

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Vizualizarea imaginilor medicale

LUCRARE DE DIPLOMĂ

Coordonator științific

Ș.l. dr. ing. Paul-Corneliu HERGHELEGIU

Absolvent

Silviu-Andrei MOTFOLEA


Iași, 2022

**DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII
LUCRĂRII DE DIPLOMĂ**

Subsemnatul(a) MOTFOLEA SILVIU-ANDREI ,
legitimat(ă) cu CI seria MZ nr. 988250 , CNP 1990714226747
autorul lucrării VIZUALIZAREA IMAGINILOR MEDICALE

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea IULIE a anului universitar 2021-2022 , luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data
07-07-2022

Semnătura


Rezumat

Folosind metodele de diagnosticare neinvazive precum tomografie computerizată sau rezonanță magnetică nucleară, rezultă un volum de date structurat, tridimensional, valorile punctelor reprezentând intensități ce variază pentru diferite tipuri de țesuturi. Metoda de vizualizare *ray casting* (proiecție a razelor) permite, spre deosebire de metodele de vizualizare pe suprafață, vizualizarea interiorului volumului fără folosirea metodei *clipping planes* (planuri de tăiere a obiectului ce permit excluderea subvolumelor din vizualizare). În redarea volumului, funcțiile de transfer sunt folosite pentru a determina proprietățile voxelului în culoare și opacitate folosind valorile luminanței.

În această lucrare este propusă o implementare a unei aplicații de vizualizare a datelor medicale folosind tehnica *ray casting*. Folosind această aplicație, pot fi create funcții de transfer unidimensionale care atribuie pentru diferite valori ale intensității în volumul de date, culoare și transparență.

Deoarece diferite organe pot avea țesuturi asemănătoare, așadar valori ale intensității asemănătoare, este dificil de construit o funcție de transfer unidimensională care să facă distincție clară între două sau mai multe regiuni similare. O soluție propusă în această lucrare este construirea unei măști de segmentare semantică cu ajutorul unei rețele neuronale. Aplicația permite atribuirea de culori și transparență pentru fiecare clasă de segmentare disponibilă, aceste valori fiind combinate cu cele din funcția de transfer.

Scopul acestui proiect este implementarea unei aplicații pentru vizualizarea datelor medicale volumetrice folosind metoda *ray casting* și oferind posibilitatea de încărcare sau modelare a funcțiilor de transfer unidimensionale. Pe lângă acestea, redarea poate fi îmbunătățită prin încărcarea sau crearea automată a unei măști de segmentare semantică. Atunci când aceste tehnici sunt aplicate pe un set de date volumetrice obținut prin imagistică medicală, poate fi obținută o reprezentare semnificativă ce poate ajuta un specialist în diagnosticarea pacienților sau un student la facultatea de medicină să vizualizeze corpul uman.

Cuprins

1	Introducere	1
2	Fundamentarea teoretică și documentarea bibliografică	3
2.1	Redarea imaginilor medicale	4
2.2	Lucrări similare	7
3	Proiectarea aplicației	11
3.1	Tehnologii folosite	11
3.2	Setul de date folosit	12
3.3	Augmentarea vizualizării folosind segmentarea semantică	12
3.4	Clase dezvoltate	14
4	Implementarea aplicației	19
4.1	Aplicația de redare	19
4.2	Interfața cu utilizatorul	22
5	Testarea aplicației și rezultate experimentale	27
5.1	Elemente de configurare	27
5.2	Rezultate obținute în aplicația de redare	28
5.3	Rezultate experimentale obținute în antrenarea modelului de segmentare semantică	29
6	Concluzii	33
	Bibliografie	35
	Anexe	37
	Anexa 1. main.cpp	37
	Anexa 2. Volume.h	42
	Anexa 3. Volume.cpp	43
	Anexa 4. Shader.h	49
	Anexa 5. Shader.cpp	50
	Anexa 6. Interface.h	53
	Anexa 7. Interface.cpp	55
	Anexa 8. VectorColorPicker.h	66
	Anexa 9. VectorColorPicker.cpp	67
	Anexa 10. SettingsEditor.h	71
	Anexa 11. SettingsEditor.cpp	73
	Anexa 12. Loader.h	79
	Anexa 13. Loader.cpp	79
	Anexa 14. PytorchModel.h	82
	Anexa 15. PytorchModel.cpp	82

Anexa 16. raycasting.vert	86
Anexa 17. raycasting.frag	86
Anexa 18. run.py	88
Anexa 19. jobs.py	90
Anexa 20. util.py	90
Anexa 21. data/loading.py	91
Anexa 22. data/visualization.py	94
Anexa 23. data/generic.py	96
Anexa 24. data/ct_org.py	98
Anexa 25. model/train.py	99
Anexa 26. model/test.py	104
Anexa 27. model/loading.py	104
Anexa 28. model/deploy.py	105
Anexa 29. model/__init__.py	105
Anexa 30. model/unet3d.py	106
Anexa 31. config/config.yaml	108
Anexa 32. config/base_data.yaml	108
Anexa 33. config/ct_org.yaml	108
Anexa 34. config/base_model.yaml	108
Anexa 35. config/UNet3D.yaml	109
Anexa 36. config/MyUNet3D.yaml	109
Anexa 37. config/load_data.yaml	109
Anexa 38. config/load_model.yaml	109
Anexa 39. config/train_model.yaml	110
Anexa 40. config/test_model.yaml	110
Anexa 41. config/deploy.yaml	110

Capitolul 1

Introducere

Datele volumetrice sunt un set de eşantioane ce reprezintă una sau mai multe proprietăţi ale unui punct într-o locaţie tridimensională. Vizualizarea volumelor constă în redarea unei proiecţii 2D a unui set de date 3D. Una dintre metodele populare pentru această sarcină este ray casting. Aceasta este un tip de redare directă a volumelor (DVR, Direct Volume Rendering) ce constă în emiterea de raze dinspre punctul de observaţie prin volum şi eşantionarea valorilor în puncte echidistante, rezultatul final fiind compunerea acestor valori. Funcţiile de transfer sunt funcţii ce transformă valorile scalare stocate în seturile de date volumetrice pentru a îmbunătăţi vizualizarea acestora.

Sarcina redării imaginilor volumetrice este mai dificilă decât cea a redării unei imagini bidimensionale deoarece datele reprezentate sunt mult mai numeroase. Spre exemplu, o imagine FHD (Full High Definition) conţine două milioane de valori RGB, adică şase milioane de valori scalare, iar într-un set de date volumetric medical uzual sunt două sute de milioane de valori scalare. Unităţile de procesare grafică (GPU) au avansat în ultimele decenii, iar în momentul actual această sarcină poate fi realizată folosind chiar şi procesoare grafice pentru consumatori.

Implementarea propusă în această lucrare se bazează pe OpenGL pentru redarea datelor volumetrice şi ImGui pentru interfaţa grafică. Algoritmul de ray casting este implementat într-o singură etapă, calculând intersecţia razelor cu un AABB¹ şi compunând culorile şi transparenţele din funcţia de transfer pentru valorile intensităţilor în puncte echidistante folosind tehnica back-to-front. Punctele extreme ale AABB-ului sunt obţinute din interfaţă, astfel putând fi excluse segmente din volumul de date, obţinându-se un efect similar cu clipping planes aliniat la axele de referinţă. Pentru a vizualiza regiunile de interes utilizatorul poate aplica transformări de rotaţie, scalare şi translaţie. Pentru acestea pot fi folosite atât mouse-ul cât şi tastatura.

Pentru obţinerea unei măşti de segmentare în mod automat poate fi utilizată o reţea neuronală antrenată pentru această sarcină, iar pentru determinarea arhitecturii şi parametrilor potriviţi vor fi necesare multiple experimente. În acest sens a fost implementată o aplicaţie care să faciliteze antrenarea reţelelor atât local cât şi folosind servicii de tip PaaS² folosind fişiere de configurare ce pot fi suprascrise din lina de comandă, astfel putând fi create modele diferite cu modificări minimale ale codului sursă. Odată obţinut un model viabil, acesta trebuie încărcat în aplicaţia de vizualizare şi, utilizând preprocesări şi postprocesări asemănătoare antrenării, poate fi redată masca de segmentare.

În capitolul 2 sunt prezentate comparativ realizări actuale pe aceeaşi temă şi sunt descrise specificaţiile privind caracteristicile aşteptate de la aplicaţie. De asemenea sunt prezentate aspecte teoretice necesare pentru proiectarea şi implementarea aplicaţiei.

¹Casetă de delimitare aliniată pe axă sau axis aligned bounding box

²Platform as a service reprezintă un tip de serviciu cloud ce permite folosirea unui mediu de dezvoltare online.

În capitolul 3 este descrisă proiectarea aplicației de redare, sunt prezentați algoritmi principali și sunt descrise componentele pe care le-am dezvoltat și implementat. Mai este descris și framework-ul cu care vor fi efectuate experimente pentru antrenarea rețelei neuronale și modalitatea în care aceasta va fi utilizată în aplicația de redare. Capitolul 4 descrie funcționalitatea sistemului și cum au fost implementate componentele principale. Este descris și modul în care sunt încărcate și salvate funcțiile de transfer și măștile de segmentare.

Capitolul 5 începe prin prezentarea modalității de compilare a aplicației și a punerii în funcționare a acesteia. Apoi sunt prezentate interacțiunile pe care le poate avea utilizatorul cu aplicația și sunt discutate câteva exemple de vizualizare a unor imagini volumetrice din diferite unghiuri și diferite niveluri de zoom. Spre finalul acestui capitol sunt prezentate rezultate experimentale pentru antrenarea rețelei neuronale folosită pentru segmentarea semantică, și sunt analizate și rezultatele aplicării acestui model în aplicația de redare. Lucrarea se încheie cu concluziile referitoare la rezultatele obținute și la utilitatea aplicației propuse, împreună cu eventuale direcții de dezvoltare.

Capitolul 2

Fundamentarea teoretică și documentarea bibliografică

Datele volumetrice rezultate prin imagistică medicală sunt aranjate pe o grilă de voxelii într-un sistem cartezian. Datele medicale pot fi obținute prin diverse modalități de scanare, cele mai uzuale fiind radiografia, tomografia computerizată (CT), imagistică prin rezonanță magnetică (RMN) și ultrasunetele. Redarea directă a datelor volumetrice este o metodă de vizualizare eficientă și flexibilă pentru implementarea unei aplicații de redare interactivă. Funcțiile de transfer sunt necesare pentru redarea directă a volumului, deoarece acestea au rolul de a face regiuni de interes vizibile prin atribuirea de proprietăți optice voxelilor, precum culoare și opacitate. Funcțiile de transfer bune permit vizualizarea regiunilor de interes fără ca acestea să fie ascunse de regiuni neimportante. În figura 2.1 ilustrat modul în care pot să difere intensitățile pentru materiale diferite. Această informație poate fi utilizată pentru modelarea unei funcții de transfer care evidențiază numai anumite țesuturi sau materiale în obiectul vizualizat. De obicei funcțiile de transfer sunt unidimensionale, adică acestea transformă în mod direct luminanța în culoare și opacitate, dar există și funcții multidimensionale care pot folosi gradientii datelor sau alte proprietăți ale acestora.

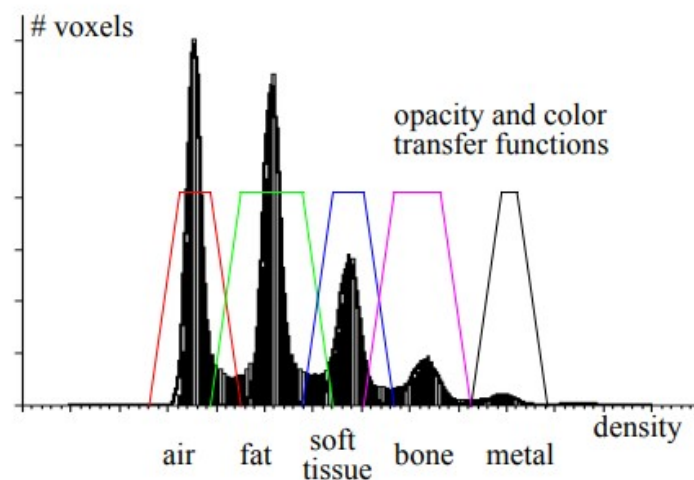


Figura 2.1: Histogramă a intensităților voxelilor și o clasificare a acestora în diferite materiale. (Sursa imaginii: Kaufman et al. 2005[1])

Crearea unei funcții de transfer bune poate fi dificilă având în vedere domeniul mare în care poate lua valori luminanța fiecărui voxel. Așadar, explorarea funcției de transfer ar trebui augmentată cu metode suplimentare de vizualizare a datelor, spre exemplu histograme ale densităților în volum. O histogramă a valorilor densității poate fi un indicator util pentru că subliniază structurile dominante cu intervale înguste de densitate[1][2]. În cazul în care

datele conțin zgomot sau sunt reprezentate multe regiuni diferite cu densități similare, poate ajuta și includerea primei derivate în analiza bazată pe histogramă. Valoarea primei derivate (rezistența gradientului) este utilă, deoarece are valori maxime locale la densități în care există schimbări între diferite caracteristici[3][1][2]. Cunoașterea densităților la care există granițele caracteristicilor restrânge considerabil sarcina de explorare a funcției de transfer[1]. Funcțiile de transfer multidimensionale folosesc gradientul volumului în timpul redării, adică diferențe centrale pentru a obține vectorul gradient la fiecare voxel. Metoda diferențelor centrale aproximează gradientul ca diferența valorilor datelor a doi voxeli vecini de-a lungul unei axe de coordonate, împărțit la distanța fizică [4][2].

O altă modalitate de a analiza datele este căutarea modificărilor topologice în izocontururile volumului, cum ar fi o îmbinare a două contururi, numite puncte critice. Prin sortarea punctelor critice în funcție de densitate se poate construi un arbore de contur. Se poate folosi arborele de contur fie pentru a genera funcția de transfer automat, sau pentru a ghida utilizatorii în procesul de explorare a volumului [1][5].

Vizualizarea datelor medicale este dificilă din cauza multitudinii detaliilor de diferite dimensiuni, dar aceasta poate fi simplificată prin evidențierea părților relevante. Tema propusă constă în implementarea și evaluarea unui sistem de redare a unor astfel de date. Integrarea unui mecanism de segmentare semantică a volumelor în procesul de redare permite utilizatorului să vizualizeze regiunile de interes.

2.1 Redarea imaginilor medicale

Ray casting a predominat ca abordarea cea mai versatilă pentru redarea directă a datelor volumetrice datorită flexibilității sale în producerea de imagini de înaltă calitate ale seturilor de date medicale și scanărilor industriale [6]. Datele volumetrice sunt încărcate într-o textură 3D *GL_TEXTURE_3D* și apoi trecute la shader ca un *sampler3D*. Acolo GLSL folosește o funcție de căutare a texturii pentru a obține o valoare de luminanță interpolată liniar.

Modelul ray casting se bazează pe calculul punctelor de intersecție dintre raze și cubul ce conține datele volumului. Pentru fiecare fragment este proiectată o rază dinspre punctul de observare. Se eșantionează iluminarea $I(x, y, z)$ din textură la puncte echidistante de-a lungul razei. Valorile opacității sunt obținute prin interpolare, iar acestea sunt apoi compuse cu fundalul prin compunerea back-to-face pentru a obține culoarea pixelului. Pentru fiecare rază trebuie să determinăm coordonatele punctelor sale de intrare și ieșire în raport cu volumul, acest lucru fiind realizat prin intersecția lor cu un AABB. Acest algoritm este ilustrat în figura 2.2.

Cea mai simplă tehnică de ray casting este cea în două treceri. Denumirea provine de la faptul că redarea este efectuată în două etape:

1. O primă trecere pentru a calcula geometria limită, adică punctele de intrare și de ieșire ale razelor generate de fiecare pixel din fereastră de vizualizare, redându-le la o textură auxiliară.
2. O a doua trecere pentru a efectua eșantionarea și compunerea efectivă.

În prima trecere caseta de delimitare a volumului, definită ca un cub de două unități, este redată într-un *GL_FRAMEBUFFER*. O transformare a modelului va compensa diferențele de dimensiune dintre axe și orientarea spațială. *GL_FRAMEBUFFER* va avea două variabile legate de acesta: una va colecta coordonatele spațiale ale punctelor de intrare, date de fața frontală a cubului, cealaltă va colecta punctele de ieșire, date de fața din spate.

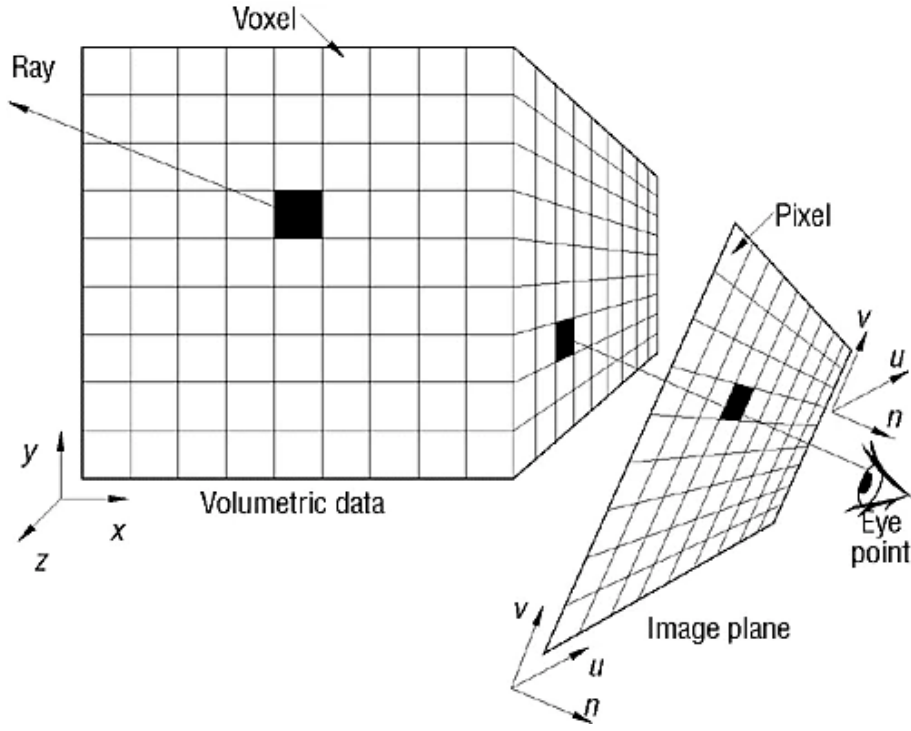


Figura 2.2: Ilustrare a principiului de funcționare a metodei ray casting de redare a imaginilor volumetrice. (Sursa imaginii: Palmer et al. 1998[7])

Deși este simplu de implementat, tehnica ray casting în două treceri este suboptimă [8]. O soluție mai bună este calcularea punctelor de intrare și ieșire ale razelor direct în shaderul fragmentului. Pentru a face acest lucru, trebuie transmise mai multe informații shaderului: poziția camerei (în coordonatele modelului) și câmpul vizual. Ecuația parametrică a razei este:

$$r = o + tv, \quad (2.1)$$

unde o este originea razei și v este direcția. În timp ce coordonata sa z este egală în valoare absolută cu distanța focală f , aceasta poate fi calculată imediat din câmpul vizual α :

$$f = \frac{1}{\tan \frac{\alpha}{2}}. \quad (2.2)$$

Tehnica de redare folosită în acest proiect este compunere alfa back-to-front (din spate în față) descrisa de ecuațiile:

$$c_i = c_i + (1 - a_i)c_{i+1}, \quad (2.3)$$

$$a_i = a_i + (1 - a_i)a_{i+1}, \quad (2.4)$$

unde c_i și a_i reprezintă culoarea și opacitatea pixelului la pasul i din compunere.

Unul dintre motivele importante pentru redare din spate în față este că permite oprirea timpurie a eșantionării atunci când culoarea fragmentului este saturată. Cu toate acestea, nu reduce foarte mult timpul de calcul din cauza naturii paralele ale procesoarelor grafice.

În redarea volumului, funcțiile de transfer [4] sunt folosite pentru a determina proprietățile voxelului în culoare și opacitate folosind valorile luminanței. Pentru a reprezenta structura unei entități înglobate într-un volum este necesară eliminarea structurilor nedorite, care ascund regiunile de interes. Acest lucru este de obicei realizat printr-o funcție de transfer care mapează valorile dintr-un set de date la anumite culori și opacități. O definiție de bază

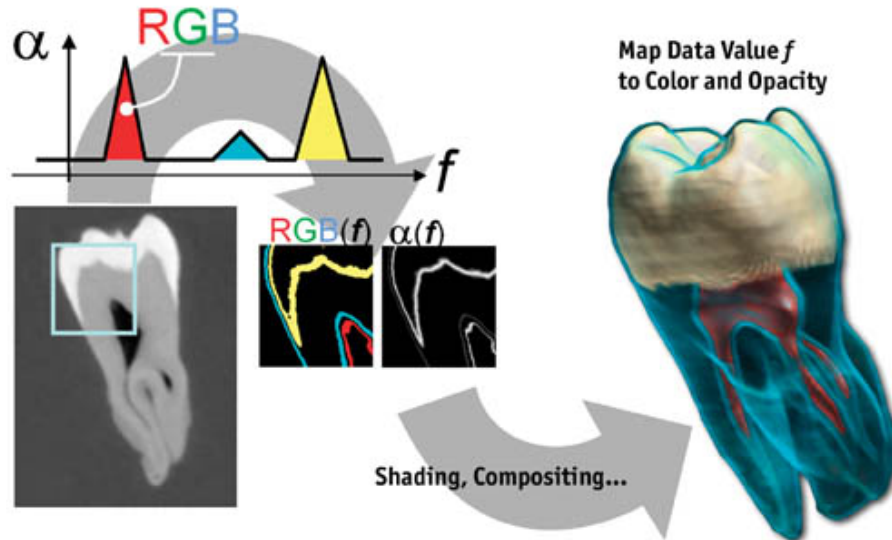


Figura 2.3: Ilustrare a aplicării funcției de transfer pentru evidențierea materialelor diferite dintr-o scanare CT a unui dinte. (Sursa imaginii: GPU Gems, capitolul 39[6])

a unei funcții de transfer, T , este exprimată ca:

$$c = T(i), \quad (2.5)$$

unde i este intensitatea punctului curent. Pe urmă aceste date vor fi folosite în tehnica de compunere alfa back-to-front prezentată mai sus. Un exemplu de aplicare a unei astfel de funcții poate fi regăsit în figura 2.3.

Rata de eșantionare poate fi scăzută în timpul interacțiunii utilizatorului cu volumul, adică în timpul aplicării transformărilor de rotație, translație și scalare, astfel crescând numărul de cadre redade pe secundă pe sistemele cu performanțe mai scăzute, făcând mai ușoară încadrarea în fereastra de redare a regiunilor de interes. Atunci când rata de eșantionare este scăzută, detaliile obiectului vizualizat vor dispărea, dar forma generală a acestuia va rămâne. După ce a fost încadrat obiectul, rata de eșantionare poate fi crescută pentru a obține o imagine mai detaliată, după cum poate fi observat într-un exemplu în figura 2.4.

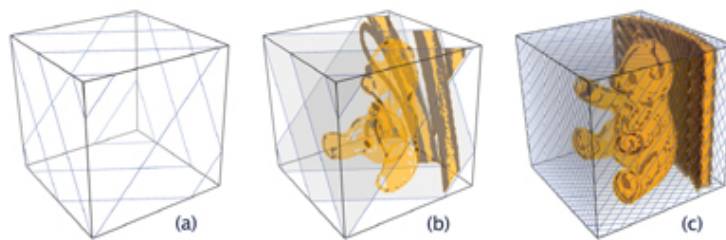


Figura 2.4: Ilustrare a redării volumului folosind rate de eșantionare diferite. Rata de eșantionare crește progresiv în imagini de la stangă la dreapta.

- (a) - rata de eșantionare prea mică pentru a fi observabil obiectul;
- (b) - rata de eșantionare suficient de mare pentru observarea structurii obiectului;
- (c) - rata de eșantionare suficient de mare pentru observarea detaliilor obiectului

(Sursa imaginii: GPU Gems, capitolul 39[6])

Clipping-ul este o tehnică fundamentală de interacțiune pentru a explora datele volumetrice medicale. Această tehnică este folosită pentru a restricționa vizualizarea la sub-volum. În timp ce funcțiile de transfer restricționează vizualizarea la acele părți care au

anumite proprietăți (valori de intensitate, gradienti) în comun, clippingul exclude anumite forme geometrice din vizualizare. Planurile de tăiere sunt translate de utilizator, iar vizualizarea este actualizată continuu. Combinația de șase planuri de tăiere poate fi utilizată pentru a defini un subvolum.

2.2 Lucrări similare

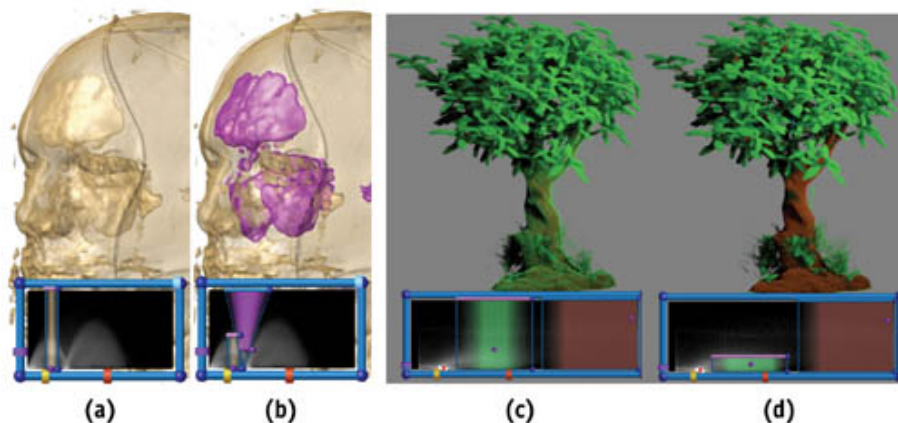


Figura 2.5: Diferență dintre funcții de transfer unidimensionale ((a) și (c)) și multidimensionale ((b) și (d)). (Sursa imaginii: GPU Gems, capitolul 39[6])

Autorii lucrării [4] prezintă posibile modalități de utilizare a funcțiilor de transfer multidimensionale într-o aplicație de vizualizare a datelor medicale volumetrice. Pentru creerea acestui tip de funcții este prezentat un widget ce permite utilizatorului să modifice o funcție de transfer în două dimensiuni: intensitatea voxelilor și magnitudinea gradientului obținut prin diferențe centrale. Autorii acestei lucrări susțin că funcțiile de transfer multidimensionale sunt eficiente pentru vizualizarea granițelor dintre diferite regiuni ale volumului de date analizat și diferite materiale în acesta. De asemenea, acest tip de funcții poate fi folosit pentru izolarea anumitor proprietăți din volum, după cum poate fi observat și în figura 2.5.

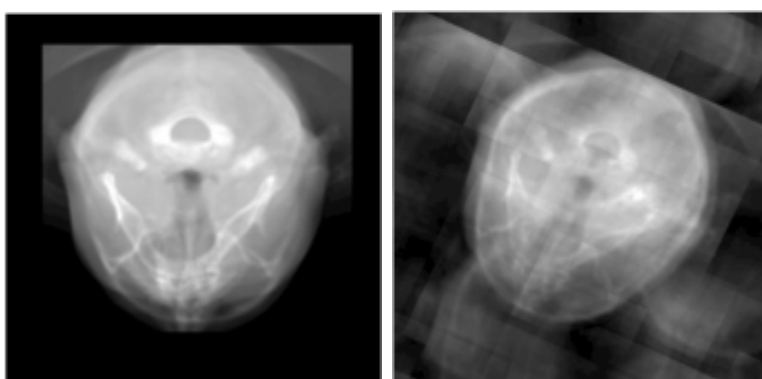


Figura 2.6: Rezultatul redării unei scanări computer tomograf a unui cap folosind tehnica redării prin metoda Fourier. În stânga proiecția este ortogonală, iar în dreapta este rotită, observându-se artefacte. (Sursa imaginii: Abdellah et al.2015 [9])

Redarea volumului prin metoda Fourier (FVR) este o tehnică de vizualizare care a fost utilizată pe scară largă în radiografia digitală. Ca rezultat al complexității în timp $O(N^2 \log N)$, oferă o alternativă mai rapidă la algoritmi de redare a volumului din domeniul spațial care sunt complecși din punct de vedere computațional $O(N^3)$. Introducerea tehnologiei compute unified device architecture (CUDA) permite algoritmilor paraleli să ruleze

eficient pe arhitecturile GPU compatibile CUDA. În [9] este prezentată o implementare accelerată de GPU de înaltă performanță a metodei FVR pe GPU-uri folosind CUDA. Această implementare poate obține o accelerare de 117x în comparație cu o implementare hibridă cu un singur thread care utilizează CPU și GPU împreună [9]. Cu toate că redarea este obținută considerabil mai rapid decât cu redarea directă a volumelor, rezultatul nu este la fel de calitativ, conținând artefacte în unele cazuri, după cum poate fi observat și în figura 2.6.

În [10] sunt investigate rețelele neuronale complet convoluționale (FCN¹) și se propune o arhitectură UNet 3D dedicată procesării imaginilor volumetrice obținute prin tomografie computerizată în scopul segmentării semantice automate. A fost comparată arhitectura propusă în lucrare cu mai multe metode și a fost arătat că arhitecturile bazate pe UNet 3D pot obține performanțe ridicate în sarcinile de segmentare multi-organe folosind un singur GPU. Abordarea propusă a segmentării imaginii 3D nu implică nicio restricție asupra formei anatomiei segmentate. Această omisiune poate duce la zone izolate la marginile organelor. De exemplu, redarea 3D detaliază slab imaginile părților subțiri ale corpului cu puțini voxelii, cum ar fi arterele și venele.

Parametrii stratului convoluțional constau dintr-un set de filtre care pot fi învățate. Fiecare filtru este mic din punct de vedere spațial. În timpul inferenței, fiecare filtru este glisat pe fiecare dimensiune a volumului de intrare și este calculat produsul între intrările filtrului și intrarea în orice poziție în spațiul intrării. Rețeaua va învăța filtre care se activează atunci când identifică un anumit tip de caracteristică vizuală, cum ar fi o margine cu anumită orientare [11].

Pentru rețeaua UNet sunt folosite în codificator straturi convoluționale cu filtre de $3 \times 3 \times 3$ urmate de funcții de activare de tip Rectified Linear Unit (ReLU)[12] și agregarea de tip Max Pooling de $2 \times 2 \times 2$. În mod uzual în rețelele UNet între codificator și decodor este un strat complet conectat (fully connected, FC) numit bottleneck. În arhitectura prezentată de autorii lucrării [10], în loc de un strat complet conectat este folosit un bloc convoluțional. Decodorul conține straturile convoluționale transpuse $3 \times 3 \times 3$ ca straturi finale, fiecare dintre ele utilizând activări ReLU. După stratul convoluțional final, se aplică o funcție de activare sigmoid cu un prag pentru a determina etichetele de segmentare semantică de ieșire [10]. Arhitectura descrisă mai sus este ilustrată în figura 2.7. În arhitectura din figură numărul de filtre pentru fiecare bloc convoluțional este o putere consecutivă a lui 2 începând de la 64, adică: primul bloc conține două straturi convoluționale, fiecare având 64 filtre, în următorul bloc straturile au 128 filtre, continuând până la blocul convoluțional bottleneck care are 1024 filtre.

Autorii lucrării [13] prezintă o arhitectură similară cu UNet 3D, diferența principală fiind faptul că bottleneck-ul și rețeaua convoluțională finală sunt înlocuite de rețele recurente bidirectionale. Autorii propun o metodă prin care segmentarea este efectuată felie după felie din volumul de date, astfel reducându-se cerințele de memorie și durata efectuării retropropagării.

În [14] este propusă o metodă de redare a imaginilor volumetrice folosind rețele neuronale. Metoda propusă, numită DeepDVR, poate învăța să redea imagini volumetrice în care regiunile de interes sunt vizibile și clar delimitate fără să fie necesară creerea unei funcții de transfer, care este o operație costisitoare pentru utilizator. Modelul prezentat a fost antrenat pe un set de date ce conținea predominant imagini cu o calitate scăzută, dar atunci când acesta a fost testat pe date cu calitate înaltă, a generat redări promițătoare. Arhitectura este, din nou, bazată pe UNet 3D, diferența principală fiind faptul că au fost înlocuite conexiunile între blocurile convoluționale și cele convoluționale transpuse cu un modul nou, iar bottleneck-ul a fost eliminat pentru a permite dimensiunea volumului de

¹FCN sau fully convolution network este o rețea neuronală în care singurele operație efectuată sunt cele de convoluție și agregare.

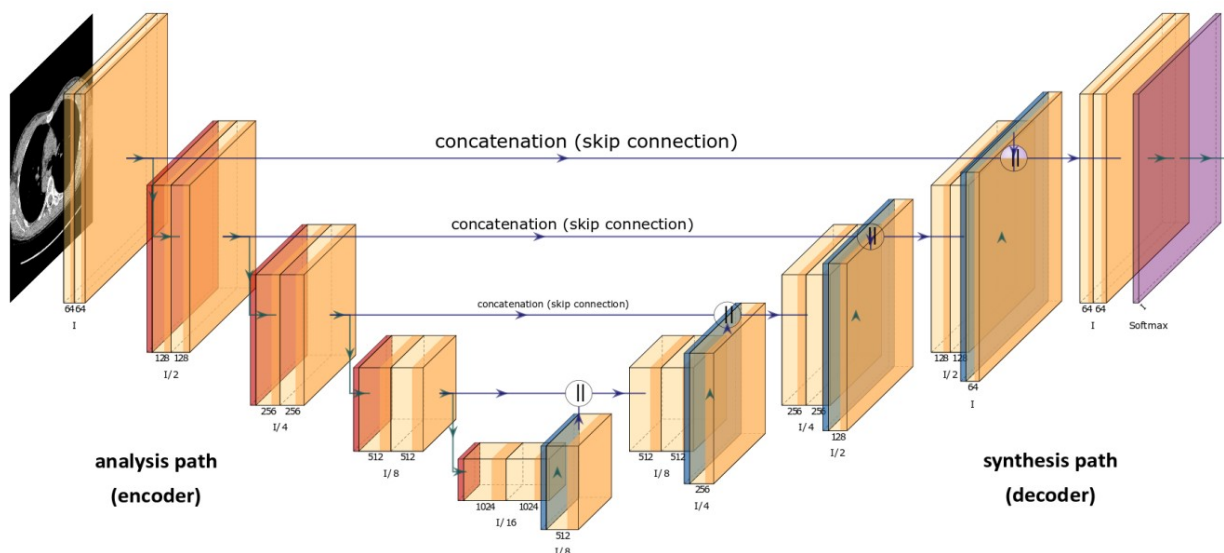


Figura 2.7: Arhitectura U-Net 3D. (Sursa imaginii: Radiuk et al. 2020[10])

intrare să fie arbitrară și dimensiunea ieșirii să fie diferită față de cea de intrare. Este de notat faptul că această rețea primește o intrare tridimensională și ieșirea va fi bidimensională, așadar modulele dintre codificatoare și decodificatoare vor avea un rol important în abstractizarea corectă a informațiilor.

Pentru aplicarea unei rețele neuronale într-o aplicație, dimensiunea acestora trebuie să fie redusă deoarece este de dorit ca rezultatul să fie obținut cât mai rapid. În acest sens sunt explorate rețele neuronale de dimensiuni reduse. O alternativă a reducerii dimensiunii rețelei este învățarea prin transfer [15].

Capitolul 3

Proiectarea aplicației

3.1 Tehnologii folosite

NIfTI¹ este un format pentru stocarea datelor rezultate prin imagistică medicală. Imaginile NIfTI sunt înregistrate într-un sistem local de coordonate. De obicei, fișierele NIfTI au extensia .nii sau .nii.gz. Antetul poate fi stocat într-un fișier separat față de date, în acesta fiind stocat dimensiunea volumului pe fiecare axă de coordonate, dimensiunea fiecărui voxel, și informații necesare pentru construirea matricei de transformare a datelor din spațiul local, în care voxelii sunt echidistanți, într-un spațiu în care redarea arată forma actuală a volumului. În unele cazuri, în antet mai poate fi o descriere.

Există o bibliotecă software denumită niftilib, scrisă în limbajul de programare C, care permite citirea și scrierea imaginilor medicale în format NIfTI. În această bibliotecă sunt definite structuri de date, denumite *nifti_image*, în care sunt stocate informațiile din antetul volumului, și datele acestuia. Tipul de date așteptat de către aplicația de redare la încărcarea unui volum este întreg cu semn pe doi octeți, adică, în C++, *short*. Pentru încărcarea măștii de segmentare este aplicat un pas suplimentar pentru transformarea datelor în valori categorice consecutive pentru fiecare clasă de segmentare.

OpenGL (Open Graphics Library) este o interfață de programare pentru redarea graficelor folosind procesoare video pentru accelerare hardware. GLUT este un set de funcții independente de sistemul de ferestre pentru scrierea programelor OpenGL, ce implementează o interfață simplă de programare a aplicațiilor pentru OpenGL. Deoarece GLUT este un proiect abandonat, a fost creată o alternativă la acesta, care implementează cel puțin aceleași funcționalități de bază. Această bibliotecă este numită freeglut, și a fost folosită în implementarea aplicației.

PyTorch este un framework open-source² de învățare automată bazat pe biblioteca Torch dezvoltat în principal de Meta AI. PyTorch folosește pentru retropropagare o metodă numită diferențiere automată. Operațiunile efectuate sunt înregistrate și apoi sunt folosite pentru a calcula gradientii. Această metodă este utilă atunci când se construiesc rețele neuronale pentru experimentare.

TorchIO este o bibliotecă Python open-source pentru încărcare eficientă, preprocesare, mărire și eșantionare bazată pe petice a imaginilor medicale 3D în învățarea automată, urmând proiectarea PyTorch. Sunt incluse în această bibliotecă clase ce facilitează stocarea, preprocesarea și vizualizarea simplă a seturilor de date volumetrice.

¹Inițiativa pentru Tehnologia Informatică a Neuroimaginii, sau Neuroimaging Informatics Technology Initiative.

²Un software open-source este distribuit sub o licență permisivă, deținătorul drepturilor de autor acorda utilizatorilor dreptul de a utiliza, modifica și distribui codul sursa. De obicei proiectele open-source sunt dezvoltate într-o manieră publică colaborativă.

MONAI este un framework open-source, bazat pe PyTorch, pentru învățarea profundă în imagistica medicală. În acesta sunt incluse arhitecturi de rețele neuronale ce pot fi folosite pentru segmentare semantică și funcții care calculează metrici pe parcursul antrenării, precum matricea de confuzie.

3.2 Setul de date folosit

Setul de date folosit atât pentru antrenarea modelului de segmentare cât și în scop demonstrativ în aplicația de vizualizare a datelor medicale este alcătuit din 140 de tomografii computerizate (CT), fiecare cu cinci organe etichetate: plămân, oase, ficat, rinichi și vezică urinară. Creierul este, de asemenea, etichetat pe minoritatea de scanări care îl arată. Imaginile provin dintr-o mare varietate de surse, inclusiv abdominale și corporale; contrast și non-contrast; tomografii cu doze mici și cu doze mari de substanță de contrast. Setul de date este ideal pentru antrenarea și evaluarea algoritmilor de segmentare a organelor, care ar trebui să funcționeze bine într-o mare varietate de condiții de imagistică. Toate fișierele sunt stocate în format Nifti-1 cu date întregi pe 16 biți și au dimensiuni variabile.

Imaginile sunt stocate ca „volum-XX.nii.gz”, unde XX este numărul cazului. Valorile numerice sunt în unități Hounsfield³. Segmentările sunt stocate ca „labels-XX.nii.gz”, unde XX este același număr cu fișierul ce conține datele volumetrice corespunzătoare.

Conform autorilor setului de date, multe imagini au fost preluate de la provocarea de segmentare a tumorilor ficatului (LiTS) [16].

3.3 Augmentarea vizualizării folosind segmentarea semantică

Funcția de cost utilizată pentru antrenarea acestei rețele este binary cross-entropy (BCE). Această funcție este folosită pentru măsurarea erorii unei reconstrucții într-o rețea de tip auto-encoder unde ieșirile sunt valori de 0 sau 1 [17].

$$l = -w[y \cdot \log x + (1 - y) \cdot \log(1 - x)], \quad (3.1)$$

unde l reprezintă costul, w ponderea funcției cost, x datele de intrare și y rezultatul dorit [17]. Etichetele în masca de segmentare sunt codificate ca numere întregi de la 0 la 6. Putem converti reprezentarea aceasta printr-o codificare one hot, adică convertim valorile categorice ale etichetelor în vectori cu valori de 0 sau 1 care reprezintă eticheta corespunzătoare, precum în tabelul 3.1.

Deoarece imaginile medicale sunt tridimensionale, așadar ocupă spațiu semnificativ în memorie, se poate simplifica problema segmentării cu o rețea neuronală prin împărțirea imaginii în regiuni disjuncte, precum cuburi. Fiecare cub poate fi analizat de rețea, și rezultatul poate fi concatenat pentru a obține o imagine de aceleași dimensiuni cu cea originală. După segmentarea inițială folosind rețeaua neuronală antrenată în PyTorch, rezultatele obținute se pot rafina în continuare extinzând regiunile de interes în spațiile apropiate cu valori similare.

Aplicația pentru antrenarea rețelelor neuronale pentru segmentarea datelor medicale volumetrice a fost realizată pentru facilitarea experimentării cu diferite arhitecturi de rețele neuronale, metode de preprocesare a datelor și hiperparametri. Aceasta funcționează pe baza unor fișiere de configurare de tip YAML⁴ folosind modulul Python OmegaConf ce permite

³Unitatea de măsură Hounsfield cuantifică radiodensitate. Aceasta mai este denumită și valoarea CT deoarece este folosită cel mai des în scanări prin tomografie computerizată.

⁴YAML este un limbaj de serializare a datelor ușor de citit de către oameni.

eticheta	codificare
fundal	0 0 0 0 0 0
ficat	1 0 0 0 0 0
vezică	0 1 0 0 0 0
plămânii	0 0 1 0 0 0
rinichi	0 0 0 1 0 0
os	0 0 0 0 1 0
creier	0 0 0 0 0 1

Figura 3.1: Codificarea *one hot* a etichetelor din setul de date CT Org.

extinderea fișierelor de configurare pentru reutilizarea lor. Sunt folosite trei categorii de fișiere de configurație:

1. pentru setul de date folosit, variabilele stocate în acestea fiind locația în memorie a setului de date, structura acestuia, dimensiunea rezultată în urma preprocesării, dimensiunea peticiilor⁵, numărul de clase de segmentare, dimensiunea batch-ului⁶ etc..
2. pentru hiperparametrii rețelei neuronale. Aceștia pot să difere considerabil între arhitecturi diferite, dar în general sunt stocate valori pentru rata de învățare, numărul de epoci și valori necesare pentru scăderea ratei de învățare pe parcursul antrenării.
3. etapele ce pot fi executate cu acest framework: încărcarea datelor, încărcarea unui model preantrenat, antrenarea unui model, testarea unui model și transformarea rețelei neuronale în TorchScript pentru a putea fi folosit în aplicația de vizualizare.

Aplicația pentru antrenarea rețelelor neuronale pentru segmentarea datelor medicale volumetrice permite executarea antrenării atât local cât și în cloud folosind un serviciu PaaS. În acest sens în lipsa setului de date acesta poate fi descărcat din cloud, fie preprocesat, fie în starea în care a fost preluat de la sursă. Ca și preprocesare a setului de date, fiecare volum din acesta este încărcat și redimensionat la o dimensiune definită în fișierele de configurare, în scopul de a standardiza dimensiunea intrării în rețeaua neuronală. Deoarece nu sunt folosite numai FCN-uri, dimensiunea imaginilor volumetrice trebuie să fie la fel pentru toate imaginile din setul de date. Același pas de redimensionare trebuie aplicat și în aplicația de vizualizare, pentru a putea fi folosit modelul antrenat. Imaginile redimensionate sunt stocate în memoria nevolatilă.

Setul de date este împărțit în 3 subseturi disjuncte, după cum urmează:

1. setul de testare, care este folosit pentru a determina performanțele modelului în urma antrenării (21 scanări CT);
2. setul de validare, care este folosit pentru stabilirea performanțelor modelului în timpul antrenării, și pe baza analizei acestor performanțe este realizată oprirea prematură a antrenării (20% din numărul total de date, exceptând cele de test, adică 28 scanări CT);
3. setul de antrenare, care este folosit pentru antrenarea rețelei neuronale (datele rămase, adică 92 scanări CT);

⁵Peticele, sau patch-uri, sunt paralelipipeduri disjuncte de aceeași dimensiune obținute prin divizia imaginii volumetrice inițiale

⁶numărul de imagini folosite simultan în antrenare

În etapa de antrenare a rețelei neuronale, dacă performanțele rețelei calculate pentru setul de validare nu se îmbunătățesc pentru mai multe epoci consecutive, antrenarea este oprită prematur pentru a evita overfitting-ul⁷.

Pentru antrenarea rețelei a fost folosit un optimizator de tip Adam[18], care este un algoritm pentru optimizarea parametrilor rețelei bazat pe gradientii de ordinul întâi ai funcției obiective stocastice, bazate pe estimări adaptive ale momentelor de ordin inferior.

Rezultatul segmentării automate poate fi augmentat cu un algoritm de *smoothing*, care pentru fiecare punct din masca de segmentare, alege cea mai predominantă clasă din proximitatea acestuia. Proximitatea unui voxel este descrisă în figura 3.2. În urma aplicării acestui algoritm sunt umplute posibile goluri în interiorul regiunilor de interes ale măștii de segmentare, dar pot fi pierdute detaliu pe suprafețele acestor regiuni.

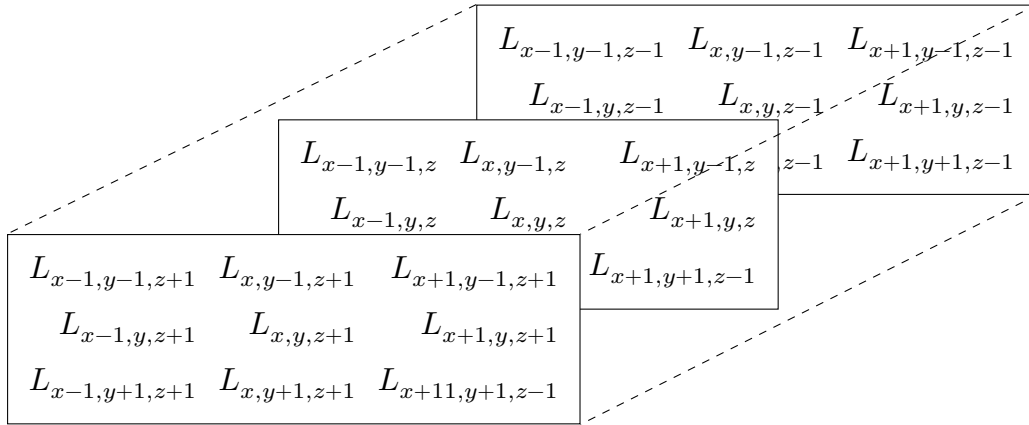


Figura 3.2: Punctele din proximitatea punctului $L_{x,y,z}$ în masca de segmentare.

3.4 Clase dezvoltate

Algorithm: Fluxul aplicației de vizualizare

```

initialize();
TF ← defaultTF;
segmentationMask ← emptyMask;
while app is running do
    renderVolume(volume, TF, segmentationMask);
    renderGUI();
    getUserInput();
    if userInput is loadvolume then
        | volume ← loadVolumeData();
    else if userInput is changeTF then
        | TF ← interpolateTFFromUI();
    else if userInput is computesegmentation then
        | segmentationMask ← generateSegmentationMask();
    end
end

```

⁷Overfitting-ul apare într-o rețea neuronală atunci când aceasta învață trăsăturile specifice setului de date de antrenare și nu are capacitatea să generalizeze pentru date neîntâlnite în etapa de antrenare.

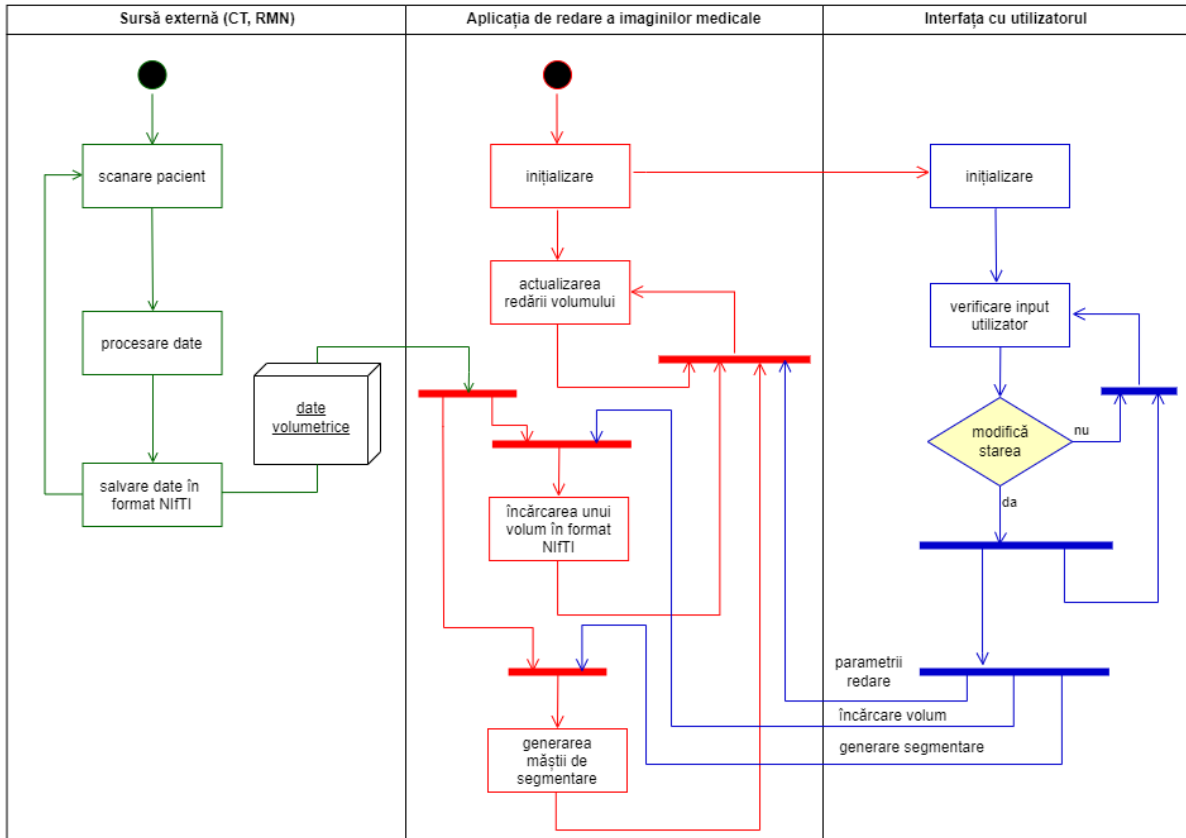


Figura 3.3: Diagrama modului de funcționare a aplicației. În stânga este reprezentat modul extern de achiziție a datelor. În centru sunt reprezentate operațiile de bază efectuate de aplicația de redare. În dreapta sunt reprezentate operațiile posibile în interfața cu utilizatorul.

În figura 3.3 sunt prezentate operațiile efectuate în timpul utilizării aplicației. Aceste operații sunt prezentate și sub formă de pseudocod în algoritmul de mai sus. În continuare vor fi prezentate clasele implementate pentru realizarea acestei modalități de funcționare.

În anexa 1 se găsește punctul de intrare în aplicație, adică funcția `main`. În același fișier se mai regăsesc și funcții de inițializare și logica principală a redării, la cel mai înalt nivel.

În etapa de inițializare, înainte de crearea ferestrei aplicației, este configurat modul de afișare din glut pentru buffer dublu și RGBA și buffer pentru adâncime. Se creează contextul în care trebuie să funcționeze ImGui și sunt apelate funcțiile pentru inițializarea acestuia. Este încărcat modelul de segmentare din LibTorch și sunt stabilite funcțiile care vor fi folosite de către clasa *Interface* pentru obținerea și manipularea datelor din nivelul aplicației. După care, sunt inițializate matricile *projection* și *view* ce vor fi folosite pentru calcularea matricei de transformare *MVP*.

Etapa de redare începe prin curățarea imaginii anterioare de pe ecran și a buffer-ului de culoare și a celui de adâncime. După care este calculată matricea de transformare MVP și încărcată în obiectul shader. De asemenea sunt încărcate și limitele și vectorul de translație pentru AABB. Dacă funcția de transfer a fost schimbată din interfață, aceasta este reîncărcată în obiectul shader. Același fapt este valabil și pentru culorile obiectelor rezultate din segmentarea semantică.

Etapa de afișare a imaginii este încapsulată de apeluri ale unei funcții de obținere a timpului curent în scopul de a calcula timpul necesar afișării unui frame și pentru a limita aplicația la 30 FPS (frame-uri pe secundă) pentru conservarea resurselor.

Clasa *Volume* are rolul de încărcare a datelor volumetrice într-o textură tridimen-

sională, și de asemenea de a genera texturile necesare pentru funcția de transfer și masca de segmentare. Codul sursă pentru această clasă este în anexele 2 și 3.

Citirea datelor volumetrice din memorie este realizată în clasa *Loader*, codul sursă al acesteia fiind în anexele 12 și 13. Încărcarea datelor volumetrice este realizată în primul rând de către biblioteca *nifti*. În urma apelului funcției *nifti_image_read* este obținut un bloc continuu de date și informațiile stocate în header-ul fișierului. Tipul de date folosit pentru stocarea volumelor din setul de date CT-ORG este *short*. După ce sunt obținute datele de la clasa *Loader*, clasa *Volume* generează texturile necesare pentru datele volumetrice și segmentarea semantică, cea din urma fiind populată ulterior. Parametrii pentru ambele texturi sunt identici, și anume: interpolare liniară a valorilor, și repetarea valorilor de pe margini în afara marginilor.

Logicile pentru generarea măștii de segmentare și pentru netezirea acesteia sunt încapsulate în clase *thread* pentru a fi executate în fire de execuție separate de cel principal, deoarece acestea sunt operații complexe cu o durată lungă de timp, iar blocarea interfeței cu utilizatorul și a logicii de redare nu este de dorit. Bineînțeles că în timp ce sunt efectuate aceste operații nu îi mai este permis utilizatorului să înceapă un nou calcul al segmentării sau o nouă netezire a acesteia până la finalizarea celui în curs.

Clasa *Shader* are rolul de a genera obiectele shader pentru shaderul vertex și cel fragment, și expune o funcție pentru încărcarea logicii acestora din memoria nevolatilă. În această clasă sunt două funcții pentru facilitarea dezvoltării aplicației, anume, pentru detectarea erorilor la compilarea și link-editarea shaderului. Mai sunt expuse și funcții wrapper la cele standard OpenGL, pentru a face mai ușoară încărcarea datelor în obiectele shader. Codul sursă pentru această clasă este în anexele 4 și 5.

În anexele 6 și 7 sunt definiția și implementarea clasei *Interface*, care este clasa principal responsabilă pentru interfața cu utilizatorul. Această clasă stochează functori care vor fi apelați pentru obținerea și manipularea datelor din aplicație. În acest fel sunt separate responsabilitățile interfeței utilizator și redarea și încărcarea datelor volumetrice. Interfața în sine este prezentată în capitolul 4. Widget-ul folosit pentru modificarea funcției de transfer este *VectorColorPicker*, a cărui cod sursă poate fi găsit în anexele 8 și 9. În această clasă sunt stocate funcționalități pentru manipularea graficului funcei de transfer, afișarea histogramei intensității volumului vizualizat curent și încărcarea și salvarea funcției de transfer din memorie.

Încărcarea modelului de segmentare și funcția pentru folosirea acestuia în scopul generării măștii de segmentare sunt în clasa *PytorchModel* în anexele 14 și 15. Funcția *forward* implementează următorul algoritm:

1. Datele stocate într-un *array* de tip *short* sunt încărcate într-un obiect Torch *Tensor* folosind funcția *from_blob*.
2. Imaginea volumetrică este interpolată, dimensiunea intrării transformandu-se din $[W, H, D]$ ⁸ în $[W_I, H_I, D_I]$ ⁹.
3. În funcție de valorile W_P, H_P, D_P ¹⁰, imaginea este împărțită în petice disjuncte folosind funcția *split*, acestea fiind pe urmă concatenate pe dimensiunea *batch* (*B*).

⁸ W = lățimea volumului, H = înalțimea volumului, D = adâncimea volumului

⁹ W_I, H_I, D_I reprezintă dimensiunea la care au fost redimensionate datele în timpul antrenării rețelei neuronale

¹⁰ W_P, H_P, D_P reprezintă dimensiunea peticelor ce au fost folosite la antrenarea modelului, dacă acestea au fost folosite. În mod uzual au fost folosite petice astfel încât imaginea redimensionată să poată fi împărțită în noua petice disjuncte de aceeași dimensiune (trei pentru fiecare dimensiune)

4. Este obținut rezultatul rețelei neuronale după apelarea funcției *forward* al acesteia, fiind dat că parametru tensor-ul ce conține peticele concatenate. Astfel intrarea rețelei neuronale are dimensiunea $[B, 1, W_P, H_P, D_P]$, unde 1 reprezintă numărul de canale, în cazul asta fiind doar luminanță. Ieșirea rețelei neuronale are dimensiunea $[B, L, W_P, H_P, D_P]$, unde L reprezintă numărul de clase de segmentare.
5. În cazul în care au fost folosite petice, imaginea este recompusă din acestea, rezultând o ieșire de dimensiune $[L, W_P, H_P, D_P]$.
6. Rezultatul obținut până în acest moment este interpolat pentru a obține o imagine care are dimensiunea celei inițiale, adică $[L, W, H, D]$.
7. Este aplicată funcția sigmoid pentru rezultatul obținut anterior, folosind un prag de 0.5. Valorile mai mari de acest prag semnifică faptul că voxelul $S_{i,x,y,z}$ aparține clasei de segmentare L_i . Astfel sunt obținute L măști de segmentare, câte una pentru fiecare clasă de segmentare.
8. Măștile de segmentare obținute anterior sunt agregate într-una singură, fiecare element din aceasta putând avea valori de la 0 la $L + 1$. Acest rezultat este transformat într-un *array* de *unsignedchar* folosind funcțiile *contiguous* și *data_ptr*. Acesta este rezultatul final returnat de funcția *forward* a clasei *PytorchModel*.

Intrarea programului este *run.py*, acesta fiind în anexa [18]. Se realizează în serie acțiunile marcate ca fiind necesare în configurarea sistemului. Rezultatul fiecărei acțiuni este salvat într-o tabelă de dispersie, și această tabelă este intrare pentru următoarea acțiune, astfel fiind satisfăcute dependențele. Acțiunile posibile implementate în Python sunt enumerate în fișierul *jobs.py* în anexa [19], iar fișierele de configurare YAML pentru acestea se află în directorul *config/jobs* în codul sursă al proiectului și în anexele [37, 38, 39, 40, 41].

Încărcarea setului de date este realizată într-o clasă ce extinde clasa abstractă *Dataset* din anexa [23], în funcție de fiecare set de date cu care vor fi făcute experimente. Din fișierul de configurare din anexa [33] sunt obținute locațiile imaginilor volumetrice și a măștilor de segmentare realizate manual. Pe baza acestora se crează o tabelă de dispersie în care cheia primară reprezintă locația unei imagini și valoarea reprezintă locația măștii de segmentare. Folosind această tabelă de dispersie sunt încărcate datele în instanțe ale clasei *Subject* din TorchIO. Funcția *get_data_loader* din fișierul *loading.py* în anexa [27] împarte setul de date în seturi disjuncte pentru antrenare, validare și testare și întoarce în tabela dependențelor câte un *DataLoader* pentru fiecare astfel de set.

Antrenarea rețelei neuronale este realizată în funcția *train* din fișierul *train.py* din anexa [25]. Valorile funcției cost și ale funcției F1 sunt stocate cu ajutorul unui *SummaryWriter* pentru a putea fi vizualizate ulterior în Tensorboard. Aceste metrice sunt și afișate la fiecare *metrics_every* epoci. În mod asemănător, sunt calculate metrice pentru rețeaua neuronală pe setul de date de validare la fiecare *validate_every* epoci. Modelul rezultat poate fi transformat în TorchScript cu funcția *deploy_model* din anexa [28]. Modulul rezultat este *torchscript_module_model.pt*, iar în instanță în care rezultatele antrenării sunt satisfăcătoare, poate fi încărcat în aplicația de redare.

Capitolul 4

Implementarea aplicației

4.1 Aplicația de redare

Obiectele shader reprezintă codul GLSL compilat pentru o singură etapă shader. Acestea pot fi create folosind funcția *glCreateShader* ce primește ca parametru tipul de shader ce trebuie creat printr-un enum, cele mai comune valori pentru acesta fiind *GL_VERTEX_SHADER* și *GL_FRAGMENT_SHADER* [19]. Prin această valoare este specificată pentru ce etapa este stocat codul din shader. Odată creat obiectul shader în acesta trebuie stocat codul sursă GLSL. Acest lucru poate fi făcut cu funcția *glShaderSource*, care primește ca parametru o listă de șiruri de caractere [19]. Când shader-ul este compilat, acesta va fi compilat ca și cum toate șirurile date ar fi concatenate. Odată ce șirurile de caractere ce reprezintă codul sursă pentru un shader au fost stocate într-un obiect shader, acesta poate fi compilat cu funcția *glCompileShader*. Compilarea shaderului poate eșua, iar pentru aceasta trebuie verificată valoarea *GL_COMPILE_STATUS* din obiectul shader [19]. În cazul în care aceasta indică faptul că a existat o eroare în procesul de compilare, eroarea poate fi identificată cu ajutorul funcției *glGetShaderInfoLog*. Atunci când au fost create toate obiectele shader necesare acestea trebuie link-editate într-un program. În această etapă pot exista erori, și acestea pot fi tratate în același mod că în etapa de compilare, singura diferență fiind verificarea valorii *GL_LINK_STATUS* în loc de *GL_COMPILE_STATUS* [19].

Shader vertex este etapa Shader programabilă din pipeline-ul de redare care se ocupă de procesarea nodurilor individuale. Un vertex shader primește un singur vârf din fluxul de vârf și generează un singur vârf în fluxul de vârf de ieșire. Trebuie să existe o corespondență 1:1 de la vârfurile de intrare la vârfurile de ieșire. În această aplicație acest shader este folosit pentru a transforma pozițiile vârfurilor din spațiu local în spațiul camerei. Shaderul vertex folosit este cel din anexa 16.

Matricea modelului M este compusă din transformarea de translație T a unui obiect, transformarea de rotație R și transformarea de scalare S . Înmulțirea poziției unui vertex v cu această matrice model transformă vectorul în spațiul global [20].

$$M = T \cdot R \cdot S \quad (4.1)$$

$$v_{world} = M \cdot v_{model} \quad (4.2)$$

Camera are, de asemenea, o matrice model care definește poziția sa în spațiul global. Inversul matricei modelului camerei V este matricea de vizualizare și transformă vârfurile din spațiul global în spațiul camerei sau spațiul de vizualizare. Odată ce vârfurile sunt în spațiul camerei, ele pot fi transformate în spațiul clip prin aplicarea unei transformări de proiecție. Matricea de proiecție P codifică cât de mult din scenă este surprinsă într-o redare prin definirea marginilor vizualizării camerei. Cele mai comune două tipuri de proiecție sunt

perspectivă și ortografică. Proiecția în perspectivă are ca rezultat efectul natural al lucrurilor care par mai mici cu cât sunt mai departe de privitor [20].

Poziția finală a vârfurilor este obținută după ecuația:

$$v_{final} = P \cdot V \cdot M \cdot v \quad (4.3)$$

Un shader fragment este etapa shader care va procesa un fragment generat de rasterizare într-un set de culori și o singură valoare de adâncime. Shaderul de fragmente este etapa pipeline-ului OpenGL după ce o primitivă este rasterizată. Pentru fiecare eșantion de pixeli acoperiți de o primitivă, este generat un fragment, fiecare dintre acestea are o poziție în spațiul ferestrei de vizualizare și conține toate valorile de ieșire interpolate pentru fiecare vârf din ultima etapă de procesare vertex. Ieșirea unui fragment shader este o valoare de adâncime și mai multe valori de culoare care pot fi scrise în buffer-urile din framebuffer-urile curente. Fragment shaders iau un singur fragment ca intrare și produc un singur fragment ca ieșire [20].

Shaderul de fragmente este folosit pentru a aplica tehnica ray casting specifică afișării datelor volumetrice. Acest algoritm este compus din 4 etape pentru fiecare pixel:

1. Este calculată intersecția volumului cu o rază ce are originea în pixel și este perpendiculară pe suprafața ecranului.
2. Este realizată eșantionarea volumului în puncte echidistante de-a lungul razei ce se află în interiorul casetei de încadrare.
3. Se obține valoarea funcției de transfer pe baza valorii intensității în punct.
4. Toate proprietățile optice rezultate sunt acumulate rezultând culoarea și opacitatea finală a pixelului.

Punctele de origine ale razelor sunt invariante la rotația și scalarea obiectului, dar acestea trebuie modificate atunci când sunt efectuate operații de translație asupra acestuia. Shaderul de fragmente utilizat pentru ray casting este în anexa 17 Opacitatea finală a fiecărui fragment depinde de rata de eșantionare. Aceasta poate fi corectată folosind formula:

$$A = 1 - (1 - A)^{\frac{s_0}{s}}, \quad (4.4)$$

unde A este valoarea opacității punctului curent, s este rata de eșantionare folosită și s_0 este o rată de eșantionare de referință. Aceasta este utilizată pentru corectarea opacității funcției de transfer ori de câte ori utilizatorul modifică rata de eșantionare s de la rata de eșantionare de referință s_0 [6].

În figura 4.1 este prezentat procesul de creare a unei redări semnificative a unui volum. Etapele ce sunt încadrate în dreptunghiuri cu linie punctată sunt opționale, sau pot fi realizate în altă ordine decât cea prezentată. În partea din dreapta este prezentat rezultatul aplicării măștii de segmentare peste rezultatul obținut în imaginea din stânga pentru aceeași etapă. Segmentarea poate fi generată sau încărcată din memorie în cazul în care a mai fost generată, sau există o segmentare manuală disponibilă. În momentul încărcării unui alt volum masca de segmentare este reinițializată, dar funcțiile de transfer și valorile parametrilor pentru ray casting rămân stocate în aplicație. Modalitatea de alterare a funcției de transfer și parametrii ce pot fi ajustați pentru vizualizare sunt prezentate în secțiunea următoare.

În figura 4.2 se regăsește o captură de ecran a aplicației atunci când a fost încărcat un volum de înaltă calitate și a fost aleasă o funcție de transfer care să evidențieze structura osoasă a pacientului. În acest, folosind widget-ul pentru crearea funcției de transfer, a fost

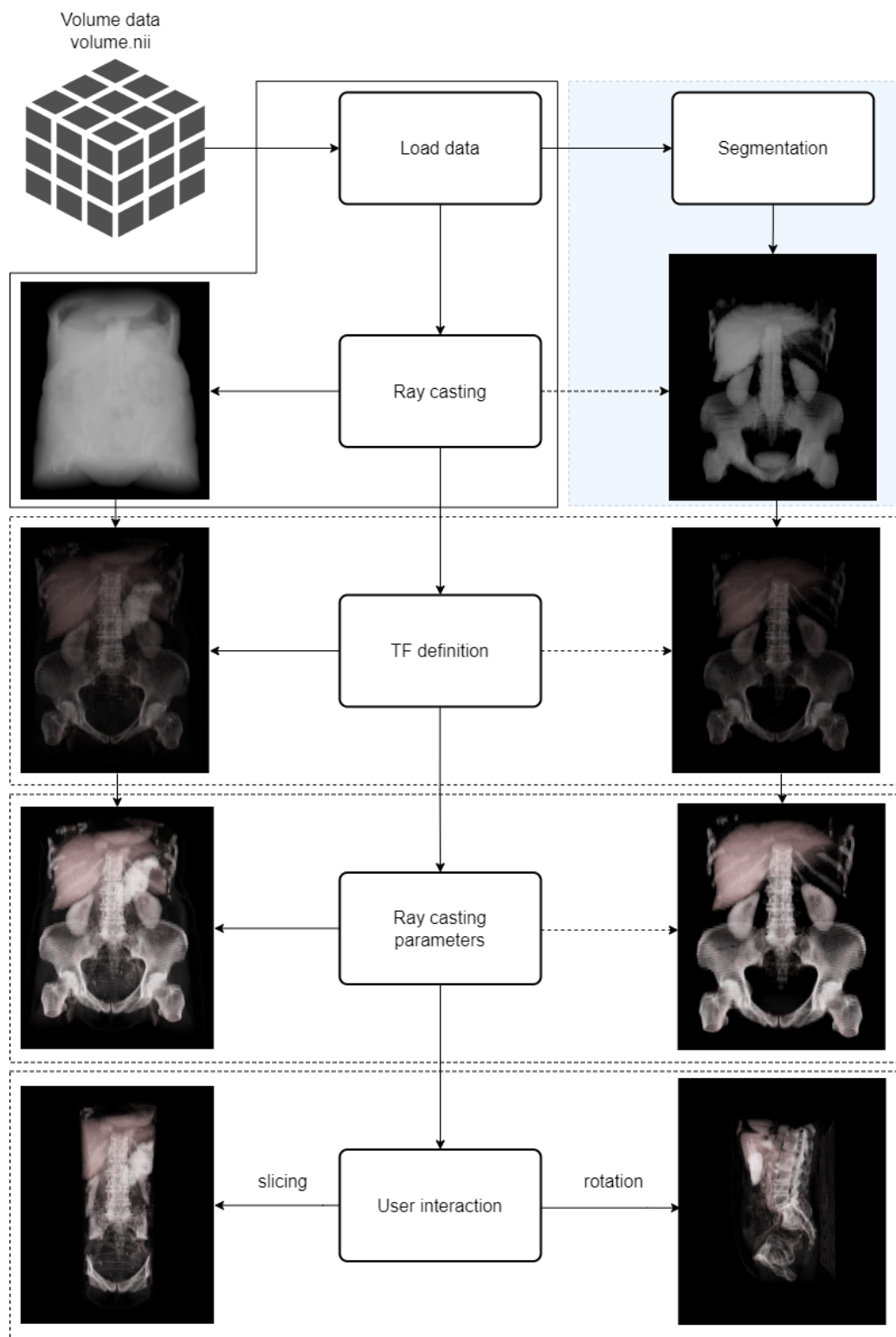


Figura 4.1: Diagrama etapelor procesului de redare a unei imagini volumetrice. După fiecare etapă redarea poate fi îmbunătățită de către utilizator prin definirea unei funcții de transfer, optimizarea parametrilor folosiți de ray casting sau prin generarea și aplicarea măștii de segmentare.

creată o formă trapezoidală, astfel funcția atribuie voxelilor cu intensitate scăzută (reprezentând țesuturi moi precum organe, piele, mușchi etc.) opacitate 0. Pentru intensitățile mai mari decât maximum existent în volumul reprezentant nu este importantă forma funcției de transfer, deoarece nu afectează redarea. În figura 4.3 se regăsește, de asemenea, o captură de ecran a aplicației în care a fost folosită o mască de segmentare pentru îmbunătățirea vizuali-

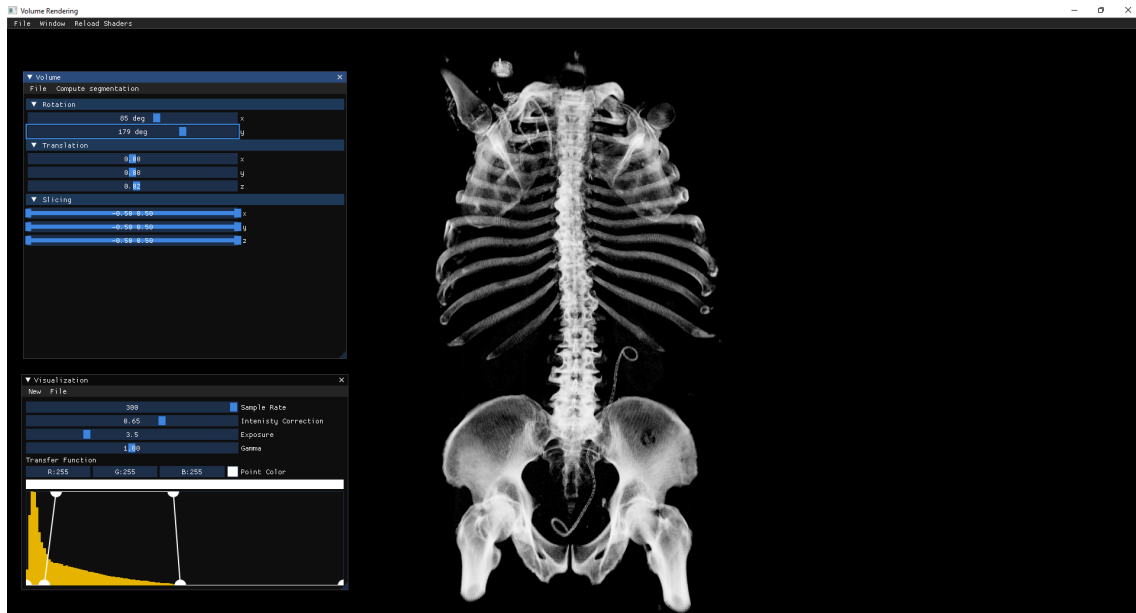


Figura 4.2: Exemplu de vizualizare a unei imagini volumetrice în aplicația C++. A fost creată o funcție de transfer astfel încât să fie evidențiată structura osoasă a pacientului.

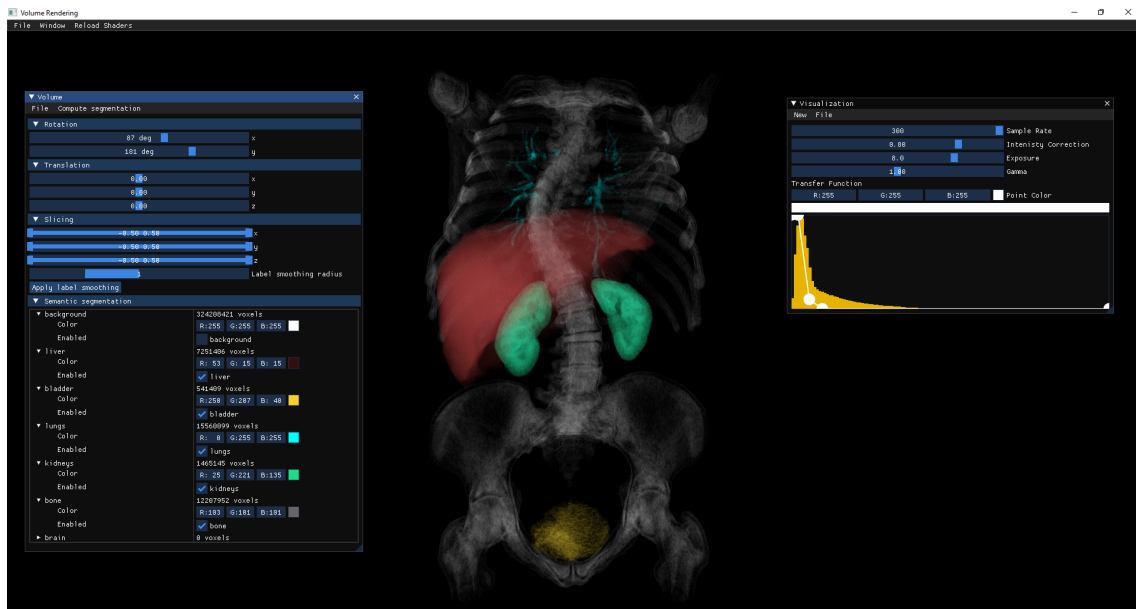


Figura 4.3: Îmbunătățirea vizualizării folosind segmentarea semantică. Fiecare tip de regiune din masca de segmentare are atribuită o culoare diferită pentru evidențiere.

zării. Fiecare etichetă are atribuită o culoare diferită pentru evidențierea diferitelor clase de segmentare.

4.2 Interfața cu utilizatorul

Dear ImGui este o bibliotecă C++ pentru interfață grafică cu utilizatorul minimalistă, fără dependențe externe. Este necesar un backend pentru a integra Dear ImGui în aplicație care transmite intrări de la periferice și este responsabil de redarea buffer-urilor rezultate. Sunt furnizate backend-uri pentru o varietate de API-uri grafice și platforme de redare, de interes fiind în special cel pentru GLUT. ImGui crează interfețe bazate pe ferestre ce pot fi redimensionate și plasate oriunde în fereastra aplicației. Poziția și dimensiunile fere-

trelor sunt stocate într-un fișier de configurare, care în mod implicit este denumit "imgui.ini". În acest fișier vor fi stocate, folosind funcții noi după modelul celor existente pentru fereștre, culori pentru elementele din interfață, marcaje pentru browser-ul de fișiere și parametrii pentru modelul de segmentare.

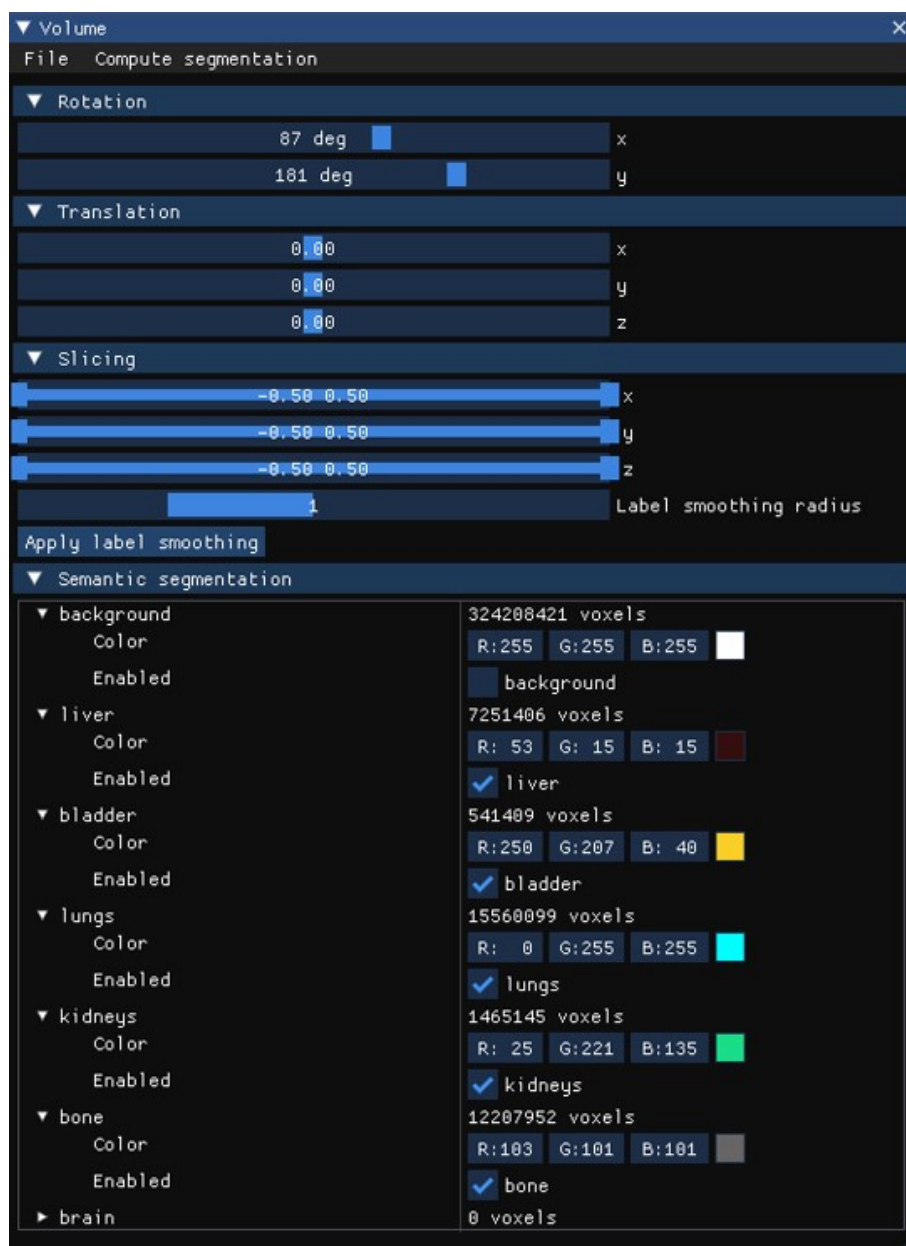


Figura 4.4: Fereastra din care pot fi manipulate perspectiva vizualizării volumului și regiunile vizibile cu ajutorul măștii de segmentare.

În figura 4.4 este o captură de ecran a ferestrei "Volume" din aplicația de vizualizare. Această fereastră este împărțită în patru regiuni ce pot fi restrânse:

1. "Rotation" - în această porțiune a ferestrei sunt două slidere care arată și manipulează unghiul din care este vizualizat volumul, pe două axe. Vizualizarea din diferite unghiuri este importantă deoarece este prezentat un obiect tridimensional într-o imagine 2D, astfel sunt pierdute multe detalii într-o singură redare, fiind necesare multiple redări din diferite unghiuri pentru redarea întregului volum.
2. "Translation" - sunt prezente trei slidere care manipulează și prezintă locația în spațiu a volumului pe cele trei axe. Acestea sunt utile atunci când este aplicat zoom asupra

volumului, astfel fiind vizibilă doar regiunea centrală. În acest caz, pentru vizualizarea cu zoom a regiunilor mai apropiate de marginile volumului, este necesară aplicarea unei translări spațiale a cubului în care se realizează redarea. O altă utilitate a operației de translare este aceea de a permite amplasarea ferestrelor interfeței în fereastra aplicației și mutarea cubului volumului astfel încât acesta să fie vizibil.

3. "Semantic segmentation" - în această regiune a ferestrei este împărțită într-un tabel cu șapte rânduri, unul pentru fiecare clasă de segmentare și doua coloane, în stânga fiind eticheta clasei de segmentare și în dreapta numărul de voxeli care aparțin acelei etichete. Fiecare rând poate de descoperit pentru a afișa încă două rânduri în care se regăsește un selector de culoare pentru clasa respectivă, și un *toggle* care controlează dacă regiunile marcate cu eticheta respectivă sunt sau nu redade.

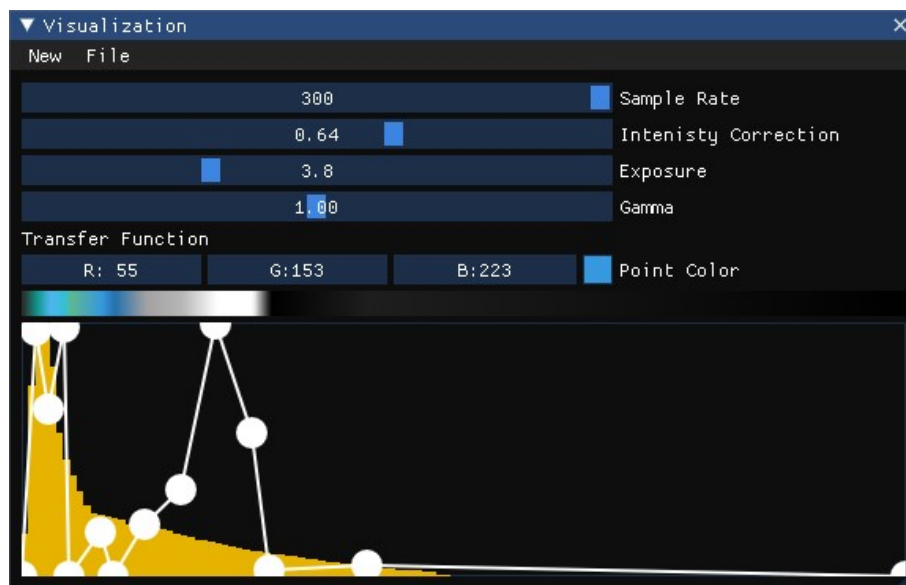


Figura 4.5: Fereastra din care pot fi modificate funcția de transfer și valorile specifice vizualizării.

În figura 4.5 este o captură de ecran a ferestrei în care pot fi schimbate aspecte legate de vizualizarea modelului. Întâi sunt afișate patru slidere care modifică valori ce afectează rezultatul redării.

1. "Sample Rate" reprezintă rata de eșantionare, adică în câte puncte echidistante sunt preluate intensități din volum pentru compunerea lor pentru obținerea rezultatului redării.
2. "Intensity Correction" este valoarea cu care este înmulțită intensitatea la fiecare pas din etapa de compunere.
3. "Exposure" este folosit pentru modificarea intensității rezultatului obținut după etapa de compunere.
4. "Gamma" este valoarea folosită pentru corecția gamma.

În partea de jos a acestei ferestre este un widget care permite creerea unei funcții de transfer liniare prin amplasarea punctelor într-un grafic. Distanța punctelor față de axa Ox reprezintă valoarea transparenței. Punctele pot fi adăugate, șterse și repositionate cu ajutorul mouse-ului. Atunci când este selectat un punct, acestuia îi poate fi atribuită o culoare. Rezultatul poate fi previzualizat în partea de sus a graficului. Funcția de transfer care este trimisă către obiectul shader este obținută prin interpolare liniară pentru puncte și culorile

lor atribuite. Graficul funcției de transfer este peste o histogramă a intensităților volumului curent, pentru a ajuta utilizatorul în creerea unei funcții de transfer relevante.

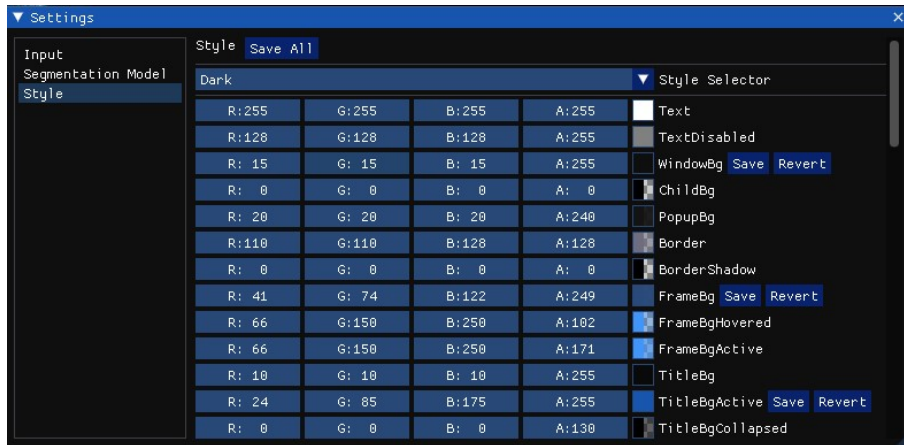


Figura 4.6: Fereastra în care pot fi schimbate setările aplicației.

ImGui facilitează modificarea culorilor din interfața cu utilizatorul, așadar în fereastra "Settings" prezentată în figura 4.6 există un tab în care sunt înșirate toate culorile ce pot fi schimbate, iar schimbările care sunt făcute pot fi salvate. Pe lângă acesta, mai există două tab-uri: unul în care pot fi schimbate vitezele cu care se rotește sau translează volumul, și celălalt conține parametrii necesari pentru funcționarea modelului de segmentare semantică (dimensiunea la care va fi redimensionat volumul înainte ca acesta să fie dat ca intrare la rețeaua neuronală).

Capitolul 5

Testarea aplicației și rezultate experimentale

5.1 Elemente de configurare

Pentru compilarea și link-editarea aplicației a fost folosit Visual Studio 2022. Standardul C++ folosit este ISO C++ 17. Trebuie folosite anumite definiții preprocesor pentru compilarea aplicației. Mai jos sunt enumerate definițiile preprocesor folosite și motivul pentru care acestea au fost folosite:

- `HAVE_ZLIB`: Încărcarea datelor volumetrice de tip NIfTI arhivate (.nii.gz);
- `USE_BOOKMARK`: Compilarea cu succes a bibliotecii externe `ImGuiFileDialog`. Această opțiune permite salvarea locațiilor favorite în memorie pentru încărcarea ușoară a datelor volumetrice;
- `_SILENCE_ALL_CXX17_DEPRECATION_WARNINGS`: Sunt folosite funcții depreciate în standardul C++ 17 în biblioteci folosite;
- `_CRT_NONSTDC_NO_DEPRECATED`: Compilarea bibliotecii `gzlib`. Funcțiile `open`, `read`, `write` și `close` sunt marcate ca fiind depreciate;
- `_CRT_SECURE_NO_DEPRECATED`: Biblioteca `ImGuiFileDialog` folosește funcții nesigure precum `scanf`.

Trebuie incluse următoarele biblioteci în directorul "extern": `niftilib`, `zlib`, `znzlib`, `freeglut`, `ImGui`, `ImGuiFileDialog` și `libtorch` (versiunea pentru CPU). De asemenea sunt necesare `dll`-ul și `lib`-ul pentru `freeglut` amplasate în directorul principal cu fișierul proiect.

Utilizarea interfeței utilizator este prezentată în capitolul anterior. Parametrii transformărilor de rotație, translație și tăiere pot fi manipulați folosind și `mouse`-ul și tastatura. Cu mișcări ale `mouse`-ului pe axa `Ox` va fi rotit volumul pe axa `Ox`, similar pentru axa `Oy`. Prin mișcarea roțiței `mouse`-ului se modifică scara la care este vizualizat volumul. Rotația poate fi efectuată și cu apăsarea tastelor 'w', 'a', 's' și 'd' sau a săgeților de pe tastatură. Pentru aplicarea translației vor fi folosite aceleași taste ca pentru rotație atunci când este apăsată concomitent tasta `ALT`.

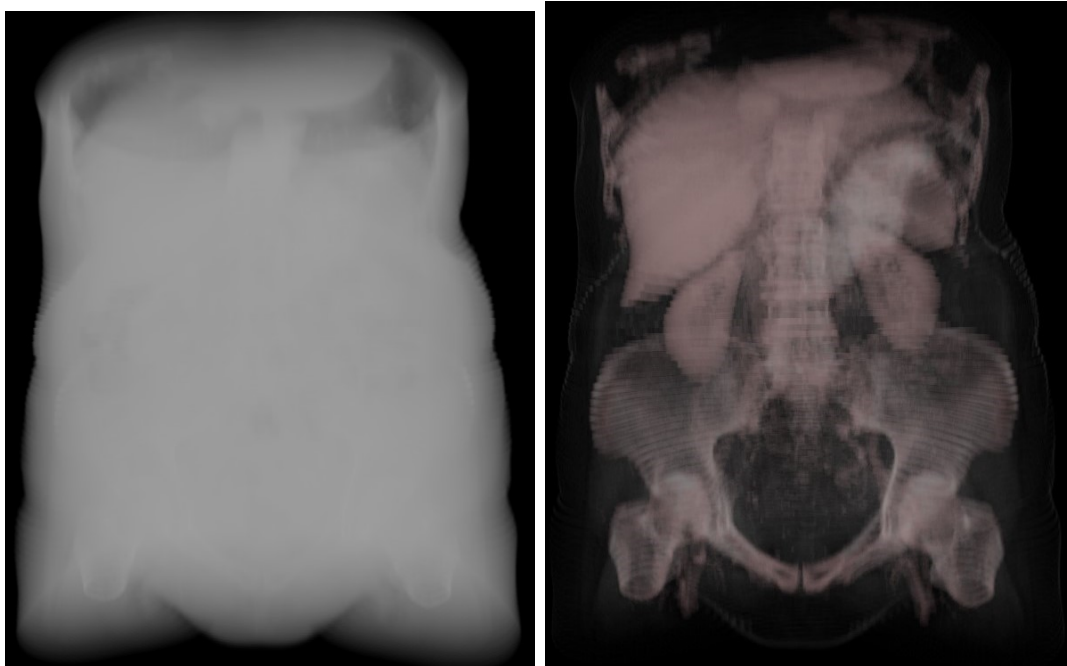


Figura 5.1: Comparație între redarea unui volum fără a fi folosită o funcție de transfer (stangă) și cu o funcție de transfer (dreapta).

5.2 Rezultate obținute în aplicația de redare

În figura 5.1 se poate observa importanța funcției de transfer în vizualizarea volumelor. Pentru a putea reda o imagine semnificativă utilizatorului trebuie furnizată sau creată o funcție de transfer care să filtreze detaliile nesemnificative și să faciliteze vizualizarea regiunilor de interes. În figura 5.1 a fost folosită o funcție simplă, dar pentru a obține rezultate mai bune ar putea fi explorate metode de ghidare a utilizatorului în căutarea unei funcții de transfer optime[5].

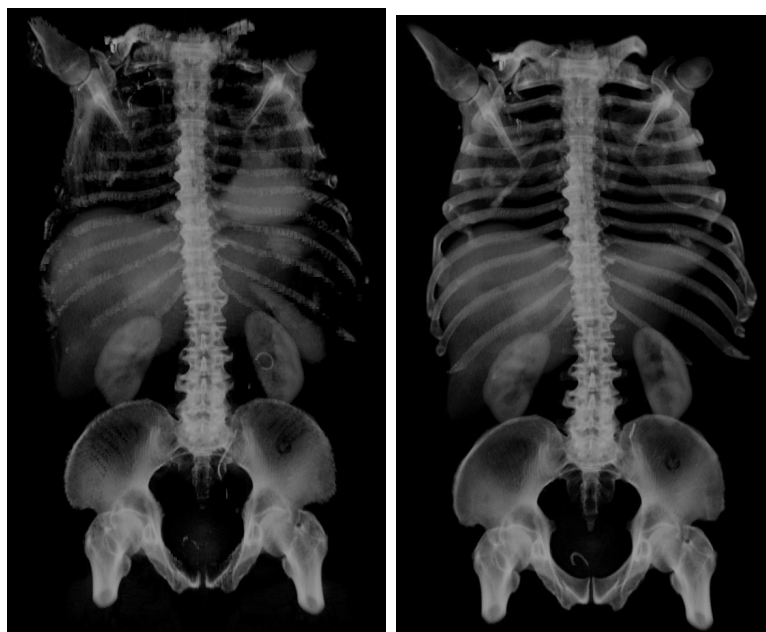


Figura 5.2: Comparație dintre segmentarea obținută cu ajutorul modelului de segmentare (stânga) și cea manuală din setul de date (dreapta) pentru aceeași imagine volumetrică. Redarea a fost realizată utilizând aplicația de redare fără aplicarea unei funcții de transfer și folosind aceași parametri de vizualizare.

În figura 5.2 este prezentat în partea stângă rezultatul aplicării unei măști de segmentare generate automat de către aplicație. Pentru obținerea unei astfel de măști sunt necesare resurse suplimentare precum mult mai multă memorie RAM (aproximativ de patru ori mai multă memorie decât este necesar doar pentru vizualizare) și timp de procesare extins (în funcție de dimensiunea volumului de date poate dura și câteva minute pentru obținerea măștii de segmentare).

5.3 Rezultate experimentale obținute în antrenarea modelului de segmentare semantică

Pot fi folosite anumite metrice pentru a studia eficacitatea unui model. Pentru clasificare poate fi calculată acuratețea, adică procentajul punctelor etichetate corect. Dar, această măsură nu este potrivită în contextul segmentării imaginilor volumetrice, deoarece o mare parte dintre punctele din volum nu sunt etichetate. O soluție ar fi folosirea măsurii F care este calculată astfel:

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)}, \quad (5.1)$$

unde

tp *true positives* reprezintă numărul de puncte etichetate corect.

fp *false positives* reprezintă numărul de puncte etichetate în mod eronat.

fn *false negatives* reprezintă numărul de puncte neetichetate în mod eronat.

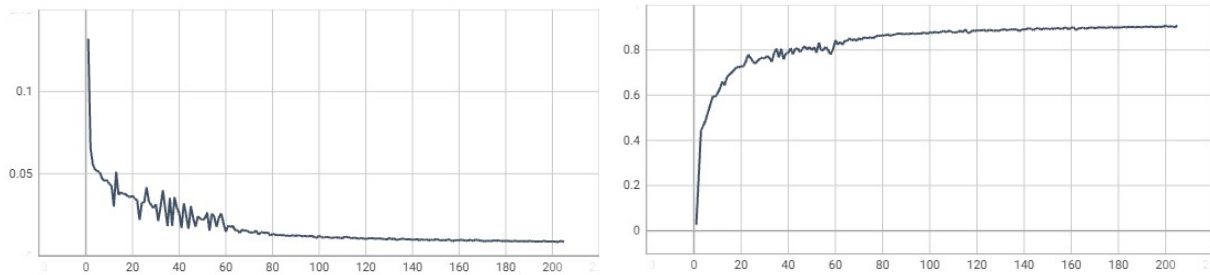


Figura 5.3: Valorile funcției cost și a măsurii F1 calculate pentru setul de date de antrenare pe parcursul antrenării.

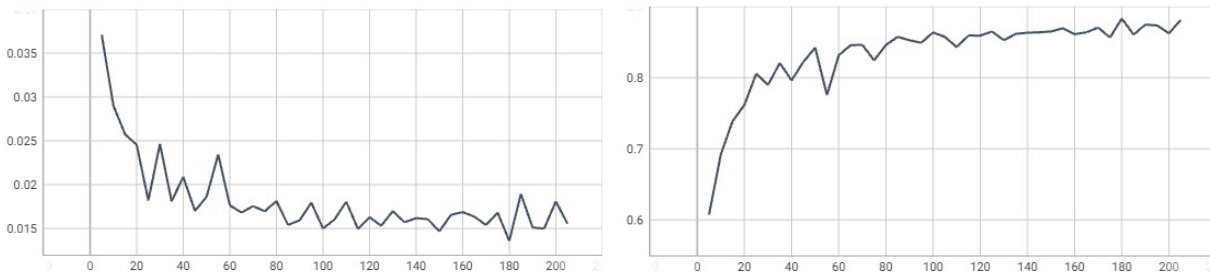


Figura 5.4: Valorile funcției cost și a măsurii F1 calculate pentru setul de date de validare pe parcursul antrenării.

La finalul antrenării, au fost folosite 21 de imagini din setul de date pentru a calcula eficacitatea modelului pe un set de date de test. Valoarea funcției F1 rezultată în etapa de testare este 0.89, puțin mai mare decât 0.88 obținută pentru setul de date de validare

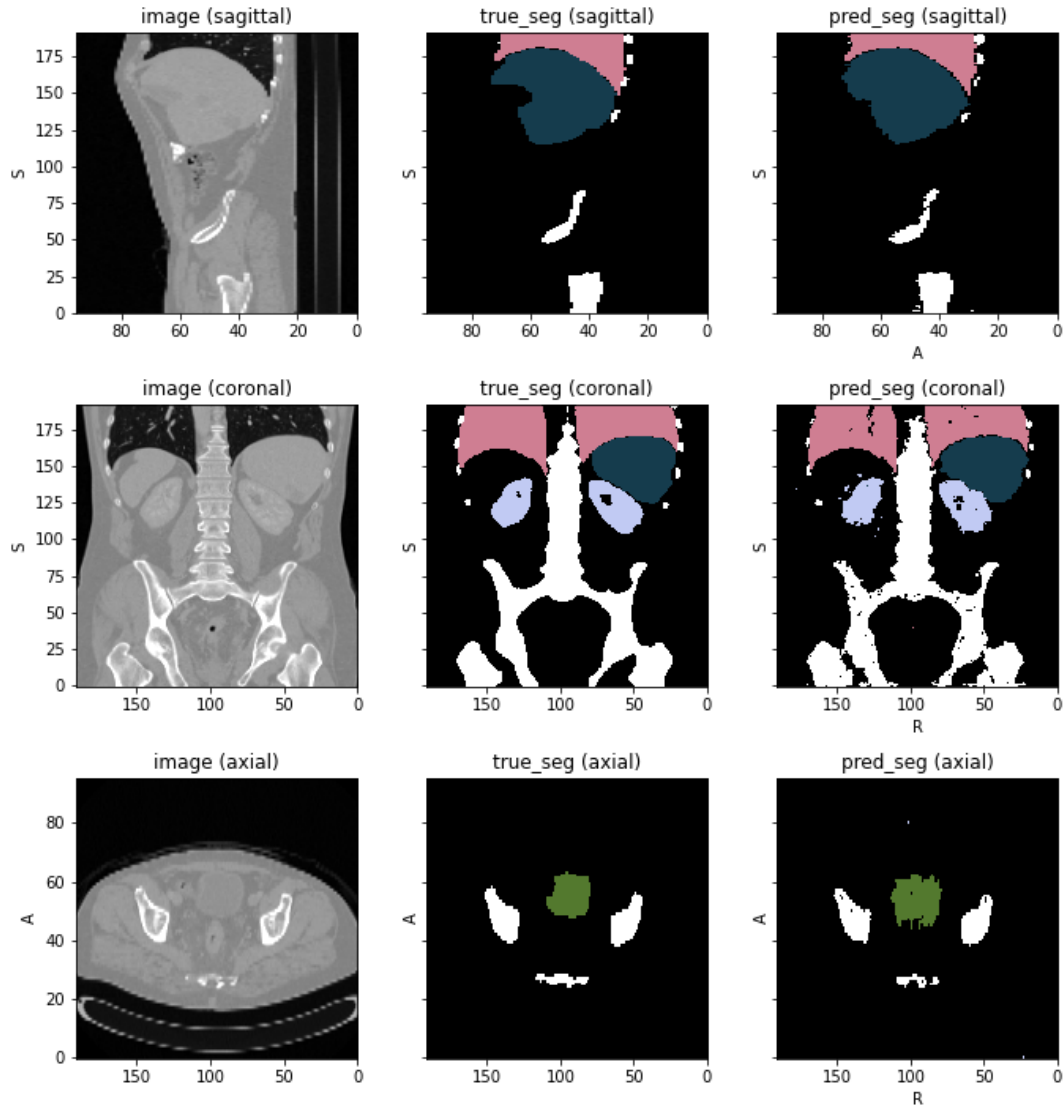


Figura 5.5: O porțiune dintr-o imagine volumetrică din setul de date de validare. În stânga este reprezentată imaginea propriu zisă, în mijloc este segmentarea manuală din setul de date și în dreapta este segmentarea automată.

și puțin mai mică decât 0.90 obținută pentru setul de date folosit la antrenare. Luând în considerare aceste valori poate fi observată o capacitate de generalizare a modelului pentru date neîntâlnite în antrenarea sa. Acest fapt este de dorit pentru folosirea cu succes în practică a modelului. În figurile 5.3 și 5.4 sunt prezentate rezultatele antrenării modelului final, obținut în urma experimentării cu mai multe arhitecturi și diferiți parametri. Modelul folosit în final este cel implementat în biblioteca MONAI care a fost antrenat fără a fi folosite petice disjuncte din imaginile din setul de date. Cu această metodă a fost obținut un model care poate generaliza mai bine într-un timp mai scurt de antrenare decât alternativele explorate pe parcursul experimentării. În figurile 5.5 și 5.2 sunt reprezentări vizuale ale rezultatului antrenării, putând fi observată similitudinea dintre masca de segmentare realizată manual și cea generată de către rețeaua neuronală. În figura 5.5 sunt afișate felii din volum și măștile de segmentare în aceleași poziții, iar în figura 5.2 sunt redări obținute cu ajutorul aplicației de vizualizare realizată în cadrul proiectului.

În figura 5.6 este realizată o comparație între rezultatele antrenării a două arhitecturi de rețele neuronale similare folosind parametrii de antrenare asemănători. Trebuie menționat faptul că o epocă de antrenare pentru rețeaua din biblioteca MONAI este realizată mai rapid

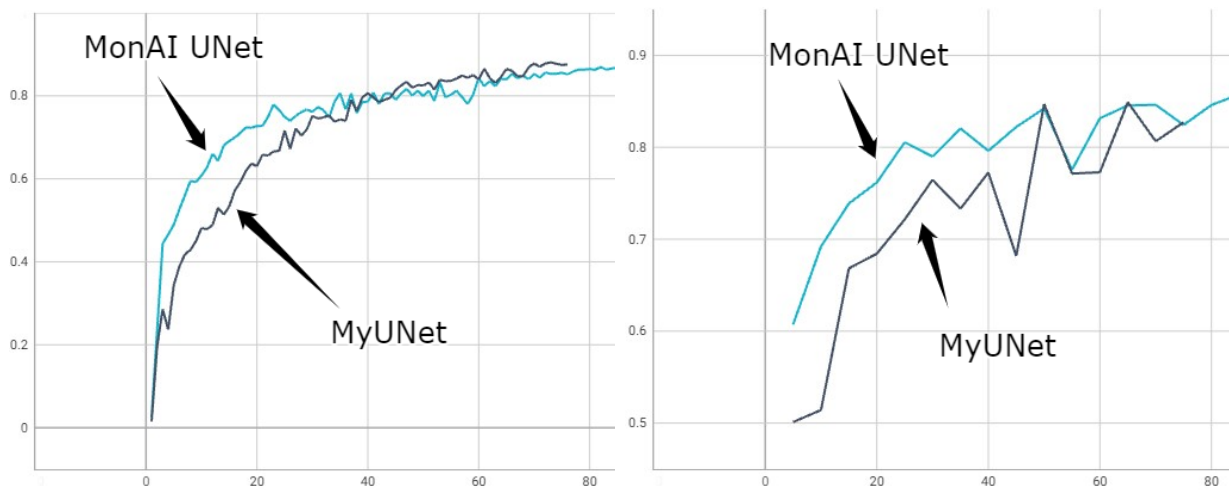


Figura 5.6: Comparație a valorilor funcției F1 între implementarea UNet din biblioteca MONAI și implementarea din anexa 6 pe parcursul antrenării (stangă) și validării (dreapta).

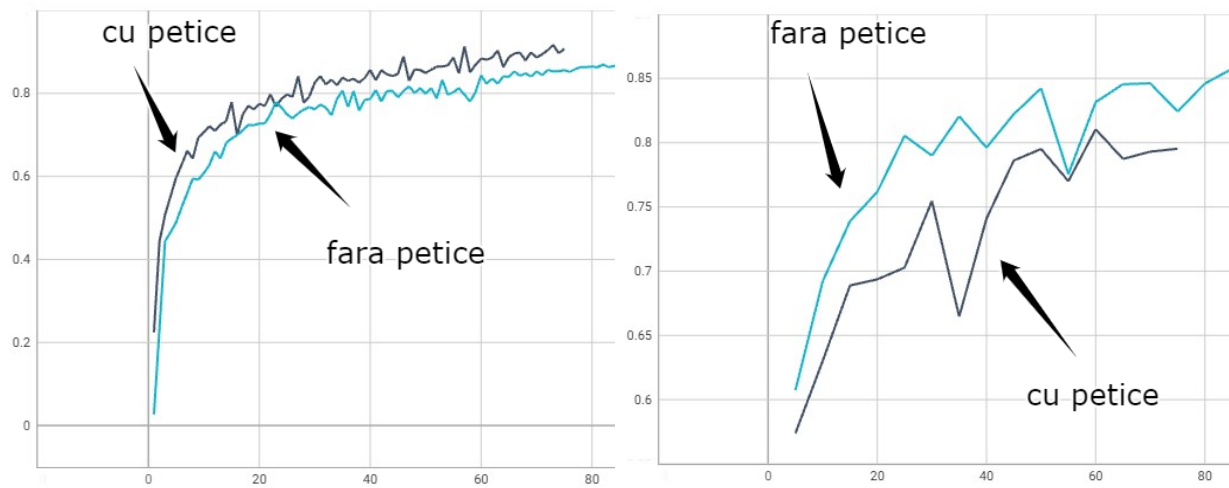


Figura 5.7: Comparație a valorilor funcției F1 între metoda partitionării intrării pe baza de petice și cea în care toată imaginea este primită la intrare de către rețeaua neurală. În stânga sunt valori pe parcursul antrenării, și în dreapta pentru etapa de validare.

decât una pentru rețeaua din anexa 6. Diferența principală dintre cele două implementări este că în MONAI UNet este implementat folosind principii de programare orientată obiect, pe când în cealaltă rețea sunt folosite liste de module din PyTorch pentru înlanțuirea blocurilor convoluționale. În figura 5.7 este realizată o comparație între o metodă de preprocesare a datelor și lipsa ei. Această metodă este împărțirea imaginilor din setul de date în petice disjuncte în scopul de a avea la intrarea rețelei imagini mai mici, astfel realizându-se inferența și retropropagarea mai rapid. Un dezavantaj al acestei metode, după cum se poate observa și în figură este că sunt obținute rezultate mai slabe pe setul de validare, fapt ce arată că modelul antrenat pe petice de date nu generalizează la fel de bine.

Capitolul 6

Concluzii

Vizualizarea imaginilor medicale volumetrice este importantă atât pentru diagnosticarea pacienților de către cadre medicale specializate, cât și pentru studenții la facultatea de medicină. Deoarece volumul de date este mare și structurat în trei dimensiuni, acesta poate fi vizualizat doar în două dimensiuni, astfel vizualizarea acestui tip de date este dificilă. În această lucrare am prezentat implementarea unor tehnici pentru redarea imaginilor semnificative în care regiunile de interes pot fi vizibile și bine delimitate.

Folosind volume din setul de date CT-ORG în aplicația de vizualizare, se pot observa detalii în scanările respective, diferite organe pot fi identificate, și având în vedere că unele scanări provin de la pacienți cu tumori maligne, în unele dintre aceste cazuri pot fi observate astfel de probleme. Este dificil de creat o funcție de transfer care redă imaginea într-un mod realist sau de înaltă calitate, dar poate fi creată o funcție de transfer care îmbunătățește substanțial vizualizarea. Rezultatele segmentării semantice automate, în cele mai multe cazuri, nu conțin erori foarte evidente, iar dacă acestea există, pot fi eliminate prin aplicarea algoritmului de netezire. În urma antrenării, aplicând modelul de segmentare pe datele de test, au rezultat măști de segmentare asemănătoare cu cele create manual de către autorii setului de date.

Dificultățile întâmpinate în implementarea soluției dorite sunt, printre altele: redarea imaginilor volumetrice în mod eficient cu suficiente detalii, construirea unei interfețe ce permite crearea funcțiilor de transfer unidimensionale, antrenarea unei rețele neuronale pentru segmentarea semantică a datelor volumetrice și încărcarea acestora în aplicația de vizualizare, aplicarea măștii de segmentare și salvare în memoria nevolatilă a funcției de transfer și a măștii de segmentare și citirea și încărcarea în memorie a datelor volumetrice.

Fiind o sarcină dificilă, metodele implementate pot fi îmbunătățite, sau pot fi implementate metode diferite care să îmbunătățească vizualizarea în cadrul imagisticii medicale. Implementarea funcțiilor de transfer multidimensionale este o primă direcție de dezvoltare, și constă în cercetarea, implementarea și testarea modalităților de creare a funcțiilor de transfer multidimensionale. De asemenea, poate fi luată în considerare și implementarea unei metode de a modifica manual rezultatul segmentării automate, deoarece metodele folosite pentru calcularea măștii de segmentare nu sunt perfecte.

Bibliografie

- [1] A. Kaufman and K. Mueller, “Overview of volume rendering,” *Visualization Handbook*, vol. 7, pp. 127–XI, 12 2005. → pg. 3, 4
- [2] C. Rezk Salama, M. Keller, and P. Kohlmann, “High-level user interfaces for transfer function design with semantics,” *IEEE transactions on visualization and computer graphics*, vol. 12, pp. 1021–8, 10 2006. → pg. 3, 4
- [3] M. Levoy, “Display of surfaces from volume data,” *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, 1988. → pg. 4
- [4] J. Kniss, G. Kindlmann, and C. Hansen, “Multidimensional transfer functions for volume rendering,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, pp. 270–285, 08 2002. → pg. 4, 5, 7
- [5] J. Zhou and M. Takatsuka, “Automatic transfer function generation using contour tree controlled residue flow model and color harmonics,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, 2009. → pg. 4, 28
- [6] M. Ikits, J. Kniss, and A. L. C. Hansen, “Volume rendering techniques,” *GPU Gems*, 2007. [Online]. Available: <https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-39-volume-rendering-techniques> → pg. 4, 6, 7, 20
- [7] M. E. Palmer, B. Totty, and S. Taylor, “Ray casting on shared-memory architectures: Memory-hierarchy considerations in volume rendering,” *IEEE Concurrency (out of print)*, vol. 6, no. 01, pp. 20–35, jan 1998. → pg. 5
- [8] P. Ljung, “Adaptive Sampling in Single Pass, GPU-based Raycasting of Multiresolution Volumes,” in *Volume Graphics*, R. Machiraju and T. Moeller, Eds. The Eurographics Association, 2006. → pg. 5
- [9] M. Abdellah, A. Eldeib, and A. Sharawi, “High performance gpu-based fourier volume rendering,” *International Journal of Biomedical Imaging*, vol. 2015, p. 590727, Feb 2015. [Online]. Available: <https://doi.org/10.1155/2015/590727> → pg. 7, 8
- [10] P. Radiuk, “Applying 3d u-net architecture to the task of multi-organ segmentation in computed tomography,” *Applied Computer Systems*, vol. 25, pp. 43–50, 05 2020. → pg. 8, 9
- [11] F.-F. Li, A. Karpathy, and J. Johnson, “Cs231n: Convolutional neural networks for visual recognition 2016.” [Online]. Available: <http://cs231n.stanford.edu/> → pg. 8
- [12] A. F. Agarap, “Deep learning using rectified linear units (relu),” *CoRR*, vol. abs/1803.08375, 2018. [Online]. Available: <http://arxiv.org/abs/1803.08375> → pg. 8

- [13] A. A. Novikov, D. Major, M. Wimmer, D. Lenis, and K. Buhler, “Deep sequential segmentation of organs in volumetric medical scans,” *IEEE Transactions on Medical Imaging*, vol. 38, no. 5, p. 1207–1215, May 2019. [Online]. Available: <http://dx.doi.org/10.1109/TMI.2018.2881678> → pg. 8
- [14] J. Weiss and N. Navab, “Deep direct volume rendering: Learning visual feature mappings from exemplary images,” 2021. → pg. 8
- [15] S. Chen, K. Ma, and Y. Zheng, “Med3d: Transfer learning for 3d medical image analysis,” 2019. → pg. 9
- [16] P. Bilic, P. F. Christ, E. Vorontsov, G. Chlebus, H. Chen, Q. Dou, C. Fu, X. Han, P. Heng, J. Hesser, S. Kadoury, T. K. Konopczynski, M. Le, C. Li, X. Li, J. Lipková, J. S. Lowengrub, H. Meine, J. H. Moltz, C. Pal, M. Piraud, X. Qi, J. Qi, M. Rempfler, K. Roth, A. Schenk, A. Sekuboyina, P. Zhou, C. Hülsemeyer, M. Beetz, F. Ettlinger, F. Grün, G. Kaissis, F. Lohöfer, R. Braren, J. Holch, F. Hofmann, W. H. Sommer, V. Heinemann, C. Jacobs, G. E. H. Mamani, B. van Ginneken, G. Chartrand, A. Tang, M. Drozdal, A. Ben-Cohen, E. Klang, M. M. Amitai, E. Konen, H. Greenspan, J. Moreau, A. Hostettler, L. Soler, R. Vivanti, A. Szeskin, N. Lev-Cohain, J. Sosna, L. Joskowicz, and B. H. Menze, “The liver tumor segmentation benchmark (lits),” *CoRR*, vol. abs/1901.04056, 2019. [Online]. Available: <http://arxiv.org/abs/1901.04056> → pg. 12
- [17] M. AI, “Pytorch documentation,” 2022. [Online]. Available: <https://pytorch.org/docs> → pg. 12
- [18] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. [Online]. Available: <https://arxiv.org/abs/1412.6980> → pg. 14
- [19] K. Group, “Opendl 4.x reference,” 2022. [Online]. Available: <https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/> → pg. 19
- [20] J. de Vries, *Learn OpenGL*, 2nd ed., 2015. [Online]. Available: <https://learnopengl.com/book/offline%20learnopengl.pdf> → pg. 19, 20

Anexe

Anexa 1. main.cpp

```
1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <algorithm>
5
6 #include <chrono>
7 #include <thread>
8
9 #include <stdio.h>
10
11 #include <GL/glew.h>
12 #include <GL/freeglut.h>
13
14 #include <glm/mat4x4.hpp>
15 #include <glm/gtx/transform.hpp>
16 #include <glm/gtc/type_ptr.hpp>
17
18 #include "Volume.h"
19 #include "Shader.h"
20 #include "PytorchModel.h"
21 #include "Interface.h"
22
23 #define FRAG_SHADER_PATH "source/shaders/raycasting.frag"
24 #define VERT_SHADER_PATH "source/shaders/raycasting.vert"
25
26 #define PYTORCH_SEGMENTATION_MODULE_PATH "segmentation_model.pt"
27
28 Volume v;
29 Shader s;
30 PytorchModel ptModel;
31 glm::mat4 projection, view, model;
32
33 glm::vec3 eyePos(0.0f, 0.0f, 1.5f);
34
35 Interface ui;
36
37 float deltaTime = FRAME_DURATION;
38
39 void render();
40
41 void display()
```

```

42 {
43     std::chrono::steady_clock::time_point begin =
44         ↪ std::chrono::steady_clock::now();
45
46     ui.render(deltaTime);
47
48     render();
49
50     ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
51
52     glutSwapBuffers();
53
54     std::chrono::steady_clock::time_point end =
55         ↪ std::chrono::steady_clock::now();
56
57     int passed = std::chrono::duration_cast<std::chrono::microseconds>(end -
58         ↪ begin).count();
59     std::this_thread::sleep_for(std::chrono::microseconds(std::max(0,
60         ↪ FRAME_DURATION * 1000 - passed)));
61
62     deltaTime = std::max(FRAME_DURATION * 1000, passed) / 1000;
63 }
64
65 void LoadShaders()
66 {
67     s.Load(VERT_SHADER_PATH, FRAG_SHADER_PATH);
68     s.use();
69
70     s.setVec2("screen", ui.windowWidth, ui.windowHeight);
71     s.setMat4("xtoi", v.xtoi);
72     s.setVec3("scale", v.sizeCorrection);
73
74     glActiveTexture(GL_TEXTURE0);
75     glBindTexture(GL_TEXTURE_3D, v.texID);
76     s.setInt("volumeTex", 0);
77
78     glActiveTexture(GL_TEXTURE2);
79     glBindTexture(GL_TEXTURE_3D, v.segID);
80     s.setInt("segTex", 2);
81
82     glActiveTexture(GL_TEXTURE1);
83     glBindTexture(GL_TEXTURE_1D, v.tfID);
84     s.setInt("tf", 1);
85
86     glActiveTexture(GL_TEXTURE3);
87     glBindTexture(GL_TEXTURE_1D, v.segColorID);
88     s.setInt("segColors", 3);
89 }
90
91 void render()
92 {
93     v.LoadTF(ui.getTFColorMap());
94     v.applySegmentationColors();
95

```

```

92     glClearColor(0, 0, 0, 1.0f);
93
94     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
95
96     model = glm::rotate(ui.angleX, glm::vec3(1.0f, 0.0f, 0.0f));
97     model *= glm::rotate(ui.angleY, glm::vec3(0.0f, 1.0f, 0.0f));
98     model *= glm::rotate(ui.angleZ, glm::vec3(0.0f, 0.0f, 1.0f));
99     model *= glm::translate(glm::vec3(-0.5f, -0.5f, -0.5f));
100    model *= glm::translate(glm::vec3(-ui.translationX, -ui.translationY,
    ↪ -ui.translationZ));
101
102    s.setVec3("translation", glm::vec3(ui.translationX, ui.translationY,
    ↪ ui.translationZ));
103
104    s.setVec3("bbLow", ui.bbLow);
105    s.setVec3("bbHigh", ui.bbHigh);
106
107    s.setMat4("modelMatrix", model);
108
109    ui.zoom = std::max(-0.75f, std::min(1.0f, ui.zoom));
110    eyePos = glm::vec3(0.0f, 0.0f, 1.0f + ui.zoom);
111    view = glm::lookAt(eyePos,
112        glm::vec3(0.0f, 0.0f, 0.0f),
113        glm::vec3(0.0f, 1.0f, 0.0f));
114
115    s.setVec3("origin", glm::vec4(eyePos, 0) * model);
116
117    model = view * model;
118    s.setMat4("viewMatrix", model);
119    s.setMat4("MVP", projection * model);
120
121    s.setFloat("intensityCorrection", ui.intensityCorrection);
122    s.setFloat("exposure", ui.exposure);
123    s.setFloat("gamma", ui.gamma);
124    s.setInt("sampleRate", ui.sampleRate);
125
126
127    if (v.vao != NULL)
128    {
129        glBindVertexArray(v.vao);
130        glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, (GLuint*)NULL);
131    }
132 }
133
134 void init()
135 {
136     glViewport(0, 0, ui.windowWidth, ui.windowHeight);
137     glewInit();
138
139     glEnable(GL_DEPTH_TEST);
140
141     projection = glm::perspective(1.57f, (GLfloat)ui.windowWidth /
    ↪ ui.windowHeight, 0.1f, 100.f);
142     view = glm::lookAt(eyePos,

```

```

143         glm::vec3(0.0f, 0.0f, 0.0f),
144         glm::vec3(0.0f, 1.0f, 0.0f));
145
146     ptModel.LoadModel(PYTORCH_SEGMENTATION_MODULE_PATH);
147
148     ui.initialize();
149
150     ui.canLoadVolumeFunc([&] { return !v.computingSegmentation &&
151         ↪ !v.smoothingSegmentation; });
152     ui.segmentationAvailableFunc([&] { return v.segmentationData != nullptr;
153         ↪ });
154     ui.canLoadSegmentationFunc([&] { return v.volumeData != nullptr &&
155         ↪ !v.computingSegmentation && !v.smoothingSegmentation; });
156     ui.canComputeSegmentationFunc([&] { return ptModel.isLoaded &&
157         ↪ v.volumeData != nullptr && !v.computingSegmentation &&
158         ↪ !v.smoothingSegmentation; });
159     ui.canSmoothSegmentationFunc([&] { return !v.computingSegmentation &&
160         ↪ !v.smoothingSegmentation && v.segmentationData != nullptr; });
161
162     ui.computeSegmentationFunc([&] { v.computeSegmentation(ptModel); });
163     ui.smoothLabelsFunc([&](int radius) { v.applySmoothingLabels(radius); });
164     ui.loadShadersFunc([&] { loadShaders(); });
165
166     ui.loadVolumeFunc([&](const char* path) { v.load(path); });
167     ui.loadSegmentationFunc([&](const char* path) { v.loadSegmentation(path);
168         ↪ });
169     ui.saveSegmentationFunc([&](const char* path) { v.saveSegmentation(path);
170         ↪ });
171     ui.applySegmentationFunc([&]() { v.applySegmentation(); });
172     ui.getSegmentInfoFunc([&]() { return v.segments; });
173
174     ui.getHistogramFunc([&](int nbBins) {
175         float* hist = new float[nbBins];
176         memset(hist, 0, nbBins * sizeof(float));
177
178         if (v.size.x == 0 || v.size.y == 0 || v.size.z == 0)
179             return hist;
180
181         size_t size = v.size.x * v.size.y * v.size.z;
182
183         short* data = new short[size];
184
185         glActiveTexture(GL_TEXTURE0);
186         glBindTexture(GL_TEXTURE_3D, v.texID);
187
188         glGetTexImage(GL_TEXTURE_3D, 0, GL_LUMINANCE, GL_SHORT, data);
189
190         short maxI = *std::max_element(data, data + size);
191         short minI = *std::min_element(data, data + size);
192
193         if (maxI == minI || maxI == 0)
194             return hist;
195
196         int bin_size = (maxI - minI) / nbBins;

```

```

189
190     for (int i = 0; i < size; ++i)
191         hist[(data[i] - minI) / bin_size] += ((float)data[i] + minI) /
            ↪ maxI;
192
193     //for (int i = 0; i < nbBins; ++i)
194     //    hist[i] = std::log(hist[i] + 1);
195
196     float max = *std::max_element(hist, hist + nbBins);
197     for (int i = 0; i < nbBins; ++i)
198         hist[i] = hist[i] / max;
199
200     delete[] data;
201
202     return hist;
203 });
204 }
205
206 void reshape(int w, int h)
207 {
208     ui.windowWidth = w, ui.windowHeight = h;
209
210     glViewport(0, 0, ui.windowWidth, ui.windowHeight);
211     projection = glm::perspective(1.57f, (GLfloat)ui.windowWidth /
            ↪ ui.windowHeight, 0.1f, 100.f);
212
213     s.setVec2("screen", ui.windowWidth, ui.windowHeight);
214
215     ImGui_ImplGLUT_ReshapeFunc(w, h);
216 }
217
218 void keyboard_wrapper(unsigned char key, int x, int y) { ui.keyboard(key, x,
            ↪ y); };
219 void specialInput_wrapper(int key, int x, int y) { ui.specialInput(key, x, y);
            ↪ };
220 void mouse_wrapper(int button, int state, int x, int y) { ui.mouse(button,
            ↪ state, x, y); };
221 void mouseWheel_wrapper(int button, int dir, int x, int y) {
            ↪ ui.mouseWheel(button, dir, x, y); };
222 void motion_wrapper(int x, int y) { ui.motion(x, y); };
223
224 int main(int argc, char** argv)
225 {
226     glutInit(&argc, argv);
227     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
228     glutInitWindowPosition(200, 200);
229     glutInitWindowSize(ui.windowWidth, ui.windowHeight);
230     glutCreateWindow("Volume Rendering");
231
232     glutDisplayFunc(display);
233     glutIdleFunc(glutPostRedisplay);
234
235     ImGui::CreateContext();
236

```

```

237     ImGui_ImplGLUT_Init();
238     ImGui_ImplGLUT_InstallFuncs();
239     ImGui_ImplOpenGL3_Init();
240
241     glutReshapeFunc(reshape);
242     glutKeyboardFunc(keyboard_wrapper);
243     glutSpecialFunc(specialInput_wrapper);
244     glutMouseFunc(mouse_wrapper);
245     glutMouseWheelFunc(mouseWheel_wrapper);
246     glutMotionFunc(motion_wrapper);
247
248     init();
249
250     glutMainLoop();
251
252     ImGui::SaveIniSettingsToDisk(ImGui::GetIO().IniFilename);
253
254     ImGui_ImplOpenGL3_Shutdown();
255     ImGui_ImplGLUT_Shutdown();
256     ImGui::DestroyContext();
257
258     return 0;
259 }

```

Anexa 2. Volume.h

```

1  #pragma once
2  #include <limits>
3  #include <GL/glew.h>
4  #include <GL/freeglut.h>
5
6  #include <glm/mat4x4.hpp>
7  #include <glm/gtx/transform.hpp>
8  #include <glm/gtc/type_ptr.hpp>
9
10 #include "PytorchModel.h"
11
12 class Volume
13 {
14 public:
15     struct SegmentInfo
16     {
17         bool enabled;
18         float color[3];
19         size_t numVoxels;
20
21         SegmentInfo()
22         {
23             reset();
24         }
25
26         void reset()
27         {

```



```

28         enabled = true;
29         for (int i = 0; i < 3; ++i)
30             color[i] = 1.0f;
31         numVoxels = 0;
32     }
33 };
34
35 GLuint vao;
36 GLuint texID;
37 GLuint tfID;
38 GLuint segID;
39 GLuint segColorID;
40 glm::ivec3 size;
41 glm::vec3 sizeCorrection;
42 glm::mat4 xtoi;
43
44 short* volumeData;
45 uchar* segmentationData;
46 uchar* smoothedSegmentationData;
47
48 SegmentInfo segments[7];
49
50 bool computingSegmentation = false;
51 bool smoothingSegmentation = false;
52
53 void Load(const char* path);
54 void LoadTF(float data[]);
55
56 void LoadSegmentation(const char* path);
57 void saveSegmentation(const char* path);
58 void computeSegmentation(PytorchModel ptModel);
59 void calculateSegmentationInfoNumVoxels();
60
61 void applySegmentation();
62 void applySegmentationColors();
63 void applySmoothingLabels(int smoothingRadius = 0);
64 private:
65     void init();
66 };

```

Anexa 3. Volume.cpp

```

1 #include "Volume.h"
2 #include "Loader.h"
3
4 #include <fstream>
5 #include <algorithm>
6 #include <thread>
7 #include <set>
8
9
10 void Volume::Load(const char* path)
11 {

```

```

12     if(vao == NULL)
13         init();
14     else
15     {
16         GLuint textures[] = { texID, segID };
17         glDeleteTextures(2, textures);
18     }
19
20     {
21         if (volumeData)
22         {
23             delete[] volumeData;
24             volumeData = nullptr;
25         }
26         if (segmentationData)
27         {
28             delete[] segmentationData;
29             segmentationData = nullptr;
30         }
31         if (smoothedSegmentationData)
32         {
33             delete[] smoothedSegmentationData;
34             smoothedSegmentationData = nullptr;
35         }
36
37         for (int i = 0; i < 7; i++)
38             segments[i].reset();
39     }
40
41     {
42         glGenTextures(1, &texID);
43         glBindTexture(GL_TEXTURE_3D, texID);
44
45         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
46         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
47         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
48         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
49         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
50
51         volumeData = readVolume(path, size, xtoi);
52         int maxSize = std::max({ size.x, size.y, size.z });
53         sizeCorrection.x = (float)size.x / maxSize, sizeCorrection.y =
54             ↪ (float)size.y / maxSize, sizeCorrection.z = (float)size.z /
55             ↪ maxSize;
56
57         glTexImage3D(GL_TEXTURE_3D, 0, GL_LUMINANCE, size.x, size.y, size.z, 0,
58             ↪ GL_LUMINANCE, GL_SHORT, volumeData);
59
60         glGenTextures(1, &segID);
61         glBindTexture(GL_TEXTURE_3D, segID);
62
63         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
64         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
65         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);

```

```

63     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
64     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
65
66     GLubyte* buffer = new GLubyte[size.x * size.y * size.z];
67     memset(buffer, UCHAR_MAX, size.x * size.y * size.z);
68     glTexImage3D(GL_TEXTURE_3D, 0, GL_LUMINANCE, size.x, size.y, size.z, 0,
69         ↪ GL_LUMINANCE, GL_UNSIGNED_BYTE, buffer);
70     delete[] buffer;
71 }
72
73 void Volume::loadSegmentation(const char* path)
74 {
75     glm::ivec3 seg_size;
76     glm::mat4 seg_itox;
77     short* buffer = readVolume(path, seg_size, seg_itox);
78
79     if (seg_size.x != size.x || seg_size.y != size.y || seg_size.z != size.z)
80     {
81         delete[] buffer;
82         return;
83     }
84
85     size_t sizeT = size.x * size.y * size.z;
86     segmentationData = new uchar[sizeT];
87     for (int i = 0; i < sizeT; ++i)
88         segmentationData[i] = (uchar)((buffer[i] + SHRT_MAX) / ((2 * SHRT_MAX)
89             ↪ / 5));
89
90     delete[] buffer;
91
92     applySmoothingLabels();
93
94     calculateSegmentationInfoNumVoxels();
95
96     applySegmentation();
97 }
98
99 void Volume::saveSegmentation(const char* path)
100 {
101     size_t sizeT = size.x * size.y * size.z;
102
103     short* buffer = new short[sizeT];
104     for (int i = 0; i < sizeT; ++i)
105         buffer[i] = smoothedSegmentationData[i] * ((2 * SHRT_MAX) / 5) -
106             ↪ SHRT_MAX;
107
108     saveVolume(path, buffer, size);
109
110     delete[] buffer;
111 }
112
113 uchar smoothLabel(uchar* labels, int radius, int x, int y, int z, int width,
114     ↪ int height, int depth)

```

```

113 {
114     int freq[7];
115     std::memset(freq, 0, 7 * sizeof(int));
116
117     for (int i = x - radius; i <= x + radius; ++i)
118         for (int j = y - radius; j <= y + radius; ++j)
119             for (int k = z - radius; k <= z + radius; ++k)
120                 if (i >= 0 && i < width && j >= 0 && j < height && k >= 0 && k <
                     ↪ depth)
121                     freq[labels[k * width * height + j * width + i]] ++;
122
123     return std::distance(freq, std::max_element(freq, freq + 7));
124 }
125
126 void Volume::applySmoothingLabels(int smoothingRadius)
127 {
128     if (smoothingSegmentation)
129         return;
130
131     if (smoothedSegmentationData != nullptr)
132         delete[] smoothedSegmentationData;
133
134     smoothedSegmentationData = new uchar[size.x * size.y * size.z];
135     std::memcpy(smoothedSegmentationData, segmentationData, size.x * size.y *
        ↪ size.z);
136
137     if (smoothingRadius > 0)
138     {
139         std::thread smoothingThreadObj([](Volume* v, int smoothingRadius) {
140             v->smoothingSegmentation = true;
141             for (int x = 0; x < v->size.x; ++x)
142                 for (int y = 0; y < v->size.y; ++y)
143                     for (int z = 0; z < v->size.z; ++z)
144                         v->smoothedSegmentationData[z * v->size.x * v->size.y +
                            ↪ y * v->size.x + x] =
145                             smoothLabel(v->segmentationData, smoothingRadius,
                            ↪ x, y, z, v->size.x, v->size.y, v->size.z);
146             v->smoothingSegmentation = false;
147         }, this, smoothingRadius);
148
149         smoothingThreadObj.detach();
150     }
151 }
152
153 void Volume::computeSegmentation(PytorchModel ptModel)
154 {
155     if (computingSegmentation)
156         return;
157
158     std::thread segmentThreadObj([](Volume *v, PytorchModel ptModel) {
159         v->computingSegmentation = true;
160
161         v->segmentationData = ptModel.forward(v->volumeData, v->size.x,
        ↪ v->size.y, v->size.z);

```

```

162
163     v->applySmoothingLabels();
164
165     v->calculateSegmentationInfoNumVoxels();
166     v->applySegmentation();
167
168     v->computingSegmentation = false;
169 }, this, ptModel);
170
171     segmentThreadObj.detach();
172 }
173
174 void Volume::applySegmentation()
175 {
176     size_t sizeT = size.x * size.y * size.z;
177     uchar* segmentationBuffer = new uchar[sizeT];
178
179     for (int i = 0; i < sizeT; ++i)
180         if (smoothedSegmentationData[i] < 7 &&
181             ↪ segments[smoothedSegmentationData[i]].enabled)
182             segmentationBuffer[i] = UCHAR_MAX *
183                 ↪ ((float)(smoothedSegmentationData[i] + 1) / 8);
184         else
185             segmentationBuffer[i] = 0;
186
187     glBindTexture(GL_TEXTURE_3D, segID);
188     glTexImage3D(GL_TEXTURE_3D, 0, GL_INTENSITY, size.x, size.y, size.z, 0,
189         ↪ GL_LUMINANCE, GL_UNSIGNED_BYTE, segmentationBuffer);
190
191     delete[] segmentationBuffer;
192 }
193
194 void Volume::applySegmentationColors()
195 {
196     if (segColorID == NULL)
197     {
198         glGenTextures(1, &segColorID);
199         glBindTexture(GL_TEXTURE_1D, segColorID);
200
201         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
202         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
203         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
204     }
205
206     float colors[3 * 8];
207     //for (int i = 0; i < 3; ++i)
208     //    colors[i] = 1.0f;
209
210     for (int i = 0; i < 7; ++i)
211         memcpy(colors + 3 * i, segments[i].color, 3 * sizeof(float));
212
213     glBindTexture(GL_TEXTURE_1D, segColorID);
214     glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB32F, 7, 0, GL_RGB, GL_FLOAT, colors);
215 }

```

```

213
214 void Volume::LoadTF(float data[])
215 {
216     if (tfID == NULL)
217     {
218         glGenTextures(1, &tfID);
219         glBindTexture(GL_TEXTURE_1D, tfID);
220
221         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
222         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
223         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
224     }
225
226     glBindTexture(GL_TEXTURE_1D, tfID);
227     glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA32F, 256, 0, GL_RGBA, GL_FLOAT,
228         ↵ data);
229 }
230 void Volume::init()
231 {
232     GLfloat vertices[24] = {
233         0.0, 0.0, 0.0,
234         0.0, 0.0, 1.0,
235         0.0, 1.0, 0.0,
236         0.0, 1.0, 1.0,
237         1.0, 0.0, 0.0,
238         1.0, 0.0, 1.0,
239         1.0, 1.0, 0.0,
240         1.0, 1.0, 1.0
241     };
242
243     GLuint indices[36] = {
244         // front
245         1, 5, 7,
246         7, 3, 1,
247         //back
248         0, 2, 6,
249         6, 4, 0,
250         //left
251         0, 1, 3,
252         3, 2, 0,
253         //right
254         7, 5, 4,
255         4, 6, 7,
256         //up
257         2, 3, 7,
258         7, 6, 2,
259         //down
260         1, 0, 4,
261         4, 5, 1
262     };
263
264     GLuint gbo[2];
265

```

```

266     glGenBuffers(2, gbo);
267     GLuint vertexdat = gbo[0];
268     GLuint veridxdat = gbo[1];
269     glBindBuffer(GL_ARRAY_BUFFER, vertexdat);
270     glBufferData(GL_ARRAY_BUFFER, 24 * sizeof(GLfloat), vertices,
        ↪ GL_STATIC_DRAW);
271
272     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, veridxdat);
273     glBufferData(GL_ELEMENT_ARRAY_BUFFER, 36 * sizeof(GLuint), indices,
        ↪ GL_STATIC_DRAW);
274
275     glGenVertexArrays(1, &vao);
276
277     glBindVertexArray(vao);
278     glEnableVertexAttribArray(0);
279     glEnableVertexAttribArray(1);
280
281     glBindBuffer(GL_ARRAY_BUFFER, vertexdat);
282     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (GLfloat*)NULL);
283     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (GLfloat*)NULL);
284     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, veridxdat);
285 }
286
287 void Volume::calculateSegmentationInfoNumVoxels()
288 {
289     for(int i = 0; i < size.x * size.y * size.z; ++i)
290         if(segmentationData[i] < 7)
291             segments[segmentationData[i]].numVoxels++;
292 }

```

Anexa 4. Shader.h

```

1  #pragma once
2  #include <string>
3  #include <GL/glew.h>
4  #include <GL/freeglut.h>
5  #include <glm/glm.hpp>
6  #include "Volume.h"
7
8  class Shader
9  {
10 public:
11     static GLuint shader_programme;
12
13     void Load(const char* vertexPath, const char* fragmentPath);
14
15     void use();
16
17     void setInt(const std::string& name, int value) const;
18
19     void setFloat(const std::string& name, float value) const;
20
21     void setVec2(const std::string& name, float x, float y) const;

```

```

22
23     void setVec3(const std::string& name, const glm::vec3& value) const;
24     void setVec3(const std::string& name, float x, float y, float z) const;
25
26     void setMat4(const std::string& name, const glm::mat4& mat) const;
27
28 private:
29     GLuint vs;
30     GLuint fs;
31     bool hasCompileErrors(GLuint shader);
32     bool hasLinkErrors();
33 };

```

Anexa 5. Shader.cpp

```

1  #include "Shader.h"
2  #include <glm/gtc/type_ptr.hpp>
3  #include <iostream>
4  #include <vector>
5
6
7  GLuint Shader::shader_programme = NULL;
8
9
10 std::string textFileRead(const char* fn)
11 {
12     std::ifstream ifile(fn);
13     std::string filetext;
14     while (ifile.good()) {
15         std::string line;
16         std::getline(ifile, line);
17         filetext.append(line + "\n");
18     }
19     return filetext;
20 }
21
22 void Shader::Load(const char* vertexPath, const char* fragmentPath)
23 {
24     std::string vstext = textFileRead(vertexPath);
25     std::string fstext = textFileRead(fragmentPath);
26     const char* vertex_shader = vstext.c_str();
27     const char* fragment_shader = fstext.c_str();
28
29     vs = glCreateShader(GL_VERTEX_SHADER);
30     glShaderSource(vs, 1, &vertex_shader, NULL);
31     glCompileShader(vs);
32     if (hasCompileErrors(vs))
33         exit(EXIT_FAILURE);
34
35     fs = glCreateShader(GL_FRAGMENT_SHADER);
36     glShaderSource(fs, 1, &fragment_shader, NULL);
37     glCompileShader(fs);
38     if (hasCompileErrors(fs))

```



```

39         exit(EXIT_FAILURE);
40     }
41
42 void detachShaders()
43 {
44     const GLsizei maxCount = 2;
45     GLsizei count;
46     GLuint shaders[maxCount];
47     glGetAttachedShaders(shader_programme, maxCount, &count, shaders);
48
49     for (int i = 0; i < count; i++)
50         glDetachShader(shader_programme, shaders[i]);
51 }
52
53 void Shader::use()
54 {
55     if(shader_programme == NULL)
56         shader_programme = glCreateProgram();
57
58     detachShaders();
59
60     glAttachShader(shader_programme, vs);
61     glAttachShader(shader_programme, fs);
62
63     glLinkProgram(shader_programme);
64
65     if (hasLinkErrors())
66         exit(EXIT_FAILURE);
67
68     glUseProgram(shader_programme);
69 }
70
71 void Shader::setInt(const std::string& name, int value) const
72 {
73     glUniform1i(glGetUniformLocation(shader_programme, name.c_str()), value);
74 }
75
76 void Shader::setFloat(const std::string& name, float value) const
77 {
78     glUniform1f(glGetUniformLocation(shader_programme, name.c_str()), value);
79 }
80
81 void Shader::setVec2(const std::string& name, float x, float y) const
82 {
83     glUniform2f(glGetUniformLocation(shader_programme, name.c_str()), x, y);
84 }
85
86 void Shader::setVec3(const std::string& name, const glm::vec3& value) const
87 {
88     glUniform3fv(glGetUniformLocation(shader_programme, name.c_str()), 1,
89         ↪ &value[0]);
89 }
90
91 void Shader::setVec3(const std::string& name, float x, float y, float z) const

```

```

92 {
93     glUniform3f(glGetUniformLocation(shader_programme, name.c_str()), x, y,
94         ↪ z);
95 }
96 void Shader::setMat4(const std::string& name, const glm::mat4& mat) const
97 {
98     glUniformMatrix4fv(glGetUniformLocation(shader_programme, name.c_str()),
99         ↪ 1, GL_FALSE, glm::value_ptr(mat));
100 }
101 bool Shader::hasCompileErrors(GLuint shader)
102 {
103     GLint isCompiled = 0;
104     glGetShaderiv(shader, GL_COMPILE_STATUS, &isCompiled);
105     if (isCompiled == GL_FALSE)
106     {
107         GLint maxLength = 0;
108         glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &maxLength);
109
110         std::vector<GLchar> errorLog(maxLength);
111         glGetShaderInfoLog(shader, maxLength, &maxLength, &errorLog[0]);
112
113         std::cerr << errorLog.data() << std::endl;
114
115         glDeleteShader(shader);
116         return true;
117     }
118
119     return false;
120 }
121
122 bool Shader::hasLinkErrors()
123 {
124     GLint isLinked = 0;
125     glGetProgramiv(shader_programme, GL_LINK_STATUS, &isLinked);
126     if (isLinked == GL_FALSE)
127     {
128         GLint maxLength = 0;
129         glGetProgramiv(shader_programme, GL_INFO_LOG_LENGTH, &maxLength);
130
131         std::vector<GLchar> errorLog(maxLength);
132         glGetProgramInfoLog(shader_programme, maxLength, &maxLength,
133             ↪ &errorLog[0]);
134
135         std::cerr << errorLog.data() << std::endl;
136
137         glDeleteProgram(shader_programme);
138         return true;
139     }
140
141     return false;
142 }

```

Anexa 6. Interface.h

```

1  #pragma once
2
3  #include <imgui/imgui.h>
4  #include <imgui/imgui_internal.h>
5  #include <imgui/backends/imgui_impl_glut.h>
6  #include <imgui/backends/imgui_impl_opengl3.h>
7
8  #include <imguiFD/ImGuiFileDialog.h>
9
10 #include <GL/glew.h>
11 #include <GL/freeglut.h>
12
13 #include <glm/mat4x4.hpp>
14 #include <glm/gtx/transform.hpp>
15 #include <glm/gtc/type_ptr.hpp>
16
17 #include "SettingsEditor.h"
18 #include "VectorColorPicker.h"
19
20 #include "Volume.h"
21 #include "Loader.h"
22
23 #define FRAME_DURATION 32
24
25 class Interface
26 {
27 public:
28     void initialize();
29     void render(float deltaTime);
30
31     inline float* getTFColormap() { return tfWidget.colormap; }
32
33     inline void canLoadVolumeFunc(const std::function<bool()>& func) {
34         ↪ canLoadVolume = func; };
35     inline void segmentationAvailableFunc(const std::function<bool()>& func) {
36         ↪ segmentationAvailable = func; };
37     inline void canLoadSegmentationFunc(const std::function<bool()>& func) {
38         ↪ canLoadSegmentation = func; };
39     inline void canComputeSegmentationFunc(const std::function<bool()>& func)
40         ↪ { canComputeSegmentation = func; };
41     inline void canSmoothSegmentationFunc(const std::function<bool()>& func) {
42         ↪ canSmoothSegmentation = func; };
43
44     inline void computeSegmentationFunc(const std::function<void()>& func) {
45         ↪ computeSegmentation = func; };
46     inline void smoothLabelsFunc(const std::function<void(int)>& func) {
47         ↪ smoothLabels = func; };
48     inline void loadShadersFunc(const std::function<void()>& func) {
49         ↪ loadShaders = func; };
50
51     inline void loadVolumeFunc(const std::function<void(const char*)>& func) {
52         ↪ loadVolume = func; };

```

```

44     inline void LoadSegmentationFunc(const std::function<void(const char*)>&
    ↪ func) { LoadSegmentation = func; };
45     inline void saveSegmentationFunc(const std::function<void(const char*)>&
    ↪ func) { saveSegmentation = func; };
46     inline void applySegmentationFunc(const std::function<void()>& func) {
    ↪ applySegmentation = func; };
47
48     inline void getHistogramFunc(const std::function<float* (int nbBins)>&
    ↪ func) { getHistogram = func; };
49     inline void getSegmentInfoFunc(const std::function<Volume::SegmentInfo*
    ↪ ()>& func) { getSegmentInfo = func; };
50
51     void keyboard(unsigned char key, int x, int y);
52     void specialInput(int key, int x, int y);
53     void mouse(int button, int state, int x, int y);
54     void mouseWheel(int button, int dir, int x, int y);
55     void motion(int x, int y);
56
57     float angleY = 3.14f, angleX = 1.57f, angleZ = 0.0f;
58     float translationX, translationY, translationZ;
59     float zoom = 0.5f;
60
61     float intensityCorrection = 0.1;
62     float exposure = 1, gamma = 1;
63     int sampleRate = 100;
64
65     bool autoRotate = false;
66
67     glm::vec3 bbLow = glm::vec3(-0.5);
68     glm::vec3 bbHigh = glm::vec3(0.5);
69
70     int windowHeight = 1240, windowHeight = 800;
71
72 private:
73     void display();
74
75     void mainMenuBar();
76     void volumWindow();
77     void tfWindow();
78
79     void volumeFileMenu();
80     void tfFileMenu();
81
82     void segmentationPropertyEditor();
83     void sliceSlider(const char* label, float* min, float* max, float v_min,
    ↪ float v_max);
84
85     ImGuiIO* io;
86
87     std::function<bool()> canLoadVolume;
88     std::function<bool()> segmentationAvailable;
89     std::function<bool()> canLoadSegmentation;
90     std::function<bool()> canComputeSegmentation;
91     std::function<bool()> canSmoothSegmentation;

```

```

92
93     std::function<void()> computeSegmentation;
94     std::function<void(int)> smoothLabels;
95     std::function<void()> loadShaders;
96
97     std::function<void(const char*)> loadVolume;
98     std::function<void(const char*)> loadSegmentation;
99     std::function<void(const char*)> saveSegmentation;
100    std::function<void()> applySegmentation;
101
102    std::function<float* (int nbBins)> getHistogram;
103    std::function<Volume::SegmentInfo* ()> getSegmentInfo;
104
105    SettingsEditor settingsEditor;
106    VectorColorPicker tfWidget;
107
108    bool show_volume_window = true;
109    bool show_tf_window = true;
110    bool show_settings_editor = false;
111
112    const char* nifti_filter = "NIFTI (*.nii.gz *.nii){(.\.nii\.gz),.nii}";
113
114    int smoothingRadius = 0;
115
116    int oldX, oldY;
117 };

```

Anexa 7. Interface.cpp

```

1  #include "Interface.h"
2
3
4  void Interface::initialize()
5  {
6      io = &ImGui::GetIO();
7
8      io->ConfigWindowsMoveFromTitleBarOnly = true;
9      io->ConfigWindowsResizeFromEdges = true;
10     io->ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
11
12     settingsEditor.Initialize();
13 }
14
15 void Interface::render(float deltaTime)
16 {
17     ImGui_ImplOpenGL3_NewFrame();
18     ImGui_ImplGLUT_NewFrame();
19
20     display();
21     ImGui::Render();
22
23     if (autoRotate)

```

```

24         angleY += SettingsEditor::inputSettings.rotationSpeed * deltaTime /
           ↪ 1000;
25     }
26
27 void Interface::display()
28 {
29     mainMenuBar();
30
31     if (show_volume_window)
32         volumWindow();
33
34     if (show_tf_window)
35         tfWindow();
36
37     if (show_settings_editor)
38         settingsEditor.draw(&show_settings_editor);
39 }
40
41 void Interface::mainMenuBar()
42 {
43     if (ImGui::BeginMainMenuBar())
44     {
45         if (ImGui::BeginMenu("File"))
46         {
47             if (ImGui::BeginMenu("Volume"))
48             {
49                 volumeFileMenu();
50
51                 ImGui::EndMenu();
52             }
53
54             if (ImGui::BeginMenu("Transfer Function"))
55             {
56                 tfFileMenu();
57
58                 ImGui::EndMenu();
59             }
60
61             ImGui::EndMenu();
62         }
63
64         if (ImGui::BeginMenu("Window"))
65         {
66             ImGui::Checkbox("Show Volume Window", &show_volume_window);
67             ImGui::Checkbox("Show TF Window", &show_tf_window);
68             ImGui::Checkbox("Show Settings Editor Window",
           ↪ &show_settings_editor);
69
70             ImGui::EndMenu();
71         }
72
73         if (ImGui::MenuItem("ReLoad Shaders"))
74             LoadShaders();
75

```

```

76     ImGui::EndMainMenuBar();
77 }
78 }
79
80 void Interface::volumWindow()
81 {
82     ImGui::Begin("Volume", &show_volume_window, ImGuiWindowFlags_MenuBar);
83     if (ImGui::BeginMenuBar())
84     {
85         if (ImGui::BeginMenu("File"))
86         {
87             volumeFileMenu();
88
89             ImGui::EndMenu();
90         }
91         if (ImGui::MenuItem("Compute segmentation", NULL, false,
92             ↪ canComputeSegmentation()))
93         {
94             computeSegmentation();
95             LoadShaders();
96         }
97     }
98     ImGui::EndMenuBar();
99 }
100 if (ImGuiFileDialog::Instance()->Display("ChooseVolumeLoad"))
101 {
102     if (ImGuiFileDialog::Instance()->IsOk())
103     {
104         std::string filePathName =
105             ↪ ImGuiFileDialog::Instance()->GetFilePathName();
106         LoadVolume(filePathName.c_str());
107         LoadShaders();
108
109         std::string savPath = (std::string(filePathName) + ".sav");
110         if (std::fstream{ savPath })
111             tfWidget.save(savPath.c_str());
112
113         if (tfWidget.hist != nullptr)
114             delete[] tfWidget.hist;
115         tfWidget.hist = getHistogram(tfWidget.nbBins);
116     }
117     ImGuiFileDialog::Instance()->Close();
118 }
119
120 if (ImGuiFileDialog::Instance()->Display("ChooseSegmentationLoad"))
121 {
122     if (ImGuiFileDialog::Instance()->IsOk())
123     {
124         std::string filePathName =
125             ↪ ImGuiFileDialog::Instance()->GetFilePathName();
126         LoadSegmentation(filePathName.c_str());
127         LoadShaders();

```

```

127     }
128
129     ImGuiFileDialog::Instance()->Close();
130 }
131
132 if (ImGuiFileDialog::Instance()->Display("ChooseSegmentationSave"))
133 {
134     if (ImGuiFileDialog::Instance()->IsOk())
135     {
136         std::string filePathName =
137             ↪ ImGuiFileDialog::Instance()->GetFilePathName();
138         saveSegmentation(filePathName.c_str());
139     }
140     ImGuiFileDialog::Instance()->Close();
141 }
142
143 if (ImGui::CollapsingHeader("Rotation"))
144 {
145     ImGui::PushID("Rotation");
146
147     ImGui::SliderAngle("x", &angleX);
148     ImGui::SliderAngle("y", &angleY);
149     ImGui::SliderAngle("z", &angleZ);
150
151     ImGui::PopID();
152 }
153
154 if (ImGui::CollapsingHeader("Translation"))
155 {
156     ImGui::PushID("Translation");
157
158     ImGui::SliderFloat("x", &translationX, -0.5, 0.5, "%.2f");
159     ImGui::SliderFloat("y", &translationY, -0.5, 0.5, "%.2f");
160     ImGui::SliderFloat("z", &translationZ, -0.5, 0.5, "%.2f");
161
162     ImGui::PopID();
163 }
164
165 if (ImGui::CollapsingHeader("Slicing"))
166 {
167     ImGui::PushID("Slicing");
168
169     sliceSlider("x", &bbLow.x, &bbHigh.x, -0.5, 0.5);
170     sliceSlider("y", &bbLow.y, &bbHigh.y, -0.5, 0.5);
171     sliceSlider("z", &bbLow.z, &bbHigh.z, -0.5, 0.5);
172
173     ImGui::PopID();
174 }
175
176 if (segmentationAvailable())
177 {
178     if (!canSmoothSegmentation())
179         ImGui::BeginDisabled();

```



```

180
181     ImGui::SliderInt("Label smoothing radius", &smoothingRadius, 0, 3);
182     if (ImGui::Button("Apply Label smoothing"))
183         smoothLabels(smoothingRadius);
184
185     if (!canSmoothSegmentation())
186         ImGui::EndDisabled();
187
188     if (ImGui::CollapsingHeader("Semantic segmentation"))
189     {
190         segmentationPropertyEditor();
191     }
192 }
193
194 ImGui::End();
195 }
196
197 void Interface::tfWindow()
198 {
199     ImGui::Begin("Visualization", &show_tf_window, ImGuiWindowFlags_MenuBar);
200     if (ImGui::BeginMenuBar())
201     {
202         if (ImGui::MenuItem("New"))
203             tfWidget.reset();
204
205         if (ImGui::BeginMenu("File"))
206         {
207             tfFileMenu();
208
209             ImGui::EndMenu();
210         }
211
212         ImGui::EndMenuBar();
213     }
214
215     ImGui::SliderInt("Sample Rate", &sampleRate, 50, 300);
216     ImGui::SliderFloat("Intenisty Correction", &intensityCorrection, 0.01, 1,
217         ↪ "%.2f");
218     ImGui::SliderFloat("Exposure", &exposure, 1, 10, "%.1f");
219     ImGui::SliderFloat("Gamma", &gamma, 0.75, 1.25, "%.2f");
220
221     tfWidget.draw();
222
223     if (ImGuiFileDialog::Instance()->Display("ChoseTFOpen"))
224     {
225         if (ImGuiFileDialog::Instance()->IsOk())
226             tfWidget.load(ImGuiFileDialog::Instance()->GetFilePathName().c_
227                 ↪ str());
228
229         ImGuiFileDialog::Instance()->Close();
230     }
231
232     if (ImGuiFileDialog::Instance()->Display("ChoseTFSave"))
233     {

```

```

232     if (ImGuiFileDialog::Instance()->IsOk())
233         tfWidget.save(ImGuiFileDialog::Instance()->GetFilePathName().c_
                ↪ str());
234
235     ImGuiFileDialog::Instance()->Close();
236 }
237
238 ImGui::End();
239 }
240
241 void Interface::volumeFileMenu()
242 {
243     if (ImGui::MenuItem("Load", 0, false, canLoadVolume()))
244         ImGuiFileDialog::Instance()->OpenDialog("ChooseVolumeLoad", "Choose
                ↪ Volume", nifti_filter, ".");
245
246     if (ImGui::BeginMenu("Segmentation"))
247     {
248         if (ImGui::MenuItem("Load", 0, false, canLoadSegmentation()))
249             ImGuiFileDialog::Instance()->OpenDialog("ChooseSegmentationLoad",
                ↪ "Choose Segmentation", nifti_filter, ".");
250
251         if (ImGui::MenuItem("Save", 0, false, segmentationAvailable()))
252             ImGuiFileDialog::Instance()->OpenDialog("ChooseSegmentationSave",
                ↪ "Choose Segmentation", nifti_filter, ".", 1, nullptr,
                ↪ ImGuiFileDialogFlags_ConfirmOverwrite);
253
254         ImGui::EndMenu();
255     }
256 }
257
258 void Interface::tfFileMenu()
259 {
260     if (ImGui::MenuItem("Load"))
261         ImGuiFileDialog::Instance()->OpenDialog("ChoseTFOpen", "Choose TF",
                ↪ ".sav", ".");
262     if (ImGui::MenuItem("Save"))
263         ImGuiFileDialog::Instance()->OpenDialog("ChoseTFSave", "Choose TF",
                ↪ ".sav", ".", 1, nullptr, ImGuiFileDialogFlags_ConfirmOverwrite);
264 }
265
266 void Interface::segmentationPropertyEditor()
267 {
268     std::string labels[] = { "background", "Liver", "bladder", "Lungs",
                ↪ "kidneys", "bone", "brain" };
269
270     if (ImGui::BeginTable("seg_edit", 2, ImGuiTableFlags_BordersOuter |
                ↪ ImGuiTableFlags_Resizable))
271     {
272         for (int i = 0; i < 7; i++)
273         {
274             ImGui::PushID(i);
275
276             ImGui::TableNextRow();

```

```

277     ImGui::TableSetColumnIndex(0);
278     bool is_open = ImGui::TreeNode("Segment", labels[i].c_str());
279     ImGui::TableSetColumnIndex(1);
280     ImGui::Text("%d voxels", getSegmentInfo()[i].numVoxels);
281
282     if (is_open)
283     {
284         std::map<std::string, std::function<void()>> rows = {
285             {"Enabled", [&]() {
286                 if (ImGui::Checkbox(labels[i].c_str(),
287                     ↪ &getSegmentInfo()[i].enabled))
288                     applySegmentation();
289             }},
290             {"Color", [&]() {
291                 ImGui::ColorEdit3("", getSegmentInfo()[i].color);
292             }}
293         };
294         for (const auto& [label, row] : rows)
295         {
296             ImGui::PushID(0);
297
298             ImGui::TableNextRow();
299             ImGui::TableSetColumnIndex(0);
300             ImGui::TreeNodeEx("Field", ImGuiTreeNodeFlags_Leaf |
301                 ↪ ImGuiTreeNodeFlags_NoTreePushOnOpen, label.c_str());
302
303             ImGui::TableSetColumnIndex(1);
304             row();
305
306             ImGui::NextColumn();
307             ImGui::PopID();
308         }
309         ImGui::TreePop();
310     }
311
312     ImGui::PopID();
313 }
314
315 ImGui::EndTable();
316 }
317 }
318
319 void Interface::sliceSlider(const char* label, float* min, float* max, float
320     ↪ v_min, float v_max)
321 {
322     ImGuiWindow* window = ImGui::GetCurrentWindow();
323     if (window->SkipItems)
324         return;
325
326     ImGuiContext& g = *GImGui;
327     const ImGuiStyle& style = g.Style;
328     const ImGuiID id = window->GetID(label);

```

```

328     const float w = ImGui::CalcItemWidth();
329
330     const ImVec2 label_size = ImGui::CalcTextSize(label, NULL, true);
331     const ImRect frame_bb(window->DC.CursorPos, ImVec2(window->DC.CursorPos.x
    ↪ + w, window->DC.CursorPos.y + label_size.y + style.FramePadding.y *
    ↪ 2.0f));
332     const ImRect total_bb(frame_bb.Min, ImVec2(frame_bb.Max.x +
    ↪ style.ItemInnerSpacing.x + label_size.x, frame_bb.Max.y));
333
334     ImGui::ItemAdd(total_bb, id);
335     ImGui::ItemSize(total_bb, style.FramePadding.y);
336
337     bool hovered = g.HoveredWindow == window &&
    ↪ ImGui::IsMouseHoveringRect(frame_bb.Min, frame_bb.Max);
338     if (hovered)
339         ImGui::SetHoveredID(id);
340
341     if (hovered && g.IO.MouseClicked[0])
342     {
343         ImGui::SetActiveID(id, window);
344         ImGui::FocusWindow(window);
345     }
346     else if (g.ActiveId == id && !g.IO.MouseDown[0])
347     {
348         ImGui::SetActiveID(0, NULL);
349         ImGui::FocusWindow(NULL);
350     }
351
352     ImGui::RenderFrame(frame_bb.Min, frame_bb.Max,
    ↪ ImGui::GetColorU32(ImGuiCol_FrameBg), style.FrameBorderSize > 0,
    ↪ style.FrameRounding);
353
354     const float grab_padding = 2.0f;
355     const float slider_size = frame_bb.GetWidth() - grab_padding * 2;
356     const float grab_size = std::min(style.GrabMinSize, slider_size);
357     const float slider_usable_size = slider_size - grab_size;
358
359     if (g.ActiveId == id)
360     {
361         float clicked_t = ImClamp((g.IO.MousePos.x - frame_bb.Min.x) /
    ↪ slider_usable_size, 0.0f, 1.0f);
362         float new_value = ImLerp(v_min, v_max, clicked_t);
363
364         if (abs(*min - new_value) < abs(*max - new_value))
365             *min = new_value;
366         else
367             *max = new_value;
368     }
369
370     float grab_t = (*min - v_min) / (v_max - v_min);
371     float grab_pos = ImLerp(frame_bb.Min.x, frame_bb.Max.x, grab_t);
372     ImRect grab_bb1 = ImRect(ImVec2(grab_pos - grab_size * 0.5f,
    ↪ frame_bb.Min.y + grab_padding), ImVec2(grab_pos + grab_size * 0.5f,
    ↪ frame_bb.Max.y - grab_padding));

```

```

373 window->DrawList->AddRectFilled(grab_bb1.Min, grab_bb1.Max,
    ↪ ImGui::GetColorU32(g.ActiveId == id ? ImGuiCol_SliderGrabActive :
    ↪ ImGuiCol_SliderGrab), style.GrabRounding);
374
375 grab_t = (*max - v_min) / (v_max - v_min);
376 grab_pos = ImLerp(frame_bb.Min.x, frame_bb.Max.x, grab_t);
377 ImRect grab_bb2 = ImRect(ImVec2(grab_pos - grab_size * 0.5f,
    ↪ frame_bb.Min.y + grab_padding), ImVec2(grab_pos + grab_size * 0.5f,
    ↪ frame_bb.Max.y - grab_padding));
378 window->DrawList->AddRectFilled(grab_bb2.Min, grab_bb2.Max,
    ↪ ImGui::GetColorU32(g.ActiveId == id ? ImGuiCol_SliderGrabActive :
    ↪ ImGuiCol_SliderGrab), style.GrabRounding);
379
380 ImRect connector(grab_bb1.Min, grab_bb2.Max);
381 connector.Min.x += grab_size;
382 connector.Min.y += grab_size * 0.3f;
383 connector.Max.x -= grab_size;
384 connector.Max.y -= grab_size * 0.3f;
385
386 window->DrawList->AddRectFilled(connector.Min, connector.Max,
    ↪ ImGui::GetColorU32(ImGuiCol_SliderGrab), style.GrabRounding);
387
388 char value_buf[64];
389 const char* value_buf_end = value_buf + ImFormatString(value_buf,
    ↪ IM_ARRAYSIZE(value_buf), "%.2f %.2f", *min, *max);
390 ImGui::RenderTextClipped(frame_bb.Min, frame_bb.Max, value_buf,
    ↪ value_buf_end, NULL, ImVec2(0.5f, 0.5f));
391
392 ImGui::RenderText(ImVec2(frame_bb.Max.x + style.ItemInnerSpacing.x * 2,
    ↪ frame_bb.Min.y + style.FramePadding.y), label);
393 }
394
395 void Interface::keyboard(unsigned char key, int x, int y)
396 {
397     float translationSpeed = SettingsEditor::inputSettings.translationSpeed /
    ↪ FRAME_DURATION;
398
399     if (io != nullptr && io->WantCaptureKeyboard)
400     {
401         ImGui_ImplGLUT_KeyboardFunc(key, x, y);
402         return;
403     }
404
405     if (glutGetModifiers() == GLUT_ACTIVE_ALT)
406     {
407         switch (key)
408         {
409             case '+':
410             case 'w':
411                 translationZ -= translationSpeed;
412                 break;
413             case '-':
414             case 's':
415                 translationZ += translationSpeed;
416                 break;

```

```

416         default:
417             break;
418     }
419     else
420         switch (key)
421         {
422             case ' ':
423                 autoRotate = !autoRotate;
424                 break;
425             case '+':
426             case 'w':
427                 mouseWheel(0, 1, 0, 0);
428                 break;
429             case '-':
430             case 's':
431                 mouseWheel(0, -1, 0, 0);
432                 break;
433             default:
434                 break;
435         }
436
437     ImGui_ImplGLUT_KeyboardFunc(key, x, y);
438 }
439
440 void Interface::specialInput(int key, int x, int y)
441 {
442     float translationSpeed = SettingsEditor::inputSettings.translationSpeed /
443     ↪ FRAME_DURATION;
444     float rotationSpeed = SettingsEditor::inputSettings.rotationSpeed /
445     ↪ FRAME_DURATION;
446
447     if (glutGetModifiers() == GLUT_ACTIVE_ALT)
448     {
449         switch (key)
450         {
451             case GLUT_KEY_UP:
452                 translationY -= translationSpeed;
453                 break;
454             case GLUT_KEY_DOWN:
455                 translationY += translationSpeed;
456                 break;
457             case GLUT_KEY_LEFT:
458                 translationX -= translationSpeed;
459                 break;
460             case GLUT_KEY_RIGHT:
461                 translationX += translationSpeed;
462                 break;
463             default:
464                 break;
465         }
466     }
467     else
468     {
469         switch (key)
470         {
471             case GLUT_KEY_UP:
472                 angleX += rotationSpeed;

```

```

468         break;
469     case GLUT_KEY_DOWN:
470         angleX -= rotationSpeed;
471         break;
472     case GLUT_KEY_LEFT:
473         angleY -= rotationSpeed;
474         break;
475     case GLUT_KEY_RIGHT:
476         angleY += rotationSpeed;
477         break;
478     default:
479         break;
480     }
481
482     ImGui_ImplGLUT_SpecialFunc(key, x, y);
483 }
484
485 void Interface::mouse(int button, int state, int x, int y)
486 {
487     if (io != nullptr && io->WantCaptureMouse)
488     {
489         ImGui_ImplGLUT_MouseFunc(button, state, x, y);
490         return;
491     }
492
493     switch (button)
494     {
495     case GLUT_LEFT_BUTTON:
496         oldX = x, oldY = y;
497         break;
498     default:
499         break;
500     }
501
502     ImGui_ImplGLUT_MouseFunc(button, state, x, y);
503 }
504
505 void Interface::mouseWheel(int button, int dir, int x, int y)
506 {
507     if (io != nullptr && !io->WantCaptureMouse)
508         zoom += dir > 0 ? 0.05f : -0.05f;
509
510     ImGui_ImplGLUT_MouseWheelFunc(button, dir, x, y);
511 }
512
513 void Interface::motion(int x, int y)
514 {
515     float rotationSpeed = SettingsEditor::inputSettings.rotationSpeed;
516
517     if (io != nullptr && !io->WantCaptureMouse)
518     {
519         angleY += (((float)x - oldX) / windowWidth) * rotationSpeed;
520         angleX += (((float)y - oldY) / windowHeight) * rotationSpeed;
521         oldX = x, oldY = y;

```

```

522     }
523
524     ImGui_ImplGLUT_MotionFunc(x, y);
525 }

```

Anexa 8. VectorColorPicker.h

```

1  #pragma once
2
3  #include <stdint>
4  #include <string>
5  #include <vector>
6  #include <imgui/imgui.h>
7  #include <glm/glm.hpp>
8  #include <GL/glew.h>
9  #include <GL/freeglut.h>
10
11 #define IMGUI_DEFINE_MATH_OPERATORS
12 #include "imgui_internal.h"
13 #include "Loader.h"
14
15 #define COLORMAP_SIZE 256
16
17 enum ColorSpace { LINEAR, SRGB };
18
19 class VectorColorPicker {
20
21     std::vector<float> colors = { 1, 1, 1, 1, 1, 1, 1, };
22     std::vector<ImVec2> points = { ImVec2(0.f, 1.f), ImVec2(1.f, 1.f)};
23     size_t selected = -1;
24     size_t previous = -1;
25
26     bool clicked = false;
27     GLuint colormapTex;
28
29 public:
30     float colormap[COLORMAP_SIZE * 4];
31     const int nbBins = 128;
32     float* hist;
33
34     void draw();
35     void reset();
36     void load(const char* path);
37     void save(const char* path);
38
39 private:
40     void updateColormapTexture();
41     void updateColormap();
42
43     const float point_radius = 10.f;
44 };

```


Anexa 9. VectorColorPicker.cpp

```

1  #include "VectorColorPicker.h"
2  #include <algorithm>
3  #include <cmath>
4  #include <iostream>
5  #include <numeric>
6
7  #ifndef TFN_WIDGET_NO_STB_IMAGE_IMPL
8  #define STB_IMAGE_IMPLEMENTATION
9  #endif
10
11
12 inline float Length(ImVec2 v)
13 {
14     return std::sqrt(v.x * v.x + v.y * v.y);
15 }
16
17 void VectorColorPicker::draw()
18 {
19     const ImGuiIO& io = ImGui::GetIO();
20
21     ImGui::Text("Transfer Function");
22
23     if (previous != -1)
24         ImGui::ColorEdit3("Point Color", &colors[previous * 3]);
25
26     ImVec2 canvas_size = ImGui::GetContentRegionAvail();
27     size_t tmp = colormapTex;
28     ImGui::Image(reinterpret_cast<void*>(tmp), ImVec2(canvas_size.x, 16));
29     ImVec2 canvas_pos = ImGui::GetCursorScreenPos();
30     canvas_size.y -= 20;
31
32     ImDrawList* draw_list = ImGui::GetWindowDrawList();
33     draw_list->PushClipRect(canvas_pos, canvas_pos + canvas_size);
34
35     const ImVec2 view_scale(canvas_size.x, -canvas_size.y);
36     const ImVec2 view_offset(canvas_pos.x, canvas_pos.y + canvas_size.y);
37
38     draw_list->AddRect(canvas_pos, canvas_pos + canvas_size,
39         ↪ ImGui::GetColorU32(ImGuiCol_FrameBg));
40
41     ImGui::InvisibleButton("tfn_canvas", canvas_size);
42
43     if (!io.MouseDown[0] && !io.MouseDown[1])
44         clicked = false;
45     if (ImGui::IsItemHovered() && (io.MouseDown[0] || io.MouseDown[1]))
46         clicked = true;
47
48     ImVec2 bbmin = ImGui::GetItemRectMin();
49     ImVec2 bbmax = ImGui::GetItemRectMax();
50     ImVec2 clipped_mouse_pos = ImVec2(std::min(std::max(io.MousePos.x,
51         ↪ bbmin.x), bbmax.x),
52         std::min(std::max(io.MousePos.y, bbmin.y), bbmax.y));

```

```

51
52     if (clicked) {
53         ImVec2 mouse_pos = (ImVec2(clipped_mouse_pos) - view_offset) /
54             ↪ view_scale;
55         mouse_pos.x = ImClamp(mouse_pos.x, 0.f, 1.f);
56         mouse_pos.y = ImClamp(mouse_pos.y, 0.f, 1.f);
57
58         if (io.MouseDown[0])
59         {
60             if (selected != (size_t)-1)
61             {
62                 points[selected] = ImVec2(mouse_pos);
63
64                 if (selected == 0)
65                     points[selected].x = 0.f;
66                 else if (selected == points.size() - 1)
67                     points[selected].x = 1.f;
68             }
69             else {
70                 auto fnd = std::find_if(
71                     points.begin(), points.end(), [&](const ImVec2& p)
72                     {
73                         const ImVec2 pt_pos = p * view_scale + view_offset;
74                         float dist = length(pt_pos -
75                             ↪ ImVec2(clipped_mouse_pos));
76                         return dist <= point_radius;
77                     });
78
79                 if (fnd == points.end())
80                 {
81                     points.push_back(ImVec2(mouse_pos));
82                     colors.insert(colors.end(), { 1, 1, 1 });
83                 }
84             }
85
86             std::vector<std::size_t> perm(points.size());
87             std::iota(perm.begin(), perm.end(), 0);
88             std::sort(
89                 perm.begin(),
90                 perm.end(),
91                 [&](const std::size_t i, const std::size_t j)
92                 {
93                     return points[i].x < points[j].x;
94                 }
95             );
96
97             std::vector<ImVec2> old_alpha_control_pts(points.size());
98             std::copy(points.begin(), points.end(),
99                 ↪ old_alpha_control_pts.begin());
100
101             std::transform(
102                 perm.begin(),
103                 perm.end(),
104                 points.begin(),

```

```

102         [&](const std::size_t i) { return old_alpha_control_pts[i]; }
103     );
104
105     for (int i = 0; i < perm.size(); ++i)
106         while (perm[i] != i)
107         {
108             std::swap_ranges(colors.begin() + i * 3,
109                             colors.begin() + (i + 1) * 3,
110                             colors.begin() + perm[i] * 3);
111             for (int j = i + 1; j < perm.size(); ++j)
112                 if (perm[j] == i)
113                 {
114                     std::swap(perm[i], perm[j]);
115                     break;
116                 }
117         }
118
119     if (selected != 0 && selected != points.size() - 1)
120     {
121         auto fnd = std::find_if(
122             points.begin(), points.end(), [&](const ImVec2& p)
123             {
124                 const ImVec2 pt_pos = p * view_scale + view_offset;
125                 float dist = length(pt_pos -
126                                     ↪ ImVec2(clipped_mouse_pos));
127                 return dist <= point_radius;
128             });
129         selected = std::distance(points.begin(), fnd);
130         previous = selected;
131     }
132     else if (ImGui::IsMouseClicked(1))
133     {
134         selected = -1;
135         auto fnd = std::find_if(
136             points.begin(), points.end(), [&](const ImVec2& p)
137             {
138                 const ImVec2 pt_pos = p * view_scale + view_offset;
139                 float dist = length(pt_pos - ImVec2(clipped_mouse_pos));
140                 return dist <= point_radius;
141             });
142         size_t idx = fnd - points.begin();
143
144         if (fnd != points.end() && fnd != points.begin() &&
145             fnd != points.end() - 1)
146         {
147             colors.erase(colors.begin() + idx, colors.begin() + idx + 3);
148             points.erase(fnd);
149             previous = -1;
150         }
151     }
152     else
153         selected = -1;
154 }

```

```

155     else
156         selected = -1;
157
158
159     if (hist != nullptr)
160     {
161         float bin_w = canvas_size.x / nbBins;
162
163         for (int i = 0; i < nbBins; ++i)
164         {
165             const ImVec2 bin_pos = ImVec2(i * bin_w, 0);
166             const ImVec2 bin_size = ImVec2(bin_w, -hist[i] * canvas_size.y);
167
168             ImVec2 pt_min = bin_pos + view_offset;
169             ImVec2 pt_max = bin_pos + bin_size + view_offset;
170             draw_list->AddRectFilled(pt_min, pt_max,
171                                     ↪ ImGui::GetColorU32(ImGuiCol_PlotHistogram));
172         }
173
174         std::vector<ImVec2> polyline_pts;
175         for (const auto& pt : points) {
176             const ImVec2 pt_pos = pt * view_scale + view_offset;
177             polyline_pts.push_back(pt_pos);
178             draw_list->AddCircleFilled(pt_pos, point_radius, 0xFFFFFFFF);
179         }
180         draw_list->AddPolyline(
181             polyline_pts.data(), (int)polyline_pts.size(), 0xFFFFFFFF, false,
182             ↪ 2.f);
183
184         draw_list->PopClipRect();
185
186         updateColormap();
187         updateColormapTexture();
188     }
189
190 void VectorColorPicker::Load(const char* path)
191 {
192     readTF(path, points, colors);
193 }
194
195 void VectorColorPicker::save(const char* path)
196 {
197     saveTF(path, points, colors);
198 }
199
200 void VectorColorPicker::reset()
201 {
202     previous = -1;
203     selected = -1;
204     colors = { 1, 1, 1, 1, 1, 1, };
205     points = { ImVec2(0.f, 1.f), ImVec2(1.f, 1.f) };
206 }

```

```

207 void VectorColorPicker::updateColormapTexture()
208 {
209     GLint prev_tex_2d = 0;
210     glGetIntegerv(GL_TEXTURE_BINDING_2D, &prev_tex_2d);
211
212     if (colormapTex == NULL) {
213         glGenTextures(1, &colormapTex);
214         glBindTexture(GL_TEXTURE_2D, colormapTex);
215         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
216         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
217         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
218         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
219     }
220
221     glBindTexture(GL_TEXTURE_2D, colormapTex);
222     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB32F, COLORMAP_SIZE, 1, 0, GL_RGBA,
223         ↪ GL_FLOAT, colormap);
224
225     glBindTexture(GL_TEXTURE_2D, prev_tex_2d);
226 }
227 void VectorColorPicker::updateColormap()
228 {
229     int p_idx = 0;
230     for (int i = 0; i < COLORMAP_SIZE; ++i) {
231         float x = (float)i / COLORMAP_SIZE;
232         if (x > points[p_idx + 1].x)
233             ++p_idx;
234
235         ImVec2 first = points[p_idx], second = points[p_idx + 1];
236
237         float t = (x - first.x) / (second.x - first.x);
238
239         float r = (1.f - t) * colors[3 * p_idx] + t * colors[3 * (p_idx +
240             ↪ 1)];
241         float g = (1.f - t) * colors[3 * p_idx + 1] + t * colors[3 * (p_idx +
242             ↪ 1) + 1];
243         float b = (1.f - t) * colors[3 * p_idx + 2] + t * colors[3 * (p_idx +
244             ↪ 1) + 2];
245         float alpha = (1.f - t) * first.y + t * second.y;
246
247         colormap[i * 4] = ImClamp(r, 0.f, 1.f);
248         colormap[i * 4 + 1] = ImClamp(g, 0.f, 1.f);
249         colormap[i * 4 + 2] = ImClamp(b, 0.f, 1.f);
250         colormap[i * 4 + 3] = ImClamp(alpha, 0.f, 1.f);
251     }
252 }

```

Anexa 10. SettingsEditor.h

```

1 #pragma once
2
3 #include <vector>

```

```

4 #include <string>
5 #include <imgui/imgui.h>
6 #include <imgui/imgui_internal.h>
7 #include <imguiFD/ImGuiFileDialog.h>
8
9
10 class ISettings
11 {
12 public:
13     const char* name;
14
15     ISettings(const char* name) : name(name) {}
16
17     virtual void draw();
18 };
19
20 class SegmentationModelSettings: public ISettings
21 {
22 public:
23     int inputSize[3];
24     int patchSize[3];
25
26     SegmentationModelSettings() : ISettings("Segmentation Model")
27     {
28         memset(inputSize, 0, 3 * sizeof(int));
29         memset(patchSize, 0, 3 * sizeof(int));
30     }
31
32     virtual void draw();
33 };
34
35 class StyleSettings : public ISettings
36 {
37 public:
38     ImVec4 refColors[ImGuiCol_COUNT];
39     int styleIdx;
40
41     StyleSettings() : ISettings("Style")
42     {
43         styleIdx = -1;
44     }
45
46     void refFromStyle()
47     {
48         ImGuiStyle& style = ImGui::GetStyle();
49         for (int i = 0; i < ImGuiCol_COUNT; ++i)
50             refColors[i] = ImVec4(style.Colors[i]);
51     }
52
53     void styleFromRef()
54     {
55         ImGuiStyle& style = ImGui::GetStyle();
56         for (int i = 0; i < ImGuiCol_COUNT; ++i)
57             style.Colors[i] = ImVec4(refColors[i]);

```

```

58     }
59
60     virtual void draw();
61 };
62
63 class InputSettings : public ISettings
64 {
65 public:
66     float rotationSpeed;
67     float translationSpeed;
68
69     InputSettings() : ISettings("Input")
70     {
71         rotationSpeed = 1.0;
72         translationSpeed = 0.1;
73     }
74
75     virtual void draw();
76 };
77
78 class SettingsEditor
79 {
80 public:
81     void Initialize();
82     void draw(bool* enabled);
83
84     static SegmentationModelSettings segmentationSettings;
85     static StyleSettings styleSettings;
86     static InputSettings inputSettings;
87 private:
88     std::vector<ISettings*> settings;
89 };

```

Anexa 11. SettingsEditor.cpp

```

1  #include "SettingsEditor.h"
2
3  SegmentationModelSettings SettingsEditor::segmentationSettings;
4  StyleSettings SettingsEditor::styleSettings;
5  InputSettings SettingsEditor::inputSettings;
6
7  static void* Segmentation_ReadOpen(ImGuiContext*, ImGuiSettingsHandler*, const
    ↪ char* name)
8  {
9      return (void*)&SettingsEditor::segmentationSettings;
10 }
11
12 static void Segmentation_ReadLine(ImGuiContext*, ImGuiSettingsHandler*, void*
    ↪ entry, const char* line)
13 {
14     SegmentationModelSettings *settings = (SegmentationModelSettings*)entry;
15
16     int size[3];

```

```

17     if (sscanf(line, "InputSize=%i,%i,%i", &size[0], &size[1], &size[2]) == 3)
18         memcpy(settings->inputSize, size, 3 * sizeof(int));
19     else if (sscanf(line, "PatchSize=%i,%i,%i", &size[0], &size[1], &size[2])
20         ⇨ == 3)
21         memcpy(settings->patchSize, size, 3 * sizeof(int));
22 }
23 static void Segmentation_WriteAll(ImGuiContext* ctx, ImGuiSettingsHandler*
24     ⇨ handler, ImGuiTextBuffer* buf)
25 {
26     SegmentationModelSettings settings = SettingsEditor::segmentationSettings;
27     buf->appendf("[%s][ ]\n", handler->TypeName);
28     buf->appendf("InputSize=%d,%d,%d\n", settings.inputSize[0],
29     ⇨ settings.inputSize[1], settings.inputSize[2]);
30     buf->appendf("PatchSize=%d,%d,%d\n", settings.patchSize[0],
31     ⇨ settings.patchSize[1], settings.patchSize[2]);
32     buf->append("\n");
33 }
34 static void FileDialog_ReadLine(ImGuiContext*, ImGuiSettingsHandler*, void*
35     ⇨ entry, const char* line)
36 {
37     std::string* bookmarks = (std::string*)entry;
38     char buffer[256] = { 0 };
39     if (sscanf(line, "Bookmarks=%s", buffer) == 1)
40     {
41         bookmarks->assign(buffer);
42         ImGuiFileDialog::Instance()->DeserializeBookmarks(buffer);
43     }
44 }
45 static void FileDialog_WriteAll(ImGuiContext* ctx, ImGuiSettingsHandler*
46     ⇨ handler, ImGuiTextBuffer* buf)
47 {
48     buf->appendf("[%s][ ]\n", handler->TypeName);
49     buf->appendf("Bookmarks=%s\n",
50     ⇨ ImGuiFileDialog::Instance()->SerializeBookmarks().c_str());
51     buf->append("\n");
52 }
53 static void* Style_ReadOpen(ImGuiContext*, ImGuiSettingsHandler*, const char*
54     ⇨ name)
55 {
56     return (void*)&SettingsEditor::styleSettings;
57 }
58 static void Style_ReadLine(ImGuiContext*, ImGuiSettingsHandler*, void* entry,
59     ⇨ const char* line)
60 {
61     StyleSettings* settings = (StyleSettings*)entry;

```



```

62     int idx;
63     float c_r, c_g, c_b, c_a;
64
65     if(sscanf(line, "Colors[%d]=(%f, %f, %f, %f)", &idx, &c_r, &c_g, &c_b,
66         ↪ &c_a) == 5)
67         settings->refColors[idx] = ImVec4(c_r, c_g, c_b, c_a);
68 }
69
70 static void Style_WriteAll(ImGuiContext* ctx, ImGuiSettingsHandler* handler,
71     ↪ ImGuiTextBuffer* buf)
72 {
73     StyleSettings settings = SettingsEditor::styleSettings;
74
75     buf->appendf("[Colors][ ]\n", handler->TypeName);
76     for (int i = 0; i < ImGuiCol_COUNT; ++i)
77         buf->appendf("Colors[%d]=(%f, %f, %f, %f)\n", i,
78             ↪ settings.refColors[i].x, settings.refColors[i].y,
79             ↪ settings.refColors[i].z, settings.refColors[i].w);
80     buf->append("\n");
81 }
82
83 static void Style_ApplyAll(ImGuiContext* ctx, ImGuiSettingsHandler*)
84 {
85     SettingsEditor::styleSettings.styleFromRef();
86 }
87
88 static void* Input_ReadOpen(ImGuiContext*, ImGuiSettingsHandler*, const char*
89     ↪ name)
90 {
91     return (void*)&SettingsEditor::inputSettings;
92 }
93
94 static void Input_ReadLine(ImGuiContext*, ImGuiSettingsHandler*, void* entry,
95     ↪ const char* line)
96 {
97     InputSettings* settings = (InputSettings*)entry;
98
99     float val;
100
101     if (sscanf(line, "RotationSpeed=%f", &val) == 1)
102         settings->rotationSpeed = val;
103     else if (sscanf(line, "TranslationSpeed=%f", &val) == 1)
104         settings->translationSpeed = val;
105 }
106
107 static void Input_WriteAll(ImGuiContext* ctx, ImGuiSettingsHandler* handler,
108     ↪ ImGuiTextBuffer* buf)
109 {
110     InputSettings settings = SettingsEditor::inputSettings;
111
112     buf->appendf("[Input][ ]\n", handler->TypeName);
113     buf->appendf("RotationSpeed=%f\n", settings.rotationSpeed);
114     buf->appendf("TranslationSpeed=%f\n", settings.translationSpeed);
115     buf->append("\n");

```

```

109 }
110
111 void SettingsEditor::Initialize()
112 {
113     ImGuiContext& g = *GImGui;
114
115     ImGuiSettingsHandler segmentation_ini_handler;
116     segmentation_ini_handler.TypeName = "Segmentation";
117     segmentation_ini_handler.TypeHash = ImHashStr("Segmentation");
118     segmentation_ini_handler.ReadOpenFn = Segmentation_ReadOpen;
119     segmentation_ini_handler.ReadLineFn = Segmentation_ReadLine;
120     segmentation_ini_handler.WriteAllFn = Segmentation_WriteAll;
121     g.SettingsHandlers.push_back(segmentation_ini_handler);
122
123     ImGuiSettingsHandler file_dialog_ini_handler;
124     file_dialog_ini_handler.TypeName = "FileDialog";
125     file_dialog_ini_handler.TypeHash = ImHashStr("FileDialog");
126     file_dialog_ini_handler.ReadOpenFn = [](ImGuiContext*,
127     ↪ ImGuiSettingsHandler*, const char* name) { return (void*)new
128     ↪ std::string(); };
129     file_dialog_ini_handler.ReadLineFn = FileDialog_ReadLine;
130     file_dialog_ini_handler.WriteAllFn = FileDialog_WriteAll;
131     g.SettingsHandlers.push_back(file_dialog_ini_handler);
132
133     ImGuiSettingsHandler style_ini_handler;
134     style_ini_handler.TypeName = "Style";
135     style_ini_handler.TypeHash = ImHashStr("Style");
136     style_ini_handler.ReadOpenFn = Style_ReadOpen;
137     style_ini_handler.ReadLineFn = Style_ReadLine;
138     style_ini_handler.ApplyAllFn = Style_ApplyAll;
139     style_ini_handler.WriteAllFn = Style_WriteAll;
140     g.SettingsHandlers.push_back(style_ini_handler);
141
142     ImGuiSettingsHandler input_ini_handler;
143     input_ini_handler.TypeName = "Input";
144     input_ini_handler.TypeHash = ImHashStr("Input");
145     input_ini_handler.ReadOpenFn = Input_ReadOpen;
146     input_ini_handler.ReadLineFn = Input_ReadLine;
147     input_ini_handler.WriteAllFn = Input_WriteAll;
148     g.SettingsHandlers.push_back(input_ini_handler);
149
150     styleSettings.refFromStyle();
151
152     settings.push_back(&inputSettings);
153     settings.push_back(&segmentationSettings);
154     settings.push_back(&styleSettings);
155 }
156
157 void SettingsEditor::draw(bool *enabled)
158 {
159     ImGui::Begin("Settings", enabled);
160
161     static int selected = 0;
162     ImGui::BeginChild("Left", ImVec2(150, 0), true);

```

```

161     for (int i = 0; i < settings.size(); i++)
162     {
163         if (ImGui::Selectable(settings[i]->name, selected == i))
164             selected = i;
165     }
166     ImGui::EndChild();
167     ImGui::SameLine();
168
169     ImGui::BeginChild("right");
170
171     settings[selected]->draw();
172
173     ImGui::EndChild();
174
175     ImGui::End();
176 }
177
178 void ISettings::draw()
179 {
180     ImGui::Text(name);
181     ImGui::SameLine();
182     if(ImGui::Button("Save ALL"))
183         ImGui::SaveIniSettingsToDisk(ImGui::GetIO().IniFilename);
184
185     ImGui::Separator();
186 }
187
188 void SegmentationModelSettings::draw()
189 {
190     ISettings::draw();
191
192     ImGui::InputInt3("Input Size", inputSize);
193     ImGui::InputInt3("Patch Size", patchSize);
194 }
195
196 void StyleSettings::draw()
197 {
198     ISettings::draw();
199
200     ImGuiStyle& style = ImGui::GetStyle();
201
202     if (ImGui::Combo("Style Selector", &styleIdx, "Dark\0Light\0Classic\0"))
203     {
204         switch (styleIdx)
205         {
206             case 0:
207                 ImGui::StyleColorsDark();
208                 break;
209             case 1:
210                 ImGui::StyleColorsLight();
211                 break;
212             case 2:
213                 ImGui::StyleColorsClassic();
214                 break;

```

```

215     }
216
217     refFromStyle();
218 }
219
220 ImGui::Separator();
221
222 for (int i = 0; i < ImGuiCol_COUNT; ++i)
223 {
224     const char* name = ImGui::GetStyleColorName(i);
225
226     ImGui::PushID(name);
227
228     ImGui::ColorEdit4(name, (float*)&style.Colors[i],
229         ↪ ImGuiColorEditFlags_AlphaBar |
230         ↪ ImGuiColorEditFlags_AlphaPreviewHalf);
231
232     if (memcmp(&style.Colors[i], &refColors[i], sizeof(ImVec4)) != 0)
233     {
234         ImGui::SameLine(0.0f, style.ItemInnerSpacing.x);
235         if (ImGui::Button("Save"))
236             refColors[i] = style.Colors[i];
237         ImGui::SameLine(0.0f, style.ItemInnerSpacing.x);
238         if (ImGui::Button("Revert"))
239             style.Colors[i] = refColors[i];
240     }
241
242     ImGui::PopID();
243 }
244
245 ImGui::Separator();
246
247 if (ImGui::Button("Save"))
248     refFromStyle();
249 ImGui::SameLine();
250 if (ImGui::Button("Revert"))
251     styleFromRef();
252 }
253
254 void InputSettings::draw()
255 {
256     ISettings::draw();
257
258     float rot_degrees = rotationSpeed * (180.0 / IM_PI);
259     if (ImGui::SliderFloat("Rotation Speed (deg/sec)", &rot_degrees, 10.0f,
260         ↪ 90.0f, "%.1f"))
261         rotationSpeed = rot_degrees * (IM_PI / 180.0);
262
263     ImGui::SliderFloat("Translation Speed", &translationSpeed, 0.01, 0.5,
264         ↪ "%.2f");
265 }

```

Anexa 12. Loader.h

```

1 #pragma once
2 #include <vector>
3 #include <string>
4 #include <GL/glew.h>
5 #include <GL/freeglut.h>
6 #include <glm/glm.hpp>
7
8 #include "niftilib/nifti2_io.h"
9 #include <imgui/imgui.h>
10
11
12 enum class Format
13 {
14     NIFTI, UNKNOWN
15 };
16
17 Format getFileFormat(const char* path);
18
19 short* readVolume(const char* path, glm::ivec3 &size, glm::mat4 &xtoi);
20 void saveVolume(const char* path, short* data, const glm::ivec3 size);
21
22 void readTF(const char* path, std::vector<ImVec2>& alphaPoints,
23     ⇨ std::vector<float>& colors);
24 void saveTF(const char* path, std::vector<ImVec2> alphaPoints,
25     ⇨ std::vector<float> colors);

```

Anexa 13. Loader.cpp

```

1 #include "Loader.h"
2 #include <fstream>
3 #include <sstream>
4 #include <iostream>
5 #include <vector>
6 #include <algorithm>
7 #include <limits>
8
9
10 Format getFileFormat(const char* path)
11 {
12     char* s = new char[strlen(path) + 1];
13     memcpy(s, path, strlen(path) + 1);
14
15     char* last_tok = s;
16     for (char* tok = strchr(s, '.'); tok; last_tok = tok, tok = strchr(tok + 1,
17         ⇨ '.'));
18     last_tok++;
19
20     if (strcmp(last_tok, "nii") == 0 || strcmp(last_tok, "gz") == 0)
21         return Format::NIFTI;
22     else
23         return Format::UNKNOWN;
24 }

```

```

24
25 float mean(short* data, int size)
26 {
27     float sum = 0;
28     for (int i = 0; i < size; i++)
29         sum += data[i];
30     return sum / size;
31 }
32
33 short* readNIFTI(const char* path, glm::ivec3& size, glm::mat4& xtoi)
34 {
35     nifti_image* nim = nifti_image_read(path, 1);
36
37     size.x = nim->nx, size.y = nim->ny, size.z = nim->nz;
38
39     for (int i = 0; i < 4; ++i)
40         for (int j = 0; j < 4; ++j)
41             xtoi[i][j] = nim->qto_ijk.m[i][j];
42
43     short* data;
44
45     switch (nim->datatype)
46     {
47     case NIFTI_TYPE_INT16:
48         data = (short*)nim->data;
49         break;
50     default:
51         throw std::exception("Unexpected data type");
52     }
53
54     delete nim;
55
56     return data;
57 }
58
59 void writeNIFTI(const char* path, short* data, const glm::ivec3 size)
60 {
61     int64_t dims[] = { 3, size.x, size.y, size.z };
62     nifti_image* nim = nifti_make_new_nim(dims, NIFTI_TYPE_INT16, 0);
63     nim->data = data;
64     nim->fname = (char*)path;
65     nifti_image_write(nim);
66
67     delete nim;
68 }
69
70 short* readVolume(const char* path, glm::ivec3& size, glm::mat4& itox)
71 {
72     switch (getFileFormat(path))
73     {
74     case Format::NIFTI:
75         try
76         {
77             return readNIFTI(path, size, itox);

```

```

78     }
79     catch(std::exception e)
80     {
81         return nullptr;
82     }
83     default:
84         return nullptr;
85 }
86 }
87
88 void saveVolume(const char* path, short* data, const glm::ivec3 size)
89 {
90     switch (getFileFormat(path))
91     {
92     case Format::NIFTI:
93         return writeNIFTI(path, data, size);
94     default:
95         break;
96     }
97 }
98
99 void readTF(const char* path, std::vector<ImVec2>& alphaPoints,
100 ↪ std::vector<float>& colors)
101 {
102     std::ifstream file(path, std::ios::binary);
103     if (!file.good())
104     {
105         std::cout << "Error opening file " << path << std::endl;
106         return;
107     }
108
109     int size;
110     file >> size;
111
112     alphaPoints.resize(size);
113     colors.resize( 3 * size);
114
115     for (int i = 0; i < size; ++i)
116         file >> alphaPoints[i].x >> alphaPoints[i].y >> colors[3 * i] >>
117         ↪ colors[3 * i + 1] >> colors[3 * i + 2];
118
119     file.close();
120 }
121
122 void saveTF(const char* path, std::vector<ImVec2> alphaPoints,
123 ↪ std::vector<float> colors)
124 {
125     std::ofstream file(path, std::ios::binary);
126     if (!file.good())
127     {
128         std::cout << "Error opening file " << path << std::endl;
129         return;
130     }
131 }

```

```

129     file << alphaPoints.size() << std::endl;
130
131     for (int i = 0; i < alphaPoints.size(); ++i)
132     {
133         file << alphaPoints[i].x << " " << alphaPoints[i].y << " ";
134         file << colors[3 * i] << " " << colors[3 * i + 1] << " " << colors[3 *
            ↪ i + 2] << std::endl;
135     }
136
137     file.close();
138 }

```

Anexa 14. PytorchModel.h

```

1 #pragma once
2
3 #include <torch/script.h>
4 #include <torch/torch.h>
5
6 #include "SettingsEditor.h"
7
8 typedef unsigned short ushort;
9 typedef unsigned char uchar;
10
11 class PytorchModel
12 {
13 public:
14     PytorchModel();
15     void LoadModel(const char* path);
16     uchar* forward(short* data, int width, int height, int depth);
17
18     torch::Device device = torch::kCPU;
19     torch::jit::script::Module model;
20
21     bool isLoading = false;
22 };

```

Anexa 15. PytorchModel.cpp

```

1 #include "PytorchModel.h"
2
3
4 PytorchModel::PytorchModel()
5 {
6     if (torch::cuda::is_available())
7     {
8         std::cout << "CUDA is available" << std::endl;
9         std::cout << "CUDA DEVICE COUNT: " << torch::cuda::device_count() <<
            ↪ std::endl;
10         //device = torch::kCUDA;
11         //device.set_index(0);
12         //std::cout << "Using CUDA device: " << device.index() << std::endl;

```



```

13     }
14 }
15
16 void PytorchModel::loadModel(const char* path)
17 {
18     try {
19         isLoading = false;
20
21         model = torch::jit::load(path, device);
22
23         size_t dotPos = std::string(path).find_last_of(".");
24         std::string modelName = std::string(path).substr(0, dotPos);
25
26         isLoading = true;
27     }
28     catch (const c10::Error& e) {
29         std::cerr << "error loading the model " << std::endl;
30         std::cerr << e.msg() << std::endl;
31     }
32 }
33
34 uchar* PytorchModel::forward(short* data, int width, int height, int depth)
35 {
36     // iW, iH, iD = input image size
37     int inputSize[3];
38     // pW, pH, pD = output image size
39     int patchSize[3];
40     memcpy(inputSize, SettingsEditor::segmentationSettings.inputSize, 3 *
41         ↪ sizeof(int));
42     memcpy(patchSize, SettingsEditor::segmentationSettings.patchSize, 3 *
43         ↪ sizeof(int));
44
45     std::cout << device.index() << std::endl;
46
47     namespace F = torch::nn::functional;
48
49     size_t size = width * height * depth;
50
51     // [W, H, D]
52     torch::Tensor dataTensor = torch::from_blob(
53         data,
54         { 1, 1, depth, height, width },
55         torch::TensorOptions()
56         .dtype(torch::kInt16)
57         ).to(torch::kFloat32).to(device);
58
59     dataTensor = (dataTensor - dataTensor.mean()) / dataTensor.std();
60
61     std::cout << device.index() << std::endl;
62
63     dataTensor = dataTensor.transpose(2, 4);
64
65     std::cout << dataTensor.sizes() << std::endl;
66

```

```

65 // [iW, iH, iD]
66 dataTensor = F::interpolate(
67     dataTensor,
68     F::InterpolateFuncOptions()
69     .mode(torch::kNearest)
70     .size(std::vector<int64_t>{ inputSize[0], inputSize[1], inputSize[2] })
71 );
72
73 std::cout << dataTensor.sizes() << std::endl;
74
75 dataTensor = dataTensor.squeeze();
76
77 std::vector<torch::Tensor> patches{ dataTensor };
78 for (int i = 0; i < 3; ++i)
79 {
80     auto oldPatchesNb = std::distance(patches.begin(), patches.end());
81
82     for (int idx = 0; idx < oldPatchesNb; ++idx)
83     {
84         std::vector<torch::Tensor> patchesNew = torch::split(patches[idx],
85             ↪ patchSize[i], i);
86         patches.insert(patches.end(), patchesNew.begin(),
87             ↪ patchesNew.end());
88     }
89
90     patches.erase(patches.begin(), patches.begin() + oldPatchesNb);
91
92     std::vector<torch::Tensor> patchData;
93     for (torch::Tensor& patch : patches)
94         // [1, pW, pH, pD]
95         patchData.push_back(patch.unsqueeze(0));
96
97     // [B, 1, pW, pH, pD]
98     torch::Tensor nnData = torch::stack(patchData);
99
100     std::cout << nnData.sizes() << std::endl;
101
102     std::vector<torch::jit::IValue> nnInput{ nnData };
103
104     // expect [B, L, pW, pH, pD] ; L = nb labels
105     torch::Tensor outputTensor = model.forward(nnInput).toTensor();
106
107     std::cout << outputTensor.sizes() << std::endl;
108
109     patches.clear();
110     {
111         std::vector<torch::Tensor> patchesNew = torch::split(outputTensor, 1,
112             ↪ 0);
113         for (auto patch : patchesNew)
114             patches.push_back(patch.squeeze());
115     }
116
117     for (int i = 2; i >= 0; --i)
118     {

```

```

116     auto oldPatchesNb = std::distance(patches.begin(), patches.end());
117
118     for (int idx = 0; idx < oldPatchesNb; idx += inputSize[i] /
119           ↪ patchSize[i])
119     {
120         std::vector<torch::Tensor> seq;
121         seq.insert(seq.begin(), patches.begin() + idx, patches.begin() +
122               ↪ idx + inputSize[i] / patchSize[i]);
123         torch::Tensor patch = torch::cat(seq, i + 1);
124         patches.push_back(patch);
125     }
126     patches.erase(patches.begin(), patches.begin() + oldPatchesNb);
127 }
128
129 outputTensor = patches[0].unsqueeze(0);
130
131 std::cout << outputTensor.sizes() << std::endl;
132
133 outputTensor = outputTensor.transpose(2, 4);
134
135 // [L, W, H, D]
136 outputTensor = F::interpolate(
137     outputTensor,
138     F::InterpolateFuncOptions()
139     .mode(torch::kNearest)
140     .size(std::vector<int64_t>({ depth, height, width })))
141 );
142
143 std::cout << outputTensor.sizes() << std::endl;
144
145 int nbLabels = outputTensor.sizes().at(1);
146
147 outputTensor = torch::sigmoid(outputTensor) > 0.5;
148
149 outputTensor = torch::mul(outputTensor, torch::arange(1, nbLabels +
150       ↪ 1).to(device).reshape({ 1, nbLabels, 1, 1, 1 }));
151 std::cout << outputTensor.sizes() << std::endl;
152
153 // [W, H, D]
154 outputTensor = std::get<0>(torch::max(outputTensor, 1));
155 std::cout << outputTensor.sizes() << std::endl;
156
157 outputTensor = torch::flatten(outputTensor);
158 std::cout << outputTensor.sizes() << std::endl;
159
160 uchar* result = new uchar[size];
161 std::memcpy(result,
162     ↪ outputTensor.to(torch::kUInt8).contiguous().data_ptr<uchar>(), size *
163     ↪ sizeof(uchar));
164
165 return result;
166 }

```

Anexa 16. raycasting.vert

```

1 #version 400
2
3 in vec3 vp;
4
5 uniform mat4 MVP;
6
7 void main()
8 {
9     gl_Position = MVP * vec4(vp,1.0);
10 }

```

Anexa 17. raycasting.frag

```

1 #version 400
2
3 #define MAX_PASSES 1000
4 #define REFERENCE_SAMPLE_RATE 100
5
6 uniform sampler1D tf;
7 uniform sampler3D volumeTex;
8 uniform sampler3D gradsTex;
9 uniform sampler3D segTex;
10 uniform sampler1D segColors;
11
12 uniform mat4 modelMatrix;
13 uniform mat4 viewMatrix;
14
15 uniform vec2 screen;
16 uniform vec3 scale;
17 uniform mat4 xtoi;
18 out vec4 fragColor;
19
20 uniform vec3 origin;
21
22 uniform int sampleRate;
23 uniform float intensityCorrection;
24 uniform float exposure;
25 uniform float gamma;
26
27 uniform vec3 translation;
28
29 uniform vec3 bbLow;
30 uniform vec3 bbHigh;
31
32 struct Ray {
33     vec3 origin;
34     vec3 dir;
35 };
36
37 struct AABB {
38     vec3 min;

```

```

39     vec3 max;
40 };
41
42 struct Intersection{
43     vec3 p1;
44     vec3 p2;
45 };
46
47 vec3 transform(vec3 v, mat4 M)
48 {
49     return (vec4(v, 0) * M).xyz;
50 }
51
52 Intersection rayBBIntersection(Ray r, AABB aabb)
53 {
54     vec3 invR = 1.0 / r.dir;
55     vec3 tbot = invR * (aabb.min-r.origin);
56     vec3 ttop = invR * (aabb.max-r.origin);
57     vec3 tmin = min(ttop, tbot);
58     vec3 tmax = max(ttop, tbot);
59     vec2 t = max(tmin.xx, tmin.yz);
60     float t0 = max(t.x, t.y);
61     t = min(tmax.xx, tmax.yz);
62     float t1 = min(t.x, t.y);
63
64     return Intersection(transform(r.origin + r.dir * t0, xtoi) / scale + 0.5,
65                          transform(r.origin + r.dir * t1, xtoi) / scale + 0.5);
66 }
67
68 void main()
69 {
70     vec3 direction;
71     direction.xy = 2.0 * gl_FragCoord.xy / screen - 1.0;
72     direction.x *= screen.x / screen.y;
73     direction.z = - 1 / tan(3.14 / 4);
74     direction = transform(direction, viewMatrix);
75
76     Ray ray = Ray(origin + translation, direction);
77     AABB bb = AABB(bbLow, bbHigh);
78     Intersection intersection = rayBBIntersection(ray, bb);
79
80     float step = 1.0 / float(sampleRate);
81     float opacityCorrectionFactor = float(REFERENCE_SAMPLE_RATE) /
82     ↪ float(sampleRate);
83     float rayLength = length(intersection.p2 - intersection.p1);
84     vec3 deltaDir = step * normalize(intersection.p2 - intersection.p1);
85
86     float intensityScale = rayLength * step;
87
88     float len = 0.0;
89
90     vec3 boundaryLow = bbLow + vec3(0.5);
91     vec3 boundaryHigh = bbHigh + vec3(0.5);
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

```

```

92     vec3 pos = intersection.p1;
93     fragColor = vec4(0);
94
95     for(int i = 0;
96         i < MAX_PASSES && len < rayLength && fragColor.a < 1;
97         ++i, len += step, pos += deltaDir)
98     {
99         if(all(greaterThan(pos, boundaryLow)) && all(lessThan(pos,
100             ↪ boundaryHigh)))
101         {
102             float intensity = texture(volumeTex, pos).x;
103             float seg = texture(segTex, pos).x;
104             vec4 tfSample = texture(tf, intensity);
105
106             if (seg > 0)
107             {
108                 vec3 segColor = texture(segColors, seg).rgb;
109                 tfSample.rgb *= segColor;
110             }
111             else
112                 tfSample.a = 0;
113
114             intensity = max(0.0, 1 - pow((1 - intensity),
115                 ↪ opacityCorrectionFactor)) * intensityCorrection;
116             fragColor += (1.0 - fragColor.a) * vec4(tfSample.rgb, 1) *
117                 ↪ intensity * tfSample.a;
118         }
119     }
120
121     fragColor = min(vec4(1), fragColor);
122     fragColor.rgb = vec3(1.0) - exp(-fragColor.rgb * exposure);
123     fragColor = vec4(pow(fragColor.rgb, vec3(1.0 / gamma)), 1.0);
124 }

```

Anexa 18. run.py

```

1  import logging
2  import os
3  from typing import Any
4
5  from dotenv import Load_dotenv
6  from hydra.core.utils import JobReturn
7
8  Load_dotenv()
9
10 import shutil
11 from datetime import datetime
12
13 import google
14 import hydra
15 import torch
16 from hydra.experimental.callback import Callback
17 from hydra.utils import get_original_cwd

```

```

18 from omegaconf import DictConfig
19
20 import jobs
21
22 if torch.cuda.is_available():
23     torch.cuda.empty_cache()
24
25
26 class DataCallback(Callback):
27     def __init__(self) -> None:
28         ...
29
30     def on_run_start(self, config: DictConfig, **kwargs: Any) -> None:
31         if hasattr(google, "colab"):
32             getattr(google, "colab").drive.mount("/content/drive")
33
34     def on_run_end(
35         self, config: DictConfig, job_return: JobReturn, **kwargs: Any
36     ) -> None:
37         if hasattr(google, "colab") and "train_model" in config.jobs:
38             shutil.make_archive(
39                 f"/content/drive/MyDrive/{config.jobs.train_model.drive_save_
40                 ↪ e_path}/{datetime.now()}",
41                 "zip",
42                 os.environ["OUTPUT_PATH"],
43             )
44             getattr(google, "colab").drive.flush_and_unmount()
45
46 @hydra.main(config_path="conf", config_name="config")
47 def my_app(cfg: DictConfig) -> None:
48     log = logging.getLogger(__name__)
49     os.environ["OUTPUT_PATH"] = os.getcwd()
50     log.debug(f"output path: {os.getcwd()}")
51     os.chdir(get_original_cwd())
52     log.debug(f"cwd: {get_original_cwd()}")
53
54     dependencies = {}
55     for job_name, job_config in cfg["jobs"].items():
56         log.info(f"Starting stage === {job_name} ===")
57         try:
58             dependencies.update(
59                 getattr(jobs, job_config["fun"])(job_config, dependencies)
60             )
61         except Exception as e:
62             log.exception(f"Error in job {job_name}: {e}")
63
64
65 if __name__ == "__main__":
66     my_app()

```

Anexa 19. jobs.py

```

1 from data import get_data_loader
2 from models.loading import load_checkpoint
3 from models.train import train
4 from models.deploy import deploy_model
5 from models.test import test

```

Anexa 20. util.py

```

1 import random
2 import warnings
3
4 import torch
5 from monai.metrics.confusion_matrix import get_confusion_matrix
6 from torch.utils.data import DataLoader
7 from torchio import GridSampler
8
9
10 def import_tqdm():
11     try:
12         if get_ipython().__class__.__name__ == "ZMQInteractiveShell" or
13             ↪ "google.colab" in str(get_ipython()): # type: ignore
14             from tqdm.notebook import tqdm
15         else:
16             from tqdm import tqdm
17     except NameError:
18         from tqdm import tqdm
19
20     return tqdm
21
22 metrics_map = ["acc", "fpr", "fnr", "precision", "recall", "f1"]
23
24
25 def metric(y, y_pred):
26     acc = torch.sum(y == y_pred) / torch.numel(y)
27
28     confusion_matrix = get_confusion_matrix(y_pred, y)
29
30     tp, fp, tn, fn = torch.sum(confusion_matrix, (0, 1))
31
32     eps = 0.001
33     precision = tp / (torch.sum(y_pred) + eps)
34     recall = tp / (torch.sum(y) + eps)
35
36     with warnings.catch_warnings():
37         warnings.simplefilter("ignore")
38         f1 = 2 * (precision * recall) / (precision + recall + eps)
39
40     fpr = fp / (fp + tn + eps)
41     fnr = fn / (fn + tp + eps)
42

```



```

43     return acc, fpr, fnr, precision, recall, f1
44
45
46 def random_subject_from_loader(data_loader):
47     random_subject = data_loader.dataset.subjects_dataset[
48         random.randrange(0, len(data_loader.dataset.subjects_dataset))
49     ]
50     return (
51         GridSampler(
52             subject=random_subject,
53             ↪ patch_size=data_loader.dataset.sampler.patch_size
54         ),
55         random_subject["subject_id"],
56     )
57
58 def batches_from_sampler(sampler, batch_size):
59     loader = iter(DataLoader(sampler, batch_size=batch_size))
60     idx = 0
61     while idx < len(sampler):
62         yield next(loader), torch.Tensor(
63             sampler.locations[idx : idx + batch_size]
64         ).int()
65         idx += batch_size

```

Anexa 21. data/loading.py

```

1  import logging
2  import os
3  import random
4  from typing import List
5  from torchio import SubjectsDataset
6
7  import torchio.datasets
8  from omegaconf import open_dict
9  from omegaconf.dictconfig import DictConfig
10 from torch.utils.data import DataLoader
11 from torchio.data.queue import Queue
12 from torchio.data.sampler import GridSampler
13 from torchio.transforms import (
14     RandomMotion,
15     RandomBiasField,
16     RandomNoise,
17     RandomFlip,
18     Compose,
19 )
20
21 import numpy as np
22
23 import data.datasets as datasets
24 from .visualization import plot_subject
25 from .datasets.generic import Dataset
26

```

```

27
28 def __create_data_loader(
29     dataset: Dataset,
30     queue_max_length: int,
31     queue_samples_per_volume: int,
32     sampler,
33     batch_size,
34     verbose,
35 ) -> DataLoader:
36     queue = Queue(
37         subjects_dataset=dataset,
38         max_length=queue_max_length,
39         samples_per_volume=queue_samples_per_volume,
40         sampler=sampler,
41         verbose=verbose,
42         num_workers=2,
43     )
44
45     return DataLoader(
46         queue, batch_size=batch_size, shuffle=True, drop_last=True,
47         ↪ pin_memory=True
48     )
49
50 def split_dataset(dataset: Dataset, lengths: List[int]) -> List[Dataset]:
51     div_points = [0] + np.cumsum(lengths).tolist()
52
53     datasets = []
54     for i in range(len(div_points) - 1):
55         st = div_points[i]
56         end = div_points[i + 1]
57         datasets.append(SubjectsDataset(dataset._subjects[st:end]))
58
59     return datasets
60
61
62 def get_data_loader(cfg: DictConfig, _) -> dict:
63     log = logging.getLogger(__name__)
64
65     transform = Compose(
66         [
67             RandomMotion(),
68             RandomBiasField(),
69             RandomNoise(),
70             RandomFlip(axes=(0,)),
71         ]
72     )
73
74     log.info(f"Data Loader selected: {cfg['dataset']}")
75     try:
76         log.info("Attempting to use defined data loader")
77         dataset = getattr(datasets, cfg["dataset"])(cfg, transform)
78     except ImportError:

```

```

79     log.info("Not a defined data loader... Attempting to use torchio
    ↪     loader")
80     dataset = getattr(torchio.datasets, cfg["dataset"])(
81         root=cfg["base_path"], transform=transform, download=True
82     )
83
84     for subject in random.sample(dataset._subjects, cfg["plot_number"]):
85         plot_subject(
86             subject,
87             os.path.join(
88                 os.environ["OUTPUT_PATH"], cfg["save_plot_dir"],
89                 ↪     subject["subject_id"]
90             ),
91         )
92
93     sampler = GridSampler(patch_size=cfg["patch_size"])
94     samples_per_volume = len(sampler._compute_locations(dataset[0])) # type:
95     ↪     ignore
96
97     with open_dict(cfg):
98         cfg["size"] = dataset[0].spatial_shape
99
100     val_size = max(1, int(0.2 * len(dataset)))
101     test_set, train_set, val_set = split_dataset(
102         dataset, [21, len(dataset) - val_size - 21, val_size]
103     )
104
105     train_loader = __create_data_loader(
106         train_set,
107         queue_max_length=samples_per_volume * cfg["queue_length"],
108         queue_samples_per_volume=samples_per_volume,
109         sampler=sampler,
110         verbose=log.level > 0,
111         batch_size=cfg["batch"],
112     )
113
114     val_loader = __create_data_loader(
115         val_set,
116         queue_max_length=samples_per_volume * cfg["queue_length"],
117         queue_samples_per_volume=samples_per_volume,
118         sampler=sampler,
119         verbose=log.level > 0,
120         batch_size=cfg["batch"],
121     )
122
123     test_loader = __create_data_loader(
124         test_set,
125         queue_max_length=samples_per_volume * cfg["queue_length"],
126         queue_samples_per_volume=samples_per_volume,
127         sampler=sampler,
128         verbose=log.level > 0,
129         batch_size=cfg["batch"],
130     )

```

```

130     return {
131         "data_loader_train": train_loader,
132         "data_loader_val": val_loader,
133         "data_loader_test": test_loader,
134     }

```

Anexa 22. data/visualization.py

```

1  import logging
2  import os
3  from glob import glob
4
5  import imageio
6  import numpy as np
7  import torch
8  from tensorboardX import SummaryWriter
9  from torchio import GridAggregator, LabelMap, ScalarImage
10 from torchio.data.image import LabelMap
11 from torchio.data.subject import Subject
12 from torchio.visualization import import_mpl_plt
13 from util import batches_from_sampler, random_subject_from_loader
14
15
16 def squeeze_segmentation(seg):
17     squeezed_seg = torch.zeros_like(seg.data[0])
18     for idx, label in enumerate(seg.data):
19         seg_mask = squeezed_seg == 0
20         squeezed_seg[seg_mask] += (idx + 1) * label[seg_mask]
21     return squeezed_seg.unsqueeze(0)
22
23
24 def create_gifs(path_in: str, path_out: str):
25     filenames = [path for path in glob(f"{path_in}/*") if os.path.isfile(path)]
26     filenames.sort()
27
28     images = []
29     for filename in filenames:
30         images.append(imageio.imread(filename))
31     imageio.mimsave(path_out, images)
32
33
34 def plot_subject(subject: Subject, save_plot_path: str):
35     if save_plot_path:
36         os.makedirs(save_plot_path, exist_ok=True)
37
38     data_dict = {}
39     sx, sy, sz = subject.spatial_shape
40     sx, sy, sz = min(sx, sy, sz) / sx, min(sx, sy, sz) / sy, min(sx, sy, sz) /
41     ↪ sz
42     for name, image in subject.get_images_dict(intensity_only=False).items():
43         if isinstance(image, LabelMap):
44             data_dict[name] = LabelMap(
45                 tensor=squeeze_segmentation(image),

```

```

45         affine=np.eye(4) * np.array([sx, sy, sz, 1]),
46     )
47     else:
48         data_dict[name] = ScalarImage(
49             tensor=image.data, affine=np.eye(4) * np.array([sx, sy, sz, 1])
50         )
51
52 out_subject = Subject(data_dict)
53 out_subject.plot(reorient=False, show=True, figsize=(10, 10))
54
55 mpl, plt = import_mpl_plt()
56 backend_ = mpl.get_backend()
57
58 plt.ioff()
59 mpl.use("agg")
60 for x in range(max(out_subject.spatial_shape)):
61     out_subject.plot(
62         reorient=False,
63         indices=(
64             min(x, out_subject.spatial_shape[0] - 1),
65             min(x, out_subject.spatial_shape[1] - 1),
66             min(x, out_subject.spatial_shape[2] - 1),
67         ),
68         output_path=f"{save_plot_path}/{x:03d}.png",
69         show=False,
70         figsize=(10, 10),
71     )
72     plt.close("all")
73 plt.ion()
74 mpl.use(backend_)
75
76 create_gifs(
77     save_plot_path,
78     ↪ f"{save_plot_path}/{os.path.basename(save_plot_path)}.gif"
79 )
80
81 def plot_aggregated_image(
82     writer: SummaryWriter,
83     epoch: int,
84     model: torch.nn.Module,
85     data_loader: torch.utils.data.DataLoader, # type: ignore
86     device: torch.device,
87     save_path: str,
88 ):
89     log = logging.getLogger(__name__)
90
91     sampler, subject_id = random_subject_from_loader(data_loader)
92     aggregator_x = GridAggregator(sampler)
93     aggregator_y = GridAggregator(sampler)
94     aggregator_y_pred = GridAggregator(sampler)
95     for batch, locations in batches_from_sampler(sampler,
96         ↪ data_loader.batch_size):
97         x: torch.Tensor = batch["image"]["data"]

```

```

97     aggregator_x.add_batch(x, locations)
98     y: torch.Tensor = batch["seg"]["data"]
99     aggregator_y.add_batch(y, locations)
100
101     logits = model(x.to(device))
102     y_pred = (torch.sigmoid(logits) > 0.5).float()
103     aggregator_y_pred.add_batch(y_pred, locations)
104
105     whole_x = aggregator_x.get_output_tensor()
106     whole_y = aggregator_y.get_output_tensor()
107     whole_y_pred = aggregator_y_pred.get_output_tensor()
108
109     plot_subject(
110         Subject(
111             image=ScalarImage(tensor=whole_x),
112             true_seg=LabelMap(tensor=whole_y),
113             pred_seg=LabelMap(tensor=whole_y_pred),
114         ),
115         f"{save_path}/{epoch}-{subject_id}",
116     )
117
118
119 def train_visualizations(
120     writer: SummaryWriter,
121     epoch: int,
122     model: torch.nn.Module,
123     data_loader: torch.utils.data.DataLoader, # type: ignore
124     device: torch.device,
125     save_path: str,
126 ):
127     log = logging.getLogger(__name__)
128     log.info(f"visualisations for epoch: {epoch}")
129
130     try:
131         plot_aggregated_image(writer, epoch, model, data_loader, device,
132                               ↪ save_path)
133     except Exception as e:
134         log.exception(f"Failed to plot vizualization: {e}")

```

Anexa 23. data/generic.py

```

1 import logging
2 import os
3 from typing import Dict, List, Union
4
5 import multitasking
6 import torch
7 from omegaconf.dictconfig import DictConfig
8 from torchio import LabelMap, ScalarImage, Subject
9 from torchio.data.dataset import SubjectsDataset
10 from torchio.transforms import OneHot, Resize, Transform
11 from util import import_tqdm
12

```

```

13 tqdm = import_tqdm()
14
15
16 class Dataset(SubjectsDataset):
17     Log = logging.getLogger(__name__)
18
19     def __init__(self, cfg: DictConfig, transform: Transform, **kwargs):
20         self.cfg = cfg
21
22         super().__init__(
23             subjects=self._get_subjects_list(), transform=transform, **kwargs
24         )
25
26     def _create_data_map(self) -> Union[Dict[str, List[str]], Dict[str, str]]:
27         ...
28
29     def _get_subject_id(self, path: str) -> str:
30         ...
31
32     def _get_subjects_list(self) -> List[Subject]:
33         if not os.path.exists(self.cfg["cache_path"]):
34             self._generate_cache()
35             multitasking.wait_for_tasks()
36
37         subjects = []
38         for pt_name in os.listdir(self.cfg["cache_path"]):
39             pt_data = torch.load(os.path.join(self.cfg["cache_path"], pt_name))
40
41             subjects.append(
42                 Subject(
43                     subject_id=pt_data["subject_id"],
44                     image=ScalarImage(tensor=pt_data["image"]),
45                     seg=LabelMap(tensor=pt_data["seg"]),
46                 )
47             )
48         return subjects
49
50     @multitasking.task
51     def _generate_cache(self):
52         os.makedirs(self.cfg["cache_path"])
53
54         data_map = self._create_data_map()
55
56         for path, label_paths in tqdm(
57             data_map.items(),
58             total=len(data_map),
59             position=0,
60             leave=False,
61             desc="Caching pooled data",
62         ):
63             subject_id = self._get_subject_id(path)
64
65             image = ScalarImage(path, check_nans=True)
66             label_map = LabelMap(label_paths, check_nans=True)

```

```

67         if not isinstance(label_map, list):
68             one_hot = OneHot(self.cfg["num_classes"] + 1)
69             label_map = one_hot(label_map)
70             label_map = LabelMap(tensor=label_map.tensor[1:]) # type:
              ↪ ignore
71
72         image.load()
73         label_map.load()
74
75         image, label_map = self._resize(image, label_map)
76
77         torch.save(
78             {"subject_id": subject_id, "image": image.data, "seg":
              ↪ label_map.data},
79             os.path.join(self.cfg["cache_path"], f"{subject_id}.pt"),
80         )
81
82     def _resize(self, image: ScalarImage, label_map: LabelMap):
83         res_subject = Resize(self.cfg["desired_size"]).apply_transform(
84             Subject(
85                 image=ScalarImage(tensor=image.data),
86                 seg=LabelMap(tensor=label_map.data),
87             )
88         )
89         return res_subject["image"], res_subject["seg"]

```

Anexa 24. data/ct_org.py

```

1  import os
2  import re
3  import zipfile
4  from glob import glob
5  from typing import Dict
6
7  from omegaconf.dictconfig import DictConfig
8  from torchio.transforms.transform import Transform
9
10 from .generic import Dataset
11
12
13 class CTORGDataset(Dataset):
14     def __init__(self, cfg: DictConfig, transform: Transform, **kwargs):
15         if not os.path.isdir(cfg["base_path"]):
16             os.makedirs(cfg["base_path"])
17
18         with zipfile.ZipFile(f'/content/drive/{cfg["drive_path"]}', "r") as
              ↪ zip_ref:
19             zip_ref.extractall(f'{cfg["base_path"]}/../')
20
21         super().__init__(cfg, transform, **kwargs)
22
23     def _create_data_map(self) -> Dict[str, str]:
24         scan_pattern: str = self.cfg["base_path"] + self.cfg["scan_pattern"]

```



```

25     Dataset.log.debug(f"scan_pattern={scan_pattern}")
26
27     return {
28         image_path: os.path.join(
29             os.path.dirname(image_path),
30             os.path.basename(image_path).replace("volume", "Labels"),
31         )
32         for image_path in sorted(glob(scan_pattern, recursive=True),
33             ↪ key=lambda x: int(re.search(r"\d+", x).group())) # type:
34             ↪ ignore
35     }
36
37 def _get_subject_id(self, path: str) -> str:
38     return os.path.basename(path).split(".")[0]

```

Anexa 25. model/train.py

```

1  import logging
2  import os
3
4  import torch
5  from data.visualization import train_visualizations
6  from monai.metrics.cumulative_average import CumulativeAverage
7  from omegaconf import DictConfig
8  from pytorch_model_summary import summary
9  from tensorboardX import SummaryWriter
10 from torch.nn import BCEWithLogitsLoss
11 from torch.optim import Optimizer
12 from torch.optim.lr_scheduler import StepLR
13 from util import import_tqdm, metric, metrics_map
14
15 from .nets import get_net
16
17 tqdm = import_tqdm()
18
19 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
20
21
22 def _train_epoch(
23     data_loader: torch.utils.data.DataLoader, # type: ignore
24     model: torch.nn.Module,
25     criterion: torch.nn.Module,
26     optimizer: Optimizer,
27 ) -> CumulativeAverage:
28     model.train()
29     epoch_metrics = CumulativeAverage()
30
31     for batch in tqdm(
32         data_loader,
33         total=len(data_loader),
34         position=1,
35         leave=False,
36         desc="Train Steps",

```

```

37     ):
38         x: torch.Tensor = batch["image"]["data"]
39         y: torch.Tensor = batch["seg"]["data"]
40
41         optimizer.zero_grad()
42
43         logits = model(x.to(device).float())
44         loss: torch.Tensor = criterion(logits, y.to(torch.float).to(device))
45
46         loss.backward()
47         # torch.nn.utils.clip_grad_value_(model.parameters(), clip_value=1.0)
48         optimizer.step()
49
50         y_pred = (torch.sigmoid(logits) > 0.5).float()
51         epoch_metrics.append([loss.item(), *metric(y.cpu(), y_pred.cpu())])
52
53     return epoch_metrics
54
55
56 def _get_metrics_for_model(
57     data_loader: torch.utils.data.DataLoader, # type: ignore
58     model: torch.nn.Module,
59     criterion: torch.nn.Module,
60     step: str,
61 ):
62     epoch_metrics = CumulativeAverage()
63
64     for batch in tqdm(
65         data_loader,
66         total=len(data_loader),
67         position=1,
68         leave=False,
69         desc=f"{step} Steps",
70     ):
71         x: torch.Tensor = batch["image"]["data"]
72         y: torch.Tensor = batch["seg"]["data"]
73
74         logits = model(x.to(device).float())
75         loss: torch.Tensor = criterion(logits, y.to(torch.float).to(device))
76
77         y_pred = (torch.sigmoid(logits) > 0.5).float()
78         epoch_metrics.append([loss.item(), *metric(y.cpu(), y_pred.cpu())])
79
80     return epoch_metrics
81
82
83 def _validate_model(
84     data_loader: torch.utils.data.DataLoader, # type: ignore
85     model: torch.nn.Module,
86     criterion: torch.nn.Module,
87 ):
88     model.eval()
89     return _get_metrics_for_model(data_loader, model, criterion, "Validation")
90

```

```

91
92 def _test_model(
93     data_loader: torch.utils.data.DataLoader, # type: ignore
94     model: torch.nn.Module,
95     criterion: torch.nn.Module,
96 ):
97     model.eval()
98     return _get_metrics_for_model(data_loader, model, criterion, "Test")
99
100
101 def train(cfg: DictConfig, dependencies: dict) -> dict:
102     log = logging.getLogger(__name__)
103
104     if "data_loader_train" not in dependencies or "data_loader_val" not in
105         ⇨ dependencies:
106         raise Exception("Missing required dependencies: data loaders")
107
108     train_loader, val_loader = (
109         dependencies["data_loader_train"],
110         dependencies["data_loader_val"],
111     )
112
113     log.info(f"Using device: {device}")
114
115     torch.autograd.set_detect_anomaly(cfg["anomaly_detection"]) # type: ignore
116
117     if "model" in dependencies:
118         model = dependencies["model"]
119         dependencies["model"].train(True)
120     else:
121         model = get_net(cfg).to(device)
122
123     try:
124         log.info(f"Getting summary of {model.__class__.__name__}...")
125         log.info(
126             summary(
127                 model,
128                 torch.zeros((1, 1, 64, 32, 64)).to(device),
129                 show_input=True,
130                 show_hierarchical=True,
131                 show_parent_layers=True,
132             )
133         )
134     except Exception as e:
135         log.warn(f"Failed to get summary of {model.__class__.__name__}: {e}")
136
137     optimizer: Optimizer = torch.optim.Adam(model.parameters(),
138         ⇨ lr=cfg["init_lr"])
139     scheduler = StepLR(
140         optimizer, step_size=cfg["scheduler_step_size"],
141         ⇨ gamma=cfg["scheduler_gamma"]
142     )
143
144     criterion = BCEWithLogitsLoss().to(device)

```

```

142
143     writer = SummaryWriter(
144         os.path.join(os.environ["OUTPUT_PATH"], cfg["tb_output_dir"])
145     )
146
147     if device.type == "cuda":
148         log.info(
149             f"{model.__class__.__name__} Memory Usage on
150             ↪ {torch.cuda.get_device_name()}:"
151         )
152         model_memory_allocated = round(torch.cuda.memory_allocated() /
153             ↪ 1024**3, 1)
154         log.info(f"\tAllocated: {model_memory_allocated} GB")
155
156     best_val_loss = float("inf")
157     early_stopping_patience_counter = 0
158     early_stopping = False
159
160     start_epoch = 1
161
162     if "checkpoint" in dependencies:
163         checkpoint = dependencies["checkpoint"]
164         optimizer.load_state_dict(checkpoint["optim"])
165         scheduler.load_state_dict(checkpoint["scheduler"])
166         start_epoch = int(checkpoint["epoch"]) + 1
167
168     tqdm_obj = tqdm(
169         range(start_epoch, start_epoch + cfg["total_epochs"]),
170         initial=start_epoch,
171         total=cfg["total_epochs"],
172         position=0,
173         leave=True,
174         desc="Epoch",
175     )
176     for epoch in tqdm_obj:
177         epoch_metrics = _train_epoch(train_loader, model, criterion, optimizer)
178         scheduler.step()
179
180         average_metrics = epoch_metrics.aggregate()
181         tqdm_obj.set_postfix(
182             {
183                 name: average_metrics[idx].item()
184                 for idx, name in enumerate(["loss"] + metrics_map)
185             }
186         )
187
188         if epoch % cfg["metrics_every"] == 0:
189             log.info(f"Training metrics for epoch {epoch}")
190             for idx, name in enumerate(["loss"] + metrics_map):
191                 writer.add_scalar(f"training/{name}", average_metrics[idx], epoch)
192                 if epoch % cfg["metrics_every"] == 0:
193                     log.info(f"\ttraining/{name}: {average_metrics[idx]}")
194
195         if epoch % cfg["metrics_every"] == 0:

```

```

194     train_visualizations(
195         writer,
196         epoch,
197         model,
198         val_loader,
199         device,
200         f'{os.environ["OUTPUT_PATH"]}/{cfg["plots_output_path"]}',
201     )
202
203 def save_model(name):
204     checkpoints_path = os.path.join(
205         os.environ["OUTPUT_PATH"], cfg["checkpoints_dir"]
206     )
207     if not os.path.exists(checkpoints_path):
208         os.makedirs(checkpoints_path)
209     torch.save(
210         {
211             "epoch": epoch,
212             "model": model.state_dict(),
213             "optim": optimizer.state_dict(),
214             "scheduler": scheduler.state_dict(),
215         },
216         os.path.join(checkpoints_path, name),
217     )
218
219 if epoch % cfg["validate_every"] == 0:
220     val_metrics = _validate_model(val_loader, model, criterion)
221     average_metrics = val_metrics.aggregate()
222
223     average_val_loss = average_metrics[0]
224     if average_val_loss < best_val_loss:
225         best_val_loss = average_val_loss
226         early_stopping_patience_counter = 0
227         save_model(f"checkpoint_best.pt")
228     elif (
229         average_val_loss > best_val_loss + 1e-3
230         and epoch > cfg["early_stop_ignore"]
231     ):
232         Log.warn(
233             f"Validation loss is increasing. Early stopping in
234             ↪ {cfg['early_stop_patience']} -
235             ↪ early_stopping_patience_counter."
236         )
237         early_stopping_patience_counter += 1
238
239     if early_stopping_patience_counter > cfg["early_stop_patience"]:
240         early_stopping = True
241
242     Log.info(f"Validation metrics for epoch {epoch}")
243     for idx, name in enumerate(["loss"] + metrics_map):
244         writer.add_scalar(f"validation/{name}", average_metrics[idx],
245             ↪ epoch)
246         Log.info(f"\tvalidation/{name}: {average_metrics[idx]}")

```

```

245         # save_model(cfg['latest_checkpoint_file'])
246
247         if epoch % cfg["save_every"] == 0 or early_stopping:
248             save_model(f"checkpoint_{epoch:04d}.pt")
249
250         if early_stopping:
251             break
252
253     writer.flush()
254     writer.close()
255
256     return {"model": model}

```

Anexa 26. model/test.py

```

1  import logging
2
3  from omegaconf import DictConfig
4  from torch.nn import BCEWithLogitsLoss
5  from util import metrics_map
6
7  from .train import _test_model
8
9
10 def test(cfg: DictConfig, dependencies: dict) -> dict:
11     log = logging.getLogger(__name__)
12
13     criterion = BCEWithLogitsLoss()
14
15     test_metrics = _test_model(
16         dependencies["data_loader_test"], dependencies["model"], criterion
17     )
18     average_metrics = test_metrics.aggregate()
19     for idx, name in enumerate(["loss"] + metrics_map):
20         log.info(f"test/{name}: {average_metrics[idx]}")
21
22     return {}

```

Anexa 27. model/loading.py

```

1  import torch
2  from omegaconf import DictConfig
3
4  from .nets import get_net
5  from .train import device
6
7
8  def load_checkpoint(cfg: DictConfig, _: dict) -> dict:
9     checkpoint = torch.load(cfg["load_checkpoint_path"])
10
11     model = get_net(cfg).to(device)
12     model.load_state_dict(checkpoint["model"])

```

```

13
14     return {"model": model, "checkpoint": checkpoint}

```

Anexa 28. model/deploy.py

```

1  import os
2
3  import torch
4  from omegaconf import DictConfig
5
6  from models.train import device
7
8
9  def deploy_model(cfg: DictConfig, dependencies: dict) -> dict:
10     if "model" not in dependencies:
11         raise Exception("Cannot deploy! Missing model...")
12
13     example = torch.rand(1, 1, 192, 96, 192).to(device)
14
15     dependencies["model"].train(False)
16     script_module = torch.jit.trace(dependencies["model"], example) # type:
17     ↪ ignore
18
19     script_module.save(f"{os.environ['OUTPUT_PATH']}/torchscript_module_mode_
20     ↪ l.pt")
21
22     return {}

```

Anexa 29. model/__init__.py

```

1  import torch
2  from monai.networks.nets.unet import UNet
3  from monai.networks.nets.unetr import UNETR
4  from omegaconf import DictConfig
5
6  from .unet3d import UNet3D
7
8
9  def get_net(cfg: DictConfig) -> torch.nn.Module:
10     if cfg["net_name"] == "UNet":
11         return UNet(
12             spatial_dims=3,
13             in_channels=cfg.in_channels,
14             out_channels=cfg.out_channels,
15             channels=cfg.channels,
16             strides=cfg.strides,
17             dropout=cfg.dropout,
18         )
19     elif cfg["net_name"] == "MyUNet":
20         return UNet3D(cfg)
21     elif cfg["net_name"] == "UNETR":
22         return UNETR(

```

```

23         in_channels=cfg.in_channels,
24         out_channels=cfg.out_channels,
25         img_size=cfg.img_size,
26         dropout_rate=cfg.dropout,
27         res_block=cfg.res_block,
28     )
29
30     raise NotImplementedError

```

Anexa 30. model/unet3d.py

```

1  from collections import OrderedDict
2
3  import torch
4  import torch.nn as nn
5  from omegaconf import DictConfig
6
7
8  class UNet3D(nn.Module):
9      def __init__(self, cfg: DictConfig):
10         super(UNet3D, self).__init__()
11
12         features: int = int(cfg["init_features"])
13         self.nb_blocks = cfg["nb_blocks"]
14
15         self.encoders = nn.ModuleList(
16             [
17                 UNet3D._block(cfg["in_channels"], features, name="enc_1"),
18                 nn.MaxPool3d(kernel_size=2, stride=2),
19             ]
20         )
21         for idx in range(1, self.nb_blocks):
22             self.encoders += [
23                 UNet3D._block(features, features * 2, name=f"enc_{idx + 1}"),
24                 nn.MaxPool3d(kernel_size=2, stride=2),
25             ]
26             features *= 2
27
28         self.bottleneck = UNet3D._block(features, features * 2,
29             ↪ name="bottleneck")
30         features *= 2
31
32         self.decoders = nn.ModuleList()
33         for idx in range(self.nb_blocks, 0, -1):
34             self.decoders += [
35                 nn.ConvTranspose3d(features, features // 2, kernel_size=2,
36                     ↪ stride=2),
37                 UNet3D._block(features, features // 2, name=f"dec_{idx}"),
38             ]
39             features //= 2
40
41         self.output_conv = nn.Conv3d(

```



```

40         in_channels=features, out_channels=cfg["out_channels"],
41         ↪ kernel_size=1
42     )
43
44     def forward(self, x: torch.Tensor):
45         last_output: torch.Tensor = x
46         enc_outputs: list[torch.Tensor] = []
47
48         for (encoder, pool) in zip(self.encoders[0::2], self.encoders[1::2]):
49             ↪ # type: ignore
50             enc_outputs.append(encoder(last_output))
51             last_output = pool(enc_outputs[-1])
52
53         last_output = self.bottleneck(last_output)
54
55         for (upconv, decoder) in zip(self.decoders[0::2],
56             ↪ self.decoders[1::2]): # type: ignore
57             last_output = decoder(
58                 torch.cat((upconv(last_output), enc_outputs[-1]), dim=1)
59             )
60             del enc_outputs[-1]
61
62         return self.output_conv(last_output)
63
64     @staticmethod
65     def _block(in_channels: int, features: int, name: str):
66         return nn.Sequential(
67             OrderedDict(
68                 [
69                     (
70                         f"{name}_conv1",
71                         nn.Conv3d(
72                             in_channels=in_channels,
73                             out_channels=features,
74                             kernel_size=3,
75                             padding=1,
76                             bias=True,
77                         ),
78                     ),
79                     (f"{name}_Dropout1", nn.Dropout()),
80                     (f"{name}_relu1", nn.ReLU(inplace=True)),
81                     (
82                         f"{name}_conv2",
83                         nn.Conv3d(
84                             in_channels=features,
85                             out_channels=features,
86                             kernel_size=3,
87                             padding=1,
88                             bias=True,
89                         ),
90                     ),
91                     (f"{name}_Dropout2", nn.Dropout()),
92                     (f"{name}_relu2", nn.ReLU(inplace=True)),
93                 ]
94             )

```

```

91         )
92     )

```

Anexa 31. config/config.yaml

```

1 defaults:
2   - jobs:
3     - load_data
4     # - load_model
5     - train_model
6     - test_model
7     - deploy
8   - /hydra/callbacks:
9     - data_callback
10  - override hydra/job_logging: colorlog
11  - override hydra/hydra_logging: colorlog

```

Anexa 32. config/base_data.yaml

```

1 queue_length: 5
2 base_path: ???
3 batch: ???
4 patch_size: ???
5 plot_number: 1
6 save_plot_dir: "metrics/data_samples"

```

Anexa 33. config/ct_org.yaml

```

1 defaults:
2   - base_data
3
4 drive_path: 'MyDrive/CT-ORG.zip'
5 base_path: "${oc.env:TOP_PROJECT_PATH}/pytorch/extern/data/CT-ORG/"
6 scan_pattern: "volume-*.nii.gz"
7 cache_path: "${jobs.load_data.base_path}/cache/"
8 desired_size: [192,96,192]
9 patch_size: [192,96,192]
10 num_classes: 6
11 batch: 10
12 dataset: CTORGDataset

```

Anexa 34. config/base_model.yaml

```

1 drive_save_path: 'licenta/colab/outputs/'
2 checkpoints_dir: 'checkpoints'
3 validation_plots_dir: 'plots'
4 tb_output_dir: 'metrics'
5 plots_output_path: "metrics/plots/"
6 latest_checkpoint_file: 'checkpoint_latest.pt'
7 save_every: 20

```

```

8 metrics_every: 20
9 validate_every: 1
10 early_stop_ignore: 100
11 early_stop_patience: 10
12 anomaly_detection: False

```

Anexa 35. config/UNet3D.yaml

```

1 defaults:
2   - base_model
3
4 in_channels: 1
5 out_channels: 6
6 init_lr: 0.02
7 scheduler_step_size: 20
8 total_epochs: 1000
9 scheduler_gamma: 0.9
10 channels: [32, 64, 128, 256, 512]
11 strides: [2, 2, 2, 2]
12 dropout: 0.5
13 net_name: UNet

```

Anexa 36. config/MyUNet3D.yaml

```

1 defaults:
2   - base_model
3
4 in_channels: 1
5 out_channels: 6
6 init_features: 32
7 nb_blocks: 4
8 init_lr: 0.02
9 scheduler_step_size: 20
10 total_epochs: 1000
11 scheduler_gamma: 0.9
12 net_name: MyUNet

```

Anexa 37. config/load_data.yaml

```

1 # @package jobs.Load_data
2 defaults:
3   - base_job
4   - /data/ct_org@_here_
5
6 fun: get_data_loader

```

Anexa 38. config/load_model.yaml

```

1 # @package jobs.Load_model
2 defaults:

```

```
3   - base_job
4   - /hparams/UNet3D@_here_
5
6 fun: load_checkpoint
7 load_checkpoint_path: ???
```

Anexa 39. config/train_model.yaml

```
1 # @package jobs.train_model
2 defaults:
3   - base_job
4   - /hparams/UNet3D@_here_
5
6 fun: train
```

Anexa 40. config/test_model.yaml

```
1 # @package jobs.test_model
2 defaults:
3   - base_job
4
5 fun: test
```

Anexa 41. config/deploy.yaml

```
1 # @package jobs.deploy
2 defaults:
3   - base_job
4
5 fun: deploy_model
```