Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vizualizace dělení prostoru v dynamických modelech makromolekul

Plzeň 2020 Oto Šťáva

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Oto ŠŤÁVA**
Osobní číslo:	**A16B0148P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Vizualizace dělení prostoru v dynamických modelech makromolekul**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s dělením prostoru pomocí Voroného diagramů a jejich využitím pro modelování a vizualizaci modelů makromolekul.
2. Navrhněte řešení pro přehlednou vizualizaci a zkoumání vlastností takového měnícího se dělení prostoru.
3. Implementujte řešení jako rozšiřující modul do existujícího nástroje CAVER Analyst, který se používá k analýze dynamických makromolekul.
4. Ověřte řešení na reálných datech dynamických makromolekul a zhodnoťte dosažené výsledky.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2020

Oto Šťáva

Abstract

Modelling and visualization of macromolecular protein structures play an important part in medical informatics. These visualizations may be used by medical experts to search for and examine biologically active places in these molecules. Existing software solutions use algorithms based on analysis of space partitioning into Voronoi diagrams to find cavities and tunnels, which may be used as potential access paths to those active places. This thesis concerns graphical visualization of additively weighted Voronoi diagrams, which would allow further examination of their behaviour, particularly in dynamically changing molecule models. As a part of the thesis, an extension module for the software application CAVER Analyst visualizing these diagrams was implemented.

Abstrakt

Modelování a vizualizace makromolekulárních struktur proteinů jsou důležitou součástí medicínské informatiky. Tyto vizualizace mohou experti v tomto oboru využívat ke hledání a zkoumání biologicky aktivních míst v těchto molekulách. Ke hledání dutin a tunelů, které mohou představovat možné přístupové cesty k těmto aktivním místům, jsou využívány algoritmy využívající dělení prostoru do Voroného diagramů. Tato práce se zabývá grafickou vizualizací aditivně vážených Voroného diagramů za účelem výzkumu chování těchto diagramů, a to zejména nad dynamickými modely molekul. Součástí práce je softwarová implementace této vizualizace v podobě rozšiřujícího modulu pro aplikaci CAVER Analyst.

Poděkování

Rád bych tímto poděkoval panu Mgr. Martinu Maňákovi Ph.D., vedoucímu práce, za odborné rady, náměty, připomínky a celkovou podporu, jež mi poskytoval během jejího vypracování.

Obsah

1 Úvod **8**
- 1.1 Motivace . . . 8
- 1.2 Cíle práce . . . 9
- 1.3 Organizace práce . . . 9

2 Analytická část **10**
- 2.1 Voroného diagramy . . . 10
- 2.2 Racionální Bézierova křivka . . . 11
  - 2.2.1 Algoritmus de Casteljau . . . 12
- 2.3 Existující aplikace . . . 14
  - 2.3.1 Voroprot . . . 15
  - 2.3.2 BetaMol . . . 15
  - 2.3.3 Voro++ a POV-Ray . . . 16
  - 2.3.4 Knihovna awVoronoi a program VDS Viewer . . . 17
  - 2.3.5 BetaConcept . . . 17
- 2.4 Užité technologie . . . 18
  - 2.4.1 CAVER Analyst . . . 18
  - 2.4.2 OpenGL . . . 21
- 2.5 Analýza . . . 24

3 Realizační část **27**
- 3.1 Uživatelský pohled na Voronoi Visualizer . . . 27
- 3.2 Rozšíření aplikace CAVER Analyst . . . 28
  - 3.2.1 Rozšiřitelný model . . . 29
  - 3.2.2 Rozhraní vykreslovacích strategií . . . 30
- 3.3 Kvadratické racionální Bézierovy křivky . . . 31
  - 3.3.1 Distribuce vzorků . . . 32
- 3.4 Filtrování Voroného diagramu . . . 36
- 3.5 Zvýraznění některých částí diagramu . . . 36
  - 3.5.1 Uživatelské obarvení . . . 37
  - 3.5.2 Obarvení podle vah Voroného sfér . . . 38
- 3.6 Vizualizace diagramů v dynamice . . . 39

4 Dosažené výsledky **41**
- 4.1 Prostředí, metodiky a data . . . 41
  - 4.1.1 Testovací prostředí . . . 41

4.1.2 Měřicí metody 41
4.1.3 Vstupní data 42
4.2 Výkon vizualizace Voroného hran 42
4.3 Porovnání s aplikací Voroprot 47
4.4 Zhodnocení výsledků 47

5 Závěr **49**

Seznam zkratek **50**

Literatura **51**

1 Úvod

Počítačové modelování a vizualizace jsou v dnešní informační době velmi důležitým podoborem informatiky, který spojuje na první pohled velmi odlišné oblasti vědy. Přirozeným způsobem tak dochází k propojení fascinujícího světa atomů a molekul, různých oblastí matematiky a informatiky. Chování složitých molekul v různých hypotetických situacích často nelze pozorovat přímo, ale lze pracovat s geometrickým modelem, na něm provádět fyzikální simulace a pozorování provádět nepřímo. Velkou roli v tomto procesu hraje kvalitní a přehledná prezentace informací pozorovateli. Tato práce se zaměřuje na vizualizaci prostorových vztahů mezi atomy a rozšiřuje tak možnosti, které lze při pozorování chování molekul využít.

1.1 Motivace

V medicínské informatice se můžeme setkat s modelováním makromolekulárních struktur proteinů, které mohou experti v tomto oboru využívat ke hledání a zkoumání biologicky aktivních míst v těchto makromolekulách.

Molekula je skupina atomů, které jsou pohromadě drženy chemickými vazbami a společně tak získávají různé chemické vlastnosti. Makromolekulou pak nazýváme molekulu o velkém množství atomů.

Makromolekuly lze vizualizovat s přihlédnutím k různým vlastnostem, na které se může výzkumník v danou chvíli zaměřovat. Mezi nejčastější typy těchto vizualizací patří například vykreslení *van der Waalsova modelu*, v němž jsou jednotlivé atomy zobrazeny jako sféry o velikosti odpovídající van der Waalsovu poloměru daného typu atomu [1], či zobrazení atomů se spojnicemi reprezentujícími chemické vazby.

S modelem molekuly lze rovněž provádět různé fyzikální simulace. Výsledkem takových simulací pak může být *dynamika* modelu molekuly ve formě snímků pozic atomů v čase, zachycující možný pohyb jednotlivých atomů při simulaci působení vnějších vlivů prostředí.

Při návrhu léčiv lze za účelem cílené úpravy chování některých enzymů hledat přístupové tunely a dutiny k biologicky aktivním místům makromolekul. Jejich umístění je možné nalézt za použití algoritmů využívajících dělení prostoru do *Voroného diagramu*. Při konstrukci tohoto diagramu je molekula reprezentována jako množina kuliček, které jsou definovány pozicemi atomů a již zmíněnými van der Waalsovými poloměry. Prostor je poté rozdělen na

regiony náležící jednotlivým kuličkám na základě těchto pozic a poloměrů.

Pro výzkumníky zabývající se přímo výpočtem a chováním těchto diagramů nad různými druhy vstupních dat pak může být užitečná jejich přehledná vizualizace, a to nejen ve statických, ale i v dynamických množinách objektů použitých jako vstupní data pro jejich sestrojení.

Existuje několik softwarových řešení pro výzkum Voroného diagramů, avšak ne vždy je jejich vizualizace zcela přehledná, např. kvůli vykreslování hran pouze pomocí stejně silných čar, čímž zaniká efekt perspektivy; či její implementace není adekvátně výpočetně efektivní, aby bylo možné pohodlně zkoumat větší struktury. Veškerá známá řešení se také zabývají vizualizací diagramů výhradně nad *statickými*, tj. v čase neměnnými, vstupními daty.

CAVER Analyst 2.0 je existující a výzkumníky používaná aplikace, která již využívá aditivně vážené Voroného diagramy pro výpočet dutin v molekulách proteinů. Přestože aplikace tyto diagramy využívá, není v ní implementována možnost je za účelem jejich bližšího zkoumání zobrazovat.

1.2 Cíle práce

Cílem této bakalářské práce je nalézt uživatelsky přehledný a výpočetně efektivní způsob vizualizace *aditivně vážených Voroného diagramů* nad *dynamickými* (tj. v čase se měnícími) modely makromolekul či jinými množinami vstupních dat, nad nimiž lze tyto diagramy sestrojovat, a implementace této vizualizace v podobě rozšiřujícího modulu pro aplikaci *CAVER Analyst 2.0* za použití knihovny *OpenGL*, jež je dosavadní implementací aplikace využívána k vykreslování vizualizací.

1.3 Organizace práce

První, analytická část této práce se zabývá definicemi nejdůležitějších pojmů, tj. Voroného diagram, jeho části, druhy a reprezentace hran diagramu pomocí křivek; a analýzou problému vizualizace Voroného diagramů, včetně průzkumu a zhodnocení již existujících řešení. Druhá část obsahuje detaily k samotné softwarové implementaci vizualizace dynamických modelů v podobě rozšiřujícího modulu pro aplikaci CAVER Analyst 2.0 a popis technologií, které byly pro její realizaci použity. Ve třetí a poslední části se nacházejí záznamy experimentů a jejich výsledků a detailní zhodnocení každého výsledku.

2 Analytická část

V této části práce se budeme nejprve zabývat definicemi dvou důležitých pojmů: *Voroného diagram* a *racionální Bézierova křivka*. Dále si uvedeme několik existujících aplikací pracující s Voroného diagramy a poznatky z jejich průzkumu – tj. zda a jak jsou tyto aplikace schopny Voroného diagramy vizualizovat, popř. zda jsou schopny vizualizovat informaci s dělením prostoru související. Seznámíme se s technologiemi nutnými k implementaci navrženého rozšíření pro *CAVER Analyst*. V poslední sekci je samotná analýza problému vizualizace Voroného diagramu a popis obecného řešení s přihlédnutím k výhodám či nevýhodám existujících aplikací a použitých technologií.

2.1 Voroného diagramy

Pro zadanou množinu bodů $S = \{s_1,\ s_2, ...,\ s_n\} \subset \mathbb{R}^d$ definujeme Voroného diagram [9, 24] jako dělení prostoru $\mathbb{R}^d$ na Voroného buňky $V_1, \ldots, V_n$ podle následujícího předpisu:

$$V_i \ = \ \{a \in \mathbb{R}^d;\ dist(a, s_i) \leq dist(a, s_j);\ \forall s_j \in S\}. \tag{2.1}$$

Funkce *dist* určuje vzdálenost bodů prostoru $\mathbb{R}^d$ od prvků S. Každá buňka V_i tak vymezuje podprostor bodů $\mathbb{R}^d$ minimální vzdálenosti od daného prvku $s_i \in S$ vůči všem ostatním prvkům $s_j \in S$. Typicky se používá euklidovská vzdálenost:

$$dist_e(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_d - b_d)^2}, \tag{2.2}$$

validní jsou ale i jiné možnosti porovnávání vzdáleností.

U *aditivně vážených Voroného diagramů* máme pro každý bod $s_i \in S$ navíc zadanou i jeho váhu $w_i \in \mathbb{R}$. Vzdálenost bodů prostoru $\mathbb{R}^d$ od bodů s vahou pak definujeme pomocí následující funkce:

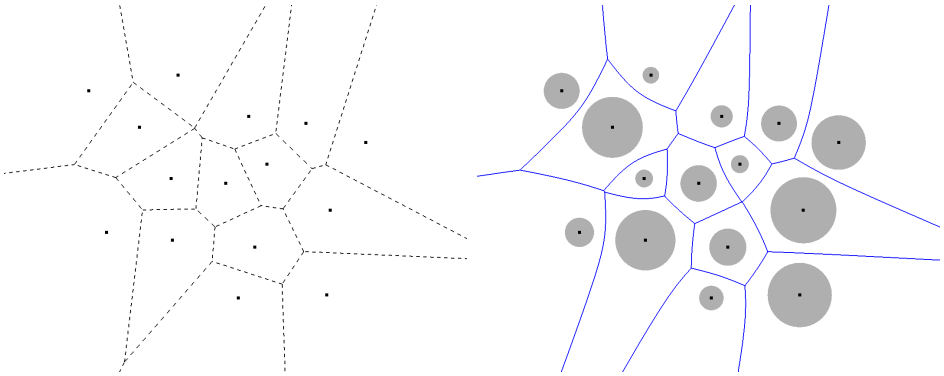
$$dist_{aw}(a, s_i) = dist_e(a, s_i) - w_i \tag{2.3}$$

Funkční hodnotu lze v tomto případě interpretovat jako vzdálenost testovaného bodu od povrchu sféry se středem v s_i a poloměrem w_i. Pokud je funkční hodnota záporná, nachází se testovaný bod uvnitř sféry; je-li kladná, nachází se vně; a je-li nulová, nachází se bod na povrchu sféry.

V trojrozměrném prostoru $\mathbb{R}^3$ se hranice Voroného buněk skládá z Voroného vrcholů, hran a stěn. *Voroného vrchol* definujeme jako bod průniku

čtveřice Voroného buněk, *Voroného hranu* jako komponentu (maximální neprázdnou souvislou množinu) průniku trojice Voroného buněk, a *Voroného stěnu* jako komponentu průniku čtveřice Voroného buněk. Voroného vrchol lze také interpretovat jako střed *Voroného sféry*, tj. prázdné sféry, která se dotýká čtyř sfér určených středy $s_i \in S$ a poloměry w_i. Poloměr této Voroného sféry, občas rovněž nazývaný vahou, může být důležitou součástí vstupních dat při využívání Voroného diagramů v rámci algoritmů pro výpočet dutin a tunelů.

Jak lze vidět na obrázku 2.1, při použití euklidovské vzdálenosti jsou hrany Voroného diagramu rovné, u aditivně vážených Voroného diagramů pak mají tvar kuželoseček a lze je tedy reprezentovat jako kvadratické racionální Bézierovy křivky.



Obrázek 2.1: Příklad Voroného diagramu pro euklidovskou vzdálenost (vlevo) a aditivně váženého Voroného diagramu (vpravo) v $\mathbb{R}^2$

2.2 Racionální Bézierova křivka

Pro $n+1$ kontrolních bodů P_0, P_1, P_2, ..., P_n s vahami w_0, w_1, w_2, ..., w_n definujeme *racionální Bézierovu křivku* $C_n(t)$ [26, 29] následujícími vztahy:

$$\begin{aligned} P_{i,n}(t) &= B_{i,n}(t) w_i P_i \\ w_{i,n}(t) &= B_{i,n}(t) w_i \\ C_n(t) &= \frac{\sum_{i=0}^{n} P_{i,n}(t)}{\sum_{i=0}^{n} w_{i,n}(t)}, \end{aligned} \tag{2.4}$$

kde $t \in \langle 0; 1 \rangle$, n je řád křivky a $B_{i,n}$ je *Bernsteinův polynom* [25]:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{2.5}$$

Racionální Bézierova křivka vždy prochází body P_0 a P_n a v případě, že jsou váhy všech jejích kontrolních bodů větší než 0, leží uvnitř jejich konvexní obálky. Vektory $(P_1 - P_0)$ a $(P_n - P_{n-1})$ jsou tečné vektory křivky v bodech P_0 a P_n.

Jak bylo zmíněno v předchozí sekci, u aditivně vážených Voroného diagramů mají Voroného hrany tvar kuželoseček, které lze vyjádřit jako kvadratické racionální Bézierovy křivky:

$$C_2(t) = \frac{\sum_{i=0}^{2} \binom{2}{i} t^i (1-t)^{2-i} w_i P_i}{\sum_{i=0}^{2} \binom{2}{i} t^i (1-t)^{2-i} w_i}. \tag{2.6}$$

V popisu Voroného hrany pomocí takové křivky pak body P_0 a P_2 odpovídají pozicím Voroného vrcholů hrany a obě váhy w_0 a w_2 jsou 1. Bod P_1 leží v průsečíku tečných vektorů, příslušná váha w_1 se ale počítá složitěji, a to přes další bod ležící na kuželosečce, který je nutné spočítat a určit, zda leží na segmentu Voroného hrany nebo v jeho doplňku. Podle toho se nastaví znaménko váhy w_1. Podrobněji je tento postup rozebrán v článku [3].

2.2.1 Algoritmus de Casteljau

Tento algoritmus založený na strategii *rozděl a panuj* generuje lomenou čáru aproximující libovolnou Bézierovu křivku rekurzivním dělením křivky na kratší úseky, přičemž nejvyšší hustota úseků se nachází v nejkřivějších částech hrany [14, 15, 29].

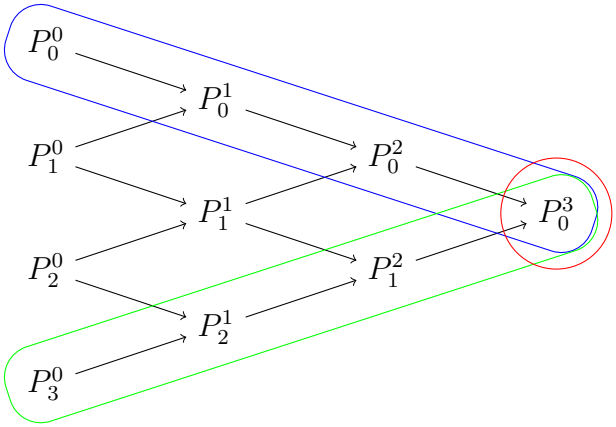
Výpočet křivky a její dělení

Pro algoritmus de Casteljau definujeme racionální Bézierovu křivku řádu n s parametrem $t \in \langle 1; 0 \rangle$ a kontrolními body P_0, P_1, …, P_n následujícím rekurentním vztahem, ilustrovaným obrázkem 2.2:

$$\begin{gathered} w_i^0 = w_i, \qquad P_i^0 = P_i \\ \text{pro } i = 0, 1, \ldots, n; \\ \\ w_i^j = (1-t) \cdot w_i^{j-1} + t \cdot w_{i+1}^{j-1}, \\ P_i^j = \frac{(1-t) \cdot w_i^{j-1} \cdot P_i^{j-1} + t \cdot w_{i+1}^{j-1} \cdot P_{i+1}^{j-1}}{w_i^j} \\ \text{pro } j = 1, 2, \ldots, n \text{ a } i = 0, 1, \ldots, n-1. \end{gathered} \tag{2.7}$$

Bod P_0^n (na schématu 2.2 vyznačený červeně) pak pro libovolnou hodnotu parametru $t \in \langle 1; 0 \rangle$ leží na této křivce.

Křivku je pak možné při přiřazení parametru $t = t_0$, kde t_0 je předělový parametr určující, v jaké části se má křivka rozdělit, vyjádřit jako dvě nezávislé Bézierovy křivky. Křivku $C_{nA}(t)$ s kontrolními body P_0^0, P_0^1, ..., P_0^n a jejich vahami w_0^0, w_0^1, ..., w_0^n (na schématu 2.2 vyznačeno modře); a křivku $C_{nB}(t)$ s kontrolními body P_0^n, P_1^{n-1}, ..., P_n^0 a jejich vahami w_0^n, w_1^{n-1}, ..., w_n^0 (na schématu 2.2 vyznačeno zeleně).



Obrázek 2.2: Schéma rekurentního vztahu definujícího Bézierovu křivku 3. řádu pro algoritmus de Casteljau (analogicky pro jejich váhy w_i^j)

Při dělení kvadratické racionální Bézierovy křivky $C_2(t)$ jsou pak po úpravě s použitím vztahu (2.4) kontrolní body křivek $C_{2A}(t)$ a $C_{2B}(t)$ následující:

$$\begin{aligned} w_{A0} &= w_0, & P_{A0} &= P_0, \\ w_{A1} &= \sum_{i=0}^{1} w_{i,2}(t_0), & P_{A1} &= \frac{\sum_{i=0}^{1} P_{i,2}(t_0)}{\sum_{i=0}^{1} w_{i,2}(t_0)}, \\ w_{A2} = w_{B0} &= \sum_{i=0}^{n} w_{i,2}(t_0), & P_{A2} = P_{B0} &= C_2(t_0), \\ w_{B1} &= \sum_{i=1}^{2} w_{i,2}(t_0), & P_{B1} &= \frac{\sum_{i=1}^{2} P_{i,2}(t_0)}{\sum_{i=1}^{2} w_{i,2}(t_0)}, \\ w_{B2} &= w_2, & P_{B2} &= P_2 \end{aligned} \tag{2.8}$$

Rekurzivní aproximace

Algoritmus de Casteljau křivku za účelem její aproximace lomenou čarou rekurzivně dělí na menší úseky. Využívá přitom *faktoru rovnosti* $f \geq 1$,

uživatelsky volitelné konstanty pro následující podmínku:

$$|P_0 - P_1| + |P_1 - P_2| < f \cdot |P_0 - P_2| \tag{2.9}$$

$$|P_0 - C(t)| + |C(t) - P_2| < f \cdot |P_0 - P_2|, \tag{2.10}$$

kde P_0, P_1 a P_2 jsou kontrolní body vstupní Bézierovy křivky a t je parametr křivky zvolený tak, aby bod $C(t)$ co nejlépe reprezentoval její křivost, zpravidla můžeme používat $t = \frac{1}{2}$. Použití nízkých hodnot faktoru křivosti f má za výsledek větší množství výsledných úseků, vyšší hodnoty detail snižují.

Dokud podmínka (2.9), resp. (2.10) není splněna, dělí algoritmus vstupní křivky na křivky „poloviční“, tj. podle předělového parametru $t_0 = \frac{1}{2}$. Každá z těchto polovičních křivek je dále použita jako vstup pro další iteraci tohoto algoritmu. Pokud podmínka splněna je, je příslušná část křivky nahrazena úsečkou spojující příslušné kontrolní body P_0 a P_2. Tímto vzniká lomená čára aproximující vstupní Bézierovu křivku [2].

Podmínka je zde prezentována ve dvou variantách. Varianta (2.9) je původní podmínkou prezentovanou v literatuře. Tato podmínka však křivost nereprezentuje dobře pro racionální Bézierovy křivky, neboť váha kontrolního bodu může výslednou křivost značně ovlivnit, především pak pro záporné váhy kontrolních bodů, kdy křivka nemusí ležet v konvexní obálce kontrolních bodů. Definujeme proto alternativní variantu (2.10) této podmínky, kde se namísto prostředního kontrolního bodu P_1 použije reprezentativní bod racionální Bézierovy křivky. Jelikož i pro racionální Bézierovu křivku platí, že $C(0) = P_0$ a $C(1) = P_2$ nezávisle na váze jejích kontrolních bodů, není nutné je nahrazovat výpočtem bodů křivky a je možné tyto kontrolní body použít přímo.

2.3 Existující aplikace

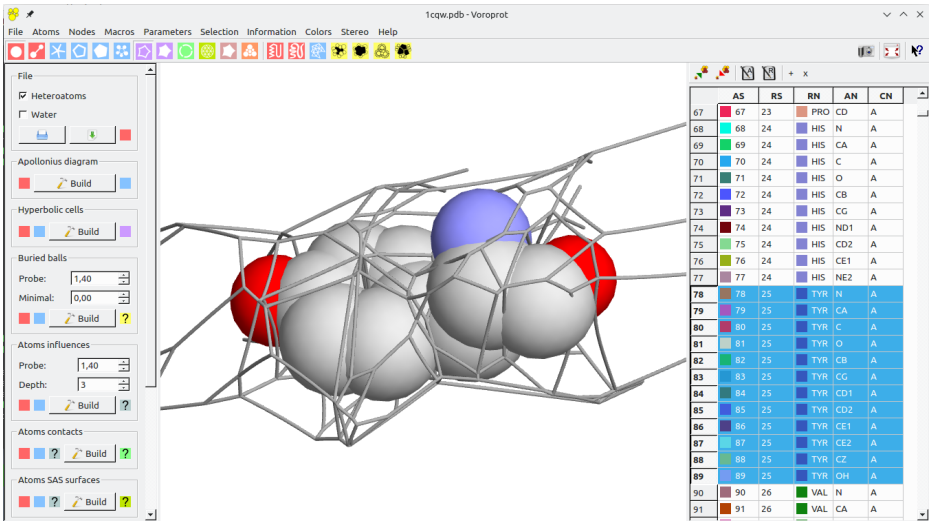
Existuje mnoho způsobů, jak vizualizovat Voroného diagramy, především pak klasické euklidovské. Ukazuje se však, že není mnoho aplikací, které by byly schopny zobrazovat *aditivně vážené Voroného diagramy*, a ty, které je vizualizují, tak zpravidla nečiní příliš efektivně, což se citelně projevuje na výkonu zejména u větších vstupních dat. Žádná ze známých aplikací navíc nepodporuje vizualizaci struktur měnících se v čase – veškeré známé vizualizace jsou nad statickými vstupními daty. Tato kapitola se věnuje popisu a zhodnocení vlastností několika zkoumaných existujících aplikací.

2.3.1 Voroprot

Voroprot je softwarová aplikace pro analýzu proteinových struktur vyvinutá v Institutu biotechnologie Vilniuské univerzity v Litvě [10]. Je schopna vizualizace těchto struktur, avšak pouze v jejich statické podobě. Voroprot poskytuje možnost konstrukce a vizualizace aditivně vážených Voroného diagramů. Obrázek 2.3 demonstruje grafické uživatelské rozhraní a vizualizační schopnosti této aplikace.

Podle dostupných zdrojových kódů tento program při vizualizaci Voroného diagramu konstruuje jeho kompletní model včetně vzorků Bézierových křivek pouze s využitím CPU.

Důsledkem konstrukce na CPU a přenosem veškerých segmentů vzorkovaných křivek na GPU je program při velkém počtu zobrazovaných Voroného buněk značně neefektivní a již při jejich zobrazování nad relativně malými proteinovými strukturami (např. 1CQW [8], vizte sekci 4.3 kapitoly o dosažených výsledcích) ztrácí svou responzivitu, což může vést k ne zcela příjemnému používání.



Obrázek 2.3: Hlavní okno aplikace Voroprot

2.3.2 BetaMol

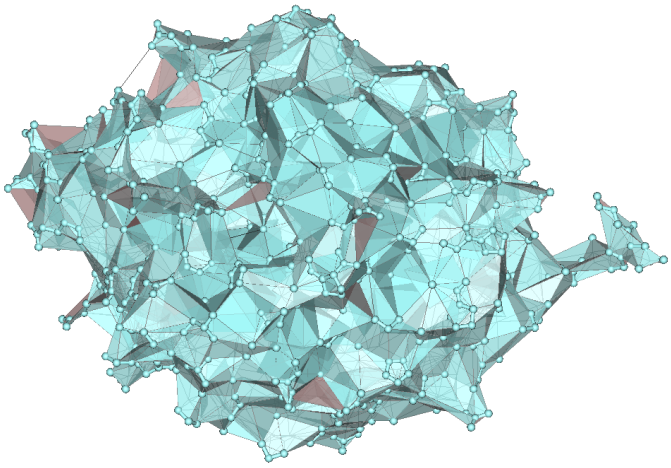
BetaMol je další aplikací vizualizující statické proteinové struktury [28]. Aplikace v kombinaci s aditivně váženými Voroného diagramy využívá *kvazi-triangulaci*, která je jejich duální strukturou.

Voroného diagramy pak BetaMol nijak nevizualizuje, poskytuje pouze zobrazení tzv. *Beta-shape* (obr 2.4) – podmnožiny kvazi-triangulace sestáva-

jící z vybraných tetrahedronů a jejich částí podle uživatelsky nastavitelných kritérií.

Aplikace podporuje nastavení průhlednosti některých částí vizualizace. Dále je možné vybrat kritérium, podle kterého se nastavují barvy dané vizualizace, podle barvy atomu, rezidua, řetězci či modelu, ke kterému daná část vizualizace náleží. Z těchto barev lze uživatelsky upravovat pouze barvu modelu, ostatní barvy jsou aplikací pevně předurčeny.

Vizualizace programu BetaMol nejsou příliš výpočetně efektivní a při zobrazování větších molekulárních struktur, jako je např. 1AON [27], která obsahuje zhruba 59 000 atomů, dochází k vážné ztrátě responzivity, kdy se rychlost vykreslování pohybuje až u jednotek snímků za sekundu, a to i při použití výkonnější dedikované GPU[1].



Obrázek 2.4: Beta-shape vizualizovaný programem BetaMol

2.3.3 Voro++ a POV-Ray

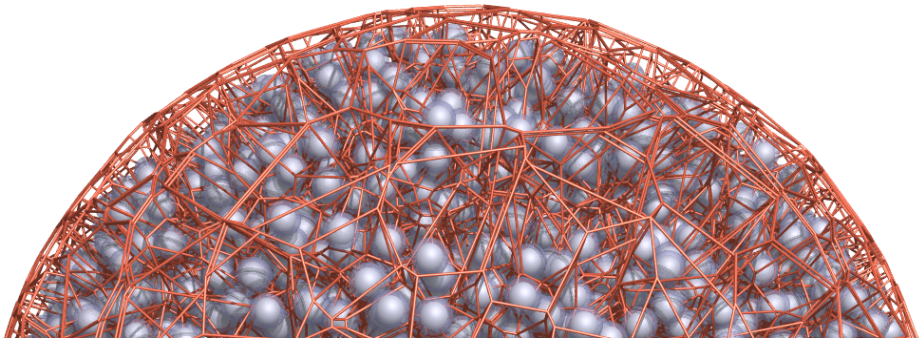
Voro++ je softwarová knihovna pro výpočet Voroného diagramů nad zadanou množinou bodů, resp. sfér stejné velikosti. Knihovna samotná umožňuje pouze výpočet euklidovských Voroného diagramů a její výstup je nutno vizualizovat pomocí externích nástrojů. Ke knihovně je také dodáván nástroj pro příkazový řádek, který dokáže přistupovat k většině funkcionality knihovny [12, 13].

Nástroj pro příkazový řádek může výsledný Voroného diagram vygenerovat např. v podobě *Scene description language* programu *POV-Ray*. Scene description language tohoto programu je jazyk kompletní v Turingově

[1]Testováno na nVidia GeForce GTX 960M

smyslu, kterým lze popsat libovolnou scénu, včetně nastavení materiálů, průhledosti prvků scény apod.

POV-Ray je úspěšný vizualizační nástroj, který je aktivně vyvíjen a udržován již od roku 1991 [23]. V programu lze vykreslit libovolnou popsanou scénu s využitím metody sledování paprsku (ray-tracing). Jak lze vidět na obrázku 2.5, jeho výstupní vizualizace je kvalitní a vizuálně působivá, není však interaktivní a hodí se tak spíše pro ilustraci ve vědeckých článcích, nikoliv k výzkumu vlastností Voroného diagramů.



Obrázek 2.5: Euklidovský Voroného diagram spočítaný programem Voro++ a vizualizovaný programem POV-Ray.

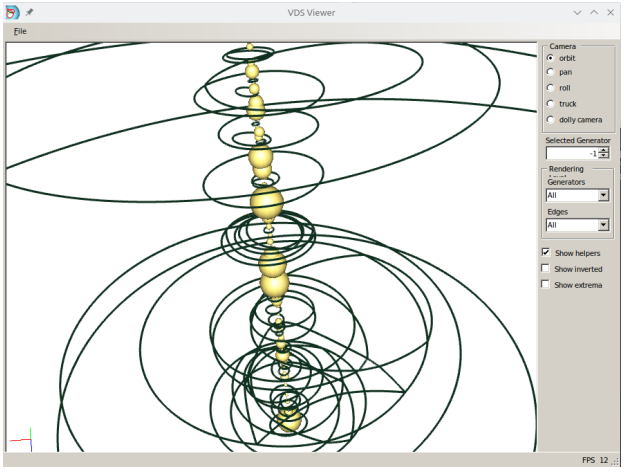
2.3.4 Knihovna awVoronoi a program VDS Viewer

VDS Viewer je jednoduchý nástroj pro prohlížení statických aditivně vážených Voroného diagramů. Je poskytován jako doplňková aplikace ke knihovně *awVoronoi* vyvíjené na Fakultě aplikovaných věd Západočeské univerzity v Plzni [7]. Tato knihovna je rovněž použita v aplikaci CAVER Analyst pro výpočet dutin a tunelů v molekulách.

Jak lze vidět na obrázku 2.6, Voroného hrany jsou v programu VDS Viewer vizualizovány jednoduchými jednobarevnými čarami knihovny OpenGL pomocí evaluátorů křivek a obsahuje základní funkcionalitu zvýraznění Voroného regionů podle jednoho vybraného identifikátoru Voroného sféry. Vykreslování křivek však není velmi efektivní a u větších modelů ztrácí responzivitu. Jednoduché čáry knihovny OpenGL mají při vykreslování pevnou šířku danou počtem pixelů – neberou v potaz perspektivu. Složitější Voroného diagramy tak mohou být značně nepřehledné.

2.3.5 BetaConcept

Program BetaConcept je nástroj pro vizualizaci dvojrozměrných aditivně vážených Voroného diagramů [5]. Jeho hlavním využitím je snadná tvorba



Obrázek 2.6: Okno programu VDS Viewer

ilustrací pro vědecké články. Jedním z příkladů výstupní vizualizace z programu BetaConcept je obrázek 2.1 v sekci 2.1 o Voroného diagramech.

Některé vlastnosti dvojrozměrných Voroného diagramů je možné zobecnit i do trojrozměrného prostoru. BetaConcept tak lze v omezené míře využít ke zkoumání těchto zobecnitelných vlastností ve zpravidla přehlednější dvojrozměrné podobě.

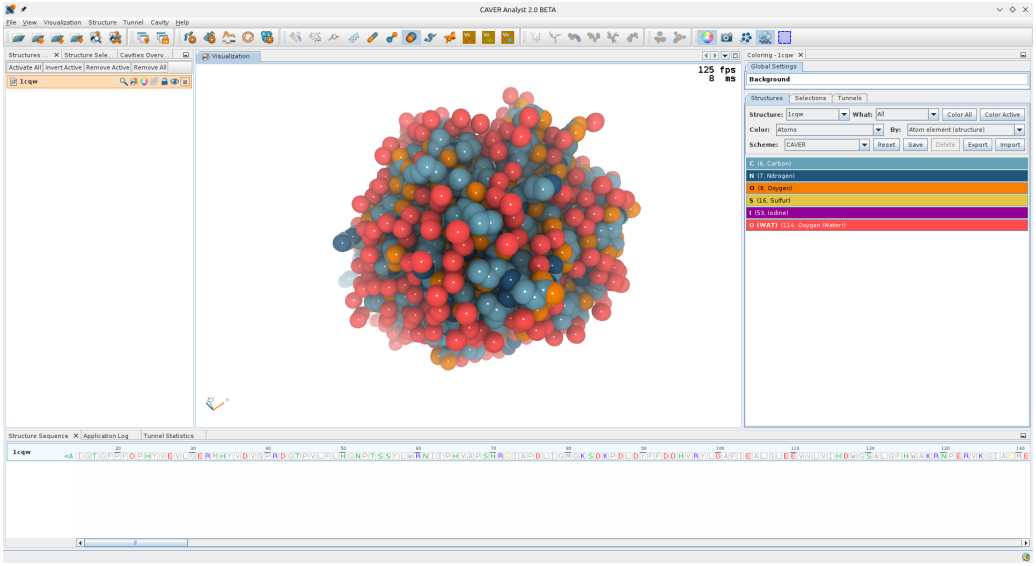
2.4 Užité technologie

Jedním z cílů práce je implementace vizualizace aditivně vážených Voroného diagramů v podobě rozšiřovacího modulu pro aplikaci CAVER Analyst. Je proto vhodné prozkoumat a zhodnotit aplikací používané technologie a aplikaci samotnou již během analýzy, neboť již ze samotného zadání mohou vyplynout omezení daná právě těmito předem určenými technologiemi.

2.4.1 CAVER Analyst

CAVER Analyst je softwarová aplikace pro analýzu a vizualizaci tunelů, kanálů a dutin ve strukturách makromolekul, zejména proteinů [6]. Je implementována v programovacím jazyce Java 1.8 a využívá platformu NetBeans verze 7.3.1. Grafické uživatelské rozhraní aplikace a vzhled vizualizace van der Waalsova modelu molekuly je vidět na obrázku 2.7.

NetBeans poskytuje rozhraní pro vykreslování grafického uživatelského prostředí, jež je rozšířením knihovny Swing ze standardní knihovny jazyka Java, a základní rozhraní pro implementaci rozšiřujících modulů. Tento systém modulů je aplikací plně využíván – CAVER Analyst skrze něj poskytuje



Obrázek 2.7: Hlavní okno aplikace CAVER Analyst 2.0

vlastní rozhraní a je tedy možné jeho funkčnosti dále rozšiřovat, čehož bylo využito k implementaci této práce.

Pro své vizualizace aplikace využívá knihovnu OpenGL. Při vykreslování atomů molekuly se využívá možnosti tzv. *instancování*, kdy CPU pro GPU připraví pouze několik modelů sfér, spojnic či jiné potřebné geometrie pro různé úrovně detailů a GPU při vykreslování scény tyto již připravené modely opakovaně používá a pouze je modifikuje na základě parametrů, jako jsou např. pozice atomu, van der Waalsův poloměr, barva a další. CPU tak nemusí připravovat geometrii pro každý atom zvlášť, čímž se významně snižují nároky na přenos dat mezi CPU a GPU, a vykreslování je díky tomu rychlé.

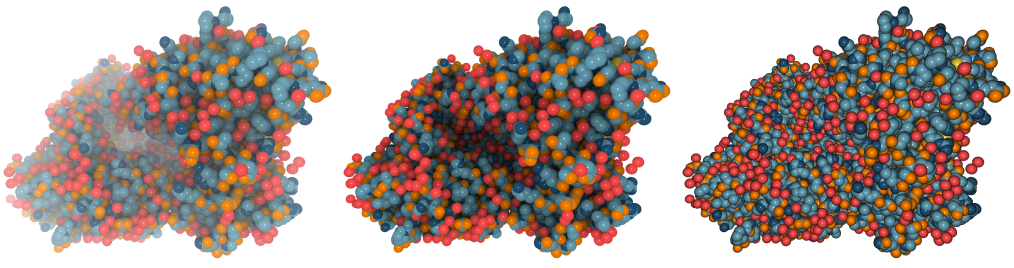
Rychlost vizualizace je důležitá i pro dynamiku modelů molekul, kterou CAVER Analyst podporuje a umožňuje její přehrávání v reálném čase. Dynamika modelu představuje pohyb jednotlivých atomů molekuly, který byl vypočten složitou fyzikální simulací a uchován v podobě diskrétních snímků. Pro zvýšení plynulosti animace navíc aplikace mezi snímky interpoluje pozice atomů pomocí Hermitovy křivky. Jak se atomy pohybují v prostoru, mění se i charakter celého modelu molekuly, tvary tunelů a dutin.

K výpočtu tunelů a dutin molekul využívá aplikace CAVER Analyst aditivně vážené Voroného diagramy, které konstruuje za použití knihovny *awVoronoi* vyvíjené na Fakultě aplikovaných věd Západočeské univerzity v Plzni [7]. Jelikož je konstrukce aditivně váženého Voroného diagramu výpočetně náročnou operací, která může trvat v rámci jednotek až desítek sekund a u velikých vstupních množin i mnohonásobně více, je v rámci za-

chování responzivity aplikace při přehrávání dynamiky nutné diagramy pro jednotlivé snímky dynamiky vypočítat předem. Topologická data diagramů jsou však veliká a nelze je tak držet v operační paměti pro všechny snímky dynamiky. Knihovna awVoronoi umožňuje CAVER Analystu načítání těchto předem vypočtených diagramů, které jsou uloženy v souboru komprimovaném proprietárním algoritmem, a to jak sekvenčně, tak s náhodným přístupem. Uživatel si tedy může vybrat libovolný snímek dynamiky a program tento snímek načte a dekomprimuje v čase nezávislém na vzdálenosti od předchozího vybraného snímku. V době psaní této práce je však modul pro dekompresi rovněž ve fázi vývoje a přestože je již tato dekomprese funkční, je zatím velmi pomalá a neumožňuje tak načítání snímků diagramu v reálném čase.

CAVER Analyst využívá některé grafické efekty obecně známé spíše ze zábavního softwaru – zde však značně zpřehledňují vizualizovanou informaci, jelikož je lidský mozek schopen tyto efekty interpretovat jako prostorový vjem. Prvním efektem je *mlha*, která barvu objektů vzdálenějších od kamery mísí s uživatelsky volitelnou barvou pozadí. Tento efekt je aplikován globálně na celou vykreslovanou scénu a bez dalších nutných úprav automaticky ovlivňuje i vizualizace dodávané rozšiřujícími moduly.

Druhým efektem je *zastínění okolím* (*ambient occlusion*). Ten simuluje chování tzv. ambientního světla, tj. světla, jehož zdrojem je celé okolí modelu (oproti např. bodovému zdroji umístěnému v prostoru). Výsledkem jsou jemné stíny na rozhraní dotýkajících se atomů a „zatemnění“ atomů, které jsou dále od okraje molekuly. Tento efekt je aplikován staticky, tj. při načítání molekuly je předem vygenerována textura obsahující data o jejím osvětlení. V případě, že bychom jej chtěli použít pro vizualizaci Voroného diagramů, bylo by nutné jej pro tento účel implementovat zvlášť, nebo za cenu snížení grafické věrnosti využít data o osvětlení atomů. Na obrázku 2.8 lze vidět, jak efekty ovlivňují vnímání prostoru ve scéně.



Obrázek 2.8: Porovnání vizualizací s efektem zastínění okolím a mlhou (vlevo), se zastíněním okolím bez mlhy (uprostřed) a s jednoduchým stínováním (vpravo).

2.4.2 OpenGL

CAVER Analyst používá k vizualizaci grafickou knihovnu *OpenGL*, resp. její *binding* pro jazyk Java nazvaný *JOGL*. Tato knihovna poskytuje aplikacím rozhraní pro hardwarovou akceleraci grafických výpočtů a primárně pracuje s grafikou na bázi geometrie trojúhelníkových sítí. Základními vstupními daty knihovny jsou pole vrcholů (tzv. *vertexů*), které jsou zasazeny do geometrických primitiv – těmi mohou být jednotlivé body, úsečky, mnohoúhelníky a nejčastěji pak trojúhelníky.

Zasazení do primitiv může být dáno více způsoby. Jedním způsobem je podle pořadí vertexů v poli – tj. pokud kreslíme například diskrétní trojúhelníky, první tři vertexy tvoří první trojúhelník, další tři tvoří druhý trojúhelník atd. Můžeme kreslit i trojúhelníky se společnými hranami (tzv. *triangle strip*), kdy první tři vertexy tvoří první trojúhelník a každý další vertex dá vzniknout novému sousedícímu trojúhelníku.

Další způsob, jakým lze definovat primitiva, je pomocí indexace – mějme dvě pole, v prvním poli se nacházejí pozice vertexů a v druhém poli se nachází posloupnost jejich indexů. Tato posloupnost pak určuje pořadí, ve kterém dochází ke spojování vertexů do primitiv v podobném duchu, jako u předchozího způsobu.

Vertex může kromě své pozice obsahovat i další uživatelsky definovaná data pro zpracování v různých fázích *vykreslovací sekvence* (*rendering pipeline*) [18]. Tato data se skládají z tzv. *komponent*. Mezi podporované datové typy patří například celá (integer) či reálná čísla (float), logické hodnoty (boolean), či vektory z předchozích typů hodnot složené. Ty pak nabývají velikosti podle počtu svých členů, tedy dvojrozměrný vektor je složen ze dvou komponent, trojrozměrný vektor je složen ze tří komponent atd.

Primitiva složená z vertexů jsou při vykreslování podrobena rasterizaci. Rasterizátor generuje *fragmenty* obrazu, které si lze zjednodušeně představit jako pixely na obrazovce. To však není zcela přesnou charakteristikou fragmentu. Například při použití multisample anti-aliasingu může fragment představovat granulárnější vzorek, který je až poté použit pro výpočet výsledné barvy pixelu, zpravidla aritmetickým průměrem barev fragmentů pixelu náležících.

Rozhraní knihovny OpenGL a její vykreslovací kontext tvoří dohromady stavový automat. Před samotným vykreslením prvků scény je vždy nutno pomocí poskytovaných funkcí nastavit tento automat do požadovaného stavu. Stavem jsou například buffery obsahující vstupní vertexy, informace o požadovaném typu vykreslovaných primitiv, textury, použité shadery apod. Některé části stavu může být nutné aktualizovat při přípravě každého snímku

– to mohou být například animovaná geometrická data. Jiné části, zpravidla například shadery, kterým se budeme detailně věnovat v následující podsekci, se připraví jednou před spuštěním vizualizace a při zpracovávání jednotlivých snímků se opakovaně používají. Volba toho, které součásti stavu jsou připraveny předem a které jsou připravovány při zpracovávání každého snímku, je na vývojáři každé aplikace – různé přístupy pak mohou mít své výkonnostní implikace.

Když je automat v požadovaném stavu, spustí se *vykreslovací volání* (*draw call*), které aktuální stav automatu označí za úplný a vyvolá signál, že může dojít k vykreslení daných prvků scény, tj. může být spuštěna dříve zmíněná vykreslovací sekvence.

Samotné vykreslení scény na obrazovku je zodpovědností ovladače grafického čipu, resp. konkrétní systémové implementace knihovny OpenGL. Zpravidla k němu nedochází přesně v době vykreslovacího volání – to pouze „uzamyká“ aktuální stav automatu pro dané elementy scény. Například při nastavení kontextu pro použití tzv. *vertikální synchronizace* může být scéna vykreslována na tzv. back-buffer, tj. části paměti reprezentující pixely aktuálně *zpracovávaného* snímku. Když je snímek „hotový“, vyčká se na synchronizační signál od monitoru a po něm dojde k výměně back-bufferu s front-bufferem, který je částí paměti reprezentující pixely aktuálně *zobrazovaného* snímku – po této výměně je možné do back-bufferu (který byl doposud front-bufferem) začít vykreslovat nový snímek.

Shadery

Shader je uživatelský program běžící na grafickém čipu (GPU), který určuje chování některých fází vykreslovací sekvence. Například při zpracování vstupní geometrie mohou shadery tuto geometrii transformovat; při rasterizaci mohou určovat barvu fragmentů a vytvořit tak například efekt osvětlení.

Jedním ze specifik shaderů oproti klasickým programům běžícím na CPU je způsob paralelního zpracování velkého množství stejnorodých dat. V případě, že shader napsán tak, aby zpracovával všechna data zcela stejně, tj. totožnou posloupností instrukcí bez jakékoliv divergence v podobě podmíněných skoků, aplikuje se tzv. model *Single Instruction, Multiple Data* (*SIMD*) – toto zjednodušeně znamená, že existuje jeden řídicí procesor, který načítá instrukce z paměti a orchestruje velké množství matematických jednotek, které tyto instrukce provádějí, každá nad různými instancemi stejnorodých datových struktur [20]. Tímto způsobem lze zpracovávat velké množství dat simultánně, což vede k zásadnímu zrychlení jejich zpracovávání oproti pří-

padu, kdy by tato data byla zpracovávána na CPU.

OpenGL pro psaní shaderů specifikuje jazyk GLSL (OpenGL Shading Language). Ten je obdobou jazyka C s jistými specifickými vlastnostmi. Jazyk obsahuje speciální konstrukce pro psaní programů pro grafické čipy, jako jsou vestavěné operace pro práci s vektory. Vstupní data určená k transformaci jsou shaderům předávána v podobě globálně definovaných *in* proměnných, pro výstupní data se definují *out* proměnné. Ke globální parametrizaci shaderu slouží tzv. *uniform* proměnné, jejichž hodnoty jsou společné pro všechna vstupní data – podmíněné skoky využívající výhradně uniform proměnné jsou považovány za staticky vyhodnotitelné a nebrání tak ve vykonávání shaderu podle modelu SIMD.

Shadery jsou společně s aplikacemi využívajícími knihovnu OpenGL distribuovány v podobě zdrojových souborů jazyka GLSL. Jejich překlad, sestavení a nahrání do paměti GPU je vždy odpovědností ovladače grafického čipu.

OpenGL podporuje několik typů shaderů – dvěma základními typy jsou *vertex shader* a *fragment shader*. *Vertex shader* [22] slouží k transformaci vstupních vertexů geometrického modelu, například k rotaci a translaci podle aktuální pozice „kamery“. Nelze jím však vytvářet nové vrcholy či primitiva. *Fragment shader* [16] ve fázi rasterizace určuje barvu fragmentu.

V této práci je použit ještě třetí typ shaderu. Je jím *geometry shader* [17], který dokáže z jednotlivých geometrických primitiv (bodů, úseček, trojúhelníků či čtyřúhelníků) po transformaci jejich vrcholů *vertex shaderem* vygenerovat zcela novou geometrii; tj. na základě těchto geometrických primitiv a jejich vrcholů je možné tímto shaderem vypočítat vrcholy nové a zasadit je do zcela nových geometrických primitiv. Počet výstupních vrcholů přitom nemusí odpovídat počtu vrcholů vstupních a zrovna tak si nemusí odpovídat typy primitiv na vstupu a výstupu.

Možnosti geometry shaderu jsou omezené: lze vytvářet pouze několik konkrétních typů primitiv (oproti tomu, co běžně nabízí rozhraní OpenGL) a počet výstupních vrcholů a úhrn jejich komponent je omezen ovladačem grafického čipu, resp. specifikací od výrobce[2].

Textury

Textura je v knihovně OpenGL sada datových polí, která mohou nabývat až tří rozměrů [21]. Data v těchto polích jsou často obrazového charakteru a jsou při rasterizaci používána fragment shadery k určení barev částí geome-

[2]např. čip nVidia GeForce GTX 960M, použitý při implementaci této práce, umožňuje generování maximálně 1024 komponent na jedno vstupní primitivum

trického modelu k navození dojmu, že je model tvořen nějakým materiálem. Ne vždy však musí textura mít charakter obrázku. Textura může být držitelem libovolných dat, jejichž účel je určen shaderem, který ji využívá. Prvky textury, ať už definují barvu či jiný typ dat, se nazývají *texely* (zkratka pojmu *texture element*).

Důležitou vlastností textury je fakt, že je možné ji v libovolném shaderu použít jako globální zdroj dat nezávislý na aktuálně zpracovávaném vrcholu, primitivu či fragmentu.

Shadery mohou k textuře přistupovat pomocí proměnných typu *sampler* [19], a to více způsoby. Je možné získávat přímo hodnoty jednotlivých texelů na základě jejich indexů. Dále lze k textuře přistupovat pomocí normalizovaných souřadnic, kde se velikost textury na dané souřadnici mapuje na interval $\langle 0; 1 \rangle$. Je možné použít souřadnice i mimo tento interval, API OpenGL umožňuje nastavení chování textury pro tyto případy – texturu je možné v tomto normalizovaném prostoru opakovat jako dlaždici s případným zrcadlením, či při překročení intervalu použít hodnotu hraničního texelu. Při použití normalizovaných souřadnic knihovna OpenGL automaticky zařizuje interpolaci mezi texely podle nastavitelných parametrů.

2.5 Analýza

Současná implementace programu CAVER Analyst poskytuje API pro získání topologie a geometrie aditivně váženého Voroného diagramu. Mezi tato data patří pozice Voroného vrcholů a definice Voroného hran v podobě kvadratických racionálních Bézierových křivek na základě identifikátorů těchto vrcholů. Zaměřme se na Voroného hrany. Pro jejich úspěšnou vizualizaci je nutné tato data transformovat do takové podoby, aby bylo možno diagram za pomoci knihovny OpenGL co nejvěrněji a výpočetně co nejefektivněji vizualizovat.

Triviálním řešením může být generování geometrie vzorkovaných Bézierových křivek na CPU. Výhodou je snadná implementace a takřka neexistující omezení, co se týče komplexity geometrie – křivku lze navzorkovat s téměř jakoukoliv granularitou, omezenou pouze paměťovými prostředky počítače. Voroného diagram s velkými vstupními daty však může obsahovat desítky až stovky tisíc hran. To je problematické nejen kvůli dlouhé době vzorkování, ale především kvůli velkému množství geometrických dat, která je nutno přenášet z hlavní paměti do paměti GPU pro následnou rasterizaci a vykreslení na obrazovku. Přenos dat mezi těmito paměťmi je zpravidla pomalý a navyšuje tak dobu zpracování snímku. Pokud navíc chceme u di-

agramu vytvořit efekt perspektivy, tj. aby se bližší hrany jevily silnější než hrany vzdálenější, je potřeba generovat geometrii křivek v podobě jakýchsi „válečků“, čímž se dále zvyšuje objem dat přenášené geometrie. Jak se ukazuje například i u zkoumané aplikace Voroprot, vizualizace Voroného diagramu tímto způsobem je výkonnostně problematická už pro statická data. Jelikož je cílem této práce vizualizovat kromě statických i dynamické Voroného diagramy, je nutné nalézt značně efektivnější řešení. Bylo by vhodné minimalizovat práci prováděnou na CPU a objem přenášených dat z hlavní paměti do paměti grafické karty.

Grafická karta může kromě rasterizace geometrie poskytnuté z hlavní paměti generovat i geometrii vlastní. Její výhodou oproti hlavnímu procesoru je masivní paralelizace shaderů na základě modelu SIMD, díky níž je schopna zpracovávat velké množství stejnorodých dat naráz (vizte podsekci 2.4.2). Stejnorodá mohou být i data kvadratických racionálních Bézierových křivek – ty lze vždy jednoznačně vyjádřit pomocí tří kontrolních bodů a jejich vah. Při použití *geometry shaderu* je tak teoreticky možné do paměti GPU přenášet pouze tyto kontrolní body, přičemž generování lomené čáry aproximující křivku a její následný převod na geometrii „válečku“ pro efekt perspektivy může být prováděno přímo na GPU.

Omezení geometry shaderu popisovaná v předchozí kapitole však nemusí vždy umožňovat vzorkování křivky s optimálním množstvím vzorků, aby výsledná aproximace byla vizuálně dostatečně přesná. Problém může nastat především u hran s velkým zakřivením. U běžných molekulárních dat zpravidla takovéto hrany nenajdeme a vzorkování samotným shaderem by tak mělo být dostatečné, avšak u dat vytvořených speciálně pro výzkum chování Voroného diagramu v určitých podmínkách se objevit mohou a je potřeba s nimi počítat.

V prvé řadě je proto nutné omezit počet komponent vertexu na minimum, aby bylo možné vertexů shaderem generovat co nejvíce. Dále by bylo vhodné najít řešení, jak zvýšit věrnost vizualizace s co nejmenším dopadem na rychlost zpracování. Možným řešením je kombinace dvou dříve zmiňovaných přístupů, tj. na CPU dle potřeby rozdělit křivky na několik částí, které budou poté detailněji navzorkovány pomocí GPU. Tímto způsobem by bylo možné vizualizovat křivku přesněji, ale držet objem dat přenášených mezi hlavní a grafickou pamětí na minimu. K tomu nám může s drobnými modifikacemi pomoci *algoritmus de Casteljau* popsaný v sekci 2.2.

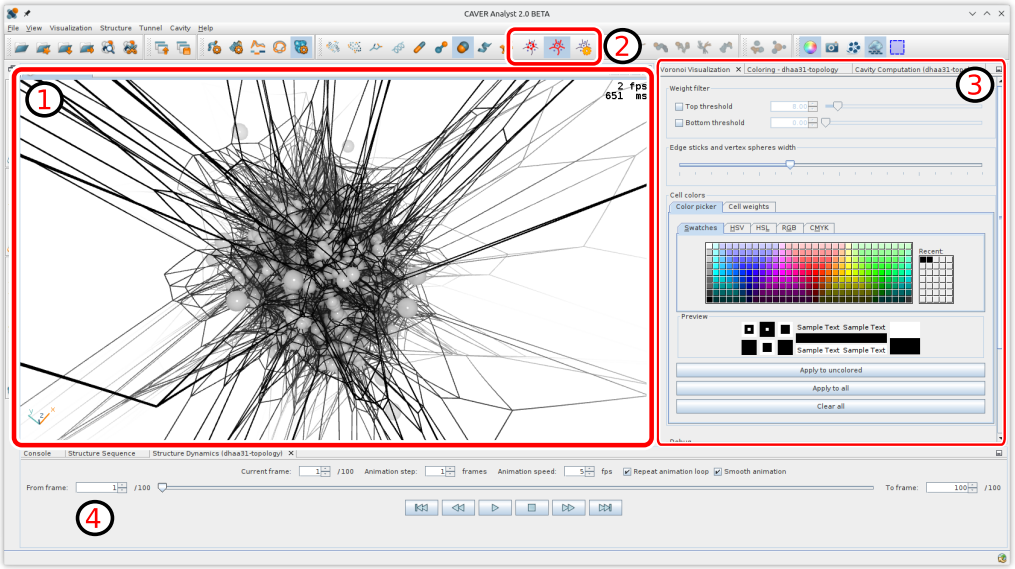
Pro maximální využití výkonnostních výhod modelu SIMD na GPU je nutné se při psaní shaderů vyvarovat dynamickým podmíněným větvením programu, neboť by došlo k divergenci posloupností instrukcí pro různá zpracovávaná data a řídicí procesor by tak nemohl ke všem řízeným ma-

tematickým jednotkám přistupovat stejně. Z tohoto důvodu by bylo využití algoritmu de Casteljau i při vzorkování křivky v rámci geometry shaderu zásadní chybou. Mezi vhodné techniky generování geometrie křivky může patřit zvolení pevného a pro všechny hrany jednotného počtu hodnot parametru t, z něhož je poté navzorkován stejný počet bodů křivky. Naivně lze zvolit hodnoty parametru rovnoměrně rozložené v intervalu $\langle 0; 1 \rangle$, avšak u křivek, u nichž si nejsou rovny váhy kontrolních bodů, může dojít ke kumulaci výsledných vzorků blíže kontrolním bodům s vyššími vahami. Pro zvýšení věrnosti vizualizace by proto bylo vhodné tento jev kompenzovat jiným než rovnoměrným rozložením parametru t.

3 Realizační část

Na základě analytické části této práce byl v jazyce Java implementován rozšiřující modul pro aplikaci CAVER Analyst 2.0 s názvem *Voronoi Visualizer*, schopný zobrazovat hrany a vrcholy aditivně vážených Voroného diagramů sestrojených nad dynamickými modely molekul.

3.1 Uživatelský pohled na Voronoi Visualizer



Obrázek 3.1: Hlavní prvky modulu Voronoi Visualizer z pohledu uživatele: (1) Zobrazení Voroného diagramu, (2) Přepínače pro aktivaci a deaktivaci vizualizace a panelu nastavení, (3) Panel nastavení modulu, (4) Modul reaguje na přehrávání animace dynamické molekuly.

Po instalaci rozšiřujícího modulu Voronoi Visualizer si lze všimnout několika nových ovládacích prvků. V horní části obrazovky se nyní nacházejí tři nová tlačítka (obrázky 3.1 a 3.2). První dvě tlačítka zleva slouží k aktivaci dvou implementovaných vizualizací Voroného diagramu – vizualizace vrcholů v podobě kuliček a vizualizace hran v podobě zakřivených válečků. Třetí tlačítko slouží k otevření ovládacího panelu nastavení modulu Voronoi Visualizer.

Panel nastavení, implementovaný pomocí knihovny Swing, lze vidět na obrázku 3.3. Obsahuje ovládací prvky, díky kterým lze interaktivně upravo-

Obrázek 3.2: Přepínací tlačítka modulu Voronoi Visualizer: (zleva) vizualizace vrcholů, vizualizace hran, panel nastavení

vat přehlednost obou vizualizací a zvýrazňovat pro uživatele důležité částí diagramu. V horní části tohoto panelu se nachází sekce s nastavením filtru podle vah Voroného sfér (*Weight filter*), kde lze nastavit jejich horní a dolní práh. Tento filtr je v základním nastavení vypnutý a lze jej zapnout zaškrtnutím alespoň jednoho z příslušných zaškrtávacích políček.

Pod nastavením tohoto filtru se nachází sekce s posuvníkem, pomocí kterého lze nastavit poloměr vizualizovaných hran a vrcholů pro možné zpřehlednění vizualizace.

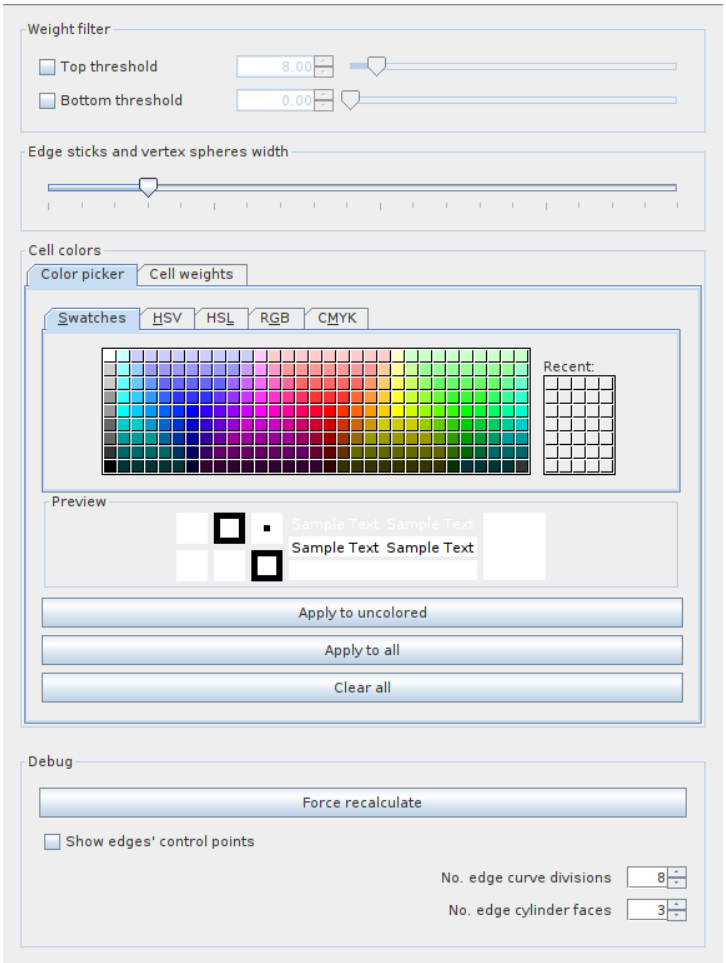
Třetí sekce (*Cell colors*) slouží k nastavení obarvení vizualizace. Obsahuje dvě karty, na základě jejichž výběru je u vizualizací nastaven režim obarvování. Prvním režimem (*Color picker*) je obarvování podle uživatelského výběru barev. V tomto režimu lze použít již existující funkci výběru atomů aplikace CAVER Analyst k označení částí molekuly – dojde tak zároveň ke skrytí částí Voroného diagramu, které nenáleží vybraným atomům – a pomocí ovládacího prvku pro výběr barev a příslušných tlačítek pod ním aplikovat vybrané barvy na danou část Voroného diagramu. Druhý režim (*Cell weights*) umožňuje obarvení částí diagramu podle vah Voroného sfér. Lze zde definovat libovolný počet prahů a jim náležící barvy – podle těchto prahů je pak obarvena výsledná vizualizace.

Poslední sekce (*Debug*) slouží především k testovacím účelům. Nachází se zde tlačítko pro vynucení vygenerování nových geometrických dat pro vizualizaci Voroného diagramu, zaškrtávací políčko pro zobrazení kontrolních bodů křivek a číselná políčka pro nastavení parametrů detailů vizualizovaných Bézierových křivek.

3.2 Rozšíření aplikace CAVER Analyst

Platforma NetBeans, jež je základním stavebním kamenem aplikace CAVER Analyst, poskytuje nástroje pro dynamické rozšiřování funkcionality aplikací. To znamená, že uživatel může získávat a instalovat balíčky rozšiřovacích modulů bez nutnosti opětovného sestavování hlavní aplikace. Modul *Voronoi Visualizer* má podobu takového rozšiřujícího balíčku.

Pro podporu rozšiřitelnosti je uzpůsobeno i rozhraní samotné aplikace CAVER Analyst. V tomto ohledu nás zajímá její tzv. *rozšiřitelný model*,



Obrázek 3.3: Panel nastavení modulu Voronoi Visualizer

který modulům umožňuje rozšiřování makromolekulárních struktur o vlastní data; a rozhraní *vykreslovacích strategií*, jehož vlastní implementací je možné přidávat vlastní prvky do vizualizované scény.

3.2.1 Rozšiřitelný model

Rozšiřitelný model aplikace CAVER Analyst umožňuje externím modulům přidružování vlastních dat k již existujícím datům molekulárních struktur. Aplikace k tomuto účelu zavádí jednoduché rozhraní (`Extension-TargetProvider`), jehož implementace mají za úkol uchovávat a poskytovat tato rozšiřující data dle individuálních potřeb rozšiřujícího modulu. Instance těchto implementací jsou poté agregovány v blíže nespecifikované abstraktní datové struktuře umožňující jejich vyhledávání na základě identifikátorů datových typů z rozhraní pro reflexi tříd platformy Java. Tento systém umož-

ňuje dynamické rozšiřování datových struktur aplikace CAVER Analyst, aniž by bylo nutné v aplikaci zavádět jakoukoliv formu přímé závislosti na rozšiřujících modulech.

Modul Voronoi Visualizer k datům o molekule tímto způsobem přidává vlastní třídu `VoronoiVisualizationExtension` poskytující data o geometrii Voroného diagramu, především pak o kontrolních bodech zakřivených Voroného hran. Tato data jsou modulem vygenerována z topologických dat knihovny awVoronoi, kterou CAVER Analyst používá ke konstrukci Voroného diagramů.

Kromě samotných geometrických dat poskytuje třída `VoronoiVisualizationExtension` rovněž uživatelsky nastavitelné parametry vizualizace, jako jsou prahy filtru, poloměr hran, data obarvení a další (více dále v příslušných sekcích).

3.2.2 Rozhraní vykreslovacích strategií

Aplikace CAVER Analyst používá pro své vizualizace rozhraní tzv. *vykreslovacích strategií*. Metody tohoto rozhraní aplikace centrálně volá při vykreslování scény, přičemž jim poskytuje aktuálně používaný kontext knihovny OpenGL. Ten vnitřně drží informaci o vykreslovací ploše (v tomto případě část hlavního okna aplikace, kde dochází k vykreslování vizualizací aplikace) a další stavové hodnoty související s hardwarově akcelerovanou grafikou. Systém vykreslovacích strategií umožňuje rozšiřovacím modulům přidávat vlastní vizualizace, které tak mohou být součástí již existující scény.

Modul Voronoi Visualizer implementuje dvě takovéto strategie. První strategie vizualizuje vrcholy Voroného diagramu sestrojeného nad aktuálně vybranou makromolekulární strukturou v podobě kuliček. Strategie využívá aplikací poskytovaného obecného základu strategie pro vykreslování barevných sfér, který umožňuje jejich vykreslování pouze na základě udání středů a poloměrů. Sféry jsou vykreslovány v dávkách pevně určené velikosti, s využitím pseudo-instancování, které implementovaná strategie využívá ke zvýšení efektivity vykreslování.

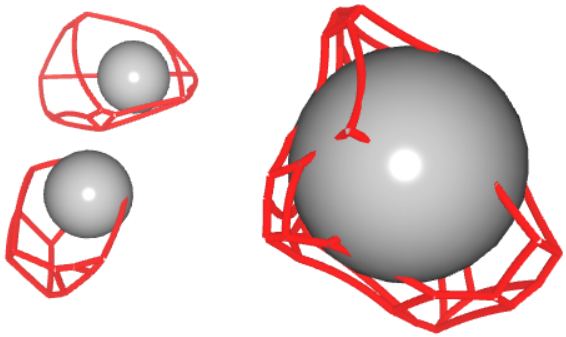
Druhá strategie vizualizuje Voroného hrany v podobě válečků opisujících dané kvadratické racionální Bézierovy křivky. Při implementaci byl kladen největší důraz na vývoj právě této strategie. K vykreslování křivek je využíván geometry shader, pomocí kterého je většina geometrie generována přímo na GPU, díky čemuž je vykreslování velmi efektivní. O této strategii pak detailně pojednává následující sekce 3.3.

Obě strategie využívají nově implementované rozšíření dat `VoronoiVisualizationExtension`.

3.3 Kvadratické racionální Bézierovy křivky

Primárním účelem třídy `VoronoiVisualizationExtension` je příprava geometrických dat do struktury, která umožní její co nejrychlejší vykreslování. Pro vykreslovací strategii vizualizující hrany aditivně vážených Voroného diagramů připravuje data o jednotlivých kvadratických racionálních Bézierových křivkách, jmenovitě jejich kontrolní body a identifikátory Voroného vrcholů, resp. Voroného sfér.

K výpočtu geometrie kvadratické racionální Bézierovy křivky v trojrozměrném prostoru používá vykreslovací strategie *geometry shader*. Ten tvoří výslednou geometrii pouze na základě kontrolních bodů tak, že křivku navzorkuje, ale místo segmentů lomené čáry vygeneruje *válečky* – geometrické útvary složené ze sekcí podobných n-bokým hranolům bez podstav. Tyto válečky zajišťují vykreslování s korektním efektem perspektivy, tj. křivky vzdálenější od kamery se jeví užší než ty, které jsou kameře blíže, což je demonstrováno na obrázku 3.4. Počet stěn těchto válečků je dán parametrem geometry shaderu a lze jej měnit v sekci *Debug* v panelu nastavení.

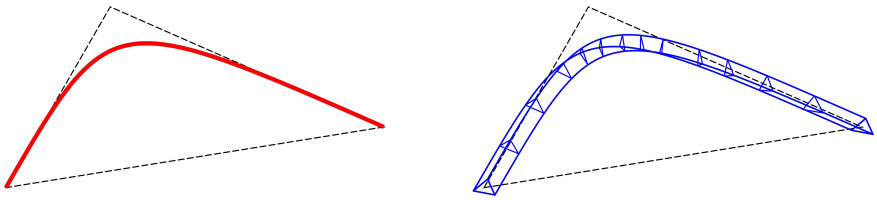


Obrázek 3.4: Efekt perspektivy u izolovaných Voroného regionů

Vstupem shaderu je trojúhelníkové primitivum, jehož tři vrcholy představují kontrolní body kvadratické racionální Bézierovy křivky. Vrcholy mají podobu čtyřrozměrných vektorů, jejichž čtvrtý prvek představuje váhu w_i kontrolního bodu P_i.

Do lokálních pomocných polí velikosti počtu vzorků jsou shaderem vygenerovány vzorky křivky, tj. pozice bodů na křivce ležících a normalizované vektory představující tečny ke křivce v těchto bodech. Počet vzorků je dán parametrem shaderu a lze jej měnit v sekci *Debug* v panelu nastavení. Z těchto dat poté shader generuje pozice vrcholů, které po propojení do trojúhelníkových primitiv tvoří prostorové válečky aproximující danou Bézierovu křivku. Tloušťka válečků je určena uniform proměnnou v geometry

shaderu a lze ji ovlivnit posuvníkem v panelu nastavení. Proces generování válečků je ilustrován na obrázku 3.5.



Obrázek 3.5: Ilustrace funkce implementovaného geometry shaderu: vstupem je trojúhelníkové primitivum (vyznačené čárkovaně), ze kterého je navzorkována Bézierova křivka (červeně vlevo) a z ní je poté vygenerována geometrie *válečku* (modře vpravo)

Jelikož geometry shader může při zpracování jednoho vstupního primitiva vygenerovat pouze omezené množství vrcholů (vizte sekci 2.4.2), jsou některé hrany příliš zakřivené, aby jím mohly být navzorkovány s dostatečnou věrností. Modul byl proto implementován tak, že pomocí modifikovaného algoritmu de Casteljau může pro jednu hranu vygenerovat více než jednu definici křivky. Modifikace algoritmu spočívá v tom, že úseky vstupní křivky, které splňují podmínku rovnosti, nejsou nahrazeny úsečkami, ale jejich kontrolní body jsou použity jako vstupní data pro geometry shader.

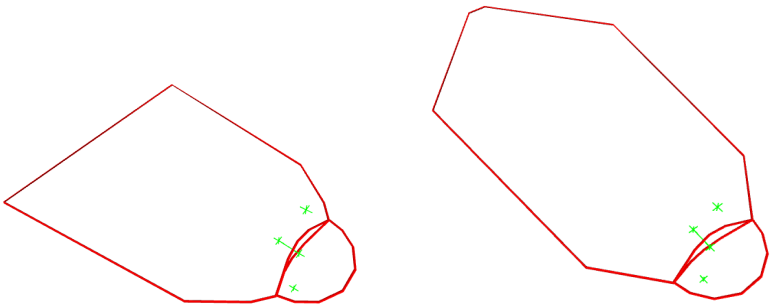
Použití geometry shaderu ke generování těchto geometrických dat zásadním způsobem snižuje nároky na CPU, který nyní pro GPU připravuje mnohem menší množství dat (pro každou z křivek pouze její kontrolní body, identifikátory vrcholů a případné textury obsahující doplňující data, nikdy však celý geometrický model křivky). Zároveň je díky němu GPU využívána k úloze, ke které se nejlépe hodí – všechny Bézierovy křivky lze totiž vzorkovat paralelně, protože jejich výpočty na sobě vzájemně nijak nezávisejí, a všechny křivky je možné vzorkovat totožnými instrukcemi, pouze s odlišnými parametry – využívá se tak výkonnostních výhod modelu SIMD.

3.3.1 Distribuce vzorků

U hran s velkým zakřivením – zejména pak u hran eliptických – při rovnoměrném rozdělení parametru t během vzorkování nastává problém. Kvůli rozdílům mezi vahami kontrolních bodů se ve zvýšené míře projevuje nelinearita rozložení vrcholů lomené čáry v prostoru.

V případě, že se liší váha prostředního kontrolního bodu od vah krajních kontrolních bodů se tyto vrcholy akumulují více u krajních kontrolních bodů křivky než u jejího středu. Jak lze vidět na obrázku 3.6 vlevo, při malém

počtu vzorků tak lomená čára křivku neaproximuje příliš věrně a vizualizace tak pozbývá své informační hodnoty.



Obrázek 3.6: Eliptická hrana o osmi vzorcích při rovnoměrném rozdělení parametru t (vlevo) a při použití transformace funkcí $f(t)$ (vpravo)

Implementovaným řešením je transformace parametru t funkcí f tak, aby se ve více vzorcích pohyboval blíže u hodnoty $\frac{1}{2}$. Na transformační funkci jsme měli několik požadavků. Pro průběh musí v definičním oboru $D_f = \langle 0; 1\rangle$ platit $f(t) = 1 - f(1 - t)$, tj. musí být středově symetrický podle bodu $[\frac{1}{2}; \frac{1}{2}]$; to z toho důvodu, aby při transformaci parametru osově souměrné křivky byla osová souměrnost po navzorkování zachována, protože v opačném případě by mohlo docházet ke zmatení pozorovatele. Dále je nutné, aby byla funkce v celém definičním oboru spojitá a platilo, že $f(0) = 0$ a $f(1) = 1$, aby i po transformaci bylo možné navzorkovat celou křivku. Z obou požadavků pak také vyplývá, že musí platit $f(\frac{1}{2}) = \frac{1}{2}$. Tyto požadavky plní například následující funkce, jež je použita v implementaci:

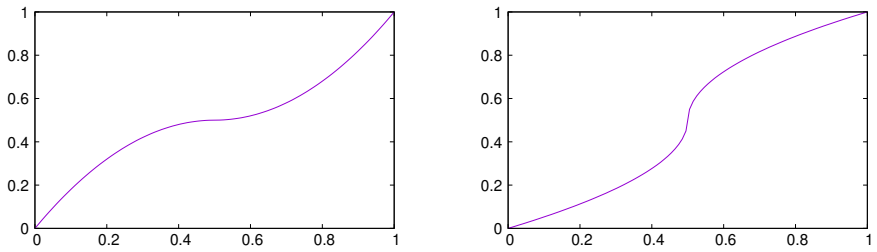
$$f(t) = \frac{1 + \mathrm{sig}(2t - 1) \cdot |2t - 1|^{a^p}}{2}, \tag{3.1}$$

kde $p \in \mathbb{R}$ je parametr určený relativně na základě poměru součtu vah krajních kontrolních bodů ku váze prostředního kontrolního bodu a konstanta $a \in (1; +\infty)$ upravuje citlivost transformace na změny poměru p. Pro určení parametru p je v implementaci použit následující vztah:

$$p = \frac{w_0 + w_2}{2 \cdot \sqrt[4]{|w_2|}} \tag{3.2}$$

V geometry shaderu nese funkce f název `centerBias`. Průběh funkce pro různé hodnoty poměru p lze vidět na obrázku 3.7.

Při dělení křivek algoritmem de Casteljau se můžeme setkat ještě s dalším jevem týkajícím se kumulace bodů. Krajní kontrolní body úseků křivek totiž nemusí mít váhy w_0, w_2 rovny 1, pokud váha w_1 prostředního kontrolního



(a) $p = 1$ (b) $p = -1$

(c) $p = \frac{1}{2}$ (d) $p = -\frac{9}{5}$

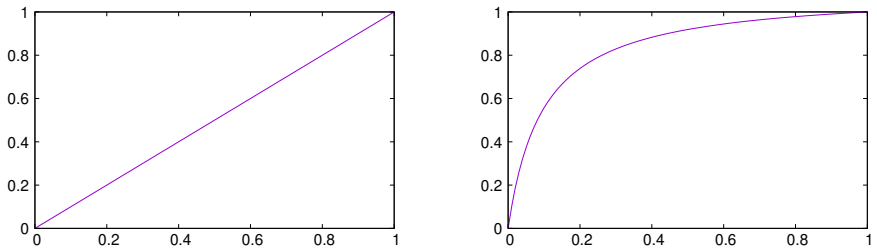
Obrázek 3.7: Průběhy transformační funkce f s konstantou $a = 2$ s různými hodnotami parametru p; vodorovná osa odpovídá hodnotě parametru t, svislá funkční hodnotě $f(t)$

bodu původní křivky není 1. Pokud se pak váhy krajních kontrolních bodů vzájemně nerovnají, akumulují se vzorky blíže ke kontrolnímu bodu s vyšší vahou.

Ke kompenzaci tohoto jevu v implementaci opět definujeme transformační funkci, tentokrát označenou g. Opět na tuto funkci máme požadavek na spojitost v definičním oboru $D_g = \langle 0; 1\rangle$ a stále musí platit, že $g(0) = 0$ a $g(1) = 1$ pro zachování schopnosti navzorkování celé křivky. Druhou podmínkou je, že $g(1-g(t)) = 1-t$, tj. funkce musí být symetrická podle přímky dané body $[0; 1]$ a $[1; 0]$, aby mohla být zachována symetrie hrany dělené na více křivek. Podmínky splňuje následující funkce:

$$g(t) = \frac{t}{a^{w_2 - w_0} \cdot (1 - t) + t}, \tag{3.3}$$

kde $w_0, w_2 \in \mathbb{R}$ jsou váhy krajních kontrolních bodů křivky a konstanta $a \in (1; +\infty)$ představuje citlivost funkce na změnu rozdílu vah w_0, w_2. V shaderu je tato funkce pojmenována `sideBias`. Průběh funkce pro různé hodnoty vah lze vidět na obrázku 3.8.

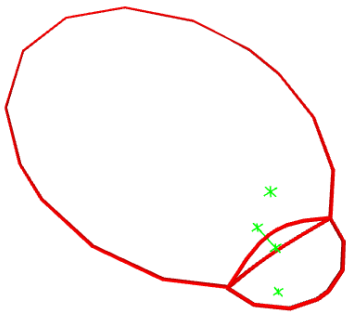


(a) $w_0 = 1, w_2 = 1$ (b) $w_0 = 1, w_2 = \frac{9}{2}$

(c) $w_0 = 1, w_2 = \frac{1}{5}$ (d) $w_0 = 1, w_2 = -1$

Obrázek 3.8: Průběhy transformační funkce g s konstantou $a = 2$ s různými hodnotami vah w_0 a w_2; vodorovná osa odpovídá hodnotě parametru t, svislá funkční hodnotě $f(t)$

Konečná hodnota parametru, která je použita k navzorkování dané hrany je tedy $g(f(t))$, kde t je původní hodnota zvolená rovnoměrným rozložením z intervalu $\langle 0; 1 \rangle$. Podobu eliptické hrany při použití dělení hran na CPU pomocí algoritmu de Casteljau a výše popisovaných transformací parametru t lze vidět na obrázku 3.9.



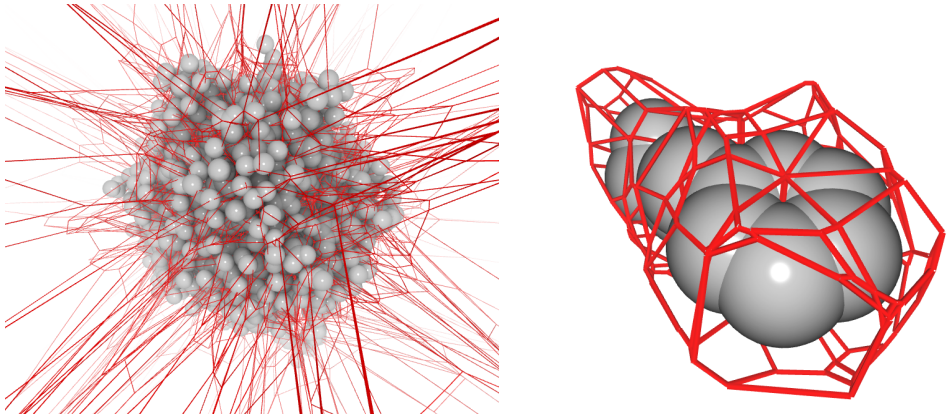
Obrázek 3.9: Finální podoba eliptické hrany při použití tranformací parametru t a dělení křivky algoritmem de Casteljau

3.4 Filtrování Voroného diagramu

Jak lze vidět na obrázku 3.10 vlevo, Voroného diagram sestrojený nad makromolekulou s mnoha atomy je velmi členitý. Jeho přímá vizualizace je tudíž značně nepřehledná. Uživatel zkoumající daný diagram často nepotřebuje vidět tento diagram celý a je tudíž vhodné mu poskytnout funkcionalitu pro zobrazení pouze určité části tohoto diagramu. Bylo proto implementováno filtrování diagramu na základě uživatelsky nastavitelných kritérií.

Prvním volitelným kritériem je váha Voroného sféry. V panelu nastavení modulu Voronoi Visualizer lze zvolit libovolný interval vah. Hrana je skryta, pokud žádný z Voroného vrcholů, které propojuje, není středem Voroného sféry, jejíž váha spadá do tohoto zvoleného intervalu.

Druhý způsob, kterým lze hrany filtrovat, je použití funkce výběru atomů, která je již existující součástí aplikace CAVER Analyst. Rozšiřující modul na změnu ve výberu automaticky reaguje tak, že zobrazí pouze ty Voroného hrany, které náleží alespoň jednomu z aktuálně vybraných atomů. Na obrázku 3.10 vpravo lze vidět příklad použití tohoto kritéria.



Obrázek 3.10: Hrany Voroného diagramu u celé molekuly (vlevo) a u vybrané skupiny atomů (vpravo)

3.5 Zvýraznění některých částí diagramu

Modul Voronoi Visualizer implementuje dva způsoby barevného zvýrazňování částí vizualizovaného Voroného diagramu. První možností je nastavení barev částí Voroného diagramu uživatelem tak, že se pomocí funkcionality výběru aplikace CAVER Analyst označí některé atomy molekuly a pomocí ovládacího panelu modulu Voronoi Visualizer se příslušnými ovládacími prvky obarvení aplikuje. V druhém případě modul umožňuje definici

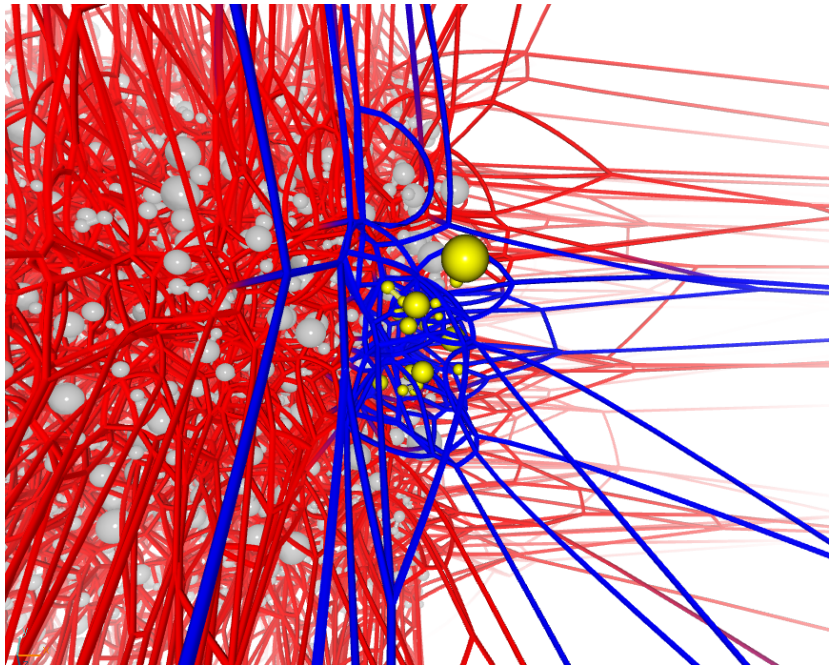
prahů vah Voroného sfér s přiřazenými barvami, mezi kterými je poté prováděna lineární interpolace podle váhy aktuálně zpracovávané Voroného sféry.

Barvy jsou shaderem vždy určeny pro vykreslované Voroného vrcholy. Na Voroného hranách je pak mezi nimi provedena lineární interpolace.

3.5.1 Uživatelské obarvení

K barvení podle výběru atomů, jehož podobu lze vidět na obrázku 3.11, jsou v modulu použity dvě textury. Data první jednorozměrné textury představují vazby Voroného vrcholů na atomy, kterým náleží – každému vrcholu náleží čtyři identifikátory atomů umístěné v textuře za sebou.

Druhá, jednorozměrná textura obsahuje barvy ve formátu RGBA, které jsou svým indexem vztaženy k identifikátorům atomů. Alfa kanál v tomto případě určuje váhu barvy a v aktuální implementaci nabývá pouze hodnot 1 či 0. Hodnota alfa kanálu 0 značí, že atomu nebyla uživatelem přiřazena žádná barva.



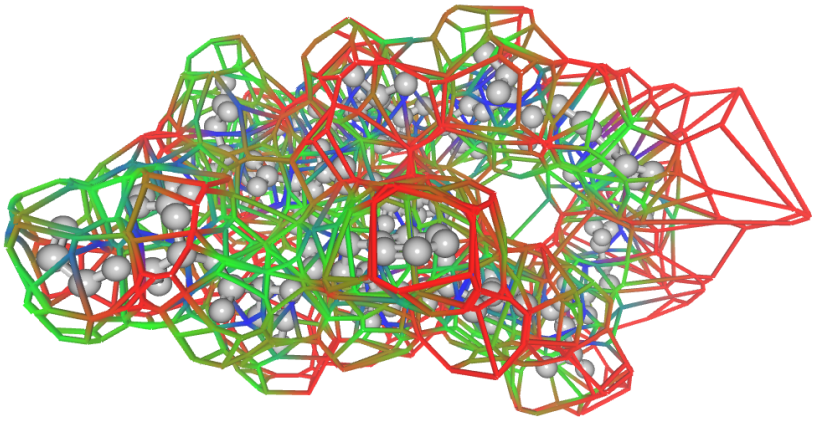
Obrázek 3.11: Hrany Voroného diagramu obarvené modře zvýrazňují hranice Voroného regionů žlutě vyznačenách kuliček

Jelikož si počty a identifikátory Voroného vrcholů v dynamice nemusejí vzájemně odpovídat napříč jednotlivými snímky, není možné vztáhnout obarvení na ně a při změně snímku jej zachovat. Barvy jsou proto výše popisovanou RGBA texturou přiřazovány atomům, jejichž množina zůstává napříč snímky neměnná.

Při přenosu geometrických dat do paměti GPU jsou pak krajním kontrolním bodům Voroného hran přiřazeny identifikátory vrcholů těmito hranami propojených. Identifikátory vrcholů jsou použity geometry shaderem k vyhledání atomům přiřazených barev, z nichž je poté váženým průměrem podle alfa kanálu určena výsledná barva Voroného vrcholu. Důsledkem použití váženého průměru je to, že k určení výsledné barvy přispívají pouze ty atomy, jimž uživatel explicitně nastavil nějakou barvu. V případě, že jsou váhy všech získaných barev nulové, tj. uživatel žádným z příslušných atomů žádnou barvu neurčil, je pro vrchol vybrána výchozí barva.

3.5.2 Obarvení podle vah Voroného sfér

Obarvování hran podle váhy Voroného sféry, demonstrované na obrázku 3.12, využívá dvě jednorozměrné textury. První textura je polem vah Voroného sfér. Váhy jsou v poli seřazeny tak, aby jejich indexy odpovídaly identifikátorům jednotlivých Voroného sfér, resp. vrcholů.



Obrázek 3.12: Hrany Voroného diagramu obarvené podle váhy Voroného sfér se základním nastavením prahů: nejnižší váha je zobrazována modře, střední zeleně a nejvyšší červeně

Druhá textura má pevně danou velikost v kódu programu a obsahuje lineárně interpolované barvy vypočtené z prahů vah ve formátu RGB. Shader k této textuře přistupuje pomocí float souřadnice v normalizovaném prostoru (tj. data textury jsou mapována na interval $\langle 0; 1 \rangle$) a při požadavku na souřadnice mimo jednotkový interval jsou použity hodnoty hraničních texelů.

Tato barevná textura je vygenerována z uživatelsky nastavitelných prahů definovaných ve vyváženém binárním stromu tak, že její první prvek odpovídá barvě nejnižší váhy, poslední prvek odpovídá barvě nejvyšší váhy, barvy

ostatních vah jsou mezi nimi proporcionálně rozmístěny a ostatní hodnoty jsou určeny lineární interpolací mezi těmito barvami. Prahy a barvy v tomto stromě lze měnit pomocí příslušné tabulky v panelu nastavení.

Geometry shader při určování barev částí Voroného diagramu opět použije identifikátory Voroného vrcholů, resp. sfér, tentokrát k získání hodnoty jejich vah z první texury. Na základě váhy získá barvu vrcholu z druhé textury.

3.6 Vizualizace diagramů v dynamice

Základním požadavkem pro jakoukoli dynamickou vizualizaci v reálném čase je zajištění co nejkratší doby zpracování vykreslovaného snímku. Toho se modul Voronoi Visualizer snaží dosáhnout využitím paralelizace generování většiny geometrie na GPU pomocí geometry shaderu a celkové minimalizace procesů, které je nutné provádět pro každý snímek.

Aktuální implementace načítání dat dynamiky Voroného diagramů v aplikaci CAVER Analyst je však úzkým hrdlem. Protože je výpočet Voroného diagramu náročný a objem jeho topologických dat pro množství snímků tak veliký, že jej nelze pro všechny snímky držet v operační paměti, je nutné tato data dynamicky načítat z komprimovaného souboru na disku (vizte sekci 2.4.1). Dekomprese implementovaná knihovnou awVoronoi však v čase psaní této práce není dostatečně efektivní.

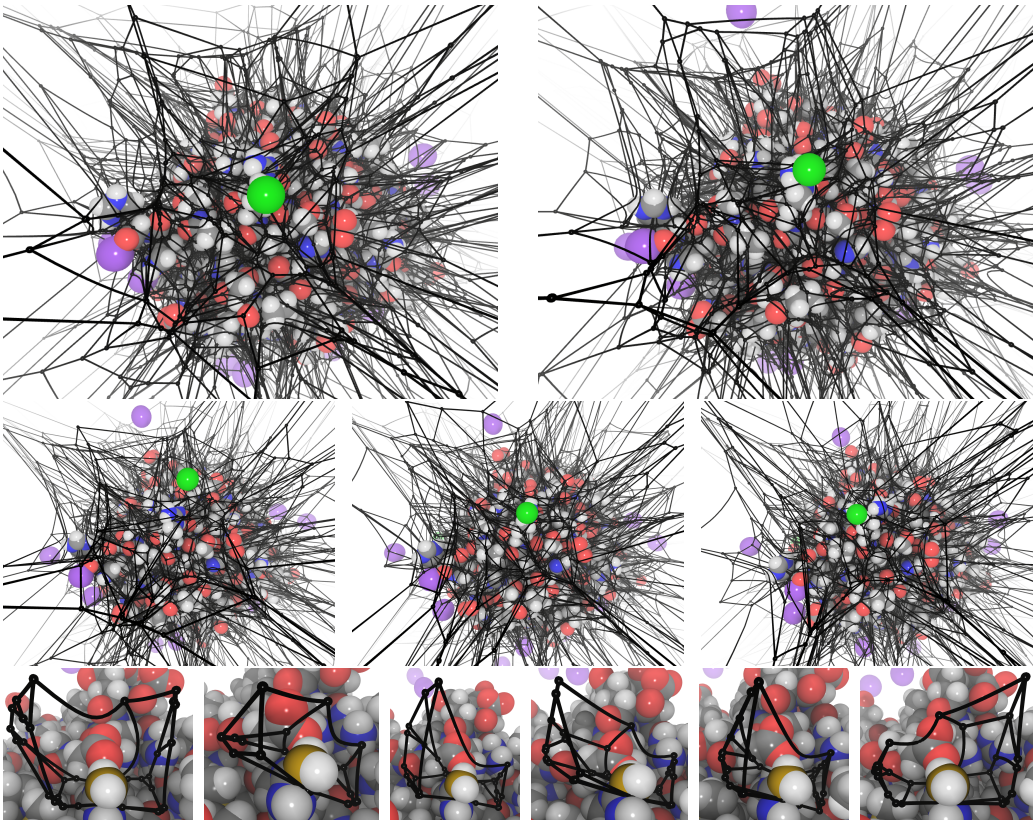
Z tohoto důvodu implementuje modul Voronoi Visualizer LRU cache geometrických dat, která je uchovávána v operační paměti za použití `LinkedHashMap` [11] ze standardní knihovny jazyka Java. LRU (Least Recently Used) je strategie výměny prvků cache, kdy je při překročení její kapacity odstraněn prvek, k němuž nebylo přistupováno po nejdelší dobu. Pro každý vizualizovaný snímek, jehož data se nenachází v cache, jsou načtena topologická data Voroného diagramu knihovnou awVoronoi. Z nich Voronoi Visualizer vygeneruje data zakřivených hran, vrcholů a obarvení, která jsou uložena do této cache, kde klíčem pro získání těchto dat je číslo aktuálního snímku. Díky této cache může vizualizace efektivně fungovat nad omezenou množinou snímků bez neustálého načítání dat z disku.

Cache typu LRU pravděpodobně není zcela ideální volbou, protože uchovává pouze omezené množství již přehraných snímků. S plynulostí animace tak pomáhá pouze při opakovaném přehrávání malého úseku dynamiky. Lepším řešením by mohlo být předběžná příprava snímků, aby alespoň jejich malá část mohla být plynule přehrávána okamžitě. Ukládání animace do cache však není cílem práce – původně se předpokládalo, že dekomprese

dynamiky diagramu bude dostatečně rychlá, aby bylo možné ji přehrávat v reálném čase bez dodatečných změn v načítání ze strany vizualizátoru.

Pro přehlednost vizualizace používá CAVER Analyst při přehrávání dynamiky defaultně malou rychlost 5 FPS. Takto malá frekvence změny snímků má za výsledek velmi trhanou animaci, proto aplikace umožňuje vyhlazení animace interpolací pomocí Hermitovy křivky. Tento způsob interpolace však nelze použít pro dynamiku Voroného diagramů, neboť se mezi jednotlivými snímky může měnit jejich topologie. Topologii mezi snímky by bylo nutné nějakým způsobem dopočítávat. To však není triviálním problémem a bylo by nutné provést další rozsáhlou analýzu – modul Voronoi Visualizer proto aktuálně interpolaci nepodporuje a v případě, že se přehrávač dynamiky nachází v pozici mezi snímky, zobrazuje poslední platný snímek Voroného diagramu. To může být pro pozorovatele matoucí, proto se s aktuální implementací doporučuje vyhlazování animace v aplikaci vypnout odškrtnutím políčka *Smooth animation* v přehrávači dynamiky.

Obrázek 3.13 ilustruje vizualizaci Voroného diagramů na dynamickém modelu molekuly proteinu a detail Voroného buňky pro jeden vybraný atom.



Obrázek 3.13: Příklad Voroného diagramů a detail vývoje jednoho regionu v několika snímcích dynamického modelu proteinu.

4 Dosažené výsledky

Po ukončení implementace byl modul Voronoi Visualizer podroben konečnému testování. Zaměřujeme se především na výkonnostní nároky, tj. časy zpracování snímků a generování dat Voroného hran na CPU; a vizuální stránku vykreslovaných aditivně vážených Voroného diagramů.

4.1 Prostředí, metodiky a data

4.1.1 Testovací prostředí

Testování bylo prováděno na počítači s CPU Intel Core i5-6300HQ o taktu 2,30 GHz, s RAM velikosti 8 GB a s GPU Nvidia GeForce GTX 960M s VRAM velikosti 4 GB. Použitým operačním systémem byl Manjaro Linux s verzí jádra `5.6.8-1-MANJARO`. Použitým ovladačem pro grafický čip byl oficiální proprietární ovladač společnosti Nvidia ve verzi `435.21`.

4.1.2 Měřicí metody

K měření výkonu je u aplikace CAVER Analyst použit vestavěný měřič délky zpracování snímku. K měření aplikace Voroprot byla kvůli absenci vlastního měřiče použita utilita s názvem MangoHud [4]. Původním záměrem bylo pro garanci konzistence mezi měřenými aplikacemi použití utility MangoHud i k měření aplikace CAVER Analyst, avšak platforma Java přes všechny pokusy jeho použití znemožnila.

Všechna měření jsou prováděna v maximalizovaných oknech na monitoru s rozlišením 1920×1080, přičemž vykreslovací plocha je vždy zvětšena na maximální velikost, jakou aplikace umožňuje. Pokud není uvedeno jinak, testy jsou provedeny na vizualizaci hran Voroného diagramu společně s van der Waalsovým modelem molekuly.

Modul Voronoi Visualizer do aplikačního logu zapisuje naměřené časy zpracování geometrie Voroného diagramu, tj. získání dat knihovny awVoronoi (vč. případné konstrukce aditivně váženého Voroného diagramu), generování textur, filtrování Voroného sfér a generování dat o zakřivení hran. Příklad takového zápisu se nachází na obrázku 4.1. K měření jsou použity tyto logované hodnoty při opakovaném generování dat vizualizace, tj. když data knihovny awVoronoi jsou již hotová a dochází pouze ke generování křivek a filtrování diagramu samotným modulem Voronoi Visualizer.

```
[17:44:22] == Starting Voronoi visualization recalculation
[17:44:24] Fetch of triangulation data took 1615 ms
[17:44:24] Cell-to-atoms texture generation took 20 ms
[17:44:24] Cell weights texture generation took 13 ms
[17:44:24] Cell filtering took 30 ms
[17:44:24] Edge generation took 187 ms
[17:44:24] == Voronoi visualization recalculation took 1874 ms
```

Obrázek 4.1: Úryvek z aplikačního logu – generování dat pro vizualizaci

4.1.3 Vstupní data

Modul byl testován na různých typech dat. Primární funkcí aplikace CAVER Analyst je vizualizace makromolekulárních struktur – pro zátěžové testování na větších datech proto byly využívány struktury z databáze RCSB PDB. Pro ověření schopnosti modulu zobrazovat některé konkrétní jevy, které se mohou u aditivně vážených Voroného diagramů vyskytnout, pak byla použita specificky vytvořená testovací data.

4.2 Výkon vizualizace Voroného hran

Při testování jsme měřili přípravu dat pro vizualizaci modulem Voronoi Visualizer a doby zpracování snímků pro vizualizaci Voroného hran. Testovány byly vizualizace jak u modelů s vypnutou dynamikou, tak při jejím přehrávání.

Ukazuje se, že díky LRU cache dat snímků, jež byla implementována k potlačení dlouhých dob dekomprese snímků Voroného diagramu, se výkon vizualizace mezi statickým a dynamickým modelem liší pouze při prvotním načítání dat knihovny awVoronoi a generování dat pro vizualizaci. Generování vizualizačních dat proto bylo měřeno odděleně a dále byly testovány časy zpracování snímků nad již vygenerovanými daty.

Pro zasazení výsledků do kontextu byly testovány časy zpracování snímků při vizualizaci samotných hran, vizualizaci samotného van der Waalsova modelu molekuly (značeno jako *atomy*) a obou těchto vizualizací zároveň. K časům zpracování jsou uvedeny odpovídající frekvence snímků za sekundu.

Umělá data 1

Data o čtyřech vstupních sférách byla uměle vytvořena za účelem vygenerování dvou eliptických hran – byla rovněž použita během vývoje modulu. Podobu vizualizace těchto dat lze vidět na obrázku 3.9 v sekci o řešení problému distribuce vzorků.

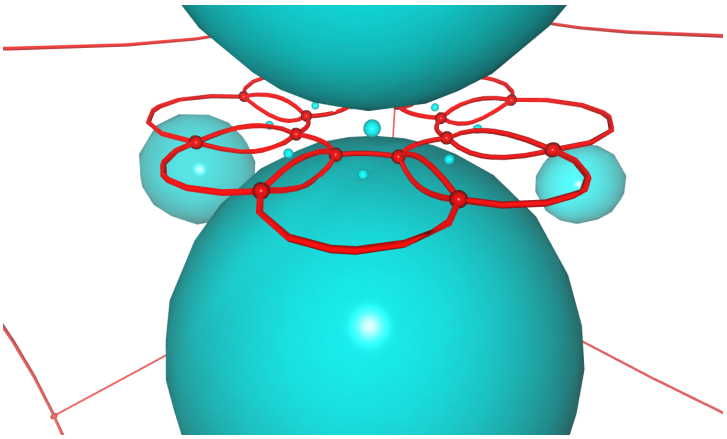
Pro takto malá data samozřejmě nemá smysl zkoumat výkon samotného zpracování dat naším programem (data vizualizace jsou zde vygenerována za méně než jednu milisekundu), důležité je však znát i čas, který zabere režie platformy Java a knihovny OpenGL, resp. komunikace s GPU. Naměřené hodnoty nám o tomto mohou poskytnout jistou představu:

- **Počet vstupních sfér (atomů):** 4
- **Typ:** Statická
- **Generování dat vizualizace:** < 1 ms
- **Zpracování snímku – hrany:** $11 - 13$ ms $(76 - 90$ FPS$)$
- **Zpracování snímku – atomy:** $14 - 15$ ms $(66 - 71$ FPS$)$
- **Zpracování snímku – hrany, atomy:** $12 - 13$ ms $(76 - 83$ FPS$)$

Umělá data 2

Uměle vytvořená množina dat pro účely testování vizualizace Voroného diagramů. Diagram v tomto modelu obsahuje více oddělených komponent hranové souvislosti – tj. pokud bychom tento diagram interpretovali jako graf, nelze z každého jeho vrcholu nalézt cestu ke všem ostatním vrcholům. Ukázku vizualizace lze vidět na obrázku 4.2.

- **Počet vstupních sfér (atomů):** 16
- **Typ:** Statická
- **Generování dat vizualizace:** $1 - 5$ ms $(200 - 1000$ FPS$)$
- **Zpracování snímku – hrany:** $11 - 13$ ms $(76 - 90$ FPS$)$
- **Zpracování snímku – atomy:** $11 - 13$ ms $(76 - 90$ FPS$)$
- **Zpracování snímku – hrany, atomy:** $11 - 13$ ms $(76 - 90$ FPS$)$



Obrázek 4.2: Hrany a vrcholy Voroného diagramu (červeně) nad uměle vytvořenou množinou vstupních sfér (tyrkysově). Za povšimnutí zde stojí „ostrůvek“ eliptických hran uprostřed.

p53

Protein, který v buňkách detekuje chyby v jejich DNA.

Původ dat: Výzkumná skupina Eleny Papaleo, Laboratoř výpočetní biologie, Kodaň, Dánsko (https://sites.google.com/site/cblcancerdk). Data byla použita jako testovací model ve společné publikaci se ZČU (http://doi.org/10.1016/j.jmgm.2017.03.018).

- **Počet vstupních sfér (atomů):** 3 008
- **Typ:** Dynamická
- **Generování dat vizualizace:** 51 – 62 ms (16 – 19 FPS)
- **Zpracování snímku – hrany:** 22 – 23 ms (43 – 45 FPS)
- **Zpracování snímku – atomy:** 13 – 14 ms (71 – 76 FPS)
- **Zpracování snímku – hrany, atomy:** 23 – 25 ms (40 – 43 FPS)

Ethanol

Simulace 500 molekul tekutého ethanolu.

Původ dat: Simulaci provedl Alexey Anikeenko (Vojvodský institut chemické kinetiky a spalování, Novosibirsk, Rusko), post-doc na Západočeské Univerzitě v Plzni v roce 2019, podle parametrů silového pole OPLS-AA ethanol (http://virtualchemistry.org).

- **Počet vstupních sfér (atomů):** 4 500
- **Typ:** Dynamická
- **Generování dat vizualizace:** 51 – 80 ms (12 – 19 FPS)
- **Zpracování snímku – hrany:** 29 – 30 ms (33 – 34 FPS)
- **Zpracování snímku – atomy:** 16 – 17 ms (58 – 62 FPS)
- **Zpracování snímku – hrany, atomy:** 32 – 33 ms (30 – 31 FPS)

Hexane

Simulace 250 molekul tekutého n-hexanu.

Původ dat: Simulaci provedl Alexey Anikeenko (Vojvodský institut chemické kinetiky a spalování, Novosibirsk, Rusko), post-doc na Západočeské Univerzitě v Plzni v roce 2019, podle parametrů silového pole CGenFF n-hexane (http://virtualchemistry.org).

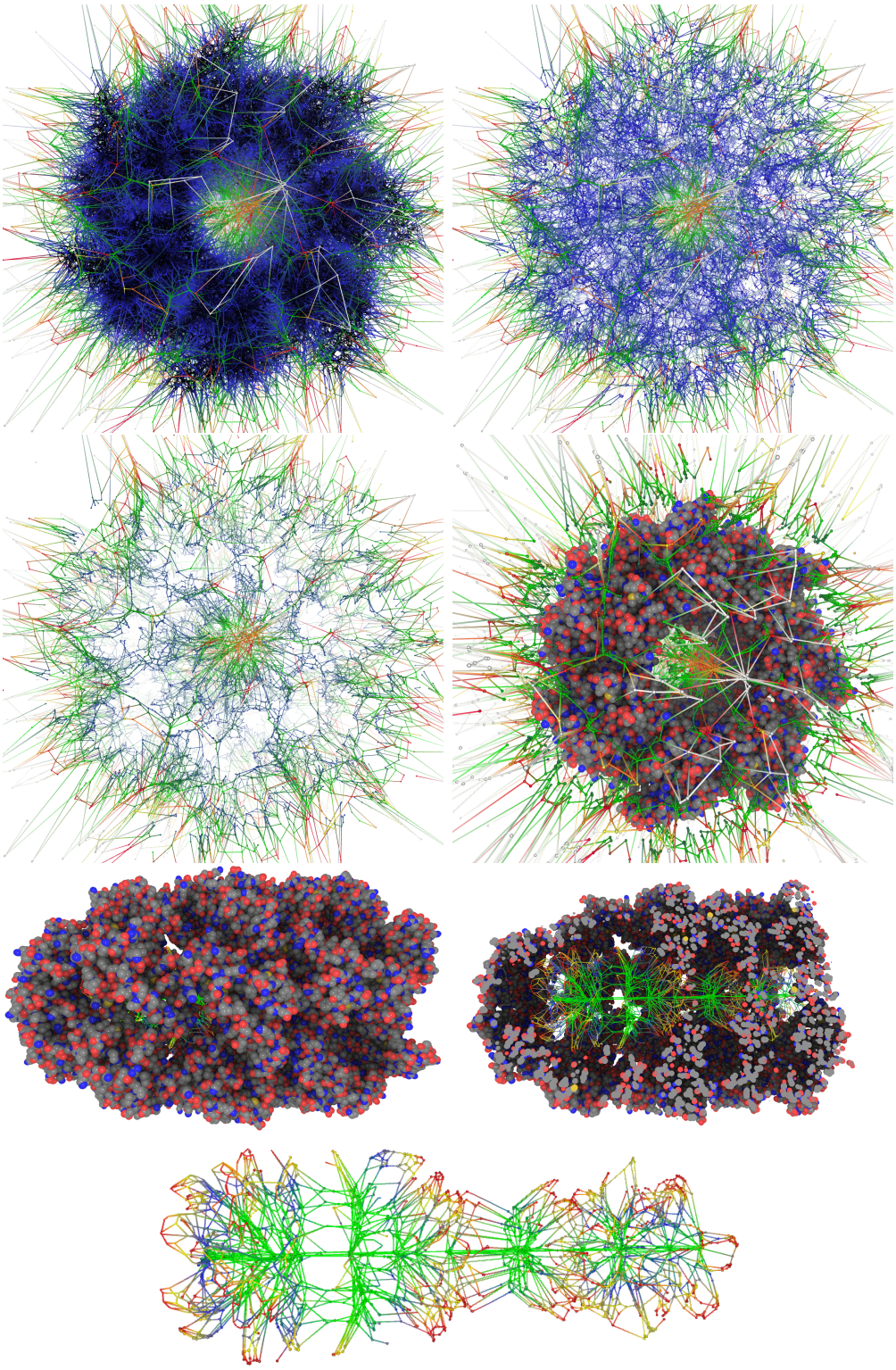
- **Počet vstupních sfér (atomů):** 5 000
- **Typ:** Dynamická
- **Generování dat vizualizace:** 79 – 105 ms (9 – 12 FPS)
- **Zpracování snímku – hrany:** 30 – 31 ms (32 – 33 FPS)
- **Zpracování snímku – atomy:** 13 – 14 ms (71 – 76 FPS)
- **Zpracování snímku – hrany, atomy:** 32 – 33 ms (30 – 31 FPS)

1AON

Chaperonin – protein asistující při skládání některých dalších proteinů. Ukázky vizualizace lze vidět na obrázku 4.3.

Původ dat: Vloženo do databáze RCSB PDB autory Xu Z., Horwich, A.L., Sigler, P.B. (https://www.rcsb.org/structure/1AON, http://doi.org/10.2210/pdb1AON/pdb)

- **Počet vstupních sfér (atomů):** 58 884
- **Typ:** Statická
- **Generování dat vizualizace:** 960 – 1059 ms (0 – 1 FPS)
- **Zpracování snímku – hrany:** 282 – 289 ms (cca 3 FPS)
- **Zpracování snímku – atomy:** 49 – 50 ms (cca 20 FPS)
- **Zpracování snímku – hrany, atomy:** 303 – 315 ms (cca 3 FPS)



Obrázek 4.3: Různé konfigurace vizualizace Voroného hran a vrcholů s obarvením podle vah Voroného sfér nad modelem chaperoninu 1AON

4.3 Porovnání s aplikací Voroprot

Protože Voroprot podporuje vizualizaci hran aditivně váženého Voroného diagramu a stejně jako CAVER Analyst umožňuje načítání dat molekul ve formátu PDB, je nasnadě přímé porovnání s implementovanou vizualizací modulu Voronoi Visualizer, a to jak z hlediska vizuálního, tak z hlediska výkonu.

Aplikace Voroprot implementuje vizualizaci Voroného hran a stěn na základě výběru atomů molekuly. Vizualizace je kvalitní, hrany jsou stínované a nejsou patrné žádné významnější defekty ve vykreslovaných modelech hran. Při výběru velkého množství atomů však vizualizace ztrácí na přehlednosti a velmi rychle i na výkonu. Při testování aplikace na struktuře 1CQW [8] – relativně malém proteinu o cca 2 800 atomech, u kterého by se dala očekávat vizualizace bez větších výkonnostních problémů – se doba zpracování snímku při zobrazení celého diagramu pohybuje mezi $246-250$ ms ($3-4$ FPS). Po vypnutí vykreslování van der Waalsova modelu molekuly se doba zpracování ustáluje na 246 ms (4 FPS). Při vykreslování samotného van der Waalsova modelu molekuly bez Voroného hran pak vizualizace dosahuje doby zpracování $16-17$ ms ($59-60$ FPS) – zde zřejmě dochází k omezení frekvence snímků na hodnotu obnovovací frekvence monitoru (popř. na pevně určenou hodnotu 60 FPS), neboť ani při snížení počtu vykreslovaných atomů nedochází ke snížení doby zpracování.

Při zobrazování stejné struktury v aplikaci CAVER Analyst s modulem Voronoi Visualizer je výkon mnohonásobně vyšší, doba zpracování snímku se pohybuje mezi $22-26$ ms ($39-43$ FPS). Při vypnutí vykreslování van der Waalsova modelu molekuly se doba zpracování snímku pohybuje mezi $22-23$ ms ($47-48$ FPS), při zobrazení samotného van der Waalsova modelu se pak doba snímku zkracuje na $13-16$ ms ($66-75$ FPS).

Při velkém přiblížení či nastavení tloušťky hran mohou být u vizualizace patrné defekty, jako je defaultní malý počet stěn válečků a absence jejich podstavných stěn – za normálních podmínek však tyto defekty nejsou tak vážné a v zásadě nenarušují informační hodnotu vizualizace. Díky vestavěnému efektu mlhy aplikace CAVER Analyst může být vizualizace rozsáhlejších částí diagramu mírně přehlednější.

4.4 Zhodnocení výsledků

Naměřené výsledky ukazují, že výpočet geometrie zakřivených hran na GPU pomocí geometry shaderu může být krok správným směrem. Samotná vizualizace je responzivní při zobrazování makromolekul o počtech atomů

pohybujících se v řádech vyšších jednotek tisíců, a to přestože efektivita její implementace zatím není kompletně optimalizována. Při porovnání s konkurenčními aplikacemi lze tuto skutečnost považovat za úspěch.

U vizualizace hran si lze v ojedinělých případech všimnout některých defektů, jako jsou „překroucení“ způsobená nesprávným generováním vrcholů válečků při některých konfiguracích nebo mezer mezi hranami v místech, kde se stýkají. Tyto defekty zpravidla nemají vliv na informační hodnotu vizualizace, ale bylo by vhodné je potlačit pro zlepšení uživatelské přívětivosti aplikace.

Rozsáhlejší optimalizace je zatím nasnadě při generování vizualizačních dat. V případě, že by dekomprese dynamiky dosahovala vyšších rychlostí a nebyla by tak potřebná cache geometrie v operační paměti, by se generování mohlo stát úzkým hrdlem. Pokusné profilování této části programu napovídá, že by zde mohla napomoci například optimalizace kódu, který generuje prostřední kontrolní body křivek Voroného hran.

5 Závěr

Cílem této práce bylo nalezení způsobu, jak přehledně a výpočetně efektivně vizualizovat aditivně vážené Voroného diagramy v dynamických modelech makromolekul, a implementovat tuto vizualizaci v podobě rozšiřujícího modulu pro aplikaci *CAVER Analyst 2.0*.

Bylo navrženo řešení, které využívá *algoritmus de Casteljau* pro segmentaci *kvadratických racionálních Bézierových křivek* na CPU a *geometry shader*, pomocí kterého lze díky masivní paralelizaci zpracování stejnorodých dat na GPU velmi efektivně generovat požadovanou geometrii pro jejich vizualizaci.

Rozšiřující modul *Voronoi Visualizer* pro aplikaci CAVER Analyst, implementovaný v rámci této práce, používá takovýto shader ke generování prostorových *válečků* opisujících zakřivené Voroného hrany, čímž je využíván výpočetní potenciál GPU ke generování velkého množství geometrie, potřebné ke zvýšení přehlednosti vizualizace pomocí efektu perspektivy. Díky shaderu je vizualizace dostatečně rychlá pro zajištění příjemného používání aplikace s velkou škálou vstupních dat. Modul uživateli umožňuje měnit parametry vizualizace pomocí filtrů a obarvování hran za účelem jejího cíleného zpřehlednění se zaměřením na konkrétní části pozorovaného Voroného diagramu.

Modul je schopen vizualizovat Voroného diagramy i nad přehrávanou dynamikou. Zde sice naráží na technické limitace aktuální implementace dekomprese dat diagramu v aplikaci CAVER Analyst, avšak díky implementované cache snímků je schopen plynule přehrávat alespoň krátkou část takovéto dynamiky.

Testování na reálných i uměle vytvořených datech ukazuje, že použití *geometry shaderu* k výpočtu geometrie na GPU může být krok správným směrem. Vizualizace Voroného diagramu je efektivní, zejména pak v porovnání s konkurenční vizualizací aplikace Voroprot.

Práci je možné dále rozšiřovat – lze například uvažovat nad možnými dodatečnými optimalizacemi implementovaných vizualizací (např. využít skutečného instancování geometrie k vizualizaci Voroného vrcholů), nad přidáním dalších forem vizualizací Voroného diagramu (např. Voroného stěn), či nad interpolací topologií pro vyhlazování animace diagramu v dynamice.

Seznam zkratek

- **CPU** – *Central Processing Unit* – hlavní procesor počítače
- **FPS** – *Frames Per Second* – frekvence snímků / snímky za sekundu
- **GPU** – *Graphics Processing Unit* – grafický procesor/čip
- **LRU** – *Least Recently Used* – strategie výměny prvků cache odstraňující prvky, které byly naposledy použity před nejdelší dobou
- **RAM** – *Random Access Memory* – zde hlavní paměť počítače
- **RCSB PDB** – *Research Collaboratory for Structural Bioinformatics – Protein Data Bank* – databáze modelů proteinů
- **RGB** – *Red, Green, Blue* – definice barvy na základě tří základních barev: červené, zelené, modré
- **RGBA** – *Red, Green, Blue, Alpha* – definice barvy na základě tří základních barev: červené, zelené, modré; s alfa kanálem, který může určovat např. průhlednost
- **SIMD** – *Single Instruction, Multiple Data* – model paralelizace zpracování stejnorodých dat používaný na *GPU*
- **VRAM** – *Video Random Access Memory* – zde paměť grafické karty

Literatura

[1] Bondi, A. Van der Waals Volumes and Radii. *Journal of Physical Chemistry*. 1964, 68, 3, s. 441–451. doi: 10.1021/j100785a001.

[2] Christersson, M. *De Casteljau's Algorithm and Bézier Curves* [online]. 2017. Dostupné z: https://www.malinc.se/m/DeCasteljauAndBezier.php.

[3] Deok-Soo, K. – Youngsong, C. – Donguk, K. Euclidean Voronoi Diagram of 3D Balls and its Computation via Tracing Edges. *Computer-Aided Design*. 2005, 37, 13, s. 1412–1424. doi: 10.1016/j.cad.2005.02.013.

[4] Flightlessmango. *MangoHud: A Vulkan and OpenGL overlay for monitoring FPS, temperatures, CPU/GPU load and more — GitHub* [online]. Dostupné z: https://github.com/flightlessmango/MangoHud.

[5] Jae-Kwan, K. et al. Voronoi diagrams, quasi-triangulations, and beta-complexes for disks in R^2: the theory and implementation in BetaConcept. *Journal of Computational Design and Engineering*. 2014, 1, 2, s. 79–87.

[6] Jurcik, A. et al. CAVER Analyst 2.0: analysis and visualization of channels and tunnels in protein structures and molecular dynamics trajectories. *Bioinformatics*. 2018, 34, 20, s. 3586–3588. doi: 10.1093/bioinformatics/bty386.

[7] Maňák, M. *AwVoronoi* [online]. Dostupné z: https://sourceforge.net/projects/awvoronoi.

[8] Newman, J. et al. *RCSB PDB – 1CQW: NAI COCRYSTALLISED WITH HALOALKANE DEHALOGENASE FROM A RHODOCOCCUS SPECIES* [online]. Dostupné z: https://www.rcsb.org/structure/1CQW.

[9] Okabe, A. et al. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Second Edition*. Wiley Series in Probability and Statistics. Wiley, 2008. doi: 10.1002/9780470317013.

[10] Olechnovic, K. – Margelevičius, M. – Venclovas, C. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics*. 12 2010, 27, 5, s. 723–724. ISSN 1367-4803. doi: 10.1093/bioinformatics/btq720. Dostupné z: https://doi.org/10.1093/bioinformatics/btq720.

[11] Oracle Corporation. *LinkedHashMap (Java Platform SE 8)* [online]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html.

[12] Rycroft, C. H. *Multiscale Modeling in Granular Flow*. PhD thesis, Massachusetts Institute of Technology, 2007. Dostupné z: http://people.seas.harvard.edu/~chr/publish/phd.html.

[13] Rycroft, C. H. *Voro++* [online]. Dostupné z: http://math.lbl.gov/voro++/about.html.

[14] Shene, C.-K. *Finding a Point on a Bézier Curve: De Casteljau's Algorithm* [online]. Dostupné z: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-casteljau.html.

[15] Shene, C.-K. *Subdividing a Bézier Curve* [online]. Dostupné z: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/bezier-sub.html.

[16] The Khronos Group Inc. *Fragment Shader — OpenGL Wiki* [online]. 2018. Dostupné z: https://www.khronos.org/opengl/wiki/Fragment_Shader.

[17] The Khronos Group Inc. *Geometry Shader — OpenGL Wiki* [online]. 2019. Dostupné z: https://www.khronos.org/opengl/wiki/Geometry_Shader.

[18] The Khronos Group Inc. *Rendering Pipeline — OpenGL Wiki* [online]. 2019. Dostupné z: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview.

[19] The Khronos Group Inc. *Sampler (GLSL) — OpenGL Wiki* [online]. 2019. Dostupné z: https://www.khronos.org/opengl/wiki/Sampler_(GLSL).

[20] The Khronos Group Inc. *Shader — OpenGL Wiki* [online]. 2019. Dostupné z: https://www.khronos.org/opengl/wiki/Shader.

[21] The Khronos Group Inc. *Texture — OpenGL Wiki* [online]. 2019. Dostupné z: https://www.khronos.org/opengl/wiki/Texture.

[22] The Khronos Group Inc. *Vertex Shader — OpenGL Wiki* [online]. 2017. Dostupné z: https://www.khronos.org/opengl/wiki/Vertex_Shader.

[23] The POV-Team. *POV-Ray. Persistence of Vision Pty. Ltd. (2004). Persistence of Vision (TM) Raytracer. Persistence of Vision Pty. Ltd., Williamstown, Victoria, Australia.* [online]. Dostupné z: http://www.povray.org.

[24] Voronoï, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die Reine und Angewandte*

Mathematik. 1908, 133, s. 97–178. doi: 10.1515/crll.1908.133.97. Dostupné z: http://eudml.org/doc/149276.

[25] Weisstein, E. W. *Bernstein polynomial — MathWorld – A Wolfram resource* [online]. Dostupné z: https://mathworld.wolfram.com/BernsteinPolynomial.html.

[26] Weisstein, E. W. *Bézier curve — MathWorld – A Wolfram resource* [online]. Dostupné z: https://mathworld.wolfram.com/BezierCurve.html.

[27] Xu, Z. – Horwich, A. – Sigler, P. *RCSB PDB – 1AON: CRYSTAL STRUCTURE OF THE ASYMMETRIC CHAPERONIN COMPLEX GROEL/GROES/(ADP)7* [online]. Dostupné z: https://www.rcsb.org/structure/1AON.

[28] Youngsong, C. et al. BetaMol: a molecular modeling, analysis and visualization software based on the beta-complex and the quasi-triangulation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*. 2012, 6, 3, s. 389–403.

[29] Zbyněk Šír, B. J. On de Casteljau-type algorithms for rational Bézier curves. *Journal of Computational and Applied Mathematics*. 2015, 288, s. 244–250. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0377042715000497.