Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Pohybování 3D objekty pomocí gest

2015

Lukáš Medelský

Vedoucí práce: doc. RNDr. Michal Krupka, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Lukáš Medelský
Název práce:	Pohybování 3D objekty pomocí gest
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Michal Krupka, Ph.D.
Počet stran:	35
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Lukáš Medelský
Title:	Moving 3D objects by gestures
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Computer Science, full-time form
Supervisor:	doc. RNDr. Michal Krupka, Ph.D.
Page count:	35
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce má za cíl umožnit uživateli pohybovat, otáčet nebo měnit velikost 3D objektu pomocí pohybu ruky se speciální rukavicí. Popisuji zde programovací jazyk Java použitý při vývoji a také multiplatformní rozhraní OpenGL, s jehož funkcemi pracuji. Dále jsou popsány algoritmy, které v aplikaci při práci s obrazem využívám.

Synopsis

The purpose of this thesis is to allow a user to translate, rotate or scale 3D object by moving hand with a special glove. There is described programming language Java which I use to develope this application and also multiplatform application programming interface OpenGL. Furthermore, there are described algorithms which I use to work with image.

Klíčová slova: pohyb, 3D objekt, OpenGL, Java

Keywords: move, translate, 3d object, OpenGL, Java

Rád bych poděkoval doc. RNDr. Michalu Krupkovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

1 Úvod **9**

2 Programovací jazyk Java **10**
2.1 Popis . . . 10
2.2 Překlad . . . 10
2.3 Syntaxe . . . 10
2.4 Správa paměti . . . 11
2.5 Objektově orientovaný model . . . 11
2.6 Vývojové prostředí . . . 13
2.7 JavaFX . . . 14
2.8 Java 3D . . . 15

3 OpenGL **16**
3.1 Historie . . . 16
3.2 Jak funguje . . . 16
3.3 Syntaxe . . . 16
3.4 Grafická primitiva . . . 17
3.4.1 Bod . . . 17
3.4.2 Úsečka . . . 18
3.4.3 Polygon (konvexní) . . . 18
3.5 Transformace . . . 18
3.5.1 Rotace (rotate) . . . 18
3.5.2 Translace (translate) . . . 19
3.5.3 Změna měřítka (scale) . . . 19

4 Aplikace **20**
4.1 Popis . . . 20
4.2 Použité algoritmy . . . 20
4.2.1 Převod RGB do HSV . . . 20
4.2.2 Kalibrace barev . . . 21
4.2.3 Filtrování barev . . . 21
4.2.4 Eroze . . . 23
4.2.5 Středy terčů . . . 25

5 Uživatelská dokumentace **26**
5.1 Spuštění . . . 26
5.2 Průvodce kalibrací (wizard) . . . 26
5.3 Hlavní okno . . . 29
5.4 Pohybování 3D objektem . . . 29
5.4.1 Rotace . . . 30
5.4.2 Posunutí . . . 30
5.4.3 Změna velikosti . . . 31

Závěr 32

Conclusions 33

A První příloha 34

B Obsah přiloženého CD/DVD 34

Bibliografie 35

Seznam obrázků

1 Ukázka vývojového prostředí IntelliJ IDEA 13
2 Proces vykreslení na obrazovku 17
3 Filtrace kalibrovaných barev 23
4 Matematická eroze 24
5 Středy terčů (kvůli viditelnosti zvětšeny) 26
6 Úvodní stránka průvodce 26
7 Příprava na kalibraci bez fotky 27
8 Příprava na kalibraci s fotkou 27
9 Ukázka kalibrace červené barvy s označeným červeným terčem . . 28
10 Poslední krok průvodce 28
11 Hlavní okno aplikace 29
12 Ukázka rotace 30
13 Ukázka posunutí 30
14 Ukázka změny velikosti 31

Seznam zdrojových kódů

1 Hello World aplikace 11
2 Vytvoření třídy v Javě 12
3 Dědičnost v Javě 12
4 Vytvoření rozhraní a příklad implementace 13
5 JavaFX - ukázka 14
6 Java 3D - ukázka 15
7 OpenGL - více verzí funkce 17

Seznam algoritmů

1 Převod z RGB do HSV 21
2 Kalibrace barev 21
3 Filtrace kalibrovaných barev 22
4 Testování podobnosti odstínu barvy 22
5 Matematická eroze obrazu 23
6 Eroze jednoho pixelu 24
7 Střed terčů 25

1 Úvod

Cíl práce je vytvořit aplikaci umožňující otáčet 3D objekt pomocí pohybu ruky se speciální rukavicí. Funguje na podobném principu jako například Kinect od firmy Microsoft. Aplikace je psána v jazyce Java, tudíž je multiplatformní. V první části práce se zabývám použitým programovacím jazykem Java, její historii a vývojem. Malá ukázka syntaxe, její objektově orientované programování a rozhraní Java 3D. Dále je představeno vývojové prostředí IntelliJ IDEA, ve kterém pracuji. Navazuje kapitola o multiplatformním rozhraní OpenGL. Zde je trochu z historie této knihovny a popis jak knihovna funguje. Představeny jsou různé funkce a transformace, které OpenGL umí. Také ukázka použití a možností, kde lze s touto knihovnou pracovat.

Další kapitola již představuje samotnou aplikaci. Ze začátku tato sekce obsahuje seznámení se s ní, následně popis algoritmů použitých v aplikaci a jejich pseudokódy. Použil jsem například algoritmus pro převod obrázku z barevného modelu RGB do barevného modelu HSV a naopak. Algoritmus pro vyfiltrování třech barev (červená, zelená a modrá). Dále matematickou erozi a poté výpočet středu shluku pixelů, abych dostal pouze jeden pixel od každé ze tří barev. Následující sekce se věnuje struktuře projektu. Jsou zde popsány třídy a co obsahují a také použití dalších knihoven.

V poslední části práce je uživatelská dokumentace, kde je podrobně krok za krokem, pomocí obrázků a komentářů, popsán postup, jak aplikaci ovládat. Následuje poslední kapitola, ve které můžete vidět ukázku přímo z aplikace.

2 Programovací jazyk Java

Kapitola obsahuje popis programovacího jazyka Java, historii a ukázku kódu. V této části jsem čerpal z [1], [2], [3]

2.1 Popis

Java byla oficiálně představena v roce 1995 firmou Sun. Je to objektově orientovaný multiparadigmatický programovací jazyk a zároveň jeden z nejpoužívanějších programovacích jazyků na světě. Velkým přínosem je jeho přenositelnost programů mezi libovolné platformy. Programy mají pracovat na systémech, jako jsou čipové karty (JavaCard), mobilní telefony, desktopové počítače nebo třeba distribuované systémy pracující na řadě spolupracujících počítačů. Je silně staticky typovaný.

2.2 Překlad

Standardně program v Javě prochází pěti fázemi: editace, překlad (kompilace), zavedení (load), ověření (verifikace) a provádění. Překlad Javy neprobíhá klasicky do jazyka relativních adres (čili do .OBJ), což je v podstatě strojový kód, ale do instrukční sady nazývané *bytecode* (česky bajtkód) nebo také *p-code (portable code)*. Bajtkód je nezávislý na cílovém počítači, proto se programátor se vůbec nemusí starat o to, kde program bude spuštěn. Přeložený program je uložen v souborech s příponou *.class*. Tento soubor je pak načten do paměti a probíhá ověření bajkódu. Poté je program spuštěn pomocí interpretu. Java je tudíž jazyk interpretovaný, podobně jako BASIC. Interpretace je úkolem speciálních programů, souhrnně nazývaných *Java platforma*, které jsou ovšem předem pro tuto platformu připraveny. Java platforma se skládá ze dvou hlavních částí. První je *virtuální stroj* (JVM - Java Virtual Machine), který se skládá z části zajišťující vazbu na hardware a z části interpretující bajtkód. Druhou část tvoří Java Core API, což je velké množství knihovních tříd, které jsou považovány za standardní, takže se musí vyskytovat v každém prostředí, kde se Java používá.

2.3 Syntaxe

Je do velké míry odvozena od jazyka C++. Program se skládá ze tříd. Tyto třídy jsou složeny z definice dat a operací pracující s těmito daty. Data objektů jsou **atributy** a operace nad těmito daty se označují **metody**. Každá metoda se skládá z hlavičky, která definuje viditelnost metody (např. public, private, protected), návratový typ, jméno metody, parametry a tělo obsahující příkazy. Příkazy v těle pracují s atributy nebo lokálními proměnnými. Provádění příkazu zahrnuje vyhodnocení výrazů. Jakákoliv hodnota, proměnná nebo výraz je nějakého typu.

```
1
2  /**
3   * Created by Medel on 02/07/2015.
4   */
5
6  package helloworld;
7
8  /**
9   * The HelloWorld class implements an application that
10  * simply prints "Hello World!" to standard output.
11  */
12
13 public class HelloWorld {
14
15    /**
16     * Creates a new instance of HelloWorld
17     */
18    public HelloWorld() {
19    }
20
21    /**
22     * @param args the command line arguments
23     */
24    public static void main(String[] args) {
25       System.out.println("Hello World!"); // Display string Hello
             World!.
26    }
27 }
```

Zdrojový kód 1: Hello World aplikace

2.4 Správa paměti

Java podporuje automatickou správu paměti. To mimo jiné znamená, že se programátor ve většině případů nemusí explicitně starat o to, jakým způsobem se již nepotřebné objekty z paměti uvolní a jakým způsobem je jejich paměť organizovaná, popřípadě zda a jak se tato paměť defragmentuje. V některých případech je ovšem potřeba si dávat pozor na vytváření a rušení objektů. Je to operace poměrně náročná na výpočetní výkon mikroprocesoru, proto správné zacházení s pamětí, což je hlídaní zda se zbytečně nevytváří dočasné objekty, může někdy i velmi výrazně ovlivnit rychlost běhu aplikace.

2.5 Objektově orientovaný model

Java je objektově orientovaný jazyk, proto při psaní programu v Javě používáme třídy a objekty. Třída je návrh, podle kterého jsou vytvářeny jednotlivé objekty. Nový objekt lze vytvořit pomocí konstruktoru *new* například: `Bicycle bike = new Bicycle();`. Pak lze novému objektu změnit za pomocí metod dané

```
1
2  class Bicycle {
3
4        int speed = 0;
5        int gear = 1;
6
7        void changeGear(int newValue) {
8            gear = newValue;
9        }
10
11       void speedUp(int increment) {
12           speed = speed + increment;
13       }
14
15       void applyBrakes(int decrement) {
16           speed = speed - decrement;
17       }
18
19       void printStates() {
20           System.out.println("Speed: " + speed + " gear: " + gear);
21       }
22 }
```

Zdrojový kód 2: Vytvoření třídy v Javě

třídy jeho stav například: `bike.changeGear(3);`. Objektově orientované programování umožňuje třídám dědit obecně používané stavy od jiných tříd. U našeho příkladu s kolem by to bylo třeba horské nebo silniční kolo. V Javě se pro to používá klíčové slovo *extends*. Uvádí se při definici třídy a následuje název třídy, od které bude dědit.

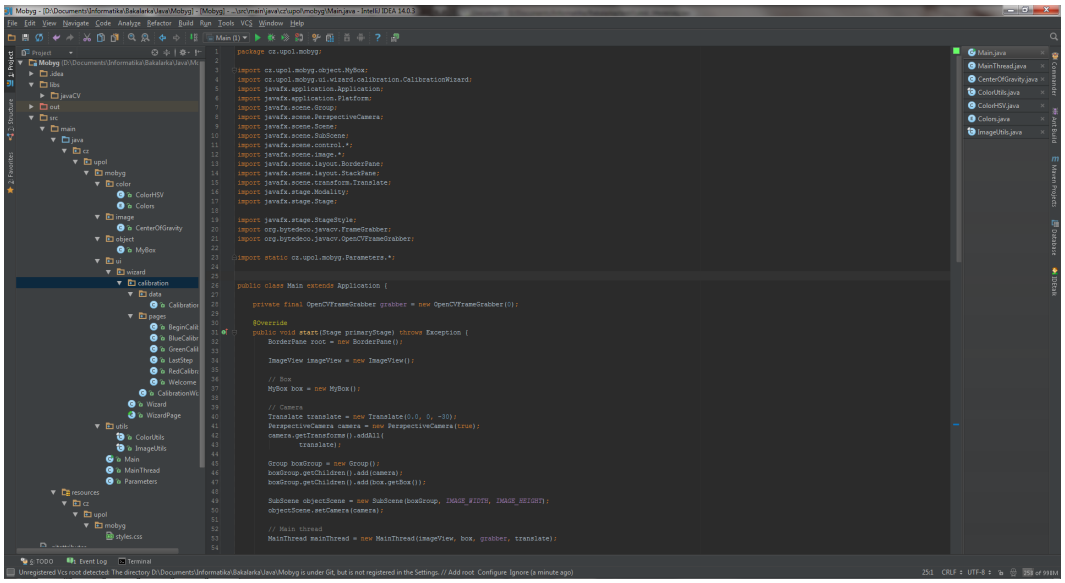
```
1
2  class MountainBike extends Bicycle {
3      // zde budou přidané vlastnosti a metody
4  }
```

Zdrojový kód 3: Dědičnost v Javě

Dále se nachází v Javě také rozhraní. Rozhraní je v podstatě skupina metod s prázdnými těly. Jestliže vaše třída implementuje nějaké rozhraní, tak se všechny metody, které jsou definované v rozhraní, musí objevit ve zdrojovém kódu třídy předtím, než bude třída úspěšně zkompilována. Rozhraní se vytvoří pomocí klíčového slova *interface* a pro implementaci použít klíčové slovo *implements*.

```
1  interface Bicycle {
2       public void speedUp(int increment);
3  }
4
5
6  class MyBicycle implements Bicycle {
7
8    @Override
9     public void speedUp(int increment) {
10      // potřeba nadefinovat tělo metody
11    }
12 }
```

Zdrojový kód 4: Vytvoření rozhraní a příklad implementace



Obrázek 1: Ukázka vývojového prostředí IntelliJ IDEA

2.6 Vývojové prostředí

Pro vývoj Javy existuje spousta vývojových prostředí, jako jsou IntelliJ IDEA, Eclipse, NetBeans nebo třeba JDeveloper. Při zpracování mé práce používám zmíněné IntelliJ IDEA vyvíjené společností JetBrains. Samozřejmě lze psát kód v klasickém poznámkovém bloku a pro kompilaci použít příkazový řádek, ale vývojové prostředí poskytuje spoustu nástrojů a usnadní programátorovi práci. Chytré upozorňování na chyby v kódu, doplňování kódu, obrovské množství konfigurovatelných nástrojů. Podpora mnoha jazyků, jako třeba SQL, HTML, CSS, PHP a další. Samozřejmě obsahuje nástroje jako profiler (kde se program nachází nejvíce času), debug mode(pro postupně procházení programu za běhu) nebo code coverage (při testech zobrazí, které části kódu byly otestovány a které ne).

2.7 JavaFX

JavaFX je moderní framework pro tvorbu RIA[1] aplikací, který jsem využil i pro tvorbu mé aplikace. Používá se jak pro tvorbu desktopových aplikací, tak pro webové applety nebo mobilní aplikace. Lze vyvíjet stejně jako u staršího Swingu tak, že se vytváří instance jednotlivých prvků, které pak vkládáme do layoutů (kontejnerů). Druhý způsob je použití FXML. Je to jazyk pro návrh formulářů, který je odvozen z XML. FXML se nemusí psát ručně a lze použít grafický designer Java Scene Builder, což je samostatná aplikace, ale je možné ho propojit s IntelliJ IDEA a editovat tak přímo ve vývojovém prostředí. Opět můžeme aplikaci s použitím JavaFX přenášet na různé platformy. Nahrazuje starší Swing, který nepodporuje nové technologické prvky, jako podporu dotykových zařízení, vytváření animací a práce s nimi. Klíčové rysy jsou například *SceneGraph*, což je stromová struktura definující uživatelské rozhraní, velké množství nadefinovaných komponent, možnost použít CSS stylů na všechny prvky (lze rozdělit práci grafikovi a programátorovi) a nebo také podpora pro 3D nástroje.

```
1  public class AppExample extends Application {
2        public static void main(String[] args) {
3        // spouštěcí metoda aplikace
4        launch(args);
5        }
6
7     @Override
8     public void start(Stage primaryStage) {
9
10    Button btn = new Button();  // vytvoření tlačítka
11    btn.setText("Testovací text");  // nastaví tlačítku nápis
12
13    Pane root = new Pane();   // vytvoří kořenový node (tzv. root)
14    root.getChildren().add(btn);  // Přidá se tlačítko do rootu
15
16       // vytvoření scénu ve které se bude všechno z rootu
             vykreslovat.
17       Scene scene = new Scene(root, 300, 250);
18
19       primaryStage.setTitle("Ukázka");// nastaví Stage(okno) nadpis
20     primaryStage.setScene(scene); // nastaví do Stage scénu
21     primaryStage.show();   // zobrazí stage
22     }
23 }
```

Zdrojový kód 5: JavaFX - ukázka

[1] Rich Internet Application

2.8 Java 3D

Java 3D je programovací rozhraní pro 3D aplikace na platformě Java, založené na vykreslování do scény. Do verze Java 1.6 běželo na OpenGL[2] a poté na JOGL[3]. JOGL je knihovna, která obaluje OpenGL a umožňuje tak použití v Javě. V porovnání s jinými řešeními, Java 3D není pouze obal kolem těchto grafických rozhraní, ale rozhraní, které zapouzdřuje programování grafiky pomocí objektově orientovaného přístupu. Umožňuje vytvořit tří-rozměrnou grafickou aplikaci nebo webový applet a poskytuje konstrukce na vysoké úrovni pro tvorbu a manipulaci s 3D geometrií.

```
1    // vytvoří kvádr o daných rozměrech
2   Box myBox = new Box(width, height, depth);
3
4    // nový materiál
5   final PhongMaterial redMaterial = new PhongMaterial();
6    redMaterial.setSpecularColor(Color.ORANGE);
7    redMaterial.setDiffuseColor(Color.RED);
8
9    // nastaví materiál našemu kvádru
10   myBox.setMaterial(redMaterial);
11
12   // vytvoření kamery
13   PerspectiveCamera camera = new PerspectiveCamera(true);
14   camera.getTransforms().addAll (
15             new Rotate(-10, Rotate.Y_AXIS),
16             new Rotate(-10, Rotate.X_AXIS),
17             new Translate(0, 0, -10));
18
19   // změna velikosti objektu
20   Scale scale = myBox.getScale();
21   scale.setX(2);
22   scale.setY(10);
23   scale.setZ(10);
```

Zdrojový kód 6: Java 3D - ukázka

[2]Open Graphics Library
[3]Java OpenGL

3 OpenGL

Kapitola představuje multiplatformní rozhraní OpenGL pro práci s počítačovou grafikou. V této části jsem čerpal z [4], [5], [6]

3.1 Historie

Knihovna OpenGL byla vytvořena firmou SGI[4] jako aplikační programové rozhraní - API[5] k akcelerovaným grafickým kartám resp. celým grafickým subsystémům. Předchůdcem byla knihovna IRIS. OpenGL byla navržena, aby byla použitelná na různých typech grafických karet a aby bylo možné ji použit i v případě, že žádný grafický akcelerátor není nainstalován. Knihovna je multiplatformní, tudíž ji lze použít na různých platformách.

3.2 Jak funguje

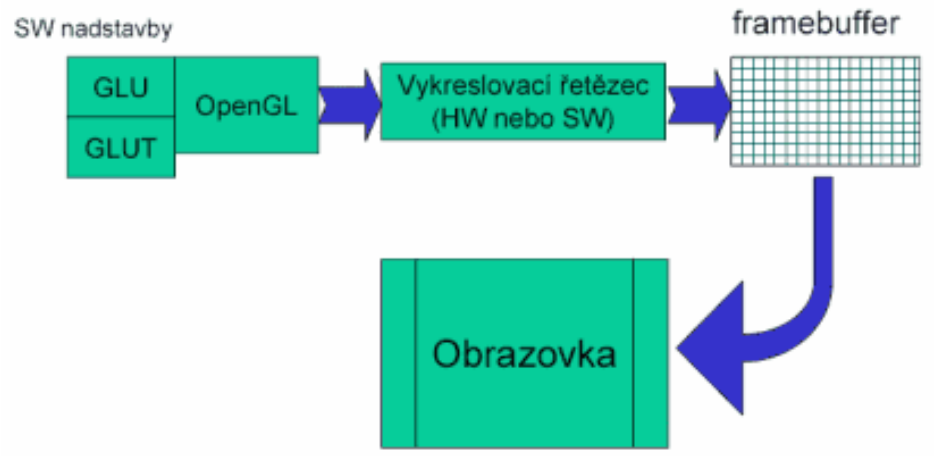
Z programátorského hlediska se OpenGL chová jako stavový automat. Během zadávání příkazů pro vykreslování lze průběžně měnit vlastnosti vykreslovaných primitiv (barva, průhlednost) nebo celé scény (transformace). Tento stav je zachován, než ho opět změníme. Díky tomuto způsobu mají funkce menší počet parametrů a jedním příkazem můžeme ovlivnit vykreslení celé scény, jako třeba zobrazení vyplněných polygonů (filled model). Výsledkem volání těchto funkcí je rastrový obrázek uložený ve *framebufferu*, kde každý pixel má přiřazenou barvu, hloubku, alfa složku nebo další atributy. Z framebufferu lze získat pouze barevnou informaci a tu je možné následně zobrazit na obrazovce, jak můžeme vidět na obrázku č.2. OpenGL nezaručuje, že při spuštění identického programu, používajícího knihovnu OpenGL, na různých platformách nebo různých grafických kartách dostaneme vždy identický výsledek. Pokud bychom pixel po pixelu porovnali výsledné obrázky, mohli bychom zjistit mírné rozdíly v barvách. Celkové geometrické a barevné podání scény by však mělo být zachováno.

3.3 Syntaxe

Většina funkcí, které OpenGL obsahuje, používá syntaxi, kde už podle jména funkce je zřejmé kolik parametrů a jakého typu jsou. Samozřejmě syntaxe se liší podle programovacího jazyka, ale vždy je to velmi podobné a obsahují stejné funkce. Na příkladu `glColor3f(0.0f, 1.0f, 0.0f)` si popíšeme syntax funkci deklarovaných v OpenGL. Jméno funkce začíná prefixem *gl*. Za prefixem následuje hlavní jméno funkce, která označuje vytvářený objekt nebo vlastnost, která se bude měnit. Hlavní jméno začíná velkým písmenem a u víceslovných názvů vždy každé další slovo začíná také velkým písmenem. Dále je číslo určující počet parametrů. Pokud funkce nemá parametr, číslo se nepíše. Jako poslední

[4]Silicon Graphics Inc.

[5]Application Programming Interface



Obrázek 2: Proces vykreslení na obrazovku

je jedním nebo dvěma znaky uveden typ parametrů. V OpenGL většina funkcí existuje ve více verzích, lišících se typem nebo počtem parametrů.

```
1    glVertex2i(int1, int2);
2    glVertex2d(double1, double2);
3    glVertex3f(float1, float2, float3);
4    glVertex4f(float1, float2, float3, float4);
```

Zdrojový kód 7: OpenGL - více verzí funkce

3.4 Grafická primitiva

V OpenGL lze vykreslovat pomocí funkcí pouze základní prvky - primitiva, ze kterých se poté skládají složitější tělesa a celé scény. Je celkem deset typů primitiv, z nichž si popíšeme bod, úsečku a trojúhelník. Vykreslování primitiva začíná vždy příkazem *glBegin (typ primitiva)*, končí pak příkazem *glEnd()*. Mezi tyto příkazy lze zadat libovolný počet vrcholů primitiva příkazem *glVertex*()*. Vrcholy jsou určené svými souřadnicemi.

3.4.1 Bod

Je nejjednodušší primitivum v OpenGL. Klíčové slovo pro kreslení bodů je `GL_POINTS`. Každý bod je zadán jedním příkazem *glVertex*()*. Vlastností bodu, která je stejná pro všechna primitiva, je barva vrcholu. Další vlastnosti je velikost. Implicitně je nastavena na 1.0, což je jeden pixel na obrazovce. Třetí vlastností je antialiasing, kterou lze vypnout nebo zapnout. Využívá se pro rozmazání hran. Při vypnutí se body zobrazují jako čtverce. V opačném případě se zobrazují jako kruhy s rozmazanými okraji.

3.4.2 Úsečka

V OpenGL je úsečka definována svými dvěma koncovými body. Klíčové slovo pro vykreslování úsečky je `GL_LINES`. Poté nasledují dvojice vrcholů, kde každý reprezentuje jednu úsečku. Minimálně je potřeba zadat aspoň jednu dvojici vrcholů. Stejně jako u bodu, má úsečka možnost změnu barvy. Lze vykreslit celou úsečku jednou barvou (barva se zadá předem) nebo každému vrcholu nastavit jinou barvu a grafická karta potom provede lineární interpolaci barev mezi vrcholy. Druhou vlastností je tloušťka úsečky, kde základní hodnota je 1.0 a musí být větší jak 0. Poslední vlastností je maska. Je to šestnáctibitová hodnota, která specifikuje vzorek, jak má být úsečka vykreslena.

3.4.3 Polygon (konvexní)

Polygon je nejsložitější z primitiv OpenGL. Musí platit, že všechny vrcholy budou ležet v jedné rovině a výsledný polygon bude konvexní. Často je podmínka konvexnosti těžko splnitelná, proto se polygon rozděluje na trojúhelníky. Klíčové slovo pro kreslení polygonu je `GL_POLYGON`. Následují souřadnice jednotlivých vrcholů. S barvou polygonu je to stejné jako u úsečky. Lze nastavit jednotnou barvu celému polygonu nebo každému vrcholu zvlášť. Jedna z vlastností plošných útvarů je *výplňový vzorek*. Zadá se bitová maska o velikosti 32x32 bitů, která určuje vzorek, jímž bude polygon vyplněn.

3.5 Transformace

Pro vyjádření lineárních transformací se v počítačové grafice používají *transformační matice*. Mezi lineární transformace patří translace, rotace, změna měřítka a zkosení. Všechny lineární transformace lze vyjádřit maticí o $n * n$ rozměrech. Existují tři transformační matice. *ModelView matrix*, která nastavuje pozici kamery a používá se i pro manipulaci s objekty ve scéně. *Projection matrix* se používá pro nastavení perspektivní projekce kamery. A poslední, *Viewport matrix*, slouží po provedení perspektivní projekce k mapování objektů z abstraktních souřadnic do souřadnic okna.

3.5.1 Rotace (rotate)

Těleso je otočeno o zadaný úhel θ okolo osy procházející počátkem soustavy.

- Rotace kolem osy X

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotace kolem osy Y

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotace kolem osy Z

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.5.2 Translace (translate)

Posune objekt o zadaný vektor $[x, y, z]$.

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.5.3 Změna měřítka (scale)

Slouží k zadání transformace změny měřítka. Těleso se zvětší nebo zmenší ve směrech odpovídajícím jednotlivým souřadným osám.

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4 Aplikace

Kapitola popisuje aplikaci samotnou, její fungování a použité algoritmy.

4.1 Popis

Jak již bylo zmíněno aplikace funguje na podobném principu jako Kinect od firmy Microsoft. Ovšem cílem nebylo udělat stejnou aplikaci ani na stejné úrovni. To by bylo příliš složité na úroveň bakalářské práce a také velmi časově náročné. V této aplikaci jde o vlastní vyzkoušení problematiky spojené s rozpoznáním obrazu (bez použití složitých knihoven) a podle změn pohybovat s 3D objektem.

Program tedy umožňuje uživateli pohybovat 3D objektem v počítači pomocí pohybu ruky před kamerou. Pro správnou funkčnost je potřeba použití speciálně upravené rukavice. Tato rukavice má na třech prstech přilepené barevné terče. Na palci se nachází modrý, na ukazováčku červený a na prostředníčku zelený terč. Díky těmto terčům rozpoznávám tři odlišné body. Osvětlení prostředí, kde se uživatel nachází, není vždy stejné a intenzita světla může být různá, proto při spuštění aplikace musí uživatel nakalibrovat barvy, což se provede tak, že označí jeden ze tří terčů podle toho, kterou barvu zrovna kalibruje. Po skončení kalibrace už uživatel může pohybovat rukavicí před kamerou a provádět transformace s objektem. Lze provádět tři transformace: rotaci, posunutí a změnu velikosti.

Rozpoznání obrazu funguje tak, že podle přesně nakalibrovaných barev se vyfiltrují z obrázku jen ty barvy, které jsou přibližně stejné. Poté se provede matematická eroze, která zanechá pouze velké shluky pixelů daných třech barev, což znamená, že zůstanou pouze pixely z terčů. Nakonec se provede výpočet středu shluku pixelů od každé barvy a zbude pouze jeden červený, modrý a zelený pixel. Po výpočtu rozdílů těchto pixelů z předchozího snímku a aktuálního se provede transformace.

4.2 Použité algoritmy

Kapitola představuje použité algoritmy a k čemu v aplikaci slouží. Je to převod z RGB modelu obrázku do HSV modelu, kalibrace barev, vyfiltrování třech barev, matematická eroze a výpočet středu terčů.

4.2.1 Převod RGB do HSV

Funkce, která převede obrázek z RGB modelu do HSV modelu. V algoritmu pro filtrování barev potřebuji určit podobné barvy (s nějakým rozsahem), jako jsou ty kalibrované a k tomu se přímo nabízí HSV model. Je to z důvodu, že tento model nejvíce odpovídá lidskému vnímání barev. *Hue* nastavuje barevný tón, *Saturation* odpovídá sytosti nebo také čistotě barvy a *Value* je hodnota jasu nebo-li množství bílého světla.

Algoritmus 1 Převod z RGB do HSV

Vstup: *imageRGB* - RGB obrázek

```
1:  procedure RGBtoHSV
2:      imageWidth ← getWidth(imageRGB)
3:      imageHeight ← getHeight(imageRGB)
4:      result[][] ← new array[imageWidth][imageHeight]
5:      for i = 0 to imageHeight do
6:          for j = 0 to imageWidth do
7:              colorRGB ← getColor(imageRGB, j, i)
8:              colorHSV ← convertRGBtoHSV(colorRGB)
9:              result[j][i] ← colorHSV
10:     return result
```

4.2.2 Kalibrace barev

Je nemožné nastavit pevné hodnoty barev z terčů rukavice, jelikož se během dne mění intenzita světla a barvy jsou světlejší nebo naopak tmavší. Nebo při použití umělého světla dostanou barvy jiný odstín. Proto je nutné použít při spuštění aplikace kalibraci barev, aby odpovídaly barvy aktuálním podmínkám světla.

Algoritmus 2 Kalibrace barev

Vstup: *calibratingColor* - barva která se kalibruje

```
1:  procedure calibrateColor
2:      pickedRGBColor ← getSelectedColor(image)
3:      pickedHSVColor ← convertRGBtoHSV(pickedRGBColor)
4:      if calibratingColor = RED then
5:          redHSV ← pickedHSVColor
6:      if calibratingColor = GREEN then
7:          greenHSV ← pickedHSVColor
8:      if calibratingColor = BLUE then
9:          blueHSV ← pickedHSVColor
```

4.2.3 Filtrování barev

Poté co jsou barvy nakalibrované, je potřeba ostatní barvy odfiltrovat. Algoritmus má k dispozici hodnoty jednotlivých složek kalibrovaných barev (*redHue*, *redSat*, ...) a obrázek jako pole pixelů. Prochází pole pixel po pixelu a kontroluje zda aktuální barva pixelu splňuje podmínky podobnosti nějaké z kalibrovaných barev (červené, zelené nebo modré). To znamená, jestli jsou všechny složky testované barvy v nastaveném intervalu podobnosti kalibrované barvy. Pokud ano, nastaví se pixelu jedna ze základních barev *Red (255, 0, 0), Green (0, 255, 0)* nebo *Blue (0, 0, 255)* podle toho, které barvě je aktuální pixel podobný. Pokud ne, pak se pixel nastaví na černou barvu.

Algoritmus 3 Filtrace kalibrovaných barev

Vstup: *arrayHSV* - pole HSV barev

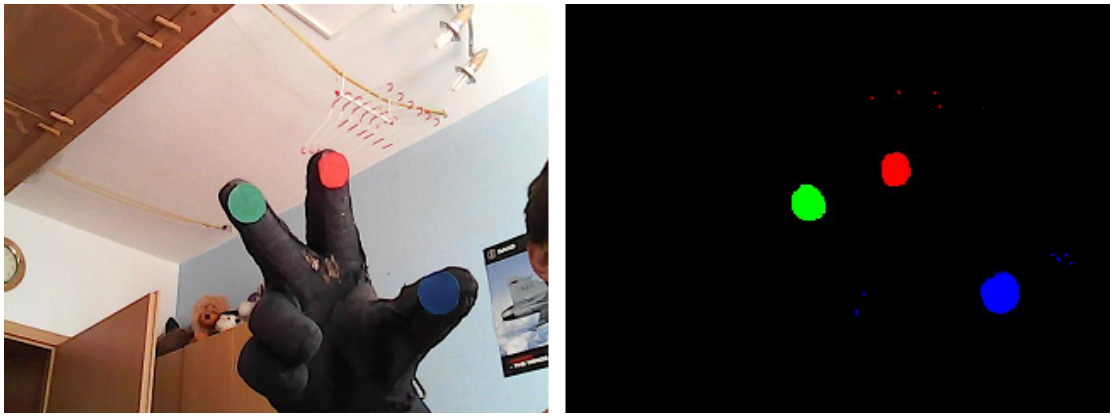
Výstup: *result* - pole RGB barev

```
1: procedure filterPoints
2:     for i = 0 to height do
3:         for j = 0 to width do
4:             hsv ← arrayHSV[j][i]
5:
6:             h ← getHue(hsv)
7:             s ← getSaturation(hsv)
8:             v ← getValue(hsv)
9:
10:            if (isColorIn(redHue, 15, h)
    and (s > redSat) and (v > redVal)) then
11:                result[j][i] ← rgb(255, 0, 0)
12:            else
13:                if (isColorIn(greenHue, 15, h)
    and (s > greenSat) and (v > greenVal)) then
14:                    result[j][i] ← rgb(0, 255, 0)
15:                else
16:                    if (isColorIn(blueHue, 15, h)
    and (s > blueSat) and (v > blueVal)) then
17:                        result[j][i] ← rgb(0, 0, 255)
18:                    else
19:                        result[j][i] ← rgb(0, 0, 0)
20:    return result
```

Algoritmus 4 Testování podobnosti odstínu barvy

Vstup: *center* - hodnota kalibrované barvy, *range* - vzdálenost od barvy, *color* - testovaná hodnota barvy

```
1: procedure isColorIn
2:     firstDistance ← abs(center − color)
3:     secondDistance ← 360 − firstDistance
4:     distance ← min(firstDistance, secondDistance)
5:     return distance ≤ range
```



Obrázek 3: Filtrace kalibrovaných barev

4.2.4 Eroze

Eroze a Dilatace jsou základní metody matematické morfologie, která se používá pro extrakci obrazových komponent, předzpracování i segmentaci obrazu. Princip eroze je zmenšení obrázku a odstranění nežádoucích osamocených pixelů a také doplnění chybějících. Základní princip matematické eroze pro černobílé obrázky je pohyb pomocného binárního obrazu nebo-li sondy přes všechny pixely obrazu a sledování jak velkou částí je sonda pod objektem na obrázku.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Příklad pomocné sondy

Označme $\mathbf{B}$ vstupní binární obraz a $\mathbf{S}$ jako sondu. Označením $\mathbf{S_{xy}}$ rozumíme výsledný obraz vniklý translací sondy, kde střed sondy je v bodě (x, y). Erozí obrazu $\mathbf{B}$ za použití sondy $\mathbf{S}$ vznikne obraz $\mathbf{E}$. V bodě o souřadnicích (x, y) je v obraze $\mathbf{E}$ hodnota 1, jestliže je v obraze $\mathbf{B}$ hodnota 1 alespoň na těch místech, kde je hodnota 1 v sondě $\mathbf{S_{xy}}$. Jinak je hodnota 0. Operaci eroze definujeme následovně:

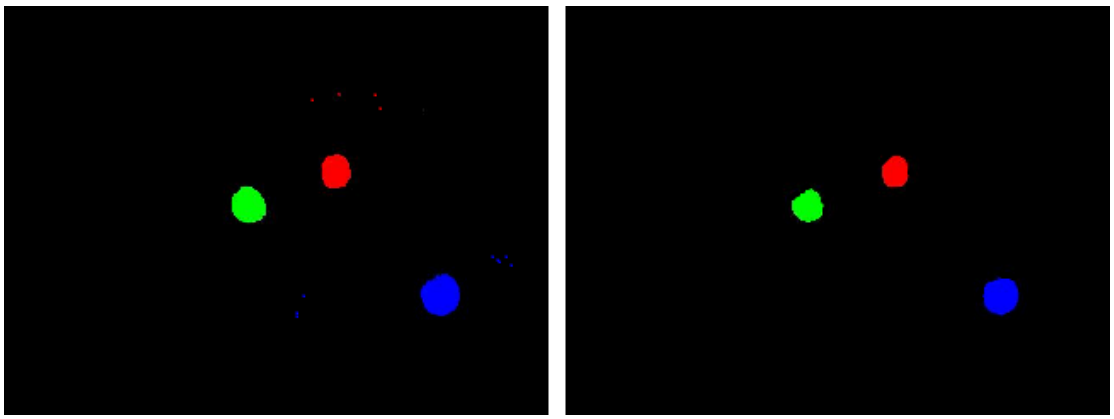
$$\mathbf{E} = \mathbf{B} \otimes \mathbf{S} = \{(x, y) | \mathbf{S_{xy}} \subseteq \mathbf{B}\}$$

Algoritmus 5 Matematická eroze obrazu

```
1: procedure EROSION
2:     for i = 0 to height do
3:         for j = 0 to width do
4:             result[j][i] ← erodeOnePixel(rgbArray, j, i)
5:     return result
```

Algoritmus 6 Eroze jednoho pixelu

```
1:  procedure erodeOnePixel
2:      red, green, blue ← 255
3:      for i = 0 to 2 do
4:          for j = 0 to 2 do
5:              testX ← x − 1 + i
6:              testY ← y − 1 + j
7:              if isInArray(rgbArray, testX, testY) then
8:                  red ← min(red, getRed(rgbArray[testX][testY]))
9:                  green ← min(green, getGreen(rgbArray[testX][testY]))
10:                 blue ← min(blue, getBlue(rgbArray[testX][testY]))
11:     return rgb(red, green, blue)
```



Obrázek 4: Matematická eroze

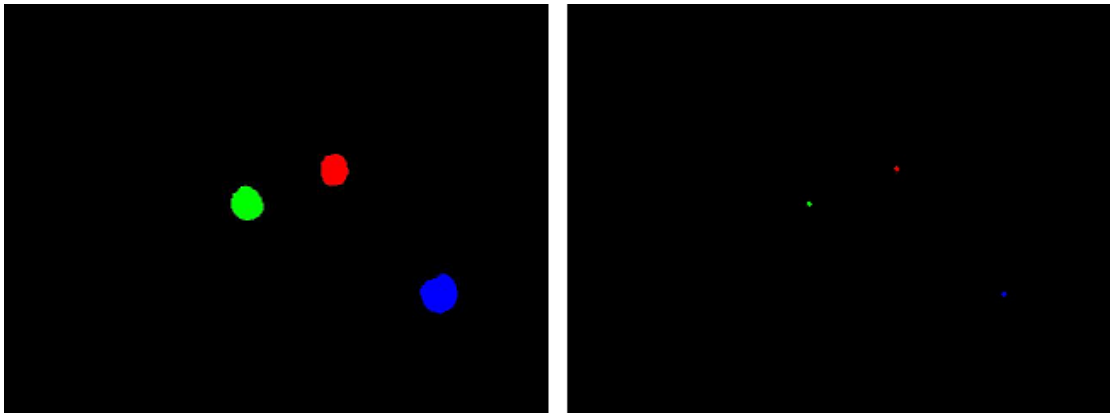
4.2.5 Středy terčů

Po provedení eroze zůstanou v obrázku pouze tři shluky pixelů (každý má svou barvu - červená, zelená, modrá) odpovídající terčům. Nyní je potřeba vypočítat střed každého terče a získat tak tři body, podle kterých se budou počítat změny mezi snímky z kamery. Algoritmus funguje na principu výpočtu těžiště útvaru. Prochází se postupně celé pole pixelů, pokud narazí na jednu ze tří barev, přičte hodnoty do akumulátorů pro souřadnice x a y a také do akumulátoru pro celkový počet pixelů dané barvy. Při každém kroku cyklu se na konci nastaví aktuální pixel na černou barvu. Na konci algoritmu se pro každou barvu vypočtou výsledné souřadnice a nastaví příslušná barva. Po skončení algoritmu je celé pole černé a pouze jeden pixel červené, zelené a modré barvy.

Algoritmus 7 Střed terčů

```
1: procedure getMiddleRGBPixels
2:     colorAccumX ← 0
3:     colorAccumY ← 0
4:     colorAccumN ← 0
5:     for i = 0 to height do
6:         for j = 0 to width do
7:             testX ← x − 1 + i
8:             testY ← y − 1 + j
9:             if getColor(rgbArray[j][i]) = 255 then
10:                colorAccumX ← colorAccumX + j)
11:                colorAccumY ← colorAccumY + i)
12:                colorAccumN + +
13:            result[j][i] ← rgb(0, 0, 0)
14:    if colorAccumN ≠ 0 then
15:        result[colorAccumX/colorAccumN][colorAccumY/colorAccumN] ←
   color
16:    return result
```

V uvedeném algoritmu jsou hodnoty **Red, Green a Blue** nahrazeny hodnotou **Color**.



Obrázek 5: Středy terčů (kvůli viditelnosti zvětšeny)

5 Uživatelská dokumentace

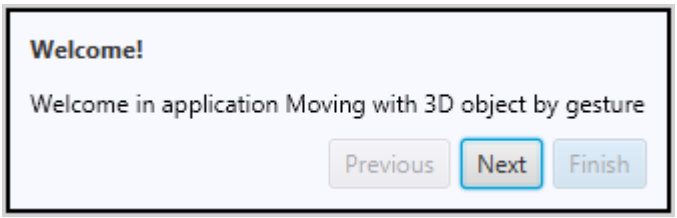
Tato kapitola seznámí uživatele s aplikací a slouží jako návod, jak aplikaci používat.

5.1 Spuštění

Aplikace se neinstaluje a spouští se přímo z *Executable Jar File*. Tento spustitelný soubor naleznete na `CD/src/Mobyg/out/artifacts/Mobyg_jar/Mobyg.jar`

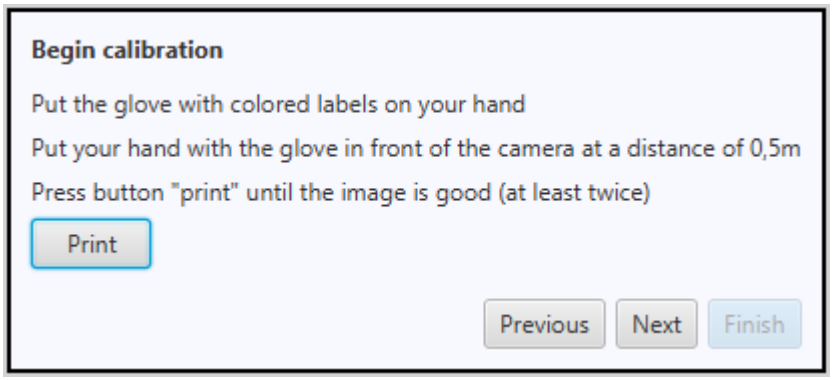
5.2 Průvodce kalibrací (wizard)

Jakmile spustíte aplikaci, v pozadí se otevře hlavní okno a v popředí průvodce kalibrací barev - wizard. Vždy je možnost se vrátit na kteroukoli z předchozích stránek kliknutím na tlačítko *Previous*. První stránka průvodce je pouze uvítací, jak můžete vidět na následujícím obrázku.



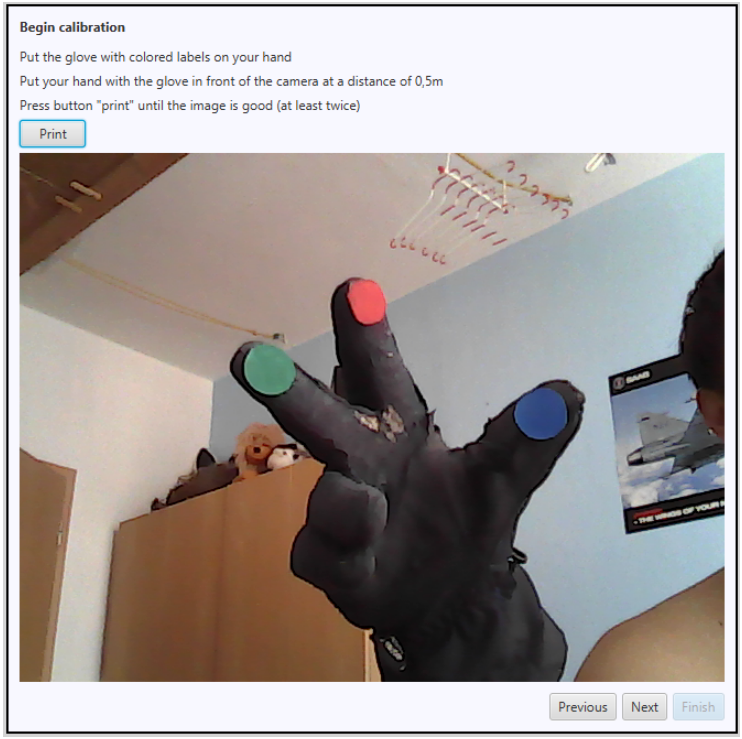
Obrázek 6: Úvodní stránka průvodce

Kliknutím na tlačítko *Next* přejdete na následující stránku, kde si nasadíme speciální rukavici s barevnými štítky. Nastavíme ruku s rukavicí před kameru, přibližně do vzdálenosti padesáti centimetrů a zmáčkneme tlačítko *Print*.



Obrázek 7: Příprava na kalibraci bez fotky

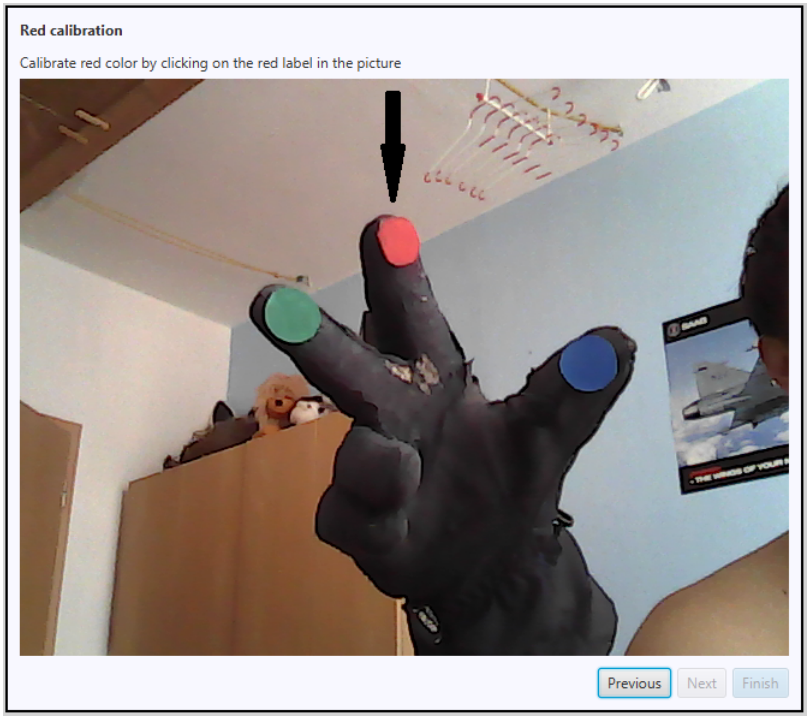
Po zmáčknutí tlačítka *Print* se níže zobrazí fotka z webkamery. Je vhodné zmáčknout tlačítko vícekrát (alespoň dvakrát), než budou barvy na fotce kvalitní a dobře viditelné, jako na obrázku 8. Poté můžeme přejít na další stránku průvodce kliknutím na tlačítko *Next*.



Obrázek 8: Příprava na kalibraci s fotkou

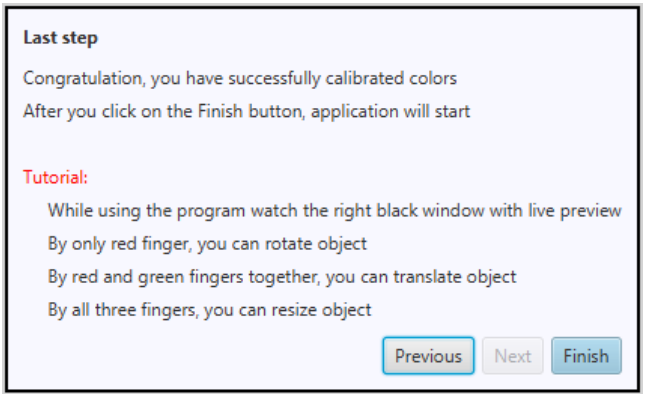
Na následujících třech stránkách jste postupně vyzváni ke kalibraci každé ze tří barev. Postupně to jsou červená, zelená a modrá. Všechny tři stránky vypadají stejně s rozdílem nadpisu a instruktážního textu, kde je třeba se dívat jakou barvu právě kalibrujete. Kalibrace se provádí velmi jednoduše. Stačí vždy kliknout do obrázku na terč se stejnou barvou, jakou právě kalibrujete a poté kliknout na

tlačítko *Next*. Dokud nekliknete do obrázku (nenakalibrujete barvu) nelze jít na další krok v průvodci. Pokud jste klikli na špatnou barvu, můžete se opravit novým kliknutím. Jestli si potřebujete opravit nějakou předchozí kalibraci, stačí kliknout na tlačítko *Previous*.



Obrázek 9: Ukázka kalibrace červené barvy s označeným červeným terčem

Po projití kalibrace všech tří barev se dostaneme na poslední stranu wizarda. Barvy jsou nakalibrované a vše je připravené na spuštění hlavní aplikace pomocí tlačítka *Finish*.

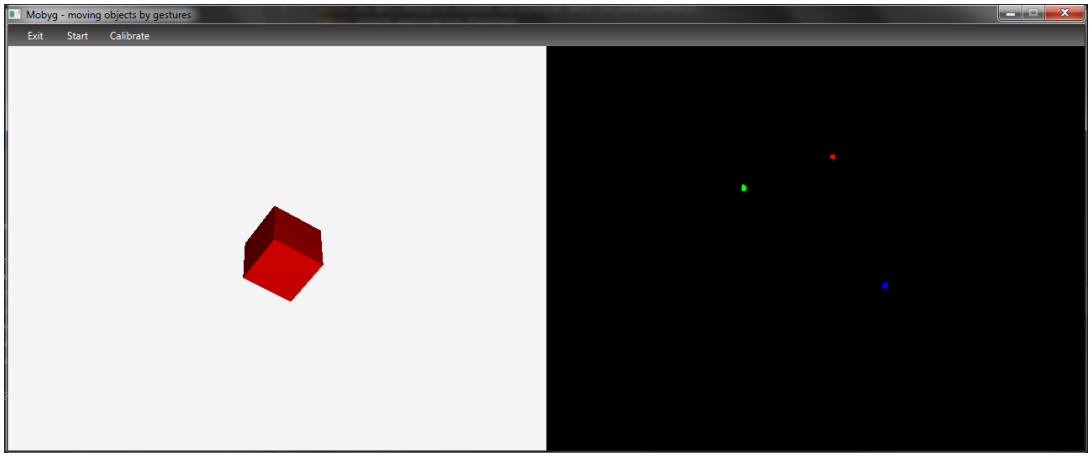


Obrázek 10: Poslední krok průvodce

5.3 Hlavní okno

Zde se provádí pohyb s 3D objektem a ovládání celé aplikace.

V horní části je menu s tlačítky *Exit*, *Start* a *Calibrate*. Tlačítko *Exit* slouží k řádnému ukončení aplikace. Následující *Start* spustí hlavní vlákno aplikace. Pokud již aplikace běží, je uživateli zabráněno ji opět spustit a je informovaný dialogovým oknem. Stejně tomu je pokud by chtěl uživatel spustit aplikaci bez kalibrace barev. Opět je informován dialogovým oknem, že je potřeba nakalibrovat barvy. Poslední z menu nabídky je tlačítko *Calibrate*, které slouží k vyvolání průvodce kalibrací. Lze toto tlačítko použít i za běhu aplikace a znovu nakalibrovat barvy. Tato událost zastaví běh aplikace a umožní uživateli opětovnou kalibraci barev. Levá polovina okna obsahuje scénu s 3D objektem a v reálném čase zobrazuje transformace s ním prováděné. V druhé polovině hlavního okna se nachází scéna s náhledem na středy terčů rukavice, která je také v reálném čase. Zde můžeme vidět, které z terčů aplikace aktuálně rozpoznává a lze se podle toho orientovat v jaké poloze se nachází rukavice.



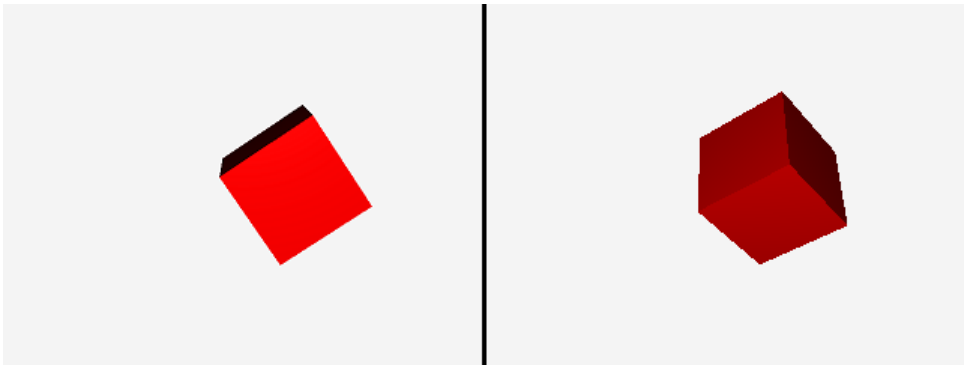
Obrázek 11: Hlavní okno aplikace

5.4 Pohybování 3D objektem

V této části je vysvětleno ovládání aplikace, tedy jak pohybovat s objektem. Je důležité sledovat zároveň černé okno (náhledové - preview) v pravé části hlavního okna, kde můžete vidět pohybující se pixely červené, zelené a modré barvy. Každý odpovídá jednomu z terčů, podle toho, které jsou aktuálně viditelné. Pokud některou z barev, které ukazujete, nevidíte v náhledovém okně, zkuste lehce naklopit nebo natočit ruku vzhledem ke kameře. Někdy se mohou terče ztratit, protože změna naklopení rukavice způsobí změnu osvětlení terčů a tudíž i jejich barvy. Je proto důležité dodržovat úhel rukavice a kamery stejný jako při kalibraci.

5.4.1 Rotace

Rotace se provádí pouze jedním prstem s červenou barvou (ukazováček). Lze rotovat pohybem ruky nahoru a dolů nebo doleva a doprava.



Obrázek 12: Ukázka rotace

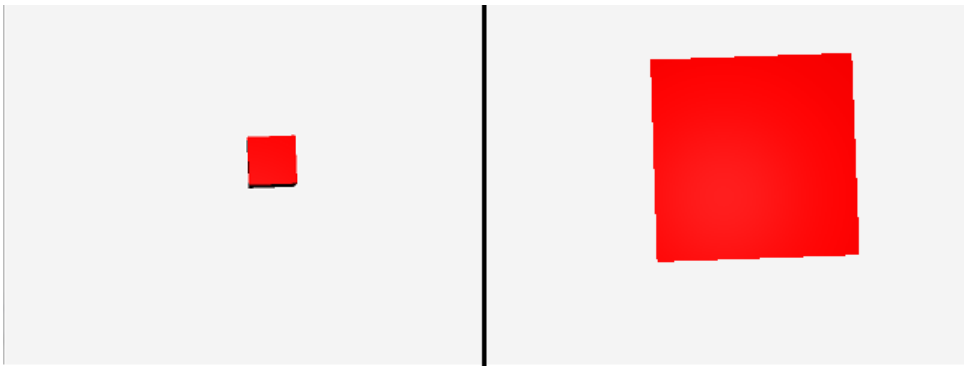
5.4.2 Posunutí

Posunutí se provádí dvěma prsty, červeným (ukazováčkem) a zeleným (prostředníčkem). Pokud nejsou oba prsty ukázané, posun se neprovede. Pro kontrolu sledujte preview. Opět lze posouvat všemi směry.

Obrázek 13: Ukázka posunutí

5.4.3 Změna velikosti

Na změnu velikosti je potřeba použít všechny tři prsty. Směrem doleva po ose X (vodorovně) se objekt zmenšuje. Naopak pohybem směrem doprava se objekt zvětšuje. Opět je dobré kontrolovat preview.



Obrázek 14: Ukázka změny velikosti

Závěr

Cílem práce bylo vytvořit aplikaci, která uživateli umožňuje pohybovat 3D objekty pomocí pohybu ruky před webkamerou s použitím speciálně upravené rukavice. Rukavice měla mít tři barevné terče s červenou, zelenou a modrou barvou. Cíle bakalářské práce byly splněny. Aplikaci by šlo dále vylepšovat a optimalizovat, ale to nebylo původním záměrem. Vyzkoušel jsem si problematiku rozpoznání obrazu a práci s grafikou. Dále popisuje multiplatformní rozhraní OpenGL a práci s ním. Část je i věnována jazyce Java, ve kterém je celá aplikace napsána. Dalšími použitými technologiemi jazyku Java je JavaFX pro tvorbu aplikace a jejího grafického rozhraní nebo Java3D pro práci s 3D objekty, světlem, kamerou a dalšími. Krátce je představeno i vývojové prostředí IntelliJ IDEA, které při vývoji používám. Práce obsahuje pseudokódy použitých algoritmu a představení aplikace spolu s uživatelskou dokumentací.

Conclusions

Main goal of work was to create an application that allows user to move with 3D objects by moving your hand in front of the webcam with a specially modified glove. The Glove have three colored targets with a red, green and blue color. Goals of this bachelor work were fulfilled. Application would be further improved and optimized, but it was not the original intention. I tried the problem of recognition image and graphic work. This thesis also describes the multiplatform interface OpenGL and work with it. One part is also devoted to Java, in which is the whole application written. Other Java technology used is JavaFX for creating application and its graphic interface or Java3D for working with 3Dobjects, light, camera and others. Briefly introduced is the development environment IntelliJIDEA, which I used to develope this application. Thesis includes pseudocodes, algorithms and presentation of application along with the user documentation.

A První příloha

Speciálně upravená rukavice

B Obsah přiloženého CD/DVD

`doc/`
: Text práce ve formátu PDF včetně zdrojových souborů potřebných pro vygenerování PDF dokumentu.

`src/`
: IntelliJ IDEA projekt s kompletní aplikací

Bibliografie

[1] Pavel Herout. *Učebnice jazyka Java*,
České Budějovice 2001, ISBN 80-7232-115-3

[2] *Monitorování procesů a správa paměti v JDK6 a JDK7*. Dostupný z:⟨http://www.root.cz/clanky/monitorovani-procesu-a-sprava-pameti-v-jdk6-a-jdk7-1/#k01⟩

[3] Brett Spell. *Java Programujeme profesionálně*,
1. vydání, Praha 2002, ISBN 80-7226-667-5

[4] *Grafická knihovna OpenGL: základní geometrické prvky*. Dostupný z:⟨http://www.root.cz/clanky/opengl-3-zakladni-geometricke-prvky/#ic=serial-box&icc=text-title⟩

[5] Jiří Žára a kol. *Moderní počítačová grafika*,
2. vydání, Brno 2004, ISBN 80-251-0454-0

[6] Eduard Sojka, Jan Gaura, Michal Krumnikl *Matematické základy digitálního zpracování obrazu*,
Plzeň 2011