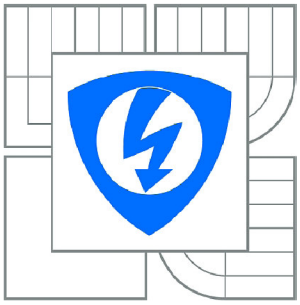
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF BIOMEDICAL ENGINEERING

INTERAKTIVNÍ JAVA APPLET PRO 3D VIZUALIZACI OPTICKÉHO DISKU OKA

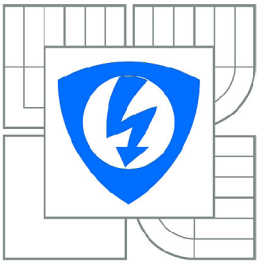
INTERACTIVE JAVA APPLET USED FOR 3D OPTIC NERVE HEAD VIZUALIZATION

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — Bc. JAROSLAV ŠIKL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ROMAN PETER
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav biomedicínského inženýrství

Diplomová práce

magisterský navazující studijní obor
Biomedicínské a ekologické inženýrství

Student:	Bc. Jaroslav Šikl	***ID:***	80461
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Interaktivní Java applet pro 3D vizualizaci optického disku oka

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vyšetřovací metodou konfokální laserové skenovací oftalmoskopie. Navrhněte a realizujte Java applet pro 3D vizualizaci topografie optického disku oka s vhodnou mírou uživatelské interaktivity. Software otestujte na dostupných topografických datech pořízených přístrojem HRT. Práce musí obsahovat technickou a uživatelskou dokumentaci.

DOPORUČENÁ LITERATURA:

[1] Ciulla, T.A., Rwgillo, C.D. Retina and Optic Nerve Imaging, Lippincott Williams & Wilkins, 2003
[2] Iester, M. Optical Nerve Head and Retinal Nerve Fibre Analysis, European Glaucoma Society, 2005
[3] Daniel Selman, Java 3D Programming, Manning Publications, February 2002

Termín zadání:	12.10.2009	***Termín odevzdání:***	21.5.2010

Vedoucí práce: Ing. Roman Peter

prof. Ing. Jiří Jan, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	Jaroslav Šikl
Bytem:	Okružní 1945/32, Žďár nad Sázavou 3, 591 01
Narozen/a (datum a místo):	28.4.1971 v Novém Městě na Moravě

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Ing. Jiří Jan,Csc., předseda rady oboru Biomedicínské a ekologické inženýrství

(dále jen "nabyvatel")

Čl. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

☐ disertační práce

☐ diplomová práce

☐ bakalářská práce

☐ jiná práce, jejíž druh je specifikován jako .....
(dále jen VŠKP nebo dílo)

Název VŠKP:	Interaktivní Java applet pro 3D vizualizaci optického disku oka
Vedoucí/ školitel VŠKP:	Ing. Roman Peter
Ústav:	Ústav biomedicínského inženýrství
Datum obhajoby VŠKP:	7. nebo 8. června 2010 *

VŠKP odevzdal autor nabyvateli *:

☐ v tištěné formě – počet exemplářů 2

☐ v elektronické formě – počet exemplářů 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění..

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   ☐ ihned po uzavření této smlouvy

   ☐ 1 rok po uzavření této smlouvy

   ☐ 3 roky po uzavření této smlouvy

   ☐ 5 let po uzavření této smlouvy

   ☐ 10 let po uzavření této smlouvy
   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….….. ……

Nabyvatel Autor

ANOTACE

Tato diplomová práce se zabývá návrhem a realizací interaktivního Java appletu pro 3D vizualizaci topografie optického disku oka. Primárním účelem 3D vizualizace je oftalmologie a to především při diagnostice zeleného zákalu – glaukomu. Program může sekundárně sloužit i jako názorná pomůcka, pomocí níž je možné prakticky se seznamovat s transformací 2D obrazu na 3D model a s vlivem úprav a nastavování jednotlivých parametrů na vykreslování tohoto modelu ve 3D prostoru.

V práci jsou navrženy dvě verze appletu, které byly vytvořeny ve vývojovém prostředí NetBeans verze 5.5, a jejich finální realizace je provedena jako aplikace typu .html, pracující v okně webového prohlížeče s vhodnou mírou uživatelské interaktivity. První verzí realizovaného appletu je jednoduchá intuitivní aplikace, v níž jsou přednastaveny některé parametry zobrazení tak, že po načtení obrázku je automaticky vykreslen 3D model. Applet následně dovoluje nastavovat u tohoto zobrazení velikost, 3D hloubku zobrazení, vyhlazení hran povrchu a možnost volby zobrazení šedotónového, nebo barevného obrázku. Druhá verze realizovaného appletu je rozšířením prvního appletu o možnost zobrazení os *x*, *y*, *z* ve 3D prostoru, vykreslování povrchu 3D modelu pomocí jednotlivých bodů v prostoru, čar nebo sítě, nebo osvětlení jeho povrchu.

Software byl otestován na dostupných topografických datech pořízených přístrojem HRT a součástí jsou tedy i ukázky dosažených výsledků. Práce obsahuje také technickou a uživatelskou dokumentaci.

Klíčová slova: Java, applet, 3D vizualizace, optický disk oka.

ABSTRACT

The aim of this thesis was to design and implement the interactive Java applet used for topography 3D optic nerve head visualization. Primary purpose of the 3D visualization is ophthalmology, especially for diagnosis of glaucoma. More over should serve as a training material, which enables to study the transformation of 2D figures to 3D model and to test the effects of adjustments and regulations to imaging of the model in 3D space.

Two program versions were designed in this work using the development system NetBeans version 5.5. Their final realization is implemented as .html application working in web browser window. First version of the designed applet is simple intuitive application with several default settings, so that the figure is opened automatically to 3D model. Subsequently, the applet enables to set the size of the image, 3D depth of the image, smoothing of the surface and selection of monochrome or coloured image. The second version is the extension of the first version of the designed applet, and provides the possibility to display *x*, *y*, *z* axis, depiction of the 3D model surface by dots, curves or grid, and illumination of the surface.

Software was tested on available topographic data acquired by HRT equipment and in different web browsers. Technical documentation and user's manual are also involved in this thesis.

Keywords: Java, applet, 3D visualization, optic nerve head.

BIBLIOGRAFICKÁ CITACE

ŠIKL, J. *Interaktivní Java applet pro 3D vizualizaci optického disku oka*: *diplomová práce*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 52 s. 6 příl. Vedoucí diplomové práce Ing. Roman Peter.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Interaktivní Java applet pro 3D vizualizaci optického disku oka jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 20. května 2010 ……

podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Romanu Peterovi za velmi užitečnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce. Také děkuji manželce Michaele a dceři Lauře za podporu během celého mého studia.

V Brně dne …… …….

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 Úvod

Tato práce se zabývá návrhem interaktivního Java appletu, pomocí něhož bude možné provést 3D vizualizaci optického disku oka s využitím konfokální laserové skenovací oftalmoskopie. Navrhovaný Java applet bude zpracovávat topografické obrázky, které jsou získány z Heidelbergova sítnicového tomografu – HRT (Heidelberg Retina Tomograph) převedením raw dat do formátu PNG. Na základě těchto obrazů pak bude zobrazovat 3D vizualizaci scény v okolí výstupu očního nervu na sítnici.

Primárním účelem 3D vizualizace je oftalmologie a to především při diagnostice zeleného zákalu – glaukomu, kde 3D zobrazení optického disku oka může zřetelně ukázat rozdíl mezi miskovitým tvarem normálního oka a glaukomatózního oka, kde je pohár mnohem hlubší a vykazuje vyšší strmost podél okrajů. Program může sekundárně sloužit i jako názorná pomůcka , která by mohla být využita k seznámení s transformací 2D obrazu na 3D model a s vlivem změn jednotlivých parametrů na vykreslování modelu ve 3D prostoru.

Glaukom je velmi závažné onemocnění probíhající pomalu a nepozorovaně. Většina pacientů nepozoruje žádné příznaky do doby, než se jim zhorší vidění. To už je ovšem glaukom ve stavu velmi pokročilém a ztráta vidění je nevratná. Glaukom je spolu s věkem podmíněnou degenerací sítnice jednou z nejčastějších příčin trvalé slepoty u starších lidí. V České republice postihuje 2% obyvatel [6].

Návrh programu pro 3D vizualizaci má být koncipován tak, aby použití Java appletu bylo interaktivní a předpokládá se také jeho umístění do webového rozhraní, ke kterému se mohou připojit počítače postavené na různých systémových platformách. Program by tedy měl být systémově nezávislý. Z tohoto pohledu se pro tento účel jeví použití Java appletu, který lze vytvořit pomocí programovacího jazyka Java, jako nejvýhodnější, i když základní knihovny programovacího jazyka Java v současnosti neposkytují efektivní nástroje pro trojrozměrnou vizualizaci a bude tedy nutné překonat některé problémy z toho plynoucí.

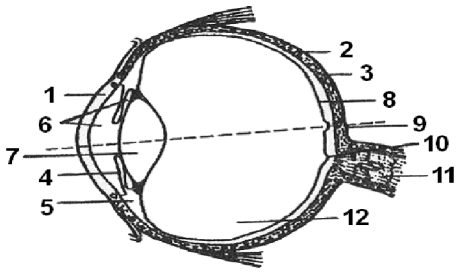
2 Současný stav řešené problematiky

Diagnostika glaukomu, zvláště v jeho časných stadiích, patří k obtížným problémům očního lékařství. Zvláště u těžké krátkozrakosti, kde je frekvence glaukomu častější, je jeho rozpoznání ještě obtížnější, protože terč zrakového nervu mívá atypický tvar, nitrooční tlak může být při rutinním použití Schiotzova tonometru naměřen falešně nízký a v zorném poli bývá obtížné odlišit od sebe projevy degenerativních změn chorioretinálních od změn glaukomových.

2.1 Anatomie oka

Světlo je viditelnou částí spektra elektromagnetického záření v rozmezí 400 – 700 nm [4]. Působí na biochemické a fyziologické funkce organismu a také na psychický stav. Zrak je jedním z nejcitlivějších smyslových orgánů lidského organismu. Pro zelenou barvu světla je práh citlivosti 5 $\mu W/m^2$ [4]. Pro vnímání jasu je rozhodující jeho časové a prostorové rozložení na sítnici – prahový jas je 1 $\mu cd/m^2$ (5 – 15 světelných kvant) [4]. Prostřednictvím zraku získává člověk téměř 80 % všech informací z okolního prostředí [4]. Fyzikální podstata funkce oka jako optické zobrazovací soustavy je známá, ovšem zpracování obrazu a jeho přenos do center vidění v mozku je stále náplní základního výzkumu mnoha vědeckých pracovišť [5]. Lékařský obor, který se zabývá nemocemi oka i jeho přídavných orgánů (tj. slzným systémem, očními víčky, atd.), a také chirurgií zrakových drah, které zahrnují kromě oka mozek a oblasti okolo mozku, se nazývá oftalmologie.

Ke zrakovému ústrojí patří také přídavné oční orgány. Těmi jsou okohybné svaly, které zajišťují postavení a pohyby očních bulbů, dále víčka, spojivky (conjunctiva) a slzný aparát, které chrání oko před vysycháním, poškozením a záněty. Zevně pod stropem očnice je umístěna slzná žláza (glandula lacrimalis). Ta produkuje slzy, které odtékají slznými kanálky ve vnitřním koutku, do dutiny nosní. Základní stavba oka je na obr.2.1. Oko je složeno z těchto částí [3]: 1. rohovka – cornea, 2. bělima – sclera, 3. cévnatka – chorioidea, 4. duhovka – iris, 5. řasnaté tělísko – corpus ciliare, 6. přední a zadní komora oční, 7. čočka – lens, 8. sítnice – retina, 9. žlutá skvrna – fovea centralis (místo nejostřejšího vidění, kde je největší hustota čípků), 10. slepá skvrna (výstup zrakového nervu, kde nejsou tyčinky a čípky), 11. zrakový nerv – nervus opticus, 12. sklivec – corpus vitreum.



***Obr.2.1**: Stavba oka [3].*

Paprsky světla procházejí rohovkou, komorovou vodou, čočkou, sklivcem a dopadají na zadní část oční koule pokrytou blánou, která se nazývá sítnice (retina). Sítnice se skládá z elementů citlivých na světlo, které jsou dvojího druhu a to: tyčinky a čípky. Podrážděním tyčinek a čípků (pro barvu modrou, zelenou a červenou) vzniká černobílé a barevné vidění. Barevný vjem mohou zprostředkovat pouze čípky (fotopické vidění), tyčinkami lze vnímat pouze rozdíly jasů (skotopické vidění). Při dostatečné hladině osvětlení převládá vnímání čípky a zrakový vjem je barevný. Při velmi slabém osvětlení vnímá oko tyčinkami, Ty nejsou citlivé na barvu, proto je zrakový vjem pouze v odstínech šedi. Místem nejostřejšího vidění je žlutá skvrna (fovea centralis), kde jsou nahromaděny čípky. Existují tři typy čípků s citlivostí na vlnové délky odpovídající modré, zelené a červené barvě. Různá intenzita podráždění všech čípků vyvolá vnímání celé barevné škály, které je založeno na principu míšení barev.

2.2 Glaukom

Glaukom je generický termín pro specifickou oční chorobu, která má mnoho příčin a různé klinické projevy. *Glaukos* v souladu s "Etymologickým slovníkem řeckého jazyka", jehož autorem je Chantraine [20], znamená „barva moře“. *Glaucoma* vyjadřuje tedy stav oka, jehož zornice má nazelenalý odraz mořské vody. Původně tedy termín *glaukoma* neoznačoval určitou oční chorobu, ale pouze charakteristický vzhled zornice, jenž je běžný pro odlišení dalších patologických entit, zvláště katarakty. V konečném stádiu glaukomu je totiž rohovka (přední průhledná stěna oka) šedozeleně zakalená a duhovka za ní je ztenčená a světle zelená. Oko má tedy nazelenalý vzhled. Podstatou onemocnění ale není zakalení nějaké struktury uvnitř oka. Podstatou glaukomu je zvýšení tlaku tekutiny uvnitř oka (nitroočního tlaku). Vysoký nitrooční tlak utlačuje vlákna zrakového nervu. Při dlouhodobém působení vysokého nitroočního tlaku dochází k odumírání těchto vláken. K poškození přispívá také zhoršené prokrvení zrakového nervu. U některých glaukomů hraje roli také rodinná anamnéza. Někdy může být glaukom vyvolán dlouhodobým užíváním kortikosteroidních léků nebo kapek. Důvody vysokého nitroočního tlaku jsou dva. Jednou příčinou může být nadměrná tvorba nitrooční tekutiny. Druhým důvodem může být nedostatečná průchodnost odtokových kanálků, které jsou umístěné v úhlu mezi rohovkou a duhovkou (v tzv. komorovém úhlu).

Glaukom je jeden z nejzávažnějších problémů oftalmologie a patří na čelní místo mezi příčinami slepoty, na které se podílí v celosvětovém měřítku 13 % [20]. Přibližně 1,5 až 2 % populace nad 40 let věku má glaukomové poškození zrakového nervu spojené se ztrátou vidění a zorného pole [20]. Výskyt stoupá s přibývajícím věkem až na 3,5 % u osob ve věku 70 až 75 let [20]. Je známa také řada typů glaukomů, které mohou postihovat i generaci mladší. Tím pádem se glaukom stává problémem celospolečenským.

V literatuře zabývající se glaukomem [5] se uvádí, že ve své podstatě termín „glaukom“ zahrnuje široké spektrum nemocí. Bohužel tento termín není používán jednotně. Některé knihy, zejména z kontinentální Evropy [5], definují glaukom jako skupinu stavů s jedním společným znakem – zvýšeným nitroočním tlakem. Jiné definice popisují glaukom výlučně jako stav, kdy dochází k poškození zrakového nervu a ztrátě vidění. V běžné oftalmologické praxi se ukázalo jako užitečné používat termín „glaukom“ jak u všech pacientů se zvýšeným nitroočním tlakem, tak i u všech pacientů s glaukomovým poškozením zrakového nervu.

2.2.1 Typy glaukomu

V literatuře se obvykle používá tato klasifikace glaukomu [20]:

1. Primární glaukom: (prvotní) – je diagnostikován bez předchozího jiného onemocnění. Často je neznámého původu. Dále se dělí na:

- Glaukom s uzavřeným komorovým úhlem (Glaucoma anguláre) – u tohoto typu glaukomu (angulárního) je úhel mezi duhovkou a rohovkou velmi úzký, kořen duhovky může zablokovat oblast odvodného kanálku. Tím dojde k uzavření odtoku nitrooční tekutiny. Tlak oka je výrazně zvýšen, oko je překrvené. Dochází k záchvatovitým pocitům mlhavého vidění a bolestem oka, které mohou vyústit až v akutní glaukomový záchvat. Dále se obvykle ještě dělí na:

  - glaucoma intermitens *(subacutum),*
  - glaucoma acutum,
  - glaucoma chronicum.

- Glaukom s otevřeným komorovým úhlem (Glaucoma chronicum simplex) – probíhá nejčastěji jako tzv. chronický prostý glaukom. Odtok z oka je zhoršen z různých příčin, ale komorový úhel, kde se nachází odvodný kanálek, je volný.

- Smíšená forma – je kombinací obou předchozích typů.

2. Sekundární glaukom: (druhotný) – je nejčastěji zjištěn jako následek zvýšení nitroočního tlaku. Bývá způsoben např. postižením samotného oka zánětem, úrazem, nebo jiným systémovým onemocněním. Dále se dělí opět na:

- Glaukom s otevřeným úhlem - který je dále kvalifikován podle příčin na:
  - steroidní glaukom
  - pseudoexfoliativní glaukom
  - pigmentový glaukom
  - po alfachymotripsinu
  - neovaskulární, hemorhagický glaukom
  - fakolytický glaukom
  - pooperační
  - epitheliální invaze
  - Fuchsova endoth. dystrofie
  - retinopathia pigmentosa
  - heterochromická iridocyclitis
  - glaukomatocyklitická krize - Possner-Schlossman
  - Cleft syndrom po traumatu
  - hyphaema
  - myopia gravis degenerativa
  - sferofakie
  - pulsující exofthalmus
  - esenciální atrofie duhovky
  - herpetické keratitidy
- Glaukom s uzavřeným úhlem - který je dále kvalifikován podle příčin na:
  - miotiky indukovaný, inverzní
  - pozánětlivé synechie - iris bombata
  - změna polohy čočky
  - intumescentní katarakta
  - pseudofakie
  - neovaskulární glaukom
  - iridokorneální endoteliální syndromy
  - tumory, cysty
  - stavy po úrazech, poleptáních

3. Kongenitální glaukom: – (vrozený) je poměrně vzácný a zjistí se při narození, nebo krátce po narození. Je způsoben nedostatečným vývojem v komorovém úhlu, jehož důsledkem je ztížení odtoku nitrooční tekutiny.

- Infantilní, kongenitální glaukom - hydrophtalmus
- Spojený s kongenitálními anomáliemi

2.2.2 Diagnostika glaukomu

Obecně platí, že primární glaukom je geneticky podmíněn. Nejsou sice známé přesné dědičné faktory, ale je znám rodinný charakter onemocnění. Přestože primární glaukom s otevřeným úhlem nemá žádné klinické projevy ve svých počátcích, je třeba věnovat pozornost i nespecifikovaným vizuálním symptomům vyšetřovaného – tj. pocit diskomfortu, horší vidění při slabším osvětlení, potíže při čtení. Na základě anamnézy je pak pozornost soustředěna na možné rizikové faktory [20]:

- vysoká myopie,
- diabetes mellitus,
- systémová hypertenze či hypotenze,
- rodinný výskyt,
- předchozí trauma (důležité pro vznik sekundárního glaukomu).

Zvýšený nitrooční tlak není konstantním nálezem u všech typů glaukomů, jak se dříve uvádělo, ale je pouze jedním z rizikových faktorů. Nitrooční tlak je tedy jasnou příčinou poškození u glaukomu kongenitálního, chronického glaukomu s uzavřeným komorovým úhlem a dalších glaukomů sekundárních. Jestliže je u pacienta naměřena vyšší hodnota nitroočního tlaku (nejlépe aplanační tonometrií), je pacient pozván k opakovanému měření tlaku v různých denních dobách a na základě výsledku se stanoví průměrná výše nitroočního tlaku. Je-li na základě předchozích vyšetření u nemocného diagnostikován glaukom, zahajuje se léčba. V současné době se klade důraz na co nejčasnější diagnostiku glaukomového postižení s následným optimálním terapeutickým postupem – jedině tak lze předejít postupnému poklesu zrakových funkcí.

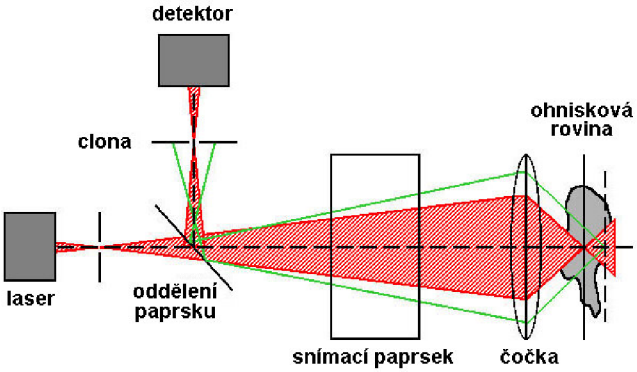
S výjimkou chirurgické léčby lze v našich podmínkách přesunout celou problematiku včasné diagnózy a léčby medikamentózní i laserové do ambulantních provozů, tak jak je tomu ve vyspělých zemích světa [9]. Vyžaduje to však dokonalé přístrojové vybavení a vysoké odborné kvality a zkušenosti oftalmologa-glaukomatologa, na kterém leží v takovém případě daleko větší míra odpovědnosti. Zcela typickým způsobem na sebe upozorní akutní glaukomový záchvat – glaucoma angulare acutum, který pokud je rozpoznán, správně diagnostikován a okamžitě řešen laserovou iridotomií, může být pouze jediným v projevu takovéhoto typu glaukomu. Akutní glaukomový záchvat patří ke stavům vyžadujícím neodkladnou lékařskou pomoc, proto musí být pacient při podezření na záchvat akutního glaukomu okamžitě odeslán k oftalmologovi.

Nejobtížnějším typem glaukomu pro stanovení diagnózy je primární glaukom s otevřeným komorovým úhlem. Tento typ glaukomu představuje 2/3 z celkového výskytu glaukomů [5]. Čím dříve je tento typ glaukomu (glaucoma chronicum simplex) diagnostikován a čím menší či žádné funkční změny jsou přítomny, tím větší je pravděpodobnost kompenzace nitroočního tlaku monoterapií. V oftalmologické praxi se lze nejčastěji setkat s těmito diagnostickými metodami:

- **optická koherentní tomografie** (OCT), jenž dovoluje „histologii“ a biomorfometrii sítnice, čímž je velmi přínosná v oblasti papily, např. při diagnostice glaukomu a akutních i chronických afekcí centrální krajiny [10],
- **Heidelbergův retinální tomograf** (HRT), který se používá především k vyšetření oblasti makuly a terče zrakového nervu [7],
- **laserová skenovací polarimetrie (glaukomový analyzátor)** (GDx) umožňující detekovat změny tloušťky RNFL v peripapilární oblasti terče zrakového nervu a uplatňující se tedy především v diagnostice a sledování pacientů s glaukomem stejně tak, jako Retina Thickness Analyzer zobrazující topografii sítnice a terče zrakového nervu a měřící přímo tloušťku sítnice. [10]

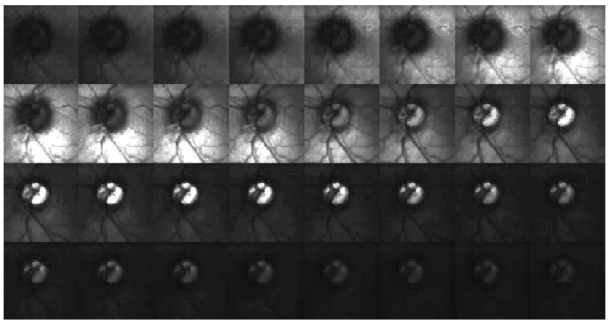
2.3 Konfokální laserová skenovací oftalmoskopie

V systému laserového snímání je laserový paprsek soustředěný na jeden bod zkoumaného objektu, světlo odražené z toho bodu se vrací stejným způsobem skrz optiku, je odděleno od dopadajícího laserového paprsku, a je odkloněné k detektoru. To dovoluje měřit odražené světlo jen v jednom jednotlivém bodu objektu. K tomu, aby vznikl dvojrozměrný obraz, je osvětlovací laserový paprsek vychylován pravidelně ve dvou rozměrech kolmých k optické ose pomocí skenovacích zrcadel.



Obr.2.2: *Konfokální optická soustava [8].*

V konfokální optické soustavě (obr.2.2) je umístěná malá clona před detektorem v místě, které je opticky spřažené s ohniskovou rovinou systému osvětlení. Světlo, které je odražené z objektu v ohniskové rovině, je soustředěno do otvoru, projde jím a je detekováno. Ovšem, světlo odražené z vrstev trojrozměrného objektu nad nebo pod ohniskovou rovinou není zaměřené do otvoru, a jen malý zlomek ho může projít a je detekován. Proto je mimo-ohniskové světlo vysoce potlačené a potlačení se zvyšuje velmi rychle se vzdáleností od ohniskové roviny. V důsledku to znamená, že konfokální systém laserového snímání má vysoké optické rozlišení. Tento systém také umožňuje skutečné trojrozměrné zobrazování. Dvojrozměrný obraz je optický úsek v ohniskové rovině a když pohybujeme ohniskovou rovinou a získáváme obrazy v různých hloubkách, získáme sérii obrazů optických úseků, které tvoří vrstvu po vrstvě trojrozměrný obraz trojrozměrného objektu. Tato procedura je známa jako laserová skenovací tomografie [8]. Obr.2.3 ukazuje vrstvu po vrstvě trojrozměrný obraz zrakového nervu. Tato série se sestává z 32 konfokálních řezů obrazem, v různých ohniskových rovinách.



Obr.2.3*: Obraz zrakového nervu - ve vrstvách [8]*

2.3.1 Princip Heidelbergova sítnicového tomografu

Heidelbergův sítnicový tomograf (obr.2.4). – HRT (Heidelberg retina tomograph) je konfokální systém laserového snímání určený pro snímání a analýzu trojrozměrných obrazů očního pozadí Umožňuje kvantitativní měření topografie očních struktur a sledování topografických změn [1, 2]. V klinické praxi je HRT používán od roku 1990, typ HRT II je k dispozici od roku 1999. HRT III je nejmodernější verze, která umožňuje dlouhodobé sledování a porovnávání výsledků. Nejdůležitější technické parametry uváděné v literatuře pro jednotlivé typy HRT, jsou v tab.2.1 [21].

Obr.2.4: *Heidelbergův sítnicový tomograf [8].*

Tab.2.1: *Porovnání parametrů HRT, HRT II a HRT III [21].*

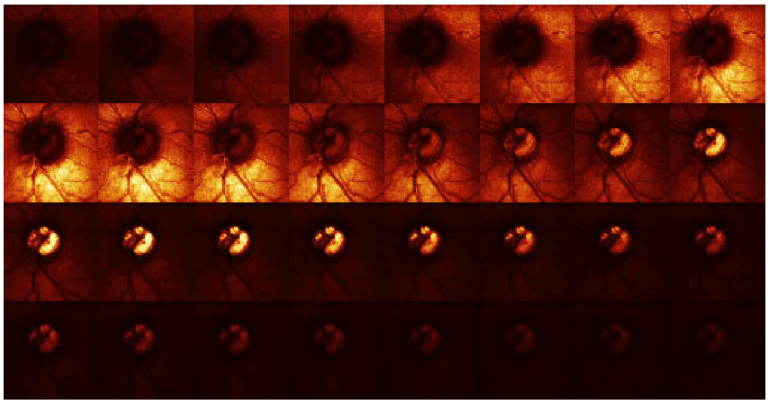
		HRT	**HRT II**	**HRT III**
Velikost zorného pole	Transversální	10° x 10° 15° x 15° 20° x 20°	15° x 15°	15° x 15°
	Longitudinální	0.5 až 4.0 mm	1.0 až 4.0 mm	1.0 až 4.0 mm
Velikost rámce digitálního obrazu	2-D obraz	256 x 256 pixelů	384 x 384 pixelů	384 x 384 pixelů
	3-D obraz	256 x 256 x 32 voxelů	384 x 384 x 16 až 384 x 384 x 64 voxelů	384 x 384 x 64 voxelů
Celkový akviziční čas	2-D obraz	0.032 sec	0.025 sec	0.024 sec
	3-D obraz	1.4 sec	1.0 sec	1.0 sec
Rozsah zaostření		-12 až +12 dioptrií	-12 až +12 dioptrií	-12 až +12 dioptrií (sférické) -6 až +6 dioptrií (cylindrické)
Optické rozlišení	Transversální	10 μm	10 μm	10 μm
	Longitudinální	300 μm	300 μm	300 μm
Digitální rozlišení	Transversální	10 až 20 μm/pixel	10 μm/pixel	10 μm/pixel
	Longitudinální	62 až 128 μm/plochu	62 μm/plochu	62 μm/plochu
Světelný zdroj		Diodový laser, 675 nm	Diodový laser, 670 nm	Diodový laser, 670 nm

HRT se nejčastěji používá ke kvantitativnímu stanovení topografie sítnice a kvantifikaci topografických změn. Např. pro popis výstupu zrakového nervu, analýzu prohlubní způsobujících skvrny na sítnici, nebo analýzu defektů vrstev nervového vlákna. Při zeleném zákalu dochází ke ztrátě vláken zrakového nervu, která způsobuje změny v trojrozměrné topografii výstupu zrakového nervu na sítnici, přičemž při této ztrátě nemusí roky docházet k žádným subjektivním vjemům při zhoršování periferního vidění (obr.2.5).

Obr.2.5: *Normální pohled (vlevo) a pohled člověka s rozvinutým glaukomem (vpravo) [18].*

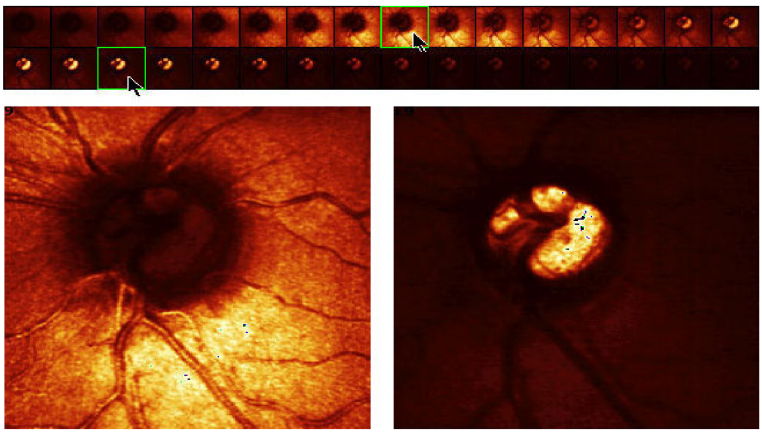
2.3.2 Výstupní obrazy z Heidelbergova sítnicového tomografu

Obr.2.6 ukazuje vrstvu po vrstvě trojrozměrný obraz zrakového nervu. Jedná se o sérii pseudo-zabarvených snímků z přístroje HRT II. Tato série se sestává ze 32 konfokálních řezů obrazem, v různých ohniskových rovinách. Zorné pole je v tomto příkladu 15° [8]. Série začíná ohniskovou rovinou lokalizovanou ve sklivci. Celý obraz se zdá tmavý, protože všechny struktury jsou mimo ohnisko. Jak se ohnisková rovina přesunuje dozadu, sítnice se stává jasnější. Nejjasnější se stane když je ohnisková rovina umístěna na jejím povrchu. Jakmile je ohnisková rovina přesunuta více dozadu, sítnice se dostává mimo ohnisko, stane se temnější a místo toho se rozjasní spodní část oční dutiny. Když se ohnisková rovina přesune za spodní část dutiny, celý obraz opět ztmavne. Celkový rozsah této série obrazů do hloubky, tzn. vzdálenost mezi prvním a posledním obrazem, je 2,5 mm [8]. To znamená, že vzdálenost ohniskové roviny mezi každými dvěma následujícími obrazy je asi 80 μm.



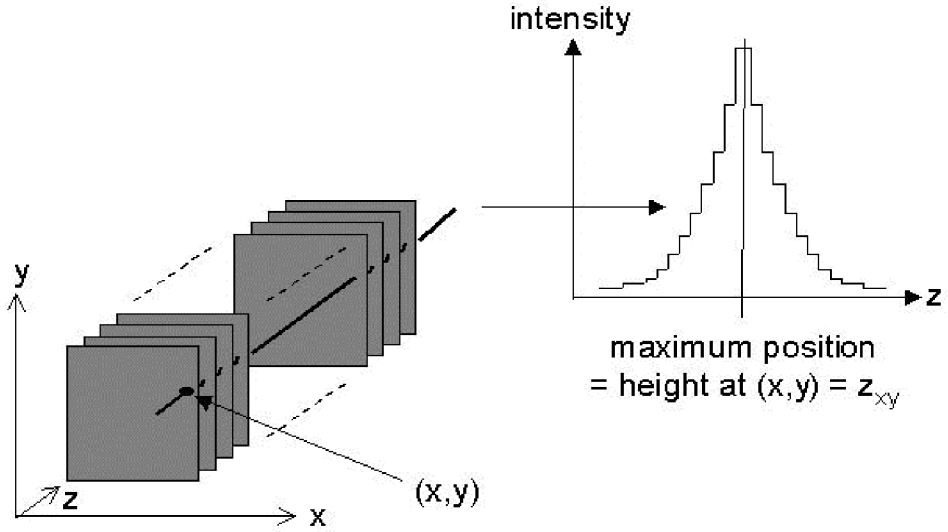
Obr.2.6: *Trojrozměrný obraz zrakového nervu - ve vrstvách [8].*

Na obr.2.7, jsou v plném rozsahu zobrazeny dva jednotlivé sekční obrazy ze série 32 pseudo-zabarvených snímků z přístroje HRT II. z obr.2.6. V levém snímku na obr.2.7 je ohnisková rovina umístěna na povrchu sítnice. Sítnice se jeví jasně, zatímco výstupní jamka nervu je tmavá. Ve snímku na pravé straně, kde je ohnisková rovina umístěna 0,8 mm více vzadu, je sítnice tmavá a spodní část výstupní jamky nervu se jeví jasně [8]. To dává dobrou představu o trojrozměrné informaci obsažené v takových konfokálních obrazových sériích.



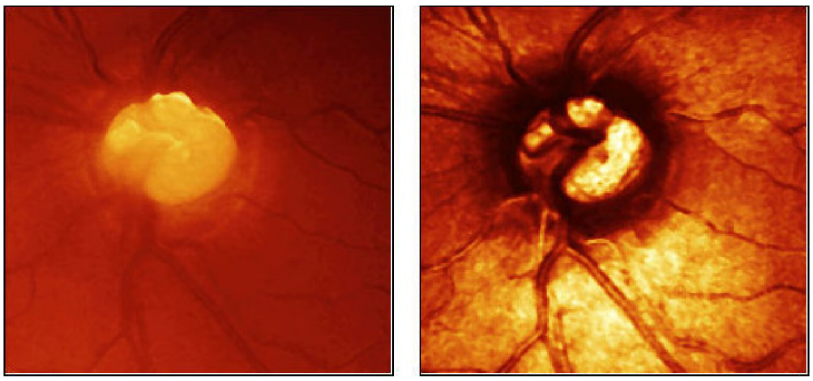
Obr.2.7: *Jednotlivé sekční obrazy ze série [8].*

Vrstvený trojrozměrný obraz je používán pro výpočet topografie povrchu odrážejícího světlo. Pro každé umístění (x,y) v obrazových rovinách, obsahuje série distribuci odražené světelné intenzity podél optické osy, tedy osy z. Tato distribuce intensity se nazývá konfokální *z-profil*. Konfokální *z-profil* je symetrická distribuce s maximem v místě povrchu který odráží světlo. Kvůli konfokálnímu potlačení klesá rychle měřená intensita se zvyšující se vzdáleností od povrchu. Proto je stanovením pozice maxima *z-profilu* možno určit umístění odrážející se plochy podél osy z, jímž je právě vrchol *z-profilu* (obr.2.8).



Obr.2.8: *Konfokální z-profil [8].*

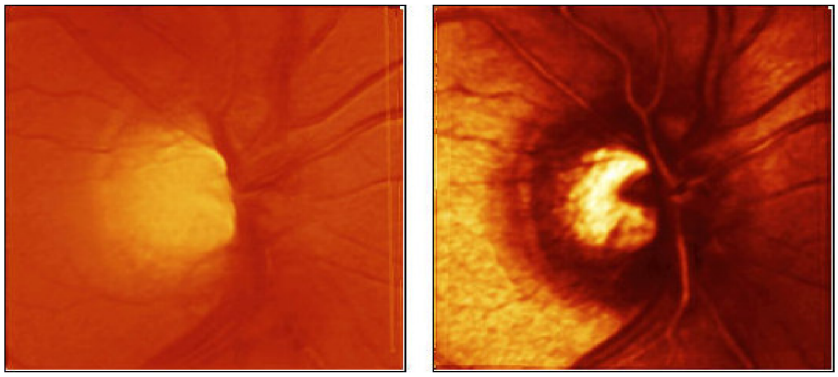
Jakmile je tato kalkulace provedena ve všech místech sekčních obrazových rovin, je výsledkem matice 256 x 256, což je více než 65000 nezávislých měření, každé s reprodukovatelností asi 10 až 20 μm [8]. Vizualizací matice s měřením vrcholu *z-profilu* se zobrazí obraz. To je dosaženo transformací každého specifického vrcholu do specifické barvy shodné s barevnou stupnicí, kde tmavé barvy představují přední, vystouplé struktury a světlé barvy představují propadlé, proláklé struktury.



Obr.2.9*: Barevně kódovaný obraz topografie [8].*

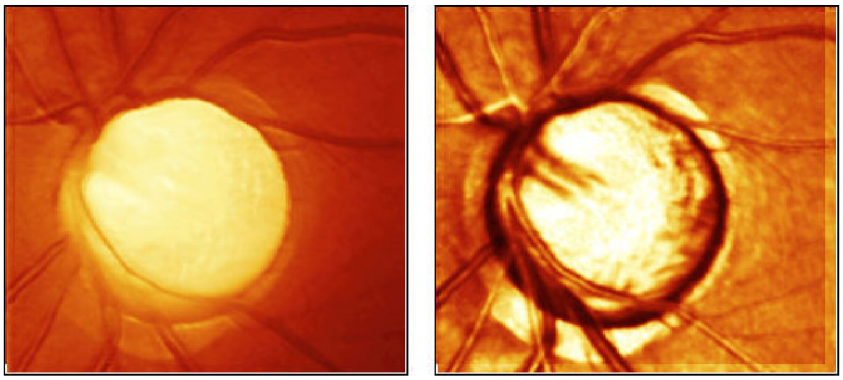
Výsledkem je barevně kódovaný obraz topografie, který je na obr.2.9 vlevo. Jáma zrakového nervu se jeví jasně protože je vyhloubená; zvýšený sítnicový povrch se jeví tmavý. Pomocí informace v topografii obrazu, můžeme určit trojrozměrné vlastnosti zkoumané struktury. Pravý snímek na obr.2.9 ukazuje obraz který je sumou 32 sekčních obrazů a který reprezentuje intenzitu obrazu v každém bodu, srovnatelnou s fotografií očního pozadí. Diky intenzitnímu obrazu je možno kvalitně provést multimodální flexibilní registraci barevného obrazu získaného z Fundus kamery a takto slicovany obraz pak použít jako texturu a vytvořit tak 3D model ONH (optical nerve head) s reálnými barvami.

Cílem topografické analýzy optického disku je buď kvantitativní popis jeho aktuálního stavu a klasifikace (např. normální, nebo abnormální), nebo srovnání více topografických obrazů, k tomu aby bylo možné určit topografické změny a kvantifikovat postup zeleného očního zákalu [1]. Existují v podstatě dva způsoby jak hodnotit aktuální stav topografie zrakového nervu: 1) interaktivní měření a 2) parametrický popis, tj. stanovení sady stereometrických parametrů. Následující příklady demonstrují interaktivní měření.



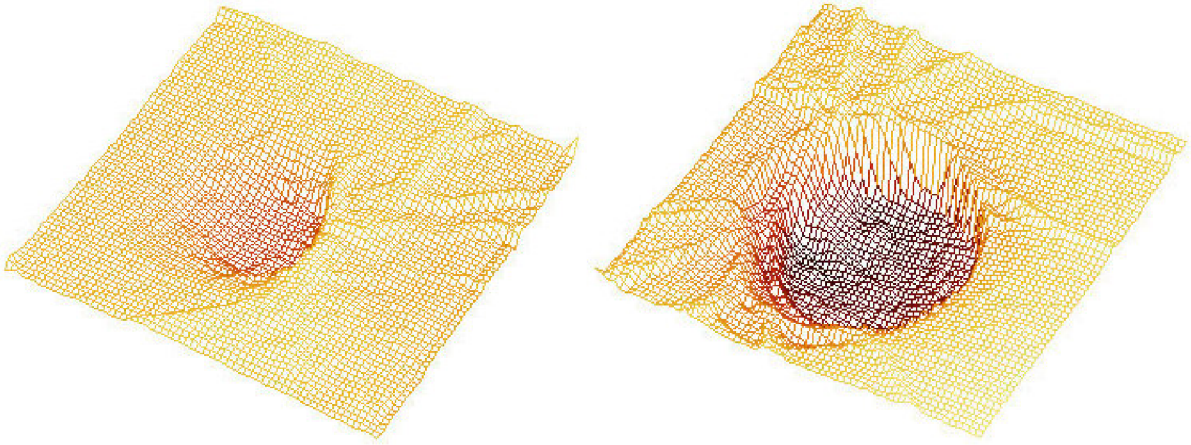
***Obr.2.10**: Obraz topografie (vlevo) a odrazivosti (vpravo) zrakového nervu normálního [8].*

Obr.2.10 ukazuje topografický obraz (snímek vlevo) a obraz odrazivosti (snímek vpravo) normálního zrakového nervu a obr.2.11 u glaukomatózního zrakového nervu. V barevně kódovaných topografických obrazech, se kódují jasnými barvami prohloubené struktury, zatímco tmavé barvy kódují zvýšené struktury. Velikost zorného pole v obou vyšetřeních je 10° [8]. Vysoce rozdílná velikost a tvar pohárů je zřejmá. V glaukomatózním oku se pohár jeví mnohem větší a hlubší.



***Obr.2.11:** Obraz topografie(vlevo)a odrazivosti(vpravo) zrakového nervu glaukomatózního [8].*

Topografické obrazy mohou být také zobrazeny jako pseudo-3D obrazy (obr.2.12). Tyto obrazy zřetelně ukazují rozdíl mezi miskovitým tvarem normálního (snímek vlevo) a glaukomatózního oka (snímek vpravo).Glaukomatózní pohár je mnohem hlubší a vykazuje vyšší strmost podél okrajů.



***Obr.2.12**: Pseudo-3D obrazy [8].*

3 Známé způsoby práce s Java applety

Jak již bylo zmíněno v úvodu, návrh 3D vizualizace optického disku oka má být koncipován tak, aby použití navrženého programu bylo interaktivní a předpokládá se také jeho umístění do webového rozhraní, ke kterému se mohou připojit počítače postavené na různých systémových platformách. Program tedy musí být systémově nezávislý a musí podporovat možnost implementace do webového rozhraní. Použití appletu, který lze vytvořit pomocí programovacího jazyka Java, je tedy pro tento účel nejvýhodnější, protože právě systémovou nezávislost a možnost implementace do webového rozhraní applety podporují.

3.1 Java applety

Applet je speciální typ programu Java, který lze stáhnout z internetu a spustit v prohlížeči s podporou technologie Java. Applet je obvykle vložen do webové stránky a spouští se v kontextu prohlížeče. Applet musí být podtřídou třídy **`java.applet.Applet`**, která poskytuje standardní rozhraní mezi appletem a prostředím prohlížeče. Pokud je applet načten do webové stránky, prohlížeč řídí zpracování appletu voláním určitých metod. Třída **`Applet`** v zásadě obsahuje 4 metody [17]:

- **`init()`** – tato metoda slouží k libovolné inicializaci, kterou applet vyžaduje. Je volána za atributy **`param`** značky **`applet`**,
- **`start()`** – tato metoda je automaticky volána po metodě **`init()`**. K jejímu volání dochází také pokaždé, když se uživatel vrátí na stránku s appletem po zobrazení jiných stránek,
- **`stop()`** – tato metoda je automaticky volána kdykoliv uživatel opustí stránku obsahující applety. Pomocí uvedené metody lze zastavit animaci,
- **`destroy()`** – tato metoda je volána pouze při normálním ukončení prohlížeče.

Jazyk Java je možné uplatnit mnoha různými způsoby. Mezi ty nejčastější patří Java applety a aplikace. Rozdíl mezi appletem a aplikací je v tom, že applet funguje v kontextu webového prohlížeče a přitom je zpravidla integrován do webové stránky, zatímco aplikace se spouští samostatně mimo prohlížeč. Applety jsou proto vhodné zejména k poskytování funkcí na webových stránkách, které vyžadují vyšší úroveň interaktivity nebo animace, než lze dosáhnout pomocí jazyka HTML, např. v grafických hrách, pro složité úpravy nebo interaktivní vizualizace dat.

Applet je při každé návštěvě webové stránky, v níž je obsažen, načten do internetového prohlížeče a je spouštěn v prostředí Virtual Java Machine. Výhodou je, že applet se může stáhnout na pevný disk uživatele pouze jednou a nemusí se stahovat pokaždé při přístupu na danou webovou stránku. Applet může mít zpřístupněny některé své vlastnosti a metody jako veřejné, což umožňuje externím programům manipulaci s nimi.

Applety lze tvořit prakticky v jakémkoli prostředí programovacího jazyka Java. Pro psaní kódu appletů je zapotřebí znát programovací jazyk Java, jeho syntaxi a sémantiku. Pro vložení appletu do HTML stránky se používá značka APPLET. Ta má tyto klíčové parametry:

- CODE – který obsahuje název třídy appletu,
- CODEBASE – udává cestu k této třídě,
- WITDH – určuje šířku okna appletu,
- HEIGHT – určuje výšku okna appletu.

Odkaz pak může vypadat následovně:

```
<APPLET CODE="MujApplet.class" WIDTH=200 HEIGHT=80>
</APPLET>
```

Pokud je applet součástí nějakého balíčku, musí parametr CODE obsahovat jméno včetně jména balíčku (např. *MujBalik.MujApplet.class*). Java applety lze rovněž spouštět pomocí utility appletviewer, která je součástí JDK (Java Development Kit). Parametrem této utility je soubor HTML, který odkazuje na applet.

Vzhledem k tomu, že applet bude pracovat s grafickým uživatelským rozhraním GUI, je třeba zajistit interaktivitu a to pomocí objektů sady **`Swing`**, která je k této činnosti určena. Sada **`Swing`** nabízí kromě grafických komponent také bohatou podporu operací zpět (undo), textovou sadu s rozsáhlými možnostmi přizpůsobení, integrovanou podporu internacionalizace a funkce zpřístupnění. Sada **`Swing`** také nabízí speciální podtřídu třídy Applet a názvem **`javax.swing.JApplet`**.

Tuto třídu je vhodné použít pro všechny applety, které vytvářejí své grafické uživatelské rozhraní pomocí komponent **`Swing`**. Aby mohly programy fungovat na různých platformách, poskytuje sada **`Swing`** různé motivy vzhledu, včetně možnosti vytvořit vlastní grafické motivy. Sada **`Swing`** má pochopitelně k dispozici také základní funkce uživatelského rozhraní, jako je přetažení (drag-and-drop), obsluha událostí, přizpůsobitelné vykreslování a správa oken.

3.2 Možnosti zpracování obrazu a 3D vizualizace v Javě

Pro zobrazování ve 3D prostoru se nejvíce používají dvě metody. Jedná se o metodu *rekonstrukčního* zobrazování, která využívá celou sérii sekčních obrazů (obr.2.3) a z těchto „řezů“ se pak vhodnými algoritmy rekonstruuje model původní scény. Nebo lze využít modelování 3D scény pomocí topografického obrazu. Na základě topografie snímku je pak vypočten a vygenerován třetí rozměr, který je nakonec zobrazen ve 3D prostoru jako model původní scény.

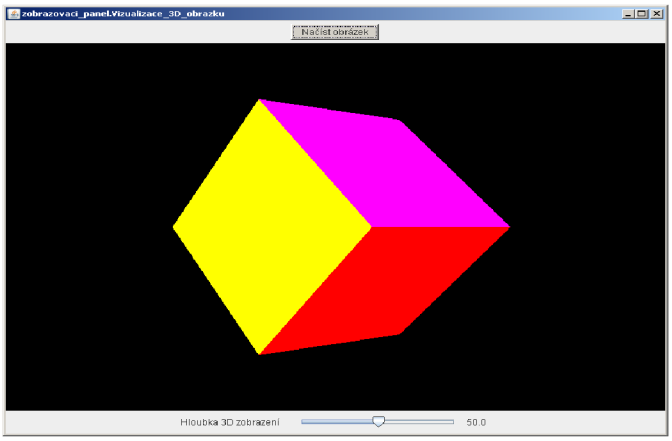
Vzhledem k tomu, že základní knihovny programovacího jazyka Java v současnosti neposkytují efektivní nástroje pro trojrozměrnou vizualizaci, je třeba hledat alternativní nadstavby. Aby bylo možné efektivněji zpracovávat data ve 3D prostoru, byla Java rozšířena o některé přídavné knihovny a moduly. Níže jsou uvedeny a stručně popsány některé z nich.

3.2.1 Java 3D

Java 3D je rozšířením ke knihovnám jazyka Java a slouží pro práci s trojrozměrnou grafikou. Programy napsané v Java 3D mohou běžet na mnoha odlišných typech počítačů a přes internet. Java3D je volně šiřitelná aplikace a k dispozici je momentálně verze 1.5.2. Knihovna tříd Java3D poskytuje rozhraní které je jednodušší než rozhraní mnoha jiných grafických knihoven, ale zároveň je dostačující k tvorbě her a animací. Java 3D staví na existujících technologiích jako je DirectX a OpenGL takže programy neběží tak pomalu jak by bylo možné očekávat. Java 3D také umožňuje začlenit objekty vytvořené 3D modelovacími nástroji jako je TrueSpace nebo dokonce VRML modely [13]. V Java 3D lze jednotlivá umístění popsat souřadnicemi *x*, *y* a *z*. Souřadnice se zvětšují směrem doprava na ose *x*, směrem nahoru na ose *y* a směrem do obrazovky na ose *z*.

Pro použití knihoven je třeba pomocí příkazů **`import`** na začátku programu importovat potřebné části knihovny Java 3D. Při programování pomocí Java 3D se vytváří tzv. vesmír (třída **`Universe`**), do něhož se obvykle vkládá struktura **`BranchGroup`** obsahující objekt, nebo skupinu objektů pomocí konstruktoru **`addChild()`**. Objekty je pochopitelně možné „transformovat“, tedy pohybovat s nimi, měnit jejich barvu, nastavit kolik světla odrážejí, nebo lze aplikovat textury na jejich povrch. Takové změny umožňuje dělat třída **`Appearance`**, která obsahuje příslušné metody. 2D obrázky typu **`Image`** lze v prostředí Java 3D také zpracovávat a to pomocí třídy **`TextureLoader`** použitím metody **`ImageComponent2D getImage()`**.

Pro vykreslování 3D grafiky se používá speciální „plátno“ (zobrazovací plocha) s názvem **`Canvas3D`**. Jedná se o obdélníkový pohled na objekty ve vesmíru, resp. plátno se vloží do rámu a pak se vytvoří vesmír který se zobrazí na plátně. **`Canvas3D`** využívá výhod grafické karty ke zvýšení výkonu. Naneštěstí to znamená, že to nefunguje příliš dobře se **`Swing`** GUI. Tyto komponenty se nazývají "odlehčené". Problémem může být, že odlehčené komponenty mohou být skryty **`Canvas3D`** i když by měly být viditelné. Řešením může být použití starších komponent **`AWT`** místo komponent **`Swing`**. Na obr.3.1 je vlastní návrh funkčního appletu, vytvořeného pomocí výše uvedených tříd, který dokáže zobrazit na ploše 3D objekty typu **`Node`**.

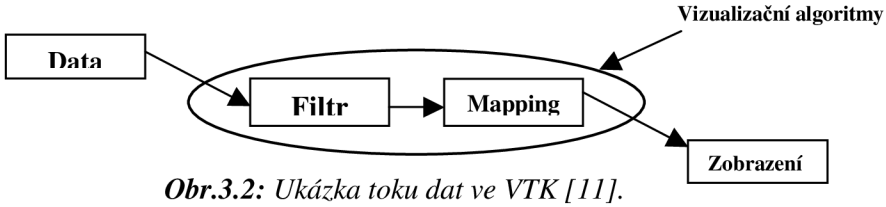


Obr.3.1: *Ukázka objektu vytvořeného pomocí tříd Java 3D.*

3.2.2 Java VTK

Pro vytvoření 3D rekonstrukcí lze použít knihovnu VTK (The Visualization Toolkit). VTK je multiplatformní systém. VTK samo o sobě neobsahuje grafické uživatelské prostředí. Uživatel píše aplikace v jazyce C++, příp. využívá wrapperů pro jazyky Tcl, Java, Python či některého z jazyků pro platformu .NET [VTKNET]. V současnosti je dostupných přes 800 tříd pro nejrůznější použití. Knihovna VTK je dostupná jako tzv. „open source“, to znamená že je pro zájemce volně k dispozici, včetně zdrojových kódů. Momentálně je k dispozici verze 5.4.0. Je vytvořena na bázi jazyka C++ s objektově orientovanou strukturou. VTK jako knihovna pro vizualizaci a zpracování 3D grafiky vykazuje vysoký stupeň abstrakce – lze vytvářet grafické a vizualizační aplikace rychle. Nevýhodou knihovny VTK mohou být její relativně vysoké nároky na výkon a do jisté míry i její obsáhlost [15]. Při implementaci VTK do appletu by mohly být právě nároky na obrovský prostor pro ukládání dat, velkou nevýhodou, protože by mohlo docházet ke swapování (tj. přesunu dat z operační paměti na pevný disk), čímž by se zpomaloval běh appletu.

Obr.3.2 zachycuje ukázku toku dat ve VTK [11]. Základní princip práce spočívá v předání vstupů, které se budou dále zpracovávat, nějakému *filtru*, který data zpracuje. Pak zpracované údaje předá *mapperu*, který data rozvrhne a pošle dál na zpracování. Data lze do zobrazované scény vložit dvěma způsoby: přímým vytvořením (konstrukcí) objektu pomocí nabízených funkcí, nebo načtením ze souboru. V takovém případě VTK nabízí mnoho funkcí pro načítání dat (např. pro načtení dat vstupních lze použít **`vtkTIFFReader`**). Pro snadnou manipulaci s 3D modelem nabízí VTK několik tříd, které zajistí např. rotaci objektu, přiblížení oddálení apod. V modulu pro 3D rekonstrukci a vizualizaci lze využít **`vtkRenderWindowInteractor`** který tyto funkce podporuje.



Obr.3.2: *Ukázka toku dat ve VTK [11].*

Třídy lze rozdělit do těchto skupin: objekty grafické pipeline, objekty vizualizační pipeline a pomocné objekty [12]. Objekty grafické pipeline jsou elementy scény a objekty potřebné pro rendering. Z objektů vizualizační pipeline se sestaví graf toku dat. Jejich hlavní úlohou je transformace vstupní data do reprezentace, kterou je možné vykreslit. Zatímco objekty vizualizační pipeline jsou striktně rozděleny na objekty datové a výkonné, objekty grafické pipeline mají většinou charakteristiku obou. Jejich podobjekty jsou závislé na hardware, resp. na grafickém rozhraní.

Vizualizační pipeline začíná zdrojem. Zde objekty třídy **`vtkSource`** generují polygonální reprezentace geometrických objektů. Ty jsou napojeny na **`vtkPolyDataMapper`** a **`vtkGlyph3D`**. Filtr **`vtkGlyph3D`** na pozici každého vrcholu vstupního datasetu zkopíruje **`Glyph`**. **`Glyph`** je definován vstupními polygonálními daty filtru, jeho směr a velikost může být závislá na normále resp.

vektoru. Objekt **`vtkPolyDataMapper`** mapuje polygonální data na grafická primitiva. Hraje roli supertřídy pro poly data mappery, které jsou závislé na konkrétním zařízení či rozhraní [19].

Grafická pipeline začíná definováním základních elementů scény, objekty **`vtkActor`**. Ty jsou vloženy do **`vtkRendereru`**, který má za úkol jejich vykreslení, rendering. Obdobně by se do **`vtkRendereru`** vkládalo i světlo, **`vtkLight`**. Okno, do kterého lze renderovat, je poskytnuto objektem **`vtkRenderWindow`**. Základní interaktivitu, jakou je rotace a posuv objektu, zajišťuje objekt **`vtkRenderWindowInteractor`** [19].

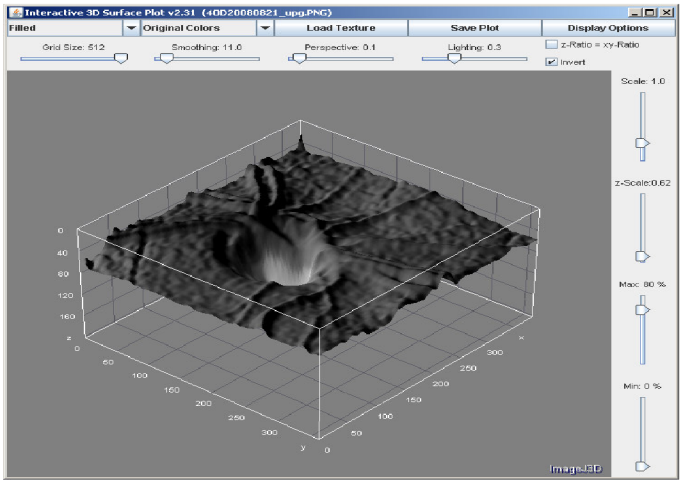
3.2.3 Image J

Na rozdíl od předchozích přídavků knihoven je ImageJ program. Je distribuovaný zdarma a slouží ke zpracování obrazu. Je naprogramován v jazyce Java a byl inspirován programem NIH Image pro Macintosh. Program může běžet jako online applet, nebo je možné stáhnout jej z Internetu jako aplikaci samostatně běžící na počítači s nainstalovanou Javou 1.4 nebo vyšší. Distribuce z Internetu jsou dostupné pro Windows, Mac OS, Mac OS X a Linux. Momentálně je k dispozici verze 1.41.

Program může zobrazovat, editovat, analyzovat, zpracovávat, ukládat a tisknout 8-mi, 16-ti a 32 bitové obrázky. Může číst mnoho obrazových formátů včetně TIFF, GIF, JPEG, BMP, DICOM, FITS a také "surová" obrazová data. Podporuje také práci se sérií obrazů, které sdílejí jednotlivé okno. Je multi-vláknový, takže časově náročné operace jako obrazové čtení souboru může být vykonáváno paralelně s dalšími operacemi. Program může počítat velikost oblastí, či hodnoty pixelů definovaných uživatelem a může měřit vzdálenosti a úhly. Dále může vytvářet histogramy a kreslit čárové profily. Podporuje i standardní funkce zpracování obrazu: nastavení kontrastu, ostrosti, vyhlazování, filtraci.

ImageJ dokáže geometrické transformace jako změnu velikosti, rotaci a otáčení. U obrazu může lze měnit měřítko od 1:32 do 32:1. Všechny analýzy a funkce zpracovací jsou dostupné v každém faktoru zvětšení. Program podporuje práci s velkým množstvím oken (obrazů) současně, což je omezeno pouze využitelnou pamětí. K dispozici je také prostorová kalibrace, která poskytuje reálné rozměrové měření v milimetrech. Dostupná je také kalibrace ve stupních šedé. Velkou výhodou je, že zdrojový kód programu je volně dostupný. ImageJ byl také navržen jako program s otevřenou architekturou, což poskytuje rozšiřitelnost programu pomocí pluginů napsaných v jazyce Java. Pluginy mohou být vytvořeny použitím vestavěného editoru i Java kompilátoru. Uživatelsky psané pluginy umožňují řešit téměř jakékoliv zpracování obrazu, nebo problémy analýzy [13].

Pro zpracování obrazů ve 3D prostoru zde slouží plugin *Interactive 3D surface plot*. Pomocí tohoto pluginu lze nastavovat velikost 3D vzorkovací mřížky, vyhlazování hran, úhel pohledu, osvětlení a také změnu velikosti objektu. Lze měnit způsob vykreslování povrchu objektu pomocí bodů, čar, sítě, výplně, nebo „isolines“. Další možností je pseudo zabarvení objektu pomocí předdefinovaných odstínů, nebo Look up Tables. Je také možnost načtení a použití vlastní textury. Na obr.3.3 je ukázka programu ImageJ. Jedná se o plugin *Interactive 3D surface plot*, pomocí něhož je ve 3D prostoru zobrazen topografický snímek z obr.4.5.



Obr.3.3: *Ukázka programu ImageJ – plugin Interactive 3D surface plot.*

4 Vlastní návrh 3D modelu optického disku oka

V první řadě je třeba zvážit jak by měl být applet ovládán a jak by měl vypadat. Protože se předpokládá, že aplikace bude umístěna jako applet do webového rozhraní, je koncepce ovládání následující:

1) Přednostně ovládání pomocí myši:

- Po stisku a při držení levého tlačítka myši (tzv. funkce dragged), se bude ovládat rotace 3D modelu v osách *x* a *y*. Standardně bude samozřejmě možné klikáním na levé tlačítko ovládat funkce grafických komponentů v okně appletu.
- kliknutím na pravé tlačítko bude možné přepínat mezi módem pro změnu velikosti 3D modelu (tj. přiblížení/oddálení), módem zobrazení třetího rozměru v ose *z* (tj. hloubky 3D zobrazení) a módem vyhlazení povrchu obrázku.
- Otočným kolečkem myši se bude měnit velikost 3D modelu, nebo se bude nastavovat třetí rozměr v ose *z*, nebo se bude volit velikost vyhlazení povrchu (v závislosti na výběru příslušného posuvníku pomocí pravého tlačítka).
- Mohlo by být také užitečné ovládat posun 3D modelu po ploše v ose *x* a *y*. K tomu by mohlo být využito stisknutí a držení (funkce dragged) pravého tlačítka myši.

2) Ovládací prvky v okně appletu:

- V první řadě bude třeba vyvolat pomocí grafického tlačítka dialog nabídky pro výběr obrázku ze složky na pevném disku počítače.
- Pro změnu velikosti, zobrazení třetího rozměru v ose *z* a vyhlazení povrchu budou v appletu posuvníky, kterými bude možné funkce ovládat jak v reálném čase, tak i přednastavit hodnotu, kterou bude možné aplikovat na obrázky otevírané následně. Vedle posuvníku musí být také popis, aby bylo jasné jakou má posuvník funkci.
- Nutný bude také displej s aktuální hodnotou, který bude zobrazovat hodnotu nastavenou kolečkem myši resp. jezdcem posuvníku (tj. velikost obrázku, nebo třetího rozměru v ose *z*, vyhlazení povrchu). Pro rychlou orientaci a informaci o tom který komponent je aktivní, by bylo vhodné měnit barvu pozadí aktivního displeje i posuvníku.
- Dalším rozšířením, které ale není pro správné zobrazování nutné, je možnost vybrat z nabídky mód zobrazení tj. vykresleni pomocí bodů, čar, sítě, povrchu, osvětlení modelu.
- Dalším rozšířením by mohlo být volitelné zobrazování os *x*, *y*, *z*.
- A dále by také mohlo být užitečné zobrazovat (např. formou stavového řádku) cestu k adresáři a umístěním obrázku na disku (tj. odkud byl obrázek načten).

3) Možnost ovládání klávesnicí:

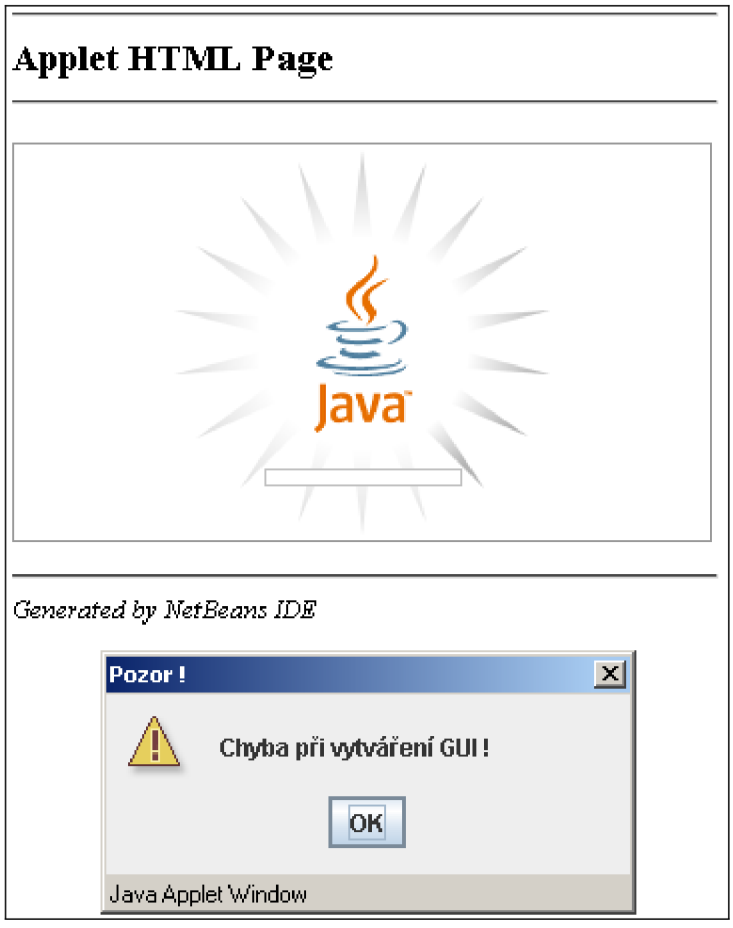
- Přepínání mezi komponenty v okně bude možné standardně pomocí klávesy Tab; po přepnutí se bude funkce komponentu ovládat pomocí kláves Enter, Space, Tab nebo šipek (to je již dáno u jednotlivých komponentů implicitně).
- Rotaci 3D modelu v osách *x* a *y*, bude možno ovládat pomocí kláves se symbolem šipek.
- Posun 3D modelu v osách *x* a *y* bude ovládán klávesami PgDn, PgUp, Home, End.
- Přepínání mezi módem pro změnu velikosti 3D modelu (tj. přiblížení/oddálení) a módem zobrazení třetího rozměru v ose *z*, bude možné klávesou Space.
- Třetí rozměr v ose *z*, velikost 3D modelu, nebo změna velikosti se budou měnit pomocí kláves Q a W Protože klávesy Q a W sousedí s klávesou Tab bude tak ovládání ergonomické (postačí 3 prsty levé ruky).

4.1 Zobrazení 2D obrázku v prostředí 3D

Pro vytvoření a odladění appletu bylo použito vývojové prostředí NetBeans verze 5.5. V tomto prostředí byl nejdříve vytvořen jednoduchý applet, který načte obrázek z pevného disku uživatele a následně jej zobrazí ve 3D prostoru. Applet je nazván **`NacteniObrazku`** a popis jeho zdrojového kódu je v Technickém manuálu (kapitola 1).

Pro načtení obrázku z pevného disku je nejlepším řešením použití dialogu nabídky, který je každému uživateli PC znám z mnoha jiných aplikací a jehož obsluha je uživatelsky nenáročná. Tento dialog se bude vyvolávat pomocí grafického tlačítka třídy **`Button`**. Toto tlačítko musí být, stejně jako dialog nabídky, ošetřeno pomocí událostí (Events), aby byla zajištěna jeho funkčnost při vyvolávání dialogu nabídky.

Po spuštění appletu, je vždy nejdříve volána metoda **`init`**. V té je vytvořena obsluha výjimky (exception handler), která bude volána v případě chyby při inicializaci grafických komponentů. Jedná se o standardní ošetření které je automaticky vytvářeno při tvorbě appletů v prostředí NetBeans. Popis vytvoření metody a popis její funkce je v Technickém manuálu (kapitola 1.2). Na obr.4.1 je zobrazen informační dialog třídy **`JOptionPane`**, který bude zobrazen v případě chyby při inicializaci grafických komponentů [16].



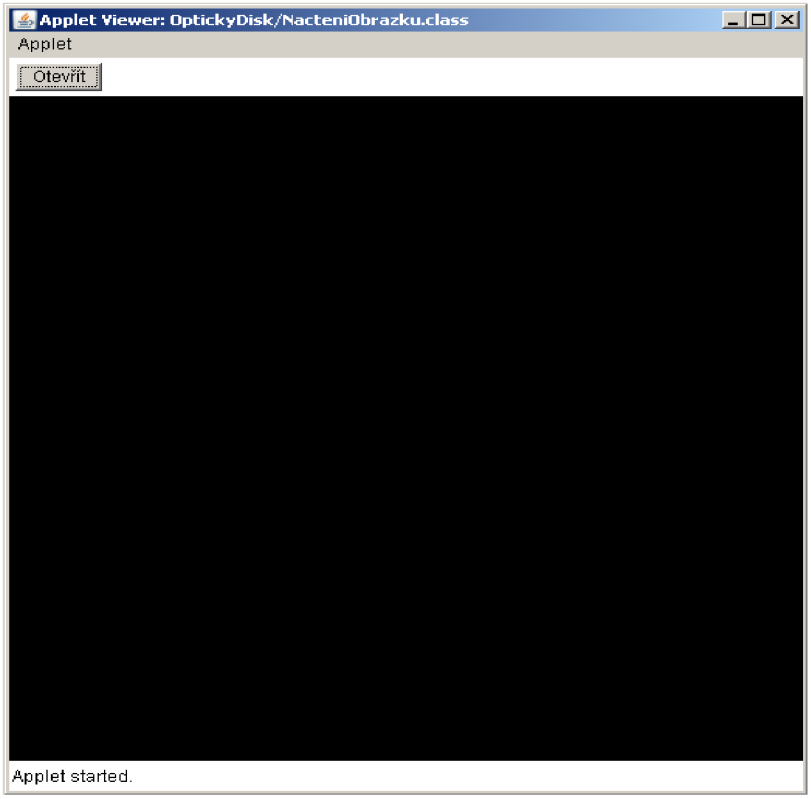
Obr.4.1: *Upozornění při vyvolání výjimky.*

4.1.1 Vytvoření grafického uživatelského rozhraní

Pro vytvoření grafického uživatelského rozhraní (GUI) je volána vlastní metoda s názvem **`inicializaceKomponentu`**. V této metodě je nejdříve inicializováno prostředí třídy **`SimpleUniverse`** (podtřída třídy **`VirtualUniverse`**, do kterého je implementováno 3D pozadí třídy **`Canvas3D`**, v němž se budou zobrazovat obrázky ve 3D prostoru.

Dále jsou v metodě načítány ostatní grafické komponenty. Nejdříve dva panely třídy **`Panel`**: **`hlavniPanel`**, na který je umístěno již nakonfigurované 3D pozadí a dále **`horniPanel`** do kterého se umístí tlačítko pro otevření dialogu nabídky.

Po spuštění kódu se v integrovaném prohlížeči vývojového prostředí zobrazí okno které je na obr.4.2. Popis této metody je v Technickém manuálu (kapitola 1.3).

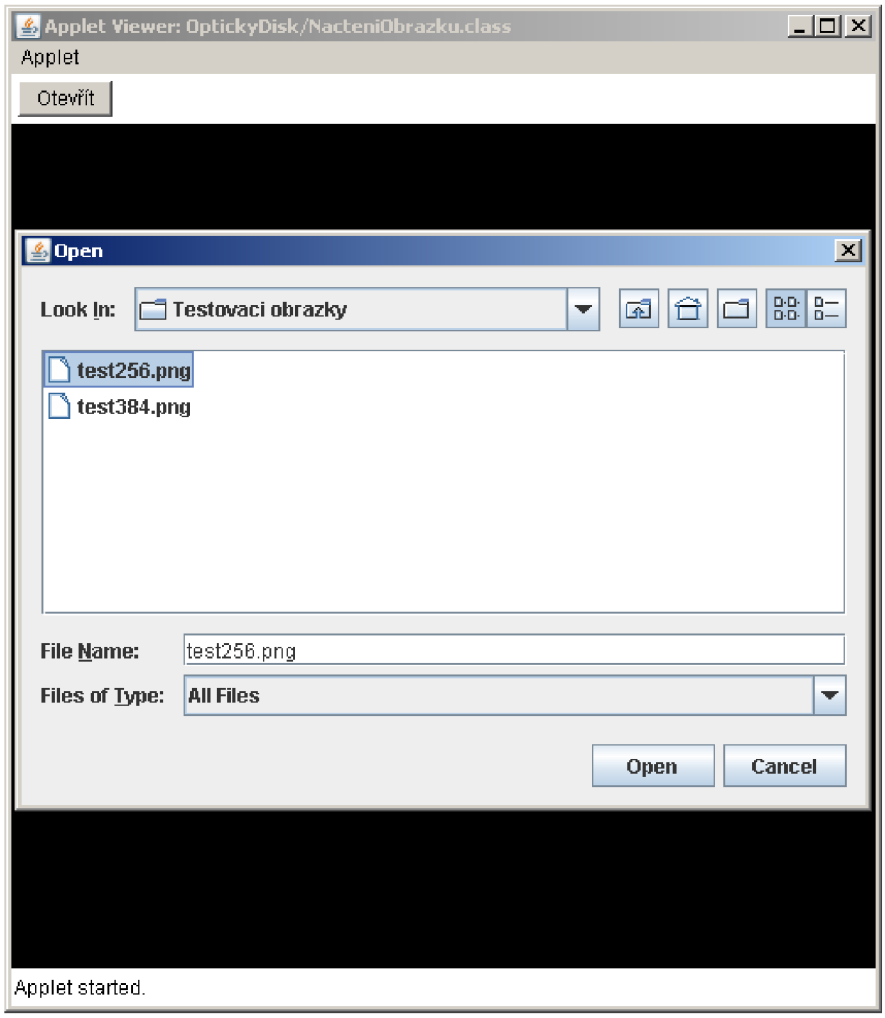


Obr.4.2: *Inicializace grafických komponentů.*

4.1.2 Obsluha událostí

Obsluha událostí vyvolaných grafickými komponenty (zde grafickým tlačítkem a dialogem načtení) se musí naprogramovat a ošetřit v abstraktní metodě **`public void actionPerformed`**`(ActionEvent e)`. Pomocí metody **`getSource`** se nejdříve rozliší, který z grafických komponentů událost vyvolal. Pak se vytvoří algoritmus, který zajistí že po stisku grafického tlačítka bude vyvolán dialog nabídky. Po výběru obrázku v dialogu zajistí další část algoritmu načtení informací nutných pro práci s obrázkem tj. název vybraného obrázku a lokalizaci umístění na pevném disku uživatele.

Na obr.4.3 je ukázka zobrazení okna, ve kterém je vyvolán dialog nabídky. Popis algoritmů metody pro obsluhu událostí je v Technickém manuálu (kapitola 1.4).



Obr.4.3: *Vyvolání dialogu pro načtení obrázku.*

4.1.3 Vytvoření scény

V další části appletu je naprogramována vlastní metoda s názvem **`vytvorScenu`**, v níž bude vytvořen objekt třídy **`BranchGroup`**, který zajistí seskupení a zobrazení scény na 3D pozadí. Pro vykreslení scény byl použitý objekt třídy **`Shape3D`**. U tohoto objektu se musí pomocí metod **`setGeometry`** a **`setAppearance`** nastavit jednak vlastnosti týkající se geometrie objektu (tvar objektu, souřadnice vrcholů apod.), ale také vlastnosti týkající se vzhledu objektu (textura povrchu, průhlednost objektu apod.). Pomocí metody **`compile`** bude nakonec celá scéna seskupena, což zajistí její kompaktnost při vykreslování ve 3D prostředí.

Vytvoření geometrie objektu

Do appletu je dopsána další vlastní metoda, s názvem **`vytvorGeometrii`**. V této metodě je vytvořen objekt třídy **`GeometryArray`**, typu **`QuadArray`**. Pomocí konstruktoru tohoto objektu bude dle zadaných souřadnic vytvořena oblast čtvercového tvaru. Aby však bylo možné na povrch této oblasti namapovat texturu, musí být povolena práce s texturou a je také nutné povolit zadávání souřadnic. Zadání souřadnic je možné provést pomocí třídy **`Point3f`** a metody **`setTextureCoordinate`**, která je instancí třídy **`QuadArray`**.

Vytvoření vzhledu objektu

Další vlastní metodou, která je v programu vytvořena je metoda pro nastavení vzhledu objektu. Ta je nazvána **`vytvorVzhled`**. V této metodě byl vytvořen objekt třídy **`Appearance`**, pomocí něhož lze provést samotné namapování textury na oblast třídy **`QuadArray`**. Aby však mohl být načten obrázek který bude použitý jako textura, musí být nejdříve vytvořen tzv. zavaděč textury, kterým je objekt třídy **`TextureLoader`**. Z tohoto zavaděče pak bude obrázek přes objekt třídy **`ImageComponent2D`** převeden do objektu třídy **`Texture2D`**, kam bude načten metodou **`setImage`**. Samotné namapování textury do objektu třídy **`Appearance`**, se provede pomocí metody **`setTexture`**.

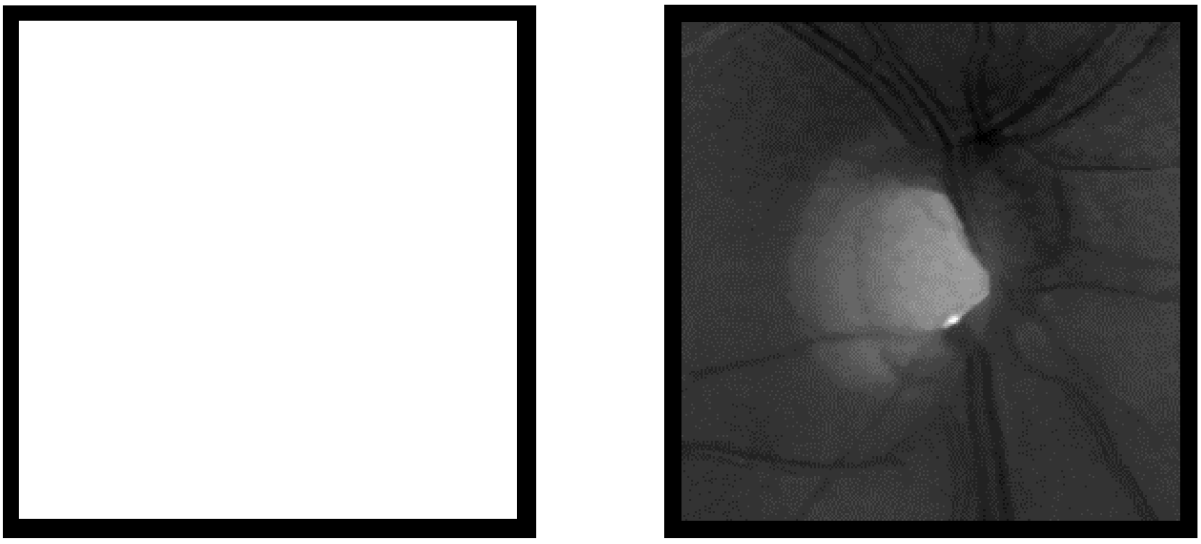
Popis algoritmů metody pro vytvoření scény je v Technickém manuálu (kapitola 1.5), stejně jako popis algoritmů metody pro vytvoření geometrie objektu a popis algoritmů metody pro vytvoření vzhledu objektu.

4.1.4 Podmínky pro načtení obrázku

Po spuštění appletu lze v jeho 3D prostředí zobrazovat 2D obrázky a to jako textury mapované na objekt třídy **`QuadArray`**. Je zde ovšem jedna zásadní podmínka pro správné zobrazení a to, že rozměry obrázku v pixelech, musejí být celočíselnou mocninou čísla 2 [14]. Lze tedy zobrazovat pouze obrázky, které mají rozměry 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9 … což vyjádřeno čísly znamená rozměry v pixelech: 2, 4, 8, 16, 32, 64, 128, 256, 512 … atd.

Vstupní topografický snímek který se bude zpracovávat má ovšem rozměry 384 x 384 pixelů, takže tuto podmínku nesplňuje. Na obr.4.4 vlevo, je zobrazení zkušebního snímku, jehož rozměry 384 x 384 pixelů neodpovídají této podmínce. Pokud se snímek ořízne tak, aby jeho rozměr výše uvedené podmínce vyhovoval, je bez problému zobrazen. Na obr.4.4 vpravo je zobrazen zkušební snímek oříznutý na 256 x 256 pixelů.

Tento program je jednoduchý, ale protože by bylo nutné topografické snímky před načtením do tohoto programu ořezávat na některý z výše uvedených rozměrů, není tato metoda pro vytvoření appletu dle zadání vhodná. Byly tedy vyzkoušeny další přístupy, které by měly tento nedostatek odstranit.



Obr.4.4: *Vliv rozměru obrázku na zobrazení: 384x384px (vlevo), 256x256px (vpravo).*

4.2 Vykreslení jednotlivých pixelů obrázku v prostředí 3D

Aby se zamezilo tomu že se nezobrazí obrázky o rozměrech které nejsou podporovány, je třeba použít postupné vykreslení jednotlivých pixelů v obrázku. Základem tohoto postupu je vytvoření algoritmu, ve kterém budou pixely z obrázku načteny pomocí třídy **`PixelGrabber`** a pak vykresleny použitím třídy **`GeometryArray`**. Za tímto účelem byl vytvořen applet **`VykresleniPixelu`**, který vychází z předchozího appletu. Metody **`init`**, **`inicializaceKomponentu`** a **`vytvorScenu`** zůstanou beze změn. Musí se však upravit ostatní metody, a kvůli výpočtům spojeným s použitím třídy **`PixelGrabber`** je třeba vytvořit navíc další metodu, kde budou prováděny výpočty tak, aby co nejméně zpomalovaly vykreslení jednotlivých pixelů obrázku a tím i chod programu.

4.2.1 Obsluha událostí

V metodě pro obsluhu událostí **`actionPerformed`** nastane jediná změna. Do metody bude přidán ještě další řádek s příkazem který volá vlastní metodu pro výpočet parametrů jednotlivých pixelů. Tato nová metoda je nazvána **`vypocty`**. Popis metody pro obsluhu událostí je v Technickém manuálu (kapitola 2.1).

4.2.2 Čtení a normalizace hodnot jednotlivých pixelů

V metodě **`vypocty`** se provádí čtení pixelů obrázku a převod na hodnoty vhodné pro kreslení. Metoda proběhne jednou před vykreslením grafiky. Protože kreslení je náročné na výpočetní výkon je třeba provést v této metodě co nejvíce výpočtů, aby byl dostatek výpočetního výkonu pro vykreslení. Aby bylo možné použít konstruktor třídy **`PixelGrabber`**, je třeba znát některé další parametry. Jedná se v první řadě o šířku a výšku obrázku, a také celkový počet pixelů v obrázku. Parametry šířky a výšky obrázku se zjistí pomocí již zmíněné třídy **`TextureLoader`** a celkový počet pixelů se pak z těchto parametrů vypočítá.

Do konstruktoru třídy **`PixelGrabber`** se musí zadávat objekt typu **`Image`**, ze kterého je pak provedeno čtení pixelů. Pro čtení pixelů musí být do konstruktoru dále zapsány souřadnice oblasti ze které mají být pixely čteny (pro zpracování celého obrázku jsou souřadnice počátku 0, 0 a koncové souřadnice mají hodnotu celkové šířky a výšky obrázku). Dalším nutným parametrem je zadání správně nadimenzovaného pole (dle celkového počtu pixelů v obrázku), do kterého budou informace zapsány, a posledními dvěma parametry se určuje délka jednoho čteného řádku (při zpracování celého se čte po celých řádcích tj. od 0 do hodnoty šířky řádku). Systém čtení a indexování pixelů pomocí třídy **`PixelGrabber`** je názorně uveden v tab.4.1.

***Tab.4.1**: Čtení a indexování pixelů pomocí třídy PixelGrabber.*

0	**1**	**2**	**3**	**…**	**šířka - 1**
šířka	**šířka + 1**	**šířka + 2**	**šířka + 3**	**…**	**(2x šířka) - 1**
2x šířka	**(2x šířka) + 1**	**(2x šířka) + 2**	**(2x šířka) + 3**	**…**	**(3x šířka) - 1**
3x šířka	**(3x šířka) + 1**	**(3x šířka) + 2**	**(3x šířka) + 3**	**…**	**(4x šířka) - 1**
…	**…**	**…**	**…**	**…**	**…**
(výška – 1) x šířka	**((výška – 1) x šířka) + 1**	**((výška – 1) x šířka) + 2**	**((výška – 1) x šířka) + 3**	**…**	**(výška x šířka) - 1**

Normalizace odstínu pixelů

Zápis informací o odstínu jednotlivých pixelů v obrázku do pole se provede pomocí metody **`grabPixels`**. Applet je zaměřen na zpracování obrázků ve formátu PNG, který používá tzv. RGBA barevný model. V tomto modelu je zakódována informace o odstínu barvy každého pixelu do sedmi bitů ve formátu ARRGGBB, kde:

- první bit A určuje průhlednost pixelu,
- druhá dvojice bitů RR určuje červenou složku barvy,
- třetí dvojice bitů GG určuje zelenou složku barvy,
- čtvrtá dvojice bitů BB určuje modrou složku barvy.

Každá dvojice bitů určující barevnou složku může nabývat hodnot 00 až FF (vyjádřeno hexadecimálně). To znamená, že odstíny mohou nabývat hodnot 000000 až FFFFFF, což dekadicky vyjádřeno znamená hodnoty od 0 do 16777215. Použitím třídy **`PixelGrabber`** se získá pole hodnot, kde odstínu barvy bílé odpovídá hodnota -1 a odstínu barvy černé odpovídá hodnota -16777216. Tyto hodnoty, které jsou ve formátu **`integer`**, však není možné použít přímo pro vykreslování barev v nadstavbě Java 3D. Tato nadstavba totiž používá vlastní třídu **`Color3f`**, ve které se pro zobrazení barev používají proměnné typu **`float`**. Navíc ve třídě **`Color3f`** odpovídají odstínu bílé barvy hodnoty **`(1.0f, 1.0f, 1.0f)`** a odstínu černé barvy hodnoty **`(0.0f, 0.0f, 0.0f)`**. I zde ovšem platí, že první číslice určuje červenou složku barvy, druhá zelenou složku barvy a třetí modrou složku barvy. Je tedy nutné provést normalizaci hodnot odstínů, tak aby byly v rozmezí 0 až 1, přičemž hodnota 0 bude odpovídat odstínu černé barvy a hodnota 1 odstínu bílé barvy.

Normalizace hodnot odstínů pixelů je provedena tak, že každý prvek pole, který je typu **`integer`**, je nejdříve převeden na typ **`float`**, pak je hodnota každého prvku vydělena maximem (tj. pro odstín barvy bílé je vypočtena hodnota $-5{,}96 \cdot 10^{-8}$ a pro odstín barvy černé hodnota -1). Následně je k výsledku dělení přičtena hodnota 1, čímž se získá pro černou hodnota 0 a pro bílou barvu hodnota 0,99999994. Tyto hodnoty již lze využít pro kreslení ve třídě **`Color3f`**.

Vytvoření vzhledu objektu

Podobně jako byl normalizován odstín barvy, je třeba normalizovat také rozměry pro vykreslení. V nadstavbě Java 3D se pro vykreslení bodů používá třída **`Point3f`**, ve které se pro zobrazení bodu používají tři proměnné typu **`float`**. První proměnná odpovídá umístění bodu na ose x, druhá umístění bodu na ose y, a třetí umístění bodu na ose z. Počátek souřadného systému leží v levém horním rohu a je určen hodnotami **`(0.0f, 0.0f, 0.0f)`**. Je však možné používat i hodnoty, které jsou záporné a také hodnoty které jsou větší než **`1.0f`**.

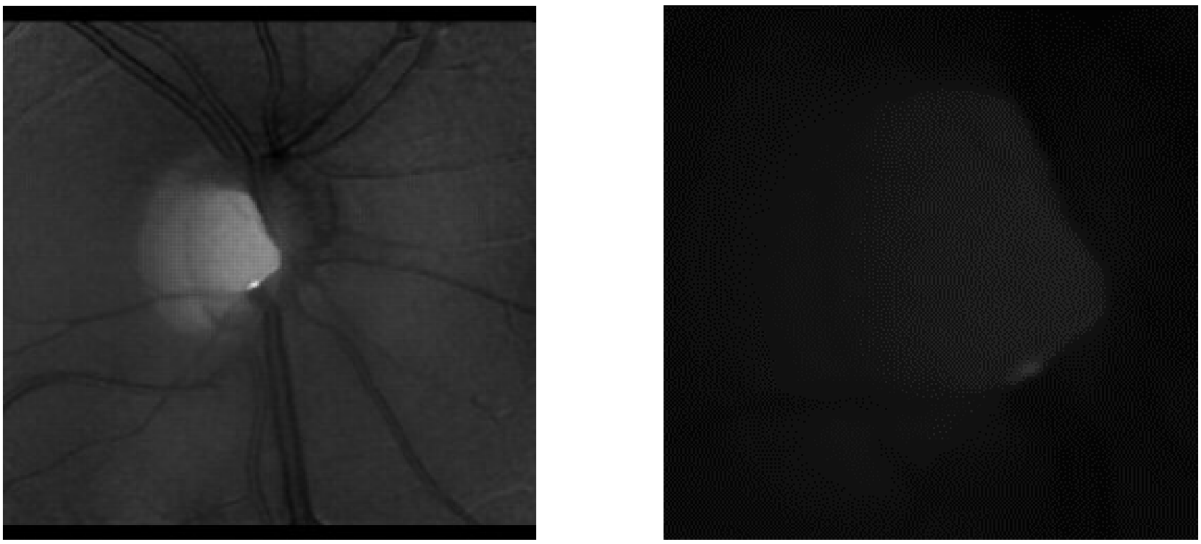
Aby bylo možné vykreslit pixely pomocí třídy **`Point3f`** od souřadnic **`(0.0f, 0.0f, 0.0f)`** do **`(1.0f, 1.0f, 1.0f)`**, je nutné normalizovat hodnoty šířky a výšky obrázku. Normalizace spočívá v převrácení hodnoty šířky (resp. výšky) obrázku, čímž se zajistí že obrázky o různých rozměrech jsou zobrazeny ve stejné velikosti na stejné ploše 3D oblasti. Např. obrázek o velikosti 384 x 384 px má první pixel na souřadnicích **`(0.0f, 0.0f, 0.0f)`** a následující pixel je vykreslen na souřadnicích posunutých o 1/384, tedy **`(0.0026f, 0.0026f, 0.0026f)`**. Obrázek o velikosti 100 x 100 px má první pixel také na souřadnicích **`(0.0f, 0.0f, 0.0f)`**, ovšem následující pixel je vykreslen na souřadnicích posunutých o 1/100, tedy **`(0.1f, 0.1f, 0.1f)`**. Velikost obou obrázků na obrazovce je pak stejná. Rozdíl je v tom, že u prvního obrázku je vykresleno 384 x 384 = 147456 pixelů, ale u druhého je na stejné ploše vykresleno 100 x 100 = 10000 pixelů.

U tohoto řešení by ovšem mohl nastat problém s menšími obrázky, u kterých by mohly být jednotlivé pixely vykresleny tak daleko od sebe, že by obrázek přestal být čitelný. Na obr.4.5 vlevo je obrázek 384 x 384 pixelů a na obr.4.5 vpravo je výřez z téhož obrázku o velikosti 128 x 128 pixelů. Obrázek vpravo je méně čitelný, protože má na stejné ploše a při stejné šířce (resp. výšce) 3x méně pixelů a tyto pixely jsou od sebe ve vzdálenosti 3x větší, než na obrázku vlevo. Tento problém však

lze v nadstavbě Java 3D vyřešit pomocí třídy **`PointAttributes`**, ve které je možné nastavit mj. velikost vykreslovaného pixelu (viz. část 4.2.4 - Vytvoření vlastností jednotlivých pixelů). Výpočet optimální velikosti zobrazovaného bodu. vychází z předpokladu, že se bude zpracovávat čtvercový obrázek a z faktu, že velikost oblasti na které je obrázek zobrazován je 256 x 256 pixelů.

Je také obvyklé, že počátek souřadného systému je ve středu oblasti pro zobrazování a také je to mnohem užitečnější kvůli přehlednosti při vykreslování a práci s obrázky. Proto je třeba vypočítat hodnoty posunutí na osách *x* a *y*, které zajistí že obrázek bude vykreslen na střed 3D oblasti. Obrázek se bude vykreslovat pomocí třídy **`Point3f`** od souřadnic **`(0.0f, 0.0f, 0.0f)`** do **`(1.0f, 1.0f, 0.0f)`**. Střed obrázku je určen polovinou jeho šířky a polovinou jeho výšky a je tedy bude v bodě **`(0.5f, 0.5f, 0.0f)`**. Z tohoto faktu vychází i výpočet proměnných.

Popis metody **`vypocty`**, která je určena pro čtení a normalizaci hodnot jednotlivých pixelů obrázku a převod na hodnoty vhodné pro vykreslení je v Technickém manuálu (kapitola 2.2).



Obr.4.5: *Vliv vzdálenosti pixelů na zobrazení: 384x384px (vlevo) a 128x128px (vpravo).*

4.2.3 Vytvoření geometrie jednotlivých pixelů

Metoda **`vytvorGeometrii`** se pro vykreslování jednotlivých pixelů musí změnit naprosto celá. Bude totiž použita třída **`PointArray`**, která je opět potomkem třídy **`GeometryArray`**.a je určena k vykreslování bodů v nadstavbě Java 3D. Do jejího konstruktoru se zadává celkový počet bodů které mají být vykresleny, dále se musí povolit zadávání koordinátů bodu a také se musí povolit zadávání barevného odstínu bodu. Samotné vykreslení pixelů se provede pomocí dvou smyček, přičemž jedna bude vykreslovat pixely v řádku zleva doprava po celé šířce obrázku a druhá bude posunovat tuto první smyčku shora dolů, tak aby byly vykresleny všechny řádky po celé výšce obrázku. V těle smyček je třeba zajistit správné indexování jednotlivých pixelů, tak aby systém vykreslování odpovídal systému načtení z tab.4.1. To je provedeno pomocí proměnných které slouží k řízení smyček (viz. Technický manuál kapitola 2).

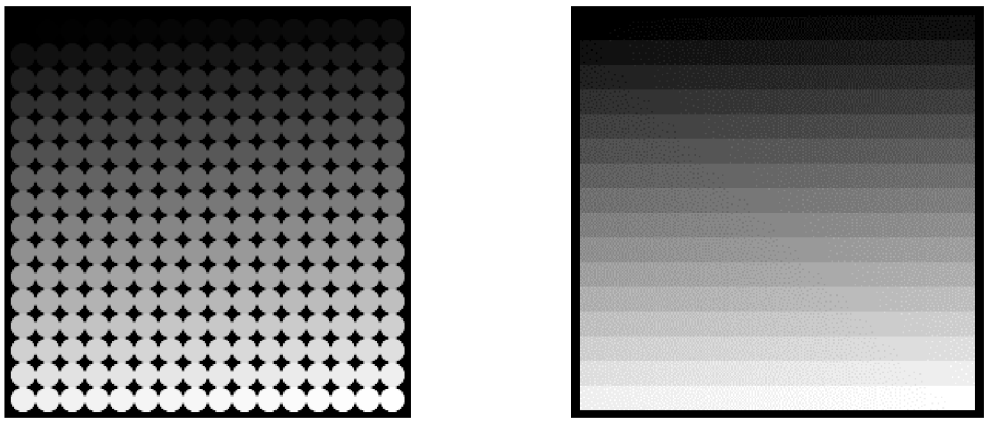
Topografický snímek, který bude zpracováván, je šedotónový. U šedotónových obrázků platí pro barevné složky: RR = GG = BB. Proto bude v každé barevné složce použita stejná hodnota. Toto ovšem platí pouze u šedotónových obrázků, takže při zobrazování obrázků barevných není možné tento algoritmus použít. Protože se vykresluje 2D obrázek v osách *x* a *y*, není souřadnice *z* prozatím využita a její hodnota je 0.

Popis metody která zajistí vykreslení jednotlivých pixelů v obrázku je v Technickém manuálu (kapitola 2.3).

4.2.4 Vytvoření vlastností jednotlivých pixelů

V metodě pro definování vlastností bodů **`vytvorVzhled`** je opět nejdříve vytvořen objekt třídy **`Appearance`**. Pro nastavení vlastností jednotlivých vykreslovaných bodů je zde využita třída **`PointAttributes`** (viz část 4.2.2 - Vytvoření vzhledu objektu). Pomocí této třídy je vytvořen objekt s názvem **`vlastnostiBodu`**, u kterého je možné nastavovat velikost zobrazovaných bodů a povolit, nebo zakázat antialiasing. Vliv povolení, nebo zakázání antialiasingu je demonstrován na obr.4.6, který zobrazuje body šedé škály. Samotné nastavení vlastností jednotlivých vykreslovaných bodů se provede pomocí metody **`setPointAttributes.`**

Popis metody, která slouží k definování vlastností bodů, je v Technickém manuálu (kapitola 2.4).



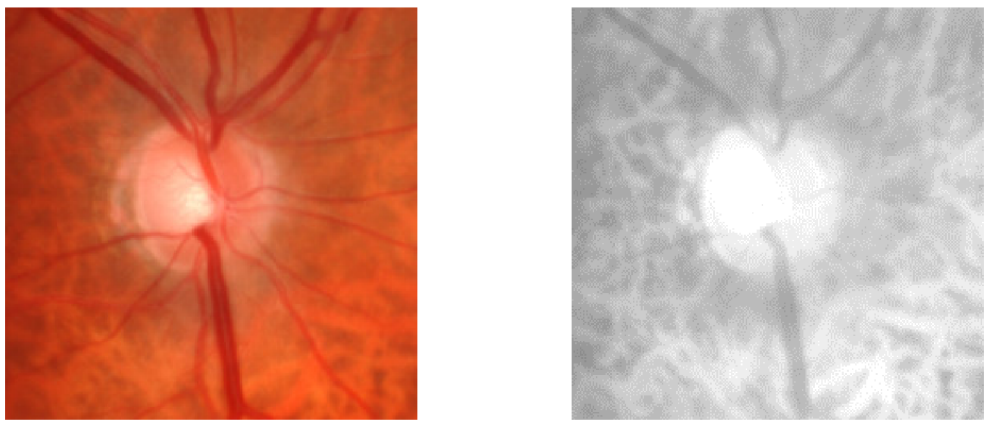
Obr.4.6: *Antialiasing povolen (vlevo), antialiasing zakázán (vpravo).*

4.2.5 Podmínky pro správné zobrazení jednotlivých pixelů

Všechny metody použité v appletu pro načtení jednotlivých pixelů obrázku s názvem **`VykresleniPixelu`** jsou uvedeny v Technickém manuálu (kapitola 2). Aby byl 2D obrázek ve 3D prostředí appletu správně zobrazen, musí však splňovat 3 podmínky:

1) obrázek musí být šedotónový – pokud by byl barevný, nebudou správně zobrazeny barvy,
2) obrázek musí mít čtvercový tvar (tzn. výška = šířka) – pokud je obdélníkového tvaru, dojde k prostorovému zkreslení (tzn. bude zobrazen jako čtverec),
3) musí se jednat o obrázek typu PNG nebo GIF.

Topografické snímky, které budou zpracovány v appletu, splňují všechny tyto podmínky. Problém by však nastal při zobrazování snímků z fundus kamery, které jsou kvůli použití laseru o vlnové délce 670 nm zabarveny do odstínu červené barvy. Zde by nebyly správně vypočteny barevné odstíny jednotlivých pixelů a obrázek by byl zobrazen chybně v odstínech šedé, tak jak je to zachyceno na obr.4.7.



Obr.4.7: *Obraz v originálních barvách (vlevo) a obraz s chybnou informací o barvě (vpravo)*

4.3 Vykreslení barevných odstínů

Barevné snímky z fundus kamery splňují podmínku čtvercového tvaru, jsou registrovány na intenzitní obraz HRT. Proto mají stejný rozměr 384 x 384 pixelů jako topografické snímky. Pro správné vykreslení jednotlivých pixelů v barevném obrázku je však nutné dekódovat jejich RGBA model – hexadecimální barevný model ve formátu PNG, popsaný v části 4.2.2.

Protože RGBA model se používá pouze u obrázků typu PNG a GIF, je také vhodné doplnit dialog pro načítání souborů o tzv. **`FileFilter`**, který zajistí že bude možné vybírat pouze soubory s předem definovanou koncovkou (v tomto případě soubory obrázků PNG). Byl tedy vytvořen applet s názvem **`VykresleniBarev`**, který vychází z appletu **`VykresleniPixelu`**. Do metody **`inicializaceKomponentu`** bude tedy třeba implementovat již zmíněnou třídu **`FileFilter`**. Dále bude třeba pozměnit metody **`vypocty`** a **`vytvorGeometrii`** tak, aby byly schopny pracovat s jednotlivými složkami barev v RGBA modelu. Ostatní metody zůstanou beze změny.

4.3.1 Filtr pro výběr souboru

V metodě **`inicializaceKomponentu`** je pro načtení obrázku použitý komponent **`JFileChooser`**. Aby bylo možné pomocí tohoto dialogu načítat pouze soubory s koncovkou PNG, je třeba vytvořit tzv. „filtr souborů“, který zajistí, že se v dialogu budou zobrazovat pouze soubory s definovanou koncovkou. K tomuto účelu slouží třídy **`FileFilter`** a **`FileNameExtensionFilter`**.

Pomocí metody **`setAcceptAllFileFilterUsed`** se zajistí, že se v dialogu načtení zobrazí pouze uživatelsky definované filtry a ne defaultní filtr, který umožňuje načtení všech souborů. Tento defaultní filtr je možné vidět na obr.4.3, kde je v dialogu načtení v řádku „Files of Type:“ uvedena hodnota „All Files“. V závorce konstruktoru třídy **`FileNameExtensionFilter`** se musí definovat popis a typ souborů zobrazovaných pomocí filtru k výběru. Tento popis a typ se pak zobrazí v dialogu načtení v řádku „Files of Type:“. Popis metody je v Technickém manuálu (kapitola 3.1).

4.3.2 Zobrazení obrázků obdélníkového tvaru

Není obtížné upravit applet tak, aby dokázal správně zobrazovat i obrázky obdélníkového tvaru. Postačí k tomu provést úpravu v metodě **`vypocty`**.

Nejdříve se musí větvením určit zda je větší rozměr šířky, nebo výšky obrázku. Pak se z většího rozměru vypočítá vzdálenost proměnná pro určení vzdálenosti bodů od sebe (převrácená hodnota rozměru). Dále se podle většího rozměru vypočítají proměnné pro posun obrázku na střed 3D oblasti, které byly popsány v části 4.2.2. Vypočítané proměnné pak budou použity v části 4.3.4 k vykreslení pixelů v metodě **`vytvorGeometrii`**`()`. Popis této úpravy v metodě **`vypocty`** je v Technickém manuálu (kapitola 3.2).

4.3.3 Dekódování barev z RGBA modelu

Aby mohly být v metodě **`vytvorGeometrii`** použity všechny 3 složky barvy pro správné zobrazení, musí se v metodě **`vypocty`** provést ještě úprava smyčky pro určení odstínu pixelů.

Pro dekódování jednotlivých složek barvy, byl vytvořen vlastní algoritmus který používá princip „maskování“ a pomocí něhož jsou všechny složky přesně dekódovány. Jak již bylo zmíněno v části 4.2.2, každá složka barvy RGBA modelu může nabývat hexadecimálních hodnot 0_{16} – FF_{16}, čemuž odpovídají dekadické hodnoty 0_{10} – 255_{10} (viz tab.4.2).

Tab.4.2: *Hodnoty odstínů barvy v RGBA modelu.*

Význam bitů (barevná složka):		RR	GG	BB	A
		červená	zelená	modrá	průhlednost
Hexadecimálně	Minimum	00	00	00	0
	Maximum	FF	FF	FF	8
Dekadicky	Minimum	0	0	0	0
	Maximum	255	255	255	8

Protože není třeba pracovat s atributem průhlednosti A, bude hodnota v modelu RGB pro bílou barvu $00_{16}00_{16}00_{16}$ a pro černou barvu $FF_{16}FF_{16}FF_{16}$. Tomu odpovídají dekadické hodnoty $00_{10}00_{10}00_{10}$ až $255_{10}255_{10}255_{10}$. Takto je třeba dekódovat celý řetězec RRGGBB. Ten ovšem nabývá hodnot 000000_{16} až $FFFFFF_{16}$, což dekadicky vyjádřeno znamená hodnoty 0_{10} až 16777215_{10}. Výstupní hodnoty z objektu **`PixelGrabber`** se kterými se v metodě pracuje jsou navíc posunuté a nabývají hodnoty -1_{10} až -16777216_{10}. Nejdříve je tedy nutné získat pomocnou hodnotu v rozsahu 0_{10} až 16777215_{10}, což se provede přičtením +1 k hodnotě odstínu pixelu z objektu **`PixelGrabber`** a zjištěním absolutní hodnoty tohoto čísla.

Pak se vypočítá barevný odstín červené RR a to tak, že se na celý řetězec RRGGBB aplikuje již zmíněná „maska“. Aplikace „masky“ spočívá v tom, že řetězec se dělí hodnotou s váhovým faktorem, který je určen mocnitelem dle pořadí hodnoty zprava. Zjednodušeně: hodnoty GGBB jsou maskovány nulami. Např. pokud v řetězci RRGGBB budou zapsány hodnoty $AA_{16}BB_{16}CC_{16}$, musí se pro výpočet hodnoty AA_{16} aplikovat maska s váhovým faktorem 4, takže její hodnota bude 10000_{16} (1). Protože se však v programu pracuje s dekadickými hodnotami, musí se celý výpočet provést v dekadické soustavě (2):

$$AABBCC_{16} / 10000_{16} = AA_{16}, \quad (1)$$

$$11189196_{10} / 65536_{10} = 170{,}73358154296875_{10}. \quad (2)$$

Hodnota dekadického základu 170_{10} z rovnice (2) odpovídá hexadecimální hodnotě AA_{16}. Po dělení však ještě vznikl zbytek, v němž jsou stále zakódovány informace o odstínech GG a BB. Pro výpočet barevného odstínu zelené GG se nyní musí zbytek po dělení nejdříve vynásobit hodnotou masky (3), čímž se získá zpět kódovaná hodnota řetězce s odstíny zelené a modré barvy, ovšem teď již bez hodnoty barvy červené (ta byla v základu). Posléze se musí na tuto hodnotu aplikovat opět odpovídající maska (4) Ta bude mít v tomto případě váhový faktor 2 a její hodnota tedy bude 100_{16} čemuž odpovídá dekadická hodnota 256_{10}:

$$0{,}73358154296875_{10} \cdot 65536_{10} = 48076_{10} = BBCC_{16} \quad (3)$$

$$48076_{10} / 256_{10} = 187{,}796875_{10} \quad (4)$$

Hodnota dekadického základu 187_{10} z rovnice (4) odpovídá hexadecimální hodnotě BB_{16}. Hodnota odstínu modré BB, se získá analogicky jako v předchozím případě: zbytek po dělení se násobí maskou (5) a váhový faktor masky je 0 (hodnota by se tedy měla dělit číslem 1):

$$0,\ 796875_{10} \cdot 256_{10} = 204_{10} = CC_{16} \quad (5)$$

Hodnota dekadického výsledku 204_{10} z rovnice (5) odpovídá přesně hexadecimální hodnotě CC_{16}. Takto je tedy možné přesně dekódovat RGBA model. Popis metody a algoritmus který odpovídá výše popsanému způsobu jsou v Technickém manuálu (kapitola 3.3).

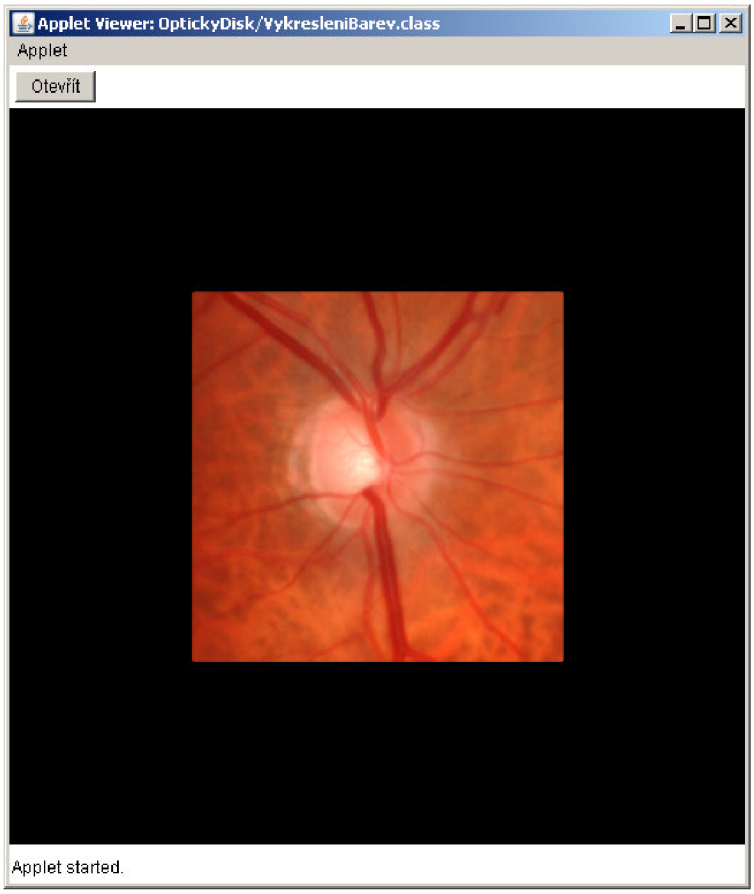
4.3.4 Vytvoření geometrie barevných pixelů

V metodě **`vytvorGeometrii`** nastanou dvě změny. První změna se týká vykreslení jednotlivých složek barvy pixelu. Tyto složky již byly vypočteny v metodě **`vypocty`** a musí být nyní zadány do konstruktoru objektu pro určení barvy, třídy **`Color3f`**:

Druhá změna se týká vykreslení obrázků, které nejsou čtvercového tvaru. V metodě **`vypocty`** již byly upraveny výpočty příslušných hodnot, takže místo normovaných hodnot šířky a výšky stačí dosadit do smyčky pro vykreslení vzdálenost dvou bodů. U osy *y* je nutné použít hodnotu zápornou, aby bylo dodrženo pravidlo směru vykreslování (vykresluje se směrem od nuly k záporným hodnotám). Popis celé metody pro vytvoření geometrie je v Technickém manuálu (kapitola 3.4).

4.3.5 Podmínky pro zobrazení barevných pixelů

Metody appletu s názvem **`VykresleniBarev`**, které slouží k zobrazování barevných obrázků, jsou uvedeny v Technickém manuálu (kapitola 3). Ukázka výstupu z tohoto appletu, tedy barevný obrázek, je na obr.4.8. Pro správné zobrazení je nyní již nutné dodržet pouze jedinou podmínku: načtený obrázek musí být ve formátu PNG, nebo GIF. Díky použití filtru souborů lze nyní do appletu načíst pouze obrázky typu PNG takže tuto podmínku není možné porušit. 2D obrázky mohou být jak šedotónové, tak i barevné a budou ve 3D prostředí zobrazeny správně, protože je použito dekódování RGBA modelu. Díky úpravám je zobrazení správné i v případě, že obrázek je obdélníkového tvaru. Nedojde tedy již k prostorovému zkreslení.



Obr.4.8: *Zobrazení barevného obrázku pomocí vykreslování jednotlivých pixelů.*

4.4 Interaktivita

Protože se předpokládá, že uživatel bude ovládat práci s obrázkem pomocí myši, byl vytvořen další applet s názvem **`Interaktivita`**, který vychází z předchozího appletu s názvem **`VykresleniBarev`**. Do tohoto appletu je třeba nejdříve implementovat příslušné třídy a jejich abstraktní metody pro práci s myší. První třídou je **`MouseListener`**, obsahující pět metod, které musí být povinně implementovány do appletu:

1. **`public void mouseClicked`**`(MouseEvent e)` – obsluhuje kliknutí myší,
2. **`public void mousePressed`**`(MouseEvent e)` – obsluhuje stisk tlačítka myši,
3. **`public void mouseReleased`**`(MouseEvent e)` – obsluhuje uvolnění tlačítka myši,
4. **`public void mouseEntered`**`(MouseEvent e)` – obsluhuje vstup kursoru do okna,
5. **`public void mouseExited`**`(MouseEvent e)` – obsluhuje opuštění kursoru z okna.

Druhou třídou je **`MouseMotionListener`**, obsahující dvě metody, které musí být povinně implementovány do appletu:

1. **`public void mouseDragged`**(`MouseEvent e`) – obsluhuje pohyb kursoru při stisknutém tlačítku na myši,
2. **`public void mouseMoved`**(`MouseEvent e`) – obsluhuje pohyb kursoru po obrazovce.

Třetí a poslední třídou je **`MouseWheelListener`**, která obsahuje jednu povinnou metodu, která musí být implementována do appletu:

`public void mouseWheelMoved`(`MouseWheelEvent e`) – obsluhuje kolečko myši.

Aby ovšem ovládání pomocí myši fungovalo, musí být příslušný grafický komponent (objekt třídy **`Canvas3D)`** přidán do metody obsluhy událostí (viz Technický manuál kapitola 4).

4.4.1 Otáčení obrázku pomocí myši

Pro otáčení obrázku je použita metoda **`mouseDragged`**(`MouseEvent e`), protože se předpokládá, že otáčení se bude provádět pohybem myši v příslušném směru 3D oblasti, při stisknutém tlačítku. Nejdříve se provede rozpoznání, které tlačítko na myši bylo stisknuto, protože otáčení se má provádět pouze levým tlačítkem, a pravé a prostřední tlačítko by mělo zůstat jako rezerva pro další funkce. Dále se pomocí metody **`getX`** (resp. **`getY`**) zjistí aktuální pozice kursoru myši na obrazovce a odečtením (resp. přičtením) poloviny šířky (resp. výšky) 3D prostoru (která je 256 pixelů), se posune počátek souřadného systému do středu tohoto 3D prostoru. Následně je třeba také zajistit, aby interakce probíhala pouze pokud je kursor myši ve 3D prostoru appletu. To je zajištěno tak, že se zjišťuje zda je kursor myši v oblasti ±256 pixelů od středu 3D prostředí v osách x i y. Pokud kursor myši opustí 3D oblast, je ovládání zakázáno:

Pro získání hodnot, které budou vhodné pro ovládání rotace se použijí dočasné hodnoty, do nichž bude uložena poslední známá pozice kursoru a které se budou zjišťovat při stisku tlačítka myši pomocí již zmíněných metod **`getX`** a **`getY`**. Protože hodnotu je třeba načítat již při stisknutí tlačítka myši je toto načtení provedeno v abstraktní metodě **`mousePressed`**. Dočasné hodnoty, které byly získány v metodě **`mousePressed`**, se pak budou porovnávat s aktuální pozicí v metodě **`mouseDragged`**.

Pokud jsou hodnoty dočasné hodnoty a aktuální pozice rozdílné, vyhodnotí se pohyb kursoru v příslušné ose. Následně je dočasná hodnota přepsána hodnotou nové pozice kursoru a povel k rotaci je inkrementován (resp. dekrementován) o zvolenou hodnotu. Tato hodnota inkrementace (resp. dekrementace) byla zvolena empiricky při testování appletu. Jako nejlepší se osvědčila hodnota $\pi / 90$, při které jeden krok rotace odpovídá úhlu 2°. Takto jsou získány povely k rotaci v obou směrech na osách x a y. Nakonec ještě musí být zavolána transformační metoda, která zajistí otáčení obrázkem (viz. část 4.4.2). Popis metody pro otáčení obrázku myší je uveden v Technickém manuálu (kapitola 4.1).

4.4.2 Transformační funkce

Funkce pro transformace obrázku jsou naprogramovány ve vlastní metodě s názvem **`transformujObraz`**. Pro otáčení obrázku, které je popsáno v části 4.4.1, se zde musí nejdříve vytvořit transformační funkce pro pohyby, kterými jsou objekty třídy **`Transform3D`**. Pro pohyb v každé ose musí být vytvořen jeden objekt této třídy. Pomocí metody **`rotX`** (resp. **`rotY`**) se do objektu zapíše hodnota pro řízení rotace, která způsobí pootočení objektu o úhel, který odpovídá této hodnotě.

Metodou **`rotX`** se provádí rotace roviny XY okolo osy x, což způsobuje že celá tato rovina se pohybuje (rotuje) ve směru osy y. Musí být tedy přiřazena k objektu jehož pohyb je řízen hodnotou pro rotaci ve směru osy y, jinak by ovládání pomocí myši bylo neergonomické. Totéž platí analogicky i pro metodu **`rotY`**. Aby byla zajištěna funkce otáčení v obou osách (x i y) zároveň, musí se použít metoda **`mul`**, pomocí které se oba objekty sloučí.

Nakonec se transformační funkce musí implementovat do transformační skupiny. Pomocí metody **`setTransform`** jsou sloučené objekty implementovány do transformační skupiny, která bude přidána do metody **`vytvorScenu`** (viz. část 4.4.3).Popis celé metody pro transformace obrázku je uveden v Technickém manuálu (kapitola 4.2).

4.4.3 Implementace transformačních funkcí do skupiny

Aby mohly být transformační funkce implementovány do transformační skupiny, musí se provést úpravy v metodě **`vytvorScenu`**.

V té musí být nejdříve vytvořena transformační skupina třídy **`TransformGroup`**. Vlastní implementace transformačních funkcí je pak provedena voláním metody **`transformujObraz`** a přidáním vykresleného objektu (na který mají být transformace aplikovány) do transformační skupiny. Zde je použitý objekt třídy **`Shape3D`**, který je pomocí konstruktoru **`addChild`** přidán do výše vytvořené transformační skupiny.

Původně byl objekt třídy **`Shape3D`** přidán přímo do objektu třídy **`BranchGroup`**. V tomto appletu už ale byl objekt třídy **`Shape3D`** přidán kvůli interaktivitě do transformační skupiny, takže aby se 3D scéna zobrazila a bylo možné s ní otáčet, musí se do objektu třídy **`BranchGroup`**, přidat celá transformační skupina. Popis celá upravená metoda je uvedena v Technickém manuálu (kapitola 4.3).

4.4.4 Změna měřítka obrázku

Pro změnu měřítka obrázku je nutné vytvořit v metodě **`transformujObraz`** novou transformační funkci a vytvořit objekt, pomocí kterého bude zvětšování (resp. zmenšování) obrázku prováděno. Pro samotnou změnu měřítka obrázku je využita metoda **`setScale`**, která je řízena proměnnou typu **`float`**. Počáteční hodnota proměnné je 1, protože tato hodnota zajišťuje, že obrázek bude zobrazen ve 100% velikosti. Aby bylo možné měnit měřítko obrázku v předpokládaném rozsahu 100 až 300%, musí tato proměnná nabývat hodnot 1,0 až 3,0. Objekt se pak musí sloučit s ostatními objekty transformačních funkcí. Stane se tak součástí transformační skupiny, která je na konci metody volána. Popis celé upravené metody pro transformace obrázku je uvedena v Technickém manuálu (kapitola 4.4).

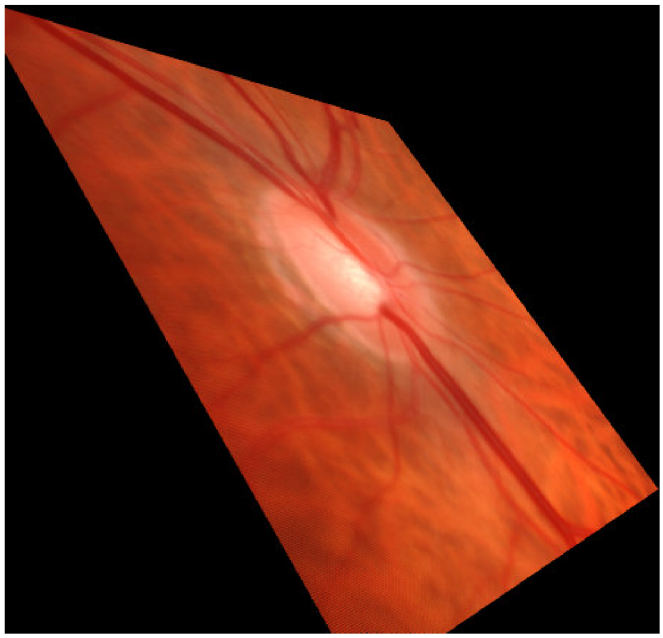
Pro zvětšování (resp. zmenšování) obrázku se předpokládá využití kolečka myši. Ovládací funkce pro změnu měřítka se tedy musí naprogramovat v abstraktní metodě **`mouseWheelMoved`**. Aby kolečko myši bylo funkční pouze v případě že je kursor na 3D pozadí, jsou využity abstraktní metody **`mouseEntered`** a **`mouseExited`**. V nich je nadefinována hodnota příznaku pro práci s kolečkem, který je pak možné použít v abstraktní metodě **`mouseWheelMoved`** pro určení zda bude kolečko myši pracovat, nebo ne.

Pro samotné ovládání měřítka obrázku, je prvním krokem zachycení události při pohybu kolečka myši pomocí proměnné **`e`**. Ta je při pohybu kolečka myši vpřed (resp. vzad) inkrementována (resp. dekrementována) o hodnotu 1. To je provedeno pomocí metody **`getWheelRotation`**. Tato proměnná ovšem nemůže být použita přímo pro řízení změny měřítka obrázku, protože se jedná o proměnnou typu **`integer`** a má také nevyhovující rozsah hodnot. V metodě **`transformujObraz`** je totiž pro řízení změny měřítka připravena proměnná typu **`float`**. Proměnná je tedy převedena na typ **`float`** a je upravena tak, aby její velikost byla vhodná k ovládání změny měřítka. Změna měřítka je nastavena o 5% při pootočení kolečka myši o jednu aretovanou polohu. Tato hodnota byla stanovena při testování programu a jevila se jako nejoptimálnější. Aby bylo možné měnit měřítko obrázku pouze v určitém rozsahu (zde 100 až 300%), je také stanovena maximální a minimální hodnota proměnné. Popis celé metody pro změnu měřítka obrázku je uvedena v Technickém manuálu (kapitola 4.4).

4.4.5 Podmínky pro interaktivní ovládání

Pro ovládání rotace a měřítka obrázku se musí dodržet pouze jedna podmínka, a to aby se hodnoty proměnných nedostaly mimo povolený rozsah (**`integer`** , **`float`**), což při normálním použití appletu není pravděpodobné. Applet má však ještě jeden nedostatek. Po ukončení práce s jedním obrázkem je v případě načtení další obrázek zobrazen s hodnotami rotace a měřítka, které byly naposledy použity u předchozího obrázku. V appletu je tedy provedena ještě jedna změna. Ta zabraňuje aby si applet „pamatoval“ poslední nastavení těchto hodnot. V metodě **`actionPerformed`** jsou nastaveny proměnné pro rotaci a změnu měřítka na inicializační hodnoty, čímž se tato chyba odstraní. Popis upravené metody **`actionPerformed`** je uveden v Technickém manuálu (kapitola 4.5).

Všechny metody appletu s názvem **`Interaktivita`**, které slouží k zobrazování barevných obrázků a jejich interaktivnímu ovládání, jsou uvedeny v Technickém manuálu (kapitola 4). Ukázka výstupu z tohoto appletu je na obr.4.9.



Obr.4.9: *Zobrazení barevného obrázku pomocí vykreslování jednotlivých pixelů.*

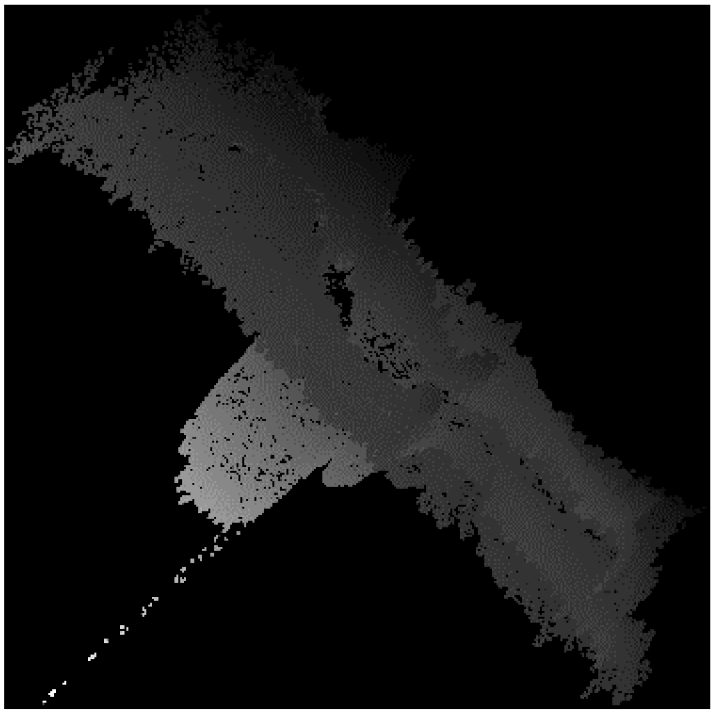
4.5 Trojrozměrné zobrazování

Jak již bylo popsáno v části 2.3.2, 3D zobrazení optického disku oka může zřetelně ukázat rozdíl mezi miskovitým tvarem normálního a glaukomatózního oka, kde je pohár mnohem hlubší a vykazuje vyšší strmost podél okrajů. Protože v topografických obrazech z výstupu HRT je zanesena informace o rozměru v ose *z* a ta odpovídá světlosti (resp. tmavosti) daného bodu, lze jednotlivé pixely v obrázku transformovat pomocí třídy **`Point3f`** o příslušnou vzdálenost po ose *z*. Proto byl vytvořen applet s názvem **`Zobrazeni3D`**, který vychází z předchozího appletu **`Interaktivita`**.

4.5.1 Transformace bodů v prostoru

Vzhledem k tomu že informace o rozměru v ose *z* je v šedotónových topografických obrazech zanesena v odstínu každého pixelu, využije se smyčka zpracování v metodě **`vypocty`** a smyčka vykreslení v metodě **`vytvorGeometrii`** z appletu **`VykresleniPixelu`**.

Protože proměnné z pole vytvořeného pomocí třídy **`PixelGrabber`** nabývají hodnot od 0,0 do 1,0 lze je použít přímo pro transformaci v závorce konstruktoru třídy **`Point3f`**. Ukázka výstupu z takto upraveného appletu je na obr.4.10.



Obr.4.10: *Transformace bodů ve 3D prostoru.*

Toto zobrazení má ovšem ještě několik nedostatků. Prvním z nich je, že se 3D model neotáčí kolem počátku v ose *z*. Důvodem je fakt, že rozměr v ose *z* se zobrazuje od počátku k záporným hodnotám a proto je rotace nesymetrická. Pro odstranění tohoto problému se musí nejdříve určit hodnoty nejsvětlejšího a nejtmavšího bodu v obrázku, což se provede v metodě **`vypocty`**. Posunutí 3D modelu na střed osy *z* se pak provede v metodě **`vytvorGeometrii`**. Upravená metoda **`vypocty`** je v Technickém manuálu (kapitola 5.1).

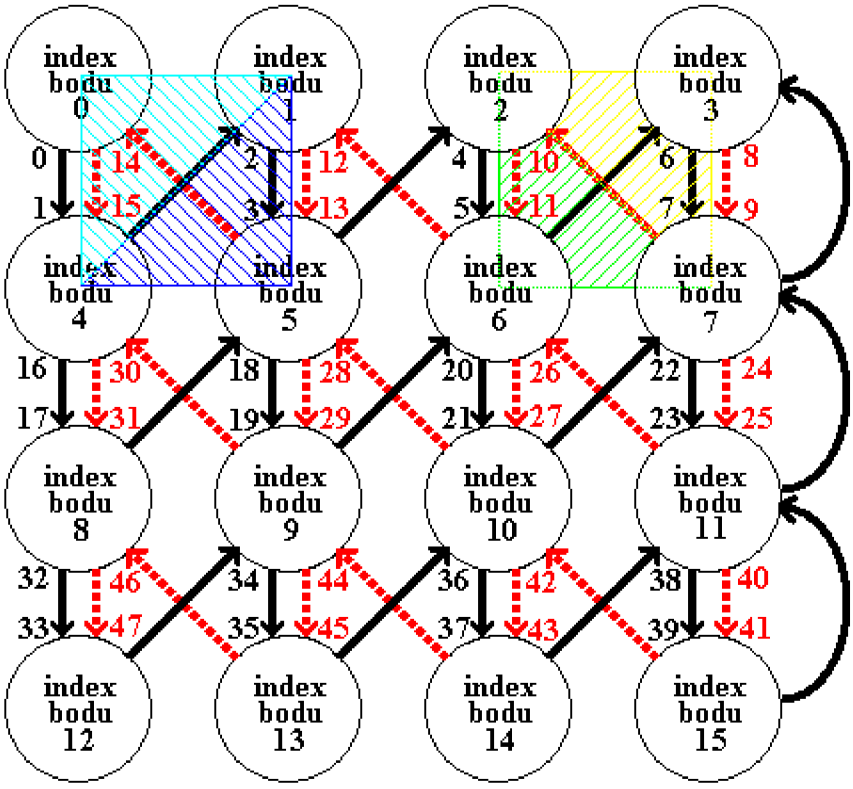
4.5.2 Zobrazení povrchu 3D modelu

Dalším nedostatkem zobrazení z části 4.5.1 je absence povrchu 3D modelu. Ve 3D prostředí jsou totiž zobrazeny pouze jednotlivé body v prostoru. 3D model je proto v některých částech transparentní a tak není zobrazení prohloubených (resp. vyvýšených) částí konkrétní. Pro generování povrchu 3D modelu se musí vytvořit nová metoda, která byla nazvána **`vytvorPovrch`**. Touto metodou bude nahrazena dosavadní metoda pro vykreslování bodů s názvem **`vytvorGeometrii`**. Také další metoda s názvem **`vytvorVzhled`** se nyní stává nepotřebnou, protože jsou v ní definovány vlastnosti bodů, které však již nyní nebudou vykreslovány.

V nové metodě s názvem **`vytvorPovrch`**, je pro zobrazení povrchu modelu vytvořen objekt třídy **`TriangleStripArray`**, který vykresluje pásy trojúhelníků. V konstruktoru je nutné mj. zadat kolik vrcholů trojúhelníků se bude celkem pro vykreslení používat a také pomocí jednorozměrného pole typu **`integer`** počet vrcholů trojúhelníků v jednom pásu, kvůli možnosti použití vykreslování ve více pásech. Protože použití více pásů pro vykreslení by bylo v tomto případě nepřehledné a také obtížnější, byla zvolena varianta kdy je vykreslován jediný pás a proto také toto pole bude mít stejný počet prvků jako je hodnota proměnné s počtem trojúhelníků. Před vytvořením vlastního algoritmu vykreslování, je však ještě nutné vypočítat počet vrcholů použitých trojúhelníků. Výpočet bude proveden v metodě **`vypocty`**, protože postačí zjistit tuto hodnotu pouze jednou před vykreslováním 3D modelu.

Výpočet počtu vrcholů trojúhelníků vychází z obr.4.11, kde je naznačen postup vykreslování jednotlivých trojúhelníků. Vrcholy trojúhelníků v horním a dolním řádku jsou pro vykreslení použity 2x a ostatní vrcholy uprostřed jsou pro vykreslení použity 4x. Na této skutečnosti je založen i výpočet celkového počtu vrcholů trojúhelníků v metodě **`vypocty`**.

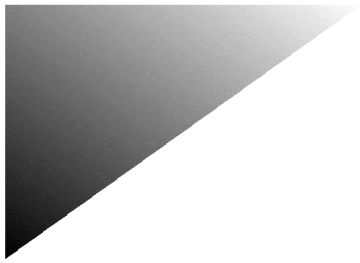
Při vykreslení pomocí trojúhelníkové sítě je nutné dodržet pořadí zadávání souřadnic, protože povrch je zobrazován na vnější straně 3D modelu směrem k uživateli pouze při tzv. „cik-cak“ zadávání souřadnic zleva doprava. Pokud by se souřadnice zadaly zprava doleva, byl by povrch vykreslen na vnitřní straně, kde je pro uživatele neviditelný. Z tohoto principu vychází i algoritmus pro vykreslování povrchu topografického obrázku. První část pásu je vykreslena na vnější straně. Pak se mění smysl zadávání souřadnic a vykreslení pokračuje na vnitřní straně. Po návratu na počáteční pozici, je opět změněn smysl zadávání souřadnic, pás je posunutý o řádek níže a vykreslí se opět vnější strana, tentokráte ale ve druhém řádku. Takto se pokračuje až se pás posune na předposlední řádek a vykreslí se poslední pás nejdříve z vnější strany a po změně smyslu zadávání souřadnic i z vnitřní strany. Po takto provedeném vykreslování je zajištěno, že 3D model bude pro uživatele viditelný z obou stran i při otáčení celým 3D modelem kolem počátku.



Obr.4.11: *Vykreslení povrchu 3D modelu pomocí trojúhelníkové sítě.*

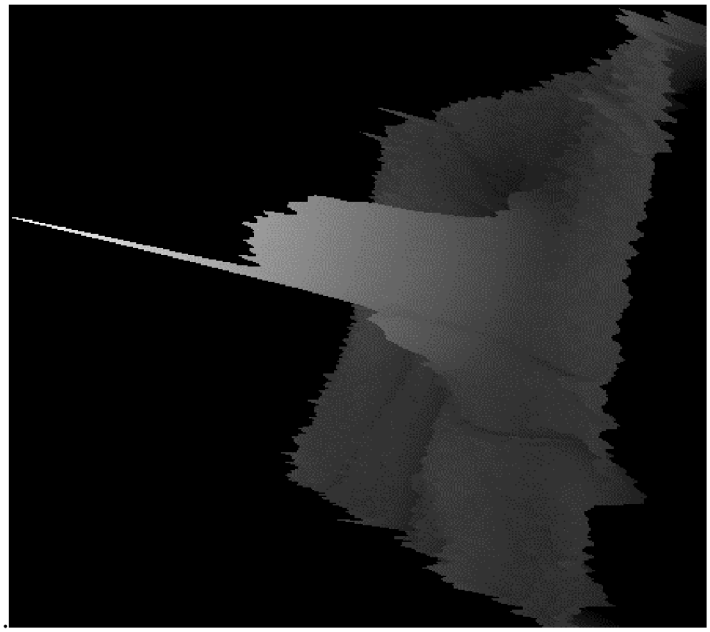
Celý systém vykreslování je uveden na obr.4.11. Je zde znázorněn obraz se 16-ti pixely, které mají indexy 0 – 15. Vykreslování začíná od bodu s indexem 0 ve směru černých šipek. U šipky je číslem naznačeno v jakém pořadí se budou body zadávat. U bodu s indexem 0 je počátek šipky s číslem 0, což znamená že bod je výchozí. Dále tato šipka pokračuje k bodu s indexem 4 a zde je u konce šipky číslo 1, což znamená že bod je zadán druhý v pořadí. Černá šipka se pak vrací k bodu 1 a na jejím konci je číslo 2, což značí že bod je zadán třetí v pořadí. Tím je vykreslen první trojúhelník, který má tyrkysovou barvu a je zobrazen na vnější straně, směrem k uživateli. Obdobně je jako čtvrtý v pořadí zadán bod 5. Tím se vykreslí trojúhelník modré barvy a tak se pokračuje až k bodu 7.

Od bodu s indexem 7 je nejdříve nutné vrátit se zpět k bodu 3, což je naznačeno oblou černou šipkou na boku. Nyní se změní smysl vykreslování a bude vykreslen povrch na vnitřní straně, která je pro uživatele skryta. To je naznačeno pomocí červených přerušovaných šipek. Od bodu 3, který je zadán jako osmý v pořadí (číslo 8 u počátku červené šipky) se pokračuje k bodu 7 (devátý v pořadí) a pak k bodu 2 (desátý v pořadí). Tím je na vnitřní straně vykreslen trojúhelník, který je v obr.4.11 naznačen žlutou barvou. Zadáním jedenáctého bodu v pořadí, tady bodu s indexem 6 se vykreslí zelený trojúhelník. Takto se pokračuje zpět až k bodu 4. Zde je opět změněn smysl vykreslování a od bodu 4 se začíná vykreslovat druhý pás s povrchem na vnější straně k uživateli, obdobně jako bylo provedeno vykreslování od bodu 0.



Obr.4.12: *Odstín barvy vykresleného trojúhelníku na povrchu 3D modelu.*

Barva trojúhelníku vykresleného na povrchu 3D modelu je dána odstínem barvy jednotlivých vrcholů. Trojúhelník tedy není jednobarevný, ale odstín barvy jednoho vrcholu přechází postupně do odstínu barvy dalšího vrcholu, tak jak je to naznačeno na obr.4.12, kde je použita barva horního pravého vrcholu: bílá, horního levého vrcholu: šedá, a dolního levého rohu: černá.

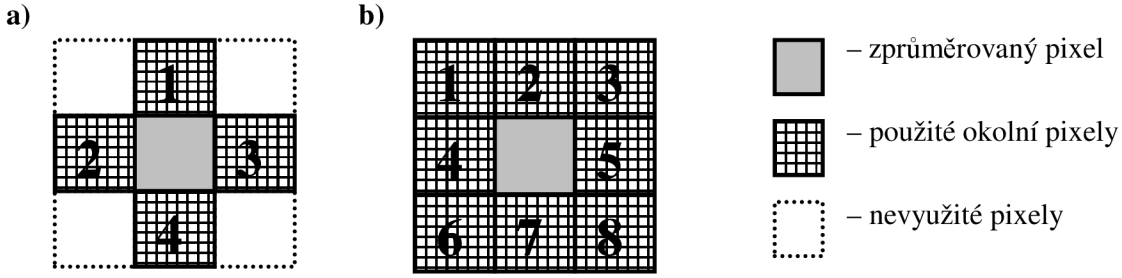


Obr.4.13: *3D model s vykresleným povrchem.*

Vlastní algoritmus vykreslování povrchu je opět proveden pomocí vnořených smyček, kdy dvě vnořené smyčky vykreslují souřadnice v celém řádku a vnější smyčka posunuje vykreslování na další řádky po celé výšce. Dvě vnořené smyčky jsou použity proto, že jedna řídí vykreslování na vnější straně 3D modelu směrem zleva doprava pomocí inkrementace a druhá řídí vykreslování na vnitřní straně 3D modelu směrem zprava doleva pomocí dekrementace. Popis celé metody **`vytvorPovrch`**() pro vykreslení povrchu 3D modelu je v Technickém manuálu (kapitola 5.2). Ukázka výstupu z takto upraveného appletu je na obr.4.13

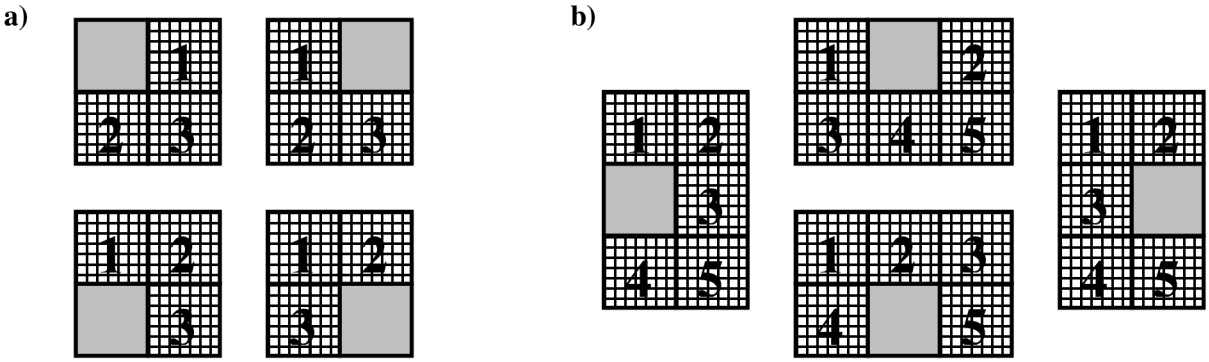
4.5.3 Vyhlazení hran ve 3D modelu

Nedostatkem 3D modelu z obr.4.13 jsou ovšem ještě viditelné vrcholy a hrany struktur ve 3D modelu. Pro jejich potlačení se používají metody vyhlazování. Byla tedy vytvořena nová vlastní metoda s názvem **`vyhlazeniPovrchu`**, ve které je pro vyhlazení hran vytvořen algoritmus, který využívá metodu zprůměrování hodnoty pixelu s ohledem na hodnoty okolních pixelů, a to v předem vymezené oblasti. Pro vyhlazení hran u menších 3D modelů vytvořených z obrázků do rozměru 128 x 128 pixelů je dostačující zprůměrovat hodnotu pixelu pomocí hodnot dalších čtyř okolních pixelů (obr.4.14a). Pro 3D modely vytvořené z obrázků s větším rozměrem se však tento algoritmus stává méně účinným a je tedy nutné definovat větší okolí. Pro topografický obrázek o rozměru 384 x 384 pixelů bylo tedy zvoleno zprůměrování hodnoty pixelu s ohledem na hodnoty všech osmi okolních pixelů (obr.4.14b).



Obr.4.14: *Vymezení oblasti pro zprůměrování hodnoty pixelu: a) 4 pixely, b) 8 pixelů*

Aby byla metoda pro vyhlazení povrchu účinnější, aplikuje se algoritmus vícekrát po sobě. V Příloze 1 jsou ukázky výstupu z appletu, který využívá k zprůměrování hodnoty čtyř okolních pixelů podle obr.4.14a. V Příloze 2 jsou ukázky výstupu z appletu, který využívá k zprůměrování hodnoty osmi okolních pixelů podle obr.4.14b. Z těchto obrázků je patrné, že při vícenásobném použití algoritmu, je pro vyhlazení povrchu obrázku 384 x 384 pixelů účinnější metoda, která využívá zprůměrování hodnoty pomocí osmi okolních pixelů. Algoritmus tedy vychází z vymezené oblasti na obr.4.14b. U této oblasti nastane problém při vyhlazování hodnot pixelů umístěných v rozích, resp. na vnějších stranách obrázku. K tomu se musí vymezit odlišné oblasti, které jsou na obr.4.15a, resp. 4.15b.



Obr.4.15: *Vymezení oblasti pro zprůměrování hodnoty pixelu: a) v rozích, b) na stranách*

Při testování appletu bylo zjištěno, že smysluplný počet opakování algoritmu, který je výše popsán, je pro výchozí obrázek o rozměru 384 x 384 pixelů maximálně 10x. To je patrné i z obrázků v Příloze 2, kde lze pozorovat největší účinek vyhlazení při pětinásobném opakování algoritmu, ovšem při jeho další aplikaci již nejsou změny vrcholů a hran struktur povrchu tolik patrné. Proto je počet paměťových polí, do nichž se budou ukládat „vyhlazené“ hodnoty a hodnoty posunu, stanoven na 10 a paměťová smyčka, pomocí níž se do polí budou hodnoty ukládat bude mít 10 cyklů.

Při vyhlazování povrchu jsou také znovu zjišťovány hodnoty nejtmavšího (resp. nejsvětlejšího) bodu po vyhlazení. Tyto hodnoty jsou zjišťovány, protože s postupným vyhlazováním povrchu se bude posunovat také střed 3D modelu. Hodnoty pak budou použity pro výpočet nového posunutí 3D modelu do středu souřadné soustavy. Hodnota posunutí se musí s každým krokem vyhlazení opravit, aby mohl 3D model rotovat kolem počátku souřadného systému i při změně svého tvaru příp. velikosti, kterou může vyhlazení způsobit.

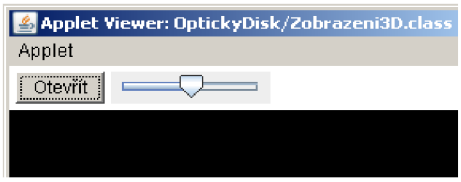
Protože metoda pouze ukládá hodnoty nutné pro vyhlazení povrchu do paměťových polí a nemusí tedy pracovat v reálném čase, je volání provedeno pouze jednou při načtení nového obrázku z metody **`vypocty`** a to ihned po získání normovaných hodnot pixelů, se kterými se při vyhlazování pracuje. Popis celé metody s názvem **`vyhlazeniPovrchu`**, pomocí které se provede vyhlazení povrchu 3D modelu, je v Technickém manuálu (kapitola 5.3).

4.5.4 Interaktivní ovládání vyhlazení povrchu

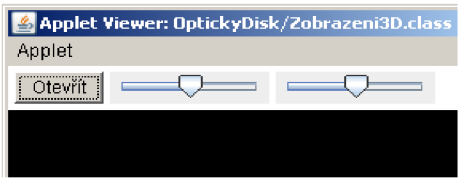
Aby bylo možné vyhlazování povrchu interaktivně ovládat, je do GUI přidán nový ovládací prvek: posuvník třídy **`JSlider`**, který je součástí knihovny **`swing`** (viz. obr.4.16). Posuvník je umístěn vedle tlačítka „Otevřít“ pro otevírání dialogu načtení.

Aby byla zajištěna funkčnost posuvníku, musí být implementována třída **`ChangeListener`**. Pomocí této třídy se zajistí funkčnost tohoto prvku a bude možné zjišťovat polohu jezdce posuvníku. U této třídy je vyžadována povinná implementace abstraktní metody **`stateChanged`**, ze které se posléze bude volat nová vlastní metoda s názvem **`vyhlad`**.

Metoda s názvem **`vyhlad`** je vytvořena z toho důvodu, aby mohla být volána jak z metody **`stateChanged`**, tak i po načtení obrázku z abstraktní metody **`actionPerformed`**, kvůli zobrazení 3D modelu s povrchem vyhlazeným dle inicializační hodnoty. Toto druhé volání metody je provedeno po ukončení všech výpočtů a uložení hodnot vyhlazeného povrchu do paměťových smyček. Na konci metody **`vyhlad`** je volání metody pro vytvoření 3D scény, které sem bylo přeneseno z abstraktní metody **`actionPerformed`**. Popis celé metody **`vyhlad`**, zajišťující interaktivní ovládání pro vykreslení povrchu 3D modelu je v Technickém manuálu (kapitola 5.4).



Obr.4.16: Posuvník pro vyhlazení povrchu.



Obr.4.17: *Posuvník pro nastavení hloubky 3D.*

4.5.5 Interaktivní ovládání hloubky 3D zobrazení

Podobně jako v části 4.5.4 je pro ovládání hloubky 3D zobrazení použitý posuvník. Posuvník je umístěn do GUI tak jak je to zachyceno na obr.4.17.

Obsluha událostí se opět musí naprogramovat v metodě **`stateChanged`**. Je zde vytvořena nová vlastní metoda s názvem **`modeluj`**, která bude sloužit pro nastavení 3D hloubky modelu. Důvodem jejího vytvoření je opět možnost aby mohla být volána jak z metody **`stateChanged`**, tak i po načtení obrázku z metody **`vyhlad`**. Volání pro vytvoření scény, se tedy nyní musí přesunout z metody **`vyhlad`** na konec nově vytvořené metody **`modeluj`**. Popis celé metody **`modeluj`**, která zajistí interaktivní ovládání pro nastavení 3D hloubky modelu je v Technickém manuálu (kapitola 5.5).

5 Realizace funkčního appletu

Řešení vlastního appletu vychází z již vytvořeného appletu s názvem **`Zobrazeni3D`** (viz. Technický manuál kapitola 6 a 7). Prvním řešením by měl být co nejjednodušší, uživatelsky přívětivý applet s velmi přehledným ovládáním. Po načtení obrázku by se měl v appletu za pomoci vhodně zvolených inicializačních hodnot automaticky vytvořit 3D model, který by bylo možné lehce ovládat a upravovat. Druhým řešením by měl být applet složitější, ve kterém by mělo být možné nastavovat větší množství hodnot pro ovládání i zobrazování.

5.1 Jednoduchý intuitivní applet

Pro návrh jednoduchého intuitivního ovládání byl vytvořen applet s názvem **`OptickyDisk1`**, který vychází z předchozího appletu s názvem **`Zobrazeni3D`**. V první řadě bude navrženo jednoduché intuitivní GUI, které by mělo být co nejpřehlednější a mělo by umožňovat jednoduchou obsluhu appletu. Dále bude navržena vhodná míra interaktivity, aby mohl uživatel applet lehce ovládat. Nakonec by měl být návrh otestován a případné nedostatky by měly být opraveny a odstraněny.

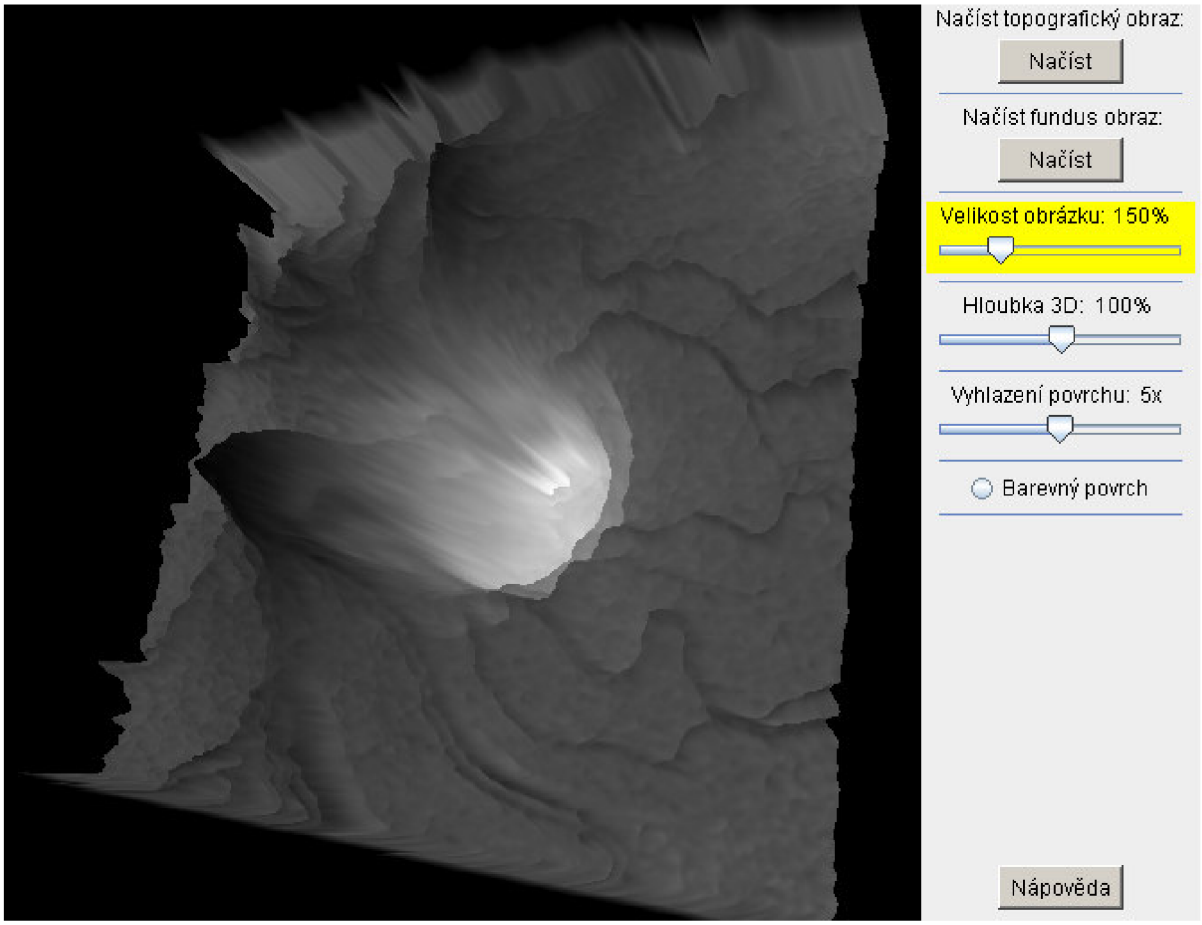
Aby ovládání appletu bylo ergonomické, přehledné a intuitivní, mělo by být co nejvíce hodnot nastaveno automaticky. V GUI ovšem musí být některé prvky, které nemohou být zautomatizovány a uživatel s nimi musí pracovat. Mělo by jich ovšem být co nejméně a měly by být snadno dostupné a ovladatelné. Návrh vychází z předpokladu, že uživatel bude applet ovládat především pomocí myši pravou rukou. Ovládací prvky by tedy měly být z pohledu uživatele na pravé straně okna. Nutnými komponenty jsou tlačítka která budou vyvolávat dialogy nabídky pro načtení šedotónového topografického obrázku, podle kterého se bude řídit hloubka 3D zobrazení, a barevného obrázku odrazivosti, který bude použitý jako textura pro zobrazení povrchu. Další komponenty v GUI budou posuvníky s přednastavenými inicializačními hodnotami, které však po načtení bude možné měnit. Posledním komponentem bude volba, využívající třídu **`JRadioButton`**. Pomocí této volby bude možné přepínat mezi zobrazením šedotónovým a barevným. Každý komponent bude mít příslušný popis a u komponentů pro nastavování hodnot, budou displeje se zobrazením aktuální hodnoty. Otáčení 3D modelu bude řešeno pomocí levého tlačítka myši, tak jak to je navrženo v předchozím appletu, stejně jako změna velikosti (tzn. přiblížení – oddálení), která bude řešena pomocí kolečka myši. Nově zde bude možnost ovládat kolečkem myši i posuvníky, mezi nimiž bude možné přepínat pomocí pravého tlačítka myši. Navíc stiskem a držením pravého tlačítka myši (funkce dragged) bude možné posunovat celý vykreslený model po ploše 3D pozadí.

5.1.1 Návrh GUI

Nejdříve je třeba implementovat některé nové komponenty, které v appletu s názvem **`Zobrazeni3D`** doposud nebyly:

- druhý dialogy nabídky, který bude sloužit k načítání obrázků z fundus kamery,
- komponent třídy **`JRadioButton`**, který se bude využívat k přepínání zobrazení povrchu 3D modelu (šedotónový/barevný),
- tlačítko třídy **`Button`**, které bude sloužit k zobrazení okna s nápovědou,
- posuvník třídy **`JSlider`**, který se bude používat paralelně s kolečkem myši k nastavování změny velikosti (tj. přiblížení/oddálení) obrázku,
- komponenty třídy **`Label`**, které mají v názvu předponu **`stav-`** budou použity jako displeje k zobrazení hodnot nastavení (tj. velikosti vyhlazení povrchu, 3D rozměru v ose *z*, přiblížení/oddálení) a ty které mají v názvu předponu **`popis-`** budou použity pro popis komponentů,
- pro oddělení komponentů v okně se použijí oddělovače třídy **`Jseparator.`**

Dojde tedy k několika změnám v metodě **`inicializaceKomponentu`**. Nejdříve musí být načteny nové komponenty. Pak budou pomocí metody **`setLabel`** do těchto komponentů vepsány nápisy, které budou vyjadřovat jejich funkci. Pro upřesnění funkce ostatních komponentů budou vytvořeny další pomocné nápisy pomocí třídy **`Label`**. U posuvníků je třeba pomocí metod **`setValue`** resp. **`setMaximum`** nastavit inicializační hodnoty jezdců resp. rozsahy posuvníků. Musí být také nastaven filtr souborů u dialogu nabídky pro načítání obrázku z fundus kamery, aby bylo možné načítat pouze obrázky typu PNG. Také musí být nové tlačítko, dialog nabídky, nápověda, ovladač velikosti zobrazení i komponent třídy **`JRadioButton`** a jeho popis, přidány do příslušných metod pro zpracování událostí. Dalším krokem je zadávání souřadnic u všech nově vytvořených komponentů, které určují kde v okně appletu mají být tyto komponenty umístěny. Po zadání souřadnic je ještě nutné každý komponent vložit do panelu, ve kterém má být zobrazován. Posledním krokem je obarvení komponentů tak, aby měly všechny světle šedou barvu pozadí, jakou mají posuvníky. Popis metody je v Technickém manuálu (kapitola 6.1). Návrh GUI je zobrazen na obr.5.1.



Obr.5.1: *Návrh GUI appletu s názvem OptickyDisk1.*

5.1.2 Návrh interaktivity

Protože transformace bodů v prostoru je určena z informací uložených v topografickém obrázku, musí být v první řadě načten tento obrázek, jinak by nemohl být vykonáván posun pixelů obrázku do prostoru v ose *z*. K tomu je využito tlačítko a metoda **`actionPerformed`** z předchozího appletu **`Zobrazeni3D`**. Pro načtení obrázku z fundus kamery bude použito další tlačítko a musí být do metody **`actionPerformed`** dopsána obsluha tohoto tlačítka a následně vyvolaného dialogu. Na konci této obsluhy dialogu pro načtení obrázku z fundus kamery je volána nová vlastní metoda s názvem **`cteniBarvy`**, kde budou získány informace o barvě povrchu obrázku. Tato metoda slouží k načtení obrázku z fundus kamery a bude diskutována v části 5.1.3. V metodě **`actionPerformed`** je také provedena obsluha komponentu třídy **`JRadioButton`**. Poslední úpravou metody je obsluha

tlačítka nápovědy. Celé okno nápovědy i s textem je uvedeno v Příloze 3. Tím je celá metoda s názvem **actionPerformed** upravena. Její popis je uveden v Technickém manuálu (kapitola 6.2).

Další metodou zajišťující interaktivitu je metoda **stateChanged**. Ta byla opět již naprogramována v appletu **Zobrazeni3D**. Do algoritmu pro nastavení vyhlazení resp. 3D hloubky, bude pouze přidán příkaz pro zobrazení aktuální hodnoty na displeji. Musí být také ještě přidána metoda pro ovládání změny velikosti obrázku. Tím je celá metoda s názvem **stateChanged** upravena. Její popis je uveden v Technickém manuálu (kapitola 6.2).

Aby bylo možné ovládat applet pouze myší, bude použita doposud nevyužitá abstraktní metoda **mouseClicked**. V té bude naprogramováno ovládání posuvníků kolečkem myši, ovládání komponentu třídy **JRadioButton** pomocí kliknutí na jeho upřesňující popis a také přepínání mezi jednotlivými posuvníky, k čemuž bude využito pravé tlačítko myši. Bude využita metoda **getButton**, pomocí níž lze zjistit na které tlačítko myši bylo kliknuto. Podle navrácené hodnoty se zjistí zda se jedná o levé tlačítko (hodnota 1), nebo o pravé tlačítko (hodnota 3). Levým tlačítkem se bude ovládat pouze přepínání komponentu třídy **JRadioButton** (i kliknutím na popis tohoto komponentu). Kliknutím na pravé tlačítko myši bude realizováno přepínání mezi jednotlivými posuvníky. Protože je do obsluhy událostí myši **MouseListener** přidán i popis komponentu třídy **JRadioButton**, je třeba jej eliminovat, protože přepínání mezi posuvníky by jinak fungovalo i při kliknutí na tento komponent. Samotné rozpoznání, přepínání a obarvení posuvníků se bude provádět v nové metodě s názvem **prepni**. Popis celé metody s názvem **mouseClicked** je uveden v Technickém manuálu (kapitola 6.2).

V nové metodě s názvem **prepni** je provedeno rozpoznání, přepínání a obarvení posuvníků. Posuvník, který lze ovládat pomocí kolečka myši, je obarven žlutou barvou. Defaultně je to posuvník pro ovládání změny velikosti měřítka modelu. Pokud je rozpoznáno kliknutí na pravé tlačítko, je provedeno obarvení následujícího komponentu. Při prvním kliknutí je obarven posuvník pro změnu hloubky 3D. Při třetím kliknutí je obarven posuvník pro vyhlazení povrchu. Při dalším kliknutí je obarven opět posuvník pro změnu velikosti měřítka. Takto lze mezi posuvníky přepínat pravým tlačítkem, je-li kursor v oblasti pro 3D zobrazováni. Popis celé metody s názvem **prepni** je uveden v Technickém manuálu (kapitola 6.2).

Poslední úprava nastane v metodě pro ovládání kolečkem myši s názvem **mouseWheelMoved**. Opět se vychází ze stejné metody naprogramované v předchozím appletu **Zobrazeni3D**. Ovšem ihned po získání hodnoty navrácené při pohybu kolečka je provedeno volání nové vlastní metody s názvem **kolecko**, která jako argument bude posílat hodnotu proměnné navrácené při pohybu kolečka. Popis celé upravené metody **mouseWheelMoved** je v Technickém manuálu (kapitola 6.2).

Do metody **kolecko** se přesune algoritmus pro změnu velikosti obrázku, který byl původně v metodě **mouseWheelMoved**. V tomto algoritmu ovšem bude provedeno několik dalších úprav. Musí se zde totiž nejdříve zjistit, který komponent je aktivní pro ovládání kolečkem myši. Z posuvníku se pak pomocí metody **getValue** přečte aktuální hodnota, která je přepočítána tak aby nabývala hodnot, které jsou vhodné k řízení transformace (zde např. rozsahu **1.0f** až **3.0f**). Analogicky jsou vytvořeny také algoritmy, které ovládají hloubku 3D a vyhlazení povrchu, ovšem s tím že nakonec je volána příslušná metoda pro zpracování (**modeluj**, resp. **vyhlad**). Dále je již použitý původní kód z předchozího appletu **Zobrazeni3D** a na závěr se také volá metoda **transformujObraz** tak jako v původním algoritmu.Popis celé nové metody **kolecko** je uveden v Technickém manuálu (kapitola 6.2).

Pro posun 3D modelu po ploše v ose *x* a *y* bude využito stisknutí a držení pravého tlačítka myši. Pro tuto funkci se musí upravit kód v metodě **mouseDragged**. Původní kód zůstane zachován, ale musí být přeskupen (viz. Technický manuál kapitola 6). V novém kódu se nejdříve zjistí zda je posun 3D modelu po ploše povolen. Dále se s využitím dočasných hodnot určí směr posunu. Následně jsou o hodnotu **0.01f** inkrementovány resp. dekrementovány proměnné typu **float**, které řídí posun. Hodnota **0.01f** upravuje rychlost posunování tak, aby odpovídala rychlosti pohybu kurzoru po ploše. Pomocí testování posunu, byl tento koeficient rychlosti stanoven právě na 0,01. Hodnoty proměnných

pro posun jsou pomocí metody **`set`** zapsány do vektoru třídy **`Vector3f`**, který bude v metodě **`transformujObraz`** řídit posun. Popis celé upravené metody s názvem **`mouseDragged`** je uveden v Technickém manuálu (kapitola 6.2).

Do metody **`transformujObraz`** je následně přidána transformační funkce, která bude posunování 3D modelu po ploše ovládat pomocí metody **`setTranslation`** a která bude přidána do transformační skupiny metodou **`mul`**. Také do metody **`vytvorScenu`** musí být přidán řádek, který zajistí nastavení hodnot vektoru třídy **`Vector3f`** který řídí posun, aby i při načtení obrázku byl vektor definován. Popis obou upravených metod s názvem **`transformujObraz`** a **`vytvorScenu`** je uveden v Technickém manuálu (kapitola 6.2),

Posledním krokem je úprava metod **`mousePressed`** a **`mouseReleased`**. V nich se musí zajistit, aby posun 3D modelu po ploše byl možný pouze při stisku pravého tlačítka myši. Popis metod s názvem **`mousePressed`** a **`mouseReleased`** je v Technickém manuálu (kapitola 6.2).

Applet je nyní navržen tak, bylo možné ovládat jej pouze pomocí třítlačítkové myši se scrollovacím tlačítkem. Pokud by však bylo nutné z jakéhokoliv důvodu ovládat applet bez pomoci myši, musí se implementovat ovládání pomocí klávesnice, což zajišťují metody třídy **`KeyListener`**. To se provede zápisem za klíčové slovo **`implements`** k ostatním interaktivním metodám v deklaraci veřejné třídy. Povinné metody pro obsluhu událostí ve třídě **`KeyListener`** jsou: **`keyTyped`**, **`keyPressed`** a **`keyReleased`**. Aby ovládání pomocí kláves bylo vždy funkční, musí se přidat do obsluhy událostí komponenty, mezi nimiž se přepíná pomocí klávesy Tab (tj. tlačítka a 3D pozadí).

Metody **`keyTyped`** a **`keyReleased`** zůstanou nevyužité a programovat se bude pouze metoda **`keyPressed`**. Pomocí metody **`getKeyCode`** se zjistí kód stisknuté klávesy (viz. tab. 5.1). Pak budou klávesy rozděleny do skupin, které volají stejnou metodu pro zpracování.

***Tab.5.1**: Kódování kláves*

Klávesa:	PgUp	PgDn	End	Home
Kód:	33	34	35	36
Funkce:	posun nahoru	posun dolů	posun doprava	posun doleva

Klávesa:	←	↑	→	↓
Kód:	37	38	39	40
Funkce:	rotace doleva	rotace nahoru	rotace doprava	rotace dolů

Klávesa:	Space	Q	W
Kód:	32	81	82
Funkce:	přepínání	posuvník min.	posuvník max.

Rotace a posunování modelu jsou ovládány pomocí příslušných transformačních funkcí, které jsou v metodě **`transformujObraz`**. První skupinu tedy tvoří klávesy s kódem 33 až 40, které ovládají rotaci a posunování modelu ve 3D prostředí, protože volají metodu **`transformujObraz`**. Druhou skupinu tvoří klávesy s kódem 81 (Q) a 82 (W). Ty volají metodu pro ovládání parametrů pomocí kolečka myši **`kolecko`**. Pro tyto klávesy je vytvořen algoritmus, který slouží jako emulace kolečka myši pomocí dvou kláves. Poslední je klávesa s kódem 32, Space (mezerník). Tato klávesa volá metodu **`prepni`**, protože je určena pro přepínání mezi posuvníky. Popis celé metody s názvem **`keyPressed`** je v Technickém manuálu (kapitola 6.2).

5.1.3 Další úpravy appletu

Úpravy zdrojového kódu musí být provedeny také v dalších metodách. Aby bylo možné používat komponent třídy **`JRadioButton`**, který bude využíván k přepínání zobrazení povrchu 3D modelu (šedotónový/barevný), musí se provést úprava v metodě pro vykreslení povrchu obrázku s názvem **`vytvorPovrch`**. V předchozím appletu **`Zobrazeni3D`**, byly vykreslovány pixely pouze v odstínech šedé. Nyní je metoda upravena tak, aby bylo možné volit mezi vykreslením pixelů

v odstínech šedé, nebo pomocí tří barevných složek ve formátu RGB. Popis celé upravené metody s názvem **`vytvorPovrch`** je uveden v Technickém manuálu (kapitola 6.3).

Další změna je v metodě **`vypocty`**, kde je zajištěno, že při načítání topografického obrázku bude tento obrázek vždy zobrazen. To poslouží uživateli jako kontrola, že byl obrázek opravdu načten. Popis celé upravené metody s názvem **`vypocty`** je uveden v Technickém manuálu (kapitola 6.3).

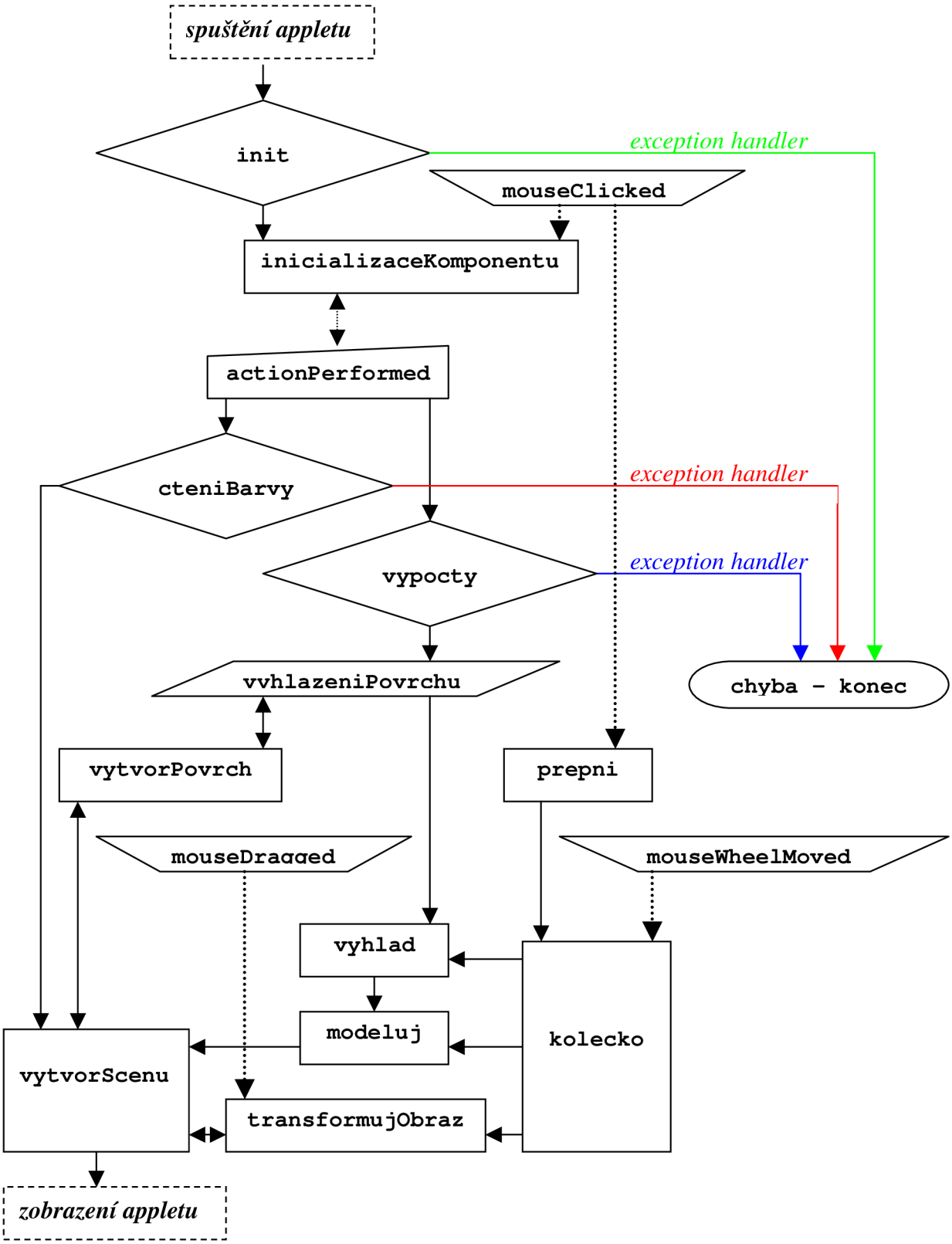
Nakonec bude vytvořena nová vlastní metoda s názvem **`cteniBarvy`**, která je volána z abstraktní metody **`actionPerformed`** (viz. část 5.1.2). Zde budou získány informace o barvě povrchu obrázku. Tato metoda je založena na metodě **`vypocty`** z appletu **`VykresleniBarvy`**. Rozdíl je v tom, že proměnné mají na konci názvu připsáno písmeno F, aby bylo jasné, že se jedná o práci s obrázkem z fundus kamery. Dalším rozdílem je, že i v případě nenačtení topografického obrázku je obrázek z fundus kamery zobrazen, aby měl uživatel kontrolu že obrázek byl opravdu načten. Popis celé metody s názvem **`cteniBarvy`**, je uveden v Technickém manuálu (kapitola 6.3).

V tab.5.2 je přehled všech metod použitých v appletu OptickyDisk1. Je zde provedeno rozdělení na metody převzaté z knihoven jazyka Java (využité a povinné nevyužité) a na metody vlastní, které byly pro applet vytvořeny v této práci.

Tab.5.2: *Přehled metod použitých y appletu OptickyDisk1.*

Název metody	Popis funkce
Převzaté metody:	
public void init()	inicializace appletu
public void actionPerformed(ActionEvent e)	vyvolání akcí
public void mouseDragged(MouseEvent e)	pohyb se stisknutým tlačítkem myši
public void mousePressed(MouseEvent e)	stisk tlačítka myši
public void mouseReleased(MouseEvent e)	událost při uvolnění tlačítka myši
public void mouseEntered(MouseEvent e)	je-li mys na plátně, pak je funkční
public void mouseExited(MouseEvent e)	je-li mys mimo plátno, pak nefunguje
public void mouseWheelMoved(MouseWheelEvent e)	událost při otočení kolečka myši
public void mouseClicked(MouseEvent e)	kliknuti tlačítkem myši
public void stateChanged(ChangeEvent e)	událost při změně jezdce posuvníku
public void keyPressed(KeyEvent e)	ovládání pomocí kláves
Převzaté metody - nevyužité:	
public void keyTyped(KeyEvent e)	nevyužito
public void keyReleased(KeyEvent e)	nevyužito
public void mouseMoved(MouseEvent e)	nevyužito
public void start()	nevyužito
public void stop()	nevyužito
public void destroy()	nevyužito
Vlastní vytvořené metody:	
public void inicializaceKomponentu()	vytvoření GUI
private void cteniBarvy()	zpracování barevného obrázku
private void vypocty()	zpracování šedotónového obrázku
private void vyhlazeniPovrchu()	funkce pro vyhlazení
private void prepni()	přepínání mezi posuvníky
private void kolecko(int rotace)	změna parametrů kolečkem myši
private void vyhlad(int v)	ovládání vyhlazení
private void modeluj(int m)	ovládání 3D hloubky
private void transformujObraz()	pohyb obrázkem
private Geometry vytvorPovrch()	vykreslení povrchu
private BranchGroup vytvorScenu()	implementace scény

Na obr.5.2 je pomocí vývojového diagramu znázorněn zjednodušeně běh appletu OptickyDisk1. Do obrázku nejsou kvůli přehlednosti zahrnuty metody, které jsou využívány pouze pro načítání aktuální pozice kursoru, nebo metody které omezují činnosti při umístění kursoru mimo applet. Podrobný popis vývojového diagramu je v Technickém manuálu (kapitola 6.4).



Obr.5.2: *Schéma běhu appletu OptickyDisk1.*

5.2 Rozšířené možnosti appletu

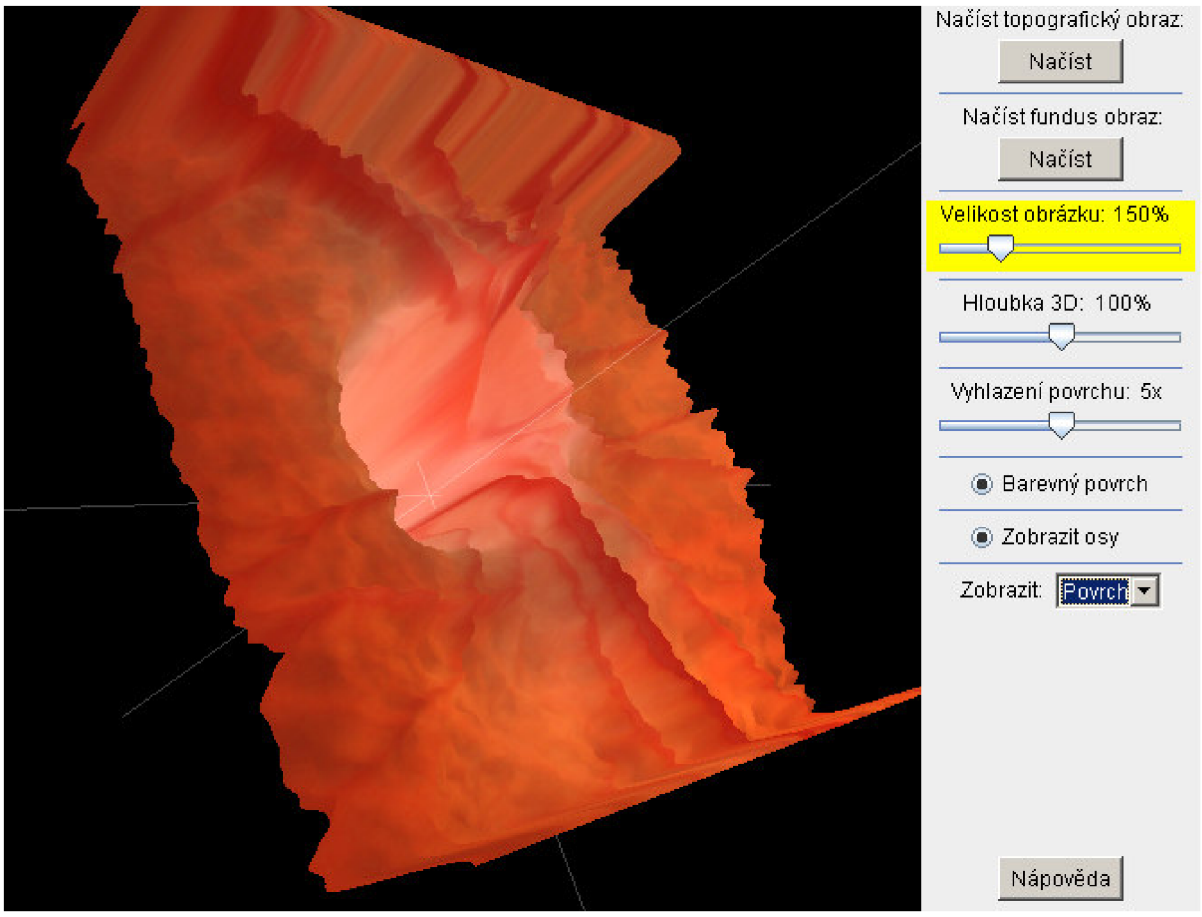
Applet **`OptickýDisk1`** vytvořený v části 5.1 je koncipován tak, aby byl jednoduchý a intuitivní, a obsahuje tedy pouze základní funkce a nastavení nutné pro práci s topografickým obrázkem. V nadstavbě Java 3D je však možné naprogramovat i mnoho dalších funkcí, které však nebyly do appletu **`OptickýDisk1`** přidány, protože by se applet stal složitějším a mohla by být snížena jeho přehlednost a tím pádem také komfort ovládání. Byl tedy vytvořen ještě druhý applet s názvem **`OptickyDisk2`**, který vychází z appletu **`OptickýDisk1`**. Tento applet je rozšířen o některé další funkce a nastavení, které jeho použití dělají univerzálnějším a zajímavějším. V rozšířeném appletu bude navíc možné zobrazovat osy souřadného systému *x*, *y* a *z*. Dále bude možné vykreslení jednotlivých pixelů pomocí bodů, úseček, nebo sítě. Bude také možné vytvořit osvětlený 3D model.

5.2.1 Rozšíření GUI

Protože applet bude obsahovat více ovládacích prvků, bude nutné přidat tyto prvky do GUI:

- zobrazování os bude ovládáno pomocí komponentu třídy **`JRadioButton`**,
- popisy budou vytvořeny pomocí dalších komponentů třídy **`Label`**,
- další objekt třídy **`JSeparator`** pro oddělení komponentů,
- ovládání voleb vykreslení bude ovládáno pomocí komponentu třídy **`Choice`**.

Aby byla zajištěna funkčnost komponentu třídy **`Choice`**, musí být implementována třída **`ItemListener`**. To se provede zápisem za klíčové slovo **`implements`** k ostatním interaktivním metodám. K této třídě je vyžadována povinná implementace abstraktní metody **`itemStateChanged`** jejíž naprogramování bude provedeno v části 5.2.2.



Obr.5.3: *Návrh GUI appletu s názvem OptickyDisk2.*

V metodě **inicializaceKomponentu** je nyní třeba komponenty načíst a popsat. Do komponentu třídy **Choice** je metodou **insert** vepsán text popisu volby a pak index, pod kterým bude příslušná volba volána. Je také nastaveno, že po načtení appletu bude na komponentu třídy **Choice** zvolena volba s indexem 0 tj. vykreslení jednotlivých bodů. Je také nutné nastavit koordináty, které určí kam budou komponenty umístěny a také se musí komponenty přidat do ovládacího panelu. Komponenty které mají defaultně bílé pozadí, musí být navíc obarveny a komponenty které vyžadují interaktivní ovládání, musí být přidány do příslušné metody obsluhy událostí. Tím je úprava metody dokončena. Popis celé upravené metody je v Technickém manuálu (kapitola 7.1). Návrh GUI je zobrazen na obr.5.3.

5.2.2 Rozšíření interaktivity

Nejdříve bude ošetřena možnost přepnutí komponentu **JRadioButton** pomocí kliknutí levým tlačítkem myši na popis u tohoto komponentu. To bude provedeno v metodě **mouseClicked** analogicky jako u popisu komponentu **JRadioButton** v části 5.1.2, pomocí metody **doClick**. Popis celé metody je v Technickém manuálu (kapitola 7.2).

Dále bude provedena obsluha událostí při změně stavu komponentu **JRadioButton**. To bude provedeno v metodě **actionPerformed**. Nejdříve se bude pomocí metody **getSource** zjišťovat zda byla událost vyvolána příslušným komponentem. Pokud tedy bude na komponent kliknuto, v dalším řádku se pomocí metody **isSelected** načte do proměnné typu **boolean** hodnota příslušející aktuálnímu stavu. Následně se volá metoda pro zobrazení scény, ve které bude dle hodnoty této proměnné zjištěno, zda se mají osy přidat do scény. Tím je úprava metody dokončena. Popis celé metody je v Technickém manuálu (kapitola 7.2).

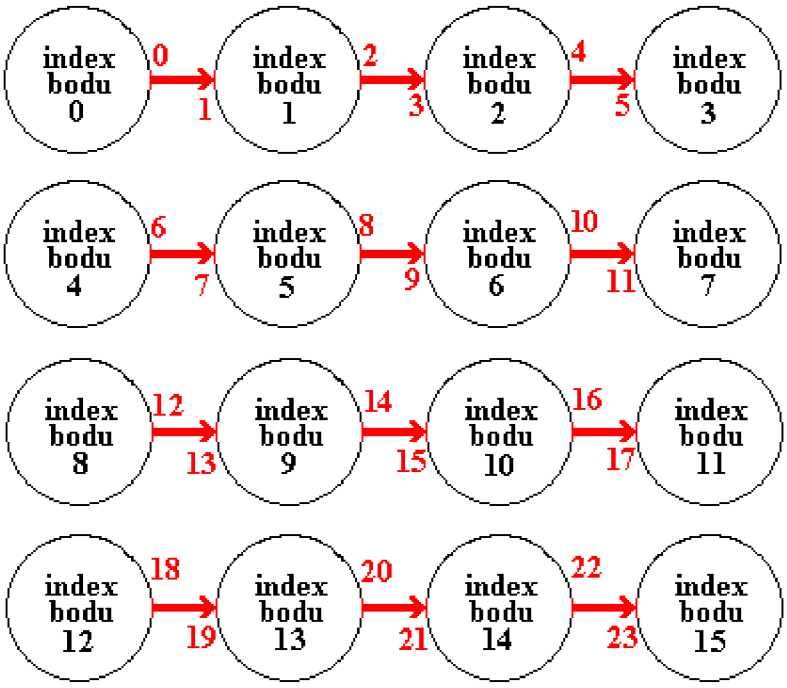
Nakonec musí být naprogramována metoda pro obsluhu událostí komponentu třídy **Choice**. Tou je metoda **itemStateChanged** a je v ní zjištěno, zda byla událost vyvolána komponentem třídy **Choice**. Pokud byla, volá se metoda pro zobrazeni **vytvorScenu**. V metodě **vytvorScenu** se ovšem musí upravit kód tak, aby metoda dokázala reagovat na změnu výběru v komponentu třídy **Choice**. Pomocí metody **getSelectedIndex** se do proměnné uloží hodnota indexu volby, která byla vybrána. Podle hodnoty této proměnné je volána příslušná metoda, která zajistí požadované zobrazení. Popis celé metody je v Technickém manuálu (kapitola 7.2).

5.2.3 Další úpravy programu

Nejdříve bude vytvořena metoda **vytvorOsy**. V té jsou vytvořeny osy *x*, *y*, *z* jejichž průsečík je v počátku soustavy. Aby osy nepůsobily rušivě, je v objektu třídy **Appearance** nastavena průhlednost. Hodnota **0.75f** v závorce konstruktoru třídy **TransparencyAttributes** znamená, že osy budou průhledné ze 75%. Pomocí třídy **Color3f** je také definována barva os, která je zde bílá. Samotné osy budou vykresleny pomocí třídy **LineArray**. Popis celé nově vytvořené metody je v Technickém manuálu (kapitola 7.3).

Aby bylo možné vykreslovat povrch obrázku jako jednotlivé body, pomocí události vyvolané komponentem **volba**, budou použity metody **vytvorGeometrii** a **vytvorVzhled** z appletu Interaktivita. Metoda **vytvorGeometrii** bude přejmenována na **vytvorBody** a aby bylo možné vykreslit body barevně pomocí obrázku z fundus kamery, použijí se stejné algoritmy pro deklaraci bodů a barev jako v metodě **vytvorPovrch**. Metoda **vytvorVzhled** zůstane beze změny. Je ovšem nutné přidat opět do metody **vytvorScenu** řádek s voláním metody **vytvorVzhled**. Popis obou upravených metod je uveden v Technickém manuálu (kapitola 7.3).

Dále musí být upraven konec metody **vypocty** tak jako v appletu **Interaktivita**. Tzn. musí se opět vypočítat proměnná která určuje vzdálenost bodů od sebe. Stejně tak se musí upravit konec metody **cteniBarvy**, do které bude přidán nový řádek, který zajistí výpočet proměnné, která určuje počet pixelů v obrázku a která je nutná pro vykreslení bodů ve třídě **PointArray** i v případě, že nebude načten topografický obrázek. Popis obou upravených metod je uveden v Technickém manuálu (kapitola 7.3).

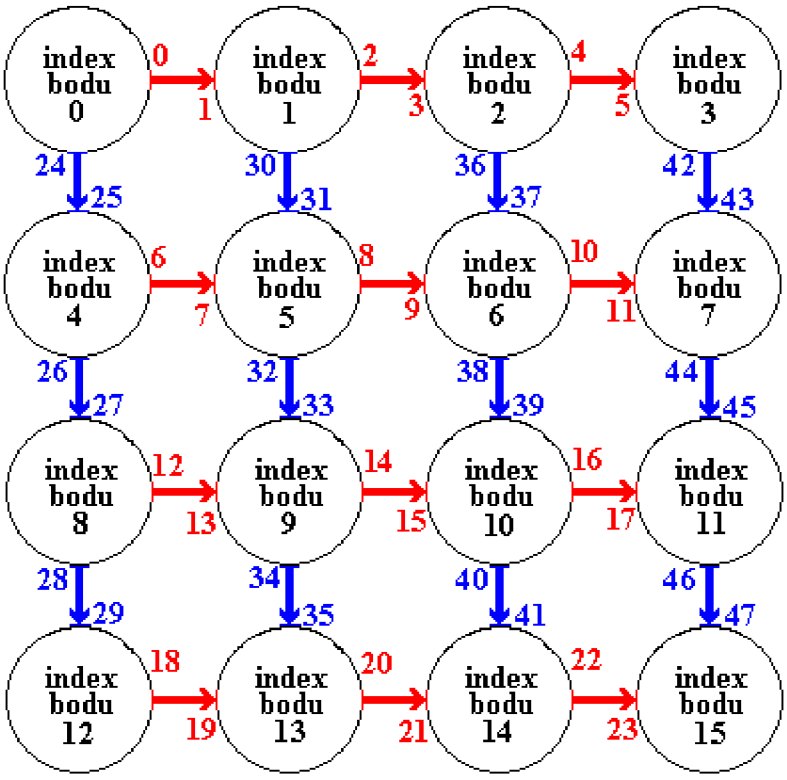


Obr.5.4: *Systém vykreslování obrázku pomocí čar.*

Pro vykreslení povrchu obrázku pomocí čar, bude vytvořena nová vlastní metoda **`vytvorCary`**. Základem metody je třída **`LineArray`**, do jejíhož konstruktoru je nutné zadat kolik bodů bude v obrázku třeba pro vykreslení všech čar. Počet bodů nutných pro vykreslení čar bude vypočítán v metodách **`vypocty`** a **`cteniBarvy`**. Výpočet vychází z obr.5.4, kde je červenými šipkami naznačen systém zadávání počátečních a koncových bodů každé z vykreslovaných čar, tj. pro vykreslení jsou zadávány všechny pixely 1x, ovšem navíc jsou ještě pro zadávání použity 2x pixely ve vnitřních sloupcích.

Metoda **`vytvorCary`** je koncipována podobně jako metoda **`vytvorBody`**. Na začátku je deklarována proměnná která bude určovat pořadí vykreslovaných bodů, dále je vytvořen objekt třídy **`GeometryArray`**, který zajistí vykreslení požadovaného tvaru. Dále jsou použity dvě vykreslovací smyčky pro vykreslení řádků a pro vertikální posun. Následuje určení indexu bodu s nímž se bude pracovat, výpočet souřadnic a odstínů barvy. Je také aplikováno větvení, které umožní volbu mezi barevným a šedotónovým obrázkem. Pak již následuje algoritmus, který zajistí zadávání bodů ve správném pořadí a tím také správné vykreslení obrázku. Popis celé nově vytvořené metody **`vytvorCary`** je v Technickém manuálu (kapitola 7.3).

Podobně je vytvořena také metoda **`vytvorSit`** pro vykreslení obrázku pomocí sítě. I zde je použita třída **`LineArray`**. Vykreslování je zde prováděno stejně jako v metodě **`vytvorCary`** v horizontálním směru, ale navíc ještě ve vertikálním směru (viz. obr.5.5). Počet čar pro vykreslení je tedy dvojnásobný. Metoda **`vytvorSit`** je opět koncipována podobně jako předchozí metody **`vytvorBody`** a **`vytvorCary`**. Rozdíl je opět pouze v systému vykreslování a tedy i v navrženém algoritmu pro vykreslování. První část algoritmu je totožná s metodou **`vytvorCary`**, protože se vykreslují čáry v horizontálním směru. Pro vykreslování ve směru vertikálním je použitý také stejný algoritmus, ale je v něm změněno indexování bodů pro vykreslení tak, aby odpovídalo systému vykreslování pomocí modrých šipek z obr.5.5. V algoritmu který vykresluje čáry vertikálně, tedy dojde k tomu, že se budou pomocí vnitřní smyčky vykreslovat nejdříve sloupce a toto vykreslování bude posunováno pomocí vnější smyčky po celé šířce obrázku. Popis celé metody **`vytvorSit`** je v Technickém manuálu (kapitola 7.3).

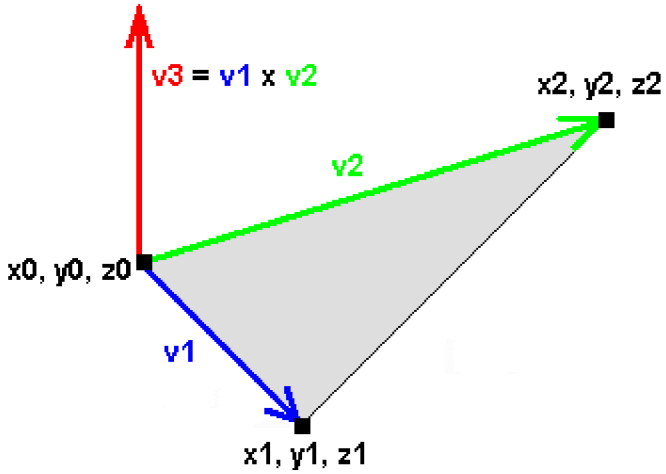


Obr.5.5: *Systém vykreslování obrázku pomocí sítě*

Pro vykreslení povrchu je použita metoda **`vytvorPovrch`**, která je naprosto shodná se stejnojmennou metodou použitou v appletu **`OptickyDisk1`**.

Pro osvětlení povrchu modelu je navržena nová vlastní metoda s názvem **`Osvetleni`**. K osvětlení povrchu se zde bude využívat podobný algoritmus jako v metodě **`vytvorPovrch`**, ale aby byly zobrazeny a osvětleny vykreslené trojúhelníky, budou se muset vygenerovat jejich normálové vektory.

V metodě **`vytvorVzhled`** se nejdříve musí nastavit vlastnosti povrchu, jako je odstín světla odraženého do okolí (ambinent), odstín vlastního vyzařování povrchu (emissive), odstín rozptýlené složky světla (diffuse), odstín odlesku materiálu povrchu (specular). Tyto vlastnosti jsou sdruženy do objektu třídy **`Material`**. Musí být také určeny vlastnosti samotného světelného zdroje, který povrch osvětluje. To se provede v nově vytvořené vlastní metodě s názvem **`vytvorSvetlo`**, která využívá metody třídy **`Light`**. Musí být určen vektor směru odkud světlo přichází, barva světla (zde bílá), dále že se jedná o přímé světlo a nakonec se musí vymezit oblast působení. Je také nastaveno světlo (resp. osvětlení) v okolí 3D modelu. To je provedeno v nově vytvořené vlastní metodě s názvem **`vytvorOkolniSvetlo`**, která opět využívá metody třídy **`Light`**. Je zde vytvořeno světlo v okolí (zde barva šedá), a je vymezena a nastavena oblast působení světla. Stejně jako je z metody **`vytvorScenu`** volána metoda **`vytvorVzhled`**, musí být volány také metody **`vytvorOkolniSvetlo`** a **`vytvorSvetlo`**.



Obr.5.6: *Princip výpočtu normálových vektorů..*

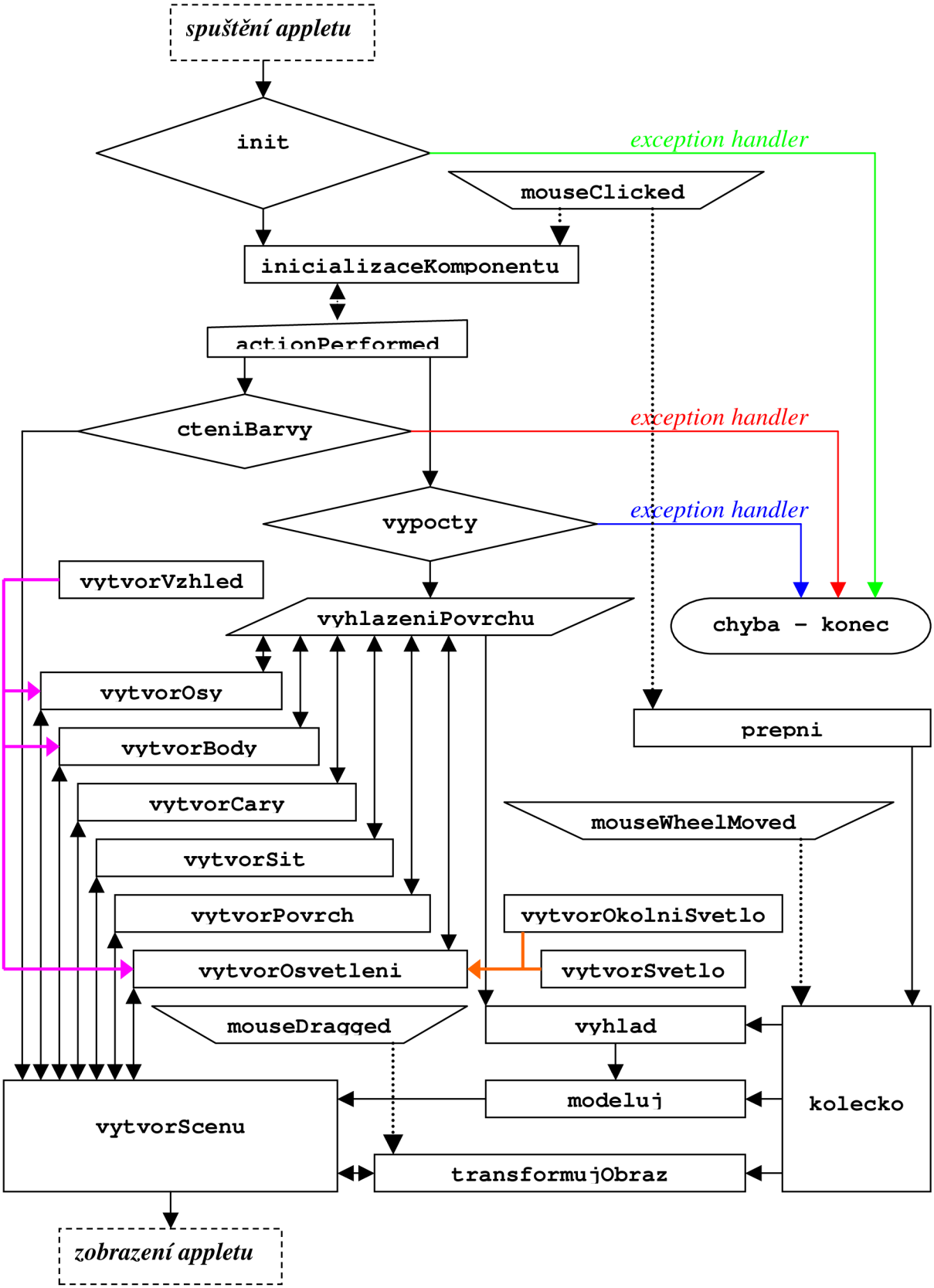
Jak již bylo zmíněno, k osvětlení povrchu se bude využívat podobný algoritmus jako v metodě **`vytvorPovrch`**. Bude tedy použitý objekt třídy **`TriangleStripArray`**. Algoritmus ale bude nutné upravit tak, aby dokázal vypočítat normálové vektory vykreslovaných trojúhelníků. V algoritmu se nejdříve určí souřadnice bodů v prostoru, které tvoří vrcholy trojúhelníku. Ty se zapíší do proměnných typu **`float`**. Vektory odvěsen trojúhelníku **`v1`** a **`v2`** jsou získány odečtením počáteční souřadnice od koncové a to v každém rozměru. Normálový vektor **`v3`** se získá tak, že vypočítané vektory odvěsen **`v1`** a **`v2`** se vektorově násobí pomocí metody **`cross`**. Názorně je tento postup ilustrován na obr.5.6. Normálový vektor **`v3`** je třeba ještě znormalizovat metodou **`normalize`**. Nakonec se pomocí smyčky nastaví vždy těmto třem bodům v jednom trojúhelníku jejich normálový vektor. Totéž se opakuje pro všechny body tvořící trojúhelníky v obrázku. Všechny metody pro osvětlení povrchu 3D modelu jsou popsány v Technickém manuálu (kapitola 7.3).

Nakonec musí být ještě upraveno okno s nápovědou. Do něho přibyl nový text, který upřesňuje práci s novými komponenty a funkcemi. Koncepce vytvoření nápovědy je stejná jako u appletu **`OptickyDisk1`**. Kvůli množství textu zde není uveden celý kód. Celé okno nápovědy i s textem je uvedeno v Příloze- Uživatelský manuál.

Přehled metod, které byly přidány do appletuOptickyDisk2, je v tab.5.3. Jedná se o 8 vlastních metod, které byly pro applet vytvořeny v této práci. Na obr.5.7 je pomocí vývojového diagramu znázorněn zjednodušeně běh appletu OptickyDisk2. Do obrázku opět nejsou kvůli přehlednosti zahrnuty metody, které jsou využívány pouze pro načítání aktuální pozice kursoru, metody, které omezují činnosti při umístění kursoru mimo applet, nebo metody pro ovládání pomocí klávesnice. Všechny metody ve vývojovém diagramu jsou popsány v Technickém manuálu (kapitola 7).

Tab.5.3: *Vlastní metody přidané do appletu OptickyDisk2.*

Název metody	Popis funkce
Přidané vlastní metody:	
private void vytvorOsy()	vytvoření os x, y, z
private Geometry vytvorBody()	vykresleni pomocí bodů
private Geometry vytvorCary()	vykresleni pomocí čar
private Geometry vytvorSit()	vykresleni pomocí sítě
private Geometry Osvetleni()	osvětlení povrchu
private Light vytvorOkolniSvetlo()	vytvoření okolního světla
private Light vytvorSvetlo()	vytvoření okolního světla
private Appearance vytvorVzhled()	nastavení vzhledu bodů



Obr.5.7: *Schéma běhu appletu OptickyDisk2.*

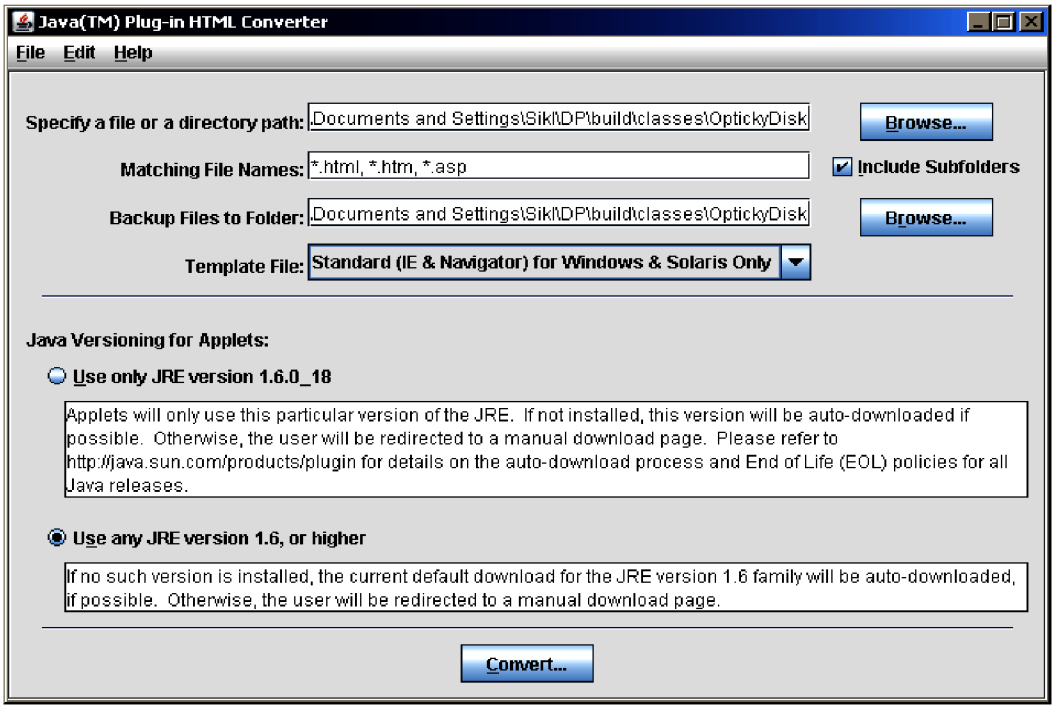
5.3 Implementace appletu do webové stránky

Po kompilaci zdrojového textového programu (*.java*) je k dispozici tzv. „byte kód“, který je již spustitelnou verzí programu. Tento soubor má příponu *.class* a pokud je používáno vývojové prostředí NetBeans, jako v této práci, je automaticky vytvářen v adresáři *.../build/classes/*. V části 3.1 již bylo zmíněno, že ke spuštění appletu je třeba vytvořit párový tag APPLET ve webové stránce. Tím se jednak sdělí webovému prohlížeči kde přesně je umístěn soubor přeloženého programu (byte kódu) a konkretizuje se též poloha a velikost GUI appletu ve webové stránce. Vývojové prostředí NetBeans také automaticky generuje do adresáře spouštěcí HTML stránku, která již obsahuje tento kód. Pro spuštění appletu pomocí webové stránky je však nezbytnou podmínkou, aby bylo nainstalováno tzv. JRE (Java Runtime Environment), což je spouštěcí prostředí Javy. To má v sobě zahrnuto i JVM (Java Virtual Machine), pomocí něhož se byte kód po spuštění interpretuje a tak je získán program, který obsahuje instrukce pro daný operační systém.

5.3.1 Detekce JRE (Java Runtime Environment)

Vzhledem k tomu, že JRE již nebývá součástí webových prohlížečů, musí uživatel toto prostředí stáhnout a nainstalovat z oficiálních stránek Javy [22]. Existuje ovšem také možnost automaticky detekovat přítomnost tohoto prostředí a případně jej stáhnout i instalovat, pokud zcela schází v hostitelském počítači, nebo pokud je nainstalovaná verze nevyhovující pro požadavky appletu. Aby byla přítomnost JRE automaticky detekována a případně stažena a instalována před spuštěním appletu, je třeba složitější HTML kód. Ten je ovšem možné nechat vygenerovat automaticky. Toto automatické generování se provádí pomocí plug–in programu *HtmlConverter.exe* (viz. obr. 5.8), který je součástí instalace JDK a je umístěn v *C:/Program Files/java/jdk1.6/bin/*.

Po spuštění se v řádku *Specify a file or directory path* klikne na tlačítko *Browse* a nastaví se cesta k adresáři *build*, kde se nachází spouštěcí *.html* stránka, jejíž kód má být modifikován. V části *Java Versioning for Applets* je vhodné zkontrolovat, zda je označena volba *Use any Java 1.6, or higher*, čímž bude zajištěno, že na klientském spouštěcím počítači bude zjišťována přítomnost instalace JRE verze 1.6 nebo vyšší. Kliknutím na tlačítko *Convert* se do stránky vygeneruje příslušný kód. Pokud nyní nebude na hostitelském počítači nainstalováno JRE verze 1.6 nebo vyšší, bude provedena jeho automatická instalace.



Obr.5.8: *Automatické generování kódu pro detekci JRE.*

5.3.2 Modul zabezpečení

Obecně platí, že každý applet, který je stahován jako součást webové stránky, může představovat určité bezpečnostní riziko, protože se vlastně jedná o spouštění cizího programu na hostitelském počítači. Již při návrhu prvních verzí jazyka Java bylo bráno v úvahu toto riziko a byl navržen určitý model zabezpečení, tzv. sandbox. Sandbox je vlastně bezpečnostní prostředí kolem aplikace, které dovolí aplikacím provádět pouze bezpečné operace (tj. zobrazení, výpočet, atd.), ovšem zakáže činnosti potenciálně nebezpečné (tj. přístup k souborům, periferním zařízením, síťovému připojení, atd.). Tímto zákazem se však mohou výrazně zúžit aplikační oblasti pro použití appletů, proto byl vyvinut způsob umožňující spuštění appletu i mimo sandbox. Tento způsob spočívá v digitálním podpisu vzdáleného kódu. Programátor pak tímto certifikátem ručí za bezpečnost svého programu. Pak už se rozhodne sám uživatel, jestli spuštění programu povolí v rámci sandboxu nebo mimo něj.

Podrobný popis vytváření certifikátu je nad rámec této práce. Takže bude pouze stručně popsáno jak byl vytvořen certifikát pro applet `OptickyDisk1`. Při vytváření certifikátů pro další applety je postup analogický. Nejdříve je tedy třeba vytvořit pracovní adresář pomocí programu *Průzkumník*, nebo *Total Commander* (např. *C:\key*). Další práce probíhá v příkazovém řádku (*Start/Spustit/cmd*). Pomocí programu *keytool.exe* (součást JDK - *C:/ProgramFiles/Java/jdk1.6.0/bin*) se musí vytvořit dvojice šifrovacích klíčů, které budou uloženy do malé databáze (úložiště certifikátů):

C:\Progra~1\Java\jdk1.6.0\bin\keytool –genkey –alias zde_napsat_*jmeno_prijmeni* **–keypass** vymyslet_libovolne_*heslo_ke_klicum*(min_6_znaků!) **–keystore c:\key\uloziste.jks –storepass** vymyslet_libovolne_*heslo_do_uloziste*(min_6_znaků!)

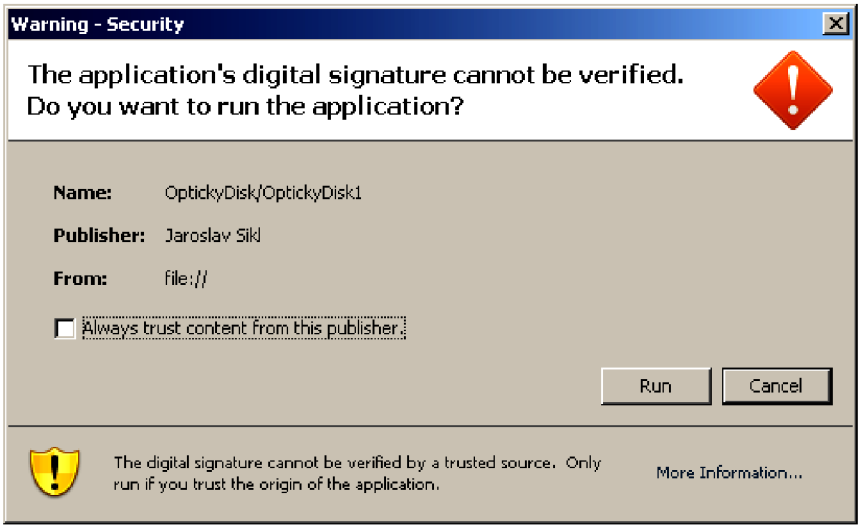
Po spuštění požádá program o zadání základních informací. Po potvrzení správnosti údajů jsou klíče vytvořeny a uloženy do úložiště certifikátů. Dále se musí zkopírovat adresář s kódem z adresáře *.../build/classes* do *c:\key*. Program včetně zachování adresářové struktury se zabalí do archivu JAR následujícím příkazem (program *jar.exe* je také součástí JDK):

C:\Progra~1\Java\jdk1.6.0\bin\jar cf c:\key\applet.jar –C c:\key ubmi

Digitální podpis archivu *applet.jar* vyžaduje použití programu *jarsigner.exe* (součást JDK). Při jeho použití je nutné zadávat stejné hodnoty, které byly určeny při tvorbě párů šifrovacích klíčů.

C:\Progra~1\Java\jdk1.6.0\bin\jarsigner –keystore c:\key\uloziste.jks –storepass *heslo_do_uloziste* **–keypass** *heslo_ke_klicum* **–signedjar c:\key\safeapplet.jar c:\key\applet.jar** *jmeno_prijmeni*

Pak se zkopíruje archiv *c:\key\safeapplet.jar* do adresáře *.../build/classes/*. Po otevření .html stránky (zde stránka *OptickyDisk1.html*) se v textovém editoru (např. *Poznámkový blok*) připíše do zdrojového kódu, do tagu APPLET následující řádek: ARCHIVE = “safeapplet.jar”. Musí se také upravit velikost zobrazované oblasti v okně webového prohlížeče na WIDTH = 668 HEIGHT = 512.



Obr.5.9: *Certifikát zabezpečení appletu.*

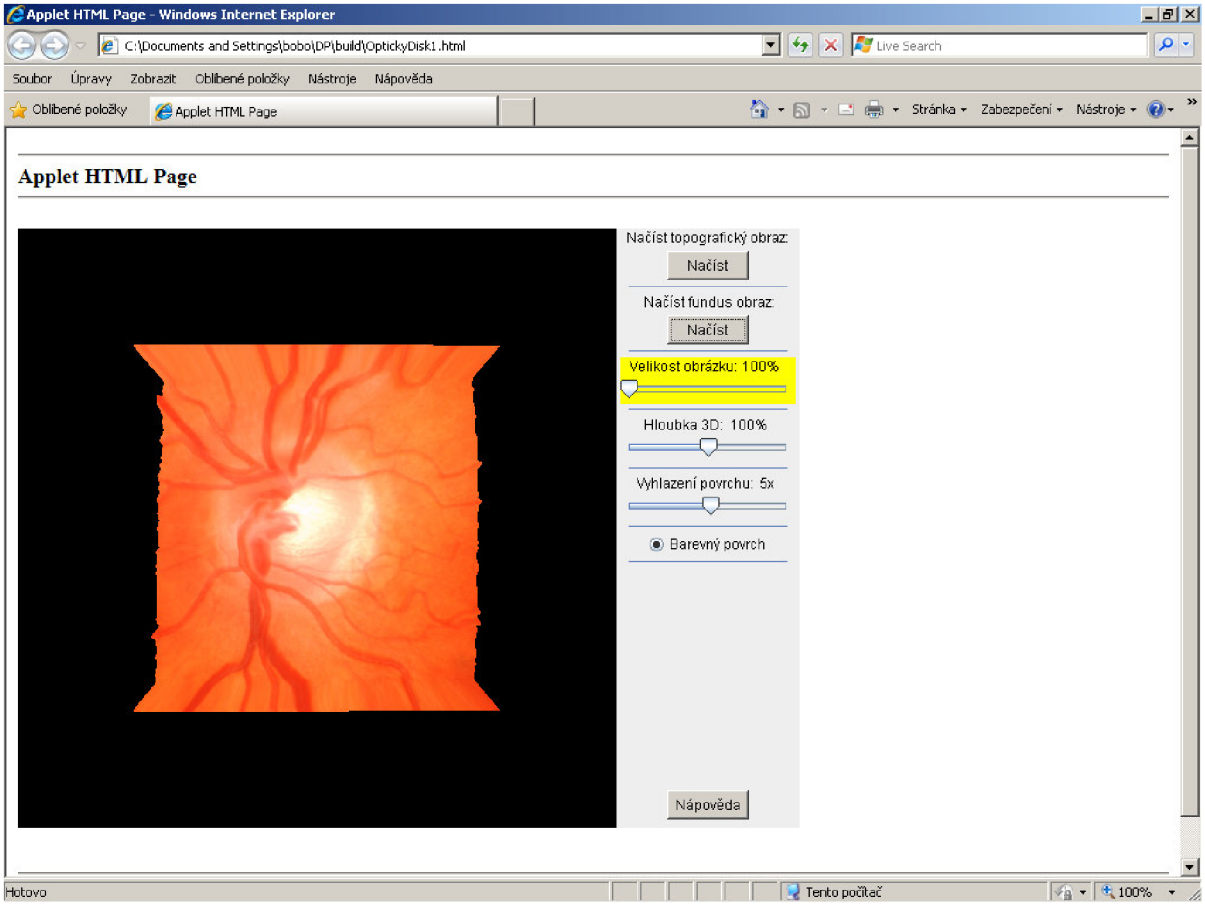
Po načtení stránky ve webovém prohlížeči a automatickém spuštění appletu se zobrazí okno certifikátu, které je na obr.5.9. Kliknutím na tlačítko *Run* se potvrdí bezpečnost programu a povolí se jeho běh i mimo vývojové prostředí. Kliknutím na tlačítko *Cancel* zůstanou práva programu nezměněna, tj. má zakázaný přístup k lokálním systémovým prostředkům.

Pro vývojové účely se mohou v prostředí Javy povolit veškerá práva appletu. Toto je možné i bez digitálního podpisu přeložené verze programu. K tomuto účelu slouží program *policytool.exe* (viz. obr.5.10), který je opět součástí JDK.

Obr.5.10: *Program pro povolení práv appletu pro vývojové účely.*

Po spuštění tohoto programu se objeví okno v němž se nejdříve klikne na tlačítko *Add Policy Entry* a poté na *Add Permission*. V nově otevřeném dialogovém okně se vybere *All Permission* a klikne se na *OK*. Poté se již jen klikne na tlačítko *Done*. V menu *File/Save As* se pak uloží soubor zásad do adresáře *C:/Documents and Settings/nazev_profilu/* pod názvem *java.policy*.

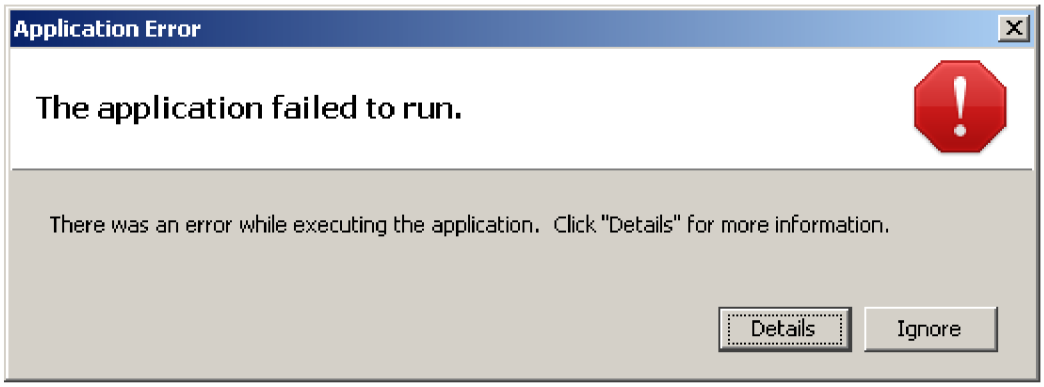
5.4. Spuštění a testování appletu



Obr.5.11: *Applet s názvem OptickyDisk1, spuštěný v okně webového prohlížeče.*

Na obr.5.11 je ukázka appletu **`OptickyDisk1`** spuštěného v okně webového prohlížeče *Internet Explorer 7*. Spuštění obou verzí appletů bylo vyzkoušeno také v dalších prohlížečích, které jsou nejběžněji používány: *Mozilla Firefox 3.6*, *Opera 9.5*, *Netscape Navigator 9* a *Safari 4.42*. Applety pracovaly bez problémů ve všech těchto prohlížečích.

Ovšem v případě, že uživatel nebude mít nainstalovanou nadstavbu Java3D, nedojde ke spuštění appletu, protože nebudou dostupné knihovny pro práci ve 3D prostředí. Zobrazí se tedy varování dle obr.5.12.



Obr.5.12: *Varování při chybějící instalaci nadstavby Java3D*

Dále byl běh appletů otestován na dvou typech počítačů. Při zobrazování obrázků do cca 128 x 128 pixelů, je běh appletu plynulý i na notebooku s jednojádrovým procesorem o frekvenci 1,4 GHz, operační pamětí o velikosti 512 MB typu DDR a FSB 266 MHz, a grafickou kartou Radeon ATI Mobility 9000 s operační pamětí velikosti 32 MB typu GDDR. Větší obrázky nad 128 x 128 pixelů byly také zpracovány, ale při používání funkcí které vyžadují více prostředků pro výpočty, nebo grafické zpracování (např. změna 3D hloubky, vyhlazení povrchu) již applet nepracoval plynule, což se projevilo blikáním obrazu a delší prodlevou při zobrazování, která byla v řádu jednotek sekund. Při zpracovávání topografických obrázků a obrázků z fundus kamery, které mají rozměr 384 x 384 pixelů, nebyl výkon tohoto notebooku pro některé operace dostačující, aby byl zajištěn uživatelský komfort při používání aplikace.

Druhým typem počítače bylo stolní PC se čtyřjádrovým procesorem QuadCore 9650 o frekvenci 4 x 3 GHz, operační pamětí o velikosti 4 GB typu DDR2 a FSB 1066 MHz, a grafickou kartou GeForce 8800GT s operační pamětí velikosti 1 GB typu GDDR3. Při zpracovávání topografických obrázků a obrázků z fundus kamery, které mají rozměr 384 x 384 pixelů, pracoval na tomto PC applet **`OptickDisk1`** bez problémů. Při používání funkcí které vyžadují více prostředků pro výpočty, nebo grafické zpracování bylo také zaznamenáno blikání obrazu, ale bez většího vlivu na uživatelský komfort. U rozšířeného appletu **`OptickyDisk2`** byly však již patrné menší prodlevy při práci s osvětleným 3D modelem, ovšem pouze při použití funkce vyhlazování povrchu. Vykreslování bylo pomalejší a blikání obrazu bylo již na hranici zajištění uživatelského komfortu při práci s aplikací.

Nakonec bylo provedeno testování appletu na dostupných datech. K dispozici byly tři šedotónové topografické obrázky výstupu očního nervu a k nim náležející tři barevné obrázky z fundus kamery. Při testování bylo zjištěno, že v appletu je ošetřena pouze možnost aby nebyly použity obrázky, které nemají stejný rozměr. Není ovšem ošetřena možnost, že bude načten topografický obrázek spolu s obrázkem z fundus kamery, které mají stejné rozměry, ale které k sobě nenáleží (tj. jsou např. od různých pacientů). Toto ošetření by sice mohlo být založeno např. na čtení názvu obrázků, to ovšem nevylučuje možnost záměny nebo případnou nemožnost zobrazení při chybě v názvu, nebo při přejmenování jednoho z obrázků. Ošetření by bylo velmi komplikované a zřejmě by vyžadovalo implementaci nějakého ochranného prvku již při zpracovávání obrázků. Proto je ponecháno pouze na uživateli, aby vybral správné obrázky které k sobě náleží. Výsledky testování s pořízenými náhledy 3D modelů jsou uvedeny v Přílohách 3 a 4.

V Příloze 5 je uživatelský manuál, ve kterém je podrobně popsáno ovládání funkcí appletu a to jak pomocí myši, tak i pomocí klávesnice. V Příloze 6 je Technická dokumentace.

6 Závěr

Cílem této práce bylo seznámit se s vyšetřovací metodou konfokální laserové skenovací oftalmoskopie a na základě výstupních dat Heidelbergova sítnicového tomografu (HRT) navrhnout Java applet pro 3D vizualizaci optického disku oka s vhodnou mírou interaktivity.

Nejdříve byly prostudovány různé přístupy pro vizualizaci objektů ve 3D prostoru pomocí programovacího jazyka Java a jeho nadstaveb Java 3D a VTK. Pro tuto práci byl zvolen programovací jazyk Java s nadstavbou Java 3D, protože se předpokládá implementace programu do rozhraní webových prohlížečů postavených na různých systémových platformách. Proto je zde multiplatformita jazyka Java výhodná stejně jako možnost implementace programu do webového prohlížeče formou Java appletu.

Ve fázi návrhu bylo vyzkoušeno několik přístupů pro vykreslování obrázků v prostředí Java3D a bylo vytvořeno několik pomocných appletů, pomocí kterých byly vyzkoušeny různé metody pro práci s obrázky ve 3D prostředí. Z těchto pomocných appletů pak byly vybrány ty metody, které se jevily jako nejlepší možné řešení pro realizaci finálního appletu.

Výsledkem této práce je realizace dvou appletů. Applet s názvem **`OptickyDisk1`** je jednoduchý intuitivní program, v jehož grafickém uživatelském prostředí jsou již přednastaveny některé funkce, takže po načtení obrázku z pevného disku počítače je automaticky vykreslen 3D model. Applet po načtení obrázku a vykreslení jeho 3D modelu v prostoru dovoluje nastavovat u tohoto zobrazení velikost, dále 3D hloubku zobrazení a lze také aplikovat algoritmus pro vyhlazení hran povrchu. Je zde také možnost volby zobrazení obrázku šedotónového, nebo barevného. Dalším výstupem této práce je applet s názvem **`OptickyDisk2`**. Tento applet je rozšířením prvního a je v něm navíc možné zobrazit ve 3D prostoru osy *x, y, z*, které mohou být užitečné pro lepší orientaci. Dále jsou zde rozšířeny možnosti pro vykreslování povrchu 3D modelu. Je možné zobrazovat model pomocí vykreslení jednotlivých bodů v prostoru, dále pomocí čar nebo sítě a také je zde možnost zobrazit 3D model pomocí osvětlování jeho povrchu.

Protože applety mají defaultně zakázaný přístup k souborům na pevném disku uživatele, byl pro implementaci appletů do webového rozhraní vytvořen certifikát zabezpečení, po jehož potvrzení uživatelem je možné načítat soubory z jeho pevného disku. Protože platnost certifikátu bývá omezená, je v práci stručně popsán také postup pro vytvoření tohoto certifikátu.

Vytvořené applety byly otestovány na dostupných datech z HRT a jeho funkčnost byla prověřena v různých webových prohlížečích.

Součástí práce je také technická a uživatelská dokumentace. Technická dokumentace obsahuje podrobný popis metod, jejich zdrojového kódu a funkcí. V uživatelském manuálu je přehledný návod jak s appletem pracovat.

7 Seznam literatury

[1] CIULLA, Thomas A., REGILLO, Carl D., HARRIS, Alon. *Retina and Optic Nerve Imaging*. Hagerstown: Lippincott Williams & Wilkins, 2003. 398 s. ISBN 0-7817-3433-9.

[2] IESTER, M., GARWAY-HEATH, D., LEMIJ, H. Optic Nerve Head and Retinal Nerve Fibre Analysis. *European Glaucoma Society*. Savona: Dogma, 2005. 111-113s. ISBN 88-87434-30-1.

[3] HONZÍKOVÁ, N., HONZÍK, P. *Biologie člověka*. Elektronická skripta VUT v Brně, 2003. 134 s.

[4] ROZMAN, J., CHMELAŘ, M., JEHLIČKA, K. *Terapeutická a protetická technika*. Elektronická skripta VUT v Brně, 2004. 134 s.

[5] FLAMMER, J. *Glaukom*. Praha: Triton, 2003. 420 s. ISBN 80-7254-351-2.

[6] VALEŠOVÁ, L. *www.ordinace.cz* [online]. 2007 [cit. 2010-05-18]. Zelený zákal (glaukom). Dostupné z www: <http://www.ordinace.cz/clanek/zeleny-zakal-glaukom>.

[7] KUBĚNA, T., ČERNOŠEK, P. *www.hrt2.cz* [online]. 2005 [cit. 2010-05-18]. HRT II. Dostupné z www: <http://www.hrt2.cz>.

[8] Heidelberg Engineering GmbH. *www.agingeye.net* [online]. 2009 [cit. 2010-05-18]. Quantitative Three-Dimensional Imaging of the Posterior Segment with the Heidelberg Retina Tomograph. Dostupné z www: <http://www.agingeye.net/glaucoma/heidelberg.pdf>.

[9] SEDLÁČKOVÁ, V. *Využití laseru v diagnostice očních nemocí*. Brno, 2008. 60 s. Bakalářská práce. MU Brno, LF.

[10] HLOŽÁNEK, M. *Přístrojová technika v oftalmologii*. Praha: ART et FACT, 2006. 28 s. ISBN 80-902160-9-9.

[11] CAMPR, P. *Získávání 3D modelu lidských tkání z obrazových dat CT*. Plzeň, 2005. 58 s. Diplomová práce. ZČU Plzeň, FAV, KKY.

[12] KAISER, J. *Obecný modulární grafický subsystém pro prostředí MVE-2*.Plzeň, 2006. 63 s. Diplomová práce. ZČU Plzeň, FAV, KIV .

[13] ImageJ. *rsbweb.nih.gov* [online]. 2004 [cit. 2010-05-18]. ImageJ. Dostupné z www: <http://rsbweb.nih.gov/ij/index.html>.

[14] Java 3D. *www.java3d.org* [online]. 2001 [cit. 2010-05-18]. Tutorial. Dostupné z www: <http://www.java3d.org/tutorial.html>.

[15] Kitware Inc. *www.vtk.org* [online]. 2008 [cit. 2010-05-18]. The VTK User's Guide. Dostupné z www: <http://www.vtk.org/VTK/resources/software.html>.

[16] FLANAGAN, D. *Programování v jazyce JAVA*. Brno: Computer Press, 1997. 488 s. ISBN 80-85896-78-8.

[17] ZAKHOUR, Sharon. Java 6 výukový kurz. Brno: Computer Press a.s., 2007. 536 s. ISBN 978-80-251-1575-6.

[18] Glaukom In *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 19. 2. 2008, 24. 4. 2010, [cit. 2010-05-18]. Dostupné z www: <http://cs.wikipedia.org/wiki/Glaukom>.

[19] ŽÁRA, J; BENEŠ, B; FELKEL, P. Moderní počítačová grafika. Praha: Computer press, 2005. 609 s. ISBN 80-251-0454-0.

[20] RŮŽIČKOVÁ, E. *www.glaukom.cz* [online]. 2003 [cit. 2010-05-18]. Stručný průvodce glaukomovým onemocněním. Dostupné z www: <http://www.glaukom.cz/kategorie.asp?idk=129>.

[21] Heidelberg Engineering GmbH. *www.heidelbergengineering.com* [online]. 2004 [cit. 2010-05-18]. Product Literature. Dostupné z www: <http://www.heidelbergengineering.com/international/products/hrt-retina/product-literature>.

[22] Oracle. *java.sun.com* [online] 2010 [cit. 2010-05-18]. Java SE Downloads. Dostupné z www: <http://java.sun.com/javase/downloads/index.jsp>

ABECEDNÍ PŘEHLED POUŽITÝCH ZKRATEK A SYMBOLŮ:

2D	2-Dimensional, dvojrozměrný
3D	3-Dimensional, trojrozměrný
C++	objektově orientovaný programovací jazyk
DDR	Double Data Rate, typ paměti používané v počítačích
End	klávesa na počítačové klávesnici
FSB	Front Side Bus, obousměrná datová sběrnice
GDDR	Graphic Double Data Rate, specifický typ paměti pro grafické karty
GDx	Laserová skenovací polarimetrie
GIF	*Graphics Interchange Format*, grafický formát určený pro rastrovou grafiku
GUI	Graphic User Interface, grafické uživatelské rozhraní
Home	klávesa na počítačové klávesnici
HRT	Heidelberg Retinal Tomograph, Heidelbergův sítnicový tomograf
HTML	HyperText Markup Language, značkovací jazyk pro hypertext
JDK	Java Development Kit, vývojové prostředí jazyka Java
JRE	Java Runtime Environment, prostředí pro běh programů v jazyce Java
JVM	Java Virtual Machine
OCT	Optická koherentní tomografie
ONH	Optical Nerve Head, optický disk oka
OS	Operační Systém
PC	Personal Computer, osobní počítač
PgDn	Page Down, klávesa na počítačové klávesnici
PgUp	Page Up, klávesa na počítačové klávesnici
PNG	Portable Network Graphics, grafický formát pro bezeztrátovou kompresi grafiky
px	Pixel, pixely
RGB	Red Green Blue, aditivní způsob míchání barev
RGBA	rozšíření RGB o tzv. alfa kanál A s informací o průhlednosti konkrétního pixelu
Space	mezerník, klávesa na počítačové klávesnici
Tab	Tabulátor, klávesa na počítačové klávesnici
Tcl	programovací jazyk založený na zpracování seznamů
VRML	Virtual Reality Modeling Language, programovací jazyk pro grafický formát
VTK	The Visualization Toolkit