FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Úvod do počítačové grafiky KIV/UPG

První odevzdání: pasivní vizualizace
Počet hodin: 31

Pavel Třeštík
A17B0380P

12. května 2021

Obsah

1 Úvod

1.1 Pasivní vizualizace

Cílem první části práce je pasivní vizualizace. Ta se skládá z načtení obrázku ve formátu PGM, jeho zobrazení a vyznačení několika důležitých bodů v tomto obrázku.

Formát PGM uchovává černobílý obrázek. Na začátku souboru obrázku jsou některé základní informace specifikující přesný formát souboru a základní informace obrázku. Po těchto datech následují hodnoty, kde každá hodnota reprezentuje jeden pixel obrázku. Všechny hodnoty (včetně informací na začátku) jsou odděleny libovolným bílým znakem.

Důležitými body jsou minimální a maximální převýšení a maximální stoupání. Za převýšení je v této práci považována absolutní hodnota bodů, tedy nejmenší (resp. největší) hodnota bodu v obrázku. Za maximální stoupání je považován bod, který má největší rozdíl s jedním jeho sousedním bodem.

Dalším požadavkem je, aby se obrázek zvětšoval/ zmenšoval podle velikosti okna a zachoval poměr velikostí stran s načtenými hodnotami.

1.1.1 Formát souboru PGM

Všechny hodnoty v souboru PGM jsou odděleny "bílými znaky".

První hodnotou v souboru je tzv. "magické číslo", které přesněji specifikuje formát PGM. Tato třída umí zpracovat dva typy magických čísel a to "P5"a "P2". Rozdílem mezi těmito čísly je, že hodnoty pixelů jsou brána jako binární data v případě "P5"a jako ASCII čísla v případě "P2".

Další hodnotou je šířka obrázku, následována výškou obrázku. Poslední hodnotou je potom maximální hodnota šedé, která se v obrázku nachází.

Všechny výše uvedené hodnoty jsou ve formátu ASCII a mohou se zde vyskytovat komentáře, které začínají znakem "#". Komentář platí do konce řádky.

Po těchto datech jsou hodnoty jednotlivých pixelů. V případě binárních dat, jsou hodnoty naskládány jedna za druhou bez jakýchkoliv mezer. V případě ASCII hodnot je každá hodnota oddělena od té další bílou mezerou.

1.1.2 Zobrazení obrázku

Zobrazení obrázku je relativně snadné, protože načtené hodnoty pixelů jsou řazeny po řádkách. Stačí tedy projít pole načtených hodnot a převést je na RGB hodnoty, které jsou potom zapsány do obrázku.

Obrázek se má zvětšovat do velikosti okna, se zachováním poměru stran. Změna velikosti má být realizována použitím bilineární interpolace, kterou umí udělat knihovna awt, která je ke kreslení použita, takže není třeba ji implementovat.

1.1.3 Nalezení významných bodů

Pro nalezení minimálního a maximálního převýšení stačí projít hodnoty pixelů obrázku a pouze vybrat nejvyšší a nejmenší hodnotu pixelu.

Maximální stoupání už je složitější. Bod maximálního stoupání, je pixel, který má největší rozdíl s jeho přímým sousedním pixelem. V potaz jsou brány pouze sousední pixely napravo, nalevo, nad a pod zvoleným pixelem. Pro nalezení maximálního stoupání je tedy potřeba projít všechny pixely obrázku a porovnat zrovna procházený pixel s jeho sousedy.

Procházet každý pixel pro nalezení max. stoupání je tedy nutností, ale porovnávat procházený pixel s každým jeho sousedem není úplně optimální. Například pokud jednou spočítám rozdíl pixelu se sousedním pixelem napravo, nepotřebuji znovu počítat rozdíl mezi tímto párem pixelů, potom co se přesunu na tohoto souseda.

Použitý algoritmus tedy vynechá první řádku pixelů a začne od druhé. Poté prochází pixely řádky a dělá rozdíly mezi horním a pravým sousedem, po těchto rozdílech se přesouvá na další pixel. Protože dělá rozdíl s pravým sousedem, nemůže procházet až do posledního pixelu na řádce a proto se z předposledního pixelu řádky přesouvá na další řádku. Po dosažení předposledního pixelu v obrázku končí tato část algoritmu. Protože byla částečně vynechaná první řádka poslední sloupec pixelů, je třeba je dodělat zvlášť. Pro řádku je třeba dodělat rozdíly mezi pravými sousedy a pro sloupec je třeba dodělat rozdíly mezi spodními sousedy. Po dodělání těchto rozdílů je algoritmus ukončen, protože by měly být vypočítány rozdíly mezi všemi přímo-lehlými dvojicemi pixelů a tedy mělo by být nalezeno max. stoupání.

1.1.4 Vykreslení šipek

Protože již vím, kde v obrázku leží významné body, na které šipky ukazují, zbývá pouze určit směr šipky tak, aby stále byla v obrázku. Toho lze dosáhnout poměrně jednoduchým způsobem. Stačí vzít pozici bodu, vzhledem k obrázku a vést šipku ze směru, kde je nejvíce místa. Na příklad pokud významný bod bude v 80% šířky obrázku, šipka bude vedena z pravé strany, protože tímto směrem je 80% šířky volné. Stejným způsobem je udělána výška.

Text šipky ale není brán v potaz a je jednoduše vypsán, tak aby text šiky začínal či končil u počátečního bodu šipky.

Ačkoliv momentálně není řešena kolize šipek, tak by nemělo být příliš složité vyřešit kolizi šipek. Kolize by mohla být řešena například tak, že pokud jsou počáteční body šipek příliš blízko u sebe, tak jeden z bodů bude posunut ve směru nejvíce zbývajícího místa na obrázku.

1.2 Interaktivní vizualizace

1.2.1 Obarvení mapy

Obarvení mapy je velice přímočaré. Udělají se intervaly o velikosti výšky vrstevnic a bod dostane barvu, podle toho ve kterém intervalu se nachází. Barvy jsou získány z lineárního gradientu, který je složen ze tří barev od modré (nízké hodnoty), přes žlutou, do červené (vysoké hodnoty).

1.2.2 Nalezení vrstevnic

Pro nalezení vrstevnic se používá Marching Square algoritmus. Tento algoritmus prochází mřížku hodnot (v tomto případě pixel) čtvercovým okýnkem a dělá čáry v tomto okýnku, podle hodnot bodů na vrcholech čtverce. V této práci se navíc vyskytuje problém, že obrázek se zvětšuje spolu s oknem. Toto značně ztěžuje přesně vykreslit vrstevnice.

1.2.3 Kliknutí do mapy

Po kliknutí do mapy se zobrazí převýšení v bodě kliknutí a zvýrazní se vrstevnice, která je nejblíže svou hodnotou. Pro získání hodnoty převýšení jsou přepočteny souřadnice kliknutí v obrázku na souřadnice v původních datech.

Z původních dat je získána hodnota, která je zobrazena a použita k nalezení nejbližší vrstevnice a to tak, že pokud zbytek po dělení 50 (výška vrstevnice) je menší než 25, jsou zvýrazněny vrstevnice s nejbližší nižší hodnotou pod získaným převýšením. Při zbytku větším než 25 nastává opačný případ a jsou zvýrazněny vrstevnice s nejbližší vyšší hodnotou oproti převýšení.

1.2.4 Statistiky

Statistiky jsou získány z původních dat. Jsou zobrazeny 2 kategorie. Důležité body v obrázku (např.: minimum, maximum, medián) a histogram převýšení. Protože významných bodů není tolik, je rozumné je vypsat hodnotami a navíc přidat graf. Smysluplný graf pro tyto hodnoty je Boxplot, protože tento graf slouží k zobrazení pravě těchto významných bodů. U histogramu je lepší pouze graf, protože hodnot by bylo příliš. Histogram sám o sobě je typ grafu.

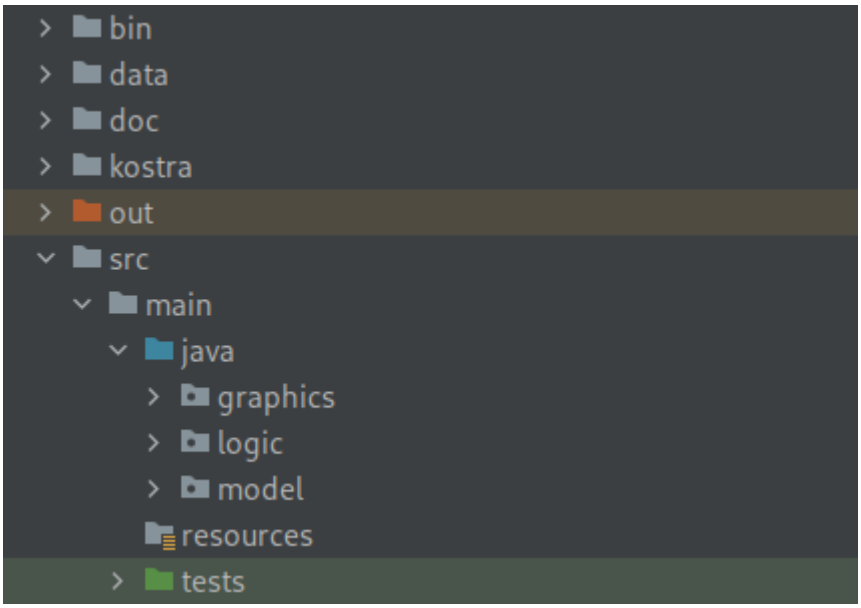
1.2.5 Legenda

Aby uživatel poznal přibližné výšky z grafu, je třeba mu dát vědět, která barva je jaká výška. K tomuto slouží legenda, která zobrazí barvu s popiskem, pro které hodnoty je tato barva použita.

2 Implementace

2.1 Struktura projektu

Projekt nebyl navržen nad žádnou specifickou architekturou. Do určité míry struktura projektu připomíná architekturu MVC, ta ale není striktně dodržena. Na Obrázku 2.1 je vidět adresářová struktura projektu.



Obrázek 2.1: Struktura projektu

2.1.1 Přehled adresářů

Důležitými adresáři jsou **doc** a **src** (viz struktura na Obrázku 2.1).

Adresář **doc** obsahuje dokumentaci k programu.

Adresář **src** obsahuje všechny zdrojové soubory. Podadresář **main/java** obsahuje zdrojové .java soubory, které jsou rozděleny v balíčkách `graphics`, `logic`, `model`.

Další adresáře a podadresáře nebudou s touto prací odevzdávány, tudíž mohou být ignorovány.

2.1.2 Přehled tříd

Package: graphics

Obsahuje dvě tři: **DrawingPane**, **UserInterface** a **ExportDialog**.

Třída **DrawingPane** obstarává kreslící plátno aplikace. Třída **UserInterface** obstarává grafické okno aplikace, ve kterém je umístěno kreslící plátno (instance třídy **DrawingPane**). Třída **ExportDialog** poskytuje dialog pro export mapy do formátu PNG. V dialogu se zadává výška a šířka obrázku.

Package: logic

Obsahuje třídu **PGMLoader**, která načítá PGM soubor. Zdrojový soubor této třídy navíc obsahuje soukromou třídu **IntWrapper**, která má veřejný atribut typu `int` a konstruktor nastavující tento atribut na 0.

V druhé části je zde přidána třída **Contours**, která slouží k vyhledání vrstevnic.

Package: model

Obsahuje třídu **UPGImage**, která uchovává načtená data ze souboru PGM a umožňuje práci s těmito daty. A dále třídu **ContourCell**, která uchovává data o vrstevnicích pro jejich snazší vykreslení.

2.2 Programátorská dokumentace

2.2.1 Třída: PGMLoader

Načítání souboru

Soubor je nejdříve celý načten do paměti. Až z bufferu, do kterého je načten jsou poté získávány informace obrázku. Protože je celý obrázek načítaný do paměti, tak pro prevenci použití příliš velkého množství paměti je velikost vstupního souboru omezena na 50MB.

Soubor je možné načítat průběžně bez bufferu použitím velmi podobného způsobu načítání. Jediným rozdílem by bylo to, že místo procházení bufferu a udržování pozice v něm je procházen a čten stream souboru. Tuto implementaci jsem také zkoušel, ale protože v jednom vlákně byla mnohonásobně

pomalejší než načítání přes paměť, tak jsem tento způsob načítání odebral.

Postup načítání

- `UPGImage loadPGMFileThroughMemory(File f)` - začíná proces načítání souboru. Načte soubor `f` do paměti a poté předává pole typu `byte` ke zpracovávání. Vrací načtená data uložená v instanci třídy `UPGImage`
- `UPGImage parseBuffer(byte[] buffer)` - zpracuje buffer načteného souboru. Vrací instanci třídy `UPGImage` nebo `null` při chybě zpracování
- `UPGImage loadHeaderFromBuffer(byte[] buffer, IntWrapper position)` - první krok ve zpracování bufferu. Z bufferu načte základní informace o obrázku (magické číslo, šířka, výška a max. hodnota šedé). S těmito informacemi vytvoří instanci `UPGImage`, kterou vrací. Do parametru `position` je uložena pozice, kde načítání skončilo a tudíž kde začínají hodnoty pixelů
- `void loadAsciiPixelsFromBuffer(UPGImage img, byte[] buffer, IntWrapper position)` - začne načítat hodnoty pixelů, které interpretuje jako ASCII hodnoty. Načtené hodnoty ukládá do `img`
- `void loadBytePixelsFromBuffer(UPGImage img, byte[] buffer, IntWrapper position)` - začne načítat hodnoty pixelů jako číselné hodnoty v binární podobě. Načtené hodnoty ukládá do `img`
- `String readAsciiFromBuffer(byte[] buffer, IntWrapper position)` - načítá buffer od `position` dokud nenarazí na bílý znak, který je specifikovaný v atributu třídy `byte[] byteWhiteSpaces`. Načtený byty interpretuje jako znaky, ze kterých udělá řetězec (String). Tento řetězec je návratová hodnota metody a do `position` je uložena pozice začátku další hodnoty. Tato metoda je používána pro načítání hodnot pixelů, ale i v `loadHeaderFromBuffer`
- `int skipLineInBuffer(byte[] buffer, int pos)` - pokud `readAsciiFromBuffer` narazí na znak "#", zavolá tuto metodu, která prochází buffer dokud nenarazí na konec řádky ("\n"). Buffer začíná procházet od `pos` a vrací pozici nové řádky

2.2.2 Třída: UPGImage

Třída uchovává data obrázku a poskytuje operace na ním. Třída má řadu atributů. Následuje popis významných metod v této třídě. Třída má, mimo popsané metody, některé typické metody, jako jsou gettery.

- `void createImage()` - tato metoda musí být zavolána na každou novou instanci `UPGImage`. Metoda vytvoří instanci atributu `BufferedImage initialImage`. Metoda musí být volána zvlášť, protože instance `UPGImage` je vytvořena bez hodnot pixelů, proto je tato metoda volána, až po tom co jsou načteny pixely
- `void scaleImage(float scale)` - metoda vytváří instanci atributu `BufferedImage actualImage`, která je zvětšený/ zmenšený obrázek z atributu `initialImage`
- `void findPoints()` - hledá významné body obrázku. Nejdříve se prochází pole pixelů a hledá se minimální a maximální převýšení a poté volá metodu `findDiff()`, která hledá maximální stoupání
- `void findDiff()` - hledá maximální stoupání v obrázku algoritmem popsaným v sekci 1.1.3
- `void sortRasterArray()` - seřadí načtené pole hodnot a nad ním hledá významné body. Těmi jsou průměr, medián a kvartily.
- `JFreeChart createHeightBoxPlot()` - vytvoří Boxplot, který jako data bere načtené hodnoty. Hodnoty, které tento graf vyznačuje, by měly odpovídat průměru, mediánu, kvartilům, minimu a maximu.
- `JFreeChart createHeightHistogram()` - vytvoří Histogram načtených převýšení. Převýšení blízko u sebe průměruje do jedné kategorie.
- `void initContourColors()` - připraví barvy, kterými jsou vybarveny body v různých výškách. Tyto barvy jsou počítány jako linearní gradient tří barev. Začínající na odstínu modré, přes žlutou, do odstínu červené.

2.2.3 Třída: UserInterface

Je to spouštěcí třída aplikace. Třída vytváří velmi jednoduché grafické rozhraní, které obsahuje kreslící plátno. Mimo kreslící plátno obsahuje také menu lištu, která má položku **File**, pod kterou je možnost **Load PGM file**. Tato možnost umožní načíst nový obrázek za běhu aplikace.

Nově také třída vytváří panel legendy ve spodní části okna. Na středu okna je **JTabbedPane**, který má 3 listy. List s kreslícím plátnem. List se statistikami důležitých bodů obrázku a histogram převýšení v obrázku. Nakonec přibyly dvě nové položky menu **File** file a to: **Export map to PNG** a **Export map to SVG**.

2.2.4 Třída: ExportDialog

Pomocná třída k **UserInterface**. Je spouštěna při kliknutí na **Export map to PNG**, kdy spustí nové okno, ve kterém po uživateli požaduje šířku a výšku exportovaného obrázku. Okno momentálně nemá žádné chybné hlášky při zadání chybné hodnoty, nebo nezadání žádné hodnoty. Při nevalidní hodnoty se okno jednoduše zavře.

2.2.5 Třída: DrawingPane

Třída poskytuje kreslící plátno. Plátno je nastaveno na černou barvu, která je viditelná při velikosti okna neodpovídající poměru stran zobrazovaného obrázku. Třída má důležitý atribut `UPGImage img`, ze kterého jsou získávány hodnoty k vizualizaci. Následuje popis metod.

- `void setImg(UPGImage img)` - setter atributu `img`. Také volá `createImage()` tohoto atributu
- `void drawImageScaled(Graphics2D g2)` - změní velikost `actualImage` atributu `img` a potom ho vykreslí. Dále z atributu `img` získá významné body a zavolá vykreslení šipek na tyto body
- `void drawArrowWithText(Graphics2D g2, int x, int y, int width, int height, String text)` - začne proces vykreslení šipky na bod `[x, y]`. V této metodě je vypočítán směr šipky z parametrů `x, y, width, height`. Nakonec je k vypočítanému počátku šipky vykreslen `text`

- `void highlightPoint(Graphics2D g2)` - slouží k vypsání hodnoty bodu, na který uživatel klikl. Přepočte souřadnice do načtených dat pro získání hodnoty. Pro zvýraznění hodnoty je vykreslen jako pozadí čísla bílý obdélník. Na přesné místo kliknutí uživatelem je nakreslena malá černá elipsa.

- `void drawContourSquare(Graphics2D g2, ContourCell c)` - nakreslí řádku, dle typu `ContourCell`. Typ nabývá hodnot 0-15. Přesný obraz co je nakreslen je specifikován Marching Square algoritmem.

- `void drawToImage(BufferedImage target, int w, int h)` - vykreslí mapu do obrázku `target`, jehož šířka je `w` a výška `h`.

- `void drawToSVG(Graphics2D svg)` - vykreslí mapu do plátna `svg`.

2.2.6 Třída: ContourCell

Třída uchovává data každého čtverce, který používá Marching Square algoritmus. Poskytuje gettery a k pár atributem i settery, jinak nemá žádnou funkčnost.

2.2.7 Třída: Contours

Poskytuje většinu logiky k Marching Square algoritmu. Kvůli optimalizaci programu neobsahuje všechny funkce, která by tato třída běžně měla.

- `ContourCell[] getAllCells(UPGImage img)` - vytvoří pole všech čtverců pixelů co se nacházejí ve scalnutém obrazu. Všechny relevantní informace získá z parametru `img` a vytvořené pole vrací jako návratovou hodnotu. Všem `ContourCell` v poli nastaví všechny získatelné informace, ale některé informace musí být dopočítány později.

- `void determineCellTypeAndHeight(ContourCell[] cells, int contourH, int contourI)` - dopočítá chybějící informace do `ContourCell` v poli `cells` na základě parametru `contourH`. V programu je navíc používá index vrstevnice, proto je i ten předán parametrem `contourI`.

2.3 Uživatelská dokumentace

2.3.1 Závislosti

Program je psán v jazyce Java. Byl psán a spouštěn Javou verze 11.0.10, proto uživateli doporučuji použít tuto verzi.

Program zobrazuje grafy a umožňuje uživateli exportovat mapu do formátu SVG. Proto má závislosti na knihovnách JFreeChart (zdroj: `https://www.jfree.org/jfreechart/`) a JFreeSVG (zdroj: `https://www.jfree.org/jfreesvg/`).

2.3.2 Linux

Překlad

V kořenovém adresáři projektu se nachází skript **build.sh**. Tento skript přeloží zdrojové soubory do adresáře **bin**.

Také je možné vygenerovat programátorskou Javadoc spuštěním skriptu **makedoc.sh**. Javadoc je vytvořen do adresáře **doc/javadoc**

Spuštění

V kořenovém adresáři se také nachází skript **run.sh**, který spustí aplikaci. Skript předává parametry spouštěné aplikaci. Aplikace potřebuje právě jeden parametr ke spuštění. Očekávaný parametr je soubor typu PGM.

2.3.3 Windows

Překlad

V kořenovém adresáři projektu se nachází skript **Build.cmd**. Tento skript přeloží zdrojové soubory do adresáře **bin**.

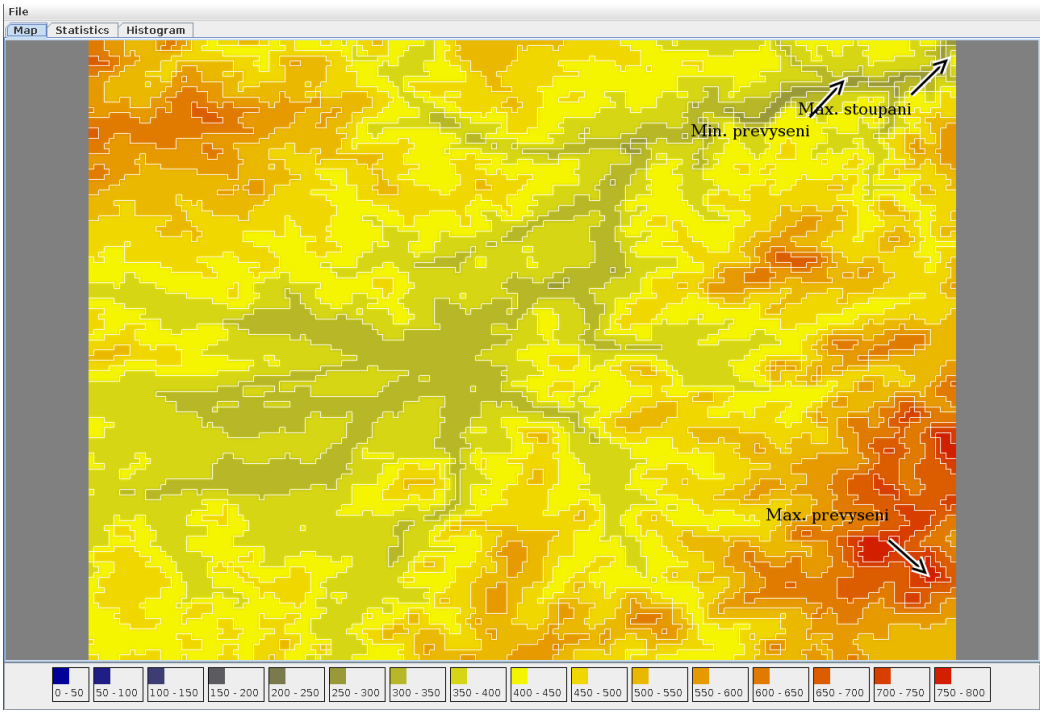
Také je možné vygenerovat programátorskou Javadoc spuštěním skriptu **Makedoc.cmd**. Javadoc je vytvořen do adresáře **doc/javadoc**

Spuštění

V kořenovém adresáři se také nachází skript **Run.cmd**, který spustí aplikaci. Skript předává parametry spouštěné aplikaci. Aplikace potřebuje právě jeden parametr ke spuštění. Očekávaný parametr je soubor typu PGM.

2.3.4 Ovládání programu

Základními funkcemi programu je vizualizace mapy a statistik. Za docílením přehlednosti jsou proto použity záložky. Na následujícím obrázku (Obrázek 2.2) je vidět okno aplikace.



Obrázek 2.2: Okno aplikace

Na záložce mapy, je možné kliknout do obrázku, pro zobrazení hodnoty převýšení v kliknutém bodě a zvýraznění vrstevnic, které jsou nejblíže svou hodnotou k hodnotě tohoto bodu. Při kliknutí do šedé oblasti mimo obrázek je zvýraznění odstraněno. Ve spodní liště je legenda mapy popisující hodnoty použitých barev. Při velkém počtu hodnot, kdy se hodnoty nevejdou do mapy, se ve spodní části okna zobrazí posuvník na šoupání do strany.

Při kliknutí na záložku **Statistics** se zobrazí statistiky v textové podobě na levá straně okna. Na pravé straně se zobrazí Boxplot graf těchto statistik.

Při kliknutí na záložku **Histogram** se zobrazí histogram převýšení.

Program dále nabízí možnost exportovat obrázek mapy z první záložky. Jsou dvě možnosti exportu. První je export do bitmapového obrázku s příponou**.png**.

Druhá je export do vektorové grafiky s příponou **.svg**.

Oba obrázky jsou uloženy do kořenového adresáře projektu. Bitmapový obrázek je ukládán jako **raster_map.png** a vektorový jako **vector_map.png**.

2.4 Závěr

Bohužel jsem na práci neměl moc čas a navíc jsem jí odkládal na poslední chvíli a to se trochu neblaze projevilo. Ačkoliv se na práci najde hodně věcí, co by se daly vylepšit, základní funkčnost by měla být splněna. Jednou z nejnáročnějších akcí v programu je hledání a především vykreslování vrstevnic. Tyto operace mohou trvat několik vteřin. Na velmi limitovaný čas, co jsem na práci strávil, věřím, že se mi povedlo poměrně slušně optimalizovat hledání a uchovávání vrstevnic. Problémem ale je jejich vykreslení a opakované hledání. Pro co největší přesnost jsou vrstevnice hledány ve scalenutém obrázku, tudíž vrstevnice jsou hledány znova při každém zvětšení či zmenšení okna. Toto by šlo omezit například přidáním posluchače na změnu velikosti okna, který zamezí počítání vrstevnic do doby, než je okno ustáleno. Ovšem s dlouho trvajícím vykreslením je těžší něco udělat a nevím, jaké jsou v tomto směru možnosti.

Mimochodem, ačkoliv aplikace umožňuje načtení nového PGM souboru za běhu z uživatelského rozhraní, je na zvážení, zda tuto funkci zachovat. Důvodem je, že načtení více než jednoho obrázku skrz tuto možnost, na až několik minut zasekne celý program na akci odstranění legendy, aby mohla být vygenerována nová. Toto je údajně způsobeno layoutem a použitím jiného layoutu by se měl tento problém vyřešit.