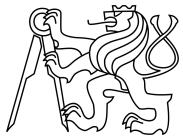
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Simulátor paralelních vnořovacích algoritmů
Student:	Bc. Ivo Kolář
Vedoucí:	Ing. Michal Šoch, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Seznamte se s problematikou vnořování propojovacích sítí paralelních počítačů. Následně se seznamte s aktuálními technologiemi tvorby internetových aplikací. Vyberte vhodnou technologii pro realizaci internetové aplikace, která bude fungovat jako simulátor vybraných vnořovacích algoritmů. Zaměřte se na grafickou podobu simulace, aby bylo zřejmé, jak vlastní vnořovací algoritmus funguje. Aplikaci poté navrhněte, prakticky naimplementujte, otestujte a řádně zdokumentujte.

Aplikace musí simulovat minimálně následující vnořovací algoritmy:
- vnoření úplného stromu do hyperkrychle,
- vnoření hyperkrychle do mřížky,
- vnoření mřížky do mřížky.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Simulátor paralelních vnořovacích algoritmů

Bc. Ivo Kolář

Vedoucí práce: Ing. Michal Šoch, Ph.D.

4. ledna 2017

Poděkování

Chtěl bych tímto poděkovat vedoucímu práce panu Ing. Michalu Šochovi, Ph.D. za to, že si našel čas dělat vedoucího této diplomové práce a také za jeho vstřícné jednání a za hodnotné připomínky, které k této práci měl.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. ledna 2017 .....

Odkaz na tuto práci

Kolář, Ivo. *Simulátor paralelních vnořovacích algoritmů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Následující text se zabývá problematikou paralelních vnořovacích algoritmů. Popisuje nejprve teoreticky různé sítě, které se do sebe dají vnořovat a následně i vnořovací algoritmy, které zachovávají co nejlepší vlastnosti daných sítí. Součástí této práce je i program, který vybrané algoritmy dokáže nasimulovat ve 3D animaci. Celá tato práce bude sloužit jako výuková pomůcka pro studenty předmětu Paralelní algoritmy a systémy.

Klíčová slova Paralelní algoritmy, Paralelní vnořovací algoritmy, hyperkrychle, mřížka, binární strom, 3D, simulace

Abstract

The following text deals with the issue of parallel embeding algorithms. It describes the first theoretically different networks, which in themselves can be nested and subsequently embeding algorithms that preserve the best characteristics of the network. Part of this work is a program that can simulate the selected algorithms in 3D animation. All this work will serve as a teaching tool for students enrolled Parallel Algorithms and Systems.

Keywords Parallel algorithms, Parallel embeding algorithms, hypercube, mesh, binary tree, 3D, simulation

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Než se pustíme do problematiky paralelních vnořovacích algoritmů, bylo by dobré si definovat některé základní pojmy. Základem jsou paralelní počítače. Zjednodušeně řečeno se jedná o velké množství procesorů, které jsou navzájem propojeny počítačovou sítí.

Paralelní počítače jsou v dnešní době velmi důležité. Pomáhají počítat výpočty, které by na jednom počítači trvaly velmi dlouho. Jako příklad může sloužit předpověď počasí. Při počítání jaké bude počasí se pracuje s velkými daty a pokud by výpočet např. pro následující týden měl na normálním počítači trvat třeba měsíc, by byl už k ničemu. [1]

Způsobů jak mohou být procesory v síti propojeny je velké množství. Některá nejčastější propojení dostala jména podle toho, jak konkrétní sítě vypadají a jaké mají vlastnosti. Podrobněji bude toto vysvětleno v dalších kapitolách. Důležité je, že jednotlivé sítě jdou na sebe navzájem transformovat. Způsoby, jakými se sítě na sebe namapují, mají různé vlastnosti. Některé zachovávají vlastnosti původního paralelního počítače, jiné je mohou zcela změnit.

Např. pokud namapujeme jednu síť na druhou stylem, že se všechny procesory z původní sítě namapují pouze na jeden konkrétní procesor z cílové sítě, pak počítač samozřejmě fungovat bude, ale nebude již tak efektivní při paralelních výpočtech, protože všechnu práci bude mít na starosti pouze 1 procesor.

Naštěstí pro konkrétní sítě existují algoritmy, které dokáží původní počítač namapovat na cílový se zachováním co největšího výpočetního výkonu. Některými vybranými algoritmy se tato práce zabývá.

V následujících kapitolách si nejprve konkrétně definujeme cíl této práce a vysvětlíme si některé základní pojmy. Následně si podrobněji popíšeme vybrané algoritmy, které bude výsledný program umět simulovat. Pak bude následovat analýza technologií, ve kterých by program mohl být napsán a detailní popis té nejvhodnější. V další, nejrozsáhlejší kapitole, je vysvětleno konkrétní fungování programu se zaměřením na zajímavá řešení. Jako poslední bude

následovat popis ovládání programu, výsledky testování programu včetně uživatelského testování a celkové zhodnocení splnění cílů.

Kapitola 1

Cíl práce

Cíl této práce je vytvořit algoritmy, které budou vizuálně simulovat následující algoritmy:

- Vnoření mřížky do mřížky
- Vnoření hyperkrychle do 2D mřížky
- Vnoření úplného stromu do hyperkrychle

Simulované algoritmy budou pro jednotlivé sítě vždy stejné. V této práci tedy nejde o hledání, či vymýšlení nových algoritmů, ale o co nejpřesnější simulaci již existujících.

Pro jednoduchost a lepší názornost budou rozměry jednotlivých sítí omezeny. Je zbytečně složité simulovat vnoření hyperkrychle o dimenzi 10, když na pochopení funkčnosti algoritmu stačí vidět simulaci hyperkrychle o dimenzi 3 nebo 4.

Primární cíl je tedy vytvořit aplikaci, která názorně, nejlépe ve 3D pohledu nasimuluje vybrané algoritmy. Aplikace může rovněž sloužit k pochopení, jak vypadají konkrétní sítě. Před startem každé simulace bude možné si sítě nejdříve vizualizovat a různě měnit úhel pohledu a přiblížení.

Kapitola 2

Vysvětlení základních pojmů

Tento program je primárně určený pro studenty předmětu Paralelní algoritmy a systémy, který se učí na magisterském studiu na FITu[1] na ČVUT[2]. Předpoklad je, že ten, kdo si práci bude číst, by již měl znát některé základní věci jako je třeba model počítače, nebo základy grafových algoritmů. V této kapitole tedy budou vysvětleny pouze věci, které přímo souvisí s touto prací.

Následující sekce obsahuje části volně i doslovně přeložené ze skript Prof. Ing. Pavel Tvrdík, Csc. - Parallel algorithms and computing[2]

2.1 Paralelní počítač

Základní prvek, se kterým se bude v této práci pracovat je tzv. paralelní počítač. Zjednodušeně se jedná o graf, kde uzly zastupují jednotlivé procesory a hrany dráty, kterými jsou procesory propojeny.

To, jakým způsobem jsou procesory mezi sebou propojeny, určuje vlastnosti daných počítačů. Způsobů propojení je mnoho. Nejsou zde popsány všechny, ale jenom ty, které jsou použity v této práci.

2.2 Paralelní vnořovací algoritmus

Paralelní vnořovací algoritmus se dá popsat jako způsob mapování uzlů a hran jednoho grafu na druhý. Cílem vnořovacího algoritmu je, aby byly co nejvíce zachovány vlastnosti původního grafu. Konkrétní vlastnosti jsou popsány níže.

2.3 Typy paralelních počítačů

V následující sekci jsou popsány vybrané typy paralelních počítačů, které se používají v této práci.

[1] FIT - Fakulta Informačních technologií
[2] ČVUT - České vysoké učení technické

2.3.1 Mřížka

Paralelní počítač zapojený do mřížky o dimenzi 2 připomíná pravidelnou obdélníkovou síť. Jiné dimenze mřížek, než 2 se v této práci používat nebudou. Z každého uzlu vedou hrany pouze do přímo sousedních uzlů.

2.3.2 Hyperkrychle

Pro jednoduchou představu - zapojení počítačů do hyperkrychle připomíná hrací kostku (pro krychli o dimenzi 3), kde rohy kostky jsou jednotlivé procesory a hrany jsou dráty, kterými jsou procesory propojené.

Hyperkrychle o dimenzi 3 je speciální případ čtvercové mřížky o dimenzi 3 a rozměrech x=2, y=2, z=2.

Přesnější popis je ten, že pokud uzly po jednom očíslujeme, tak číslo každého uzlu bude mít v binární soustavě nanejvýš stejný počet cifer, jako je dimenze hyperkrychle. Pokud bude počet cifer menší, doplní se zleva nulami tak, aby počet cifer odpovídal dimenzi hyperkrychle. V dalším kroku u čísla kteréhokoliv z uzlů postupně znegujeme všechny bity. Tím vznikne n nových čísel, kde n je počet bitů v čísle uzlu = dimenze hyperkrychle. Tato čísla jsou čísla uzlů, do kterých vedou hrany z vybraného uzlu. Tento postup se postupně aplikuje na všechny uzly.

Př: máme hyperkrychli o dimenzi 4, vezmeme si třeba uzel č.8. Číslo 8 se dá v binární soustavě zapsat jako 1000. Postupným znegováním jednotlivých bitů vzniknou čísla 1001 = 9, 1010 = 10, 1100 = 12 a 0000 = 0. Z uzlu č.8 tedy vedou 3 hrany do uzlů č.9, 10, 12 a 0.

2.3.3 Strom

Strom je grafová struktura splňující následující podmínky.

- Strom má vždy jeden kořen - uzel, který nemá žádný nadřazený uzel
- Neobsahuje cykly
- Uzlům v nejnižší úrovni se říká listy
- Vzdálenost od kořene k nejnižšímu listu se nazývá hloubka stromu

Úplný strom je ten, kdy z každého uzlu, který není list, vede stejný počet hran a všechny listy jsou ve stejné úrovni.

V této práci se bude používat pouze úplný binární strom, tedy takový, kdy každý uzel, který není list má právě 2 potomky. Počet uzlů v takovém stromě je $2^n - 1$, kde n je hloubka stromu.

2.4 Vlastnosti vnořovacích algoritmů

Zde jsou popsány vlastnosti, které se v programu počítají - u každé simulace budou zobrazeny jejich konkrétní hodnoty. Vlastností je samozřejmě více, ostatní zde ale popsány nejsou.

2.4.1 Zatížení - LOAD

Zatížení jednoho cílového uzlu je počet uzlů původní sítě, které se na jeden konkrétní uzel namapují. Výsledný load vnoření je největší zatížení jednoho uzlu v cílové síti.

Nejlepší případ je, když je celkové zatížení vnoření rovno konstantě 1. To znamená, že se každý uzel z původní sítě namapuje právě na 1 uzel v síti cílové.

Horší, ale stále dobrý případ je, když je zatížení konstantní. To znamená, že má pokaždé stejnou hodnotu, bez ohledu na rozměry sítí, co se do sebe vnořovaly.

Nejhorší případ může nastat, pokud load přímo souvisí s velikostmi sítí. Tato situace např. nastane, pokud bychom všechny uzly z původní sítě namapovali na jeden konkrétní uzel. Load by měl hodnotu stejnou jako je počet uzlů zdrojové sítě.

2.4.2 Linkové zahlcení - ECNG

Zahlcení jedné cílové hrany je počet hran zdrojového grafu, které se na konkrétní hranu namapují.

Hranové zahlcení vnoření je největší zahlcení jedné cílové hrany.

Opět, stejně jako u zatížení platí, že čím menší je hodnota zahlcení, tím lépe. Opět nejlepší je, pokud je hodnota konstantní a nesouvisí s velikostmi sítí.

Čím je tato hodnota větší, tím pomalejší bude komunikace mezi procesory v cílové síti. Pokud bude mít jedna hrana hodnotu ECNG 2, pak komunikace po této hraně bude dvakrát pomalejší, než by byla ve zdrojové síti.

2.4.3 Dilatace - DIL

Dilatace neboli protažení udává, o kolik se jedna hrana zdrojového grafu vedoucí mezi dvěma sousedními uzly „protáhne“ při namapování na cílový graf. Pro lepší pochopení poslouží příklad: pokud budou původně sousední uzly ve zdrojové síti po vnoření do cílové také sousední, pak bude dilatace hrany 1. Pokud mezi nimi bude jeden další uzel, pak se původní hrana bude muset zdvojnásobit a dilatace bude 2. Pokud budou mezi nimi 2 uzly, pak se hrana ztrojnásobí a její dilatace bude mít hodnotu 3, atd.

Dilatace vnoření je největší dilatace jedné hrany.

Kapitola 3

Detailní popis vnořovacích algoritmů

Vnořovacích algoritmů existuje velké množství. Neexistuje jeden univerzální algoritmus pro všechny typy sítí, který by zajistil co nejlepší parametry vnoření. Vždy záleží na tom, které sítě se do sebe vnořují. Zde jsou tedy popsány pouze algoritmy, které se vyskytují v programu.

3.1 Vnoření mřížky do mřížky

Algoritmů vnoření mřížky do mřížky existuje víc. V této práci jsou simulovány dva vybrané.

3.1.1 Obdélníková do čtvercové

Obdélníková mřížka se namapuje na čtvercovou postupně z levého horního rohu směrem doprava. V momentě, kdy dorazí na konec čtvercové mřížky, tak dojde k dvojímu „přehnutí“ obdélníkové sítě (nejdříve směrem dolů, potom ihned ve směru doleva) a její mapování pokračuje dalším řádkem z druhé strany. V místě přehybu dojde k situaci, kdy některé uzly mají load 2. Tím pádem celé vnoření má load 2. Toho se dá elegantně využít tak, že na rovné části se uzly budou rovnou mapovat po dvou. Celkový load se tím nezvýší, ale zvětší se velikost obdélníkové mřížky, která se vejde do dané čtvercové.

Způsobů, jakým lze obdélníkovou mřížku otáčet je několik, výše popsaná možnost je použita v programu. Dále lze mřížku mapovat tak, že se po prvním přehnutí bude mřížka mapovat ve směru dolů, jakmile dorazí na konec, tak ve směru doleva a poté nahoru. V momentě, kdy dojde na místo, kde už je namapovaná část, tak se síť znovu přehne a pokračuje opět ve směru doprava. Dá se zjednodušené říci, že je to had, který se roluje zvenku směrem do středu. Další možnost je mapování nejdříve shora dolů, a střídavě nahoru a dolů.

Parametry tohoto vnoření jsou: load: 1, dil: 2, ecng: 2 [2]

3.1.2 Čtvercová do obdélníkové

Čtvercová mřížka se do obdélníkové mapuje poměrně jednoduše. Základ je rozdělit původní čtvercovou na sloupce a ty jednotlivě mapovat na obdélníkovou mřížku. Mapovací algoritmus je pro všechny sloupce úplně stejný.

Algoritmus funguje tak, že se sloupec začne mapovat na mřížku stejně, jako kdyby se jednalo o mapování jedna ku jedné. Levý horní uzel zdrojové mřížky se namapuje na levý horní uzel cílové mřížky. Uzel pod ním se namapuje hned pod něj atd. Změna nastává v momentě, kdy uzly naplní první sloupec cílové mřížky. Ta má menší výšku než zdrojová, tudíž je zapotřebí mapování v tento moment změnit. To se vyřeší tím, že dojde u cílové mřížky k posunu o jeden sloupec doprava a mapování pokračuje stejným způsobem zespoda nahoru. Pokud je zdrojová mřížka hodně vysoká a nestačí v cílové 2 sloupce, tak při naplnění dalšího sloupce dojde opět k posunu o jeden sloupec doprava a mapování pokračuje tentokrát shora dolů.

Mapování dalšího sloupce zdrojové mřížky začíná na první pozici nejbližšího volného uzlu v cílové mřížce napravo od toho, kde skončilo mapování předchozího sloupce. Je to z důvodu zachování co nejmenší dilatace hran.

Tento algoritmus může mít mnoho variant. Aby byla dilatace co nejmenší, je možné uzly mapovat po dvou. Tím pádem může být cílová mřížka menší, než pokud by byl load 1.

Parametry tohoto vnoření jsou: load: 2, dil: $\left\lceil \sqrt{\frac{z1}{z2}} \right\rceil$, ecng: $1 + \left\lceil \sqrt{\frac{z1}{z2}} \right\rceil$ [2]

3.2 Vnoření hyperkrychle do 2D mřížky

Použitý algoritmus pro vnoření hyperkrychle do 2D mřížky používá Mortonovu křivku. Jedná se o střídavé mapování uzlů v lexikographickém pořadí střídavě ve směru X a Y. Toto mapování je rekurzivní. Při pohledu na mřížku připomíná opakující-se písmeno *Z*.

Parametry tohoto vnoření jsou: load: 1, dil: 2^{k-1} [2]

3.3 Vnoření úplného stromu do hyperkrychle

K tomu, aby bylo možné vnořit úplný binární strom do hyperkrychle je zapotřebí použít malý trik. Pomocí zdvojení kořenu se nejprve vytvoří vyvážený bipartitní graf s 2^{n+1} uzly. Nejprve najdeme vnoření levé části grafu do hyperkrychle o jeden rozměr menší. Následně stačí najít automorfismus daného vnoření ve druhé hyperkrychli a kořeny navzájem propojit. Tento způsob mapování je rekurzivní.

Parametry tohoto vnoření jsou: load: 1, dil: 2, ecng: 1 [2]

Kapitola 4

Možnosti řešení

Dnešní nabídka moderních technologií skýtá velké množství možností, v čem je možné aplikaci vytvořit.

4.1 Aplikace na PC

Pokud by výsledný program fungoval jako aplikace na PC, nebyl by problém si perfektně vyhrát s grafikou tak, aby byl výsledný dojem pro uživatele co nejlepší. Program by fungoval na stejném principu jako 3D videohry s tím rozdílem, že místo zabíjení nepřátel by uživatel koukal na 3D animace algoritmů.

Nevýhoda je, že by se jednalo o software, který by bylo potřeba nějak instalovat, což může některé uživatele odradit. Dále by bylo nutné řešit kompatibilitu pro různé operační systémy. Vzhledem k tomu, že je program určen pro studenty, kteří pro různé předměty mohou používat různé operační systémy, tak by program optimalizovaný pro jeden systém byl pro mnoho studentů nedostupný.

Řešení by samozřejmě byla optimalizace, aby program fungoval všude, nicméně to by zbytečně zvyšovalo složitost tvorby programu a čas investovaný do optimalizace by bylo samozřejmě lepší investovat do přehlednosti samotných animací.

4.1.1 Možné technologie

V této části jsou popsány možné technologie, ve kterých by bylo možné program vytvořit.

4.1.1.1 OpenGL

Jedna z možností, jak program napsat jako aplikaci na PC je přímé využití technologie OpenGL.

OpenGL[3] je průmyslový standard specifikující multiplatformní rozhraní (API) pro tvorbu aplikací počítačové grafiky. [3]

Pro potřeby tohoto programu je tento nástroj zbytečně složitý, hodí se spíše k tvorbě her, nebo programů, které obsahují složitější animace náročné na výpočetní výkon. Tato aplikace obsahuje jednodušší animace a navíc se program skládá ze dvou částí - jedna pro uživatelské rozhraní, a druhá pro samotnou animaci. Technologie OpenGL řeší hlavně tu část s animací.

4.1.1.2 3D engine

3D engine je softwarový balík obsahující třídy a funkce pro většinu herních komponent, které se ve většině her opakují. Není tak potřeba je pokaždé programovat znovu. Příklady takových funkcí jsou detekce kolizí, načítání modelů, zobrazování scény, fyzické vlastnosti objektů a další. [4]

Použití již hotového enginu značně usnadňuje samotné programování a umožňuje to vývojářům zabývat se přímo konkrétními problémy dané aplikace a ne tím, jak danou věc vůbec naprogramovat.

Z praktického hlediska je toto lepší volba, než vytvářet aplikaci v OpenGL, ale má i své nevýhody.

Jedna z nevýhod je, že se jedná o zbytečně robustní nástroj. V této práci jde o to vytvořit co nejpřehlednější animace, které by měly běžet co nejefektivněji a pokud možno na každém počítači. Aplikace vytvořené v enginu jsou samy o sobě náročné díky tomu, že daný engine již obsahuje mnoho funkcí, které se ve 3D hrách běžně používají, pro účely této práce jich je ale většina zbytečná.

Další nevýhoda je licence. V současnosti může být engine zdarma pro nekomerční účely, ale může se kdykoliv stát, že se firma vyvíjející engine rozhodne, že bude chtít peníze i za nekomerční verze.

4.2 Webová aplikace

Vytvoření aplikace pro web má jednu velkou výhodu, a to tu, že uživatel nepotřebuje nic instalovat, stačí mu pouze otevřít prohlížeč a zadat do něj patřičnou URL[4] adresu. Dále bude aplikace fungovat na všech operačních systémech. Uživateli k přehrání animace stačí pouze mít fungující webový prohlížeč.

Vytvoření aplikace pro web má další výhodu v tom, že jde snadno oddělit rozhraní pro ovládání od samotné animace. Jako ovládací prvky se dají použít standartní HTML[5] komponenty a část s animací bude ovládat speciální technologie pro to určená.

[3] Open Graphics Library
[4] Uniform Resource Locator
[5] HyperText Markup Language

4.2.1 Možné technologie

V této části je seznam možných technologií pro animace. Na rozhraní pro ovládání stačí základní webové technologie HTML a CSS[6].

4.2.1.1 Základní animace v CSS3

Technologie CSS3 umožňuje vytváření jednoduchých animací. Tyto animace jsou primárně určeny pro různé efekty na webu, kde není potřeba použití dalších technologií. V základu využívají css atribut *transform*. Pomocí tohoto atributu lze u prvku měnit některé vlastnosti. [5]

- *translate(10px, 20px)*
  posune prvek ve směru osy x o $10px$ a ve směru osy y o $20px$.
- *rotate(90deg)*
  otočí prvek o 90 stupňů
- *scale(2)*
  zvětší prvek o dvojnásobek původní velikosti

Samotné animace mají speciální parametry, které nastavují jejich vlastnosti. [6]

- *transition-duration*
  délka animace zadaná v sekundách
- *transition-delay*
  specifikace zpoždění před začátkem animace
- *transition-propert*
  specifikace css vlastnosti, na kterou se bude aplikovat animace
- *transition-timing-function*
  volba funkce, která se použije pro výpočet průběhu animace [7]
  - *ease*
    Výchozí hodnota. Určuje přechodový efekt s pomalým startem, pak zrychlí a pžed koncem zpomalí
    (ekvivalent beziérovy křivky (0.25,0.1,0.25,1))
  - *linear*
    Určuje přechodový efekt se stejnou rychlostí od začátku do konce (ekvivalent beziérovy křivky (0,0,1,1))
  - *ease-in*
    Určuje přechodový efekt s pomalým startem (ekvivalent beziérovy křivky (0.42,0,1,1))

[6] Cascading Style Sheets

- *ease-out*
  Určuje přechodový efekt s pomalým startem (ekvivalent beziérovy křivky (0.42,0,1,1))
- *ease-in-out*
  Určuje přechodový efekt s pomalým začátkem a koncem (ekvivalent beziérovy křivky (0.42,0,0.58,1))
- *steps(int,start/end)*
  Určuje funkci stupňování, se dvěma parametry. První parametr udává počet intervalů ve funkci. Musí to být kladné celé číslo (větší než 0). Druhý parametr, který je volitelný, je buď hodnota *start* nebo *end*, a určuje bod, ve kterém změna hodnot v intervalu nastane. V případě, že druhý parametr je vynechán, je použita hodnota *end*
- *step-start*
  Ekvivalent k steps(1, start)
- *step-end*
  Ekvivalent k steps(1, end)
- *cubic-bezier(n,n,n,n)*
  Definuje vlastní beziérovu křivku. Možné hodnoty jsou číselné hodnoty od 0 do 1

Složitější animace se vytváří pomocí atributu *@keyframes*. Ten slouží k definici jednotlivých kroků animace. V každém kroku je nutné zadat procento v průběhu animace a vlastnosti prvku, jaké má v dané době mít. [8]

V CSS je tedy možné při troše šikovnosti vytvářet i relativně složité animace, nicméně pro účel této práce tato technologie není nejvhodnější. CSS animace je v základu dobré používat pro jednodušší efekty, kdy je zapotřebí pouze upravit vlastnosti prvku ve 2D. Vytvářet v CSS interaktivní 3D animaci by bylo velmi časově náročné.

4.2.1.2 SVG

SVG[7] je technologie pro tvorbu vektorové grafiky pomocí XML[8] kódu. Grafika se skládá z jednoduchých automatických tvarů (čtverec, kruh, čára, polygon...). Každý takový tvar má v XML speciální značku. Pomocí atributů se mu dají nastavit specifické vlastnosti jako třeba barva, rozměry, pozice a další.[9]

SVG může být použito jako obrázek v externím souboru, nebo přímo jako součást HTML kódu. Některé vlastnosti jednotlivých prvků obrázku jdou nastavovat pomocí CSS. Výhoda toho, že se dá SVG použít přímo v HTML kódu je možnost jeho kód generovat dynamicky.

[7] Scalable Vector Graphics
[8] eXtensible Markup Language

Pomocí SVG jdou vytvářet animace. Fungují podobně jako u CSS animací, s tím rozdílem, že vlastnosti prvků a animací se zadávají přímo jako součást XML. Pro animace existuje tag *<animate>*, který se umístí přímo k prvku, na který se má aplikovat. Pomocí atributů tagu animate se určují vlastnosti konkrétní animace jako je její délka, zpoždění, změny prvku atd.[10]

4.2.1.3 WebGL

WebGL[9] je JavaScriptové API pro akcelerované vykreslování grafiky do HTML canvasu; rozhraní je navrženo tak, aby bylo architektonicky identické s OpenGL ES 2.0.[11]

Na rozdíl od CSS a v SVG se jedná o technologii primárně určenou pro složitější scény a animace.

4.2.1.4 WebGL framework

Použití přímého WebGL je relativně složité. Naprogramovat v něm jednoduchý příklad zabere poměrně velké množství kódu.[11] Naštěstí existují frameworky, které většinu základních komponent, které jsou pro vývoj 3D aplikace potřeba, mají už připravených. Jedná se o podobné věci, jako jsou 3D enginy pro vývoj 3D aplikací pro PC verze.

Na rozdíl od 3D enginů ale nejsou tak robustní. Jedná se většinou o balík funkcí, které je možné volat z javascriptového kódu. Existuje jich poměrně velké množství, kde základní funkce potřebné pro vytvoření 3D aplikace obsahuje většina z nich.[12]

[9] Web Graphics Library

Kapitola 5

Realizace

Tato část se detailně věnuje samotnému vývoji aplikace. Jsou zde rozebrány použité technologie a způsob, jakým je program vytvořen. Zvlášť zde jsou rozebrány zajímavé části programu.

5.1 Použité technologie

V této části jsou představeny veškeré technologie, které jsou v aplikaci použity. Největší důraz je kladen na technologie pro tvorbu 3D animací.

5.1.1 Framework pro práci s 3D animací

Pro implementaci této úlohy byl nakonec vybrán WebGL framework Three.js.

Three.js je javascriptový framework umožňující pohodlnou práci s WebGL. Jedná se o balík již vytvořených funkcí, které řeší standartní problémy, s kterými se člověk při programování 3D aplikace potýká. Příklady takových funkcí jsou například různá ovládání kamery, základní 3D prvky, několik základních druhů světla a další.

Mezi druhy světla patří například směrové světlo - simulace slunce, všechny paprsky svítí rovnoměrně pod určitým úhlem a jsou rovnoběžné. Dále pak bodové světlo - simulace rozsvícené lampy, světlo svítí rovnoměrně do všech stran z jednoho bodu a také světlo napodobující reflektor - svítí z jednoho bodu pouze v určitém směru. Další z důležitých komponent, co již ve frameworku jsou, jsou různé materiály - každý z nich má jiné vlastnosti odlesku, což určuje, jak realisticky bude výsledná scéna vypadat. [13]

Není cíl zde do detailu popisovat všechny funkce dané knihovny, ale pouze pro představu nastínit, co knihovna umí a proč je vybrána pro řešení této práce.

Člověk tedy nemusí znovu programovat tyto základní funkce a může se rovnou soustředit na požadované unikátní chování výsledného programu.

Navíc, díky tomu, že se jedná pouze o relativně malou knihovnu a ne celý 3D engine, tak tato knihovna nebude zbytečně zpomalovat běh programu na starších strojích.

5.1.1.1 Konkrétní použité části

Aplikace, která je cílem této práce má co nejjednodušeji a nejpřehledněji simulovat algoritmy. Tudíž není potřeba použít všechny funkce, které knihovna nabízí, ale jenom některé z nich.

- **3D objekty** Z připravených objektů je použit objekt koule pro zobrazení uzlu a válec pro zobrazení hrany. Na hrany sítě, která se animuje, je použit objekt čáry. Válec se totiž při vykreslování podle zadaných rozměrů vykreslí pomocí trojúhelníků, které jsou mezi sebou propojeny tak, aby výsledný objekt měl tvar válce. Animace takového objektu by byla výpočetně náročná, protože by bylo potřeba při každém obnovení scény přemisťovat všechny body, ze kterých se objekt skládá. U objektu čáry se pouze definuje počáteční a cílový bod, a během animace stačí tyto body aktualizovat. Animace objektu čára je tedy výpočetně výrazně jednodušší a rychlejší.[13]

- **Světelné objekty**

  Světla jsou v programu použita velice jednoduše, pouze pro to, aby byly dobře vidět jednotlivé uzly a scéna vypadala hezky. Konkrétně se jedná o jedno ambientní světlo (osvětluje všechny plochy ze všech stran rovnoměrně stejnou barvou) a dvě světla simulující reflektory (spotlight).[13]

- **Kamera**

  Knihovna Three.js disponuje několika různými kamerami. Konkrétně CubeCamera, OrthographicCamera, PerspectiveCamera. V této práci je použita PerspectiveCamera spolu s funkcí zvanou Trackball. Jedná se o funkci, se kterou lze pomocí myši libovolně měnit úhel pohledu na daný model. Pro tuto situaci je volba této kamery nejlepší. Nehodí se to ale všude, pokud by někdo programoval hru, ve které by chodil s postavičkou a kamera by simulovala její pohled, tak by se měl správně použít jiný typ kamery a ovládání. [13]

- **Raycast**

  V českém překladu se jedná o sledování paprsku. Funkce sleduje pozici kurzoru myši a tu mapuje na souřadnice ve scéně. Pokud zjistí, že je kurzor nad určitým objektem, vyvolá se událost, při které dojde k aktualizaci daného objektu. Toto je použito při najetí myší nad konkrétní uzel, který se rozsvítí a s ním automaticky i další v druhé síti, na který se bude tento konkrétní mapovat. [13]

5.1.2 Standartní webové technologie

Jak již bylo zmíněno výše, celá aplikace bude k dispozici přímo na webu, bez nutnosti cokoliv instalovat. K tomu je ale potřeba použití dalších technologií, které umožňují webovou aplikaci vytvořit.

Tyto technologie jsou v aplikaci použity pro vytvoření uživatelského rozhraní, pomocí kterého bude možné zadávat parametry animací a samotné animace ovládat.

5.1.2.1 HTML

HTML - HyperText Markup Language je značkovací jazyk pro vytváření webových aplikací. Používá XML strukturu tvořenou ze speciálních HTML tagů. [14]

5.1.2.2 CSS

CSS - Cascading Style Sheets je speciální jazyk sloužící pro nastavení vzhledu určitým html tagům. V základu se dá pomocí css nastavit u každého prvku jeho pozice a styl jako např. velikost, barva, určité specifické chování po najetí myší atd. [15]

5.1.2.3 Javascript

Javascript je programovací jazyk pro použití na webových stránkách.[16]
V této aplikaci je použit nejvíce. Je v něm napsané jádro celého programu.

5.1.3 Další knihovny a frameworky

Kromě základních webových technologií jsou v aplikaci použity i některé frameworky a knihovny usnadňující programování, nebo stylování komponent.

5.1.3.1 SASS

SASS[10] je rozšiřující knihovna pro lepší psaní CSS stylů. Umožňuje například definovat v CSS proměnné, do kterých se uloží třeba barvy, nebo velikosti. Klasické CSS to neumí a když je např. jedna barva na webu použita na více místech, tak je nutné ji pokaždé znovu psát přímo jejím kódem. To samo o sobě nevadí, problém ale nastává, pokud je potřeba danou barvu změnit. V klasickém CSS je nutné projít všechny soubory a daný kód barvy nahradit. Pokud se použije SASS, pak stačí pouze přepsat hodnotu jedné proměnné a problém je vyřešen. Knihovna také umožňuje lépe specifikovat styly jednoduchým zanořováním do sebe, případně psát vlastní funkce (v terminologii SASSu to jsou mixiny).[17]

[10]Syntactically Awesome Style Sheets

5.1.3.2 jQuery

jQuery je javascriptový framework, který umožňuje efektivnější psaní kódu. V této práci je použit pro ovládání programu a tlačítek pro animace. Umožňuje jednoduše definovat události s tlačítky (např. kliknutí) a následně zavolat akci, která se má provést.[18] Na jádro programu - samotné animace je použit pouze framework Three.js.

5.1.3.3 Web optimization

Web optimization je modul od Microsoftu vytvořený pro vývoj aplikací v ASP.NET, který vezme všechny styly a scripty a vytvoří z nich jeden soubor, který obsahuje minifikovaný kód.[19] To, že je kód minifikovaný vůbec nezmění jeho funkčnost. Znamená to pouze, že jsou dlouhé názvy proměnných nahrazeny jedno či dvou znakovými a jsou odstraněny zbytečné komentáře, mezery a odřádkování. Pro člověka je takový kód nečitelný - je celý v jednom řádku, ale zmenší se tím velikost přenášených dat ze serveru a počet HTTP požadavků. Místo několika souborů se načítají pouze 2 - jeden pro styly a druhý pro scripty.

Modul také umožňuje automatické načítání scriptů a CSS souborů. Programátor může například definovat, ze které složky se budou scripty načítat a modul obstará jejich vložení do hlavičky HTML stránky. Při vytvoření nového souboru tedy není zapotřebí ho pokaždé ručně vkládat do HTML stránky. [19]

5.1.4 Další komponenty

5.1.4.1 Ikony

U tlačítek pro ovládání animací jsou použity ikonky ze sady Glyphicons - http://glyphicons.com/. V originální formě by bylo nutné na ně koupit licenci, ale díky tomu, že jsou součástí CSS frameworku Bootstrap, tak je možné jejich použití zadarmo.[20]

5.2 Části programu

Celá aplikace je pro větší přehlednost rozdělena do několika modulů, které na sobě navzájem tolik nesouvisí a úprava některého z nich ovlivní ostatní jen minimálně.

5.2.1 Grafické rozhraní pro uvládání

Grafické rozhraní celé aplikace je navrženo tak, aby bylo co nejpřehlednější a jakýkoli uživatel dokázal program bez problémů ovládat bez speciálního návodu.

Tato část bude podrobněji popsána v kapitole Ovládání programu.

5.2.2 Modul pro vykreslování komponent

Tato část používá knihovnu Three.js. Obsahuje tak trochu „herní“ strukturu. Co to konkrétně znamená, bude vysvětleno níže. Hlavní úkol tohoto modulu je zajistit co nejlepší zobrazování animací, které budou zadány jako parametr ve speciální datové struktuře. Tento modul tedy vůbec nepočítá konkrétní vnoření, pouze je dostane zadané a zajistí potřebnou animaci.

5.2.3 Modul pro výpočty vnoření

Tento modul je naprogramovaný v čistém javascriptu, bez dalších frameworků. Obsahuje výpočty pro definování sítí a funkce pro mapování jednotlivých uzlů na sebe při vnoření.

Tato část původně nemusela být vůbec naprogramovaná v javascriptu, ale klidně v jakémkoliv jiném jazyce. Modul, který má na starosti zobrazování by si jenom po sítí zavolal URL, na kterém by služba běžela a dané rozhraní by vrátilo podle požadavku patřičnou definici sítě nebo vnoření. Nicméně by v programu byla zbytečná komunikace se serverem, která by jednak mohla běh programu zpomalovat a za druhé by byla na serveru potřeba speciální technologie, která by podporovala jazyk, ve kterém by byl modul vytvořený - PHP nebo ASP.NET.

Největší výhoda tohoto přístupu je ta, že v momentě, kdy by se program někdy v budoucnu rozšiřoval třeba o nové algoritmy, tak by stačilo pouze připsat další vnořovací algoritmus a nebylo by nutné kvůli tomu měnit zbytek programu a zbývající moduly.

5.3 Zajímavé struktury/řešení

5.3.1 Předávání stavu

Javascript má jednu krásnou vlastnost, a to tu, že se dají ukazatele na funkce ukládat do proměnných. Z jednoho místa poté volám pokaždé tu samou funkci uloženou v proměnné, ale v závislosti na aktuálním stavu hry se provede jiný kód.

Účel této konstrukce je jednoduchý, a to zbavit se v kódu zbytečných podmínek, které by se vyhodnocovaly při každém obnovení scény. Nyní se vyhodnotí pouze v nezbytně nutných případech a to při změnách stavů.

Lépe to asi bude vidět na následujícím příkladu.

Listing 5.1: State.js

```
GAME.Helpers.State = {
    "PLAY": 1,
    "PAUSE": 2,
```

```
    "PLAY_BACK": 3,
    "PLAY_BACK_SINGLE": 4,
    "PLAY_SINGLE": 5,
    "PLAY_BACK_SINGLE_STEP": 6,
    "PLAY_SINGLE_STEP": 7
}
```

Soubor State.js obsahuje pouze číselník stavů hry/programu. Tento soubor by v programu nemusel být, stavy by se mohly třeba dynamicky načítat z databáze, nebo používat přímo čísla stavů místo textových pojmenování. Program by pak ale byl výrazně méně přehledný.

Listing 5.2: Game.js

```
...

GAME.refresh = function () {
    for (var i = 0; i < GAME.items.length; i++) {
        var item = GAME.items[i];
        item.refresh();
    }
}

GAME.setState = function (newState) {
    GAME.state = newState;

    for (var i = 0; i < GAME.items.length; i++) {
        var item = GAME.items[i];
        item.setState(newState);
    }
}

...
```

Soubor Game.js obsahuje celkově více funkcí, důležité pro nás jsou aktuálně funkce refresh a setState. Funkce GAME.refresh se volá při každém obnovení scény. Objekt GAME.items obsahuje všechny herní prvky, každý z nich musí implementovat funkce setState a refresh. Toho se docílí tím, že každý objekt, který je v programu použit, dědí od třídy BaseObject.

Listing 5.3: BaseObject.js

```
...

GAME.Objects.BaseObject.prototype.refresh = function () {

}

GAME.Objects.BaseObject.prototype.setState = function (newState) {
```

```
    if (newState == GAME.Helpers.State.PLAY) {
        if (this.refreshPlay != undefined) {
            this.refresh = this.refreshPlay;
        }
    }
    if (newState == GAME.Helpers.State.PLAY_BACK) {
        if (this.refreshPlayBack != undefined) {
            this.refresh = this.refreshPlayBack;
        }
    }
    if (newState == GAME.Helpers.State.PLAY_BACK_SINGLE) {
        if (this.refreshPlayBackSingle != undefined) {
            this.refresh = this.refreshPlayBackSingle;
        }
    }
    ...

}

...
```

Od tohoto objektu dědí úplně všechny herní objekty. Na ukázce je vidět, že každý objekt má v základu implementované funkce refresh a setState. Ostatní implementovat nemusí. Pokud objekt danou funkci nemá implementovanou, pak ke změně z původní nedojde. To se týká jednodušších objektů, například uzlů, nebo hran.

Listing 5.4: Player.js

```
...

GAME.Controllers.Player.prototype.refresh = function () {
    this.refreshDefault();
}

GAME.Controllers.Player.prototype.refreshDefault = function () {
    for (var i = 0; i < this.items.length; i++) {
        var item = this.items[i];
        item.refresh();
    }
}

GAME.Controllers.Player.prototype.refreshPause = function () {
    this.refreshDefault();
}

GAME.Controllers.Player.prototype.refreshPlay = function () {
    this.refreshDefault();
    this.dynamicSite.moveAll();
```

```
}

GAME.Controllers.Player.prototype.refreshPlayBack = function () {
    this.refreshDefault();
    this.dynamicSite.moveAllBack();
}

...
```

Ukázka objektu, který má na starosti ovládání animací a vytváření sítí. Implementuje všechny stavové funkce (v ukázce je pouze několik z nich) a v každé z nich předává svým sítím jiné instrukce, jak se mají animovat.

Hlavní výhoda tohoto přístupu je velká přehlednost kódu. Každá funkce je odpovědná za chování v konkrétním stavu hry. Hlavně je to o mnoho čistší řešení, než mít pro refresh pouze jednu funkci, ve které by byly podmínky na stav hry.

5.3.2 Raycast

Raycast, neboli sledování paprsku, je speciální funkčnost sloužící pro interakci myši s předměty ve scéně. Základ vychází z třídy THREE.Raycaster z knihovny Three.js. Při každém obnovení scény se volá její metoda setFromCamera, která dostane v parametrech aktuální pozici myši a objekt kamery. Z těchto dat spočítá a vrátí seznam herních objektů, nad kterými je aktuálně kurzor.

Každý z těchto aktivních objektů následně zavolá funkci svého rodiče pro detekci najetí kurzorem nad objekt. Spolu s tím předá ještě své ID a informaci, jestli se jedná o zdrojový, nebo cílový uzel. Jakmile se informace dostane k objektu Player, který má na starosti celkovou animaci, tak se podle ID zjistí všechny uzly, které s aktuálním souvisí. Pokud je uzel ze zdrojové sítě, najde se uzel, do kterého bude daný směřovat. Pokud je z cílové sítě, pak se najdou všechny uzly ze zdrojové, které se na něj při vnoření namapují. U všech těchto uzlů se zavolá metoda lightOn, která zajistí jejich zvýraznění - změnu barvy.

Tento princip funguje úplně stejně také na hrany. Důležité je v této fázi již mít spočítaný seznam odpovídajících uzlů a hran.

5.3.3 Výpočet souvisejících hran

Výpočet mapování jednotlivých uzlů provádí speciální funkce, která na základě čísla uzlu z jedné sítě spočítá číslo uzlu z druhé sítě, na který se bude konkrétní uzel mapovat. Podrobněji je to popsáno v další části této kapitoly.

U mapování hran to takto nefunguje. Hrany sice mají svá čísla, ale ta závisí na tom, v jakém pořadí se generují při vykreslování sítě. Důležité je, že každá hrana má u sebe uložena čísla uzlů, mezi kterými vede.

Toho je využito při výpočtu jejich mapování. Nejprve se vezme každá hrana ze zdrojové sítě. Musí to být ze zdrojové, protože počet hran v jednotlivých sítích je různý, a pokud je hran v cílové síti víc, zůstanou některé neobsazené, kdežto hrany ze zdrojové sítě se na cílovou musí namapovat všechny. Z každé hrany se vezmou čísla uzlů, které hrana propojuje. Pro každý z těchto uzlů se zjistí uzel v cílové síti, na který se uzel namapuje. Toto je již předpočítané, výpočet mapování hran se provádí, jakmile se dokončí výpočet mapování uzlů. V jednoduchém případě, pokud jsou uzly v cílové síti také sousední, stačí najít hranu, která mezi nimi vede, zjistit její id a to vrátit jako odpovídající hranu k té původní.

Složitější případ nastává u vnoření, která mají dilataci větší než 1. V obecném případě je v takový moment zapotřebí najít nejkratší cestu skrz graf, přes kterou se namapuje původní hrana. Počítat toto pro obecný graf by bylo složité. Naštěstí v této úloze tato situace nastává pouze u několika málo vnoření. Jedná se pokaždé o vnoření do 2D mřížky. Vnoření do hyperkrychle má dilataci 1. Navíc každá takto natažená hrana je natažená pouze v jednom směru. Směr může být buďto v ose x nebo y. Osa z u 2D mřížek není využita. Pokud tedy taková situace nastane, je nutné nejprve spočítat jaká je mezi cílovými uzly vzdálenost. Vzhledem ke struktuře s jakou se generuje kostra mřížky, je zjištění této informace jednoduché. Stačí porovnat souřadnice bodů a spočítat rozdíl té, která se jako jediná liší. Sousední uzly se liší v souřadnicích pokaždé o jedničku. Pokud tedy rozdíl vyjde např. 3, znamená to, že dilatace hrany je 3. Další výpočet závisí na tom, v jakém směru je hrana protažena.

Uzly v mřížce se číslují z levého horního rohu směrem dolů a poté doprava. Pokud se uzly liší v souřadnici ve směru y, pak k nalezení čísel uzlů, které jsou mezi nimi stačí k číslu původního uzlu postupně přičítat jedničku. Pokud se liší v souřadnici ve směru x, pak je nutné k číslu počátečního uzlu přičítat výšku cílové mřížky.

Tímto způsobem se najdou čísla všech uzlů, která jsou „na trase“ mezi původníma dvěma uzly ze zdrojové sítě. Pro každou dvojici mezilehlých uzlů se následně najde hrana, která mezi mini vede a přidá se do seznamu hran, na které se původní hrana namapuje.

Díky tomuto postupu je velice snadné spočítat parametry vnoření. Dilatace je největší vzdálenost mezi dvěma cílovými uzly a linkové zahlcení je největší počet hran, které se namapují na jednu cílovou hranu.

5.4 Návrh tříd

V této části jsou vysvětleny funkce jednotlivých tříd. Třídy jsou rozděleny do podkapitol podle modulů, ve kterých se v programu nacházejí.

5.4.1 Modul Game

Struktura tříd v tomto modulu není psaná speciálně pro simulaci vnořovacích algoritmů. Jedná se spíše o univerzální kostru, kterou je možné při relativně málo úpravách použít pro jakýkoliv další herní projekt.

5.4.1.1 Game.js

Třída GAME je mozek celé hry/aplikace. Je to statická třída, která inicializuje veškeré potřebné komponenty. Obsahuje také konfiguraci programu - nastavení barev uzlů, nastavení rychlosti, cestu k souborům použitým v programu a výchozí rozměry oblasti, ve které se vykresluje animace.

Třída obsahuje funkce pro detekci pohybu myši, detekci změny velikosti okna, funkci pro obnovení scény, funkci pro vykreslení scény a pomocné funkce pro vytváření potřebných objektů a změny stavů.

Při načtení programu se zavolá funkce init, která nejprve zjistí aktuální velikost okna a podle toho přepočítá konfigurační proměnné. Dále vytvoří objekty pro scénu a renderer, tomu následně předá všechny potřebné parametry, jako jsou rozměry a barva pozadí scény. Renderer poté napojí na html prvek, ve kterém se vykreslí samotná animace.

Jakmile je scéna připravená, je potřeba vytvořit prvky, které jsou pro fungování animace zapotřebí. První takový prvek je kamera, další světla, raycaster, inicializace listenerů pro pohyb myší a změnu velikosti okna a načtení písma, které je použito u zobrazení čísel uzlů.

5.4.1.2 Controllers

Ve složce Controllers jsou třídy, které mají na starosti a ovládají více podřazených objektů. Typicky se jedná o ovládání ucelenějších komponent obsahujících předem neznámé množství prvků, které spolu souvisí.

V této práci se jedná konkrétně o třídu pro ovládání kamery a třídu fungující jako hlavní ovladač animace.

5.4.1.3 Controllers/Player.js

Třída Player zajišťuje hlavní řídící mechanizmus všech animací. Na začátku dostane informace o tom, jaké sítě se do sebe budou vnořovat a také jejich rozměry. Potom dojde k vygenerování potřebných sítí. Generují se přesně tři sítě. Dvě, které budou statické - šedou barvou, jedna bude mít pozici startovní, druhá cílové a nebudou se animovat. Třetí bude mít červenou barvu a bude se animovat.

Třída také implementuje všechny stavové funkce a podle nich nastavuje konkrétní animace.

5.4.1.4 Controllers/CameraController.js

Třída CameraController má na starosti vytvoření objektu THREE.PerspectiveCamera pro kameru. Obsahuje i nastavení parametrů kamery takové, aby byl pohled na výslednou animaci co nejlepší. Kromě samotné kamery obsahuje i nastavení ovládání pohledu pomocí myši. To je řešené třídou THREE.TrackballControls z knihovny Three.js.

Ve funkci refresh se pouze aktualizuje pozice myši, pomocí které se automaticky přepočítá pohled na scénu.

5.4.1.5 Helpers

Třídy ve složce Helpers slouží pro jednoduché seznamy, nebo funkce, které se používají zcela samostatně a nejsou přímo součástí jiných objektů.

5.4.1.6 Helpers/State.js

Třída State obsahuje pouze seznam stavů. Stavy jsou interně v programu reprezentovány čísly, ale v kódu se k nim přistupuje přes textové proměnné.

Seznam stavů je následující:

- Animace všech uzlů najednou dopředu
- Animace všech uzlů najednou dozadu
- Pauza
- Plynulá animace po jednom uzlu dopředu
- Plynulá animace po jednom uzlu dozadu
- Animace pouze jednoho uzlu dopředu
- Animace pouze jednoho uzlu dozadu

5.4.1.7 Helpers/TransformType.js

Třída TransformType obsahuje seznam všech typů vnoření, co jsou v programu použity. Funguje na stejném principu jako třída State. Jednotlivá vnoření jsou v programu rozlišena pomocí čísel, v kódu se k nim ale přistupuje přes textové řetězce.

5.4.1.8 Objects

Ve složce Objects jsou umístěny třídy, které slouží pro jednotlivé objekty, případně nadřazené třídy, které dané objekty seskupují. Nejsou zde funkce pro složitější rozhodovací mechanizmy, tyto objekty se dají přirovnat k cihlám, ze kterých se postaví základ programu.

5.4.1.9 Objects/BaseObject.js

Třída BaseObject je základní herní objekt, od kterého dědí všechny další. Obsahuje důležité funkce, které musí každý objekt mít implementované, a které jsou volány třídou GAME.

- **refresh()**

  Funkce, která se automaticky volá při obnovení scény. V této funkci by měl být pouze nezbytně nutný kód. Pokud by v této funkci byla složitější konstrukce, pak by to mohlo výrazně zpomalit běh celé aplikace.

- **setState()**

  SetState je funkce, která se volá při změně stavu hry. Konkrétně se jedná o změnu typu, nebo pozastavení animace. Jednotlivé třídy, které budou od tohoto objektu dědit, ji mohou přepsat, zpravidla to však nebývá nutné. V základu je napsaná tak, že obsahuje podmínky pro všechny dostupné stavy hry. Pokud pro daný stav existuje speciální vykreslovací funkce, pak ji nastaví jako aktuální, která se bude volat při obnovení scény. Pokud taková funkce neexistuje, zůstane nastavena původní funkce.

- **showNodeNumbers()**

  Pomocná funkce pro zobrazení čísel uzlů. Tato funkce se nejprve zavolá u nejvíc nadřazeného objektu, ze kterého následně „probublá“ až ke konkrétním uzlům. Z toho důvodu je umístěna zde. Každý objekt, který ji bude používat, ji musí přepsat podle toho, jak se má v něm daná funkce chovat.

- **hideNodeNumbers()**

  Jedná se o stejný případ jako u funkce pro zobrazení čísel uzlů, s tím rozdílem, že tato funkce čísla uzlů skryje.

5.4.1.10 Objects/Node.js

Node je třída reprezentující uzel grafů. Obsahuje parametry:

- počáteční a cílovou pozici
- unikátní ID, podle kterého se v konkrétní síti rozpozná
- odkaz na rodičovský objekt (síť)
- informaci o typu - statický (bude pořád na stejném místě vykreslený poloprůhledně), nebo dynamický (bude menší, zvýrazněný a bude se animovat)

Uzel je při vykreslení tvořen Three.js komponentou SphereGeometry. Kromě parametrů obsahuje i metody:

- nastavení cílové pozice
- detekce raycastu - vyvolá událost, když uživatel najede nad uzel kurzorem myši
- přepočítání rychlosti, jakou se bude v každém směru pohybovat - volá se při vytvoření sítí a jejich umístění tak, aby se nepřekrývaly.
- rozsvícení - změna barvy po najetí myší
- posun o jeden krok dopředu
- posun o jeden krok dozadu

5.4.1.11 Objects/Edge.js

Třída reprezentující hranu grafu.

Na rozdíl od uzlu je její struktura o něco jednodušší. Její parametry jsou:

- Odkaz na rodičovský objekt
- Unikátní ID
- Odkaz na uzel, ze kterého směřuje
- Odkaz na uzel, do kterého směřuje
- informace o typu - statická, nebo dynamická, význam typu je stejný jako u uzlů

V animaci se u statických sítí hrana vykreslí pomocí tvaru válce. Je to kvůli lepším grafickým vlastnostem, než pokud by na to byl použit objekt čáry.

5.4.1.12 Objects/Site.js

Základní třída pro reprezentaci sítě. V této třídě se pouze definují parametry a funkce, které budou pro všechny sítě společné. Díky tomu bude možné program později snadno rozšiřovat. Každá nová síť podědí všechny společné vlastnosti.

Parametry:

- Vzdálenost mezi uzly.
- Seznam objektů s uzly
- Seznam objektů s hranami

- ID aktivního uzlu
- ID uzlu, který se právě pohybuje
- Příznak, zda jsou všechny uzly co se mají animovat ve startovní pozici
- Příznak, zda jsou všechny uzly co se mají animovat v cílové pozici

Metody:

- Init - inicializace sítě, vytvoření objektů pro jednotlivé uzly a hrany
- Detekde raycastu - pokud uzel detekuje, že je nad ním kurzor, zavolá tuto metodu rodiče. Rodič (konkrétní síť) předá tuto událost svému rodiči (objekt Player), ten zjistí, které všechny uzly a hrany se mají rozsvítit a předá informaci zpět konkrétní síti. Ta zajistí rozsvícení všech patřičných uzlů/hran.
- Rosvícení a zhasínání konkrétních uzlů
- Refresh - funkce volaná při každém obnovení animace.
- Výpočet rychlosti animace všech uzlů v síti
- Posun všech uzlů najednou dopředu
- Posun všech uzlů najednou zpátky
- Posun jednoho aktivního uzlu dopředu
- Posun jednoho aktivního uzlu zpět
- Aktualizace umístění hran (volá se po každém posunu uzlů)

5.4.1.13 Objects/Cube.js

Reprezentace hyperkrychle. Třída si v konstruktoru podle zadané dimenze nechá vygenerovat strukturu uzlů, jakou má mít hyperkrychle. Žádné speciální metody tato třída nemá, stačí ty, které jsou poděděné od základní.

5.4.1.14 Objects/Mesh.js

Reprezentace 2D mřížky. Funguje podobně jako třída pro hyperkrychli, pouze si na začátku nechá vygenerovat strukturu pro mřížku. Opět žádné další metody navíc kromě poděděných nepotřebuje.

5.4.2 Modul PAR

Tento modul slouží pouze pro výpočty struktur jednotlivých sítí a namapování uzlů na sebe při vnoření. Je napsán tak, že vůbec nezasahuje do výsledných animací (slouží pouze jako generování dat). Díky tomu by mohl být napsán v jakémkoliv programovacím jazyce a se zbytkem programu by komunikoval pomocí předem definovaného rozhraní.

5.4.2.1 Tree.js

Třída Tree generuje strukturu úplného binárního stromu. Pro zjednodušení je použito číslování uzlů identické s preOrder průchodem. Nejdříve kořen, pak levý a nakonec pravý podstrom

Strom pak má velmi hezké vlastnosti pro generování. První listy v každém dalším řádku (úrovni) začínají číslem 2^n, kde n je hloubka řádku. Rodičovský uzel se spočítá pomocí vzorce:

$$idParent = \left\lfloor \frac{id}{2} \right\rfloor$$

Jednoduché spočítání čísla uzlu rodiče se perfektně hodí pro vytváření hran.

Ještě je potřeba upřesnit, jak se spočítá pozice každého uzlu. Souřadnice Z je vždy konstantní (strom se vykresluje ve 2D rovině). Souřadnice Y se spočítá tak, že se aktuální hloubka uzlu vynásobí konstantou určující jak daleko budou jednotlivé řádky uzlu daleko od sebe. Souřadnice X je o něco složitější, tam je nejdřív nutné podle aktuální hloubky zjistit odsazení zleva. Strom bude osově souměrný, jinak by všechny první uzly v každém řádku byly přímo nad sebou a strom by se s rostoucí hloubkou rozšiřoval pouze na jednu stranu. Také by to šlo, ale nevypadalo by to moc pěkně. Poté se každý uzel vynásobí konstantou určující mezery mezi uzly tolikrát, jaká je jeho vzdálenost od nejlevějšího v daném řádku. Pro názornost je zde ukázka kódu, která má toto počítání na starosti.

Listing 5.5: Tree.js

```
PAR.Tree.prototype.getNodePositionX = function (id) {
    var nodeDeep = this.getNodeDeep(id);
    var nodeOffset = this.getNodeOffset(id, nodeDeep);

    var offsetFirst = Math.pow(2, this.deep - nodeDeep - 1) - 1;
    var offsetNext = Math.pow(2, this.deep - nodeDeep);

    return offsetFirst + (nodeOffset * offsetNext * this.xOffset);
}
```

5.4.2.2 Cube.js

Pozice uzlů v hyperkrychli jsou v základu zadány napevno podle toho, v jaké části hyperkrychle se daný uzel nachází. U hyperkrychlí větších dimenzí dochází ke zkopírování původní hyperkrychle a jejímu posunutí vedle či pod původní. Není to tedy tak, že by každý uzel měl nastavenou přesnou pozici, napevno se mu nastaví základní pozice v krychli do dimenze 3. U větších dimenzí se pouze využije předefinovaná krychle rozměru 3 a ke všem souřadnicím se připočte offset, který ji posune na správné místo.

Generování hran funguje přesně podle definice. Číslo každého uzlu se převede do binární podoby a v cyklu se mu postupně po jednom negují jednotlivé bity. Binární číslo s nově znegovaným bitem se opět převede do desítkové soustavy. Každé takto nově vytvořené číslo je jeden sousední uzel. Aby se předešlo duplicitním hranám (hrana vedoucí z uzlu 0 do uzlu 1 a další vedoucí z 1 do 0), tak se generují pouze ty, které směřují do uzlu s větším číslem než je ten, ze kterého hrana vychází.

5.4.2.3 Mesh.js

Mřížka se generuje dvojitým cyklem, kde první cyklus běží tolikrát, jakou má mřížka šířku a druhý podle toho, jakou má výšku. Třetí cyklus (na případnou hloubku) v tomto zadání není potřeba, ale nebylo by složité algoritmus upravit a tuto funkčnost případně přidat.

Hrany se generují tak, že z každého uzlu vedou právě dvě, jedna v horizontálním a druhá ve vertikálním směru. Je to ošetřeno podmínkou, aby hrany vedly pouze do uzlů uvnitř sítě - pokud je uzel na konci, už z něj žádná hrana nevede.

5.4.2.4 SiteSetting.js

Třída SiteSetting sbírá informace o nastavených parametrech vnoření a komunikuje s uživatelským rozhraním. Hlavní účel je zpracování zadaných dat a vrácení informace o tom, zda jsou data dostačující a je možné z nich vygenerovat sítě a vnoření.

5.4.2.5 Transform.js

Třída Transform funguje jako takový rozcestník pro generování struktur vnoření. Objekt, který ma na starost generování sítí a animací volá pouze metody této třídy. Nejdříve jí samozřejmě předá informace o typu vnoření a parametry sítí.

Důležitá je metoda *calculate*, která podle toho, o jaké vnoření se jedná, vytvoří instanci třídy pro výpočet konkrétního vnoření, které vrátí jako výstup.

Všechna vnoření mají stejnou strukturu. Jedná se o JSON objekt obsahující seznam uzlů a hran a k nim jejich související uzly/hrany. Párování uzlů

a hran je obousměrné, objekt obsahuje jak seznam mapování zdrojových uzlů/hran na cílové, tak mapování cílových na zdrojové. Je to z důvodu rychlejší odezvy při ovládání animace. Výpočet vnoření trvá takto o něco málo déle, ale při používání programu už dochází pouze k dotazování na hodnoty v existujícím objektu a není potřeba nic znovu počítat.

Ve výstupním objektu jsou kromě informací o vnoření také parametry daného vnoření, které se zobrazí v programu.

5.4.2.6 TransformTreeToCube.js

Třída TransformTreeToCube generuje vnoření stromu do hyperkrychle. Mapování uzlů ze zdrojové sítě do cílové je zadané napevno díky tomu, že je zde pouze málo možných kombinací - generují se vnoření do dimenze maximálně 5. Dalším důvodem je trochu jiné číslování uzlů, než je ve skriptech.[2] Je to z důvodu jednoduššího generování struktury stromu pro animace a také, aby animace začala od kořene. Každá animace začíná od uzlu s nejmenším číslem a pokračuje k větším.

Obrácené mapování se vygeneruje v cyklu automaticky projitím originálního mapování - vždy se vezme cílový uzel, a k němu se přiřadí jako související uzel ten, který je k němu přiřazený jako startovní.

5.4.2.7 TransformCubeToMesh.js

Generování vnoření hyperkrychle do 2D mřížky funguje podobně, jako u vnoření výše. Díky tomu, že se vnoření generuje pouze pro několik málo dimenzí (maximální dimenze je 5), tak stačí čísla uzlů na sebe namapovat napevno.

5.4.2.8 TransformRectangleToSquare.js

Vnoření mřížky do mřížky je o něco složitější. Na rozdíl od vnoření popsaných výše, je různých kombinací parametrů mnoho a není tak možné uzly na sebe namapovat napevno.

Z důvodu, že toho vnoření má load 2, je pro výpočet jednodušší generovat mapování v obráceném pořadí - v základu se nejdříve pro všechny cílové uzly vygenerují jim odpovídající zdrojové uzly a následně v obyčejném cyklu se nastaví i obrácené mapování.

Při generování odpovídajících uzlů k cílovým se nejprve ke každému uzlu spočítají parametry udávající jeho vlastnosti. Tyto vlastnosti jsou:

- ID uzlu
- Číslo řádku mřížky, ve kterém se uzel nachází
- Číslo sloupce, ve kterém se uzel nachází

- Číslo relativního řádku (hada) - had je vysoký tak, jak je vysoká zdrojová síť
- Směr mapování relativního řádku - z levého horního rohu do pravého dolního, nebo z pravého dolního rohu do levého horního
- Příznak, zda je uzel v prvním řádku mřížky
- Příznak, zda je uzel v posledním řádku mřížky
- Relativní ID - pozice v aktuálním řádku hada
- Číslo uzlu zdrojové sítě, který je prvním uzlem v relativním řádku
- Číslo uzlu zdrojové sítě, který je posledním uzlem v relativním řádku
- Příznak, zda je uzel v ohybu hada, nebo v rovné části
- Směr ohybu hada - pouze pokud je uzel v ohybu
- Číslo prvního uzlu v ohybu - pouze pokud je uzel v ohybu
- Relativní pozice uzlu v ohybu - pouze pokud je uzel v ohybu

Následuje několik různých možností podle toho, zda je uzel umístěn v ohybu nebo v rovné části. Pokud je uzel v ohybu, pak výpočet provede speciální funkce, která podle typu ohybu a pozice daného uzlu v ohybu spočítá zdrojové uzly. Tento výpočet je v kódu zadán napůl napevno. Konkrétní mapování je jiné u všech typů ohybů a u různých výšek zdrojových sítí. K relativnímu číslu se pouze přičte číslo uzlu zdrojové sítě, které je v ohybu jako první.

Pokud je uzel v rovné části, pak se využije faktu, že se na jeden cílový mapují vždy dva uzly z původní sítě. Pomocí relativní pozice uzlu v aktuálním pruhu a pomocí id uzlu zdrojové sítě a dalších parametrů se spočítají čísla dvou uzlů, která se na konkrétní uzel namapují.

Přesněji to je vidět v následující ukázce kódu:

Listing 5.6: State.js

```
...

var nodesBeforeInWrap = this.mainSetting.wrapSize *
    this.mainSetting.wrapSize;

relativeRowId = setting.relativeId % this.source.y;

startId = setting.firstNodeInLine + ((setting.relativeId -
    relativeRowId) * 2 + relativeRowId) - nodesBeforeInWrap;
```

```
// pridani uzlu co se budou mapovat do pole, ktere vrati funkce na
    vystupu
nodes.push(startId);
nodes.push(startId + this.source.y);

...
```

5.4.2.9 TransformSquareToRectangle.js

Třída TransformSquareToRectangle má na starosti algoritmus pro vnoření čtvercové mřížky do obdélníkové. Algoritmus je podrobněji popsán v dřívějších kapitolách.

Implementovaný algoritmus v této práci používá mapování uzlů po dvou. Na první uzel cílové mřížky se namapují první 2 ze zdrojové, na druhý další dva atd.

Mapování uzlů se počítá podle čísel uzlů ze zdrojové sítě. V prvním kroku dojde ke spočítání relativního čísla uzlu v aktuálním sloupci. Algoritmus je pro všechny sloupce zdrojové mřížky stejný, nezáleží tedy na tom, o kolikátý sloupec přesně jde. Pokud je relativní ID liché, odečte se od něj jednička, jakýkoliv lichý uzel se mapuje stejně jako předchozí.

Následně se spočítá pozice ve sloupci v cílové síti, číslo relativního sloupce v daném bloku cílové sítě, číslo sloupce původní sítě a číslo sloupce cílové sítě, ve kterém bude daný uzel umístěn. Z těchto hodnot se spočítá číslo prvního uzlu v cílové síti ve sloupci, na který bude uzel namapován. K číslu prvního uzlu ve sloupci v cílové síti se přičte pozice ve sloupci v cílové síti a tím se získá číslo uzlu cílové sítě, na který se má uzel namapovat.

Výpočet souvisejících hran je stejný jako u ostatních algoritmů a je popsán výše.

5.4.3 Modul GUI

Modul GUI slouží pro oddělení souborů, které mají na starost komunikaci jádra aplikace s grafickým rozhraním.

5.4.3.1 GUI.js

Třída GUI je statická a obsahuje funkce, pro napojení ovládacích tlačítek na samotnou aplikaci. To znamená, že volá akce po kliknutí na některé z tlačítek a také mění jejich stavy - rozsvítí aktivní, umožní klikání na tlačítka v momentě, kdy jsou vygenerovány sítě a kdy je již možné je ovládat. Obsahuje rovněž napojení na formulář, ve kterém se zadávají parametry vnoření a na slider měnící rychlost animace.

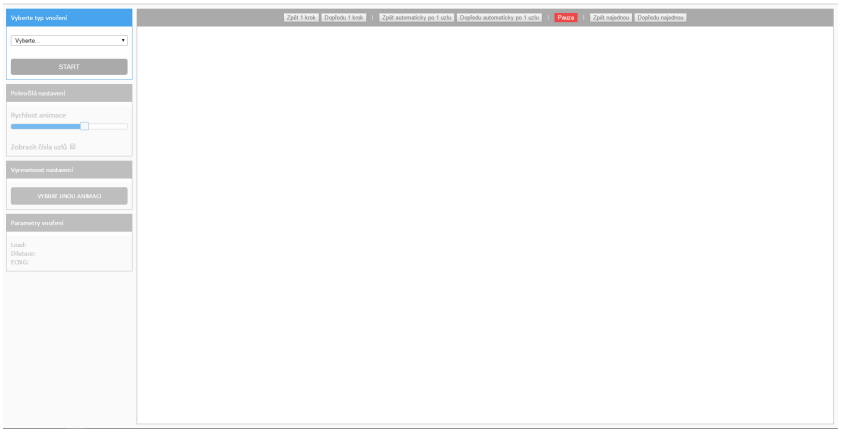
Kapitola 6

Ovládání programu

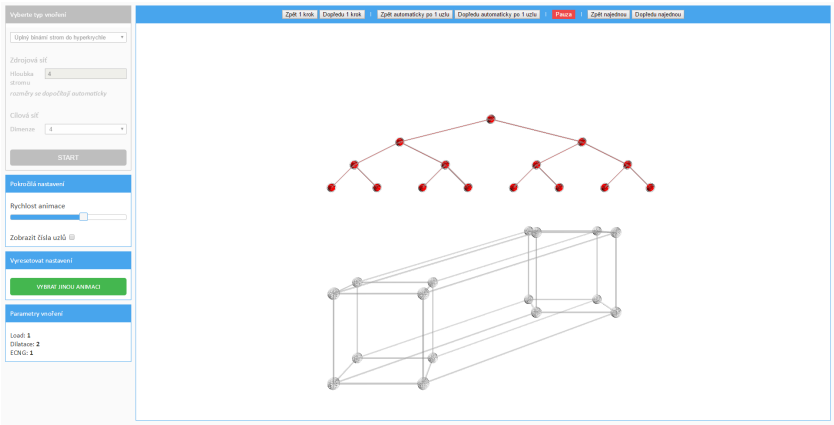
6.1 Nastavení vnoření

Na začátku jsou všechna pole deaktivovaná, kromě výběru sítí. Je to vidět na obrázku 6.1.

Jakmile uživatel vybere sítě, které se do sebe mají vnořovat, dostane se k nastavení rozměrů sítí. Některé rozměry jsou omezené tak, aby byla výsledná animace stále přehledná. Pokud by někdo nastavil, že chce vidět vnoření hyperkrychle o dimenzi 50, tak by jednak výsledná animace byla velmi náročná na výpočetní výkon, takže by běžela velmi pomalu a za druhé by obsahovala tolik uzlů a hran, že by to bylo nepřehledné. Uživatel by z takové animace neměl šanci pochopit fungování algoritmu. Uživatel tedy nastaví některé z rozměrů a další se automaticky dopočítají. Toto je jednak pro větší přehlednost a také pro to, aby výsledná animace fungovala. Pro všechny animace je použit vždy jeden algoritmus, který se použije u všech nastavených rozměrů sítí. Výběr



Obrázek 6.1: Okno programu ihned po spuštění



Obrázek 6.2: Okno programu po nastavení animace vnoření stromu do hyperkrychle dimenze 4

algoritmu tedy závisí pouze na konkrétním vnoření, nikoli na rozměrech sítí.

Jakmile jsou nastavené všechny potřebné rozměry, aktivuje se tlačítko pro vygenerování sítí.

6.2 Ovládání animace

Po kliknutí na tlačítko pro vygenerování sítí se v největším okně programu zobrazí vybrané sítě a nad nimi se aktivují tlačítka pro ovládání animace. Okno pro výběr animací a zadání rozměrů sítí se deaktivuje. Jediná možnost, jak v této chvíli parametry vnoření změnit je kliknutí na tlačítko pro reset animace a všechny parametry je nutné zadat znovu. Náhled okna po vygenerování sítě je na obrázku 6.2

Uživatel nemusí ihned po vygenerování sítí začít přehrávat animaci, může si nejdříve pečlivě prohlédnout strukturu obou sítí ze všech stran a ze všech možných přiblížení. Prohlížení sítí, tj. změna úhlu pohledu, přibližování a oddalování se provádí pomocí myši.

Nad oknem se sítěmi jsou tlačítka pro ovládání animace. Existuje několik způsobů, jak jde animace vnoření přehrát.

- Přehrání celé animace najednou dopředu a zpátky (všechny uzly se pohybují souběžně)
- Automatické přehrání celé animace po jednom uzlu. Nejdřív se přemístí první uzel, pak druhý atd. Vše probíhá automaticky, dokud se nepřesune celá síť na cílovou pozici.
- Přesun pouze jednoho uzlu. Pro přesun dalšího je nutné znovu kliknout na tlačítko. Toto je vhodné pro dobré pochopení vnořovacího algoritmu.

Uživatel si přehraje animaci pouze jednoho uzlu a nemusí pokaždé animaci ručně zastavovat.

- Pauza - animace se dá v kterémkoliv místě zastavit a případně změnit na jinou.
- Slider pro nastavení rychlosti. Pokud by byla rychlost pokaždé stejná, tak by při přehrávání po jednotlivých uzlech trvala moc dlouho a uživatele by mohla nudit, zato při posouvání všech uzlů najednou by byla až příliš rychlá a uživatel by nestihl pochopit, jak přesně dané vnoření funguje. Měnit rychlost se dá kdykoliv, klidně v průběhu samotné animace.
- Zatržítko pro zobrazení/skrytí čísel uzlů - Pokud by chtěl uživatel zobrazit čísla uzlů, aby si udělal lepší představu, které se na sebe mapují, tak je tu pro to speciální zatržítko.

Kapitola 7

Testování

Testování programu se bude skládat z několika různých částí. V první půjde o to otestovat, zda je program funkční v nejčastěji používaných prohlížečích. Program je určen pro studenty informatiky, takže starší verze Internet Exploreru se mezi ně nepočítají.

Akceptační testování slouží pro ověření, jestli program simuluje dané algoritmy správně a může tím pádem sloužit jako výuková pomůcka pro studenty.

U uživatelského testování se znalost problematiky od člověka, co to bude testovat nevyžaduje. Tam je na druhou stranu lepší, pokud člověk o tom moc neví, protože student, co bude program v praxi používat ze začátku také látce ještě nebude rozumět a ideálně pomocí tohoto programu by jí měl snáz pochopit.

7.1 Testování funkčnosti programu

Testování funkčnosti programu má za úkol ověřit, zda program lze spustit v různých prohlížečích, a že v každém z nich se chová stejně. Jde hlavně o to ověřit, jestli funguje nastavování parametrů sítí, přehrávání animací a jejich ovládání.

Jelikož program využívá moderních technologií, bude stačit, když bude fungovat v nejnovějších verzích nejčastěji používaných prohlížečů. Vzhledem k tomu, že se jedná o webovou aplikaci, tak je pro správnou funkčnost rozhodující použitý webový prohlížeč a ne nastavení uživatelova počítače, nebo operační systém. Seznam prohlížečů, ve kterých bude aplikace testována:

- Google Chrome
- Mozilla Firefox
- Internet Explorer 11
- Microsoft Edge

Tabulka 7.1: Výsledky testování funkčnosti programu

Příkaz	Chrome	Firefox	IE11	Edge
Zobrazení základního okna programu	OK	OK	OK	OK
Nastavení vnoření mřížky nx2 do mřížky 6x6	OK	OK	OK	OK
Přehrání celé animace najednou	OK	OK	OK	OK
Přehrání celé animace najednou zpátky	OK	OK	OK	OK
Přehrání celé animace po jednotlivých uzlech	OK	OK	OK	OK
Přehrání celé animace po jednotlivých uzlech zpět	OK	OK	OK	OK
Přehrání pouze 1 kroku animace	OK	OK	OK	OK
Přehrání pouze 1 kroku animace zpět	OK	OK	OK	OK
Přerušení animace v jejím průběhu	OK	OK	OK	OK
Zobrazení čísel uzlů	OK	OK	OK	OK
Skrytí čísel uzlů	OK	OK	OK	OK
Změna rychlosti animace	OK	OK	OK	OK
Zobrazení souvisejících uzlů z původní sítě	OK	OK	OK	OK
Zobrazení souvisejících uzlů z cílové sítě	OK	OK	OK	OK
Tlačítko pro reset animace	OK	OK	OK	OK
Vygenerování animace vnoření hperkrychle dimenze 3 do mřížky	OK	OK	OK	OK
Vygenerování animace stromu do hyperkrychle dimenze 3	OK	OK	OK	OK

7.1.1 Výsledky

Výsledky testů jsou uvedeny v tabulce 7.1.

7.2 Uživatelské testování

Uživatelské testování bude probíhat na lidech s různou znalostí problematiky a programování. Úmyslně jsou vybráni do testování i lidé, kteří předmět pa-

ralelní algoritmy nemají. Jedním z cílů této práce je, aby cílového uživatele práce s programem bavila. Cílová skupina jsou hlavně lidé, kteří předmět PAR mít budou, ale algoritmům ještě rozumět nebudou. Pokud tedy člověka, co o vnořovacích algoritmech nikdy neslyšel animace zaujme, nebo dokonce bude bavit, pak se bude jednat o úspěch. Rozhodně není cíl, aby animace a ovládání programu bylo tak složité, že člověka vyděsí a ten o dané problematice už nebude chtít nikdy nic slyšet.

Na začátku testování člověk dostane informaci o tom, k čemu program slouží - že se jedná o výukovou pomůcku pro zvládnutí látky jednoho předmětu. Úmyslně nedostane informace o tom, jak se má program ovládat, ani co konkrétní animace znamenají.

Uživatel nejdřív bude muset splnit různé úkoly, přičemž během toho nebude dostávat žádnou nápovědu, pokud požadovanou funkci v programu nenajde, znamená to, že rozhraní programu není navržené úplně optimálně a bude na zvážení, jestli by se daná část rozhraní neměla pro větší přehlednost upravit.

Úkoly, které uživatel dostane jsou:

- Nastavit animaci vnoření čtyřrozměrné hyperkrychle do mřížky
- Spustit celou animaci najednou
- Spustit zpětnou animaci pouze pro 1 uzel
- Nechat kompletní animaci zastavenou přesně v polovině
- Nastavit novou animaci vnoření stromu do 3D hyperkrychle
- Zjistit bez přehrání animace, který uzel ze stromu se namapuje na cílový uzel č.3
- Chvilku si s programem hrát (zkoušet měnit pohled, nastavit si libovolnou jinou animaci atd.)

Po otestování uživatel dostane tyto otázky:

- Popište zkušenost s vizuální přehledností programu
- Popište zkušenost s ovládáním 3D pohledu

7.2.1 Průběh testování

V této části jsou popsány průběhy testování u jednotlivých osob. Nejprve je u každého popsáno jak test probíhal a na konci jsou vypsány připomínky, nebo nápady, které daný člověk měl.

7.2.1.1 Tester 1

Tester 1 (Láďa) je projektový manažer a odborník na SEO. Co se týče programovacích znalostí, tak o sobě tvrdí, že programovat neumí, i když ve škole měl jeden předmět na Javu a na objektové modelování.

Ze začátku měl trochu problém, jak program ovládat. Nastavit parametry animace zvládl bez větších obtíží. Horší to bylo s ovládáním samotné animace, ale po chvíli na to přišel také a první část úkolů splnil.

Při zadávání druhé animace si všiml, že program obsahuje i pokročilá nastavení rychlosti a zobrazení čísel uzlů. Když pak dostal za úkol zjistit, který uzel se mapuje na ten s číslem 3, nebyl problém čísla zobrazit a pomocí najetí myši nad uzel správnou odpověď najít.

Přišlo mu ale trochu matoucí, že se na sebe mapují uzly s jinými čísly. Pak mu také nebylo úplně jasné, která ze sítí je zdrojová a cílová.

Celkově se s programem seznámil a naučil se ho ovládat, i když podle jeho slov vůbec netušil co a proč nastavuje a k čemu dané algoritmy slouží.

7.2.1.2 Tester 2

Tester 2 (Ondřej) je html kodér a MVC.NET programátor. Aktuálně ale dělá na své dizertační práci ve švýcarském Cernu. Na Vánoce přijel na několik dní domů a rovnou mi pomohl s testováním diplomky.

Podobně jako Láďa neměl Ondřej problém s nastavením parametrů sítí a jejich vygenerováním.

V první části ale místo přehrání po 1 kroku spustil animaci přehrávání po jednotlivých uzlech.

Ve druhé části s vygenerováním nové sítě problém nebyl. Horší ale bylo najít, kde se zobrazují čísla uzlů. Během toho zkoušel na uzel všemožně klikat levým i pravým tlačítkem, zkoušel různě měnit pohledy, jestli číslo není nikde schované. Podívat se do levého menu ho napadlo až za relativně delší dobu, když jsem mu řekl, že tam ta funkčnost někde určitě je.

Když to našel, tak mi dal připomínku, že by bylo dobré toto dát třeba jako akci po kliknutí na uzel, nebo po kliknutí pravým tlačítkem někam do části, kde se zobrazovaly sítě.

7.2.1.3 Tester 3

Tester 3 (Michal) pracuje jako grafik. Mimo jiné také dělá návrhy webů.

Formulář pro generování sítí našel poměrně rychle. Tlačítko pro přehrání celé animace mu dělalo problém najít, takto zřejmě není vhodně umístěné a otextované. U zastavení animace očekával, že přerušení dojde po kliknutí na stejné tlačítko, které animaci začalo, nečekal, že bude jinde tlačítko pro pauzu.

V další části našel související uzly pomocí toho, když na nějaký najede myší, ale s čísly uzlů byl problém. Zatržítko pro zobrazení čísel uzlů našel až po chvíli hledání.

Doporučil mi pokročilá nastavení umístit do okna s animací, lištu s tlačítky pro přehrávání dát dolů a určitě lépe popsat tlačítka.

Když dostal za úkol vyzkoušet cokoliv dalšího, tak zkusil nastavit vnoření mřížky do mřížky a cílovou síť nastavil na rozměry 200x750.

Jedna z celkových připomínek od něj byla ta, že formulář na začátku nastavení sítí je zbytečně schovaný, mohl by klidně být víc uprostřed, aby snáz upoutal pozornost. Dále by se panel s informacemi o vnoření mohl udělat méně výraznější, aby bylo jasné, že se jedná pouze o statické informace, a že není součástí ovládání. Další připomínka byla, že by se mohly uzly rozsvítit o trochu dříve, než na ně najedu myší, pokud je síť zobrazena daleko, je poměrně obtížné uzel přesně kurzorem zamířit.

7.2.1.4 Tester 4

Tester 4 (Michal) je můj bývalý spolužák FITu z bakalářského studia. Nyní pracuje jako programátor MVC .NET. Před začátkem testování mi říkal, že se na testování tohoto programu těšil celou pracovní dobu, jak byl na to zvědavý.

Michal neměl problém najít, kde se nastavují parametry vnoření a vygenerování sítí bylo bez problémů. Když měl spustit celou animaci, tak tlačítko hledal v levém sloupci níže a omylem klikl na tlačítko pro vygenerování nových sítí. Při druhém pokusu již tlačítko pro celkovou animaci našel. Přijde mu ale matoucí umístění tlačítka pro generování sítí, čekal by tam spíše tlačítko Play.

Vygenerování dalšího vnoření bylo bez problémů, díky tomu, že v minulé části hledal tlačítko pro spuštění animace v levém sloupci, tak již věděl, že se tam nachází checkbox pro zobrazení čísel uzlů a tím pádem už tuto funkčnost nemusel hledat, když byla zapotřebí.

Když si měl v dalším kroku jen tak hrát a zkoušet libovolná vnoření, tak zkusil ještě vnoření mřížky do mřížky. Po chvíli se zeptal: „Co je na tom vlastně tak těžkého?“. Po krátkém vysvětlení o co se v animacích jedná pochopil, že tam jde o to, co nejlíp namapovat jednu síť na druhou. Takže v tomto případě program splnil svůj účel.

Na závěr mi doporučil dát tlačítko pro reset animace jinam. Lépe otextovat tlačítka pro ovládání animace a ideálně je nějak zvýraznit - lépe je nastylovat a třeba i přidat ikonky.

7.2.1.5 Tester 5

Tester 5 (Martin) pracuje jako programátor MVC.NET. Jako jeden z mála má zkušenosti s paralelními počítači, protože si na jednom dělá výpočet pro svou diplomovou práci.

Jako jediný našel ihned, kde se nastavují parametry vnoření a kde se spouští celá animace. Animaci jednoho uzlu zpět zvládl najít také rychle. Další úkol - nechat animaci zastavenou v polovině pochopil trochu jinak, za-

čal animovat uzly po jednom a ve výsledku byla první půlka uzlů zpátky na startovních pozicích a druhá ještě v cílových.

Problém nastal při hledání, kde se zobrazují čísla uzlů, i když měl chvílemi několikrát kurzor nad zatržítkem, které tuto funkčnost aktivuje, tak na to klikl až po nějakém čase.

Číslování uzlů pochopil trochu jinak, než jak bylo zamýšlené. Na začátku jsou zobrazena čísla u všech uzlů - jak u statických šedých sítí, tak u červené aktivní. Když je síť na začátku, tak jsou vedle každého uzlu zobrazeny 2 číslice. Jedna červená (číslo uzlu, co se bude animovat) a jedna šedá (číslo uzlu statické sítě). Martin to pochopil tak, že červená je číslo uzlu původní sítě a šedá vedle je číslo uzlu, na který se ten konkrétní namapuje.

Když dostal za úkol libovolně si s programem hrát, tak zkusil nastavit vnoření mřížky do mřížky. Rozměry cílové mřížky zadal 20x20. Program bohužel na takové rozměry není optimalizovaný a v prohlížeči došla paměť. Při zmenšení rozměrů na 10x20 již animace fungovala.

Při velké obdélníkové mřížce se přišlo na to, že jsou uzly na konci špatně osvícené. Je to dáno umístěním směrového světla, které je moc blízko středu obrazu.

Na konci mi doporučil trochu optimalizovat objekty - koule mají zbytečně mnoho bodů a při velkém množství zabírají zbytečně moc paměti a program běží pomalu. Potom by se animace mohly lépe umisťovat do čtvercové sítě - podle zadaných rozměrů by se zvolil směr vnořování (vodorovně nebo svisle) při kterém by se do čtvercové sítě vešla větší obdélníková.

7.2.2 Shrnutí

Nastavení parametrů potřebných vnoření bylo většinou bez problémů. Chvilku každému trvalo, než se na začátku zorientoval a našel, kde má začít. Po nastavení animace a vygenerování sítí byl ale překvapivě problém najít, kde se spouští celá animace najednou. Jednak za to může nevýrazný, možná trochu nešikovně umístění pruh s tlačítky pro ovládání animace a pak také ne úplně dobrá textace tlačítek. Úkol zněl spustit celou animaci najednou a člověk si nebyl jistý, jestli tlačítko s textem „Dopředu najednou“ je to správné.

Také hodně lidem nebyl jasný rozdíl mezi tlačítky pro krokování a pro přehrání celé animace po jednom uzlu. Opět pomůže lépe tlačítka natextovat.

Najít tlačítko pro vyresetování nastavení a vygenerování nového vnoření šlo velmi rychle. Tlačítko je velmi výrazné. Bohužel na něj občas někdo klikl omylem, když očekával, že bude sloužit spíše ke spuštění animace.

Úkol najít uzel, který se bude mapovat na cílový uzel č.3 byl podle očekávání nejzajímavější. Kdo si předtím všiml, že lze zobrazit čísla uzlů, ten neměl problém. Kdo si toho ale nevšiml, tomu trvalo relativně dlouho, než tuto funkčnost našel. Také hodně lidí pochopilo zadání trochu jinak, to je ale spíše chyba zadání než programu. Jakmile byla čísla uzlů vidět a člověk, co

program testoval, najel myší nad uzel, tak okamžitě pochopil, že uzly, co se rozsvítily, spolu souvisí.

7.2.3 Připomínky, návrhy změn

V této části je seznam všech nalezených chyb, nápadů na možná vylepšení a připomínek, které vzešly z uživatelského testování. U každé připomínky bude rozhodnuto, jestli se bude do programu doplňovat.

Zobrazovat čísla uzlů po kliknutí na konkrétní uzel ve scéně
: Najít, kde se zobrazují čísla uzlů, dělalo většině lidí problém, nicméně spíš než toto by bylo jednodušší na implementaci nějak zvýraznit panel s pokročilými nastaveními.

  Přidání této funkčnosti by bylo relativně náročné a na samotnou aplikaci to nemá tak velký vliv.

  Výsledek: *Tato funkčnost se nyní přidávat nebude.*

Umístit jinam tlačítko pro reset animace
: Toto dělalo problém většině lidí a umístění tlačítka jinam, případně jeho méně výrazné nastylování nezabere moc času a na výsledné používání programu bude mít velký vliv.

  Výsledek: *Tlačítko pro reset animace bylo změněno z výrazného tlačítka na obyčejný odkaz a posunuto v levém sloupci dolů pod box s parametry vnoření.*

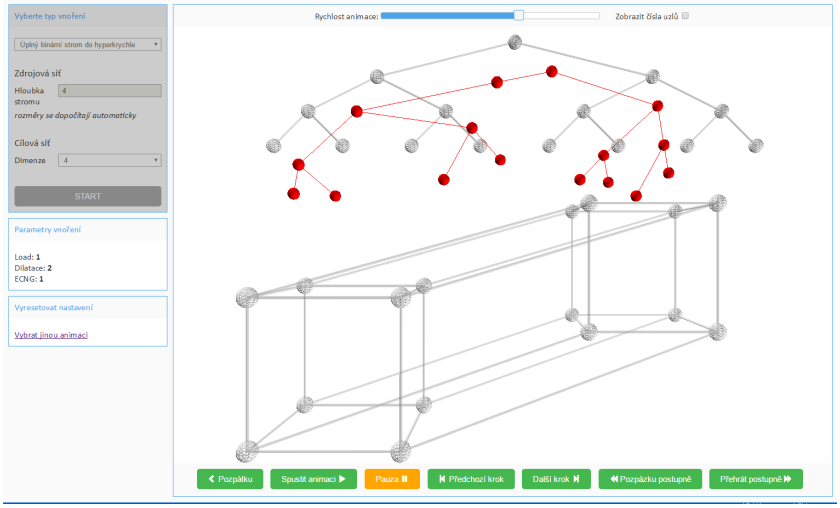
Lépe natextovat tlačítka pro ovládání animace
: Toto je nejdůležitější část celého programu. Každý, kdo program testoval, měl správně ihned najít, kde se spouští celá animace a také, co přesně dělají další tlačítka.

  Výsledek: *Texty na tlačítkách byly trochu zjednodušeny, aby více připomínaly „přehrávač“.*

Lépe umístit nebo ostylovat tlačítka pro ovládání animace
: Ze stejného důvodu, jako je popsáno výše bude tato připomínka zapracována.

  Výsledek: *Celá lišta s ovládacími tlačítky byla umístěna dolů pod scénu. Tlačítka jsou výrazněji nastylovaná a jsou celkově větší. K tlačítkům byly pro větší názornost přidány ikony. Upravená verze programu je vidět na obrázku 7.1.*

Okno s pokročilým nastavením animace umístit do scény
: Toto byla připomínka od jednoho člověka. Není to tak závažný problém kvůli kterému by někdo aplikaci nepoužíval. Navíc hrozí jiné problémy, jednak okno scény je ovládané jinou částí programu než nastavení a za



Obrázek 7.1: Aplikace po zapracování připomínek z uživatelského testování

druhé by byly možné kolize mezi oknem a samotnými modely - kvůli oknu by nemusela být vidět celá animace.

Výsledek: *Pokročilé nastavení bylo umístěno do zvláštního pruhu nad scénu.*

Formulář pro generování sítí na začátku umístit víc doprostřed

Formulář na začátku našel každý, umístění doprostřed scény by uživateli ovládání zjednodušilo, ale vyžadovalo by to poměrně dost úprav v kódu.

Výsledek: *Byly lépe odlišeny bloky, které jsou a které nejsou aktivní. Umístění zůstalo stejné.*

Panel s informacemi o vnoření udělat méně výrazný

Důvod této připomínky byl, aby se oddělily části, které souvisí s ovládáním animace a ty, které jsou pouze statické. Tento panel tam je jenom pro informace, nicméně je dobré, když si ho uživatel alespoň všimne. Navíc je dobré, když celá aplikace bude mít konzistentní vzhled.

Výsledek: *Toto nebude implementováno*

Aktivovat zvýraznění uzlů o trochu dříve, než se na ně najede myší

Aktuálně se uzel rozsvítí teprve v momentě, kdy je nad ním kurzor. Rozsvícení uzlu v momentě, kdy je kurzor blízko je složité na implementaci. Tuto funkčnost v základu zpracovává funkce z knihovny Three.js. Aby to fungovalo, tak by se musela výchozí funkčnost nějakým způsobem „obejít“. Šlo by to například tím, že by každý uzel byl obalený jiným, který by byl větší a nebyl by vidět. Není to ale čisté řešení, navíc by to mohlo způsobovat problémy při větším počtu uzlů ve scéně.

Výsledek: *Toto nebude implementováno*

Opravit pozici světla tak, aby i větší mřížka byla dobře osvícená
Toto je opravdu drobnost, asi nehrozí, že by kvůli této chybce někdo přestal program používat, ale oprava také není složitá. Jedná se o směrová světla, která mají intenzitu stejnou bez ohledu na vzdálenost. Pokud dojde k jejich posunutí dále na výslednou funkčnost to mít vliv nebude a síť bude vypadat dobře i při větších rozměrech.

Výsledek: *Vzdálenost světel od středu byla zdvojnásobena*

Optimalizovat program pro větší rozměry mřížek
V aktuální verzi program zvládne bez problému zpracovat mřížky o rozměrech do 15. Při větších rozměrech je nutný větší výpočetní výkon počítače, na kterém běží a animace se mohou zobrazovat pomaleji. Je to na úkor kvality zobrazení, která je aktuálně vysoká.

Účel tohoto programu je, aby studenti pochopili princip, jakým vybrané vnořovací algoritmy fungují. Na to stačí s přehledem rozměry, které program zvládne zobrazit. Optimalizace pro větší sítě by byla dobrá, ale aktuálně to není věc, co by bránila používání.

Výsledek: *Toto aktuálně nebude implementováno*

Upravit směr vnořování mřížky, aby se do animace vešla vždy větší zdrojová síť
Toto je dobrá připomínka pro možná budoucí rozšíření. Možných optimalizací je mnoho a jsou různě složité. V animacích je vždy použit pouze jeden algoritmus. Je to částečně i z důvodu jednoduchosti. Pokud člověk, co bude program používat nikdy o vnořovacích algoritmech neslyšel, snáze pochopí jeden konkrétní algoritmus, pokud bude pokaždé stejně, než kdy by se různě modifikoval.

Výsledek: *Toto aktuálně nebude implementováno*

7.3 Akceptační testy

Akceptační testy slouží pro ověření, zda animace vnoření v programu odpovídají tomu, jak by správně měly vypadat.

Akceptační testování prováděl doc. Ing. Ivan Šimeček, Ph.D. Je to expert na danou problematiku. Testování probíhalo na již upravené verzi programu po uživatelském testování.

S ovládáním aplikace problém nebyl. Formulář pro nastavení parametrů vnoření a tlačítko pro spuštění animace se podařilo najít okamžitě. Jediná věc, kterou bylo obtížnější najít, bylo tlačítko pro reset animace. To bylo po uživatelském testování posunuto dolů a bylo mu odebráno zvýraznění.

Animace algoritmů vnoření fungují správně. Jediná připomínka byla, že by se v okně s parametry vnoření mohly po najetí myší nad daný parametr

v animaci zvýraznit uzly, nebo hrany, kterých se daná hodnota týká. Jako příklad může sloužit parametr load. Pokud je load=2, pak by se po najetí myší rozsvítily všechny uzly, které se mapují po dvou.

Implementovat tuto funkčnost by ale bylo poměrně komplikované a bylo by to nad rámec původního zadání. Nicméně je to dobré jako jedna z možných funkčností, které by se do programu daly přidat, pokud by se program někdy v budoucnu rozšiřoval.

Kapitola 8

Zhodnocení a splnění cílů

Cílem této práce bylo vytvořit program, který má co nejpřehledněji simulovat vybrané vnořovací algoritmy.

8.1 Simulace vybraných vnořovacích algoritmů

Základní cíl této práce je simulace algoritmů pro vnoření mřížky do mřížky, hyperkrychle do mřížky a stromu do hyperkrychle. Všechny tyto animace program obsahuje. Animace jsou ve 3D, takže uživatel názorně vidí, jak přesně algoritmy fungují.

Správnost daných algoritmů byla ověřena při akceptačním testování.

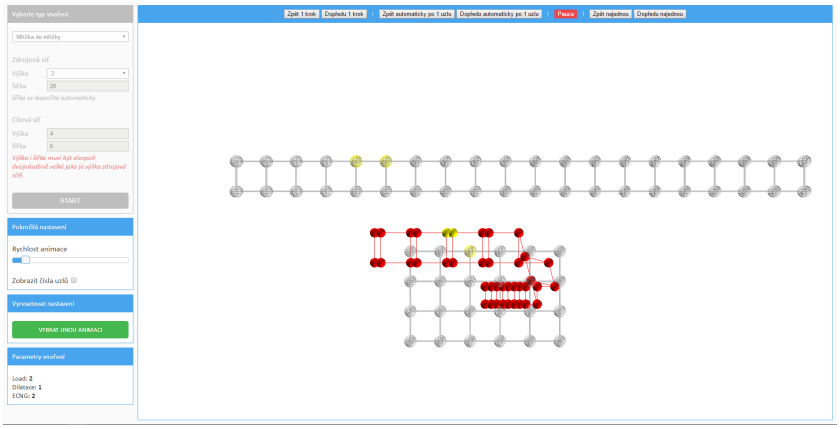
8.2 Názornost a grafické zpracování

Jedním z cílů byl důraz na grafické zpracování a uživatelskou přívětivost programu. Animace jdou ovládat několika možnými způsoby. Uživatel má možnost animovat po krocích, najednou po jednotlivých uzlech, nebo spustit animaci celé sítě najednou. Je samozřejmě možné kdykoliv animaci zastavit a detailně si vše prohlédnout. Zastavená animace mřížky do mřížky je na obrázku 8.1 Na obrázku je také vidět, jak se uzly rozsvítí po najetí myší. Žlutě označené uzly jsou ty, které spolu souvisí. Jsou to ve zdrojové síti ty, které se namapují na žlutě zvýrazněný uzel v cílové síti.

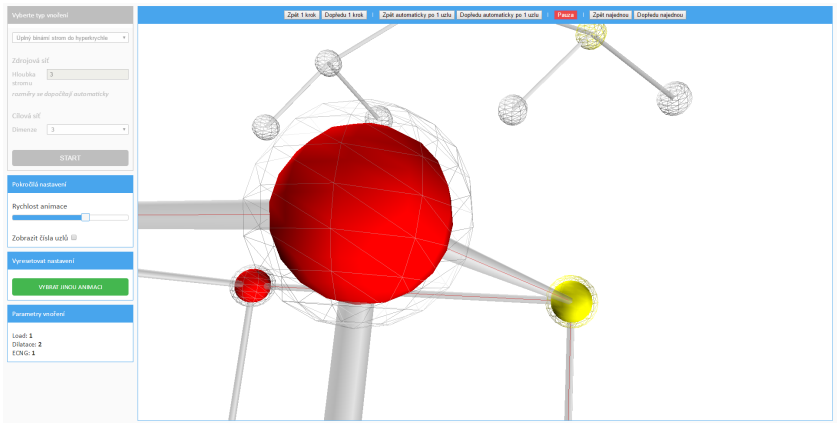
Uzly jsou nastaveny tak, aby vypadaly co nejrealističtěji i při velkém přiblížení. Na obrázku 8.2 je vidět přiblížení na jeden konkrétní uzel po dokončení animace.

8.3 Možné další pokračování práce

Možných rozšíření tohoto programu je skutečně mnoho, pokud by se všechna implementovala, tak by to dalo rozsahem mnohem víc práce, než je pro diplo-



Obrázek 8.1: Animace mřížky do mřížky zastavená v průběhu



Obrázek 8.2: Detail uzlu na konci animace vnoření stromu do hyperkrychle

movou práci nutné.

Některá možná rozšíření vzešla z uživatelského testování. Jedno z takových je například optimalizace pro větší sítě. Uživatel by si mohl při nastavování animace vybrat, v jaké kvalitě chce výslednou animaci mít. Podle toho by se upravilo rozlišení objektů, materiál a osvícení. Pokud by se vypnuly odlesky a stíny, pak by se výrazně ušetřil výpočetní výkon, ale celá síť by pak nevypadala tolik realisticky.

Další je optimalizace existujících vnoření, aby našla co nejefektivnější řešení. Toto by vyžadovalo úpravu modulu pro výpočet vnoření, aby pokaždé nepoužíval stejný algoritmus. Připravených algoritmů pro jedno vnoření by bylo více, a konkrétní použitý (nebo jejich kombinace) by se zvolil na základě nastavených rozměrů sítí.

Asi nejužitečnějším rozšířením by bylo přidání dalších vnořovacích algoritmů. Program je vytvářen modulově, takže by stačilo připsat moduly pro

další sítě a vnoření a jejich napojení by potom již nebylo složité. Program by také mohl fungovat pouze jako simulátor sítí bez funkce vnořování.

Jako další, pokud by byl program mezi studenty hodně oblíbený, by byla možnost přidání jiných než vnořovacích algoritmů. Mezi takové patří například různé řadící algoritmy, či permutace.

Závěr

Touto diplomovou prací se podařilo vyvinout aplikaci, která umí přehledně zobrazovat animace několika vnořovacích algoritmů. Během vývoje byl kladen důraz na přehlednost a pokud možno co nejlepší ovládání animací. Dále bylo myšleno i na případnou rozšiřitelnost programu v budoucnu.

Po dokončení základního vývoje aplikace bylo provedeno uživatelské testování, které mělo ověřit, zda je program skutečně použitelný pro cílovou skupinu. Vyplynulo z něj mnoho připomínek. Některé z nich byly důležité a byly do aplikace přidány. Jiné, méně důležité implementovány nebyly, ale byly ponechány jako možná další rozšíření.

Doufám, že tento program pomůže pomocí názorných animací studentům snadno pochopit danou problematiku.

Literatura

[1] Inside HPC: How HPC is Helping Solve Climate and Weather Forecasting Challenges. 2016-01-08, [cit. 2016-12-28]. Dostupné z: http://insidehpc.com/2016/01/how-hpc-is-helping-solve-climate-and-weather-forecasting-challenges/

[2] Prof. Ing. Pavel Tvrdík, Csc.: *Parallel algorithms and computing*. České vysoké učení technické v Praze, 2009.

[3] Wikipedia: OpenGL. 2016-12-06, [cit. 2016-12-28]. Dostupné z: https://cs.wikipedia.org/wiki/OpenGL

[4] Jeff Ward: What is a Game Engine? 2016-12-06, [cit. 2016-12-28]. Dostupné z: http://www.gamecareerguide.com/features/529/what_is_a_game_.php

[5] The Art Of Web: CSS: Animation Using CSS Transforms. 2016-04-09, [cit. 2016-12-28]. Dostupné z: http://www.the-art-of-web.com/css/css-animation/

[6] w3schools: CSS3 Transitions. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://www.w3schools.com/css/css3_transitions.asp

[7] w3schools: CSS3 transition-timing-function Property. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://www.w3schools.com/cssref/css3_pr_transition-timing-function.asp

[8] The Art Of Web: CSS: 3D Transforms and Animations. 2016-04-09, [cit. 2016-12-28]. Dostupné z: http://www.the-art-of-web.com/css/3d-transforms/

[9] Martin Michálek: SVG: vektorový formát, který na webu chyběl. 2016-03-08, [cit. 2016-12-28]. Dostupné z: http://www.vzhurudolu.cz/prirucka/svg

[10] Pavel Tišnovský: Grafický formát SVG a animace. 2007-09-13, [cit. 2016-12-28]. Dostupné z: https://www.root.cz/clanky/graficky-format-svg-a-animace/

[11] Ondřej Žára: Vytváříme Hello World pro WebGL. 2013-05-15, [cit. 2016-12-28]. Dostupné z: https://www.zdrojak.cz/clanky/vytvarime-hello-world-pro-webgl/

[12] Wikipedia: List of WebGL frameworks. 2016-12-15, [cit. 2016-12-28]. Dostupné z: https://en.wikipedia.org/wiki/List_of_WebGL_frameworks

[13] Jos Dirksen: *Learning Three.js - the JavaScript 3D Library for WebGL*. Packt Publishing, 2015.

[14] W3C: W3C HTML. 2016-12-28, [cit. 2016-12-28]. Dostupné z: https://www.w3.org/html/

[15] W3C: Cascading Style Sheets. 2016-12-28, [cit. 2016-12-28]. Dostupné z: https://www.w3.org/Style/CSS/

[16] w3schools: JavaScript Tutorial. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://www.w3schools.com/js/

[17] Hampton Catlin, Natalie Weizenbaum, Chris Eppstein, and numerous contributors: Sass Basics. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://sass-lang.com/guide

[18] w3schools: jQuery Tutorial. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://www.w3schools.com/jquery/

[19] Microsoft: Web Optimization. 2014-06-03, [cit. 2016-12-28]. Dostupné z: https://aspnetoptimization.codeplex.com/

[20] Bootstrap: Bootstrap. 2016-12-28, [cit. 2016-12-28]. Dostupné z: http://getbootstrap.com/components/

Příloha **A**

Seznam použitých zkratek

OpenGL Open Graphics Library

URL Uniform Resource Locator

HTML HyperText Markup Language

CSS Cascading Style Sheets

SVG Scalable Vector Graphics

XML eXtensible Markup Language

WebGL Web Graphics Library

SASS Syntactically Awesome Style Sheets

Příloha **B**

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `build` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF