokiej jakości i zbliżone są do obiektów ze świata rzeczywistego. Obecnie większość kart graficznych umożliwia jednoczesne wyświetlanie obiektów składających się z dużej liczby wielokątów oraz wykonywanie obliczeń związanych z generowaniem grafiki 3D takich jak przekształcenia geometryczne, czy teksturowanie.

Modele opisane mogą być w rozmaity sposób. Najpopularniejszym sposobem jest przedstawienie obiektu jako siatki wielokątów, najczęściej trójkątów. Każdy z punktów opisany jest trzema współrzędnymi. Podany sposób opisu wykorzystywany jest w aplikacji *Wirtualna Przymierzalnia 3D*, dlatego zostanie szerzej opisany w dalszych rozdziałach.

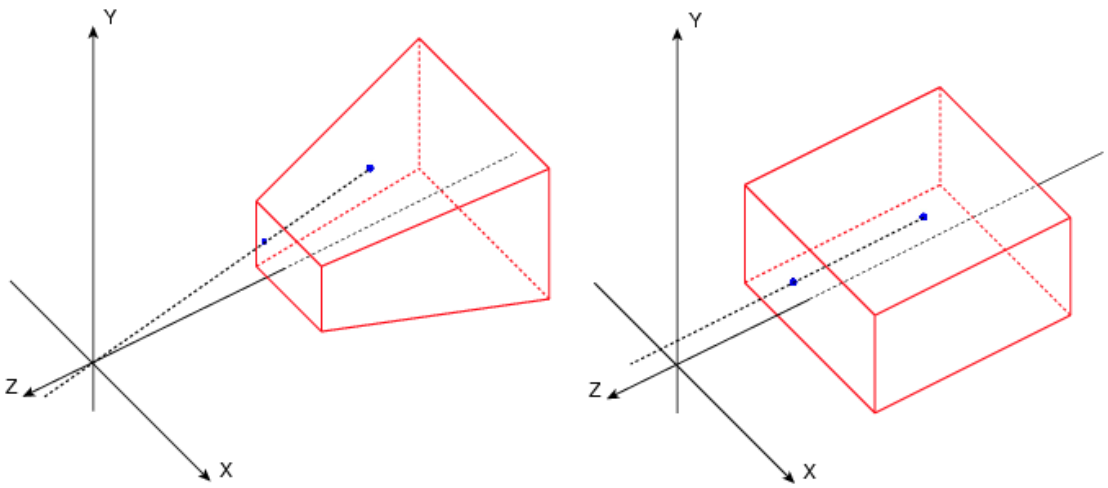
3.1. WPF 3D

WPF 3D to zestaw funkcjonalności pozwalający użytkownikowi na korzystanie z grafiki trójwymiarowej w aplikacji bez konieczności dostarczania zewnętrznych bibliotek. W odróżnieniu od innych narzędzi w .NET używających GDI, do wyświetlania grafiki 3D wykorzystywany jest DirectX. Funkcjonalność pozwala na rysowanie, transformację, czy też animację modeli trójwymiarowych. Możliwe jest to zarówno w plikach XAML, jak i cs. Generowanie, nawet skomplikowanych scen jest znacznie uproszczone, porównując do innych narzędzi, takich jak OpenGL. Dzięki temu uzyskać możemy grafikę wysokiej jakości renderowaną w czasie rzeczywistym. Pozwala to na wygodną integrację interfejsu użytkownika, grafiki 2D, grafiki 3D z innymi multimediami.

3.2. Helix 3D Toolkit

Helix 3D Toolkit jest zewnętrzną biblioteką ułatwiającą korzystanie i rozszerzającą możliwości WPF 3D. Dostępna jest na licencji MIT, dzięki czemu możliwe jest dostosowanie biblioteki do własnych potrzeb, a także partycypacja w dalszym jej rozbudowywaniu. Projekt jest na bieżąco rozwijany i aktualizowany. Źródła dostępne są pod adresem https://github.com/helix-toolkit/helix-toolkit. Do kodu biblioteki dołączone są liczne dema, pokazujące możliwości WPF 3D oraz wykorzystanie poszczególnych funkcji Helix 3D Toolkit.

Biblioteka umożliwia utworzenie kompletnej sceny 3D w zaledwie kilku linijkach. Standardowym obiektem zarządzającym sceną jest *HelixViewport3D*. Na potrzeby programu kontrolka została rozszerzona o właściwość udostępniającą macierz przekształcenia, dzięki której możliwa jest konwersja punktów z przestrzeni 2D do 3D. Wewnątrz kontrolki znajdują się trzy kluczowe obiekty: kamera, światło, kontener zawierający modele 3D. Aplikacja *Wirtualna Przymierzalnia 3D* wykorzystuje światło dzienne, by uzyskać efekt zbliżony do rzeczywistego. Domyślne ustawienia pozwalają na oświetlenie obiektów w pożądany sposób. Kontener *ItemsVisual3D* opisany został szczegółowo w rozdziale 3.3.



Rys. 6 Kamera perspektywiczna i ortograficzna

Helix 3D Toolkit udostępnia dwa rodzaje kamery: *PerspectiveCamera* oraz *OrthographicCamera*. Pierwsza z nich jest kamerą perspektywiczną. Sposób rzutowania obrazu przedstawia pierwsza grafika. W aplikacji *Wirtualna Przymierzalnia 3D* wykorzystana została kamera ortograficzna, przedstawiona na drugim obrazku. Wybrano ten sposób prezentacji, aby uniknąć niewłaściwych deformacji obrazu związanych z rzutowaniem punktów modelu. Właściwość *Target* ustawiona została na punkt (0, 0, 0).

Funkcjonalnością *Helix 3D Toolkit*, którą wykorzystano w programie jest także import modeli 3D. Biblioteka parsuje pliki zawierające definicję obiektu oraz wczytuje automatycznie dane o teksturach, kolorach i przezroczystości. Aspekt obsługi plików zewnętrznych opisany został w rozdziale 3.4.

3.3. ItemsVisual3D

Aplikacja *Wirtualna Przymierzalnia 3D* umożliwia wyświetlanie wielu modeli jednocześnie. Funkcjonalność nie jest jednak w pełni wspierana ani przez WPF 3D, ani przez *Helix 3D Toolkit*. Niezbędna była implementacja kontrolki *ItemsVisual3D* umożliwiającej wyświetlenie kolekcji modeli 3D. W tym celu wykorzystano i przystosowano kod demo *DataTemplate* napisanego dla biblioteki *Helix 3D Toolkit*.

Kontrolka dziedziczy po klasie *ModelVisual3D*. Dodatkowo rozszerzona jest o dwie właściwości: *ItemTemplateProperty* oraz *ItemsSourceProperty*. Pierwsza *DependencyProperty* jest typu *DataTemplate3D* i określa wzorzec, według którego tworzone będą obiekty. Druga z nich wskazuje na źródło kolekcji zawierającej wyświetlane modele implementującej interfejs *ICollection*. Poza właściwościami przypisującymi i pobierającymi wartości obu *DependencyProperty* kontrolka zawiera dodatkowo metodę obsługującą zmianę kolekcji. Odpowiada ona za dodawanie nowych obiektów do sceny oraz usuwanie tych, które nie powinny być już wyświetlane.

Rys. 7 Okno aplikacji z nałożonym modelem 3D

3.4. Format danych

Modele 3D opisane mogą być w rozmaity sposób. Popularnym formatem danych są pliki z rozszerzeniem *.obj. Zaletą tego typu dokumentów jest ich czytelność. Nie są one poddane kompresji, a dane zapisane są w sposób łatwy do interpretacji. Do plików *.obj dołączone są pliki *.mtl zawierające informacje o teksturach. Dwa formaty skojarzone są ze sobą przez ścieżkę. Plik opisujący model zawiera w nagłówku ścieżkę do dokumentu, w którym znajdują się informacje o teksturach.

3.4.1. Pliki *.obj

Pliki *.obj zawierają dane o wierzchołkach, pozycjach UV tekstury, normalnych oraz powierzchniach modelu. Wierzchołki modelu zapisane są w kolejności przeciwnej do ruchu wskazówek zegara. Pliki nie przechowują informacji o jednostkach.

Pliki *.obj wykorzystywane w aplikacji definiują modele 3D ubrań dostępnych dla użytkownika. Pobrane zostały z ogólnodostępnych bibliotek. Gotowe modele dodawane są jako źródła do programu.

Aby wczytać model z pliku *.obj wystarczy utworzyć instancję klasy *ModelImporter*, a następnie wywołać na nim metodę *Load*. Należy pamiętać, że plik *.obj może zawierać definicję więcej niż jednego obiektu. W aplikacji *Wirtualna Przymierzalnia 3D* istnieje tylko jedna instancja obiektu *ModelImporter*, która znajduje się w klasie *ClothingManager*. Jest ona wykorzystywana do tworzenia wszystkich modeli dostępnych w programie. Poniższy kod przedstawia przykładowe użycie w aplikacji:

```
ModelImporter _importer = new ModelImporter();
Model3DGroup Model = Importer.Load("dress_blue.obj");
```

Komentarz w pliku *.obj poprzedza znak #. Zazwyczaj na początku dokumentu znajdują się metadane opisujące model, a także aplikację, w której został wykonany. Poniżej przedstawiono fragmenty przykładowego pliku *.obj. Dodane komentarze opisujące fragmenty pliku zaznaczono pogrubioną czcionką.

```
# file generated by UVMapper
# NumVerts/NumTVerts/NumVNormals/NumFacets      3280/3321/6480/3200
# NumGroups/NumMaterials/NumRegions     1/0/0
# x/y/color/ppu     512/512/0/50.00000000

# Wskazuje na nazwę pliku, który definiuje tekstury modelu
mtllib white_dress.mtl

# Lista wierzchołków w formacie (x,y,z[,w]). w jest opcjonalne, z wartością domyślną 1.0.
v -0.03397000    0.37885001 -0.05553000
...
# Współrzędne tekstury w formacie (u, v [,w]). Wartości są z przedziału 0 - 1, w jest opcjonalne, domyślnie 0.
vt    0.75299346    0.76020932
...
# Normalne w formacie (x,y,z). Długość wektora nie może przekroczyć 1.
```

vn 0.28384000 0.45300999 -0.84511000

...

 # Powierzchnie - zawierają numery wierzchołków i współrzędne tekstur, z których są utworzone, a także normalnych (zapis w podanej kolejności).

f 962/280/4162 3136/334/6336 978/279/4178 72/282/3272

...

Pliki *.obj mogą dodatkowo opisywać punkty, linie, krzywe, co pominięto ze względu na brak użycia w aplikacji.

3.4.2. Pliki *.mtl

Pliki *mtl są plikami, z których importuje się tekstury modeli zapisanych w formacie *.obj. Plik obj może zawierać referencję do jednego lub wielu plików mtl. Pliki mtl zawierają między innymi informacje o kolorach materiałów oraz o odbiciu światła przez daną powierzchnię. Format pliku jest niezależny od oprogramowania, sprawdza się w większości środowisk. Kolory tekstur definiowane są w formacie RGB. Ich wartości mieszczą się w przedziale [0, 1].

Pliki *.mtl wykorzystywane są w aplikacji jako informacja o materiałach dla importowanych modeli. Importer modeli z biblioteki *Helix 3D Toolkit* przetwarzając pliki *.obj, automatycznie dodaje do modeli materiały z referensowanych plików *.mtl. Dzięki temu ręczny import materiałów jest zbędny. Wystarczy jedynie plik *.obj umieścić w tym samym katalogu, co odpowiadające mu pliki *.mtl.

Poniżej opisano zawartość przykładowego pliku *.mtl wykorzystanego w aplikacji *Wirtualna Przymierzalnia 3D*. Podobnie jak przy formacie *.obj, znak # rozpoczyna komentarz w pliku. Informacje opisujące plik napisano pogrubioną czcionką.

 # Bieżący plik został wygenerowany z modelu z pliku white_dress.blend

Blender MTL File: 'white_dress.blend'

 # Bieżący plik zawiera definicję jednego materiału

Material Count: 1

 # Definicja materiału o nazwie Material6527

newmtl Material6527

 # Definicja skupienia odbicia światła

Ns 96.078431

 # Definicja koloru ambient

Ka 0.000000 0.000000 0.000000

 # Definicja koloru difuse

Kd 0.640000 0.640000 0.640000

 # Definicja koloru specular

Ks 0.500000 0.500000 0.500000

 # Współczynnik załamania

Ni 1.000000

#Definicja przezroczystości (dissolve)

d 1.000000

 # Parametr ustawienia światła. Istnieje 10 różnych wartości.

 # Tu użyto typu drugiego: Highlight ON

illum 2

 # Plik tekstury

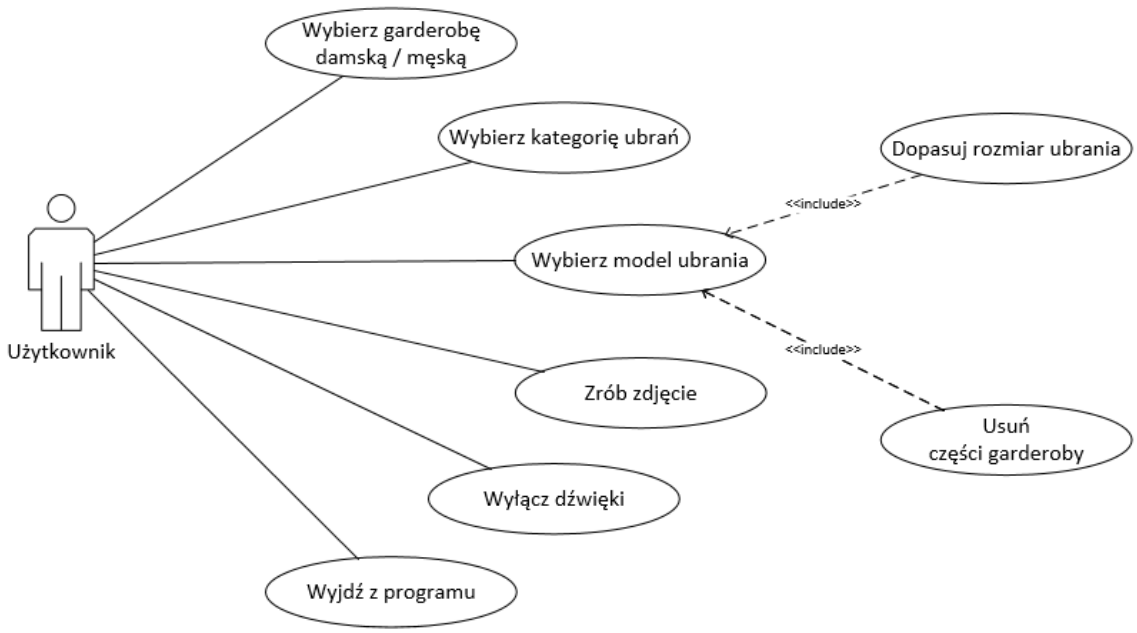
map_Kd

4. Wirtualna Przymierzalnia 3D

Aplikacja *Wirtualna Przymierzalnia 3D* implementuje system rozszerzonej rzeczywistości. Pozwala na przymierzanie różnych części garderoby. Modele ubrań wykonane są w technologii 3D. Program umożliwia użytkownikowi wybór ubrań z kategorii damskiej i męskiej, dostosowanie rozmiaru ubrania, a także zrobienie zdjęć wybranego stroju.

Aplikacja wykorzystuje czujnik ruchu Kinect. Urządzenie rozpoznaje użytkownika i umożliwia sterowanie aplikacją za pomocą gestów. Kliknięciu przycisków towarzyszą odpowiednie efekty dźwiękowe.

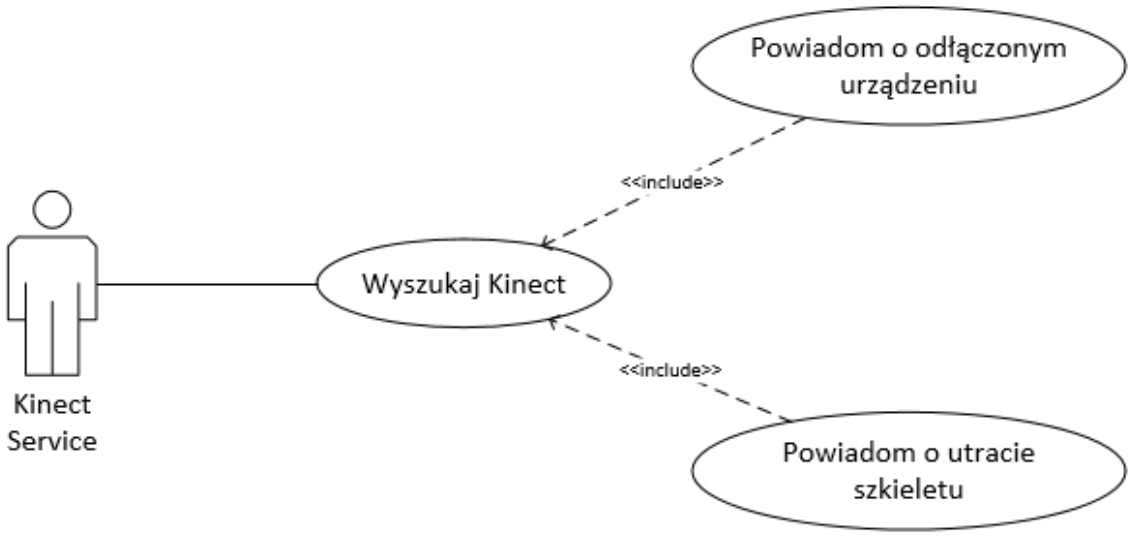
4.1. Przypadki użycia



Rys. 8 Przypadki użycia dla użytkownika aplikacji

Aplikacja daje użytkownikowi szereg możliwości. Oddzielnie dla garderoby męskiej i damskiej dostępne są kategorie ubrań, takie jak nakrycia głowy, nakrycia wierzchnie, akcesoria. Z każdej kategorii użytkownik ma możliwość wybrania po jednym elemencie. Wybrane części garderoby można dopasować do swoich wymiarów,

odpowiednio poszerzając lub zwężając ubranie. Jeśli wybrany zestaw nie będzie odpowiadać użytkownikowi, ma on możliwość usunięcia wszystkich wybranych ubrań. Jeśli użytkownik uzna wybrany zestaw za godny zapamiętania, ma on możliwość zrobienia sobie zdjęcia w wybranych częściach garderoby. Jeśli wbudowane dźwięki będą uciążliwe dla użytkownika, aplikacja umożliwia mu wyłączenie notyfikacji głosowych. Ostatnią, oczywistą funkcją jest opuszczenie programu.

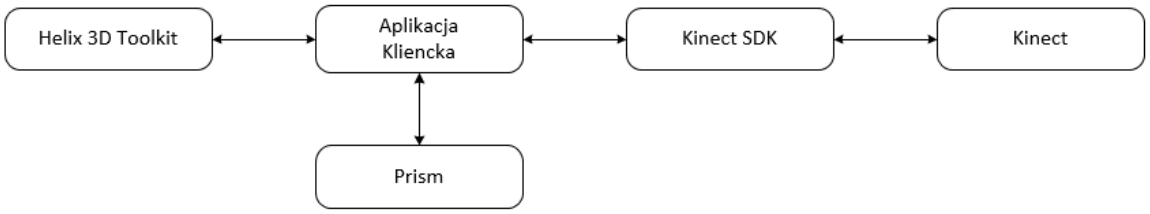


Rys. 9 Przypadki użycia dla *KinectService*

Głównym zadaniem serwisu jest kontrola połączenia z sensorem oraz notyfikacja użytkownika o wszelkich błędach. Użytkownik zostanie powiadomiony każdorazowo o utracie połączenia lub błędzie w komunikacji z Kinectem. Zadaniem serwisu będzie nieustanne odpytywanie o poprawność łącza. Kiedy problemy z sensorem zostaną rozwiązane, możliwe będzie dalsze korzystanie z aplikacji.

Kiedy szkielet użytkownika zostanie utracony, *KinectService* wyśle odpowiednią notyfikację. Podobnie jak w przypadku odłączenia sensora, dostępność danych będzie sprawdzana, a notyfikacje przestaną być wysyłane w momencie otrzymania danych o szkielecie. Po przywróceniu działania aplikacji, wszystkie dane zostaną odtworzone do wersji sprzed utraty danych szkieletowych.

4.2. Architektura komponentów



Rys. 10 Architektura komponentów

Program składa się z kilku podstawowych komponentów opisanych szczegółowo w dalszych rozdziałach. Aplikacja kliencka komunikuje się z sensorem Kinect używając *Kinect SDK*. Dodatkowo wykorzystywana jest biblioteka *Prism* oraz opisany w rozdziale 3.2 *Helix 3D Toolkit*.

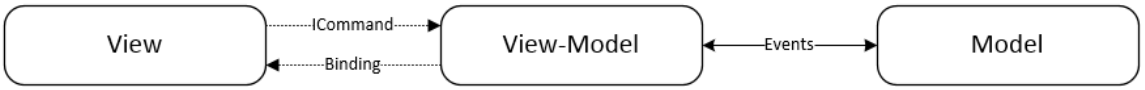
Aplikacja kliencka dostarcza użytkownikowi interfejs graficzny. Użytkownik steruje aplikacją za pomocą gestów rozpoznawanych i obsługiwanych przez wewnętrzne komponenty.

Kinect SDK dostarcza aplikacji klienckiej API, dzięki którym możliwe jest pobieranie danych z Kinecta. Mechanizmy nie są jednak ograniczone wyłącznie do odczytu, dzięki API można także wysyłać dane do urządzenia, co umożliwia przykładowo kalibrację obrazu kamery. Kinect udostępnia przez API dane odczytane ze sprzętu. Możliwe jest uzyskanie danych dotyczących obrazu, szkieletu, głębokości, dźwięku oraz innych.

Biblioteka *Prism* dostarcza mechanizmy pośredniczenia między widokiem aplikacji a jej warstwą logiczną. Stanowi tym samym część warstwy pośredniczącej między tymi dwoma komponentami, co opisane zostało w sekcji 4.3.2.

Biblioteka *Helix 3D Toolkit*, jak opisano w rozdziale 3.2 odpowiedzialna jest za przetwarzanie modeli 3D oraz ułatwioną implementację i kontrolę sceny 3D.

4.3. Wzorzec projektowy MVVM



Rys. 11 Wzorzec MVVM

Aplikacja zaprojektowana została w oparciu o wzorzec projektowy *Model-View-ViewModel*. Głównym celem wzorca jest zapewnienie separacji widoku od modelu przez wprowadzenie warstwy pośredniej, jaką jest model widoku. MVVM zapewnia także łatwość rozbudowy aplikacji i wprowadzania w niej modyfikacji.

View, czyli widok obejmuje wizualną część implementacji. Składają się na nią wszystkie okna oraz kontrolki, a także tak zwany *code-behind* obsługujący podane obiekty. Zastosowanie wzorca MVVM nie oznacza rezygnacji z *code-behind*, wręcz przeciwnie. Wszelkie zmiany widoku, których nie można uzależnić od modelu, umieszczone muszą być w kodzie kontrolki. W większości właściwości widoku zależą od danych zawartych w modelu, do których nie ma bezpośredniego dostępu. Dostarczane są one przez *ViewModel*.

Warstwę pośredniczącą między widokiem a modelem stanowi *ViewModel*. Udostępnia on widokowi dane z modelu poprzez mechanizm *Bindingu*, a także uaktualnia model, jeśli właściwości te zmienione zostaną w widoku. Dzięki temu możliwe jest sterowanie widokiem bez potrzeby bezpośredniego dostępu do jego instancji. Zmiana widoku może wywołać zmianę zachowania aplikacji dzięki interfejsowi *ICommand* zawartym w bibliotece *Prism*. Umożliwia on wywoływanie metod na komponencie *ViewModel*, kiedy widok odbierze przypisane metodom zdarzenie.

Model stanowią pozostałe klasy aplikacji. Zawierają one informacje oraz obiekty używane w programie. Widok modelu oraz model komunikują się głównie przez notyfikację zdarzeniami.

4.3.1. Zastosowanie wzorca w aplikacji

Głównym komponentem modułu widoku jest okno główne aplikacji. Zawiera ono wszystkie kontrolki używane przez użytkownika, obraz z kamery Kinecta, a także wszystkie elementy garderoby, umieszczone w kontrolce sceny 3D. Umożliwia ono także wyświetlenie pomocniczych elementów przydatnych podczas implementacji, takich jak szkielet użytkownika, czy odpowiednie notyfikacje o przechwyconych zdarzeniach. Dodatkowo do modułu widoku zaliczają się także klasy implementujące kontrolki dedykowane aplikacji, a także zestaw klas pomocniczych rozszerzających możliwości obiektów wizualnych.

ViewModel zawiera klasę główną programu, jaką jest *KinectViewModel*. Pośredniczy ona między oknem głównym a klasami zawierającymi dane aplikacji. *ViewModel* zawiera także kilka pomocniczych klas, które pośredniczą między komponentami. Klasy komponentu *ViewModel*, jak zakłada wzorzec, odpowiedzialne są głównie za zarządzanie obiektami modelu oraz interakcją z widokiem aplikacji.

Do modelu zaliczyć można głównie klasy obiektów definiujących modele ubrań. Zawierają one dane przetwarzane przez *ViewModel*.

4.3.2. Prism

Prism jest biblioteką wspomagającą budowanie aplikacji opartych na wzorcach projektowych zapewniających separację komponentów, takich jak MVVM. Umożliwia łatwą rozbudowę programu. *Wirtualna Przymierzalnia 3D* wykorzystuje bibliotekę w wersji 4.1.

Dzięki bibliotece *Prism* możliwa jest implementacja interfejsu *ICommand*. Pozwala on na zastosowanie mechanizmu *Binding* nie tylko do właściwości, lecz także do metod, czy zdarzeń. Wszystkie obiekty przycisków należące do komponentu *ViewModel* dziedziczą po *ButtonViewModelBase*. Klasa ta zawiera deklarację obsługi wywołania metody przez mechanizm *Binding*. Właściwość *ClickCommand* udostępnia komendę *_clickCommand*, która w momencie wywołania wykona metodę *ClickExecuted*. Jest ona abstrakcyjna ze względu na potrzebę implementacji zróżnicowanej obsługi przez poszczególne typy przycisków.

```
private ICommand _clickCommand;
public ICommand ClickCommand
{
    get { return _clickCommand?? (_clickCommand
                        = new DelegateCommand(ClickExecuted)); }
}
public abstract void ClickExecuted();
```

Aby umożliwić wykonanie metody *ClickExecuted* po przechwyceniu zdarzenia należy dodać odpowiedni znacznik w pliku *.xaml. W aplikacji *Wirtualna Przymierzalnia 3D* obsługa implementowana jest głównie dla zdarzenia *HandCursorClick*. Wewnątrz kontrolki *KinectButton* należy umieścić znacznik dostępny w bibliotece *System.Windows.Interactivity*. Przykład użycia pokazany jest poniżej. Komenda *ClickCommand* wykonana zostanie, gdy *KinectButton* otrzyma zdarzenie *HandCursorClick*.

```
<buttons:KinectButton>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="HandCursorClick">
            <i:InvokeCommandAction
                                Command="{Binding ClickCommand}"/>
        </i:EventTrigger>
    </i:Interaction.Triggers>
</buttons:KinectButton>
```

4.4. Rozpoznawanie gestów

Wirtualna Przymierzalnia 3D nie korzysta z zewnętrznych bibliotek rozszerzających możliwości *Kinect SDK*. Z tego względu niezbędna była implementacja mechanizmów umożliwiających pełne korzystanie z funkcji sensora. Jedną z zaimplementowanych funkcjonalności jest rozpoznawanie gestów. Zostało ono wykorzystane przy obsłudze przycisków oraz sterowaniu menu aplikacji.

4.4.1. HandTracking

Standardowo *Kinect SDK* nie udostępnia obiektu będącego wskaźnikiem w aplikacji. Niezbędne okazało się utworzenie nowej klasy imitującej kursor. Obiekt *HandTracking* odpowiedzialny jest za trzymanie oraz uaktualnianie pozycji prawej i lewej dłoni użytkownika. W aplikacji *Wirtualna Przymierzalnia 3D* kursor podąża za nimi zgodnie z logiką opisaną w dalszej części.

Za kursor służy w programie obiekt typu *Canvas*. Obiekty dziedziczące po klasie *UIElement* udostępniają metodę *InputHitTest*. Jako parametr podawany jest punkt w przestrzeni dwuwymiarowej. Metoda zwraca kontrolkę znajdującą się bezpośrednio pod wskazanym punktem lub *null,* gdy obiekt taki nie istnieje. Jeśli koordynaty wskazują na więcej niż jedną kontrolkę, zawsze zwracany jest obiekt znajdujący się na najwyższej warstwie.

Kursor pozostaje niewidoczny, gdy obie dłonie użytkownika znajdują się poza panelami przycisków. Staje się widoczny, gdy użytkownik umieści jedną z dłoni na dowolnym menu. Lewa dłoń ma w aplikacji priorytet nad prawą. Oznacza to, że jeśli obie zostaną umieszczone nad panelami, kursor zostanie pokazany w miejscu wskazywanym przez lewą dłoń.

4.4.2. KinectButton

Niezbędna okazała się implementacja przycisków obsługujących gesty dedykowane aplikacji. Standardowa klasa *Button* nie udostępnia wymaganego mechanizmu. Na potrzeby aplikacji została utworzona kontrolka *KinectButton* dziedzicząca po klasie przycisku. Dodatkowo zaimplementowano klasy *KinectRepeatableButton* oraz *KinectScreenshotButton* rozszerzające możliwości *KinectButton*.

Głównym zadaniem klasy jest obsługa gestu przyciśnięcia. W tym celu dodany został licznik mierzący czas od momentu znalezienia się kursora nad przyciskiem. Jeśli przed upływem zadanego czasu dłoń użytkownika nie wyjdzie poza obszar kontrolki, wywołane zostanie zdarzenie przyciśnięcia przycisku. Jest ono obsługiwane zależnie od przeznaczenia przycisku w klasie odpowiadającej jego modelowi widoku. Zdarzenie wywoływane jest jednorazowo. Jeśli użytkownik ponownie pozostawi dłoń nad

obszarem kontrolki na czas odpowiadający przyciśnięciu, nie zostanie ono wywołane ponownie. Zapobiega to niepożądanemu wielokrotnemu wywołaniu tej samej akcji.

Klasa *KinectScreenshotButton* posiada dodatkowy licznik, uruchamiany w momencie przyciśnięcia przycisku. Odlicza on czas do zrobienia zrzutu ekranu aplikacji. Dzięki temu użytkownik może przybrać odpowiednią pozę do zdjęcia. Po zakończeniu odliczania następuje zapisanie obrazu z kamery wraz z modelami ubrań, a także dodany zostaje znak wodny aplikacji.

KinectRepeatableButton obsługuje między innymi przyciski zmiany rozmiaru modelu, przemieszczenie ubrania, a także główny przycisk górnego menu. Zawiera on jedynie niewielką zmianę w porównaniu do klasy bazowej. *KinectButton* uniemożliwia wielokrotne wywołanie zdarzenia *Click* przez dłuższe przytrzymanie dłoni nad przyciskiem, natomiast implementacja *KinectRepeatableButton* uwzględnia tą funkcjonalność. Dzięki temu, by znacznie zmienić rozmiar lub położenie modelu, wystarczy, by użytkownik pozostawił kursor nad przyciskiem na dłuższy okres czasu.

Klasy komponentu *ViewModel* dziedziczące po klasie *ButtonViewModelBase* implementują obsługę zdarzenia *Click* zależnie od typu przycisku. Obsługa odbywa się w przeciążonej metodzie *ClickExecuted*.

4.4.3. ScrollableCanvas

Aplikacja zapewnia użytkownikowi możliwość przymierzania wielu modeli ubrań. Niestety długość paneli bocznych jest ograniczona i nie jest możliwe wyświetlenie wszystkich przycisków z częściami garderoby. Niezbędne stało się zaimplementowanie paneli, które umożliwiałyby przewijanie swojej zawartości, tak aby użytkownik miał łatwy dostęp do każdego z przycisków. Standardowe kontrolki zapewnione przez WPF do przechowywania kolekcji obiektów nie posiadają takiej funkcjonalności. Wyjątek stanowi kontrolka *ScrollViewer*, która zapewnia przewijanie zawartości, jednak nie jest przeznaczona do obsługi za pomocą gestów.

W tym celu stworzono klasę *ScrollableCanvas* dziedzicząca po standardowej klasie *ItemsControl*. Wykrywa ona położenie kursora nad bocznymi panelami i zgodnie z jego lokalizacją przewija swoją zawartość odpowiednio w górę lub w dół.

Panele boczne podzielone zostały na trzy obszary. Obszar górny zaczyna się wraz z górną krawędzią panelu i kończy się w 1/5 wysokości panelu, natomiast obszar dolny zaczyna się w połowie wysokości panelu i kończy razem z dolną krawędzią panelu. Gdy kursor znajdzie się w obszarze górnym lub dolnym, następuje wywołanie metody *FindChild*. Odpowiedzialna jest ona za wyszukanie wszystkich obiektów typu *Button* w drzewie wizualnym kontrolki kontenera. Każdy z nich zostaje przesunięty zgodnie z gestem użytkownika.

4.5. Śledzenie szkieletu

Aby uzyskać realistyczny efekt przymierzenia ubrania w aplikacji *Wirtualna Przymierzalnia 3D* niezbędne jest nie tylko wyświetlenie modelu w trójwymiarze. Istotna jest implementacja podążania obiektu za osobą, a także symulacja obrotu modelu.

Obiekty typu *Model3DGroup* posiadają właściwość *Transform* zawierającą kolekcję transformacji. Przekształcenia stosowane są w porządku ich umieszczenia w kontenerze. Zamiana kolejności transformacji skutkować może uzyskaniem przekształcenia zupełnie różnego od oczekiwanego. Z tego względu modyfikacje stosowane są według ustalonej hierarchii. Jako pierwsze wykonywane jest skalowanie, następnie rotacja modelu i ostatecznie translacja o wektor.

Aby zastosować kolejne transformacje do obiektu *Model* należy utworzyć instancję typu *Transform3DGroup*. Zawiera ona właściwość *Children* będącą kolekcją przekształceń. Następnie obiekty będące transformacjami należy dodać w kolejności ustalonej w poprzednim paragrafie. Ostatnim etapem jest przypisanie do właściwości *Transform* utworzonej uprzednio kolekcji. Przykład przedstawiono poniżej:

```
var transform = new Transform3DGroup();
transform.Children.Add(scaleTransform);
transform.Children.Add(rotateTransform);
transform.Children.Add(translateTransform);
Model.Transform = transform;
```

4.5.1. Skalowanie modelu

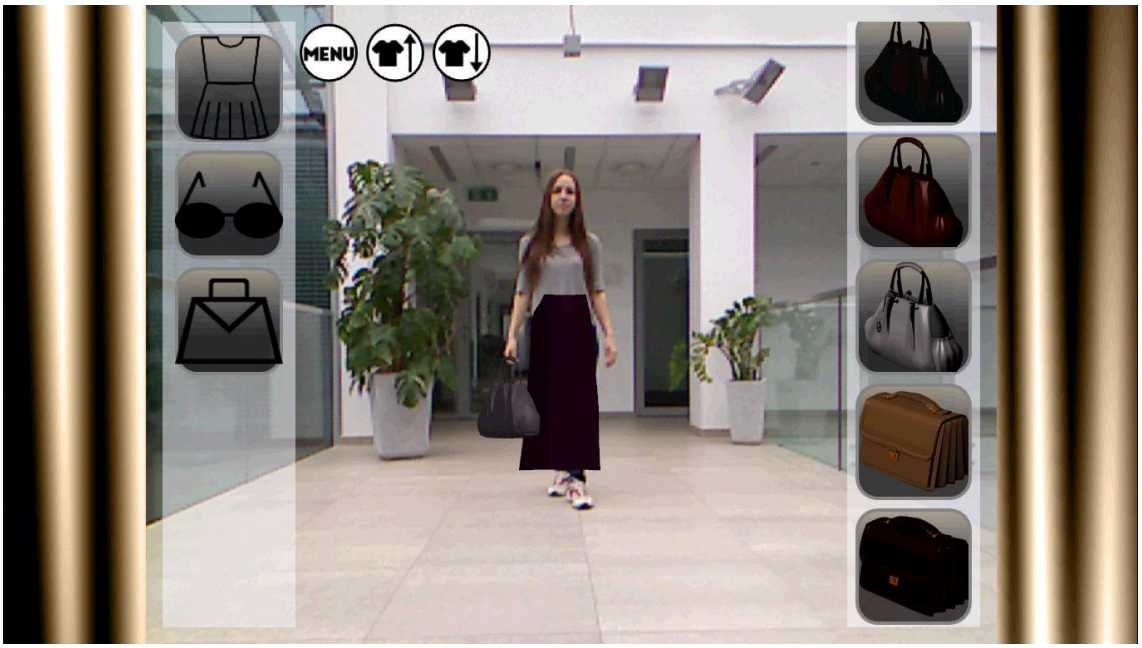
Transformacja związana ze zmianą rozmiaru modelu trzymana jest przez właściwość *ScaleTransformation*. Aktualizowana jest w aplikacji w dwóch przypadkach. Pierwszym z nich jest dostosowanie rozmiaru modelu do wymiarów użytkownika podczas przemieszczania się, drugim manualna zmiana rozmiaru przez przyciskiem w górnym menu aplikacji. W drugim przypadku zwiększane lub zmniejszane są wartości zmiennych *_widthScale* oraz *_heightScale*. Dzięki temu użytkownik ma możliwość sprawdzenia jak dany model prezentowałby się w dłuższej lub krótszej wersji.

Dopasowanie modelu do rozmiaru użytkownika następuje przez wywołanie metody *FitModelToBody*. Metoda ta dopasowuje model 3D do rozmiaru użytkownika. Każdy rodzaj ubrania ma określone dwa punkty szkieletowe, które identyfikują właściwy rozmiar. W metodzie pobierane są współrzędne śledzonych punktów, poddane konwersji do przestrzeni, definiowanej przez kontrolkę *Canvas*. Następnie pobierany jest rozmiar domyślny obiektu 3D oraz jego położenie. Zmienna lokalna *ratio* określa stosunek odległości śledzonych punktów szkieletowych do rozmiaru modelu. Następnie wartość ta mnożona jest przez wartość właściwości *Tolerance*. Jest to niezbędne ze względu na fakt, iż punkty szkieletowe znajdują się wewnątrz modelu użytkownika. Obliczone wartości przypisywane są zmiennym, a następnie wywoływana jest metoda *SetScaleTransform* obliczająca macierz skalowania.

```
private void FitModelToBody(Joint joint1, Joint joint2
                , KinectSensor sensor, double width, double height)
{
    var joint1Position = KinectService.GetJointPoint(joint1
                                                    , sensor, width, height);
    var joint2Position = KinectService.GetJointPoint(joint2
                                                    , sensor, width, height);
    var location = _basicBounds.Location;
    Point leftBound = Point3DtoPoint2D(location);
```

```
        Point rightBound = Point3DtoPoint2D(new Point3D(
                                            location.X + _basicBounds.SizeX
                                            , location.Y + _basicBounds.SizeY
                                            , location.Z + _basicBounds.SizeZ));


        double ratio = (Math.Abs(joint1Position.Y - joint2Position.Y)
            / Math.Abs(leftBound.Y - rightBound.Y));
        _widthModelScale = _heightModelScale = ratio * Tolerance;
        SetScaleTransformation();
    }
    private void SetScaleTransformation()
    {
        Transform3DGroup transform = new Transform3DGroup();
        transform.Children.Add(new ScaleTransform3D(
                                    _widthScale * _widthModelScale , 1
                                    , _widthScale * _widthModelScale));
        transform.Children.Add(new ScaleTransform3D(
                                    1, _heightScale * _heightModelScale , 1));
        ScaleTransformation = transform;
    }
```



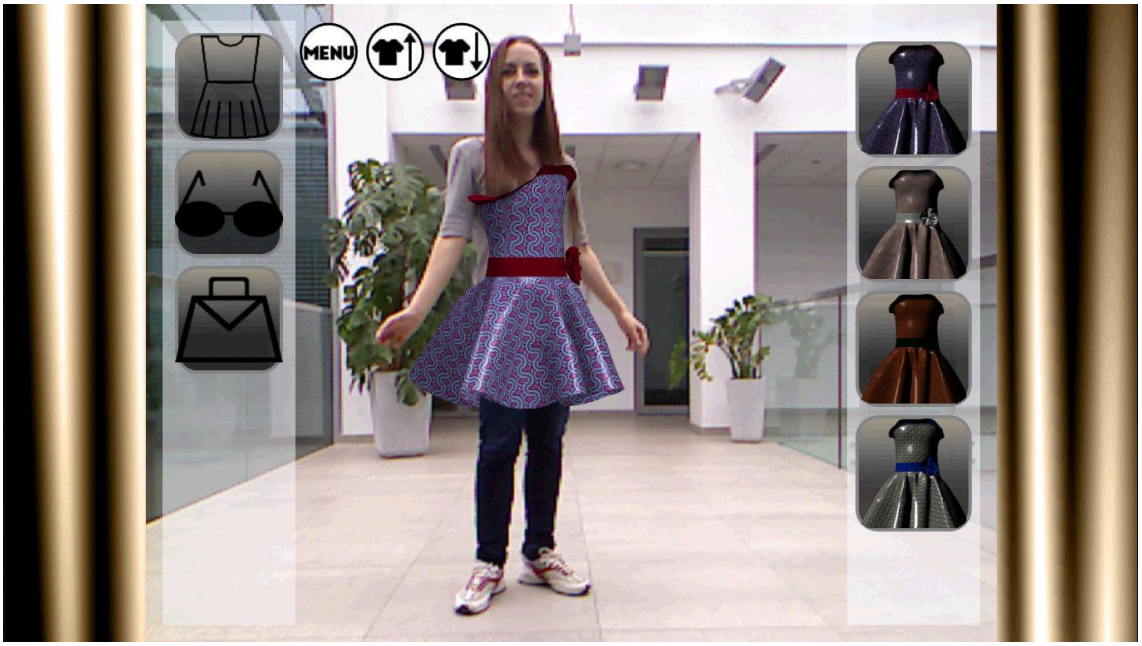
Rys. 12 Okno aplikacji z przeskalowanymi modelami

4.5.2. Rotacja modelu

Aby wyznaczyć kąt, o jaki należy obrócić model, wystarczą dwa punkty szkieletowe położone na zbliżonej wysokości. W zależności od rodzaju modelu, obliczenia stosowane są dla różnych par. W większości są to: *ShoulderLeft* i *houlderRight* lub *HipLeft* oraz *HipRight*. Poniższa metoda opisuje sposób wyznaczenia kąta między dwoma punktami szkieletowymi. Współrzędne punktów mapowane są na przestrzeń aplikacji, a następnie szukana wartość wyznaczana jest z zależności trygonometrycznych. Ze względu na różnicę układów współrzędnych niezbędny jest minus przed wartością wyrażenia. Zwracana wartość wyrażona jest w stopniach.

```
protected double TrackJointsRotation(KinectSensor sensor
                                     , Joint joint1, Joint joint2)
{
    var joint1Position = sensor.CoordinateMapper
                .MapSkeletonPointToDepthPoint(joint1.Position
                , sensor.DepthStream.Format);
    var joint2Position = sensor.CoordinateMapper
                .MapSkeletonPointToDepthPoint(joint2.Position
                , sensor.DepthStream.Format);

    return -(Math.Atan(((double)joint1Position.Depth
                - joint2Position.Depth)
                / ((double)joint2Position.X - joint1Position.X))
                * 180.0 / Math.PI);
}
```



Rys. 13 Okno aplikacji z obróconym modelem

4.5.3. Translacja modelu

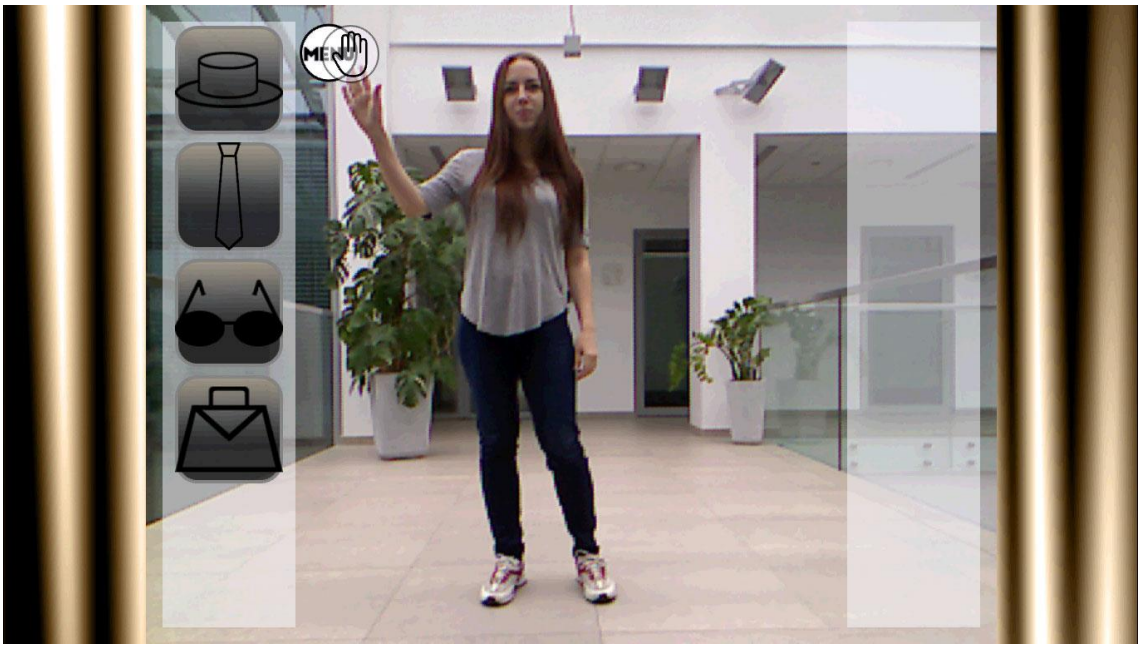
Translacja obiektu odbywa się, gdy użytkownik zmieni położenie punktu szkieletowego, za którym podąża dany model. Z powodu wykorzystania w aplikacji kamery ortograficznej, na wyświetlenie modelu wpływ mają jedynie współrzędne X oraz Y. Przekształcenie obliczane jest w metodzie *TrackSkeletonParts*. Za pomocą metody *GetJointPoint* pobierane są współrzędne punktu szkieletowego. Następnie za pomocą metody *Point2DToPoint3D* współrzędne konwertowane są do przestrzeni kontrolki sceny 3D.

4.6. Wygląd i działanie aplikacji

Rys. 14 Ekran powitalny

Dzięki intuicyjnemu interfejsowi, nauka obsługi aplikacji przebiega bardzo szybko. Przyciski zostały umiejscowione tak, aby użytkownik miał do nich łatwy dostęp. Aplikacja składa się z ekranu powitalnego i okna głównego. Pierwszy z nich pojawia się niezwłocznie po uruchomieniu aplikacji. Niedługo potem użytkownik może zacząć korzystać z aplikacji z użyciem okna głównego.

4.6.1. Ekran główny



Rys. 15 Główne okno aplikacji

Główny ekran aplikacji składa się z obszaru, na którym widoczny jest obraz pobierany z kamery Kinect, górnego menu z podstawowymi funkcjami oraz lewego i prawego panelu zawierającego ubrania do przymierzenia.

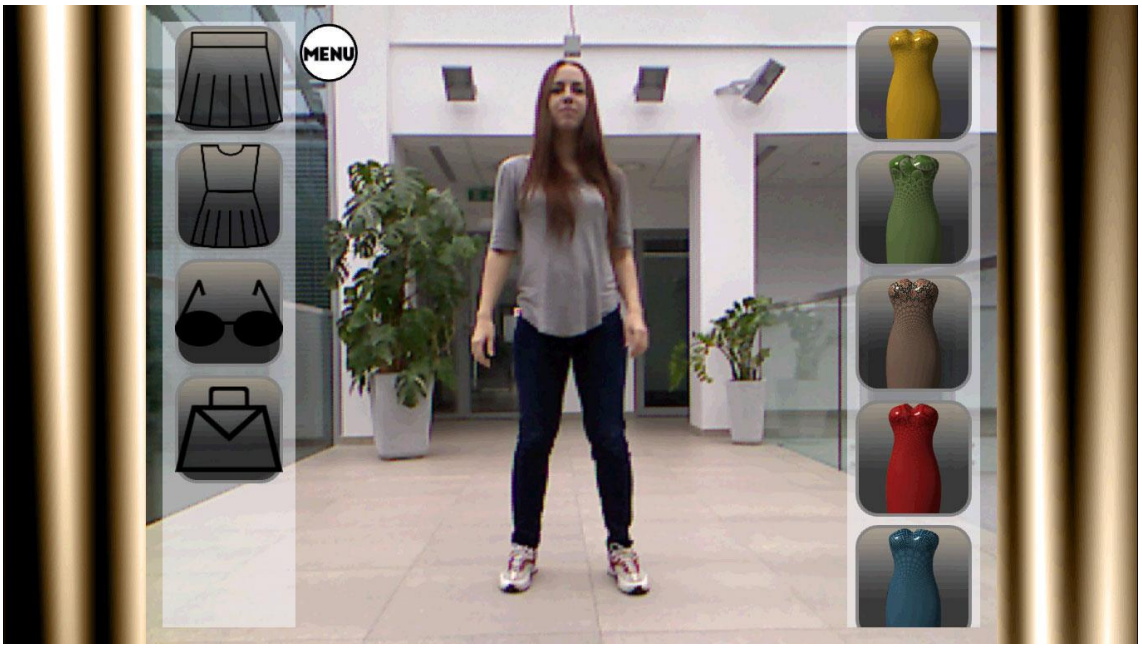
Na obraz z kamery nakładane są modele wybranych części garderoby, które przylegają do wykrytego szkieletu użytkownika nawet w trakcie jego ruchów i obrotów.

4.6.2. Górne menu

Rys. 16 Okno aplikacji z rozwiniętym górnym menu

Górne menu pojawia się na ekranie po najechaniu i kliknięciu przez użytkownika przycisku z napisem *MENU*. Pojawiają się wówczas ikony z dodatkowymi funkcjonalnościami: wykonanie zdjęcia, zmiana kategorii, zmiana rozmiaru ubrania, zmiana pozycji ubrania, czyszczenie modeli ubrań, kontrola dźwięków w aplikacji oraz wyjście z programu. Po ponownym naciśnięciu przycisku pozostałe ikony znikają z ekranu. Gdy znajdujemy się w podmenu, naciśnięcie przycisku *MENU* skutkuje powrotem do głównych funkcjonalności.

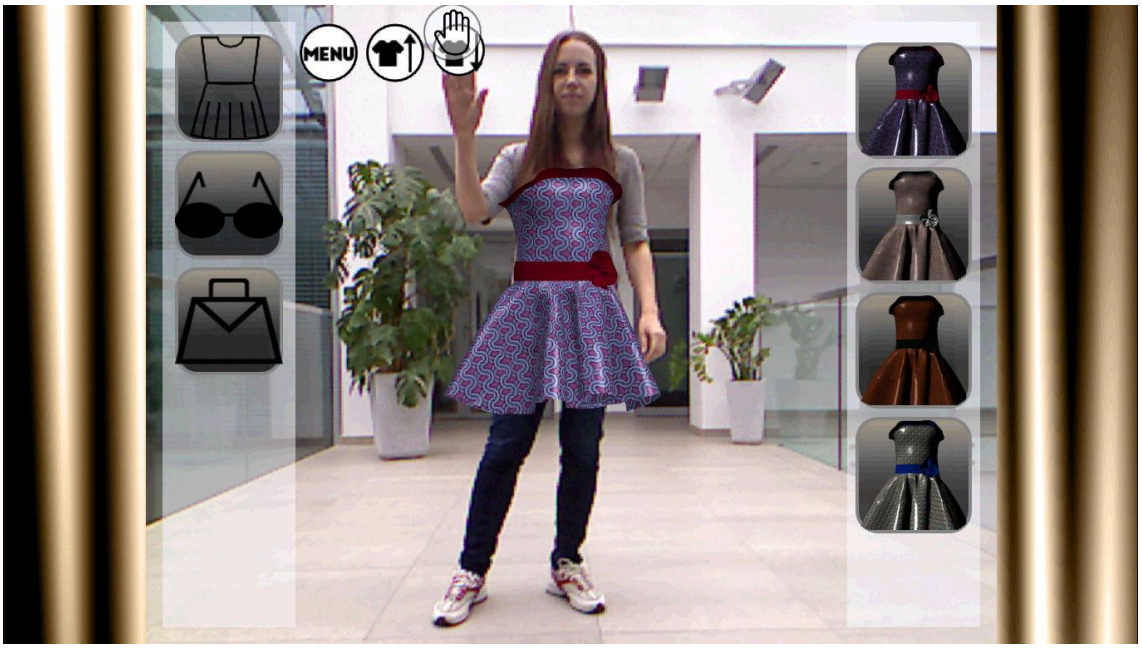
4.6.3. Boczne panele



Rys. 17 Okno aplikacji z uzupełnionymi bocznymi panelami

W panelu po lewej stronie widoczne są kategorie z pogrupowanymi częściami garderoby. Po kliknięciu przycisku z daną kategorią, w prawym panelu wyświetlają się modele dostępne do przymierzenia. Po wybraniu odpowiadającego nam ubrania, jego model nakładany jest na użytkownika i przylega do ciała nawet w trakcie ruchu. Gdy zostanie kliknięty przycisk w tej samej kategorii z modelem innej części garderoby, ubranie nałożone na osobę zostaje zamienione. Naciśnięcie na przycisk w tej samej kategorii oraz z tą samą częścią garderoby nie spowoduje żadnej akcji. Po wybraniu innej kategorii, nowo wybrane ubranie jest dodawane do stroju.

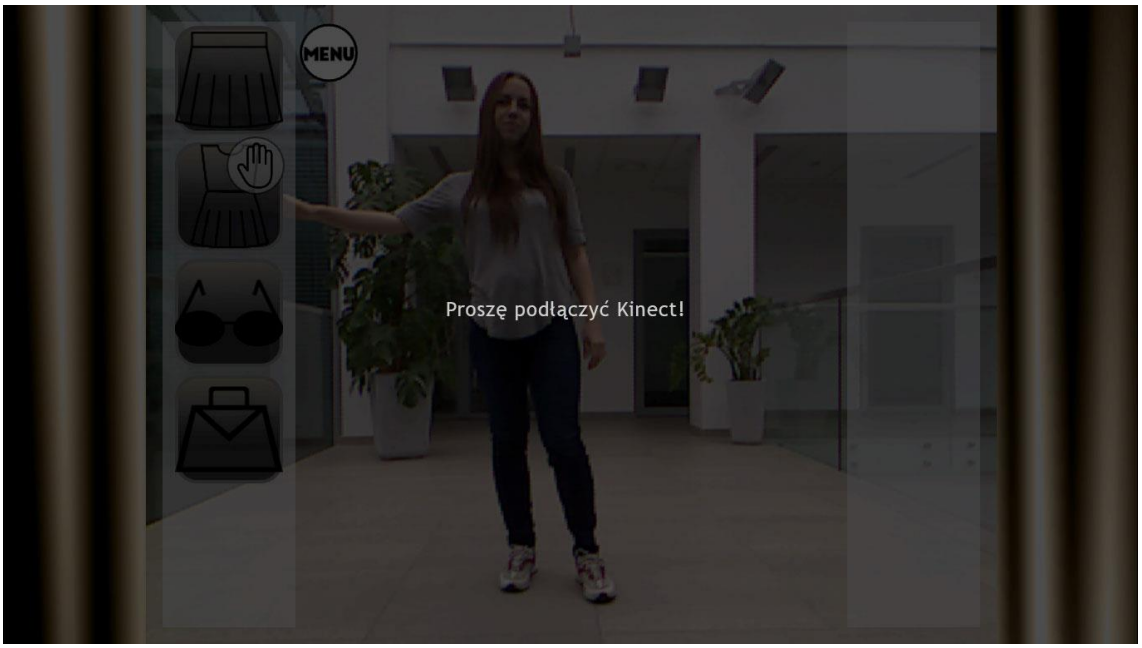
4.6.4. Obsługa aplikacji



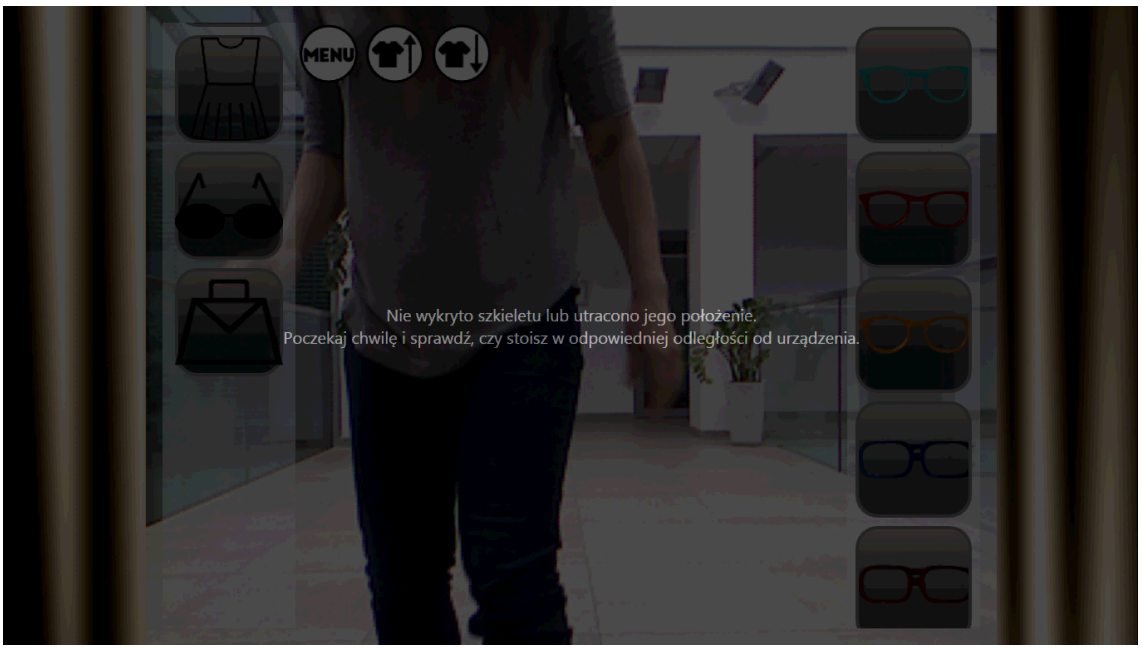
Rys. 18 Obsługa programu

Program sterowany jest za pomocą gestów. Gdy podczas użytkowania aplikacji ręka użytkownika znajdzie się nad górnym menu lub bocznymi panelami, pojawia się kursor. Aby zapewnić jak największą użyteczność aplikacji, kursor jest chowany na obszarze głównym aplikacji. Przyciśnięcie przycisku odbywa się poprzez przytrzymanie kursora nad jego obszarem. Kliknięciu towarzyszą efekty dźwiękowe.

4.6.5. Obsługa błędów



Rys. 19 Ekran z komunikatem o błędzie



Rys. 20 Ekran z informacją o utracie szkieletu

W przypadku, gdy urządzenie jest niepodłączone lub nieoczekiwanie zostanie odłączone od komputera lub zasilania, na ekranie pojawia się odpowiedni komunikat.

Po poprawnym wykryciu i inicjalizacji czujnika ekran z informacją znika i działanie aplikacji jest kontynuowane.

Aplikacja informuje użytkownika także o błędach związanych z niewykryciem szkieletu lub utracie jego położenia. Po ponownym wykryciu, wybrane wcześniej ubrania nadal przylegają do ciała i działanie aplikacji jest wznawiana.

Może się zdarzyć, że zasoby Kinecta używane będą przez inny proces, o czym użytkownik zostanie powiadomiony na ekranie. Należy wtedy upewnić się, że wszystkie aplikacje korzystające z urządzenia zostaną zamknięte i spróbować ponownie uruchomić *Wirtualną Przymierzalnię 3D*.

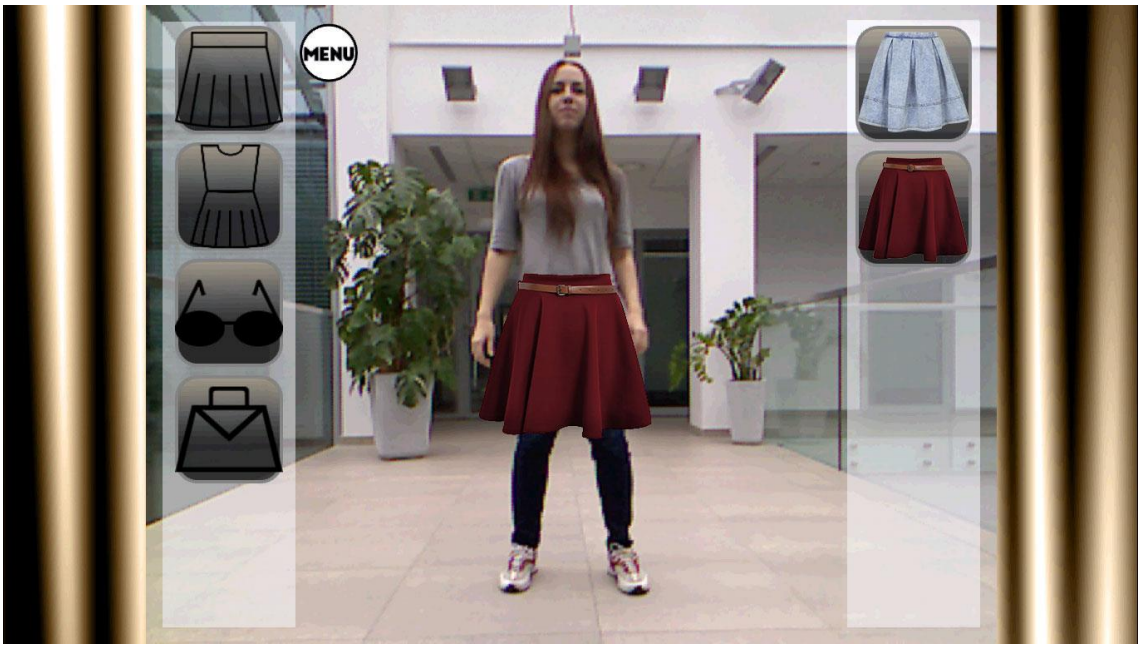
4.6.6. Zdjęcia

Rys. 21 Zdjęcie wykonane w aplikacji

Po wybraniu ikonki z aparatem, uruchomiony zostaje licznik informujący o czasie pozostałym do zrobienia zrzutu ekranu. Zdjęcia zapisywane jest na komputerze użytkownika w folderze *Moje Dokumenty*, w podfolderze *Wirtualna Przymierzalnia*. W prawym dolnym rogu widoczny jest znak wodny z ikoną programu. Wykonanie zdjęcia potwierdzone jest odpowiednim dźwiękiem.

5. Proces tworzenia aplikacji

Pierwszym etapem tworzenia programu było zapewnienie działania wszystkich funkcjonalności dla modeli ubrań 2D. Początkowo wyświetlany był jedynie szkielet użytkownika nałożony na obraz z kamery. Dane szkieletowe zmapowano do przestrzeni kontrolki wyświetlającej dane pobierane z bufora RGB. Kolejno dodano przyciski oraz zaimplementowano obsługę gestów. Ostatnim etapem pierwszej części było dodanie statycznych modeli ubrań 2D. Części garderoby zmieniały swój rozmiar w zależności od wysokości szkieletu oraz poruszały się razem z użytkownikiem. Nie posiadały jednak zdolności obracania się. Użytkownik miał możliwość zmiany rozmiaru ubrań, usunięcia wybranych części garderoby oraz mógł wykonywać zdjęcia.

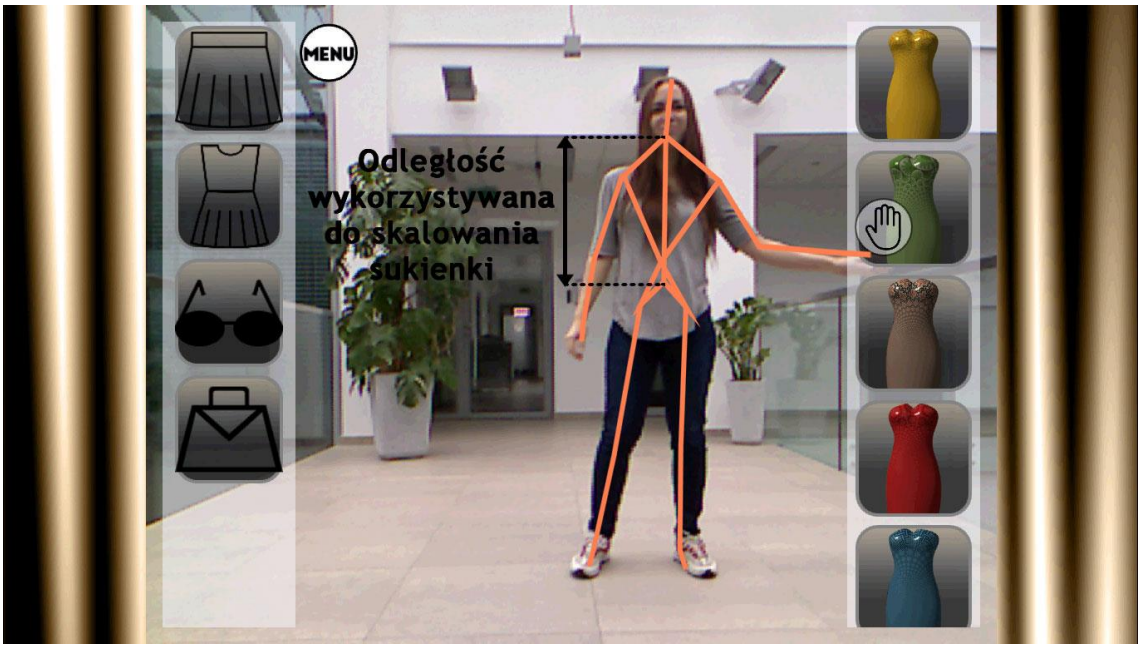


Rys. 22 Aplikacja z modelami 2D

Kolejnym krokiem było wprowadzenie mechanizmów obsługujących grafikę 3D. Wszystkie części garderoby zostały zastąpione modelami 3D. Do poprawnego skalowania, obracania i przemieszczania obiektów wykorzystywane są odpowiednie transformacje. Wykonywane są również operacje na macierzach przekształcenia w celu konwersji punktów z przestrzeni 2D do 3D.

Aplikacja nie wykorzystuje strumienia głębokości zapewnionego przez Kinect. Dlatego też skalowanie modeli ubrań oparte jest na obliczaniu odległości między dwoma danymi częściami szkieletu, a nie bezpośrednio na odległości czytanej z bufora. Dane pobierane z kamery głębokości dają również możliwość rozpoznawania użytkowników, co mogłyby wzbogacić program o możliwość uczestniczenia w zabawie więcej niż jednej osoby. Jednak z powodu ograniczeń sprzętowych związanych z ciągłym renderowaniem obiektów grafiki 3D, program nie umożliwia przymierzania ubrań wielu użytkownikom w tym samym czasie. Jeśli w zasięgu Kinecta znajduje się kilka osób, w zabawie bierze udział tylko ta, która stoi najbliżej sensora.

Niestety obraz uzyskiwany z kamery głębokości posiada inną rozdzielczość niż obraz czytany z kamery RGB. Nie można więc używać go do bezpośredniego pobierania informacji na temat szerokości użytkownika, które z pewnością zwiększyłyby precyzję w określeniu rozmiaru modeli ubrań. Wielkości części garderoby obliczane są w aplikacji za pomocą zależności między wysokością a szerokością człowieka.



Rys. 23 Obliczanie rozmiaru modelu sukienki

5.1. Napotkane problemy

Pierwszym napotkanym problemem, który towarzyszył do końca tworzenia aplikacji było mapowanie przestrzeni. Kinect udostępnia dane szkieletowe, które dzięki metodom pozyskanym z Kinect SDK można łatwo zmapować do przestrzeni aplikacji. Otrzymujemy wtedy punkty 2D z dodatkową wartością głębokości. Następnie wartości należy przeskalować do rozmiaru kontrolki wyświetlającej obraz z Kinecta. Dzięki tym operacjom uzyskane wartości pokrywają się ze strumieniem z kamery. Dużo większym problemem okazało się mapowanie obliczonych punktów na odpowiadające im punkty w przestrzeni kontrolki sceny 3D. Niezbędne okazało się obliczanie macierzy odwrotnych do macierzy projekcji oraz macierzy widoku.

Kolejnym problemem był brak wsparcia dla wyświetlania kolekcji modeli przez kontrolkę *Viewport3D*. Przy rozwiązaniu opisanym w paragrafie 3.3. wspierano się kodem demo dołączonego do biblioteki *Helix 3D Toolkit*. Niestety rozwiązanie zawierało wiele błędów, które ostatecznie udało się rozwiązać.

Podczas nakładania modeli 3D na obraz z kamery napotkano na kolejną przeszkodę. Każdy z obiektów należało dopasować do rozmiaru użytkownika. Wykorzystywano do tego celu dwa punkty szkieletowe znajdujące się na zbliżonej wysokości mające wpływ na rozmiar ubrania. Przykładowo dla sukienki odpowiednie były współrzędne bioder. Mierzono odległość między punktami i skalowano model tak, aby rozmiarem odpowiadał otrzymanym wymiarom. Kiedy użytkownik stał przodem do kamery, ubrania dopasowywały się do sylwetki. Jednak gdy użytkownik skręcił biodra, położenie punktów szkieletowych było estymowane, co powodowało otrzymywanie nieprawidłowych danych. Zastąpiono więc skalowanie na podstawie szerokości na skalowanie na podstawie długości. Dla sukienki mierzona jest długość od środka ramion do kolan lub stóp, dzięki czemu model pozostaje zawsze dopasowany rozmiarem do użytkownika.

6. Zakończenie

Stworzenie aplikacji *Wirtualna Przymierzalnia 3D* przyczyniło się do znacznego poszerzenia wiedzy z wielu dziedzin. Poznano nie tylko sposób implementacji programów wykorzystujących rozszerzoną rzeczywistość oraz SDK Kinect, lecz także ugruntowano wiedzę z zakresu grafiki 3D.

Stworzenie aplikacji przyczyniło się głównie do poznania SDK dostarczonego przez firmę Microsoft. Dedykowane oprogramowanie posiada szereg metod, które znacznie usprawniają współpracę z Kinectem. Dzięki API nie ma konieczności implementowania skomplikowanych algorytmów wykrywania szkieletu, czy pobierania położenia punktów z obrazu kamery. Dane dostarczane są przez sensor w przystępnej formie, dzięki czemu łatwo jest je przetworzyć i wykorzystać do celów aplikacji.

Stworzenie programu wykorzystującego rozszerzoną rzeczywistość wymagało odpowiedniej implementacji metod operujących na obiektach 3D. W tym celu wykorzystano zewnętrzne biblioteki zapewniające odpowiednie metody. Niezbędne okazało się jednak dodanie własnej implementacji rozszerzającej ich funkcjonalność lub korygującej błędy wykorzystywanego API.

Najważniejszymi danymi w programie są punkty szkieletowe użytkownika. To dzięki nim obliczane jest położenie modeli, ich wielkość i rotacja. Niestety wyświetlane elementy nie w pełni dopasowują swój kształt do sylwetki użytkownika. Jest to spowodowane specyfikacją użytych modeli 3D. Możliwość transformacji poszczególnych wierzchołków zapewniłaby bardziej realistyczny sposób wyświetlania ubrań, dzięki możliwości zastosowania animacji modeli. Niestety wymagałoby to wprowadzenia ogromnych zmian, a także dostarczenia własnej implementacji operacji na obiektach 3D. Zmiany takie planuje się wprowadzić podczas dalszego rozwoju aplikacji.

Wirtualna Przymierzalnia 3D należy do aplikacji nowatorskich, wykorzystujących dynamicznie rozwijające się technologie. Programy zapewniające mechanizm rozpoznawania gestów w przyszłości mogą zrewolucjonizować sposób interakcji człowieka z urządzeniami. Brak uciążliwych kontrolerów ruchu, intuicyjność i prostota obsługi sprawiają, że powstaje coraz więcej programów wykorzystujących te technologie. Kinect wykorzystywany jest nie tylko w celach rozrywkowych. Powstają

także programy do rehabilitacji oraz aplikacje umożliwiające kontrolę wyświetlanych prezentacji multimedialnych wykorzystujące sensor.

6.1. Rozwój aplikacji

Aplikacja zapewnia duży wybór ubrań i dodatków, jednak istnieje wiele możliwości rozbudowy, które pozwolą jej w przyszłości konkurować z komercyjnymi projektami.

Istotne jest dodanie takich części garderoby jak spodnie lub koszule. Modele te są trudne do implementacji ze względu na potrzebę odpowiedniej transformacji ubrań w miejscach zgięć kolan i łokci. Podobna sytuacja występuje w przypadku modeli obuwia, które do poprawnego wyświetlania potrzebują dokładnej lokalizacji stóp użytkownika, co nie jest zapewnione przez Kinect w wersji dla konsoli XBOX.

Kolejnym możliwym ulepszeniem, jakie w przyszłości można wprowadzić, jest implementacja zjawisk fizycznych i dodanie bardziej rzeczywistego zachowania się ubrań. Modele powinny dopasowywać się do wykrytego szkieletu. Podczas ruchu użytkownika powinny odpowiednio dostosowywać swój kształt i falować zgodnie z zasadami fizyki.

Spis ilustracji:

Bibliografia

1. Webb Jarrett, Ashley James, Beginning Kinect Programming with the Microsoft Kinect SDK, Apress, 2012

2. Kowalczyk Tomasz, Kinect SDK - Wprowadzenie, [online], aktualizacja 05/09/2011 [dostęp: 10/01/2015], Dostępny w Internecie: http://msdn.microsoft.com/pl-pl/library/kinect-sdk--wprowadzenie.aspx

3. 3-D Graphics Overview, [online], [dostęp: 13/01/2015], Dostępny w Internecie: http://msdn.microsoft.com/en-us/library/ms747437%28v=vs.110%29.aspx

4. Hodnick Mike, kinect-mvvm, [online], aktualizacja 20/07/2011 [dostęp: 06/11/2014], Dostępny w Internecie: https://github.com/kindohm/kinect-mvvm

5. Oystein Bjorke, Helix Toolkit, [online], aktualizacja 13/01/2015 [dostęp: 14/01/2015], Dostępny w Internecie https://github.com/helix-toolkit/helix-toolkit

6. Petzold Charles, 3D Programming for Windows, [online], aktualizacja 08/2007 [dostęp: 20/01/2015], Dostępny w Internecie: http://www.charlespetzold.com/3D/

Marta Kornaszewska Warszawa, 02.02.2015

Nr albumu 245534

Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym w pkt. 1.4.) pod tytułem „Wirtualna Przymierzalnia 3D", której promotorem jest dr inż. Paweł Kotowski, wykonałam samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Marta Kornaszewska