Vizualizarea şi procesarea în timp real a volumelor mari de date medicale

Andrei Simion

Universitatea POLITEHNICA din Bucureşti

andrei.simion90@gmail.com

Florica Moldoveanu, Victor Asavei, Alin Moldoveanu

Universitatea POLITEHNICA din Bucureşti

{ florica.moldoveanu, victor.asavei, alin.moldoveanu }@cs.pub.ro

REZUMAT

Această lucrare prezintă un studiu de caz asupra metodelor de vizualizare şi manipulare în timp real a datelor medicale utilizând o variantă optimizată a algoritmului ray casting pentru volume implementată folosind tehnici de programare GPU (Graphics Processing Unit) în OpenGL.

Cuvinte cheie

Date medicale, vizualizare volumetrică, volume ray casting, empty space leaping, sparse voxel octree

Clasificare ACM

H5.2. Information interfaces and presentation

INTRODUCERE

Vizualizarea şi manipularea în timp real a datelor medicale reprezintă una din direcţiile de mare interes actual în domeniul graficii asistate pe calculator.

Vizualizarea seturilor de date obţinute din diverse scanări medicale (CT - computer tomograf, RMN - rezonanţă magnetică nucleară) sunt destinate în principal medicilor, dar îşi pot găsi rolul şi în procesul educaţional.

Tehnicile de vizualizare a datelor volumetrice medicale sunt mai ales utile în cazul procedurilor medicale mai complicate, pentru o înţelegere mai bună a anatomiei sau a structurilor de interes.

TEHNICI DE VIZUALIZARE A VOLUMELOR

Există două tipuri de tehnici pentru vizualizarea volumelor: tehnici directe şi indirecte.

Tehnicile indirecte presupun mai întâi extragerea modelului geometric al suprafeţelor şi ulterior redarea acestuia. Printre cei mai folosiţi algoritmi din această categorie regăsim *marching cubes* şi *marching tetrahedra* [2].

Tehnicile directe, spre deosebire de cele indirecte, nu necesită o geometrie intermediară, dar de obicei costul computaţional al acestor metode este mai crescut pentru a obţine aceleaşi rezultate vizuale. Cei mai cunoscuţi algoritmi din această categorie sunt *Volume Ray Casting*, *Shear Warp* (o variantă optimizată a *Ray Casting*-ului) şi *Splatting-ul* [2].

Lucrarea de faţă prezintă o metodă optimizată de a reda şi manipula datele medicale în timp real folosind algoritmul *Volume Ray Casting* clasic şi câteva metode de optimizare pentru a îmbunătăţi performanţele acestuia.

ALGORITMUL VOLUME RAY CASTING

Algoritmul *Volume Ray Casting* este un algoritm în spaţiul imaginii, ceea ce înseamnă că redarea porneşte de la fiecare pixel al imaginii care va afişa volumul.

Ideea de bază a acestui algoritm este intersectarea corpului volumului cu o rază (având ca punct de plecare poziţia observatorului) pe care sunt eşantionate puncte echidistante şi compunerea valorilor tuturor eşantioanelor de-a lungul razei pentru a determina culoarea şi opacitatea pixelului final.

Volumele ce sunt eşantionate sunt reprezentate de obicei prin mai multe felii 2D ce alcătuiesc un bloc compact de date asemănător unui grid 3D ale cărui elemente pot fi accesate folosind trei coordonate (uzual x, y, z). Un element din acest grid se numeşte *voxel* (*volume element*) şi valoarea acestuia, cel puţin în cazul prelucrărilor de date medicale, reprezintă de obicei densitatea ţesuturilor care au fost scanate [1].

Algoritmul *Volume Ray Casting* are patru paşi de bază [1]:

1) *Intersecţia razei cu volumul*: pentru fiecare pixel din imagine o rază este trimisă prin volumul ce este de obicei închis într-o primitivă geometrică (de cele mai multe ori un cub) pentru a putea calcula mai uşor punctele de început şi oprire ale razei.
2) *Eşantionarea*: volumul este eşantionat în puncte echidistante de-a lungul razei; datorită faptului că acesta poate să nu fie aliniat cu raza, punctele eşantionate sunt interpolate triliniar folosind voxelii cei mai apropiaţi.
3) *Iluminarea*: se calculează iluminarea în fiecare eşantion folosind modelul de iluminare ales.
4) *Compunerea*: punctele eşantionate de-a lungul razei sunt compuse pentru a da culoarea finală a pixelului.

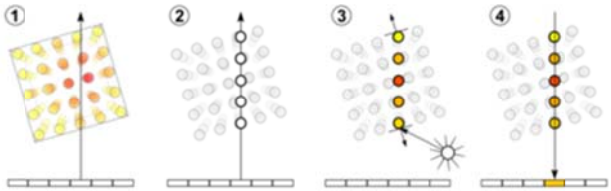

Figura 1. Paşii algoritmului Volume Ray Casting [1]

Funcţii de transfer

Cu ajutorul funcţiilor de transfer putem realiza o clasificare simplă a valorilor voxelilor. Folosindu-ne de o funcţie de transfer putem să asociem valorii voxelului atât o valoare de culoare cât şi o valoare de opacitate. Aplicând funcţii de transfer diferite, putem să deosebim diverse ţesuturi între ele [1].

APLICAȚIE PRACTICĂ

Pornind de la modelul teoretic al algoritmului *Volume Ray Casting* care a fost prezentat mai sus, a fost realizata o aplicație care permite utilizatorului atât să vizualizeze volumele medicale, cât şi să interacționeze cu acestea într-o manieră facilă, în timp real. Folosind doar mouse-ul, acesta poate să navigheze în jurul modelului pentru a-l examina din orice direcție sau unghi.

De asemenea, folosind un set de slidere care sunt poziționate pe ecran, funcția de transfer poate fi uşor modificată pentru a afişa doar densitățile ce sunt de interes.

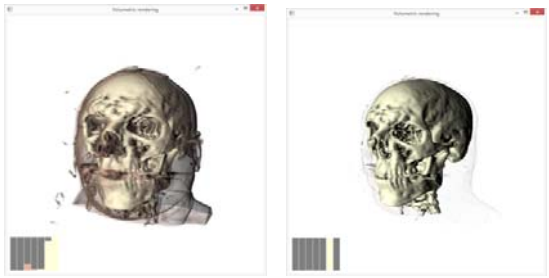

Figura 2. Interfața aplicație şi funcția de transfer modificată

Dacă este nevoie, se poate efectua o secțiune în volum (presupune vizualizarea elementelor care se află doar de o anumită parte a planului definit de utilizator) pentru a putea observa mai atent elementele ce sunt în mod normal obturate.

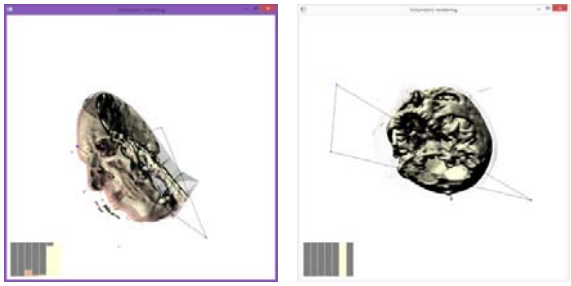

Figura 3. Plan de tăiere

Implementare

Aplicația reprezintă o implementare algoritmul *Volume Ray Casting* şi combină mai multe imagini 2D medicale obținute de-a lungul a mai multor plane de scanare.

Implementarea a fost realizată folosind limbajele de programare C++, OpenGL Shading Language (GLSL) şi se foloseşte de puterea de calcul paralel a GPU-urilor moderne pentru a reda volumele în timp real la o calitate ridicată.

Datele volumetrice pot fi achiziționate prin mai multe metode cum ar fi RMN-ul sau CT-ul. Datele de intrare folosite pentru testele efectuate în cadrul acestei lucrări au fost descărcate de la [5], au extensia **.raw* şi conțin doar valori pentru densitățile țesuturilor în format binar (în cazul seturilor de date pe 8 biți, valorile densității sunt în intervalul [0, 255]). Acestea sunt încărcate în memorie, prelucrate şi trimise mai departe către GPU.

Intersecția razei cu volumul

Pentru fiecare pixel al imaginii ce urmează să fie redată este generată o rază care intersectează volumul. Pentru a determina direcția razei, mai întâi este încadrat volumul într-o primitivă geometrică numită *bounding box* (în cazul de față, un cub).

OpenGL foloseşte pentru culori modelul RGBA (*red, green, blue, alpha*), fiecare culoare are patru componente numerice. În loc de a folosi componentele pentru valori ale culorii, acestea pot fi folosite drept coordonate într-un spațiu euclidian şi valorile să reprezinte pozițiile actuale ale volumului. Astfel putem determina direcția razei utilizând diferența normalizată dintre poziția observatorului şi fiecare poziție salvată în cub.

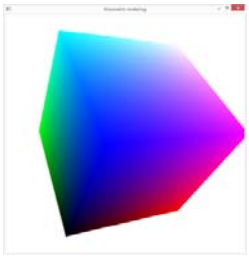

Figura 4. Culorile cubului reprezintă de fapt pozițiile acestuia în spațiul euclidian

Un alt avantaj pe care îl avem folosind acest model este faptul că valorile pot fi salvate într-o textură şi reutilizate. Acest lucru este cu atât mai util atunci când avem nevoie să facem *multi-pass rendering* şi imaginea redată este salvată într-o textură. Procesul este detaliat în secțiunile următoare.

Eşantionarea de-a lungul razei

Eşantionarea de-a lungul razei are loc la distanțe fixe. Datorită faptului că există o posibilitate ca raza să nu fie aliniată cu volumul ce trebuie redat, eşantionarea poate să aibă loc între voxeli. Pentru a rezolva această problemă, valoarea eşantionului este obținută prin interpolare triliniară între voxelii vecini.

În cazul de față, interpolarea este realizată automat de OpenGL. Atunci când creăm textura 3D ce va ține volumul în memoria GPU, o configuram astfel încât interpolarea să fie liniara. Accesul la o valoare din textura se face folosind valori în intervalul [0, 1], pentru fiecare componentă (axele x, y sau z). API-ul OpenGL realizează automat interpolarea atunci când accesăm textura.

Pentru ca rezultatele redării să fie exacte şi să nu fie pierdute anumite detalii din volum datorită interpolării, este necesar ca numărul punctelor de eşantionare să fie dublu față de numărul maxim de voxeli care pot fi eşantionați de-a lungul razei.

Iluminarea eşantioanelor

Pentru ca rezultatul final să fie similar cu modelul fizic real ce a fost scanat, trebuie ca scena să fie iluminată. Există mai multe modele în literatura de specialitate ce aproximează iluminarea şi pentru lucrarea de față s-a folosit bine cunoscutul model Phong.

Pentru a putea calcula iluminarea folosind acest model, este necesară normala la suprafață. Pentru calculul acesteia se foloseşte gradientul câmpului scalar ce exprimă variația lui în punctele de eşantionare. S-a folosit metoda diferențelor finite – aproximarea prin diferențe centrate.

Mai jos avem formulele corespunzătoare acestei metode, *f* reprezintă valoarea câmpului scalar în punctul determinat de coordonatele x, y şi z.

$$\text{grad}(f)_{x,y,z} = [\partial f(x,y,z)/\partial x, \partial f(x,y,z)/\partial y, \partial f(x,y,z)/\partial z]$$

$$\text{grad}(f(x_i, y_j, z_k)) = [\ \tfrac{1}{2}(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)),$$
$$\tfrac{1}{2}(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k))$$
$$\tfrac{1}{2}(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1}))\ ]$$

Vectorul obţinut în urma acestui calcul indica direcţia de variaţie maximă a câmpului scalar în punctul (x, y, z) al volumului [2].

A fost utilizată o funcţie de transfer pentru a defini corespondenţa dintre valoarea câmpului scalar şi culoarea / opacitatea eşantionului. Funcţia de transfer poate fi modificată în timp real de către utilizator şi este aproximată folosind *cubic splines*. Pentru fiecare valoare a densităţii ţesutului avem câte o valoare de culoare, respectiv opacitate pe care o putem folosi.

Figura 5a. Volum redat cu iluminare

Figura 5b. Volum redat folosind doar funcţia de transfer

Aceste valori sunt transmise către GPU folosind o textură 1D ce conţine vectori cu patru componente - trei componente pentru culoare şi o componentă pentru opacitate.

Calculul gradientului are loc atunci când aplicaţia porneşte şi rezultatul este salvat într-un fişier pentru fiecare model folosit deoarece este un calcul costisitor computaţional şi nu este necesară realizarea acestuia decât o singură dată.

Compunerea rezultatelor

Eşantioanele sunt compuse pentru a determina culoarea pixelului final ce va fi afişat.

S-a folosit metoda *front-to-back blending* pentru compunere – acest lucru semnifică faptul că eşantioanele sunt compuse plecând de la poziţia utilizatorului [2].

Compunerea s-a făcut după următoarele formule :

$$C_{dst} = C_{dst} + (1 - A_{dst}) * C_{src}$$
$$A_{dst} = A_{dst} + (1 - A_{dst}) * A_{src}$$

C – valoarea culorii A – valoarea opacităţii

Dst – eşantion destinaţie Src – eşantion sursa

OPTIMIZĂRI

Toate testele realizate în continuare au rulat pe o maşină hardware ce dispunea de un procesor quad-core Intel Core i7 2600k, un GPU Nvidia Geforce GTX 560 TI 1GB, memoria totală RAM a sistemului fiind 8 GB.

Rezultatele obţinute sunt pentru un set de date de intrare de dimensinuea 256x256x256, cu valori ale densităţii ţesuturilor pe 8 biţi.

Pentru primele rulări ale aplicaţiei s-au obţinut în jur de 20 FPS (*frames per second*). Deşi acest FPS permite vizualizarea şi manipularea a întreg setului de date în timp real, pentru dimensiuni mai mari algoritmul necesită îmbunătăţiri.

Adaptive ray termination

În prima implementare a algoritmului, raza care traversa volumul realiza eşantionarea în toate punctele din volum. Acest fapt presupune că eşantionarea se realiza inclusiv pentru punctele care erau obturate de voxelii precedenţi lor.

Deoarece eşantionarea se realiza pas cu pas (după ce se făcea eşantionarea într-un punct, raza era înaintată cu pasul standard), ca primă optimizare se poate opri înaintarea razei atunci când se atinge un anumit prag de opacitate (în cazul curent valoarea pragului este 0.95, opacitatea având valori în intervalul [0, 1]).

Îmbunătăţirile aduse de această metodă variază în mare parte în funcţie de volumul ce este vizualizat şi densităţile voxelilor. În cazul unui volum ce are multe densităţi de valori mari, îmbunătăţirea este una semnificativă. În cazul volumelor ce au multe eşantioane cu densităţi mai mici, timpul până la care se atinge pragul de opacitate creşte şi astfel performanţa scade.

Multipass rendering

Multipass rendering presupune că redarea imaginii se realizează în mai mulţi paşi. În primii paşi redarea se realizează într-un *buffer off-screen*, de obicei într-o textură, urmând ca în paşii ulteriori rezultatele să fie folosite ca input pentru realizarea unor tehnici mai complicate. *Multipass rendering* se foloseşte pentru efecte grafice mai complexe cum ar fi redarea umbrelor sau *deffered shading*.

Această tehnică poate fi folosită (şi este folosită de cele mai multe ori) şi cu algoritmul *Volume Ray Casting*. În varianta iniţială a implementării, raza care străbătea volumul avea o lungime fixă şi anume distanţă maximă care ar putea fi eşantionata în volum (pentru un volum care este încadrat într-un cub distanţă maximă este reprezentată de diagonala cubului).

Algoritmul a fost modificat în modul următor :

- Primul pas – este redată partea din faţă a cubului pentru a determina poziţiile de start ale razelor; rezultatul este salvat sub forma unei texturi
- Al doilea pas – este redată partea din spate a *bounding box*-ului pentru a determina poziţiile de stop ale razelor
- Pasul final – folosind poziţiile de start şi stop pentru raze, este implementat algoritmul *Volume Ray Casting*

Rezultatele optimizării sunt destul de relevante, FPS-ul aplicaţiei creşte de la 25-27 până la 58-60, există o creştere a performanţei cu aproape 100%.

Empty space leaping

Una dintre cele mai mari probleme ce apare la algoritmul *Volume Ray Casting* este utilizarea inutilă a ciclurilor de calcul şi a magistralei I/O datorită eşantionării punctelor în care opacitatea este 0. O creştere a performanţei semnificativă ar putea fi obţinută dacă raza ce intersectează volumul ar putea să ignore spaţiile irelevante.

O metodă prin care s-ar putea realiza acest lucru ar fi divizarea *bounding box*-ului în multe bucăţi mai mici decât volumul iniţial şi redarea doar a celor ce au o opacitate mai mare decât 0.

Pentru a realiza acest lucru, construim un *voxel octree* [3]. Pornim de la volumul original ce este cuprins intre [0, 0, 0] şi [1, 1, 1] şi îl împărţim recursiv în opt subdiviziuni egale până când latura cubului actual este de n ori mai mică decât latura cubului iniţial. Pentru fiecare din cuburile procesate se verifică dacă aceasta are opacitatea tuturor voxelilor conţinuţi mai mare decât 0. Dacă nu, el este ignorat şi nu mai este redat pe ecran.

Algoritmul este tot unul din categoria *multipass rendering*, redarea se face în mai mulţi paşi, cu excepţia că în primul şi al doilea pas se vor afişa toate feţele faţă / spate ale cuburilor rezultate. Un dezavantaj pentru această metodă este faptul că toate calculele ce ţin de subdivizarea *bounding box*-ului trebuie refăcute atunci când funcţia de transfer suferă modificări deoarece aceasta asociază alte valori de opacitate pentru voxeli şi modifică topologia redată.

Aplicaţia permite subdivizarea în timp real a *bounding box*-ului. Calculul se face pe un fir de execuţie separat pentru a nu bloca interfaţa grafică, permiţând utilizatorului să beneficieze de o experienţă continuă. Când rezultatele sunt gata, acestea sunt trimise către GPU şi imaginea redată este modificată corespunzător.

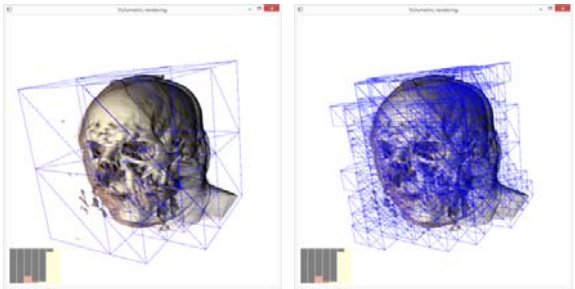

Figura 6. Subdivizarea recursivă a bounding box-ului

O altă îmbunătăţire corelată cu *empty space leaping* ce a fost implementată este împărţirea subdiviziunilor sub forma unui *sparse voxel octree* [3]. Acest lucru presupune că în cazul în care toţi cei opt copii ai unui nod al arborelui sunt vizibili (au opacitatea mai mare decât 0) este redat nodul părinte în locul copiilor. Acest lucru ajuta la reducerea numărului de forme geometrice necesare pentru a încapsula tot volumul şi, implicit, aduce un câştig de performanţă.

Această modificare a avut un impact destul de important din punctul de vedere al performanţei, în anumite cazuri dublând sau triplând numărul de cadre pe secundă.

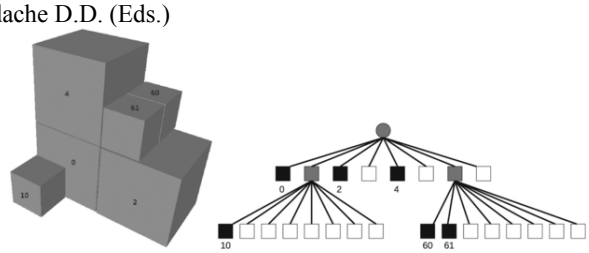

Figura 7. Sparse voxel octree [3]

CERCETĂRI ULTERIOARE

O îmbunătăţire viitoare va fi de a eficientiza accesul la memorie pe GPU – momentan setul de date este încărcat liniar în memoria GPU. Atunci când raza străbate volumul sunt şanse mari ca aceasta să nu acceseze voxeli ce sunt vecini în memorie dacă aşezarea este una liniară. Am putea îmbunătăţi eficienta cache-ului păstrând datele într-un format block – swizzled.

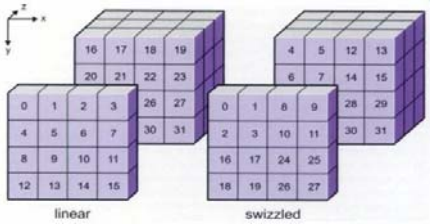

Figura 8. Layout liniar vs swizzled pentru memoria GPU [1]

O altă îmbunătăţire va fi utilizarea *Deffered shading* în cadrul calculelor de iluminare pentru un pas ulterior.

CONCLUZII

Volume Ray Casting este un algoritm eficient pentru vizualizarea şi procesarea în timp real a datelor medicale.

Comparativ cu alţi algoritmi, necesită o putere de procesare mai mare, dar folosit împreună cu metodele de optimizare prezentate în acest articol (adaptive ray termination, multipass rendering şi, în special, empty space leaping) permite vizualizarea în timp real la un număr de cadre pe secundă acceptabil.

Metoda propusă în acest articol, împreună cu optimizările realizate, duce la o aplicaţie de timp real care poate fi foarte utilă în medicină.

REFERINŢE

1. Klaus Engel , Markus Hadwiger , Joe Kniss , Christof Rezk-Salama, *Real-Time Volume Graphics*, ISBN 1-56881-266-3, CRC Press, 2006
2. Florica Moldoveanu, Victor Asavei, *Cursul şi laboratorul Tehnici de vizualizare a datelor volumetrice şi animaţie pe calculator*, http://cs.curs.pub.ro/2013/course/view.php?id=80, 2014
3. Samuli Laine, Tero Karras, *Efficient Sparse Voxel Octrees*, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games , 2010
4. Paul-Corneliu Herghelegiu, *Analiza şi vizualizarea seturilor de date biomedicale*, http://www.ace.tuiasi.ro/index.php?file=6556, 2011
5. Seturi de date folosite: http://www.volvis.org