SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
STAVEBNÁ FAKULTA

Študentská vedecká konferencia
Akademický rok 2014/2015

Sledovanie pohybu buniek počas embryogenézy živočíchov

Meno a priezvisko študenta, ročník, odbor: Natália Heliová, 4. ročník, aplikovaná matematika

Vedúci práce: Ing. Róbert Špir

Katedra / Ústav: Katedra matematiky a deskriptívnej geometrie (SvF)

Bratislava 21. apríla 2015

Obsah

Abstrakt

V tejto práci sme sa zamerali na rozširovanie funkcionality softvéru pre vizualizáciu trajektórií pohybu a delenia buniek v 3D priestore a čase počas embyogenézy živočíchov. Dáta, s ktorými náš program pracuje, sú získavané z časových sérií 3D biomedicínskych snímkov, ktoré predstavujú prvé hodiny bunkového vývoja embrya rybky Danio rerio. Pomocou rôznych numerických metód sú z týchto dát extrahované trajektórie pohybu buniek. Pre potrebu efektívneho zobrazovania bol vytvorený softvér, ktorý využíva grafickú kartu počítača a tým zrýchľuje samotnú vizualizáciu. Vizualizačný program je implementovaný v jazyku C# pomocou vývojového prostredia Visual Studio 2013 a jeho pôvodná verzia poskytovala funkcionalitu pre vykreslenie jednotlivých časových krokov vývoja embryo, spätných i dopredných trajektórií buniek a možnosť farebného zvýraznenia buniek podľa rýchlosti alebo smeru, delenia buniek a zobrazenie vlastnej farby definovanej vo vstupnom súbore. Softvér bol doplnený o ďalšiu funkcionalitu a v súčasnosti umožňuje spolu s numerickými dátami zobrazovať rezy reprezentujúce originálne mikroskopické snímky v jednotlivých rovinách, sledovanie pohybu jednej bunky a vizualizáciu strednej trajektórie viacerých bunkových populácií reprezentujúcich napríklad jednotlivé orgány.

Abstract

In this work we have focused on expanding the functionality of the software for the trajectories visualization representing cell movement and their division in space-time during the early development of organisms. The input data used by our program is obtained from a time series of 3D biomedical images representing the first hours of Danio rerio (zebrafish) embryo development. The cell trajectories are extracted from this data using various numerical methods. To make this visualization effective, we used GPU of the computer to speed up the rendering. The visualization software is written in C# programming language using Visual Studio 2013 IDE and the previous version of the software allowed us to display individual time steps of embryo development, the forward and backward cell trajectories, options to paint the cells according to their speed or direction of the movement, highlight cell divisions and use custom colouring using colour defined in input file. The software has been supplemented with additional functionality and allows us to display slices that represent original microscopy images in individual planes. Furthermore the software allows us to track the movement of every single cell and visualize the central trajectory of several cell populations representing, for instance, individual organs.

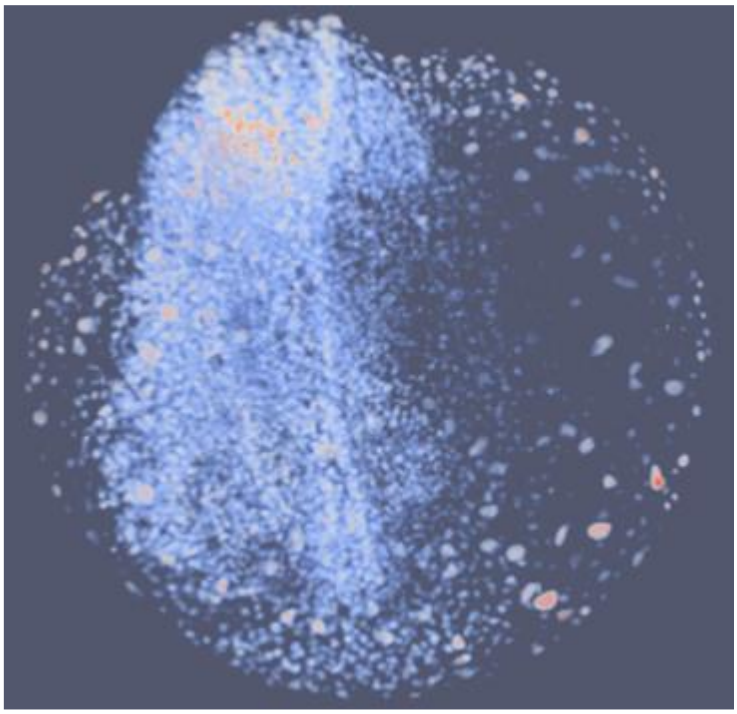
Úvod

V tejto práci sme vychádzali z bakalárskej práce na tému 4D vizualizácia pohybu biologických štruktúr, ktorej úlohou bolo vytvoriť softvér slúžiaci na efektívnu vizualizáciu výstupných dát numerického algoritmu [1] pre získanie trajektórií pohybu buniek v 3D priestore a čase.

Táto metóda vychádza z reálnych dát získaných zo 4D biomedicínskych snímiek (časová postupnosť3D snímiek), ktoré reprezentujú prvé hodiny embryogenézy živočícha Danio rerio (známa ako zebrička). Tento druh rybky je intenzívne študovaný vzhľadom na podobnosť s človekom v mnohých aspektoch a pre svoju priehľadnosť pre laserový konfokálny mikroskop, z ktorého snímky pochádzajú. Jedným z faktorov, od ktorého závisí kvalita týchto snímiek je rýchlosť ich zaznamenávania. Čím dlhší je časový krok medzi jednotlivými snímkami, tým je ich kvalita vyššia a tým lepšie pre segmentáciu, čiže hľadanie objektov v snímkach. Na druhej strane, dlhý časový krok nie je vhodný pre sledovanie pohybu buniek, pretože medzi jednotlivými snímkami môže nastať veľký pohyb buniek a tým môže prísť ku strate napríklad vzťahu materskej bunky s dcérskymi. Na obrázku 1 vidíme poslednú z 3D snímok bunkových jadier (12 hodín po oplodnení), z ktorých sú získavané trajektórie pohybu buniek pomocou viacerých krokov integrovaného pracovného postupu [2]. Ako prvý krok je filtrácia pomocou geodesic mean curvature flow (GMCF), ktorá slúži na odstránenie šumu nachádzajúceho sa v mikroskopických snímkach. Nasleduje samotné získavanie bunkových identifikátorov pomocou flux-based level set center detection (FBLSCD) predstavujúcich pozíciu bunkových jadier v priestore. Pomocou týchto identifikátorov je vytvorená 4D segmentácia reprezentujúca časopriestorovú stromovú štruktúru, z ktorej sú následne extrahované jednotlivé trajektórie spätným trasovaním špeciálneho potenciálového poľa skonštruovaného v tejto segmentácii.

S takto získanými dátami pracoval vizualizačný softvér vytvorený pre potreby bakalárskej práce, ktorý umožňoval vykresľovať jednotlivé časové kroky vývoja embrya spolu so spätnými i doprednými trajektóriami buniek. Podľa voľby užívateľa ponúkal program aj možnosť farebného zvýraznenia buniek podľa ich rýchlosti alebo smeru pohybu a zobrazenie vlastnej farby definovanej vo vstupnom súbore. Softvér bol doplnený o ďalšiu funkcionalitu a okrem vyššie spomenutého momentálne umožňuje sledovanie pohybu jednej bunky a vizualizáciu strednej trajektórie viacerých bunkových populácií. Navyše spolu s numerickými dátami je možné zobraziť rezy originálnych mikroskopických snímiek v jednotlivých rovinách.

Samotná vizualizácia je dôležitá pre interpretáciu výsledkov a poskytuje nám lepšiu predstavu o získaných dátach.



Obr. 1: Snímka bunkových jadier (12 hodín po oplodnení)

1 Implementácia

Náš vizualizačný softvér je implementovaný v jazyku C#, čo je objektový programovací jazyk, odvodený od jazyka C++, ale jeho tvorcovia boli inšpirovaní aj jazykom Java. Bol navrhnutý a implementovaný firmou Microsoft pre platformu .NET. Zo všetkých vyšších programovacích jazykov, v ktorých je možné vyvíjať programy pre túto platformu, poskytuje najväčšie možnosti, pretože bol vyvíjaný spoločne s ňou [3]. Vývojový tím na čele s dánskym softvérovým inžinierom Andersom Hejlsbergom si pri tvorbe jazyka C# dával za úlohu vytvoriť moderný a jednoduchý programovací jazyk, ktorý by bol schopný poskytovať silnú typovú kontrolu, kontrolu ohraničenia polí, detekciu pokusov na využitie neinicializovaných premenných a automatickú správu pamäte.

Jazyk C# je jednoduchý, elegantný, typovo bezpečný a umožňuje vytvárať bezpečné a robustné aplikácie, ktoré bežia na rozhraní .NET - vývojová platforma pri vytváraní aplikácií pre systémy Windows. Ako objektovo orientovaný jazyk podporuje všetky princípy objektovo orientovaného programovania (OOP), ktoré poskytuje mnoho výhod (prehľadnosť kódu, jeho znovupoužiteľnosť) a jeho filozofia je postavená na usporiadaní reálneho sveta

Pôvodný program využíval knižnicu SlimDX, ktorá umožňuje vytvárať DirectX aplikácie pre .NET. Nakoľko SlimDX v súčasnosti nie je ďalej vyvíjaný, bola táto knižnica nahradená open-source projektom SharpDX. Ide o aplikačné rozhranie pre DirectX, ktoré je priamo generované z hlavičkových súborov DirectX SDK [4]. Microsoft DirectX je sada knižníc, ktoré poskytujú API (application programming interfaces) pre jednoduchšiu tvorbu multimediálnych a herných aplikácií a umožňuje priame ovládanie hardwaru. V našom programe konkrétne využívame Microsoft Direct3D rozhranie, ktoré nám ponúka funkcie pre pohodlnú prácu s 3D grafikou. Toto rozhranie využíva grafickú kartu počítača a je nezávislé na jej type. Direct3D podporuje a vykresľuje rôzne typy primitív, v našom prípade Line List a Triangle List.

Informácie o vrcholoch a ich indexoch sú uložené v špeciálnych štruktúrach nazývajúcich sa Vertex a Index buffer.

2 Vstupné dáta

Na obrázku č. 2 vidíme časť súboru, s ktorým softvér pracuje.

```
0;0;0;0;0;3;0;0
109680;1;-55;67;42;3;1;-1;-16776961
109681;2;18;-95;37;3;2;-1;-65536
109682;3;55;42;60;3;3;-1;-16744448
109683;4;105;104;50;3;4;-1;-1
109684;5;31;147;50;3;5;-1;-1
109685;6;109;78;52;3;6;-1;-1
109686;7;-26;-76;47;3;7;-1;-1
109687;8;44;-76;34;3;8;-1;-65536
109688;9;-83;16;33;3;9;-1;-1
109689;10;-39;-1;38;3;10;-1;-16776961
109690;11;79;110;42;3;11;-1;-1
109691;12;-23;12;50;3;12;-1;-1
109692;13;126;45;43;3;13;-1;-1
109693;14;33;31;39;3;14;-1;-1
109694;15;-29;160;64;3;15;-1;-1
109695;16;-136;-28;55;3;16;-1;-1
```

Obr. 2: Formát vstupných dát

Riadok začínajúci 0 značí definovanie nového časového kroku, ktorý je daný šiestym číslom v riadku (na obrázku začiatok tretieho časového kroku). Formát ostatných riadkov je nasledujúci:

- 1. pozícia: globálne identifikačné číslo – unikátne číslo pre každý bod (tento údaj v našom programe nepoužívame)
- 2. pozícia: lokálne identifikačné číslo – unikátne číslo pre každý bod v rámci časového kroku
- 3., 4. a 5. pozícia: súradnice (x,y,z) bodu v priestore.
- 6 pozícia: časový krok
- 7. pozícia: lokálne identifikačné číslo materskej bunky z predchádzajúceho časového kroku.
- 8. pozícia: -1, ktorá pre náš program nič neznamená
- 9. pozícia: vlastná farba (nedefinovaná, ak sa rovná -1)

Pre uchovanie údajov a jednoduchšiu prácu s nimi v programe slúži nami vytvorená trieda Node. Dátový typ triedy dostal prednosť pred štruktúrou, pretože ide o referenčný typ a v programe je žiadúce udržiavať referencie (podobné smerníku v C++) každej bunky na svoju matku a všetky svoje dcérske bunky. Pri priraďovaní tried sa priradí referencia, ktorá ukazuje na daný objekt. V prípade triedy teda fungujú tieto referencie implicitne, zatiaľ čo v prípade použitia štruktúry by sa situácia skomplikovala, pretože štruktúra je hodnotový typ. Pri práci s premennou tohto typu

pracujeme priamo s jej hodnotou uloženou v zásobníku (stack), nie s odkazom na ňu a pri priraďovaní dochádza ku vzniku kópie.

Trieda Noda uchováva nasledujúce informácie:

- súradnice bunky v priestore
- lokálne identifikačné číslo
- zložky smerového vektora (od súradníc matky odčítame súradnice jednej z dcér)
- zložky smerového vektora pohybu bunky (od súradníc matky odčítame súradnice jednej z dcér)
- rýchlosť pohybu bunky (veľkosť smerového vektoru)
- referencia na materskú bunku a pole referencií na všetky dcérske bunky (premenné používané pri vykresľovaní trajektórií)
- informáciu o aktuálnej farbe a farbe zo vstupného súboru

Pri spracovávaní vstupného súboru program načíta dáta do „zubatého“ poľa, čo si môžeme predstaviť ako viacrozmerné pole s nepravidelnými rozmermi dimenzií. V našom prípade ide o 2D pole typu Node indexované podľa časového kroku, ktoré je navyše logicky poprepájané (prostredníctvom referencií medzi materskými a dcérskymi uzlami) a tvorí všeobecný strom.

Program zároveň pridáva jednotlivé vstupné farby do kolekcie typu List<Int32>, pomocou ktorej vieme rozlíšiť viaceré bunkové populácie a následne vykresľovať ich centrálnu trajektóriu.

3 Rendering

3.1 Formát Vertexu

Po spracovaní dát prechádzame k ich vykresľovaniu, pričom využívame možnosti knižnice Direct3D. V našom programe je potrebné zadefinovať dve rôzne štruktúry formátu vertexu (kľúčový bod scény – napr. vrchol trojuholníka).

Prvou je štruktúra TextureVertex, ktorú využívame iba pre vykreslenie 6 trojuholníkov tvoriacich 3 obdĺžniky, na ktoré sú mapované rezy mikroskopických snímiek. Táto štruktúra obsahuje informácie o polohe bodu a súradniciach textúry. Vector3 a Vector2 sú štruktúry knižnice SharpDX uchovávajúce súradnice bodu v priestore, resp. rovine.

```
struct TextureVertex
{
	public Vector3 Position;
	public Vector2 textureCoordinates;
}
```

Vo všetkých ostatných prípadoch využívame štruktúru Vertex, ktorá obsahuje informácie o polohe bodu a jeho farbe.

```
struct Vertex
{
	public Vector3 Position;
	public int Color;
}
```

3.2 Vertex a Index buffer

V každom zobrazovanom časovom kroku pracuje program s inou množinou vrcholov v závislosti od aktuálnych nastavení užívateľa. Pri každej zmene časového kroku dôjde k výpočtu aktuálnych informácií o vrcholoch (počet, pozície a farby), ktoré majú byť odoslané do grafickej karty a vykreslené. Tieto informácie sú uchovávané pomocou Vertex buffera. V programe využívame pri renderingu nasledujúce Vertex buffre:

- VertexBuffer vb – uchováva vrcholy a farby kociek - uzlov, ukladanie informácií do vb prebieha v metóde MakeCubeBuffers( ).
- VertexBuffer vb_lines – uchováva vrcholy a farby úsečiek – trajektórií, ukladanie informácií do vb_lines prebieha v metóde MakeLineBuffers( ).
- VertexBuffer vb_color, vb_no_color – uchovávajú vrcholy zvýrazňovaných uzlov, ktoré sa líšia iba vo farbe, ukladanie informácii do vb_color a vb_no_color prebieha v metóde Zvyraznovanie( ).
- VertexBuffer vb_rectangleXY, vb_rectangleXZ, vb_rectangleYZ – využívané pri vykresľovaní rezov reálnych dát, ukladanie informácií do týchto Vertex bufferov prebieha v metóde ZmenRez( ).
- VertexBuffer vb_sphere – uchováva vrcholy trojuholníkov tvoriace sféry využívané pri vykresľovaní centrálnych trajektórií, ukladanie informácií prebieha v metóde MakeSphereBuffers( ).

V konštruktore programu sú vytvorené všetky Vertex buffre dostatočnej veľkosti. (64MB – štyri milióny vrcholov pre jeden časový krok) a ďalej pri behu programu nedochádza k žiadnej inej alokácií pamäte pre potreby Vertex bufferov, pretože informácie v nich sú neustále prepisované. Keďže každý z Vertex bufferov vb_rectangleXY, vb_rectangleXZ a vb_rectangleYZ obsahuje iba 6 vrcholov, ich veľkosť je podstatne menšia.

Pri vykresľovaní uzlov je žiadúce správne indexovanie vrcholov. Pre tento účel slúži Index buffer, ktorý napomáha efektívnemu pamäťovému uchovávaniu. Potrebné Index buffre sú vytvorené v konštruktore programu podobným spôsobom. Ich veľkosť je 16MB. Pri zmene časového kroku sú aktuálne dáta uložené najprv v poli typu Vertex a pred samotným renderingom je potrebné ich načítanie priamo do Vertex buffera, čo sa môže diať iba ak je Vertex buffer zamknutý, pretože nechceme, aby sa s ním počas zápisu čokoľvek dialo. Uzamknutie Vertex buffera zabezpečuje metóda Lock( ). Po ukončení zápisu nesmieme zabudnúť na jeho odomknutie prostredníctvom metódy Unlock( ).

Pri renderingu využívame dve vykresľujúce metódy: DrawIndexedPrimitives( ) a DrawPrimitives( ). Obe vykresľujú základné geometrické tvary, prvá z nich na základe indexovaných vrcholov, druhá na základe postupnosti neindexovaných vrcholov.

3.3 Metóda DrawIndexedPrimitives( )

Túto metódu využívame iba pri vykresľovaní trojuholníkov, ktoré nám pomáhajú vytvoriť kocky - uzly trajektórie, ktorých pozície sa nachádzajú v strede prislúchajúcej kocky. Jej vrcholy nájdeme postupným pričítavaním a odčítavaním pevne stanovenej dĺžky (polomeru) k súradniciam uzlu. Informácie o vrcholoch sú uchovávané

pomocou Vertex buffera. Každá strana kocky je tvorená dvomi trojuholníkmi. Ak by sme ukladali informácie o každom vrchole všetkých trojuholníkov potrebných na vykreslenie kocky, potrebovali by sme pracovať s 36 vrcholmi, čo je neefektívne, keďže samotná kocka ich má iba 8 a trojuholníky ich zdieľajú. Je dôležité pracovať s čo najmenším počtom vrcholov, aby zbytočne nezaberali miesto v pamäti a nespomaľovali rendering. Preto využívame Index buffer. Zatiaľ čo do Vertex buffera uložíme informácie o 8 vrcholoch pre jednu kocku (žiadny z nich sa neopakuje), Index buffer bude obsahovať 36 indexov jednotlivých vrcholov a na základe ich správneho usporiadania vykreslíme 12 trojuholníkov, ktoré nám vytvoria jednu kocku. Tento spôsob vykresľovania je pamäťovo efektívnejší.

Pri každom pohybe trackbaru sú počítané informácie o pozíciách trojuholníkov a ich zafarbení v metóde MakeCubeBuffers( ), dokonca aj v prípade, že si užívateľ neželá uzly vykresľovať. To je spôsobené tým, že od farby uzlov závisí farba trajektórií a takisto je možná situácia, kedy je z celého Vertex bufferu zobrazených iba 12 trojuholníkov, čiže jedna kocka a to pri sledovaní pohybu konkrétnej bunky. Po napočítaní všetkých potrebných dát do Vertex a Index bufferu nasleduje volanie samotnej metódy DrawIndexedPrimitive( ). Nasledujúci kód je typickým príkladom využitia tejto metódy:

```
DrawIndexedPrimitive(
       PrimitiveType.TriangleList,//Typ primitívy
       0,                         //BaseVertexIndex - index prvého vertexu, zvyčajne 0
       0,                         //MinIndex - minimálny použitý index vertexu
       pocet_kociek * 8,          //NumVertices - počet vykresľovaných vertexov
       0,                         //StartIndex - index prvého indexu
       pocet_kociek * 12          //PrimitiveCount - počet primitív
);
```

3.4 Metóda DrawPrimitive( )

Pri všetkých ostatných možnostiach voláme metódu DrawPrimitive( ), teda pri vykresľovaní trajektórií, 2D rezov a centrálnych trajektórií tvorených zo sfér.

Pri trajektóriách vykresľujeme úsečky medzi dvojicou po sebe idúcich vrcholov a tu nám pri každej bunke stačí ukladať do Vertex buffera každý vrchol dvakrát, okrem prvého a posledného. Tým zabezpečíme správne vykreslenie trajektórie a nepoužívaním Index bufferu ušetríme určité množstvo pamäte. Volanie metódy môže potom vyzerať napr. takto:

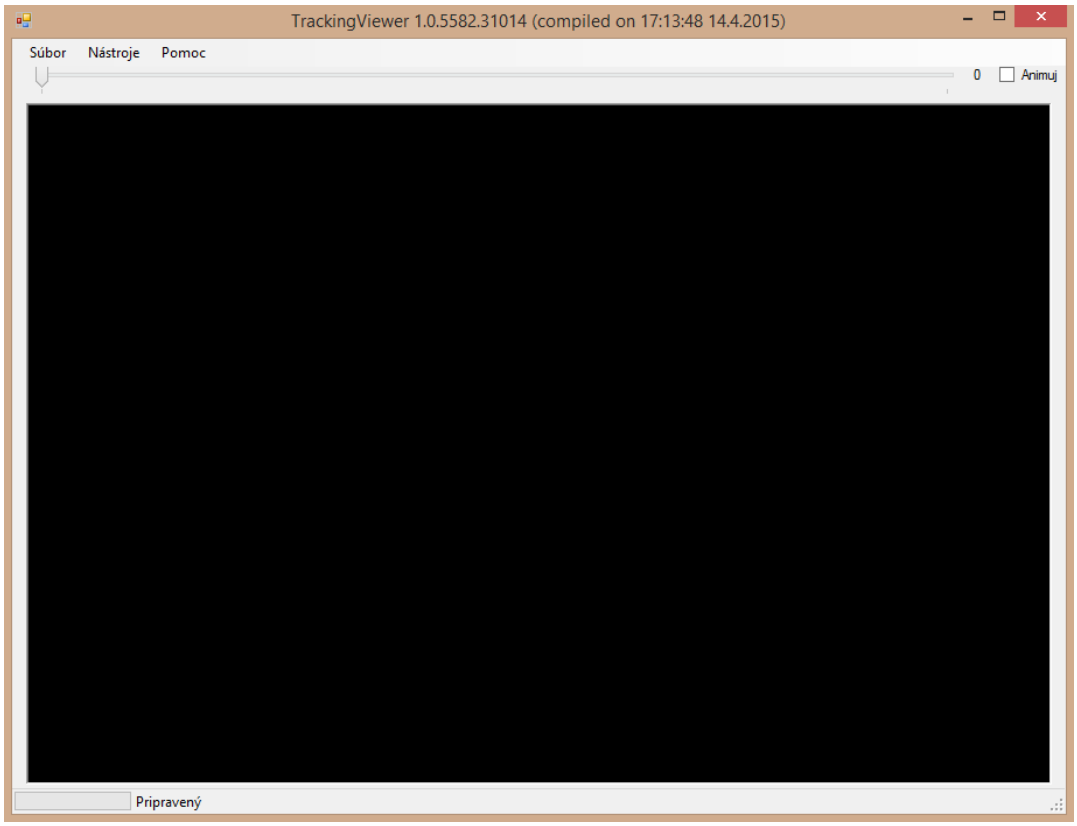
```
DrawPrimitives(
       PrimitiveType.LineList,    //Typ primitívy
       0,                         //StartVertex – index prvého vertexu
       pocet_primitiv             //PrimitiveCount - počet primitív
);
```

Pri zobrazovaní jednotlivých rezov sú vykresľované iba 2 trojuholníky a preto ich indexovanie nie je potrebné. V prípade vykresľovania sfér by indexovanie trojuholníkov, z ktorých je sféra zložená, bolo taktiež neefektívne a navyše pomerne komplikované.

4 Funkcionalita programu

4.1 Grafické rozhranie

Rozhranie programu je jednoduché a intuitívne ovládateľné (Obr. 3). Jeho podstatnú časť tvorí panel pre vykresľovanie dát. Posúvaním trackbaru dosiahneme vykreslenie ľubovoľného časového kroku, vpravo vidíme informáciu o jeho indexe. Trackbar po načítaní vstupných dát zmení maximum svojho rozsahu na počet časových krokov, ktoré sa v súbore nachádzajú. Celú scénu je možné animovať po zaškrtnutí možnosti Animuj. V hornej časti sa nachádza základné menu s tromi položkami – Súbor, Nástroje a Pomoc.



Obr. 3: Základné okno

Pod položkou Súbor ponúka program možnosti načítania súboru, uloženia snímky zobrazovaných dát vo formáte PNG (Portable Network Graphics) alebo ukončenia programu.

Po otvorení dátového súboru prebehne načítanie dát a výpočet všetkých potrebných Vertex a Index bufferov a následne dôjde k vykresleniu prvého časového kroku podľa predvolených nastavení – vykreslené sú uzly bez trajektórií v užívateľskej predvolenej farbe (červenej), pričom animácia nebeží. Všetky podstatné nastavenia vykresľovania scény meníme v položke Nástroje. Menu obsahuje aj položku Pomoc, pod ktorou nájdeme krátku nápovedu k program a informácie o programe.

V programe je možné ovládať scénu prostredníctvom myši. Ľavým tlačidlom myši otáčame scénu, pravým približujeme a stredným posúvame. Po kliknutí na panel je zachytená poloha kurzoru myši a následným pohybom nad panelom sú

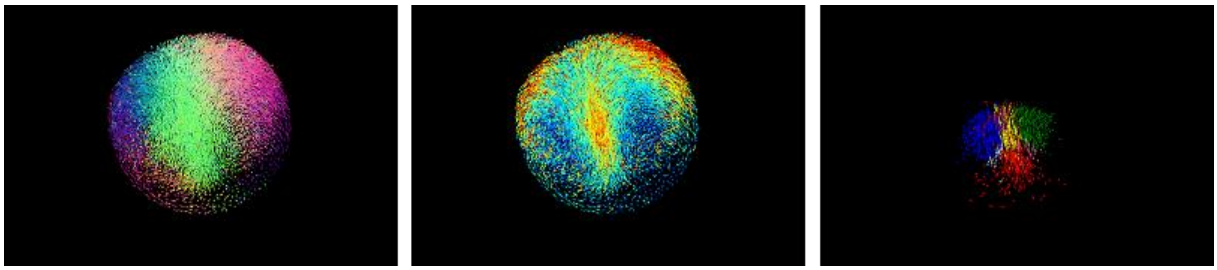
zaznamenané uhly pootočenia, posunutia a koeficient škálovania. Pomocou týchto parametrov vypočítame rotačnú, translačnú maticu a maticu zmeny veľkosti. Ich vzájomným násobením dostávame výslednú maticu, ktorá je argumentom funkcie SetTransform( ). Tá scénu vždy pred vykreslením transformuje, pričom je možné ju vrátiť do pôvodného stavu kliknutím na možnosť Resetovať ViewPort v položke Nástroje.

4.2 Režimy zafarbenia uzlov

Pod položkou Nástroje nájdeme možnosť Režim vyfarbenia, kde si môžeme vyberať medzi možnosťami zafarbenia uzlov. Metóda na výpočet Vertex a Index bufferov pre vykresľovanie kociek, MakeCubeBuffers(), sa na základe aktuálneho režimu zafarbenia rozhoduje, akým spôsobom bude generovať farbu pre vertexy:

- Podľa smerového vektora (Obr. 4 vľavo) – pri tejto možnosti vyrátame priemernú hodnotu jednotlivých zložiek smerového vektora pre interval niekoľkých materských a dcérskych buniek. Zložky výsledného vektoru preškálujeme na interval (0,255) a následne z neho generujeme farbu.
- Podľa rýchlosti (Obr. 4 v strede) – v tomto prípade vyrátame priemernú rýchlosť niekoľkých predchádzajúcich materských a nasledujúcich dcérskych buniek. Ich počet je daný premennou interval. Výsledná hodnota je určená ako minimum z priemernej rýchlosti buniek v rámci intervalu a dvojnásobku priemernej rýchlosti všetkých buniek. Tým sú eliminované prípadné extrémne hodnoty a celkové zafarbenie je rovnomerné. Využívame 255 odtieňov farebnej škály od tmavomodrej pri nízkych rýchlostiach až po červenú pre vysoké rýchlosti.
- Podľa preddefinovanej farby (Obr. 4 vpravo) - nastavenie farby načítanej zo vstupného súboru, pričom uzly s farbou hodnoty – 1 sú ignorované.

Program navyše ponúka možnosť užívateľského nastavenia farby všetkých uzlov, prípadne nezobrazovania uzlov vôbec.



Obr. 4: Režim smerového vektora, rýchlosti a preddefinovanej farby

4.3 Režimy trajektórií

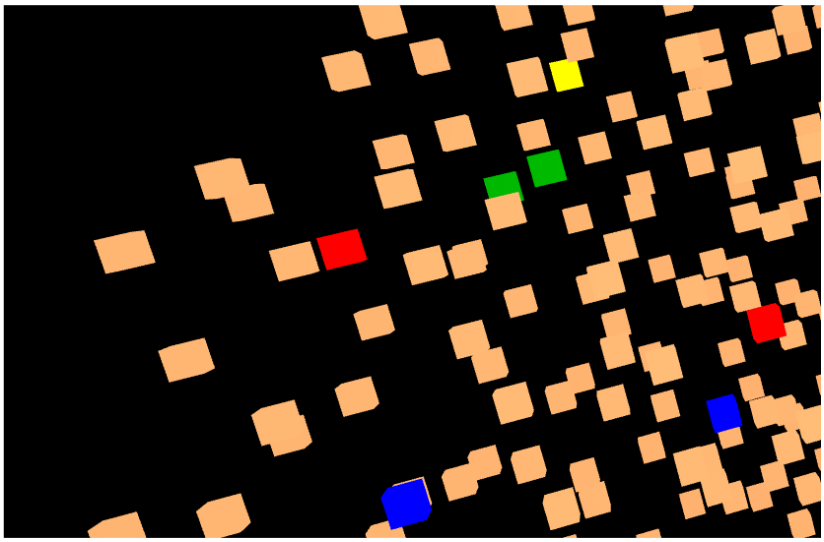
Možnosť rôznych vykresľujúcich režimov program ponúka aj pre trajektórie, pričom ich dĺžka je voliteľná a farba je zhodná s farbou prislúchajúceho uzla. O generovanie Vertex bufferu pre trajektórie sa stará metóda MakeLineBuffers(), kde využívame fakt, že naše 2D pole je poprepájané vzťahmi medzi materskými a dcérskymi bunkami a vieme sa v ňom ľahko pohybovať.

- Dopredné trajektórie (Obr. 5 vľavo) – bunky spájame so všetkými ich dcérskymi bunkami pomocou rekurzie v nasledujúcich časových krokoch. Trajektória určuje, v akom smere sa bude bunka v ďalších časových krokoch hýbať a deliť.
- Spätné trajektórie (Obr. 5 v strede) – tu je situácia priamočiarejšia, nepotrebujeme využívať rekurziu, pretože každá bunka má len jednu matku, s ktorou ju spájame. Trajektória nám ukazuje, z akého smeru daná bunka prišla.
- Obojsmerné (Obr. 5 vpravo) – bunky sú spájané aj s matkami aj s dcérami. Vidíme dopredný aj spätný pohyb bunky a pri generovaní Vertex buffera kombinujeme prvé dva postupy popísané vyššie.
- Žiadne – v tomto prípade metóda MakeLineBuffers( ) končí hneď na začiatku a vo funkcii Render( ) nedochádza k vykresľovaniu trajektórii.

Obr. 5: Dopredná, spätná a obojstranná trajektória

4.4 Zvýrazňovanie trajektórií

Ďalšia funkcionalita sa týka zvýrazňovania uzlov (Obr. 6), v ktorých končí (červená farba) alebo začína trajektória (modrá), prípadne sú zvýraznené uzly signalizujúce bunky, ktoré sa v predchádzajúcom časovom kroku delili (žltá) alebo sa v ďalšom kroku budú deliť (zelená) a bunky, ktoré sa ani nedelili, ani sa deliť nebudú (siroty - fialová). Tieto stavy sa dajú zapínať a vypínať nezávisle od seba, napríklad môžu byť zapnuté všetky naraz alebo žiaden. Zapnutie stavu sa prejaví tak, že skupina uzlov, ktorá vyhovuje danému stavu, začne blikať (pre každý stav inou farbou). Toto sme dosiahli použitím dvojice Vertex bufferov, ktoré obsahujú rovnakú množinu vrcholov líšiacich sa iba vo farbe. Jeden z Vertex bufferov obsahuje vrcholy farby daného stavu a druhý vrcholy bielej farby. Táto dvojica Vertex bufferov je následne striedavo vykresľovaná, čo spôsobí blikanie zvýrazňovaných uzlov.



Obr. 6: Zvýrazňovanie uzlov

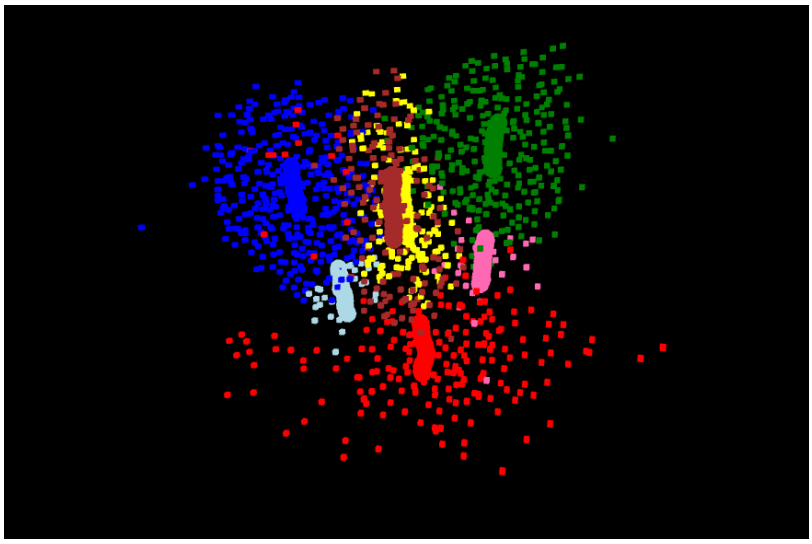
4.5 Centrálne trajektórie

Po aktivovaní tejto možnosti je volaná metóda MakeSphereBuffers( ), ktorá využíva vopred vypočítané informácie. Už pri načítaní súboru si program uchováva informácie o jednotlivých vstupných farbách v kolekcii typu List<Int32>. Pre bunky rovnakého zafarbenia je následne vypočítaná priemerná hodnota ich súradníc pre každý časový krok. Tieto informácie sú uložené v dvojrozmernom poli typu Vector3[][], kde dĺžka prvej dimenzie je rovná počtu farieb v kolekcii a druhá počtu časových krokov v súbore. Ďalej je volaná metóda MakeSphere ( ), ktorá do poľa typu Vector3[] pripraví pozície vrcholov trojuholníkov tvoriacich pomocnú sféru umiestnenú do počiatku súradnicovej sústavy. Táto sféra však nikdy nie je vykreslená a je využitá iba v metóde MakeSphereBuffers( ), kde sa k jednotlivým pozíciám vrcholov trojuholníkov postupne pripočítava daná priemerná hodnota súradníc rovnako zafarbených buniek pre konkrétny časový krok, čím sa sféra posúva na požadované miesto. Takýmto spôsobom sú vytvorené sféry pre niekoľko časových krokov pre každú farbu a vznikajú centrálne trajektórie pre všetky farby zo vstupného súboru, inými slovami, pre každú bunkovú populáciu (Obr. 7).

Vrcholy trojuholníkov tvoriacich pomocnú sféru sú vypočítané pomocou prechádzania sférickými súradnicami:

```
Vector3 point = new Vector3();
point.X = (float)(radius * Math.Sin(theta) * Math.Cos(phi));
point.Y = (float)(radius * Math.Sin(theta) * Math.Sin(phi));
point.Z = (float)(radius * Math.Cos(theta));
```

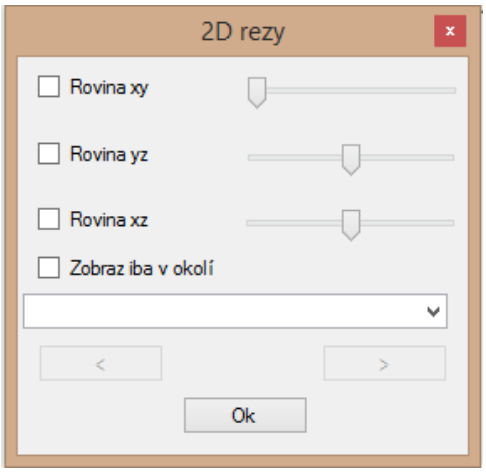
kde theta je z intervalu (0, Math.Pi) a phi z intervalu (0, 2*Math.Pi).



Obr. 7: Centrálne trajektórie

4.6 Zobrazenie 2D rezov mikroskopických snímiek

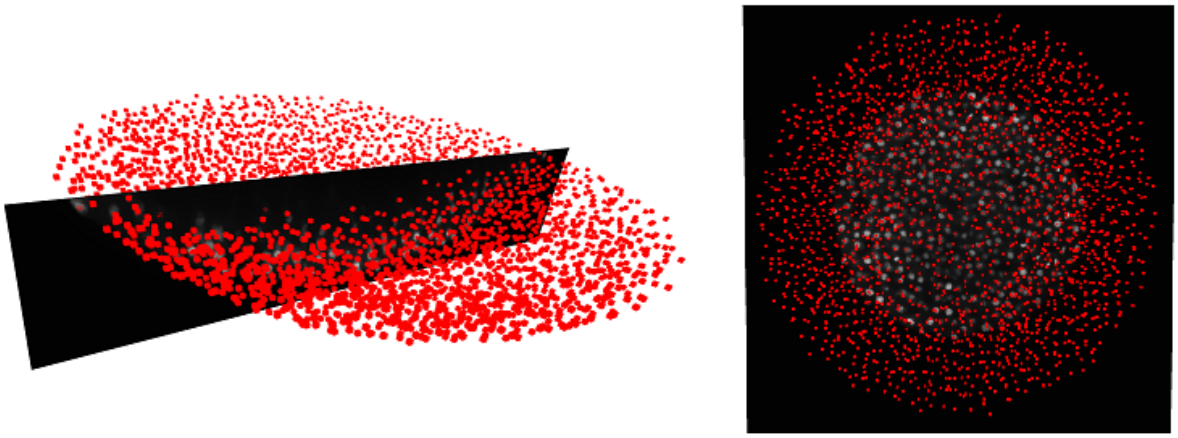
Po kliknutí na možnosť „2D rezy“ pod položkou Nástroje ⟶ Zobrazenie sa zobrazí okno (Obr. 8), ktoré ponúka možnosť vykreslenia jednotlivých rezov 3D mikroskopických snímiek, ktoré sú načítané v konštruktore programu do dvojrozmerného poľa typu byte[][]. Po zaškrtnutí checkBoxu pri požadovanej rovine sa vykreslí obdĺžnik (resp. dva trojuholníky), na ktorý je mapovaný rez dátami v konkrétnom časovom kroku.



Obr. 8: 2D rezy

Pre každú rovinu definujeme objekt Texture príslušnej výšky a šírky (tieto informácie sú získané pri načítavaní snímiek). Týmto rozmerom odpovedajú aj rozsahy jednotlivých trackbarov a pri ich pohyboch je volaná metóda, v rámci ktorej sa do textúry načíta rez roviny, ktorú určuje hodnota z trackbaru. Následne sú vykreslené odpovedajúce obdĺžniky spolu s textúrou. Pri ich vykresľovaní používame formát vertexu, ktorý okrem súradníc v priestore nesie aj informáciu o súradniciach textúry, ktoré môžu mať tvar (0,0), (1,0), (0,1), (1,1) a definujú spôsob, akým je textúra na obdĺžnik vykreslená, čiže označujú umiestnenie ľavého (resp. pravého) horného a dolného rohu obrázku.

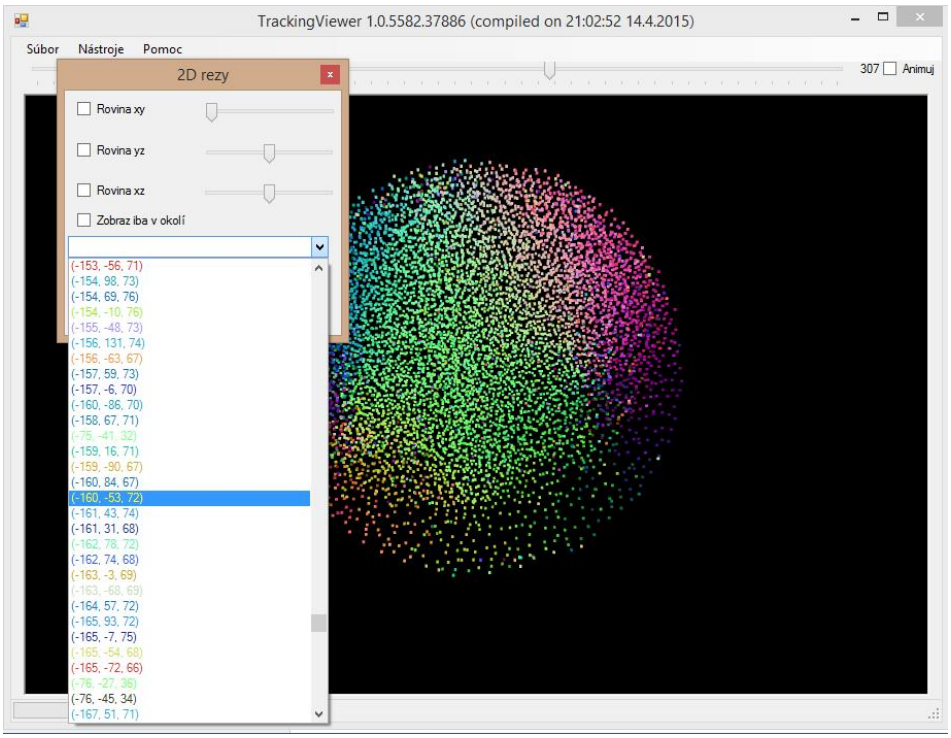
Program takýmto spôsobom vykresľuje jednotlivé rezy rovín (Obr. 9) a pomocou trackbaru umožňuje pohodlný prechod medzi ich polohami. Tak isto je možné pre konkrétne polohy rovín prechádzať časovými krokmi vývoja embrya, prípadne zobrazovať iba bunky nachádzajúce sa v tesnej blízkosti rezu.



Obr. 9: 2D rezy

4.7 Sledovanie pohybu bunky

Okno spomenuté vyššie (Obr. 8) obsahuje aj možnosť sledovania pohybu konkrétnej bunky. Tú si vyberieme prostredníctvom combo boxu, ktorý obsahuje všetky uzly daného časového kroku a pri jeho zmene sa vždy aktualizuje. Uzly sú reprezentované textovým reťazcom priestorových súradníc (Obr. 10), pričom farba textu je zhodná s farbou uzlu. Po výbere konkrétnej bunky nasleduje jej vykreslenie a je aktivované sledovanie jej pohybu. Spolu s bunkou je možná vizualizácia jej trajektórie a prislúchajúcich 2D rezov mikroskopických snímiek. Dopredný a spätný pohyb bunky sledujeme pomocou tlačidiel nachádzajúcich sa pod combo boxom. Ak aktuálne vybraná bunka nemá matku, je možné iba dopredné sledovanie jej pohybu. Naopak, ak nemá dcéru, umožnené je iba spätné sledovanie. V prípade, že bunka má viac dcér, program v rámci nich ponúka užívateľovi možnosť výberu.



Obr. 10: Combo box

5 Záver

Podarilo sa nám rozšíriť funkcionalitu programu na vizualizáciu dát reprezentujúcich pohyb a delenie buniek v 3D priestore a čase počas vývoja živočíchov. Navyše je spolu s numerickými dátami možné zobrazovať rezy originálnych mikroskopických snímiek, čo nám poskytuje lepšiu predstavu o získaných dátach. Program využíva grafickú kartu počítača, čo umožňuje rýchlejšiu a efektívnejšiu vizualizáciu, práca s ním je intuitívna a je možné ho rozšíriť o ďalšiu funkcionalitu.

Literatúra

[1] Mikula, K., Peyrieras, N., Špir, *R. Numerical algorithm for tracking cell dynamics in 4D biomedical images*. Submitted to Discrete and Continuous Dynamical Systems – Series S

[2] Mikula, K., Špir, R., Smíšek, M., Faure, E., & Peyrieras, N. (2013) *Nonlinear PDE based numerical methods for cell*. Submitted to Applied Numerical Mathematics

[3] Miroslav Virius. *C# 2010 Hotová řešení*. Computer Press, 2010

[4] SharpDX, The power of DirectX for .NET, http://www.sharpdx.org/