Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Nástroj pro volbu resekční linie při chirurgii jater

PLZEŇ, 2014 **MICHAL KLÍMA**

P R O H L Á Š E N Í

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne

……

PODĚKOVÁNÍ

Tímto bych rád poděkoval panu Ing. Miroslavovi Jiříkovi za podklady, čas a rady vynaložené na pomoc při vytváření této práce. Dále bych chtěl poděkovat mé přítelkyni Míše a svojí rodině za podporu a motivaci po celou dobu studia.

Abstrakt:

Tato práce je zaměřena na vizualizaci dat z CT snímků jater a vytvoření resekčních algoritmů, které budou schopny provádět virtuální resekci. Jako resekční algoritmy byly použity resekce podle roviny a resekce podle bodu. Vizualizace byla provedena pomocí nástroje VTK. V závěru je na několika experimentech ověřena funkčnost programu při použití dat získaných ze skutečných CT snímků. Přínosem této práce by měla být pomoc lékařům při chirurgických zákrocích na játrech.

Klíčová slova:

Resekční linie, 3D vizualizace, VTK, chirurgie jater, DICOM, resekční linie, Python, PyQt

Abstract:

The work is focused on the visualization of data from CT images of liver and algorithms of virtual resection. As resection algorithms were used resection by plane and resection by point. Visualization was performed using VTK. Several experiments verified the functionality of the program, using data obtained from real CT images. The contribution of this work should help doctors in liver surgery.

Keywords:

Resection line, 3D visualization, VTK, liver surgery, DICOM, resection line, Python, PyQt

Obsah

Seznam obrázků

Seznam zkratek

LISA	Liver Surgery Analyzer je software pro podporu chirurgických zákroků na játrech.
CT	Computed Tomography, neboli výpočetní tomografie, je zobrazovací metoda používaná v lékařství k získání snímků vnitřní stavby lidského těla.
2D	Označuje dvojdimenzionální prostor.
3D	Označuje trojdimenzionální prostor.
MRI	Magnetická rezonance je zobrazovací metoda používaná v lékařství k získání snímků vnitřní stavby lidského těla.
DICOM	Digital Imaging and Communications in Medicine je mezinárodní standard definující způsob ukládání a zpracování lékařských dat.
NEMA	National Electrical Manufactures Association je asociace zabývající se, mimo jiné, zobrazovacími metodami v lékařství.
GUI	Graphical User Interface, neboli grafické uživatelské rozhraní slouží pro interakci mezi programem a uživatelem.
VTK	The Visualization Toolkit je software obsahující knihovny pro vizualizaci dat.
Numpy	Numeric Python je knihovna funkcí rozšiřující výpočetní možnosti Pythonu.
Scipy	Scientific Computing Tools for Python je knihovna rozšiřující výpočetní možnosti Pythonu o složitější funkce.

1. Úvod

Počítačová technika dnes může pomáhat v nejrůznějších odvětvích, ať už se jedná o vědní obory, nebo o každodenní činnosti, což platí i pro lékařství. Zde mohou počítače pomáhat například lékaři při určení nemoci, jakou pacient trpí, při vyhodnocování nejrůznějších informací, nebo při podpoře chirurgických zákroků. Právě do této disciplíny se budeme snažit přispět touto prací, která je součástí větší skupiny projektů souhrnně nazývaných LISA (viz kapitola 1.1. LISA) .

Způsob, jakým jsou v dnešní době vyhodnocovány snímky z lékařských vyšetření (např. z CT) můžeme vidět ve videu na CD v příloze. Lékař musí ve většině případů pracovat s jednotlivými snímky postupně. Jeho trénované oko z nich sice dokáže na základě praxe i zkušeností vyčíst potřebné údaje, ale zabere mu to spoustu času a výsledky nemusí být dostatečně přesné. Navíc pokud si uvědomíme, že počet těchto snímků se může blížit i několika stovkám, mohl by se hodit nástroj, který by lékaři ušetřil jak čas, tak vynaloženou práci.

V této práci se konkrétně zaměříme na dvě oblasti. První z nich bude ucelení jednotlivých dat ze snímků do jednoho velkého objektu, čehož dosáhneme díky vizualizaci dat ve 3D podobě. Lékaři si zvykli na vyhodnocování dat ve 2D podobě, protože s ním přicházejí do styku častěji a jejich zkušenosti většinou vycházejí právě z práce s takovýmto zobrazením. Proto se k 3D provedení stavěli většinou spíše skepticky. Dnes se tento trend již mění a lékaři začínají uznávat výhody, jaké může přinést 3D zobrazení. Mezi tyto výhody patří například možnost nahlížení na objekt z jakéhokoli úhlu, přibližování a oddalování objektu a mnohem větší názornost, než ve 2D podobě. Druhou oblastí této práce je vytvoření nástroje, pomocí kterého bude lékař moci zvolit resekční linii pro odstranění zvolené části jater. Tento nástroj bude opravdovým přínosem, protože nyní musí lékař volit resekční linii ručně a snímek po snímku obkreslovat oblast, kterou chce odstranit.

1.1. LISA (Liver Surgery Analyzer)

Lisa je software vytvořený pro podporu chirurgických zákroků na játrech. Tento software je napsaný v programovém jazyce Python a zahrnuje proces zpracování dat z lékařských snímků (z břišního CT), do kterého patří především segmentace jater, cév a lézí. V současné době je testován radiology a chirurgy. [21]

2. Zobrazovací metody

V této kapitole budou probrány některé základní principy zobrazovacích metod používaných při lékařských vyšetřeních pacientů. Konkrétně se zaměříme na ty, které se používají pro zobrazování jater. Ve většině případů se používá sonografie, nebo CT. Ve speciálních případech je využita magnetická rezonance. [1]

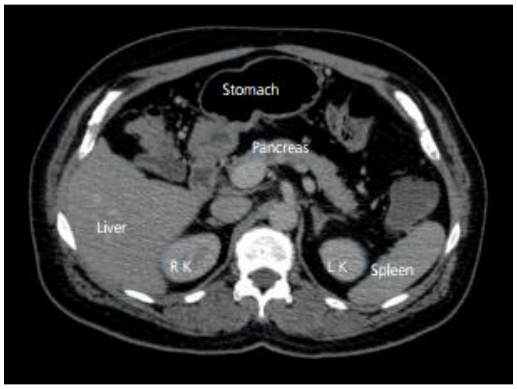
2.1 Sonografie

Sonografie využívá mechanického vlnění známého jako ultrazvuk. Toto vlnění se při průchodu vyšetřovaným objektem odráží, absorbuje, nebo rozptyluje. K odrazu dochází na rozhraní dvou prostředí, přičemž intenzita odrazu závisí na hustotě těchto prostředí. Z tohoto důvodu je velmi obtížné a často nemožné vyšetřovat orgány za skeletem. Tato metoda se proto hodí pro analýzu a zobrazování měkkých tkání, nebo parenchymu, což jsou například játra. [1]

2.2 Výpočetní tomografie (CT – computed tomography)

CT využívá digitální zpracování dat, získaných po průchodu rentgenového záření v mnoha průmětech vyšetřovanou vrstvou. Základní princip je založen na zeslabování svazku rentgenového záření, při průchodu vyšetřovaným objektem. Jde o tomografickou metodu, což znamená, že celé vyšetření se skládá z mnoha sousedících vrstev (skenů). U CT se šířka těchto skenů pohybuje v rozmezí 1-10 mm.

Záření, které projde pacientem, dopadne na detektory uložené proti rentgence. V detektorech se toto záření převádí na elektrický signál, který je následně zpracováván na počítači. Jednotlivé vrstvy se zhotovují tak, že pevné spojení rentgenky a detektorů rotuje kolem pacienta. Pro vytvoření jednoho skenu se toto spojení otočí o 360 ° v rozmezí od 0,5 – 7 sekund. Po celou dobu rotace měří detektory data, ze kterých počítač rekonstruuje obraz dané vrstvy.



Obr. 1 - CT snímek (převzato z [22])

Tento obraz je tvořen maticí bodů, nejčastěji o velikosti 512 x 512. Pro rozpoznání toho co se na obrazu nachází (např. měkké tkáně, skelet) se využívá denzity, tedy míry oslabení záření v jednotlivých místech vyšetřovaného objektu. Udává se v tzv. Hounsfieldových jednotkách (zkráceně HU) se základní stupnicí rozdělenou na 2000 stupňů a to od -1000 do +1000. Na obrazech CT snímků jsou denzity reprezentovány ve stupních šedi.

Jako výhody CT vyšetření můžeme uvést nižší cenu (oproti MRI) a rychlost vyšetření. Jako nevýhodu můžeme uvést malé množství CT přístrojů a používání rentgenového záření. Při vyšetřování jater se výpočetní tomografie používá zejména k upřesnění sonografického nálezu. [1]

2.3 Magnetická rezonance (MRI)

Tato metoda vychází z principu zkoumání změn magnetických momentů jader prvků. Rotací atomových jader s lichým protonovým číslem kolem své osy, vzniká kolem těchto jader magnetické pole (magnetický moment). Vlivem vnějšího magnetického pole dojde k tomu, že jsou rotace jádra uspořádána do jednoho směru. Poté magnetický moment koná pouze dva pohyby, kterými jsou precese a spin. Následně se využije principu rezonance, aby se magnetický moment vychýlil z původního směru o určitý úhel a precese všech protonů se synchronizovali. Toho se docílí aplikací radiofrekvenčního pulzu o frekvenci shodné s frekvencí precese protonu. Když tento pulz skončí, není zde již žádná energie, která by protony udržovala v pozměněné pozici a ty se tak začnou vracet do původního směru. Při vyšetření se zhotovují vrstvové obrazy pomocí sekvencí, což jsou série radiofrekvenčních pulzů. S jejich pomocí získáme signál ve formě elektromagnetického vlnění a jeho velikost můžeme změřit za pomoci cívek. Tento signál se pak převádí do různých odstínů šedi.

MRI se používá především při odhalování tumorů a lézí. Jako výhodu MRI můžeme uvést například vyšší citlivost při zobrazování měkkých tkání, kde některé nálezy na snímcích MRI nemusí být u jiných zobrazovacích metod patrné. Nevýhodou jsou například vyšší pořizovací náklady oproti ostatním zobrazovacím metodám [1]

3. Reprezentace a zpracování dat

V této kapitole nejdříve probereme obecné reprezentace dat používané ve vizualizačních algoritmech. Zaměříme se především na objemové reprezentace a speciální reprezentaci používanou v lékařství zvanou DICOM. V sekci zpracování dat si probereme základní principy segmentace a poslední část této kapitoly je věnována teoretickému rozboru klasifikace bodů v prostoru rozděleném rovinou.

3.1 Reprezentace dat

Při zpracování dat počítačovou technikou hraje důležitou roli reprezentace dat. To platí i pro algoritmy používané pro vizualizaci a zobrazování dat. Požadavky na to, jak by reprezentace dat měla vypadat, se můžou různit podle toho, kdo s ní bude pracovat, nebo kde má být použita. Například uživatel by chtěl mít data popsána co nejjednodušeji. Naopak z pohledu programátora, nebo zobrazovacích programů, může být vyžadováno, aby byl datový popis doplněn o další informace, například o geometrický charakter dat. [2]

3.1.1 Hraniční reprezentace objektů

Tento způsob reprezentace vychází z intuitivní představy člověka, který objekt nejčastěji definuje (třeba když ho kreslí) pomocí jeho hran. Vymezujeme tedy to, co do objektu patří a co ne, pomocí definice jeho hranice. V této reprezentaci dat se uchovává především informace o hraničních bodech tělesa (to jsou takové body, které vždy sousedí minimálně s jedním bodem vnitřním, vnějším a dalším hraničním), o vnitřních bodech tělesa často nemusíme vědět nic.

Nyní si uvedeme některé základní hraniční reprezentace:

- *Hranová reprezentace* - Je to nejstarší a nejjednodušší metoda popisu dat. Pro představu toho, jak tato reprezentace vypadá, si můžeme představit reálný model vytvořený pouze za pomoci drátů. Její implementace je velmi jednoduchá a spočívá pouze ve vytvoření dvou seznamů – hran a vrcholů. Každá položka ze seznamu hran v sobě uchovává ukazatele na dvě položky ze seznamu vrcholů.

- *Bodová reprezentace* - Zde jsou hlavními nositeli informace samotné body. Každý z bodů představuje část povrchu a uchovává v sobě například normálový vektor, nebo hodnoty svých souřadnic. Tyto informace se pak využívají v zobrazovacích algoritmech. Bodová reprezentace má velké paměťové nároky. [3]

Existuje mnoho dalších hraničních reprezentací dat. My se ale nyní zaměříme na objemové reprezentace těles, které jsou vzhledem k tématu práce důležitější.

3.1.2 Objemové reprezentace těles

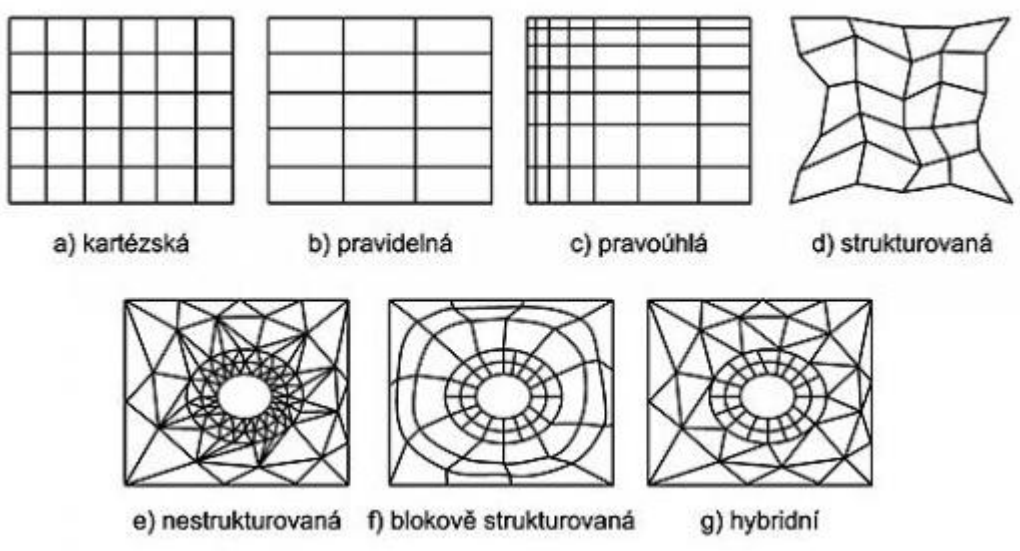
Často se můžeme setkat s tím, že máme k dispozici pouze sadu vzorků, které nesou určitý typ informace (hustota, jas, souřadnice). Ta se však vztahuje pouze k lokálnímu umístění těchto vzorků v objemu, či povrchu a celkový popis tělesa nemusíme mít k dispozici.

Pokud je sada v podobě rozptýlených dat, tak každý samostatný vzorek musí obsahovat hodnoty jasu a hustoty v daném místě (bodě) a také hodnoty svých souřadnic. Vzorky mohou být také uspořádány do tvaru pravidelných a nepravidelných mřížek.

Objemové modely tak v zásadě rozdělují celkový objekt na spousty menších, kterým se říká *voxely*. Voxelem tedy označujeme nejmenší jednotku trojrozměrného prostoru. Můžeme si ho představit jako 3D verzi pixelu (základní jednotka 2D prostoru). Má tvar krychle, nebo kvádru, který je úplně vyplněný, nebo úplně nevyplněný konstantní hodnotou. [3] [4]

Pokud bychom se konkrétně zaměřili na možnosti zobrazování medicínských dat, získaných pomocí CT a MRI tak zjistíme, že ačkoli každá z těchto metod pracuje na odlišném principu, způsob reprezentace dat je pro obě shodný a má tvar mřížky. Proto se nyní můžeme věnovat mřížkovým reprezentacím. [5]

Mřížky dělíme na pravidelné a nepravidelné. Toto základní rozdělení se dále rozvádí do celkem 7 tříd. První čtyři třídy (kartézská, pravidelná, pravoúhlá a strukturovaná) jsou strukturovány do podoby pravidelných mřížek. Tyto třídy se od sebe navzájem liší různými deformacemi svého tvaru. Další třídou je nestrukturovaná mřížka. Její buňky mohou mít prakticky jakýkoli tvar, a proto je nutné u nich uchovávat odkazy na vrcholy, ze kterých vznikly. Celá struktura této mřížky je pak uchovávána v poli buněk. Poslední dvě třídy (blokově strukturovaná a hybridní) už jsou pak pouze kombinacemi tříd předchozích. Všechny uvedené druhy mřížek můžeme vidět na obr.2. [3]



Obr. 2 - Druhy mřížek (převzato z [3])

3.1.3 DICOM

Digital Imaging and Communications in Medicine, neboli zkráceně DICOM je mezinárodní standard pro zpracování, reprezentaci a přenos medicínských obrazových dat. Byl vytvořen společností National Electrical Manufactures Association (zkráceně NEMA). Jeho hlavním smyslem je definovat univerzální a jednotný formát dat tak, aby nebylo nutné implementovat specifické protokoly při jejich zpracování na odlišných zařízeních od různých výrobců. Aby to bylo možné byl definován speciální souborový formát DICOM. [6]

Obr. 3 – DICOM logo (převzato z [6])

Souborový formát DICOM je rozšíření staršího formátu, který byl součástí NEMA standardu. Soubor DICOM obsahuje *hlavičku* (nese informace o typu scanu jakým byly snímky získány, jméno pacienta, nebo informaci o dimenzi snímku) a obrazová data. Ta mohou být komprimována a díky tomu můžeme zmenšit finální velikost souboru. Podrobné informace o struktuře tohoto formátu je možné dohledat například v [6].

3.2 Zpracování dat

3.2.1 Segmentace

Segmentace je velmi důležitý prostředek při analýze dat. Jedná se o soubor metod, které mají za úkol rozdělit obraz na několik částí se společnými vlastnostmi. Segmentaci můžeme v zásadě rozdělit na dvě části.

- *Kompletní segmentace* – výsledné oblasti korespondují se vstupním obrazem. Pokud je obraz tvořen kontrastními objekty na konstantním pozadí, je možné využít globální postupy a dosáhnout tak kompletní segmentace obrazu na objekty a pozadí. Tyto postupy nevyužívají model objektu ani znalosti o požadované segmentaci.
- *Částečná segmentace* - oblasti nemusí přímo odpovídat vstupnímu obrazu. Postupy pro částečnou segmentaci rozdělují obraz na části podle určitých vlastností, jako jsou například jas, barva a další.

Při segmentaci je rozumné postupovat tak, že se nejdříve získá částečná segmentace, jejíž výsledky jsou poté ještě zpracovány postupy kompletní segmentace. Algoritmy používané při segmentaci můžeme rozdělit do tří skupin podle toho, jakých vlastností využívají.

- *Metody využívající globální znalost o obrazu (global knowledge)* – tato znalost bývá obvykle reprezentována pomocí histogramu.
- *Metody založené na určování hranic mezi částmi obrazu (edge – based)*
- *Metody orientované na regiony (region – based)*

Mezi nejjednodušší segmentační postupy patří *prahování*. Objekty, nebo oblasti obrazu můžeme charakterizovat konstantní mírou absorpce a odrazu světla od jejich povrchu. Poté můžeme využít určitou konstantu, kterou nazveme *práh*, pro oddělení objektů a pozadí. [7]

Prahováním můžeme nazvat transformaci vstupního obrazu f na výstupní binární soubor g, podle následujícího pravidla.

$$g(i,j) = 1 \quad pro\ f(i,j) \geq T, \quad (objekty)$$
$$= 0 \quad pro\ f(i,j) < T \quad (pozadí)$$

- kde T je zvolený práh

Pokud jsou od sebe objekty navzájem dobře oddělitelné a jejich stupně šedi jsou dostatečně rozdílné od stupně šedi pozadí, je prahování vhodnou metodou segmentace.

Rozsáhlejší popis této metody a principy ostatních segmentovacích metod (především metody Graph Cut používané v projektu LISA), můžeme najít v [7].

3.2.2 Resekce podle roviny

Rovina nachází uplatnění v mnoha aplikacích. Je možné jí definovat několika způsoby, přičemž nejjednodušším je popis Kartézskou rovnici ve tvaru:

$$Ax + By + Cz + D = 0 \quad (1)$$

Po dosazení numerických hodnot za koeficienty A, B, C a D definujeme specifickou rovinu v prostoru. Pokud souřadnice libovolného bodu po dosazení vyhovují rovnici (1), tak můžeme prohlásit, že tento bod leží na této rovině. Rovnice (1) může být zjednodušena v případě, že se nám vynulují některé z koeficientů. Pokud se tak stane, rovina je poté kolmá na osu příslušného nulového koeficientu.

Mnoho problémů v počítačové grafice se týká toho, jak rozdělit vizualizovaný objekt. Vlastně se ptáme na to, v jaké části obrazu, který jsme rozdělili rovinou, leží zvolený bod, nebo zda bod leží uvnitř, nebo vně objektu jehož hrany jsou tvořeny rovinami (například čtverec).

Předpokládáme tedy, že máme zvolený bod, který označíme p_z. Postup jak rozhodnout na jaké straně roviny tento bod leží, začíná zvolením jiného bodu, který neleží v rovině. Označíme si ho p_l. Nyní využijeme obecnou rovnici roviny:

$$f(x, y, z) = Ax + By + Cz + D$$

Tuto rovnici vypočítáme pro oba body dosazením jejich souřadnic.

$$f(x_{pz}, y_{pz}, z_{pz}) \text{ , } f(x_{pl}, y_{pl}, z_{pl})$$

Rozhodnutí na jaké straně roviny leží zvolený bod, se poté provede podle následujících třech kritérií.

1. $f(x_{pz}, y_{pz}, z_{pz}) = 0$, p_z leží přímo na rovině
2. $f(x_{pz}, y_{pz}, z_{pz}) > 0 \;\; f(x_{pl}, y_{pl}, z_{pl}) > 0$

   $f(x_{pz}, y_{pz}, z_{pz}) < 0 \;\; f(x_{pl}, y_{pl}, z_{pl}) < 0$

   } , p_z leží na stejné straně jako p_l
3. Pokud ani jedno z předchozích kritérií nevyhovuje, tak p_z je na opačné straně než p_l [8]

4. Softwarové prostředky

V této kapitole se nejprve zaměříme na podrobnější popis způsobu vizualizace a reprezentace dat pomocí VTK. Dále si popíšeme programovací jazyk Python společně s některými jeho knihovnami. Také se zaměříme na možnosti, které nám Python poskytuje při vytváření grafického uživatelského prostředí.

4.1 The Visualization Toolkit (VTK)

Jedná se o software obsahující třídy pro vizualizaci, ukládání a zpracování dat .VTK bylo vyvinuto v jazyce C++, ale postupem času se upravovalo i pro použití v ostatních jazycích, například v Pythonu.

Obr. 4 – Logo VTK (převzato z [13])

VTK podporuje velké množství algoritmů a technik pro práci s grafikou, například modelování, vyhlazování a mnoho dalších. Je multiplatformní a běží tedy bez problémů na různých operačních systémech (například Linux, Windows). Také může spolupracovat s GUI nástroji Pythonu jako je například PyQt, nebo Tkinter. [9]

4.1.1 Vizualizace dat ze zobrazovacích metod

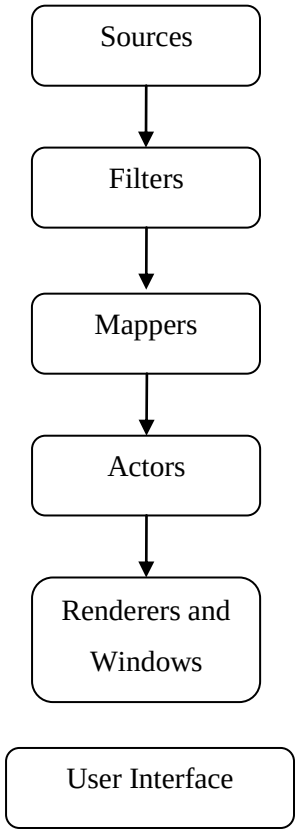
Existuje několik definic vizualizace. Například v [10, s. 5] je vizualizace definována jako: „Proces objevování, transformování, a zobrazování dat jako obrazů (nebo jiných smyslových forem) pro zvýšení porozumění a pochopení dat. ‘‘[1]. V lékařství se vizualizace zpravidla provádí ve 2D provedení (například CT snímky), ale občas se vyskytují situace ve kterých je zobrazení ve 3D přínosnější. To platí například pro pomoc při chirurgických zákrocích, u kterých je práce ve 3D názornější.

Základní částí procesu vizualizace je zdroj dat. Při vizualizaci dat z lékařských snímků jsou zdrojem dat například segmentovaná data, DICOM, případně jiné formáty. Tato data jsou následně transformována a mapována do vhodné formy pro konečnou vizualizaci. Finální částí tohoto procesu je *renderování*. Renderováním můžeme nazvat proces generování obrazů pomocí počitačů, nebo také převádění dat do podoby obrazů. [10]

[1] Text v původní formě (angličtině) – *Vizualization is process of exploring, transforming, and viewing data as images(or other sensory forms) to gain understanding and insight into the data.*

4.1.2 The Visualization Pipeline

Postup vizualizace uvedený v předchozí kapitole se používá ve VTK, kde je nazývaný jako *vizualizační potrubí* (v originálním tvaru the visualization pipeline). Jeho strukturu můžeme vidět na obrázku č. 6.



Obr. 5 – Vizualizační pipeline (převzato a upraveno z [11])

Objekty Sources (Zdroje) jsou zdroje dat. Mohou být použity dvěma způsoby a to jako *nezávislé zdroje* (struktura dat je zadávána pomocí parametrů), nebo jako tzv. *readery* (struktura dat je uložena v datovém souboru, ze kterého je načtena právě pomocí readeru). Jako příklad nezávislého zdroje můžeme uvést třídu *vtkSphereSource* (pro vytvoření koule), a jako příklad readeru třídu *vtkUnstructuredGridReader* (pro načtení dat z VTK souboru ve tvaru nestrukturované mřížky). [10] [11]

Objekty označované jako *filtry* (Filters), přijímají data od ostatních komponent pipeline, které pak modifikují vhodným způsobem, aby data mohla být použita ostatními objekty. Vyžadují tedy jeden vstup od datového objektu a generují jeden či více výstupů, které jsou opět datovými objekty.

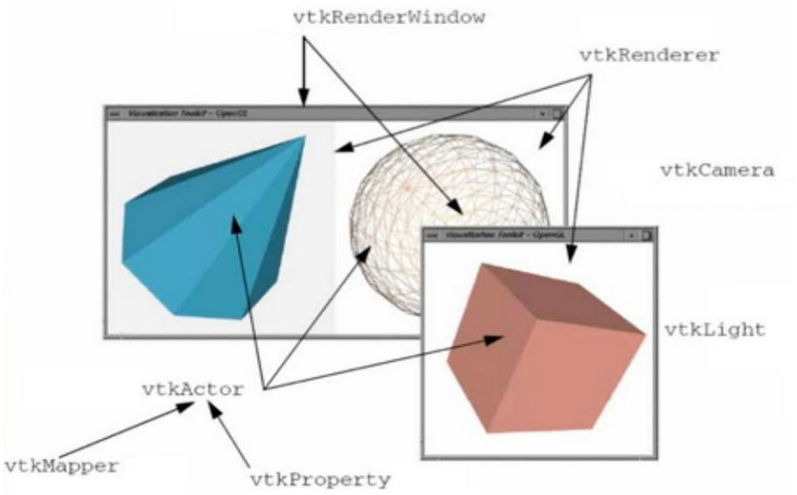
Mohou například redukovat větší soubor dat na menší, nebo slučovat více vstupů do kombinovaného výstupu. Filtry mohou být snadno zaměněny za „mappery“. Hlavní rozdíl mezi nimi je, že filtr se nachází v části pipeline nazývané jako *zpracování dat*, zatímco mapper používáme jako přechod mezi touto částí a druhou částí pipeline nazývané jako *renderování*. Také platí, že výstup mapperu je používán jako vstup actoru. Pokud je výstup použitý jako vstup pro jakoukoli jinou komponentu jedná se o filtr. [10] [11]

Actors (v přeloženém tvaru pravděpodobně *aktéři*) reprezentují grafická data, nebo vizualizované objekty. Objekty *Renderers and Windows* reprezentují to, co vidí uživatel, tedy konečné zobrazení.

4.1.3 Základní objekty pro vizualizaci ve VTK

Nyní si ukážeme jak vytvořit grafické (renderovací) okno, do kterého budeme vizualizovat. Pro tyto účely máme ve VTK k dispozici velkou škálu objektů. My si z nich uvedeme sedm základních a nejvíce používaných. [10]

- *vtkRenderWindow* - Základní objekt, který musíme vytvořit pokud chceme renderovat scénu ve VTK. Spravuje okno, do kterého budeme vizualizovat. Vykreslovat do něj můžeme za pomoci jednoho či více renderů.
- *vtkRenderer* – Tato třída zodpovídá za spolupráci světel, kamery a actoru, tak, aby byl vytvořen celkový obraz. U vtkRendereru požadujeme, aby byl vytvořen alespoň jeden actor, narozdíl od kamery a světel, které nemusí být definovány, protože si je vtkRenderer dokáže vytvořit automaticky. V této třídě jsou rovněž metody pro specifikaci pozadí a barev.



Obr. 6 – Základní grafické objekty (převzato a upraveno z [10])

- *vtkLight* – definuje zdroj světla a tím osvětluje scénu. Jak už bylo napsáno v předešlém bodě, nemusí být nutně definována. Z možností, které tato třída může provádět, patří mezi nejdůležitější zapnutí, nebo vypnutí osvětlení a nastavení pozice a barvy světla.
- *vtkCamera* – slouží pro vytvoření kamery. Ta definuje pozici a úhel pohledu, kterým se na vizualizovaný objekt díváme.
- *vtkActor* – objekt této třídy tzv. actor reprezentuje těleso v renderovacím okně.
- *vtkProperty* – pomocí třídy vtkProperty můžeme nastavovat a měnit vlastnosti daného actoru, jako je barva, průhlednost, nebo světelné efekty. Při vytvoření actoru se automaticky vytvoří i instance vtkProperty. Tu můžeme vytvořit i přímo, což přináší výhodu mnohonásobného použití stejných vlastností pro více actorů.
- *vtkMapper* – definuje geometrii objektu a „mapuje‘‘ data do podoby vhodné pro renderovaní. [10]

4.1.4 Interaktivní nástroje ve VTK

Základním nástrojem při 3D vizualizaci je možnost měnit pozici kamery tak, abychom mohli na scénu nahlížet z různých úhlů. Jednou z možností jak to provést, je změnit programový kód (například zadat nový úhel natočení tělesa) a program znovu spustit. To je však velmi nepraktické a proto do programu zavádíme další třídu, kterou je ***interaktor*** (např. *vtkRenderWindowInteractor)* a třídu definující styl interaktoru (např *vtkInteractorStyle*).

- *VtkRenderWindowInteractor* – *sleduje* události vytvořené myší, nebo klávesnicí a ty následně převádí na tzv. VTK události (VTK events)
- *vtkInteractorStyle* – definuje chování scény, když nastane událost vytvořená třídou interaktoru.

V našem programu použijeme třídu *vtkInteractorStyleTrackballCamera()*. Poté na levém tlačítku myši budeme mít definovanou funkci rotace a na pravém tlačítku funkci přibližování a oddalování [10]

Další nástroj, který budeme potřebovat je objekt třídy ***vtkImplicitPlaneWidget***. Tento ovládací prvek slouží pro vytvoření a pohyb virtuální 3D roviny v prostoru. Při vytvoření tohoto objektu se nám ve vizualizačním okně objeví rovina s šipkou na tuto rovinu kolmou. Tato šipka představuje normálu, kterou můžeme uchopit kurzorem a s její pomocí můžeme rovinu libovolně natáčet. Pokud kurzorem uchopíme přímo rovinu, můžeme ji v prostoru posouvat.

Abychom mohli objekt třídy vtkImplicitPlaneWidget používat v programu, musíme mu předat interaktor z třídy vtkRenderWindowInteractor. Toto předání se provede příkazem *SetInteractor()*. [12]

Dále potřebujeme objekt třídy ***vtkPointWidget***. Tato třída je určená pro vytvoření bodu ve 3D prostoru za pomoci 3D kurzoru. Kurzorem je možné pohybovat v ohraničeném prostoru, ve kterém je bod reprezentován křížem. Zobrazení hranic prostoru, kde můžeme s bodem pohybovat, je možné vypnout. Aby bylo možné objekt této třídy používat musíme mu předat interaktor (stejně jako v předchozím případě u třídy vtkImplicitPlaneWidget). Bodem můžeme pohybovat za pomoci myši (levé tlačítko), nebo můžeme zvětšit či zmenšit prostor kde s ním můžeme pohybovat (pravé tlačítko). [12]

4.1.5 VTK soubor

VTK má vytvořeny i svoje vlastní možnosti uchovávání a reprezentace dat a to ve formě VTK souborů. Ty mohou být vytvořeny ve dvou různých formátech, kterými jsou Legacy a XML.

Formát ***Legacy*** je textový formát, jehož velkou výhodou je názornost a srozumitelnost. Data do něj mohou být ukládána jak ručně tak i programově. Jeho základní struktura se dělí na **5 základních částí**.

1. část obsahuje pouze jednu řádku. Ta slouží jako identifikátor ve tvaru:

vtk DataFile Version x.x

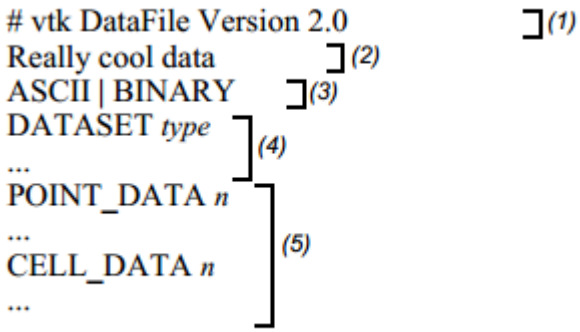
Tato forma musí být přesně dodržena s tím, že místo x.x se uvádí příslušná verze VTK souboru, která se mění v závislosti na verzi VTK (v době vytváření této práce byla nejnovější verze 3.0, ale obě předchozí, 1.0 a 2.0 jsou s ní kompatibilní).

2. část obsahuje především jméno modelu a některé dodatečné informace (nemusí být zadány). Její délka je omezena na 256 znaků

3. část udává formát v jakém je soubor napsán. Může to být ASCII nebo BINARY.

4. část definuje strukturu dat, které v souboru používáme. Vždy začíná klíčovým slovem DATASET po kterém následuje název jednoho z šesti možných typů, které lze použít: STRUCTURED_POINTS, STRUCTURED_GRID, UNSTRUCTURED_GRID, POLYDATA, RECTILINEAR_GRID, nebo FIELD. [10] [13]

5. část obsahuje popis vlastností dat a jejich celkové struktury. Najdeme v ní dvě důležité sekce oddělené klíčovými slovy POINT_DATA a CELL_DATA. Pod částí POINT_DATA nalezneme jednotlivé body a pod částí CELL_DATA jednotlivé buňky. Číslo, které se nachází za klíčovými slovy, udává počet jejich prvků, kterých je obecně *n*.



Obr. 7 – Obecná struktura VTK souboru (převzato a upraveno z [13])

Jelikož v našem programu používáme výhradně tvar VTK souboru v podobě nestrukturované mřížky (DATASET UNSTRUCTURED_GRID), uvedeme si přesný tvar VTK souboru při jejím použití.

Nestrukturovaná mřížka obsahuje všechny kombinace tvarů buněk, které můžeme ve VTK vytvořit, od vertexů (0D) až po voxely (3D). Nevýhodou ovšem je, že má díky své flexibilitě velmi vysoké paměťové nároky a měla by se používat jen v nejnutnějších případech. Je definována pomocí sekce bodů (POINTS), sekce buněk (CELLS) a sekce typů buněk (CELLS TYPES) . [10] [13]

- *Body (POINTS)* – *popisuje* body v prostoru pomocí trojic kartézských souřadnic. Ve VTK souboru tuto sekci najdeme pod klíčovým slovem POINTS, za kterým následuje číslo, udávající celkový počet bodů.

- *Buňky (CELLS) – obsahuje* seznamy bodů, které dohromady vytvoří buňku. Sekci buněk najdeme pod klíčovým slovem CELLS, za kterým následují dva parametry. Prvním z nich je celkový počet buněk, tedy i počet řádek v této sekci. Druhým parametrem je počet všech čísel (hodnot), které jsou v této sekci použity. Každý řádek představuje jednu buňku a začíná číslem, které udává počet bodů (vrcholů) této buňky.
- *Typy buněk (CELLS TYPES)* - obsahuje v každém řádku jedno číslo definující typ dané buňky. Sekce se nachází pod klíčovým slovem CELLS TYPES, za kterým následuje jeden parametr udávající počet buněk. Kolik typů můžeme uvést a jak vypadají se můžeme dozvědět například v [10, s. 15] .

Celková struktura VTK souboru (při použití tvaru nestrukturované mřížky) tak má v základní formě tento tvar:

vtk DataFile Version 2.0

ASCII

DATASET UNSTRUCTURED_GRID

POINTS *n float*

P_{x0} P_{y0} P_{z0}

P_{x1} P_{y1} P_{z1}

⋮

P_{xn} P_{yn} P_{zn}

CELLS *k m*

počet bodů$_1$ p_1 p_2 ...

počet bodů$_2$ p_1 p_2 ...

⋮

počet bodů$_k$ p_1 p_2 ...

CELLS TYPE *k*

Typ_buňky 1

Typ_buňky 2

⋮

Typ_buňky k

Druhou možností jak vytvořit VTK soubor je ***XML formát.*** Na rozdíl od Legacy formátu je složitější, ale podporuje mnohem více funkcí, mezi které patří například náhodný přístup k datům, nebo jejich komprimování. Existují dva typy VTK XML souborů.

- *Sériový* – všechna data jsou obsažena v jednom souboru a jsou zpracovávána samostatným procesem.
- *Paralelní* – Určeno pro aplikace s několika procesy prováděnými paralelně. Struktura dat je rozdělena do několika sekcí a každá z nich je přidělena jednomu procesu.

Data mohou být v XML formátu rozdělena do dvou forem. První se nazývá strukturovaná forma, která je reprezentována pravidelným polem buněk. Druhou kategorií je nestrukturovaná forma, která je reprezentována nepravidelným polem bodů a buněk. [13]

Více informací o XML formátu můžeme nalézt například v [13]

4.2 Python

Python je objektově orientovaný programovací skriptovací jazyk vyvinutý Guido van Rossumem. Je přenositelný a nezávislý na dané platformě. Jeho kód je tedy bez problémů spustitelný na unixových systémech (například Unixech a Linux), Microsoft Windows, Macintoshi a mnoha dalších operačních systémech. Programy, které jsou v Pythonu napsány se automaticky překládají do bytecode a ten je následně prováděn interpretem. Díky tomu není nutné programový kód kompilovat, nebo linkovat, jako v jiných programovacích jazycích (například v C) a programy v Pythonu tak mohou běžet rychleji.

Velkou výhodou Pythonu je jednoduchost. Jeho základy se dají naučit velmi rychle a už jen s těmito základními dovednostmi, se dají naprogramovat rozsáhlejší projekty. Python je zdarma a jeho použití není nijak omezeno ani pro komerční účely. Také v sobě ukrývá velmi dobré prostředky pro tvorbu grafických programů, nebo aplikací. Například pro práci s GUI Python podporuje mnoho nástrojů, mezi které patří Tkinter, wxPython, nebo PyQt. [14] [15]

4.3 Nástroje pro tvorbu GUI

Jako základní nástroj a prakticky zavedený standard pro tvorbu grafického uživatelského prostředí (GUI) se uvádí modul ***Tkinter***. Je to rozhraní, obsažené v Pythonu, které slouží pro přístup ke knihovně Tk.

S tímto rozhraním je možné vytvářet profesionálně vypadající GUI jednoduchou formou. Jeho výhodou je přenositelnost mezi operačními systémy (Microsoft Windows, Linux, Unix a další), velmi dobrá dokumentace a vývojová podpora. Nevýhodou však je, že není příliš rychlý a nehodí se tak pro práci s obrázky. Díky vysoké režii se v něm navíc obtížně vytváří některá neobvyklá, nebo specifická uživatelská rozhraní. [14] [15]

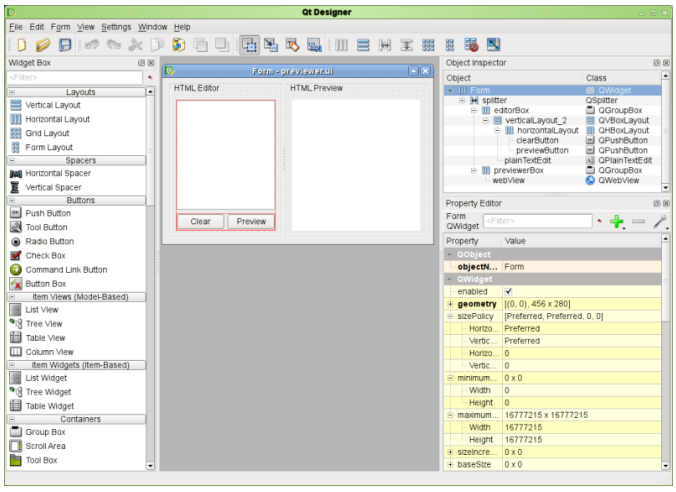
Další možností jak vytvářet GUI je ***wxPython***. Je to otevřené rozhraní pro přístup ke knihovně wxWidgets. Jelikož je postavené na třídách jazyka C++, jde o komplexnější nástroj než je Tkinter.

Jeho nevýhodou je o něco chudší dokumentace než u Tkinteru. Naopak výhodou je bohatší soubor ovládacích prvků, například pro webové aplikace v HTML. [16]

Dalším multiplatformním rozhraním pro práci s GUI v Pythonu je ***PyQt***. Toto rozhraní pracuje s knihovnou Qt, která je standardně knihovnou jazyka C++. PyQt nám poskytuje prostředí pro použití Qt v Pythonu. Je velmi rozsáhlé, k dnešnímu dni obsahuje více jak 620 tříd. [16]

Výhodou spojení Pythonu a Qt je, že díky Pythonu není nutné programy kompilovat a díky Qt se nemusí kód upravovat, protože je Qt je podporováno na většině známých platforem. Stačí, aby bylo na konkrétním počítači nainstalováno PyQt a program lze spustit bez ohledu na operační systém. [17]

Pohodlné prostředí pro vytváření GUI nabízí program ***QtDesigner***. Můžeme ho využít při tvorbě hlavních oken, rozhraní pro zprávy, grafických oken a dalších prostředků. Základní prostředí QtDesigneru pro vytváření aplikací můžeme vidět na obr. 8. Pokud chceme vytvořit grafický prvek, stačí tento prvek přetáhnout z lišty do formuláře. U vytvořených objektů je možné libovolně měnit jejich vlastnosti jako je velikost, barva, jméno a další.



Obr. 8 – Prostředí Qt Designeru (převzato z [23])

Po uložení formuláře vytvořeném v QtDesigneru je vygenerován soubor s názvem naší aplikace s koncovkou *.ui*. Abychom tento soubor mohli použít v Pythonu je nutné jej přeložit. To se provede pomocí programu *Make PyQt*, nebo spuštěním jeho konzolové aplikace mkpyqt.py v adresáři s uloženým souborem z QtDesigneru. Při spuštění aplikace mkpyqt.py se v konzoli objeví zpráva ve tvaru (předpokládáme, že název souboru z QtDesigneru je *aplikace*) :

$$.\backslash aplikace.ui \rightarrow .\backslash ui\_aplikace.py$$

Tento nový soubor s příponou *.py* již můžeme používat v Pythonu. Neměl by již být ručně modifikován, protože pokud bychom později znovu spustili program mkpyqt.py všechny změny, které jsme v souboru udělali, by byly ztraceny. [17]

4.4 Použité Pythonovské knihovny

4.4.1 Sys

System – specific parameters and function neboli *sys* je základní modul poskytující přístup k proměnným a funkcím, které spolupracují s interpretem. Do programu se importuje přes příkaz *import sys*.

Obsahuje například:

- *sys.argv* – seznam argumentů, které jsou předány skriptu při interpretaci.
- *sys.path* – seznam řetězců, které specifikují cestu k jednotlivým modulům použitých v programu [18]

4.4.2 NumPy

Numeric Python, neboli NumPy je soubor rozšíření pro Python. Tím hlavním rozšířením, které přináší, je možnost práce s vícerozměrnými objekty (velká pole, nebo matice). Zde je však několik odlišností v práci s poli vytvořenými pomocí NumPy a seznamy vytvořenými v Pythonu na které si musíme dát pozor. Například pole vytvořené v NumPy musí mít předem deklarovanou svojí délku a musíme do něj vkládat pouze objekty stejného typu. Na rozdíl od klasického Pythonu, který může velikost seznamu (pole) měnit dynamicky a ukládat do něj prakticky jakékoli typy objektů společně. Do programu se importuje přes příkaz *import numpy*. [19]

4.4.3 SciPy

Scientific Computing Tools for Python je otevřený software určený především pro vědecké práce a složitější matematické výpočty. Do programu se importuje přes příkaz *import scipy*.

V samotném jádru tohoto softwaru se nachází několik balíčků, které Scipy používá:

- *Numpy* - popsané v předešlé kapitole.
- *Scipy knihovna* - ta obsahuje kolekci speciálních algoritmů pro optimalizaci a zpracování signálu.
- *Matplotlib* - balíček sloužící k vykreslování 2D a základních 3D grafů
- *pandas, SymPy, IPython, nose* [19]

4.4.4 Argparse

Tento modul umožňuje vytvářet rozhraní pro zadávání argumentů přes příkazovou řádku. Argparse definuje, co se má vykonat po zadání příslušného argumentu od uživatele. Dále umí automaticky generovat nápovědu, nebo chybová hlášení. Do programu se importuje přes příkaz *import argparse.*

Při použití modulu argparse se nejprve vytvoří objekt příkazem *argparse.ArgumentParser*. Zde mu parametrem description můžeme zadat stručný popis toho, co program dělá a jak funguje. Tento popis se pak zobrazí při zavolání nápovědy (--help). Vytvořený objekt nese informace, nutné pro převedení příkazů z příkazového řádku, do datových typů použitelných v Pythonu. Objektu jsou jednotlivé argumenty předávány příkazem *add_argument().*

Tato příkaz může obsahovat mnoho parametrů, z nichž nejdůležitější jsou:

- *jméno* – název argumentu
- *akce* – co se má vykonat po přečtení argumentu
- *help* – popis, který se zobrazí při zobrazení nápovědy
- *const* – *konstantní* hodnota, která není čtena z příkazové řádky, ale je nutná pro běh programu. Parametr *const* může být ještě doplněn o klíčové slovo *default*. V případě, že je parametr zadán jako default je jeho hodnota rovna hodnotě předem definované v programu, pokud jí uživatel nezmění.

Pro analýzu argumentů z příkazové řádky, se volá metoda *parse_args().* Ta argumenty přečte a provede odpovídající akce. Díky této metodě také můžeme v programu přistupovat ke každému argumentu jednotlivě. Více v [18]

4.4.5 Pickle

Modul Pickle v sobě uchovává algoritmus pro serializaci a deserializaci dat (v Pythonu se tyto procesy nazývají ,,pickling‘‘ a ,,unpickling‘‘). Jde o převod Pythonovských objektů do posloupnosti bytů a naopak. Ve výchozím nastavení modul Pickle využívá ASCII reprezentaci dat, která je objemnější než binární reprezentace. Také může být využit modul cPickle, který pracuje téměř stejně jako modul Pickle, ale je rychlejší. Více v [18]

5. Implementace

Celý program je napsaný v jazyce Python a je rozdělen do dvou skriptů (*viewer3.py* a *virtual_resection.py).* Podrobný návod jak program spustit můžeme nalézt v příloze.

Skript *viewer3.py* obsahuje resekční editor a kód pro vizualizaci pomocí VTK a skript *virtual_resection.py* obsahuje kódy resekčních algoritmů. Program je možné spustit ve dvou základních režimech. Prvním z nich je *režim prohlížení* (View) a druhým je *resekční režim* (Cut). Režim prohlížení slouží pouze k vizualizaci jater. Režim resekce obsahuje resekční editor, jehož prostřednictvím volíme způsob resekce.

5.1 Vizualizace

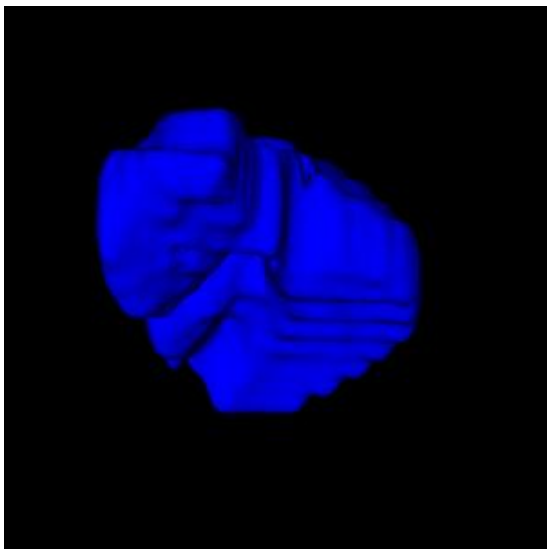
Vizualizační postup vychází z tradiční VTK pipeline, kterou jsme si definovali dříve. Jako první tedy potřebujeme data, která budeme vizualizovat. Do našeho programu se jako standardní vstup používají segmentovaná data uložená jako pickle soubor (převede se na VTK soubor), nebo už dříve vygenerovaný VTK soubor. Data z lékařských snímků jsou ovšem v datové formě DICOM. Pokud bychom tyto data chtěli vizualizovat, potřebujeme je nejdříve převést na vhodný formát a vytvořit z nich VTK soubor. Pro tento proces lze využít balíček programů nazývaný *dicom2fem* dostupný na [20]. Tento balíček je schopen transformovat jak data v podobě DICOM, tak již segmentovaná data na VTK soubor.

Pokud máme na vstupu soubor typu pickle je nutné jej nejdříve deserializovat, abychom z něj dostali potřebná data. Z těchto dat vytvoříme VTK soubor za pomoci metody *gen_mesh_from_voxels_mc* ze skriptu *seg2fem*, který najdeme rovněž v [20]. Této metodě se předávají rozměry voxelu a data zmenšená degradací (popsána v kapitole 5.3.1 Resekce podle roviny). Poté co jsme vytvořili VTK soubor, můžeme přistoupit k první části vizualizační pipeline, kterou je zdroj.

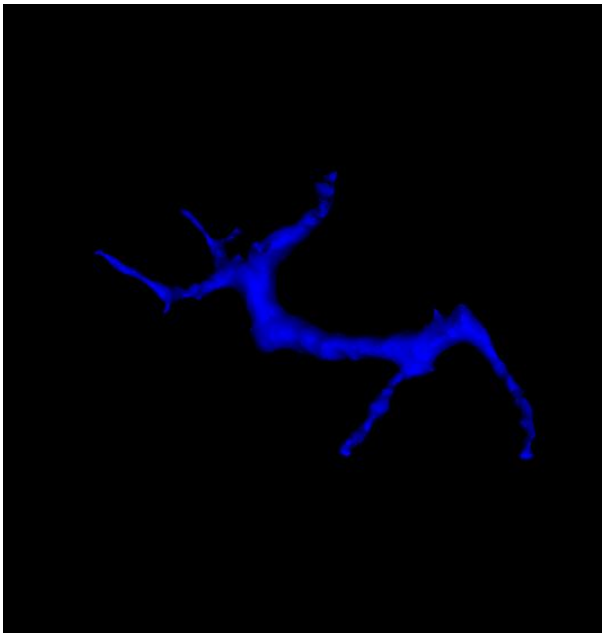
Když nahlédneme přímo do VTK souboru zjistíme, že je vždy vygenerován ve formě nestrukturované mřížky (DATASET UNSTRUCTURED GRID), pro načtení dat z tohoto souboru tak zvolíme objekt třídy *vtkUnstructuredGridReader*. Výstup readeru transformujeme filtrem *vtkDataSetSurfaceFilter*, který nám z dat extrahuje vnější povrch tělesa. (V resekčním režimu se zde využívá ještě filtr *vtkClipPolyData*, který na místech vizualizovaného objektu, kterými prošla virtuální rovina, nechává zobrazena pouze spojení buněk bez vyplnění povrchu mezi nimi. Uživatel tak snadno rozpozná část objektu, kterou označil k resekci.) Výstup filtru je předán mapperu z třídy *vtkDataSetMapper*, který data „namapuje‘‘ na grafická primitiva. Ty jsou dále předána actoru vytvořeným třídou *vtkActor*. Poté zbývá vytvořit ještě renderer (*vtkRenderer*), interaktor (*vtkRenderWindowInteractor*) a renderovací okno (*vtkRenderWindow*).

5.2 Režim prohlížení (Mód View)

Režim prohlížení je určen pouze k vizualizaci, ta se provádí podle algoritmu, který je uveden v předchozím odstavci. Na následujících obrázcích (obr. 9 a 10) můžeme vidět výsledky při použití režimu prohlížení.



Obr. 9 – Vizualizace jater (mód View)



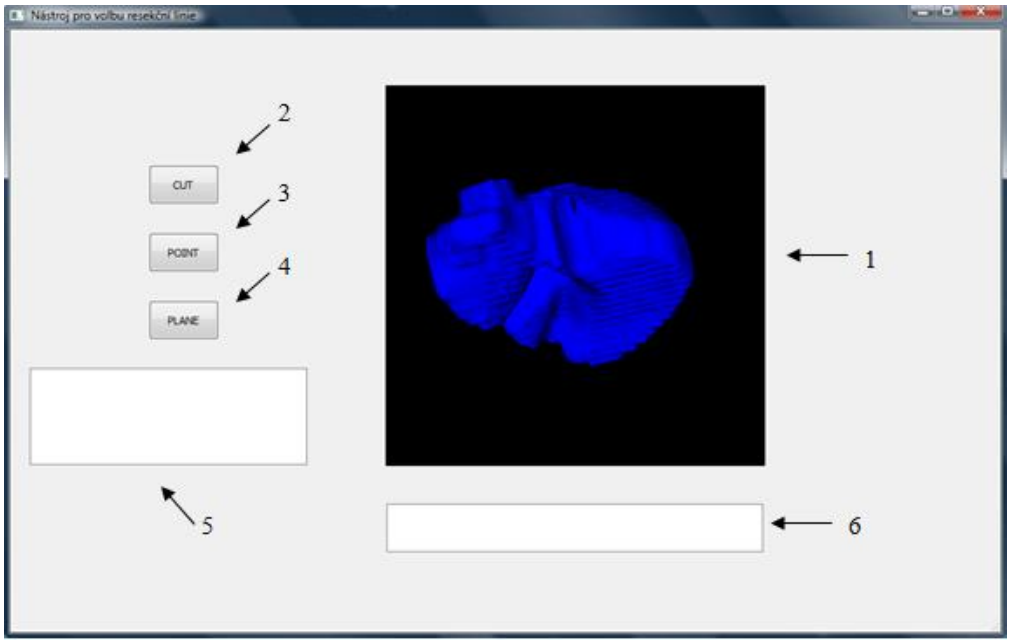
Obr. 10 – Vizualizace portální žíly (mód View)

5.3 Resekční režim (Mód Cut)

Resekční režim je určen pro volbu resekce v grafickém editoru. Resekce se v zásadě provádí na portální žíle (možno provádět i na celých játrech) a její výsledek se vždy zobrazuje na játrech.

Editor pro volbu resekční linie (obr. 11), byl vytvořen v programu QtDesigner. Nyní si popíšeme jeho jednotlivé části.

- *Vizualizační okno (1)* – renderovací okno podobné tomu, které jsme vytvářeli při vizualizaci v módu View s tím rozdílem, že QtDesigner vytváří okno pomocí třídy QWidget z knihovny PyQt.
- *Tlačítko Cut (2)* – Spouští resekční algoritmus (podle zvoleného kritéria). Před stiskem tohoto tlačítka je nutné nejprve vytvořit a nastavit rovinu, nebo bod.
- *Tlačítko Point (3)* – Vytvoří bod v prostoru, který je pak výchozím bodem pro resekci podle bodu. Bod je vytvořen třídou *vtkPointWidget.*
- *Tlačítko Plane (4)* – Vytvoří rovinu představující řez. Rovina je vytvořena pomocí třídy *vtkImplicitPlaneWidget.*
- *Informační textové pole (5)* – Zobrazuje zprávy o průběhu programu uživateli.
- *Textové pole resekce (6)* – Zobrazuje zprávy o velikosti oddělené části jater v procentech



Obr. 11 – Editor pro volbu resekční linie

5.3.1 Resekce podle roviny

Základním nástrojem tohoto resekčního algoritmu je rovina. Třída *vtkImplicitPlaneWidget* nám poskytuje dvě metody, které zde využijeme. Jednou z nich je metoda *GetNormal,* která nám vrátí normálový vektor roviny a druhou je metoda *GetOrigin,* která vrací souřadnice bodu v rovině. Na obr. 12 můžeme vidět rovinu vytvořenou při stisku tlačítka Plane. Uvnitř roviny se nachází bod, kterým můžeme pohybovat, ale pozice do které ho nastavíme, nemá na výslednou resekci žádný vliv. Hodnoty, které z roviny získáme (normála a souřadnice) se musí nejdříve transformovat v závislosti na rozměrech voxelů a velikosti degradace.

Rozměry voxelu (v programu uvedené pod jménem ***voxelsize_mm***) jsou důležitým hlediskem, které je nutné při výpočtech brát v úvahu. Velikost voxelu je v naprosté většině případů již obsažena v segmentovaných datech. Výjímkou mohou být data v původní velikosti (nezmenšená), která mají rozměry voxelu :

$$velikost\ voxelu\ (x, y, z) = (1,1,1)$$

Pokud jsou tedy rozměry voxelu rovny jedné, parametr *voxelsize_mm* nemusí být v datech obsažen a to by bez ošetření mohlo vést až k pádu programu. V případě, že by data neobsahovala rozměry voxelu i přesto, že nejsou rovny jedné, tu existuje ještě možnost předat programu tento parametr ručně. To je ale velmi výjimečná záležitost a rozměry voxelu se tak v zásadě ručně nenastavují. K parametru *voxelsize_mm* je nutné rovněž připočítat hodnotu degradace.

Vzhledem k velikosti segmentovaných dat může být doba vizualizace a následné resekce objektu velmi dlouhá. Proto se do programu zavádí proměnná ***degradace***. Tato proměnná snižuje počet dat vynecháním jejich určité části.

Pokud máme například matici M o rozměrech 6 x 6 a proměnnou degradace zvolíme 2, tak vybíráme všechny sudé řádky, respektive sloupce. Počet dat by se nám v tomto případě snížil na polovinu.

$$\begin{bmatrix} \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & \mathbf{0} & 0 & \mathbf{0} & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & 1 & \mathbf{1} & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Samozřejmě, že náš program pracuje s mnohem objemnějšími maticemi, u kterých již toto snížení má velký význam. Velikost degradace volíme především v závislosti na tom, s jakou částí jater chceme pracovat. Obecně můžeme říci, že čím více dat máme, tím větší hodnotu degradace si můžeme dovolit, aniž by se výsledný obraz zásadně změnil. Na základě experimentů se pak jako ideální nastavení ukázalo:

- *celá játra* – degradace = 5
- *portální žíla* – degradace = 2

Toto nastavení je ideální pro nezmenšená játra. U jater, která mají zmenšenou velikost, by se hodnota degradace mohla ještě zmenšit. U portální žíly může být zmenšení této hodnoty dokonce nutné, neboť portální žíla ve zmenšených datech obsahuje malé množství dat, což může v kombinaci s degradací a vyhlazováním dat způsobit rapidní zhoršení vizualizace a výsledky, které vypočítá resekční algoritmus, budou velmi nepřesné. Hodnotu degradace také můžeme zmenšit v případě, že máme k dispozici výkonnější výpočetní techniku. Naopak na méně výkonných počítačích nás degradace chrání před přetečením paměti.

Abychom při vizualizaci a výpočtech vzali v úvahu i hodnotu degradace, jednoduše jí vynásobíme s rozměry voxelu. Tím získáme výsledné rozměry voxelu, které budeme následně používat.

Aby bylo možné procházet matici dat cyklem (viz. programový kód o pár stránek dále) , je nutné upravit hodnoty získané pomocí roviny ještě z jednoho pohledu. Vzhledem k tomu, že hodnoty roviny a souřadnic byly získány z vizualizace, která (většinou) nemá jednotkové rozměry voxelu, dostávali bychom při průchodu dat cyklem s jednotkovým krokem chybné údaje.

Způsoby jak se tomu vyhnout jsou v zásadě dva. Jedním z nich je upravit krok cyklu tak, aby nebyl roven jedné, ale velikosti voxelu. Tento způsob, ale není příliš vhodný, protože může dojít k problémům (například k přetečení maximální délky cyklu). Druhým způsobem, který je použit v programu, je úprava samotných souřadnic a normály. Každou souřadnici bodu stačí vydělit příslušným rozměrem voxelu a každou hodnotu normály příslušným rozměrem voxelu vynásobit. Po této transformaci již můžeme přejít k samotnému resekčnímu algoritmu.

Normálu a souřadnice bodu dosadíme do obecné rovnice roviny.

$$f(x, y, z) = Ax + By + Cz + D$$

{normálový vektor $n = (A, B, C)$, a souřadnice bodu $p = (x, y, z)$}

Následně můžeme dopočítat parametr D

$$D = -(Ax) - (By) - (Cz)$$

Tím jsme získali kompletní rovnici popisující naši rovinu v prostoru.

Díky tomu, že lze získat souřadnice bodu, který leží přímo uvnitř roviny, můžeme modifikovat algoritmus, který jsme popsali v kapitole (**3.2.2 Resekce podle roviny**). Rozhodnutí, ve které části rovnice leží zvolený bod, se následně omezí pouze na tři kritéria.

1. $f(x, y, z) > 0$, bod leží napravo od roviny
2. $f(x, y, z) < 0$, bod leží nalevo od roviny roviny
3. $f(x, y, z) = 0$, bod leží uvnitř roviny

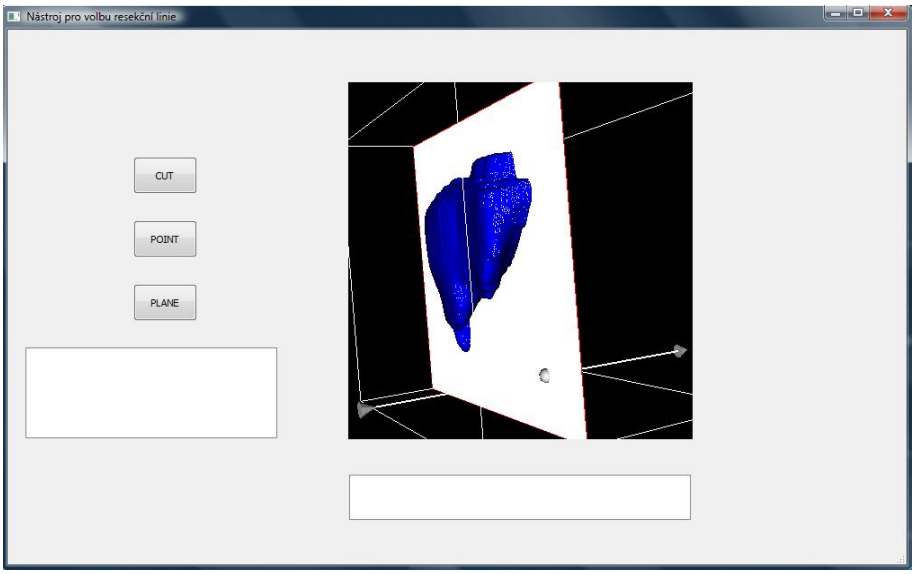
Abychom objekt rozdělili beze zbytku, zahrneme body, které leží přímo na rovině do levé části roviny. Chyba, které se dopustíme, bude minimální a na rozhodnutí nám pak stačí už jen dvě kritéria.

1. $f(x, y, z) \leq 0$, bod leží nalevo od roviny
2. $f(x, y, z) > 0$, bod leží napravo od roviny

V následné ukázce kódu můžeme vidět hlavní část resekčního algoritmu podle roviny. Algoritmus prochází data třemi cykly a pro každý bod, vyhodnotí obecnou rovnici roviny dosazením jeho souřadnic. *Leva_strana* a *prava_strana* jsou matice jedniček o stejné velikosti jako původní data. Bodu, který patří napravo je v matici *leva_strana* nastavena nula a naopak. K vizualizaci, pak primárně používáme matici *leva_strana*, která v sobě uchovává část jater, kterou chceme ponechat (v editoru ji poznáme podle toho, že není vyšrafována). Tato matice, ale stále obsahuje samé jedničky. Abychom z ní získali opět tvar jater (bez odříznuté části), vynásobíme matici *leva_strana* maticí obsahující původní data.

```
for x in range(dimension[0]):
    for y in range(dimension[1]):
        for z in range(dimension[2]):
            rovnice = a*x + b*y + c*z + d
            if((rovnice) <= 0):
                if(jatra[x][y][z] == 1):
                    levy_objekt = levy_objekt+1
                leva_strana[x][y][z] = 0
            else:
                if(jatra[x][y][z] == 1):
                    pravy_objekt = pravy_objekt+1
                prava_strana[x][y][z] = 0
```

Při tomto algoritmu zároveň ukládáme počet bodů v levé i pravé části. Tyto hodnoty následně používáme při výpočtu velikosti odstraněné části jater. Tuto velikost uvádíme v procentech a zobrazuje se v kolonce editoru *Textové pole resekce* (v době vytváření této práce funkční pouze při resekci podle roviny).

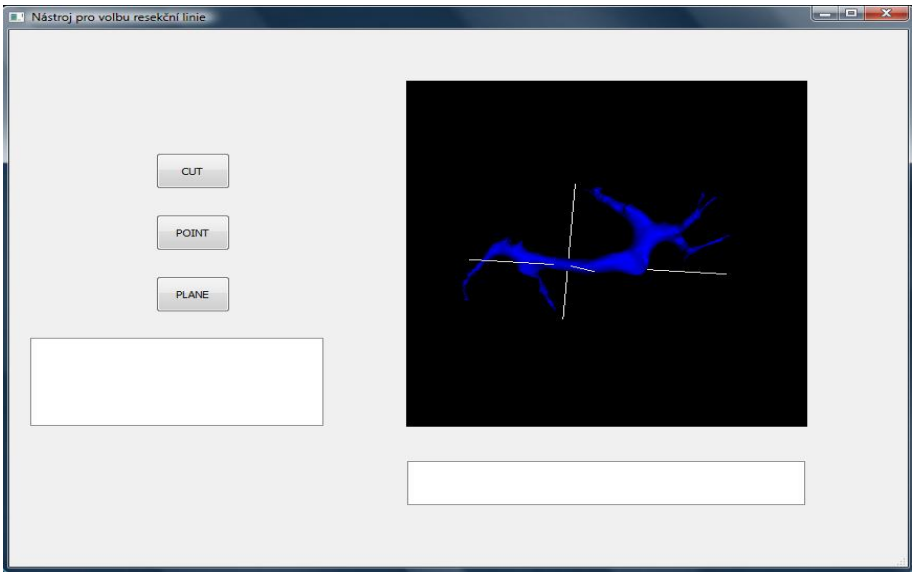


Obr. 12 – Rovina vytvořená tlačítkem Plane

5.3.2 Resekce podle bodu

Tento algoritmus byl implementován Miroslavem Jiříkem pro provádění řezu ve 2D formě. Jeho základním nástrojem je resekce s využitím kritéria podle nejmenší vzdálenosti. Kód tohoto algoritmu můžeme najít ve skriptu *virtual_resection.py* (metoda *resection*_old, spustí se při zadání parametru – *oe*). Ve 3D podobě je zatím ve zkušební formě. Na obr. 13 můžeme vidět editor při volbě bodu pro resekci. Bod se nachází na průsečíku dvou linek.

Tento algoritmus zatím nebyl nikde publikován, proto se pokusíme alespoň částečně naznačit jeho funkci. Nejdříve je vytvořena resekční matice nul o stejné velikosti, jako je matice segmentovaných dat použitých při vizualizaci. Musíme mít také na paměti, že tato data jsou zmenšena degradací a rozměry voxelu (viz kapitola 5.3.1 Resekce podle roviny). Poté co v editoru zvolíme bod (měl by ležet uvnitř portální žíly), program získá jeho souřadnice metodou *GetPosition* a hodnotu na pozici určenou získanými souřadnicemi změní na jedničku. Resekční matice je společně s maticí segmentovaných dat předána resekčnímu algoritmu, který objekt automaticky rozdělí. Toto rozdělení pracuje v zásadě tak, že zvolený bod je „nafukován‘‘ tak dlouho, až přetne portální žílu na dva kusy. Rozdělení bodů do dvou oblastí, pak probíhá na principu již zmíněného kritéria podle nejmenší vzdálenosti.

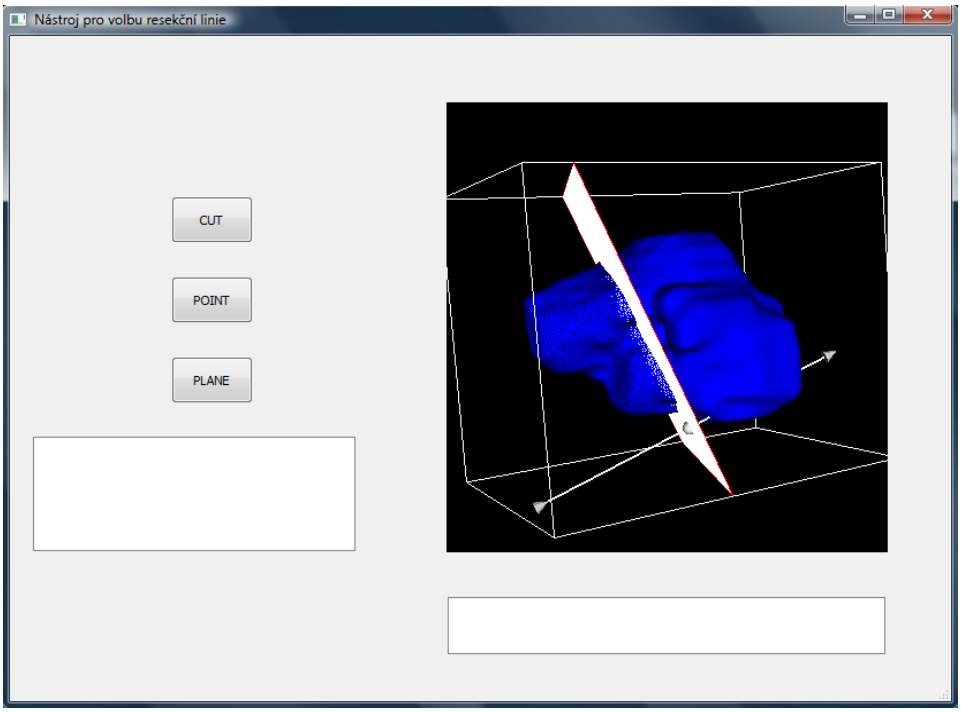


Obr. 13 – Vytvořený bod pomocí tlačítka Point

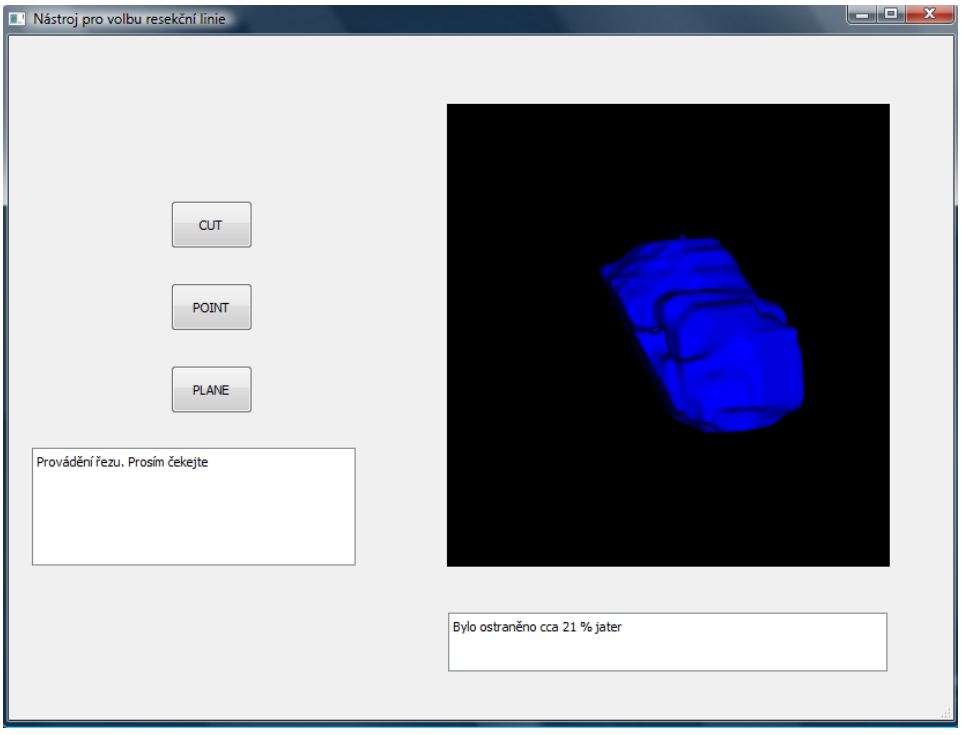
6. Experimenty

6.1 Experiment č. 1 – Nezmenšená játra

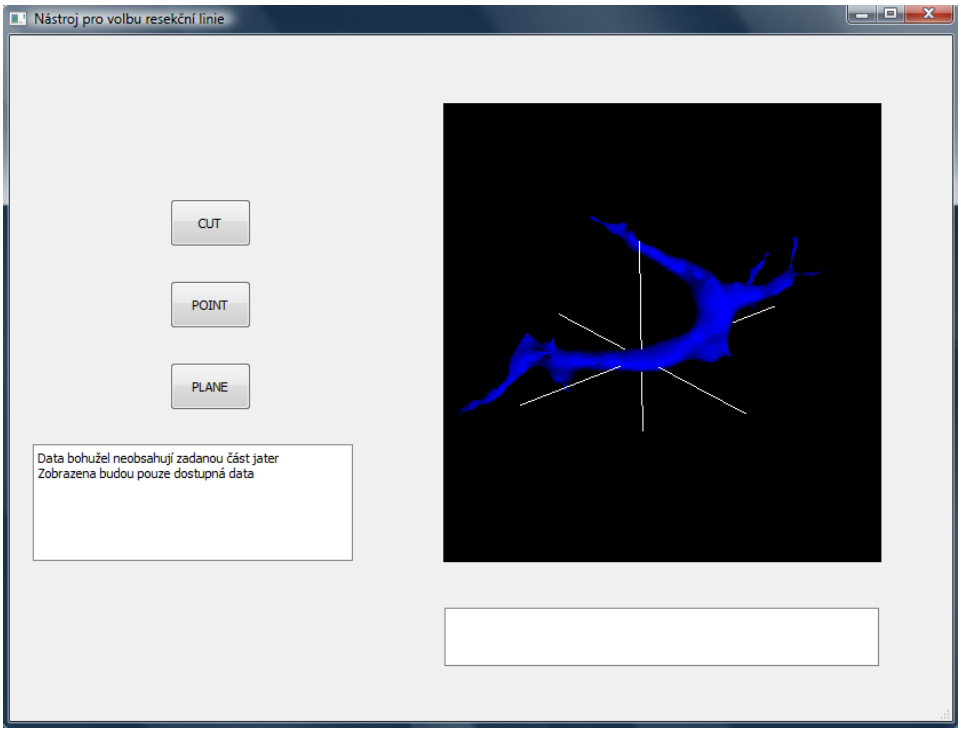
Tento experiment obsahuje vizualizaci jater a portální žíly v resekčním režimu o velikosti voxelu (1,1,1). Data, která byla využita k tomuto experimentu jsou dostupná v příloze v souboru s názvem *experiment_1*. Nejdříve je zvolena resekce podle roviny a poté resekce podle bodu.



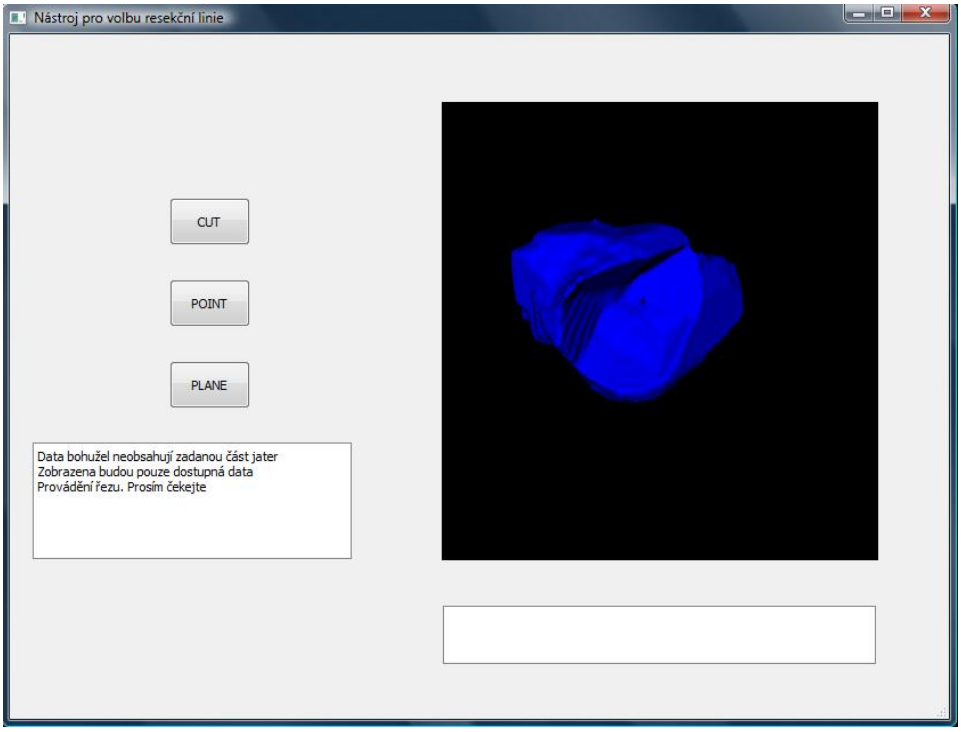
Obr. 14 – Celá játra před resekcí pomocí roviny (experiment 1)



Obr. 15 – Celá játra po resekci podle roviny z obrázku 14 (experiment 1)



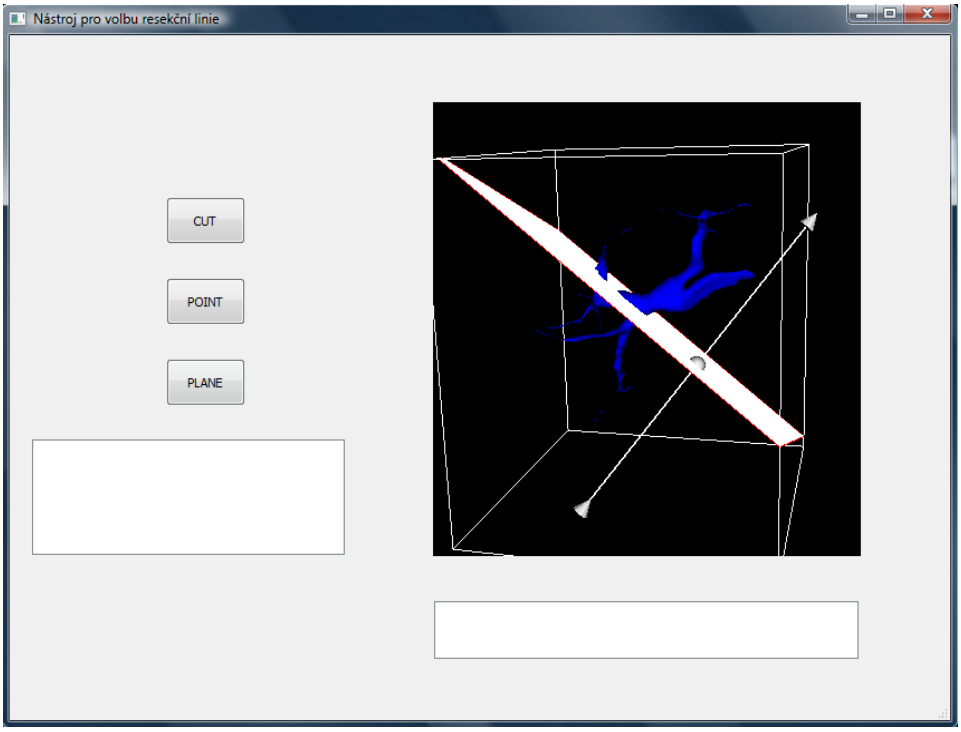
Obr. 16 – Portální žíla před resekcí podle bodu (experiment 1)



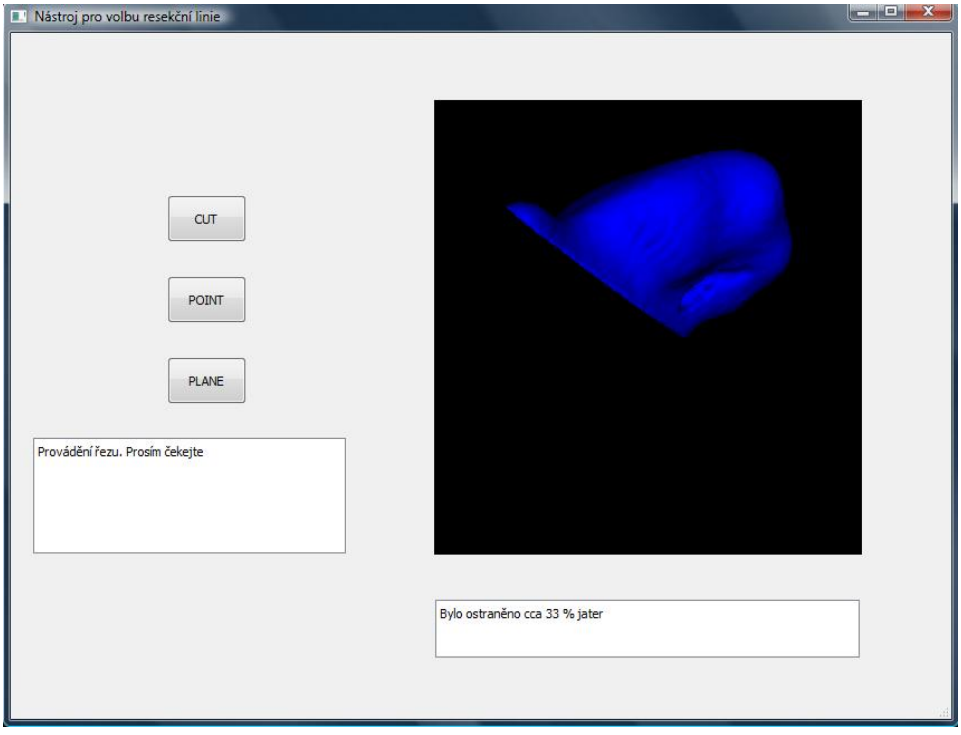
Obr. 17 – Portální žíla po resekci podle bodu z obrázku 16 (experiment 1)

6.2 Experiment č. 2 – Mírně zmenšená játra

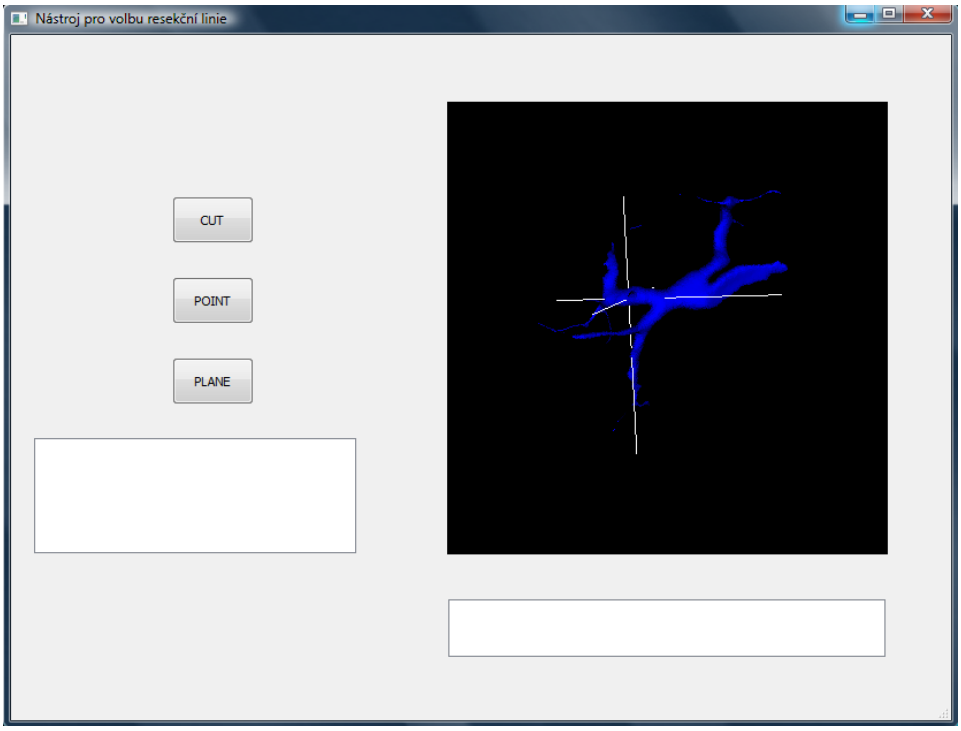
Tento experiment obsahuje vizualizaci jater a portální žíly v resekčním režimu o velikosti voxelu (1; 0,58593798; 0,58593798). Data využita k tomuto experimentu jsou dostupná v příloze, v souboru s názvem *experiment_2.pklz*. Nejdříve je zvolena resekce podle roviny a poté resekce podle bodu.



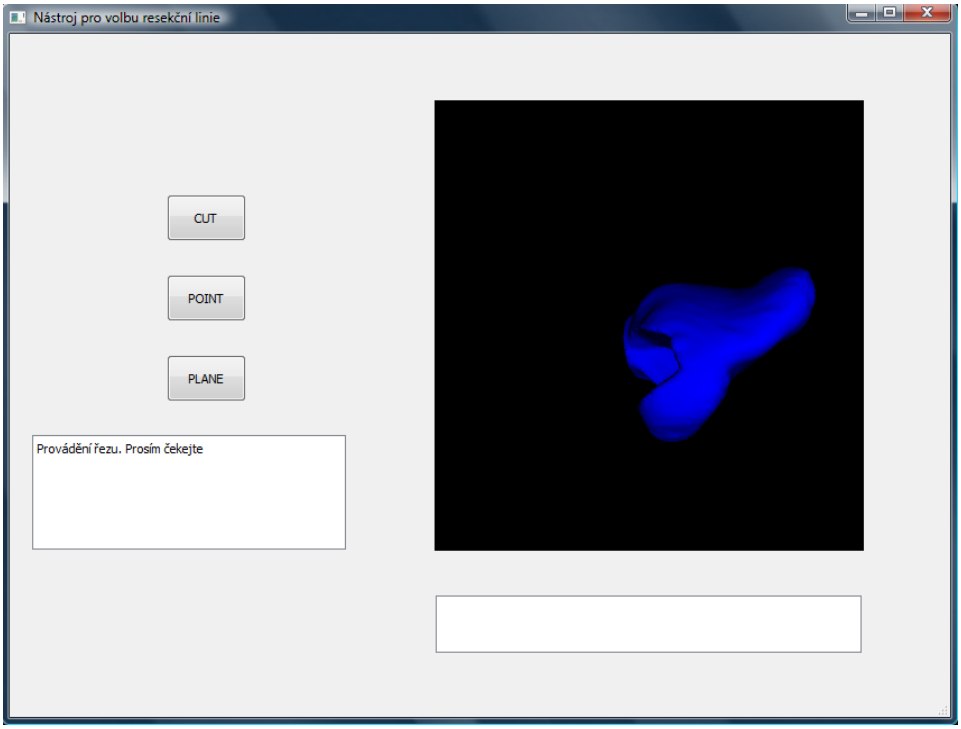
Obr.18 – Portální žíla před resekcí podle roviny (experiment 2)



Obr. 19 – Celá játra po resekci podle roviny z obrázku 18 (experiment 2)



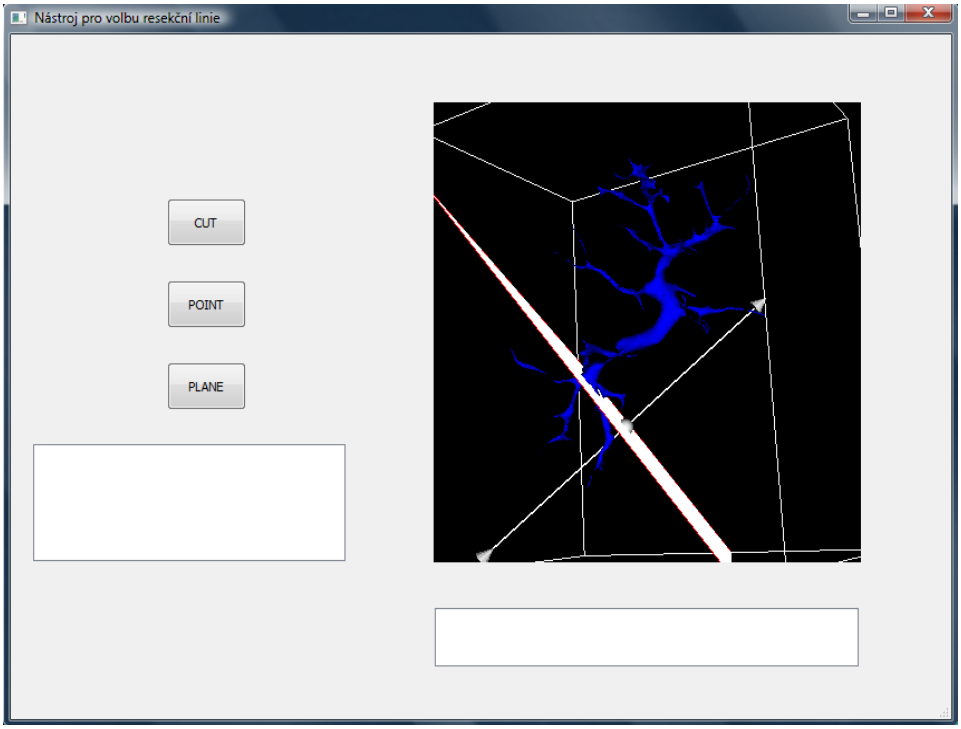
Obr. 20 – Portální žíla před resekcí pomocí bodu (experiment 2)



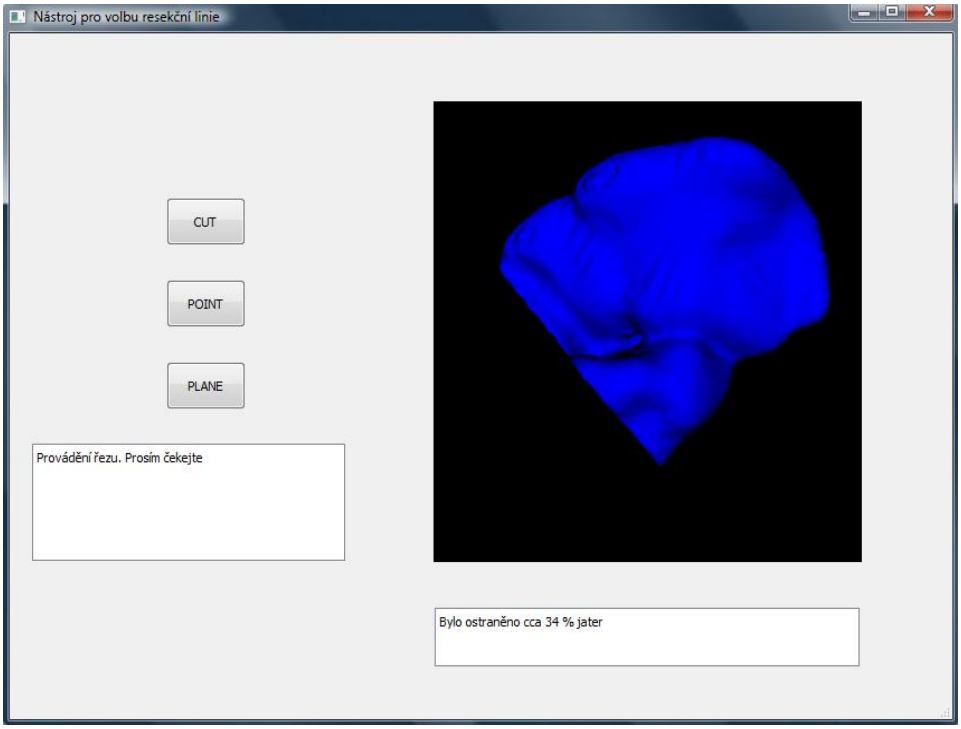
Obr. 21 – Celá játra po resekci podle bodu z obrázku 20 (experiment 2)

6.3 Experiment č.3 - Mírně zmenšená játra

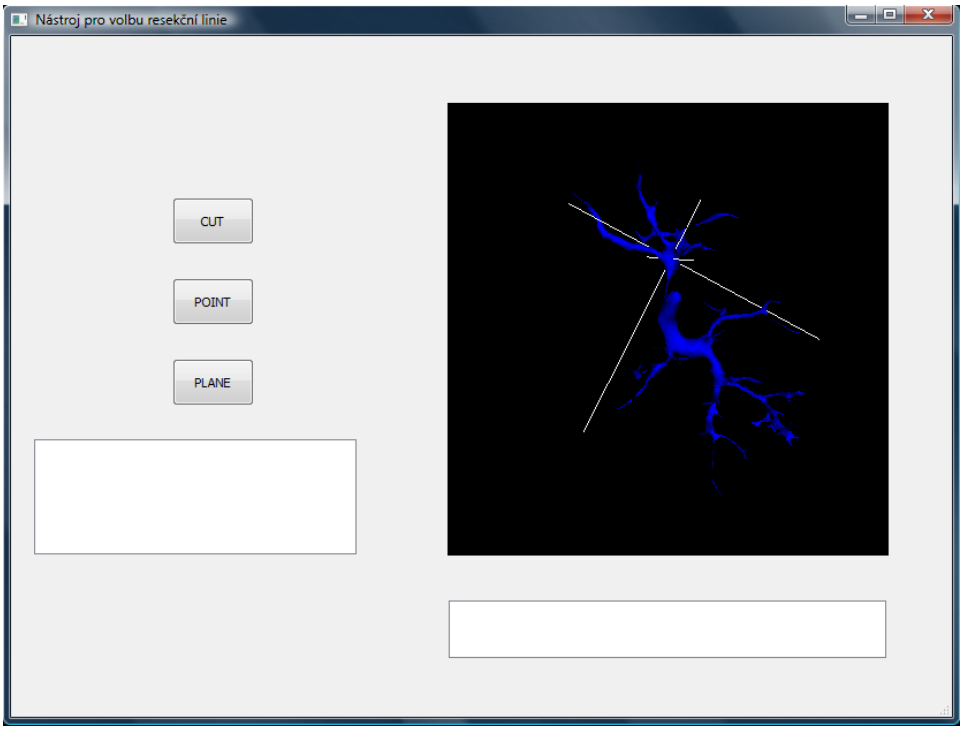
Tento experiment obsahuje vizualizaci jater a portální žíly v resekčním režimu o velikosti voxelů (1,5; 0,74219; 0,74219). Data využita k tomuto experimentu jsou dostupná v příloze, v souboru s názvem *experiment_3.pklz*. Nejdříve je zvolena resekce podle roviny a poté resekce podle bodu.



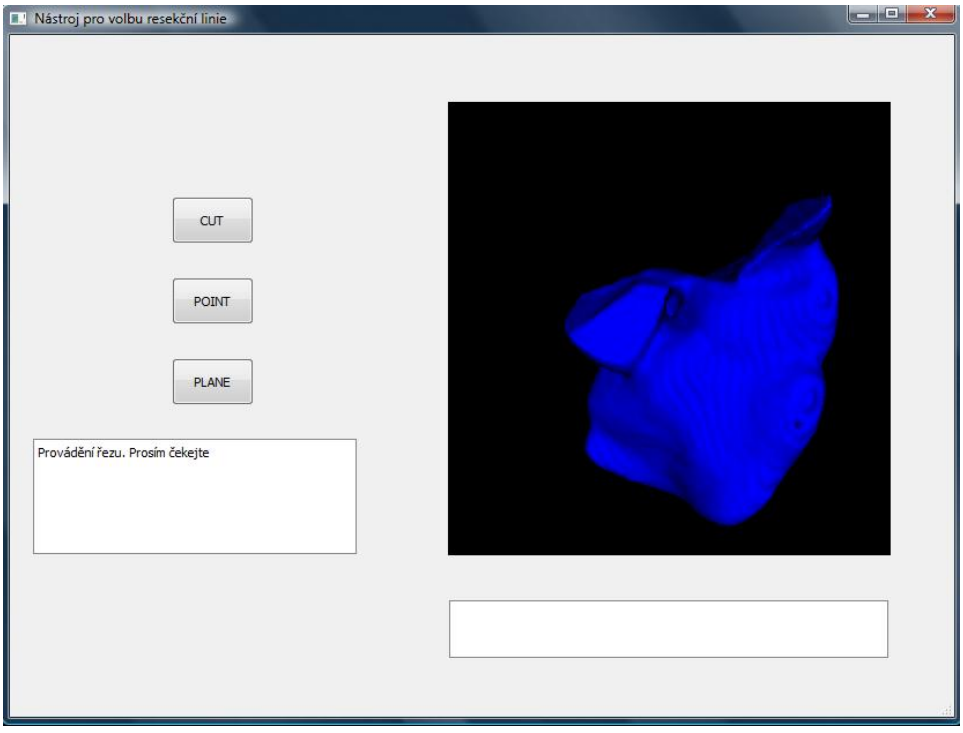
Obr. 22 – Portální žíla před resekcí podle roviny (experiment 3)



Obr. 23 – Celá játra po resekci podle roviny z obrázku 22 (experiment 3)



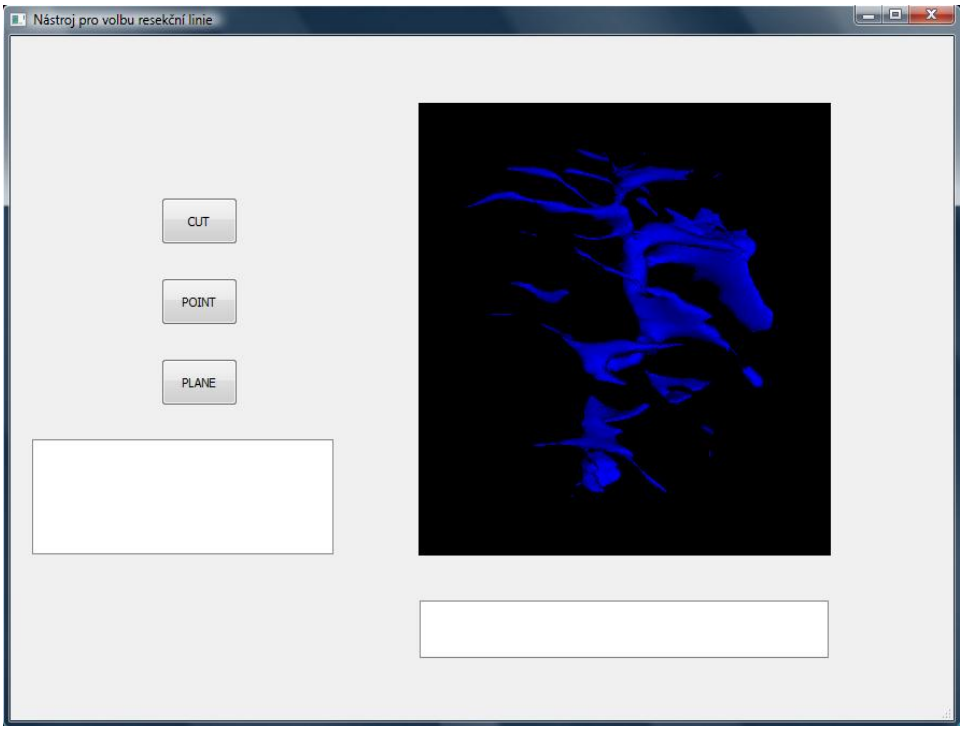
Obr. 24 – Portální žíla před resekcí podle bodu (experiment 3)



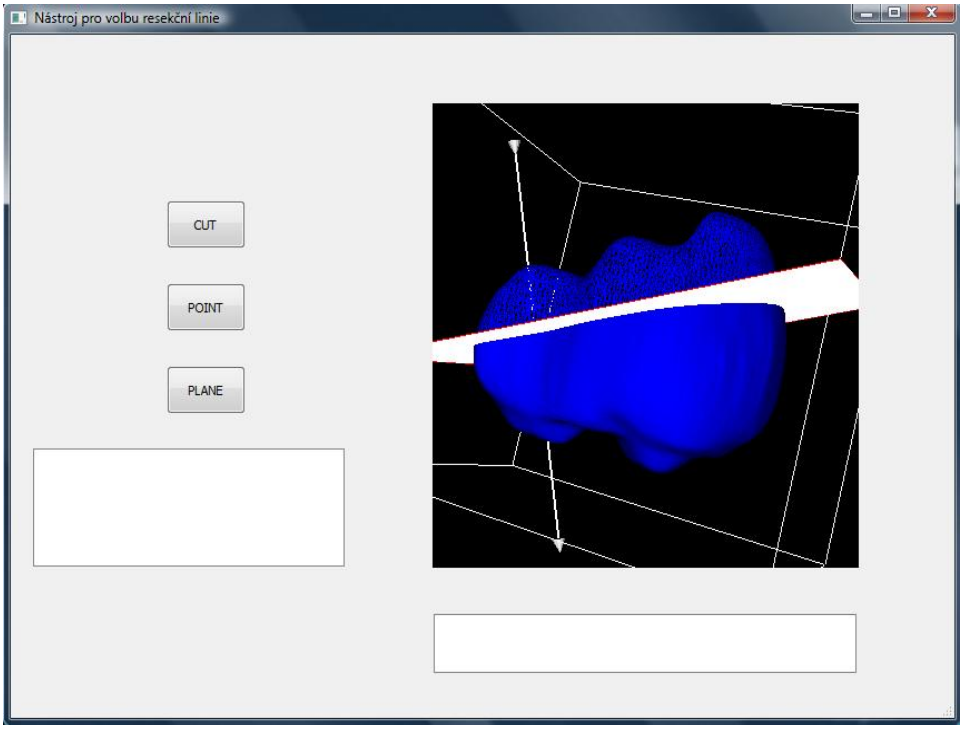
Obr. 25 – Celá játra po resekci podle bodu z obrázku 24 (experiment 3)

6.4 Experiment č. 4 – Velmi zmenšená játra

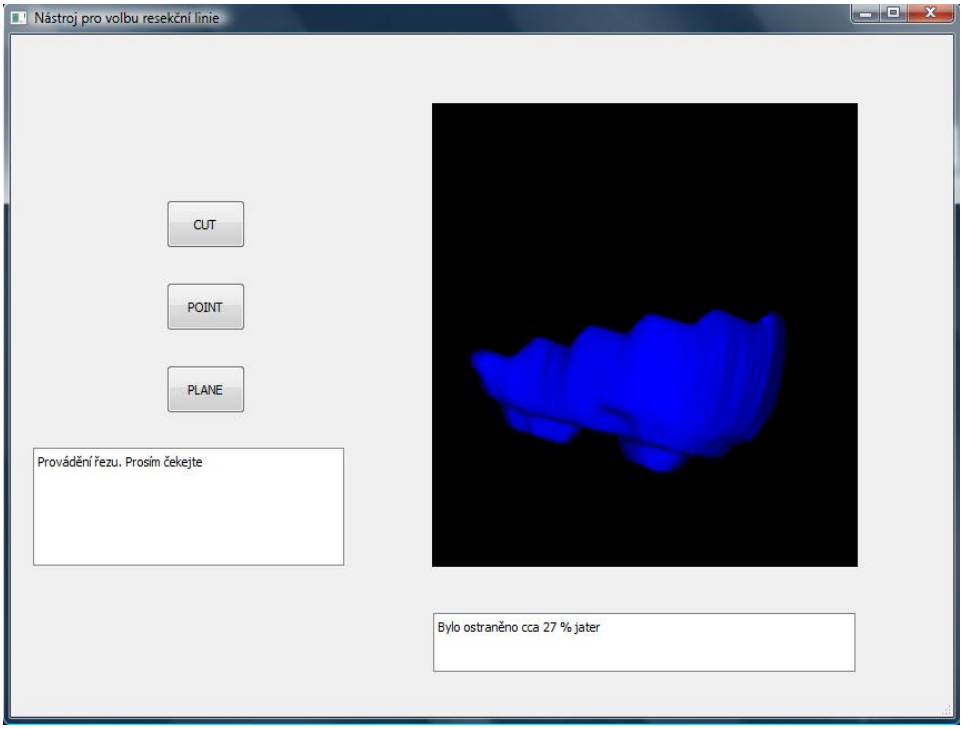
Na posledním experimentu si ukážeme některé nedokonalosti tohoto programu. Tento experiment provedeme na datech ze souboru *experiment_4.pkl*, který můžeme najít v příloze. Tyto data mají rozměry voxelu (5; 0,75; 0,75), jsou tedy velmi zmenšena. U takových dat je problém při vizualizaci portální žíly, která se rozpadá na spousty menších částí, což způsobuje problémy především při resekci podle bodu. Při resekci podle roviny způsobují zmenšená data nepřesnosti v podobě malých „hrbolů‘‘ přesahujících do odříznuté části.



Obr. 26 – Rozpadání portální žíly při vizualizaci velmi zmenšených dat



Obr. 27 – Celá játra před resekcí pomocí roviny (experiment 4)



Obr. 28 – Nepřesnost při resekci jater vytvořených z velmi zmenšených dat (experiment 4)

7. Závěr

V této práci jsme se zabývali způsobem vizualizace dat a následně vytvořením nástroje pro provedení virtuální resekce. Vizualizaci jsme provedli za použití vizualizačních knihoven softwaru VTK a to především z důvodu jeho snadného použití v programovém jazyce Python, ve kterém jsou napsány všechny programy projektu LISA. Nejdůležitější částí této práce bylo implementování resekčních algoritmů tak, aby bylo možné je využít při zpracování segmentovaných dat z lékařských snímků. Dále byl vytvořen grafický editor pro jednoduchou volbu resekční linie.

Prvním resekčním algoritmem, který byl implementován, byla resekce podle roviny. Tento algoritmus je jednoduchý a názorný. Uživatel jeho prostřednictvím může provádět resekci nastavením a natočením roviny do příslušného tvaru. Druhým algoritmem byla resekce podle bodu, která na rozdíl od předchozí metody poskytuje možnost automatického zvolení resekční linie podle bodu zvoleného uživatelem.

Funkčnost obou metod byla vyzkoušena na několika experimentech. Na těchto experimentech můžeme vidět, že přesnost výsledků, které získáme po provedené resekci, záleží především na struktuře a vlastnostech dat. Pokud jsou tato data příliš zmenšená, resekce může být provedena nepřesně. Nejlepších výsledků dosahují obě metody na nezmenšených datech, kde jsou odchylky od požadovaných výsledků minimální. Pokud jsou data zmenšené pouze částečně, metoda resekce podle roviny stále poskytuje velmi přesné výsledky. Naopak resekce podle bodu může na těchto datech mít problémy a to především z důvodu rozpadání portální žíly. Na velmi zmenšených datech může docházet k nepřesnostem u resekce podle roviny a resekce podle bodu se na takových datech stává téměř nepoužitelnou.

Programy vytvořené v rámci této práce budou v nejbližší době implementovány do již zmíněného projektu LISA a předány k testování lékařům do praxe. V návaznosti na jejich reakci poté budou algoritmy případně vylepšovány a měněny. Již teď se ale nabízí několik možných vylepšení. Například vylepšení způsobu vizualizace (aby bylo možné přesněji zobrazovat zmenšená data a tím dosáhnout lepších výsledků resekčních algoritmů), nebo vylepšení výpisů v resekčním editoru.

Seznam použité literatury

1. NEKULA, J., et al. *Radiologie*. 3. vyd. Olomouc: Univerzita Palackého v Olomouci, 2005. ISBN 80-244-1011-7.

2. ŽÁRA, J., et al. *Počítačová grafika: principy a algoritmy*. Praha: Grada, 1992. ISBN 80-85623-00-5.

3. ŽÁRA, J., et al. *Moderní počítačová grafika*. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

4. ŽELEZNÝ, M. *Zpracování digitalizovaného obrazu* [online]. [cit. 2014-4-26]. Dostupné z: http://www.kky.zcu.cz/uploads/courses/zdo/ZDO_aktual_130215.pdf

5. HUDEC, B. *Základy počítačové grafiky*. 3.vyd. Praha: ČVUT, 2001. ISBN 80-01-02290-0.

6. National Eletrical Manufactures Association. *The DICOM Standard* [online]. [cit. 2014-4-27]. Dostupné z: http://medical.nema.org

7. ŠONKA, M., V. HLAVÁČ a R. BOYLE. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thomson, 2008. ISBN 978-0-495-08252-1.

8. MORTENSON, M. *Mathematics for Computer Graphics Applications.* 2nd ed. New York: Industrial Press, 1999. ISBN 0-8311-3111-X.

9. KITWARE, I. Visualization Toolkit [online]. © 2014 [cit. 2014-5-10]. Dostupné z: http://www.vtk.org/VTK/project/about.html

10. SCHROEDER, W., K. MARTIN a B. LORENSEN. *The Visualization Toolkit:An Object-Oriented Approach To 3D Graphics*. 3. ed. New York: Kitware , 2002. ISBN 1-930934-07-6.

11. BELL, J. T. *Visualization Toolkit (VTK) Tutorial* [online]. © 2004 [cit. 2014-5-4]. Dostupné z: http://www.cs.uic.edu/~jbell/CS526/Tutorial/Tutorial.html

12. Class List. *VTK 6.2.0 Documentation* [online]., 2014 [cit. 2014-5-10]. Dostupné z: http://www.vtk.org/doc/nightly/html/annotated.html

13. KITWARE, INC. PUBLISHERS. *The VTK User's Guide:Install, Use and Extend the Visualization Toolkit*. 5. ed. Clifton Park, NY: Kitware, Inc., 2006. ISBN 1-930934-18-1.

14. HARMS, D. a K. MCDONALD. *Začínáme programovat v jazyce Python*. Brno: Computer Press, 2003. ISBN 80-7226-799-X.

15. LUTZ, M. a D. ASCHER. *Naučte se Python*. Praha: Grada, 2003. ISBN 80-247-0367-X.

16. LUTZ, M. *Programming Python*. 4.vyd. O'Reilly Media, 2011. ISBN 978-0-596-15810-1.

17. SUMMERFIELD, M. *Rapid GUI Programming with Python and Qt:the definitive guide to PyQt programming*. Upper Saddle River, NJ: Prentice Hall, 2008. ISBN 978-0-13-235418-9.

18. The Python Standard Library. *Python 3.4.0 documentation* [online]. © 1990-2014, 18.4.2014 [cit. 2014-5-10]. Dostupné z: https://docs.python.org/3/library/index.html

19. SCIPY DEVELOPERS. *Scipy* [online]. © 2014 [cit. 2014-5-10]. Dostupné z: http://scipy.org/

20. LUKEŠ, V. *Dicom2Fem* [počítačové programy]. 2013 [cit. 6.5.2014]. Dostupné z: https://github.com/vlukes/dicom2fem

21. SVOBODOVÁ, M., et al. *Lisa - Liver Surgery Analyzer Software Development*. Zatím nepublikovaný příspěvek na konferenci 6th European Conference on Computational Fluid Dynamics (ECFD VI). Španělsko:25.6. 2014.

22. CHANN, O. *ABC of emergency radiology*. 3rd ed. Chichester: Blackwell Publishing Ltd., 2013. ISBN 9780-470-67093-4.

23. Getting to Know Qt Designer. *Qt Project* [online]. © 2013 [cit. 2014-5-10]. Dostupné z: http://qt-project.org/doc/qt-4.8/designer-to-know.html

Přílohy

A. Uživatelská příručka

Naše programy (*virtual_resection.py* a *viewer3.py*) se nachazí ve složce liver – surgery na CD (složka src). Tato složka obsahuje veškeré programy používané v projektu LISA. Abychom mohli program spustit je potřeba nainstalovat několik dalších potřebných programů a doplňků. Kompletní návod jak vše nainstalovat je podrobně popsaný v souboru INSTALL.md ve složce liver – surgery na přiloženém CD, nebo na adrese https://github.com/mjirik/lisa/blob/master/INSTALL.md. Na adrese https://github.com/mjirik/lisa lze vždy nalézt aktuální verzi všech programů LISA.

Program se spouští přes překladač (v návodu je doporučený MinGW), který se ovládá podobně jako příkazová řádka. Nejdříve tedy spustíme překladač a nastavíme cestu do složky liver – surgery/src (příkaz nastavení cesty cd /c/…/liver-surgery/src). Nyní již můžeme spustit náš program a to příkazem *python viewer3.py*. Program je možné spustit ve dvou základních režimech. Prvním z nich je *režim prohlížení* (View) a druhým je *resekční režim* (Cut). Programu je také možné předat několik parametrů, které si postupně rozebereme:

- *Typ souboru (-pkl , -vtk)* – nese informace o tom, jaký typ datového souboru chceme použít pro spuštění programu. Můžeme použít celkem dva typy, z nichž jeden je.pkl a druhý *.vtk*. Při spuštění programu je důležité, aby byl datový soubor ve stejné složce jako naše programy.
- *Mód (-mode)* – tímto parametrem volíme režim, v jakém chceme program spustit. (režim resekce, nebo prohlížení)
- *Slab (-slab)* – slouží pro zadání části jater, kterou chceme zobrazit. Můžeme zadat **porta**, pro zobrazení portální žíly, nebo **liver**, pro zobrazení celých jater. (Některá data v sobě nemusí obsahovat portální žílu. Pokud v tomto případě zadáme požadavek na její zobrazení program zahlásí chybu a zobrazí celá játra).
- *Rozměry voxelu (-vs)* – umožňuje nastavit rozměry voxelu. Tento parametr se většinou nevyužíva, protože rozměry voxelu jsou již obsaženy v datech.
- *Degradace (-deg*) – nastavuje hodnoty degradace (míru zmenšení dat).

Povinný parametr je pouze typ souboru. Pokud není zadaný mód, program se automaticky spustí v zobrazovacím režimu a pokud není zadaný slab, program automaticky pracuje s celými játry. Hodnota degradace je v programu již obsažena a zadávat se rovněž nemusí.

Parametry slab, rozměry voxelu a degradace se nedají použít, pokud používáme vstupní soubor .vtk. Ačkoli je program rozdělen do dvou skriptů jeho spuštění se provádí pomocí skriptu *viewer3.py*

Příklady spuštění programu:

python viewer3.py -vtk mesh_new.vtk -mode 'Cut'

python viewer3.py -pkl experiment_1 -mode 'Cut' –deg 6 –slab 'liver'

python viewer3.py -pkl experiment_2.pklz -mode 'View' – slab 'porta' – vs [2,1,3]

B. Obsah přiloženého CD

liver – surgery	Složka, která obsahuje celý projekt LISA, včetně programů vytvořených v této práci a návodů ke spuštění. (Liver – surgery/src obsahuje hlavní programy).
Experimenty	Složka obsahující datové soubory použité při experimentech. (Před spuštěním programu je nutné je přidat do složky se skripty).
BP_prace	Obsahuje pdf soubor s bakalářskou prací.
zpracovani_CT_snimku.MOV	Video na kterém je lékař zpracovávající jednotlivé CT snímky.