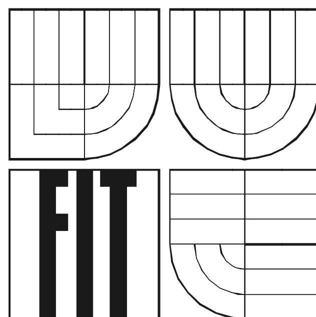
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Sdílení grafické virtuální scény po síti využité pro simulaci vysoce náročného fyzikálního děje - urychlovače částic

Diplomová práce

2006 Milan Pindryč

Sdílení grafické virtuální scény po síti využité pro simulaci vysoce náročného fyzikálního děje urychlovače částic

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně, dne 24. května 2006.

© Milan Pindryč, 2006.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů..

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pečivy
Další informace mi poskytl RNDr. Vojtěch Ullmann.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Milan Pindryč
15. května 2006

Poděkování :

Moje poděkování za odborné a organizační vedení a podporu při řešení problému patří především vedoucímu mého projektu **Ing. Janu Pečivovi.**

Abstrakt :

Tento projekt implementovaný v jazyce C++ s využitím nadstavbové knihovny nad openGL Open Inventor se zabývá distribuovaným řešením trojrozměrné simulace urychlování částic v kruhovém urychlovači, přesněji cyklotronu. Hlavním cílem práce bylo vytvořit vhodnou testovací aplikaci, která by prověřila funkčnost a použitelnost transakčního přístupu k řešení sdílených grafických scén a pomohla najít možnosti jeho vylepšení. Dalším z cílů bylo vytvořit fyzikálně věruhodný model s ohledem na náročnost výpočtů tak, aby byl schopen fungovat v reálném čase. Velký důraz je kladen také na názornost urychlování a především na přesnost a rychlost výpočtu trajektorie částic. Projekt naráží na obecné problémy distribuovaných výpočtů, především na zpoždění sítě a na velké výkonové nároky na komunikaci. Je možné měnit všechny parametry urychlovače za průběhu urychlení, což názorně demonstruje účinky a důsledky jednotlivých sil působících na částici v průběhu urychlení. Tato aplikace je určena jak pro demonstraci účinků elektromagnetických sil tak i jako ukázka možností využití sdílených grafických scén.

Klíčová slova :

OpenGl, Open Inventor, distribuovaný systém, transakční přístup, distribuční skupina, atomický multicast, aktivní transakce, datový model, cyklotron, trajektorie částic, elektromagnetické síly

Abstract

This project implemented in the C++ language with the use of library over OpenGL Open Inventor deals with 3D simulations of particle accelerators in the circular accelerator, precisely in the cyclotron. The main goal was to create testing application, which can verify usability of transaction approach for collaborative virtual environments and also can help to find improvements for this approach. Other goal was to create authentic model of cyclotron considering the difficulty of the calculations, which could work in real time. Great emphasis is also laid on the demonstration of the acceleration and mostly the calculation of particle trajectories. There are solved also some problems typical for distributed systems as network latency or performance demands on communication. It is possible to change all parameters of the accelerator during acceleration which illustrates the effects and results of each of the forces affecting the particles in the course of acceleration. This application is meant for demonstrating the affect of electromagnetic forces and for illustration of distributed graphical scenes.

Keywords

OpenGL, Open Inventor, distributed system, transaction approach, distribution group, atomic multicast, active transaction, data model, cyclotron, particle trajectories, electromagnetic forces.

Obsah

Obsah 7
Seznam obrázků 9
Seznam grafů......9
1 Úvod 10
1.1 Prostředky využité pro implementaci 10
1.2 Princip distribuovaného výpočtu 10
1.3 Fyzikální a matematická podstata urychlování částic 10
1.4 Tvorba 3D scény cyklotronu 11
1.5 Ovládání urychlovače 11
1.6 Měření a testování......11
1.7 Co bude v závěru 12
2 Prostředky využité pro implementaci 12
2.1 Volba prostředí a jazyka......12
2.2 Open Inventor 12
2.3 SoWin 13
2.4 SIM Voleon 13
2.5 Sdílené grafické scény 14
3 Princip distribuovaného výpočtu 14
3.1 Možné přístupy k řešení CVE 14
3.2 Transakční přístup 15
3.2.1 Transakční návrh 15
3.2.2 Aktivní transakce......15
3.2.3 Stavy transakcí 16
3.2.4 Kontrola souběžnosti 16
3.3 Jak fugují tyto knihovny v praxi......17
3.4 Doplnění funkčnosti knihoven......17
4 Fyzika urychlování částic 18
4.1 Proč používat urychlovače částic......18
4.2 Součásti urychlovačů částic......18
4.2.1 Terčík......18
4.2.2 Iontový zdroj částic 19
4.3 Rozdělení urychlovačů a jejich perspektivy......19
4.3.1 Lineární urychlovače 19
4.3.2 Kruhové urychlovače......19

4.3.3 Perspektivy urychlovačů 23
5 Scéna urychlovače 24
5.1 Úvodem 24
5.2 Skladba scény urychlovače 24
5.2.1 Jednoduché objekty 24
5.2.2 Objekty vytvářené z trojúhelníků 25
5.2.3 Částice 26
5.2.4 Trajektorie částice 27
5.2.5 Částice jako objemová data 29
5.2.6 Celkový pohled na scénu 30
5.3 Prostředky pro měření časů 31
5.4 Od samostatné aplikace k CVE 32
5.5 Funkčnost scény urychlovače 34
5.5.1 Callback funkce 34
5.5.2 Funkce pro výpočet pozic a rychlostí elektronů 35
5.5.3 Vytvoření transakce 36
6 Ovládání a nastavení aplikace 37
6.1 Vytvoření menu 37
6.2 Callback funkce 38
6.3 Sdílení hodnot v menu 39
6.4 Ovládání simulace 41
6.5 Parametry příkazové řádky 42
7 Měření a testování 42
7.1 Na čem se testovalo 42
7.2 Množství přenášených dat 43
7.3 Výpočetní čas pro vybavení sítě 45
7.4 Časová náročnost výpočtu pozic částic 46
7.5 Rendering scény 47
7.6 Zpoždění změn datového modelu 48
7.7 Jiné limitující faktory transakčního přístupu 48
8 Závěr 50
Literatura 51

Seznam obrázků

1 Schéma aktivní transakce 15
2 Databáze scény 16
3 Schéma cyklotronu 20
4 Celkový pohled na scénu 24
5 Postup vytváření *SoTriangleStripSet* 25
6 Pohled do prostoru urychlovače 27
7 Zobrazení trajektorie při správném nastavení 28
8 Zobrazení trajektorie při nevhodně zvolených parametrech 29
9 Ukázka scény s použitím 3D textury 30
10 Zobrazení scény v režimu měření 32
11 Zjednodušený graf scény 33
12 Ovládací prvky scény s urychlovačem 41

Seznam grafů

1 Závislost množství přenášených dat na počtu částic 44
2 Závislost množství přenášených dat na počtu počítačů v distribuční skupině 44
3 Čas potřebný pro vybavení sítě 45
4 Výpočet pozic částic 46
5 Rozložení zátěže pro dva počítače a sto částic 47
6 Výhodnost 3D textury 48

1 Úvod

V současné době se stále více hovoří o virtuální realitě, kolaborativních virtuálních prostředích a počítači podporované spolupráci. Tento projekt částečně zapadá do této oblasti, protože aplikace, která vznikla umožňuje sdílení jedné grafické scény více uživateli. Tato scéna zobrazuje simulaci pohybu částic v kruhovém urychlovači. Všichni uživatelé mohou do této simulace zasahovat a měnit její parametry. Tato část projektu není však nejdůležitější. Hlavním cílem vytvořené aplikace je ukázat možnosti řešení sdílených grafických scén a prověřit výkonové nároky použitých knihoven. Po přečtení tohoto textu by měl čtenář být schopen podobnou aplikaci vytvořit.

Projekt volně navazuje na projekt ročníkový, ze kterého využívá částečně vzhled scény s urychlovačem.

1.1 Prostředky využité pro implementaci

V této kapitole bych čtenáře rád podrobněji seznámil s prostředky využitými k tvorbě této diplomové práce a důvody které vedly k jejich volbě. Dále se zde budu zabývat stručným popisem knihoven, na kterých je tento program postaven.

1.2 Princip distribuovaného výpočtu

Tato kapitola by měla čtenáře seznámit s různými přístupy k tvorbě sdílených grafických scén. Podrobněji se bude zabývat přístupem založeným na transakcích, který je vyvíjen Ing. Janem Pečivou a je inspirován algoritmy využívanými v databázových systémech. Zmíním se i o drobných vylepšeních a rozšířeních, která na můj podnět doplnil Ing. Jan Pečiva do svých knihoven. Tyto drobnosti dle mého názoru ještě zvýší jejich použitelnost.

1.3 Fyzikální a matematická podstata urychlování částic

Na tomto místě se zmíním o důvodech, které mě vedly k volbě simulace urychlování částic v cyklotronu jako demonstrační aplikace. Pro pochopení na první pohled jednoduché simulace bude nutné objasnit fyzikální princip. Pravděpodobně se neobejdeme bez uvedení nezbytných vzorců a

jejich odvození. Čtenář se dozví, že existuje celá řada urychlovačů částic. Popíšeme si zde jejich výhody a nevýhody. Podrobněji se seznámí s principem kruhových urychlovačů a to hlavě cyklotronu. Tato kapitola vyžaduje od čtenáře dobré fyzikální znalosti z této oblasti fyziky, ale její pochopení je nevyhnutelné pro správné používání programu pro simulaci.

1.4 Tvorba 3D scény cyklotronu

V této části bude shrnuta a podrobně popsána celá podstata programu a to jak tvorba distribuované scény z jednotlivých komponent, tak i implementace urychlení částice. Převážnou část věnuji mým algoritmům použitým pro tvorbu scény a různých druhů vizualizace, nebudu se zde příliš zabývat popisem vlastní knihovny Open Inventor. Největší důraz bude kladen na implementaci algoritmů pro sdílení scény po síti, které používají dostupné knihovny. Neopomenu rovněž algoritmy pro výpočet pohybu částice v proměnlivém elektromagnetickém poli, což tvoří jádro testovací simulace.

1.5 Ovládání urychlovače

Zde bych rád popsal tvorbu jednoduchého uživatelského menu pomocí knihovny Open inventor. Věnovat se budu i postupu jakým sdílím proměnné, které řídí celou simulaci a jsou nastavovány tímto menu. Rovněž se zde zaměřím na popis ovládání celého programu. A vysvětlím funkce asociované k jednotlivým položkám v menu. Popíši jednotlivé zobrazované hodnoty jako například intenzitu magnetického pole, intenzitu elektrického pole, ratio (sfázovanost) frekvencí oběhu a frekvence elektrického pole a další.

1.6 Měření a testování

Tato kapitola, tvořící jádro této diplomové práce, bude věnována popisu implementace a prostředkům, které využívám pro měření zpoždění změn datového modelu a množství přenášených dat. Dále zde uvedu výsledky testování výkonnosti na různých systémech a v různých konfiguracích. Porovnám rovněž jakou část celkové náročnosti testovací aplikace tvoří sítová komunikace, renderování grafiky a výpočet fyzikální části simulace.

1.7 Co bude v závěru

Závěrečná kapitola bude obsahovat zhodnocení dosažených výsledků a závěrů z měření. Objeví se zde i zhodnocení z pohledu dalšího vývoje projektu. Zamyslím se i nad možnostmi praktického nasazení použitých knihoven.

2 Prostředky využité pro implementaci

2.1 Volba prostředí a jazyka

Jelikož se jedná o sítovou aplikaci bylo by vhodné, aby byla rovněž přenositelná. Program je tedy implementován v jazyce C++ s využitím nadstavbové knihovny nad openGL - implementace Coin3D, jež je dílem norské společnosti System In Motion. Ta je volně dostupná pod licencí GPL a je plně kompatibilní s Open Inventorem. Dalším důvodem pro tuto volbu byly pozitivní zkušenosti s touto knihovnou získané při tvorbě ročníkového projektu. Podporovaných platforem pro tuto knihovnu je velmi mnoho. Z těch nejznámějších: Windows, Linux, Mac od Apple a obrovské množství Unixů. My se soustředíme pouze na Linux a Windows. Na Linuxu je použitelný překladač gcc, na Windows je vhodný Microsoft Visual Studio, nejlépe ve verzi 6, který je využit pro přeložení testovacího programu - simulátoru a sice ve verzi Microsoft Visual C++ 6.0, Introductory Edition, která je pro nekomerční účely zdarma.

2.2 Open Inventor

Nyní něco blíže k designu Open Inventoru. Je to knihovna napsaná v C++ a postavená nad OpenGL, která posunuje programátora od primitivního OpenGL rozhraní na vyšší úroveň a nabízí mu rozsáhlou množinu C++ tříd. Ta podstatně zjednodušuje práci programátora a dokonce často poskytuje vyšší výkon než přímá implementace v OpenGL. Vyšší výkon je možný díky jistým optimalizacím, které Open Inventor může provádět nad daty scény. Běžný programátor také obyčejně nemá čas provádět profilování a optimalizaci renderovacích algoritmů. Proto již vyprofilované rutiny Inventoru nejsou špatnou volbou. V konečném důsledku se tak programátor může z velké části oprostit od implementačních detailů grafických prvků a používat je již jako vytvořené komponenty. U nich pak nastaví pouze několik určujících parametrů a to vše pomocí jednoduchého schématu objektového

programování. Dobrým příkladem může být základní geometrické těleso koule. U objektu určujícím toto těleso nastavíme parametry typu barva, poloměr ... a máme k dispozici těleso tvaru koule, které můžeme pomocí dalšího objektu třídy *SoTranslation*, sloužícímu k určení pozice těles v prostoru, libovolně umístit. Dalším dosti specifickým rysem Open Inventoru je fakt, že veškeré programování se dá chápat jako sestavování stromově orientovaného grafu. Jednotlivé uzly jsou jakési řídící proměnné průchodu grafem. Jimi můžeme zamezit průchod grafu s implicitním nastavením a vynutit průchod danou větví podle našich představ. Listy grafů jsou pak jednotlivé zobrazované prvky anebo jejich modifikační parametry. Jsou to například již zmiňovaný geometrický objekt koule a jeho umístění. Další neméně důležitou výhodou je že Ing. Jan pečiva doplnil Open Inventor o jednoduché API, které dává programátorovi možnost proměnit běžnou 3D grafickou scénu v grafickou scénu sdílenou a spolupracující. Není třeba provádět žádné změny kódu Open Inventoru stačí jen nahradit DLL Open Inventoru novou upravenou DLL.

2.3 SoWin

Společně s Open Inventorem je pod OS Windows hojně využívána knihovna SoWin, která je založená na stejném aplikačním modelu jako Open Inventor. Pod Operačním systémem Linux ji nahrazuje knihovna SoQt. Názorným příkladem je hned základní kámen jakékoliv nekonzolové aplikace a tou je okno aplikace. Pomocí propracované třídy SoWinViewer a jeho potomků. Tato třída je určena pro vytvoření okna se základními ovládacími prvky pro prohlížení 3D objektu. V programu je konkrétně využita třída SoWinExaminerViewer, která má těchto prvků ze všech možných potomků nejvíce.

2.4 SIM Voleon

SIM Voleon je systém pro vývoj softwaru ve formě přídavné knihovny ke Coin3D. SIM Voleon doplňuje schopnosti na polygonech založeného renderingu o vizualizací objemových dat. Poskytuje tak takzvanou technologii „objemového renderingu“. Právě objemová data jsou většinou výsledkem různých fyzikálních simulací nebo měření. Pro jejich snadné zobrazení poskytuje SIM Voleon technologii umožňující upravovat barvu a průhlednost. Práce s touto knihovnou je totožná s Open Investorem, opět jsou poskytovány uzly, které se přidají do grafu scény a doplní tak schopnosti Coinu o objemový rendering.

2.5 Sdílené grafické scény

Pro sdílené grafické scény se často používá zkratka CVE (Collaborative/Networked Virtual Environment) je rovněž využita jako součást názvu knihovny od Ing. Jana Pečivy, která umožňuje vývojáři snadný přechod od samostatné 3D aplikace k CVE bez velké námahy. Bez této knihovny bychom museli řešit spoustu problémů týkajících se konzistence dat. Většina těchto problémů spadá spíše do oblasti distribuovaných systémů a paralelního zpracování a ne do oblasti grafiky. Tato knihovna opět tvoří rozšíření knihovny Coin3D. Ne příliš rozsáhlými modifikacemi samostatných aplikací lze tedy dosáhnout poměrně robustního distribuovaného řešení, což je velkou výhodou v porovnání s řešením na nižší úrovni.

3 Princip distribuovaného výpočtu

Distribuovaný výpočet ve formě CVE je logickým důsledkem vývoje v počítačové grafice, virtuální realitě a je podporován stále se zlepšující dostupnosti internetu. CVE se používají jednak pro rozdělení zátěže vyplývající ze složitých výpočtů mezi více počítačů a nebo pro spolupráci více uživatelů na jednom projektu, popřípadě ve hrách a zábavě všeobecně.

3.1 Možné přístupy k řešení CVE

Z historického hlediska je tato oblast poměrně mladá, první pokusy pocházejí z osmdesátých let kdy Ministerstvo obrany Spojených států spustilo projekt SIMNET [7] následovaný DIS a HLA [8]. Tyto projety byly používány především pro armádní simulace. Z hlediska konzistence poskytoval systém DIS slabé záruky, nestaral se totiž o ztracené pakety ani o jejich pořadí. Systémy s lepší konzistencí jsou většinou založeny na kauzalitě. Například DIV řeší tyto problémy pomocí globálního pořadí zpráv což ovšem je velmi náročné, ale zároveň dostatečně bezpečné. Tradiční CVE přístupy využívají jeden ze dvou konceptů atomické akce a nebo single write. Single write dovolí každému klientovi jen jednu zápisovou operaci v danou chvíli. Při zpracování scény ovšem občas potřebujeme provést sadu čtecích a zápisových operací atomicky, tedy všechny nebo žádnou z nich, protože jinak by mohlo dojít k porušení konzistence scény. Koncept atomických akcí, použitý například v Age of Empires, toto částečně odstraňuje. Každý počítač má sady napevno zakódovaných operací, které jsou spouštěny atomicky. Při pečlivém designu je toto dobré a bezpečné řešení, které je ovšem špatně rozšiřitelné.

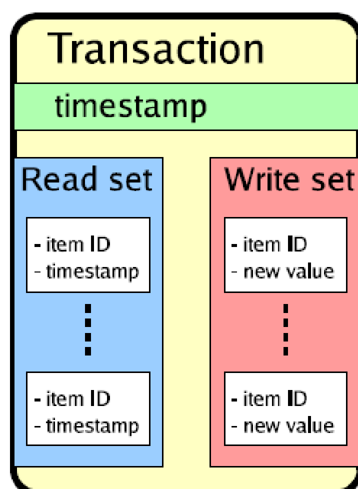
3.2 Transakční přístup

3.2.1 Transakční návrh

Transakční přístup [9] v podání Ing. Jana Pečivy předpokládá, že scéna je kompletně zkopírovaná na všech počítačích a všechny počítače spolu mohou komunikovat přes spolehlivé spojení jako TCP/IP. Protože jsou scény stejné, tak čtení může být prováděno přímo, naproti tomu zápis je prováděn prostřednictvím transakcí. Jejich návrh vychází z databázových systémů, kde jsou transakce spouštěny v několika krocích. Napřed dojde k vytvoření transakce a jejímu poslání do databázového systému. Po jejím přijetí určí plánovač optimální pořadí spuštění přijatých transakcí. Spuštění této transakce vede buďto k jejímu potvrzení nebo zrušení. Při potvrzení se vše uloží do databáze. Při zrušení se naopak vše již provedené během spouštění této transakce odstraní. Klasický transakční přístup ovšem úplně nevyhovuje potřebám CVE. Jedním z důvodů je, že databáze scény replikovaná na síti a proto je potřeba kontrola souběžného přístupu. Zamykání není z důvodů jeho velké časové náročnosti nejvhodnějším řešením. Pokud bychom ale zaručili spuštění všech transakcí na všech počítačích ve stejném pořadí, tak dostaneme všude stejné změny scény to je zaručeno atomickým multicastem. Takto jsou distribuované kopie scén udržovány synchronizované bez dalších nároků na síťovou komunikaci.

3.2.2 Aktivní transakce

Byla navržena transakce skládající se z časového údaje(časového razítka), prvků čtení a prvků zápisu. Časový údaj je generován v čase, kdy je transakce vydána klientem. Množina prvků čtení obsahuje identifikaci všech objektů a jejich časové údaje. Množina prvků pro zápis obsahuje identifikaci všech aktualizovaných prvků a jejich nové hodnoty.



Obr.1: Schéma aktivní transakce

Časové razítko transakce je unikátní v celém systému a je podle něho vytvářeno absolutní pořadí transakcí. Je generováno na jednotlivých počítačích pomocí synchronizovaných hodin. Aby v systému nedocházelo k výskytu dvou stejných časových údajů jsou ještě rozlišeny pomocí IP adresy.

3.2.3 Stavy transakcí

Aktivní transakce [9] procházejí od jejich vzniku několika stavy.

1.Vytvoření

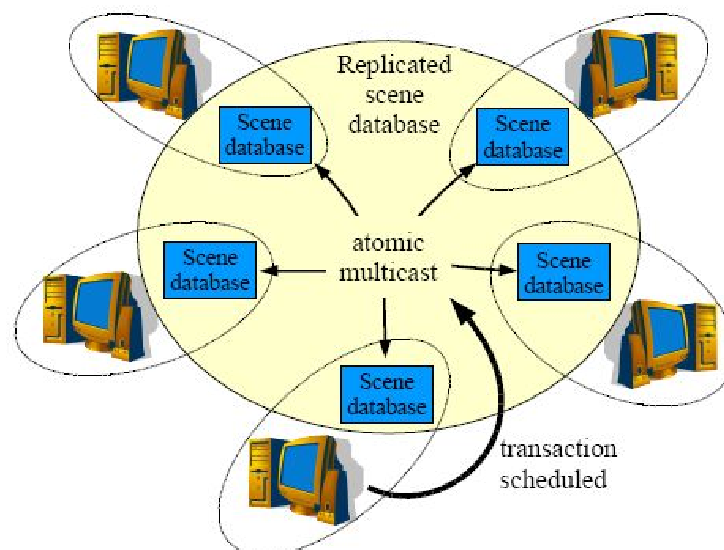
2.Naplánování

3.Příjem

4.Spekulativní spuštění

5.Validace

Transakce jsou vytvořeny klientem, zařazeny do plánovače. Následuje atomický multicast a všichni klienti dostanou danou transakci a může být zpuštěna. Protože atomický multicast je časově náročná operace, je spuštění transakce rozděleno do dvou fází, spekulativního spuštění a validace. Jakmile je transakce doručena je ihned spekulativně spuštěna. Pokud proběhne validace v pořádku je transakce potvrzena.



Obr. 2: Databáze scény

3.2.4 Kontrola souběžnosti

Problémy jako velké zpoždění, než se projeví aktualizace scény, a souběžné, ale odporující si aktualizace scény, vznikají díky latenci sítě, která se může pohybovat od 100us na lokálních sítích do

půl sekundy v internetu. První problém může být minimalizován spekulativním spouštěním transakcí, takže uživatel pracující s CVE nemusí čekat, až proběhne validace atomického multicastu [9], která může trvat značnou dobu. Druhý problém je řešen právě časovými razítky transakcí. Jedna z nich má vždy nižší časový údaj a je spuštěna jako první. Vše proběhne v pořádku. Druhá protichůdná transakce však již validací neprojde, takže je zrušena a zůstane bez efektu.

3.3 Jak fugují tyto knihovny v praxi

Praktické použití těchto knihoven je poměrně jednoduché. Vzhledem k tomu, že se jedná o novou záležitost, dochází ještě stále k jejich častým změnám, které jejich praktické nasazení zatím komplikují. Dalším drobným nedostatkem je, že k nim zatím neexistuje podrobná dokumentace, takže práce s nimi vyžaduje trpělivost. Konkrétní postupy jsou podrobně vysvětleny na příkladech v kapitolách zabývajících se vytvářením testovací aplikace. Konkrétně se jedná o kapitoly 5.4 Od samostatné aplikace k CVE, 5.5.3 Vytvoření transakce, 6.3 Sdílení hodnot v menu. Jejich zhodnocení z hlediska výkonnosti je nejlépe patrné z kapitol 7.x zabývajících se měřením a testováním.

3.4 Doplnění funkčnosti knihoven

Pro účely testovací aplikace, bylo potřeba rozšířit funkce knihovny, tak aby umožňovala distribuci hodnot uložených i v proměnných odvozených od třídy *SoMField*. Těchto typů jsou například hodnoty *SoMFFloat* a *SoMFString*. Právě proměnou typu *SoMFString* využívám pro sdílené ovládací menu urychlovače. Přes toto menu se nastavují globální proměnné, které řídí celou simulaci a jsou takto udržovány stejné pro všechny počítače v distribuční skupině[4]. Další důležitou věcí pro mě byl synchronizovaný čas, ten je nyní k dispozici v třídě *SbTimeStamp*. Dostupný synchronizovaný čas určitě nalezne spoustu uplatnění v běžné praxi s těmito knihovnami.
V průběhu psaní testovací aplikace došlo k výraznému zjednodušení způsobu připojování počítačů do distribuční skupiny. To co v první verzi testovací aplikace zabralo dvě strany zdrojového kódu je nyní možné příkazy na dva řádky. Tento kód, který se v každém programu využívajícím tyto knihovny opakoval, se začlenil přímo do knihoven. Rovněž stabilita těchto knihoven se v průběhu používání zlepšila. Používáním CVE knihoven byly odhaleny chyby, které tuto nestabilitu způsobovaly a byly následně Ing. Janem Pečivou odstraněny. Spolehlivost je tedy po celkem zásadních změnách provedených ve verzi z března 2006 dobrá.

4 Fyzika urychlování částic

4.1 Proč používat urychlovače částic

Pro studium vlastností, struktury a interakcí elementárních částic, jakož i pro aplikace v různých oblastech vědy a techniky (včetně medicíny), je potřeba použít částic urychlených na vysoké kinetické energie. Uměle urychlit dovedeme pouze stabilní elektricky nabité částice - elektrony e^-, pozitrony e^+ , protony p^+, deuterony d^+, ionty hélia He^{++}=a-částice a ionty těžších prvků. Vysokoenergetické částice bez náboje (jako jsou fotony g, neutrony n^o, neutrální piony, ...) a krátkožijící částice (p-mezony, hyperony, ...) lze pak získat sekundárně - interakcemi urychlených nabitých částic s částicemi ve vhodném terčíku. Přístroje, které působením silných elektrických a magnetických polí urychlují nabité částice, se nazývají urychlovače. Podle způsobu technické realizace a tvaru dráhy, na níž urychlování částic probíhá, rozdělujeme urychlovače na dva základní typy: lineární **a** kruhové.

4.2 Součásti urychlovačů částic

Než se budeme zabývat jednotlivými typy urychlovačů, zmíníme se o dvou součástech, které mají všechny urychlovače - zdroj urychlovaných částic a terčík.

4.2.1 Terčík

Terčík, na nějž dopadá svazek urychlených částic, je buď vnitřní - je umístěn uvnitř urychlovacího systému, nebo vnější - svazek částic je vyveden ven z urychlovací trubice. Rovněž sekundární částice, produkované na vnitřním terčíku (jako jsou p nebo K mesony), se působením magnetického a elektrického pole vyvádějí ve formě svazku do prostoru laboratoře, kde jsou umístěny měřící aparatury (detekční přístroje, bublinové komory atd.). Při dopadu urychlených částic na terčík se většina kinetické energie částic mění na teplo - ostřelovaný terčík se zahřívá. Aby nedošlo k jeho tepelnému poškození či odpaření terčíkové látky, je nutno toto ztrátové teplo (může činit i stovky wattů) odvádět - terčík se fixuje na masivní kovovou podložku s dutinou, chlazenou protékající vodou (podobně jako anody výkonových rentgenových trubic).

4.2.2 Iontový zdroj částic

Zdroj urychlovaných částic emituje do "startovacího" místa urychlovacího systému požadovaný druh částic, jako jsou elektrony, protony či těžší ionty. V nejjednodušším případě se jedná o ionizační trubici obsahující příslušný zředěný plyn (např. vodík H), kde v doutnavém výboji mezi katodou a anodou (při napětí cca stovky voltů) vznikají ionty (u vodíku jsou to protony p^+) a ty jsou pomocí tenké kapiláry vedeny "odsávací" elektrodou do urychlovacího systému. Pro urychlovače elektronů je zdrojem prostá žhavená katoda (termoemise elektronů) opatřená vhodnými urychlujícími a fokusujícími anodami - "elektronovým dělem" - podobně jako u obrazovky. U velkých urychlovačů vysokých energií se jako zdroje částic k urychlení někdy používají injektory - do hlavní komory jsou částice vstřikovány pomocným lineárním urychlovačem (s energií jednotky až desítky MeV) a následně urychlovány na požadovanou vysokou energii (GeV).

4.3 Rozdělení urychlovačů a jejich perspektivy

4.3.1 Lineární urychlovače

Lineární urychlovače urychlují nabité částice působením elektrického pole během jejich pohybu po lineární přímkové dráze. Můžeme je rozdělit na elektrostatické (vysokonapěťové) a vysokofrekvenční. Jejich principem je přidávání energie elektronu na přímkové dráze pomocí vysokého napětí. Jejich hlavní nevýhodou jsou obří rozměry.

4.3.2 Kruhové urychlovače

Velmi efektivním způsobem, jak urychlit nabité částice na vysoké energie, je jejich mnohonásobné urychlení v elektrickém poli, kam jsou částice opakovaně vraceny po kruhové dráze působením magnetického pole. Na částici s nábojem q je zde aplikována nejen elektrická urychlující síla

$$\overrightarrow{F_e} = q * \overrightarrow{E} \qquad (1)$$

ale i Lorentzova síla

$$\vec{F}_m = q * (\vec{v} \times \vec{B}) \qquad (2)$$

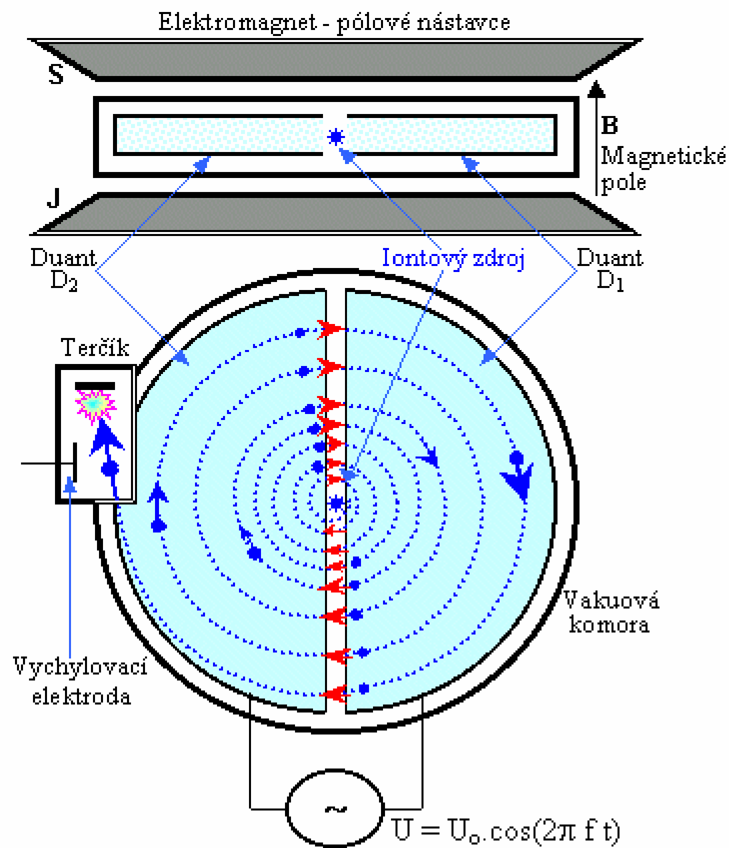
působící v magnetickém poli intenzity B kolmo ke směru pohybu nabité částice rychlostí v. Tato magnetická síla způsobuje, že nabitá částice se bude pohybovat po kruhové dráze o poloměru

$$R = \frac{m * |\vec{v}|}{q * |\vec{B}|} \qquad (3)$$

Je-li ve vhodných místech této kruhové dráhy synchronně aplikováno elektrické urychlující pole (v tečném směru), budou částice periodicky urychlovány při každém svém oběhu.

4.3.2.1 Cyklotron

Základním typem kruhového urychlovače je cyklotron [6] (první cyklotron vyvinul E.O.Lawrenc již v r.1932), jehož princip je schématicky znázorněn následujícím obrázkem.



Obr.3: Model cyklotronu

Mezi póly silného elektromagnetu jsou v ploché vakuové komoře upevněny dva duté poloválce D1 a D2, tzv. duanty, mezi nimiž je urychlovací mezera. Duanty jsou připojeny ke zdroji střídavého napětí:

$$U = U_0 * \cos(2p * f * t) \tag{4}$$

o amplitudě U_o a frekvenci f (bývá kolem 20MHz), takže v mezeře mezi deskami je střídavé elektrické pole. Nabité částice vstupují do středu urychlovací mezery z iontového zdroje . Následkem síly, kterou elektrické pole v mezeře působí na částici s nábojem q a hmotností m, je částice vtažena do jednoho z duantů (který má právě opačnou polaritu) s určitou rychlostí v_1. Uvnitř duantu, kde je elektrické pole odstíněno, působením silného magnetického pole B opíše částice půlkružnici o poloměru:

$$R_1 = \frac{m * \left|\vec{v_1}\right|}{q * \left|\vec{B}\right|} \tag{5}$$

Tento poloměr je dán rovnováhou mezi odstředivou silou a Lorentzovou magnetickou[1] silou:

$$\frac{m * v_1^{\,2}}{R_1} = q * B * v_1 \tag{6}$$

Doba, za kterou projde částice tuto půlkružnici, je:

$$T = \frac{p * R_1}{v_1} = \frac{2p * m}{q * B} \tag{7}$$

Vidíme, že tato doba (půl-perioda) oběhu částice nezávisí na její rychlosti ani na jejím poloměru dráhy. Frekvence kruhového oběhu částice tedy je:

$$f = \frac{q * B}{2p * m} \tag{8}$$

a je konstantní, protože m, q a B jsou v daném uspořádání konstanty. Jestliže jsou duanty napájeny střídavým napětím právě o této frekvenci f (je splněna podmínka rezonance či synchronizace), pak

v okamžiku kdy částice opíše půlkružnici v prvním duantu a ocitne se opět v urychlovací mezeře, je polarita duantů již opačná a částice bude opět urychlena elektrickým polem, takže do druhého duantu vletí s větší rychlostí $v_2 > v_1$. V druhém duantu se bude pohybovat opět po kružnici, nyní však o poloměru

$$R_2 = \frac{m * \left|\vec{v}_2\right|}{q * \left|\vec{B}\right|} \tag{9}$$

který je větší než byl R_1, ale se stejnou periodou a frekvencí kruhového pohybu.

Stejným způsobem je pak částice při každém svém průchodu mezerou mezi duanty znovu a znovu urychlována, přičemž se pohybuje po kružnicích s rostoucím poloměrem, tedy po spirále. Z poslední své dráhy o maximálním poloměru (blízkém poloměru duantů) je urychlená částice elektrostaticky nebo magneticky vychýlena a vyvedena do prostoru terčíku, na nějž narazí a vyvolá tam jaderné procesy.

Nastíněný princip činnosti cyklotronu bude při konstantní frekvenci fungovat jen do té doby, kdy hmotnost urychlované částice můžeme považovat za konstantní, tj. pouze v nerelativistické oblasti. Chceme-li použít cyklotronu k urychlování částic na vyšší energie, kdy rychlost částic je již srovnatelná s rychlostí světla, přestává být hmotnost částice m konstantní [1], ale zvyšuje se s rostoucí rychlostí:

$$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{10}$$

Ve stejném tempu se snižuje frekvence oběhu částic v konstantním magnetickém poli:

$$R = \frac{m_0 * \left|\vec{v}\right|}{q * \left|\vec{B}\right| * \sqrt{1 - \frac{v^2}{c^2}}} \tag{11}$$

Aby mohla být částice dále urychlována i v této relativistické oblasti, je potřeba modulovat frekvenci urychlovacího napětí tak, aby byla stále v rezonanci s frekvencí oběhu částice. Takto upravený

cyklotron se "synchronizací" se nazývá synchrocyklotron nebo relativistický cyklotron [6] (ve starší literatuře se vyskytuje i název "fázotron"). Tyto přístroje pracují v pulsním režimu, přičemž kmitočet urychlovacího napětí na duantech je modulován a mění se cca 50-krát za vteřinu z hodnot cca 25MHz na cca 12MHz, používají se pro urychlování protonů na energie do cca 1GeV.

4.3.2.2 Synchrotron

Pro urychlování částic na velmi vysoké energie vychází v kruhovém urychlovači velký poloměr jejich orbit, takže cyklotronový způsob se spirálovým pohybem částic v ploché vakuové komoře již není prakticky použitelný. Aby dokonale vakuový prostor nebyl enormně velký, stejně jako elektromagnety, je nutno použít kruhové urychlovače s pevnou kruhovou drahou. Aby se nabitá částice urychlovala a udržela se na pevné kruhové dráze o poloměru R, je potřeba aby s rostoucí rychlostí v(t) urychlovaných částic se s časem synchronně zvyšovala jak frekvence f(t) urychlovacího napětí, tak intenzita magnetického pole B(t), která již nemůže být konstantní, ale je rovněž funkcí času.

4.3.2.3 Betatron

Urychlovací trubice betatronu má tvar prstence (toroidu) zhotoveného z elektricky nevodivého materiálu (sklo, porcelán) s vysokým vakuem uvnitř. Trubice je umístěna ("navléknuta") mezi pólovými nástavci elektromagnetu, napájeného střídavým proudem. Elektrony jsou ve vhodném okamžiku (vhodné fázi periody střídavého proudu) vstřikovány do urychlovací trubice elektronovou tryskou, tvořenou žhavenou katodou, mřížkou a urychlující a fokusující anodou - je to pobobné "elektronové dělo" jako je u obrazovky. Časově proměnné magnetické pole indukuje v trubici vířivé elektrické pole, jehož elektromotorická síla, směřující podél kruhové dráhy, tyto elektrony urychluje.

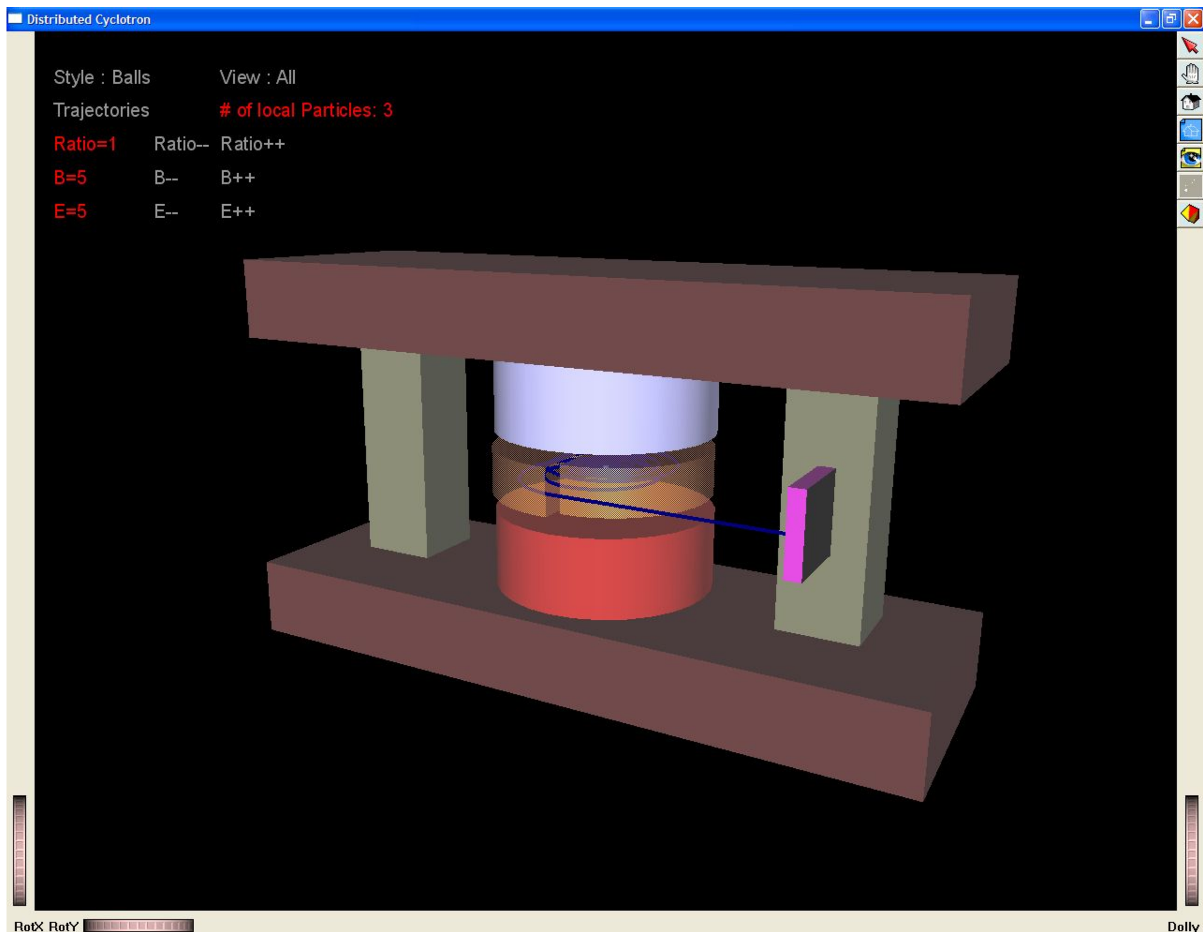
4.3.3 Perspektivy urychlovačů

Jakkoli je princip kruhového urychlování nabitých částic velmi úspěšný a efektivní, zdá se, že kruhové urychlovače se již přiblížily k hranicím svých možností. Pokud bychom chtěli nabité částice urychlovat na ještě podstatně vyšší energie při reálně dostupných průměrech kruhové dráhy (tj. průměrech urychlovacích trubic), čím dál více by se uplatňoval jev vzniku synchrotronového záření, které by odnášelo značnou část kinetické energie částic a nakonec by znemožnilo další urychlení. Zdá se tedy, že budoucí urychlovače pro nejvyšší energie budou muset být lineární.

5 Scéna urychlovače

5.1 Úvodem

Design Open Inventoru vychází z konceptu grafu scény. Tedy, scéna je složena z uzlů - anglicky nodes. Takovýto graf nám pak reprezentuje naši scénu.



Obr. 4: Celkový pohled na scénu

5.2 Skladba scény urychlovače

5.2.1 Jednoduché objekty

Scéna se skládá především z jednoduchých objektů, které poskytuje knihovna Open Inventor. Tyto objekty tvoří nepodstatnou část simulace. Jsou to dva magnety, které jsou vytvořeny pomocí funkce

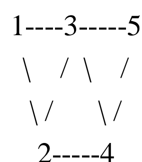
využívající třídy SoCilinder. Modrý je natočen severním pólem do středu a červený je do středu natočen pólem jižním.

Další součástí je emitor částic – elektronů, který je umístěn uprostřed mezi dvěma duanty a je opět tvořen stejnou funkcí jako magnety.

Na obrázku 2 je dále dobře patrná nosná konstrukce celého urychlovače a terčík pro dopad elektronů, který je znázorněn temně fialovou barvou. Scéna záměrně nevyužívá textury ani další světla protože podstatou je rychlost a názornost simulace nikoliv dokonalý vzhled. Textury byly původně použity, ale celou scénu značně znepřehledňovaly. V levém horním rohu je vidět menu kterému bude věnována samostatná kapitola.

5.2.2 Objekty vytvářené z trojúhelníků

Objekty vytvářené z trojúhelníků jsou v mém případě duanty-duté půlválce, které jsou tvořeny ze dvou půlkružnic a z jednoho půlkulatého pásku (půl obruče). Nejjednodušší možností jak generovat trojúhelníky je GL_TRIANGLES, kdy specifikujeme vždy tři body, které budeme nazývat vertexy. Každý z těchto vertexů se skládá ze třech prostorových souřadnic - x,y,z. Tyto tři vertexy tedy prostřednictvím OpenGL vytvoří jeden trojúhelník. Většina trojúhelníků sdílí vrcholy se svými sousedy, proto byly vymyšleny věci jako GL_TRIANGLE_STRIP, který vždy z prvních třech vertexů vykreslí první trojúhelník, ale od té chvíle již vykresluje trojúhelník s každým dalším vertexem, přičemž chybějící dva vertexy použije z předchozího trojúhelníku. Vše demonstruje následující obrázek.



Obr. 5: Postup vytváření *SoTriangleStripSet*

Kromě velmi ojedinělých případů bude použití GL_TRIANGLE_STRIP vždy rychlejší než obyčejné trojúhelníky, neboť se do OpenGL posílá méně dat pro vykonání stejné práce. Z toho vyšli i designéři Open Inventoru a navrhli dvě třídy pro rendrování trojúhelníků: SoTriangleStripSet a *SoIndexedTriangleStripSet.* Níže uvádím zdrojové kódy funkcí využívajících třídu SotriangelStripSet. Nejdříve se vygeneruje seznam řídících vrcholů a potom se z nich vytvoří trojúhelníky. Trojúhelníky v podobě SoTriangleStripSet přidáme pod hlavní kořen scény. Vše demonstruje následující útržek kódu pro vytvoření půl-obruče.

```
float c[detail_valec];
float d[detail_valec];
int i;
for(i=0;i<(detail_valec-1);i++)
  {
  float b;
  b=180/(detail_valec-1)*M_PI/180*i;
  c[i]=sin(b);
  d[i]=cos(b);
  }
c[detail_valec-1]=0;
d[detail_valec-1]=-1;

float vertices[detail_valec*2][3];
int j;
for(j=0;j<(detail_valec*2);j++)
{
vertices[j][0]=c[j / 2]*radius;
vertices[j][1]=(j % 2)*hight;
vertices[j][2]=d[j / 2]*radius;
}
 SoCoordinate3 *coords = new SoCoordinate3;
 coords->point.setValues(0,detail_valec*2, vertices);
 root->addChild(coords);

SoTriangleStripSet *strip = new SoTriangleStripSet;
strip->numVertices.set1Value(0, detail_valec*2);
root->addChild(strip);
```

5.2.3 Částice

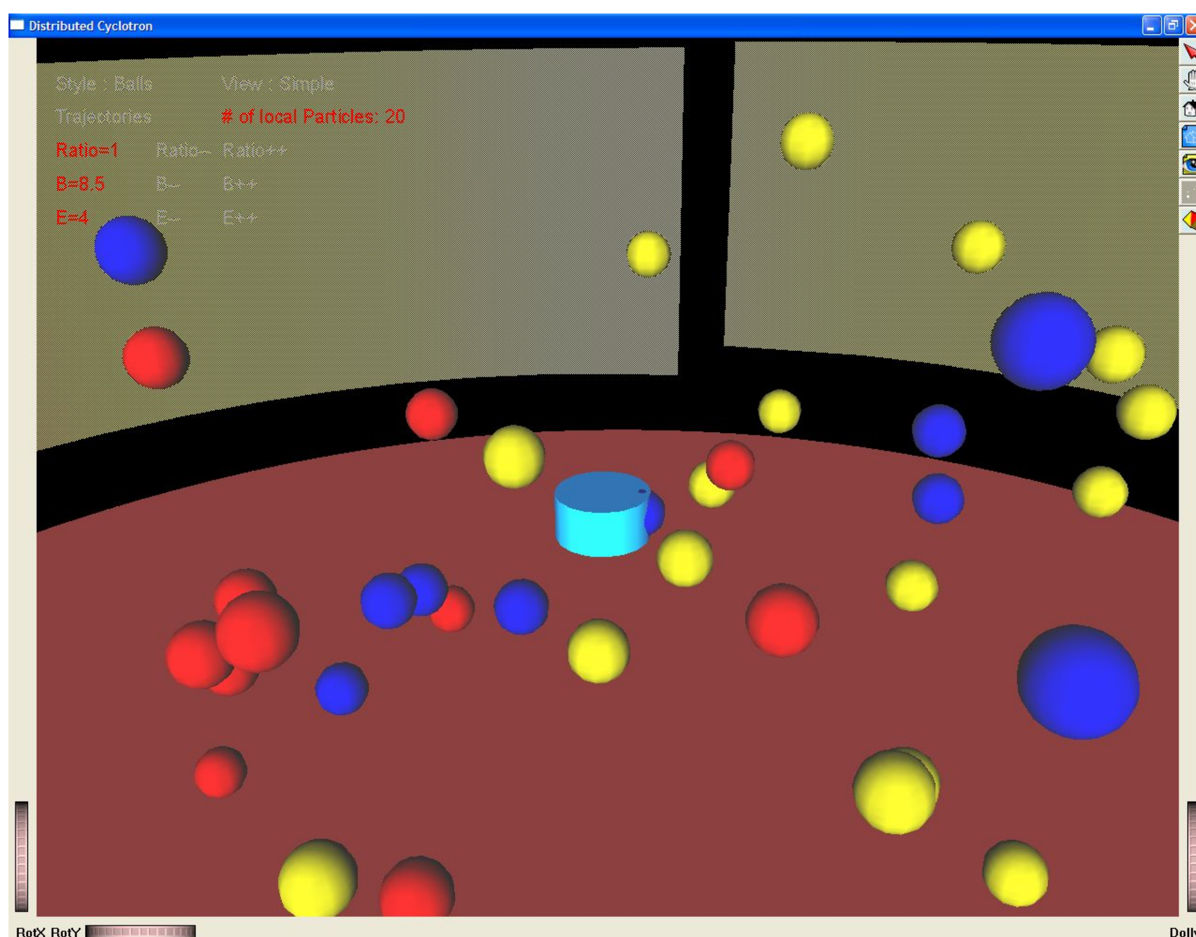
Pro částici je vytvořena vlastní třída, která uchovává potřebné informace o elektronu, jako jsou ukazatel na uzel *SoTranslation*, ve kterém je uchovávána pozice částice, x-y-z-složka rychlosti, a několik příznaků potřebných pro urychlování. Všechny částice se ukládají do vektoru částic. Jednotlivé počítače v distribuční skupině mají ještě vlastní seznam ukazatelů na objekty částic se kterými pracují. Konstruktor nastavuje všechny potřebné hodnoty příznaků do výchozích pozic a generuje náhodnou počáteční rychlost částice, což simuluje její emitování ze zdroje. Dále rovněž vytvoří podgraf scény reprezentující tuto částici. Tento podgraf se skládá z uzlu určujícího pozici, uzlu pro materiál, a uzlu reprezentujícího kouli. K vytvoření koule je použita třída *SoSphere*. Pod kořenem tohoto podgrafu je rovněž umístěn uzel *SoSwitch*, který má pouze jednoho syna a to podgraf pro vykreslování trajektorie částic.

Funkce *activate* respektive *deactivate* se starají o přidání, odebrání podgrafu z celkového grafu scény.

Funkce *getPos* vrací poslední hodnotu z uzlu *SoTranslation*, což je aktuální poloha částice v urychlovači.

Elektron má dále důležitou funkci *setPos*, která tvoří jádro celého urychlovače. Počítá totiž pozici elektronu podle okolí elektronu více se o ni zmíníme v následující kapitole.

Graficky je elektron reprezentován koulí s barvou, která se mu přiděluje při jeho vytvoření. Podle příslušnosti k jednotlivým počítačům. Další možností reprezentace jsou volumetrická data udávající hustotu elektronů v prostoru urychlovače.



Obr. 6: Pohled do prostoru urychlovače, částice řízené jednotlivými počítači jsou odlišeny barevně

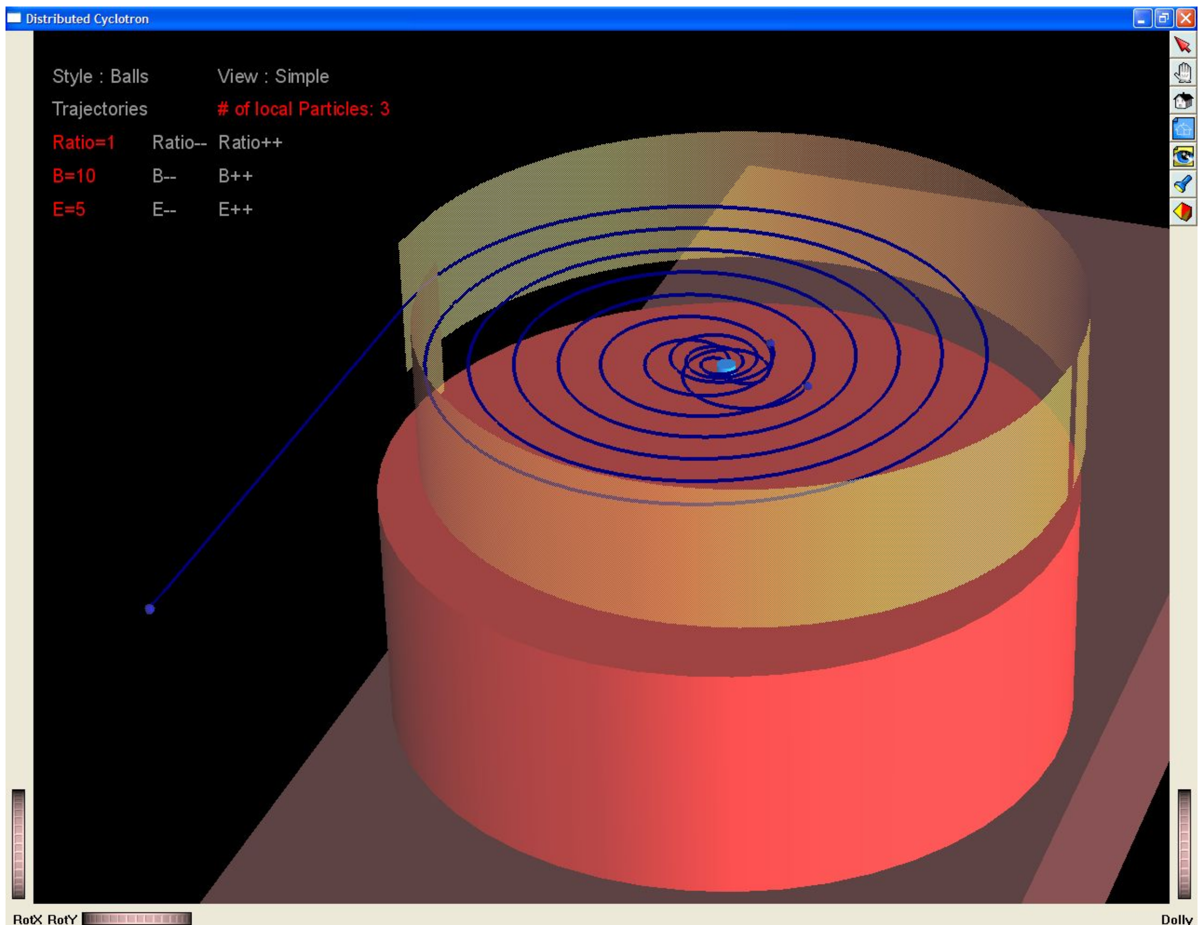
5.2.4 Trajektorie částice

Zobrazení trajektorie částice je kvůli výkonnosti provedeno pomocí třídy *SoLineSet.* Efektnější zobrazení pomocí nurbs křivek je výrazně náročnější na výkon grafiky. Každá částice má pro svoji trajektorii sestaven podgraf obsahující uzel *SoDrawStyle* pomocí kterého se nastaví tloušťka čáry. Dále obsahuje uzel *SoCoordinate3*, ve kterém jsou uloženy řídící body této křivky respektive

koncové body jednotlivých úseček z nichž je křivka složena. Vždy než počítám novou pozici částice, tak hodnotu předchozí pozice přidám do uzlu *SoCordinate3* a zaktualizuji počet řídících bodů.

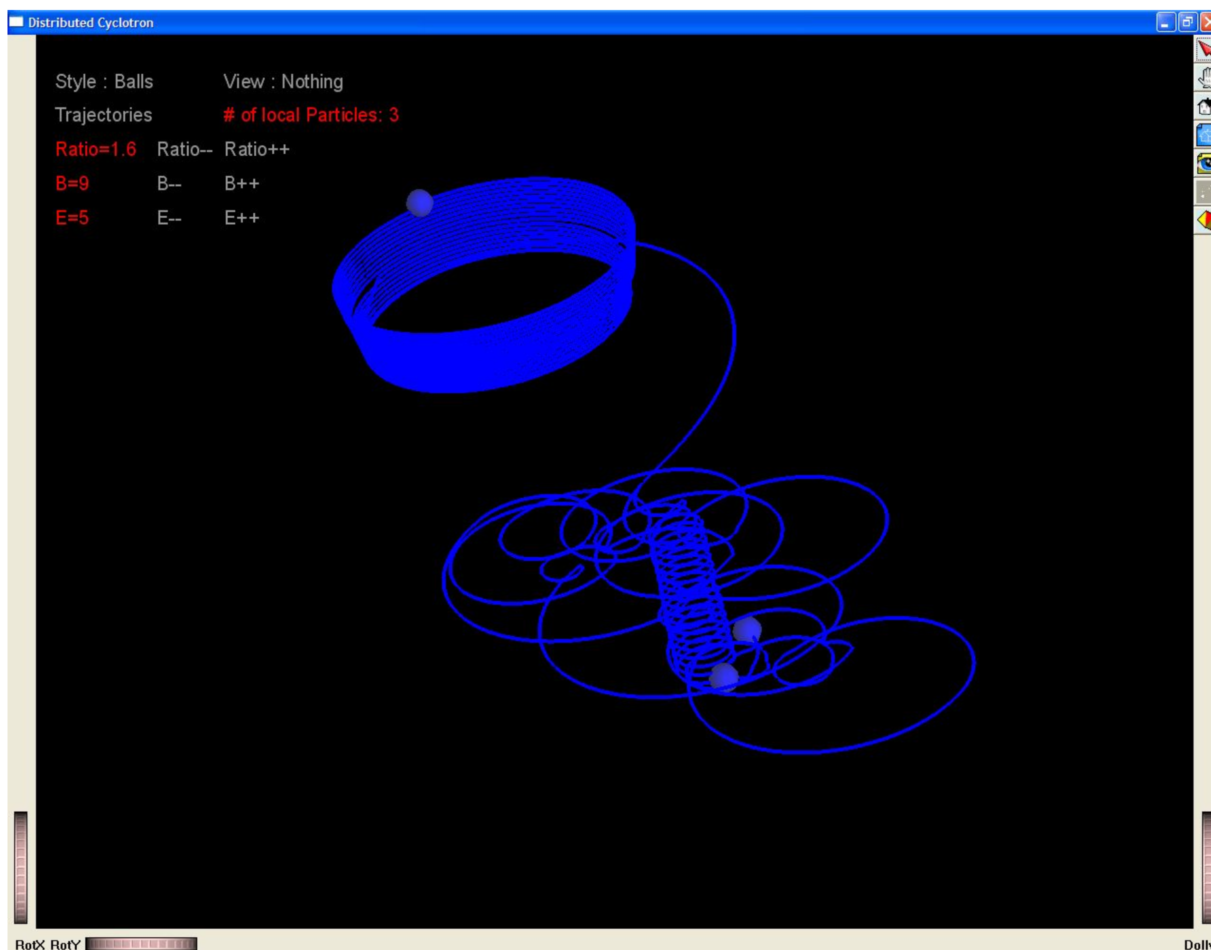
```
c->point.set1Value(i++, SbVec3f(trans->translation.getCveValue(SoField::LATEST)));
lset->numVertices.set1Value(0, c->point.getNum());
```

Pokud částice zanikne, což se může stát buďto kolizí se stěnou urychlovací komory nebo terčíkem, dojde pomocí metody *point.deleteValues(začátek, konec)* k vymazání všech řídících bodů křivky a může se začít kreslit nová trajektorie.



Obr. 7: Zobrazení trajektorie při správném nastavení

Následující obrázek ukazuje jak se částice dostala do magnetické pasti. Nedostane se tudíž do prostoru mezi duanty a nemůže dojít k jejímu urychlení na kruhové dráze setrvá dokud nenarazí do horní nebo dolní stěny urychlovače.

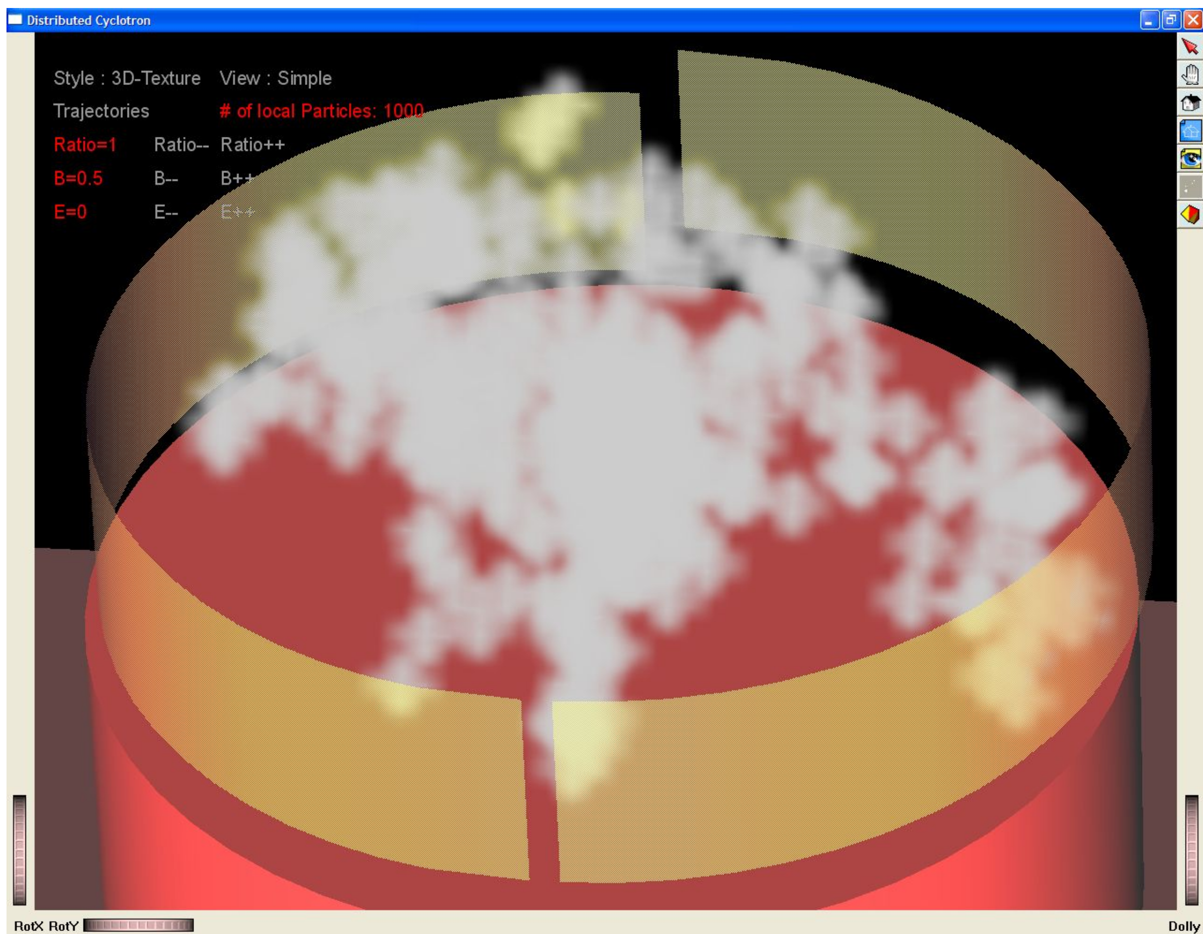


Obr. 8: Zobrazení trajektorie při nevhodně zvolených parametrech

5.2.5 Částice jako objemová data

Vzhledem k velkému počtu urychlovaných částic jsem se rozhodl zobrazovat je jako objemová data, tedy jako 3D texturu. Od tohoto zobrazení jsem si sliboval větší výkonnost při velkém počtu částic, kdy se nebudou zobrazovat jednotlivě, ale pouze budou tvořit složku průhlednosti 3D textury. Tato textura je zobrazena v objemu mezi duanty. Pro volume rendering používám knihovnu SimVoleon.

Její standardní použití probíhá v několika krocích. Nejprve je potřeba vytvořit takzvaný „voxelset“ , který v podstatě tvoří data 3D textury. Potom je potřeba tato objemová data přidat do grafu scény. Postupuje se tak jak je u Open Inventoru běžné, pod příslušný separátor se do grafu scény umístí uzel, obsahující informace o těchto datech tj. rozměry pole a ukazatel na „voxelset“. Dalším uzlem se nadefinuje jakým způsobem se mají 8-bitové hodnoty pole interpretovat Já je vyžívám jako hodnotu průhlednosti této textury. Dalšími možnostmi jsou různé předdefinované barevné mapy. Na závěr je potřeba přidat ještě uzel typu *SoVolumeRender*, který zajišťuje zobrazení těchto dat. Data jsou implicitně zobrazena do krychle o straně velikosti dva a proto je potřeba ještě přidat uzel upravující měřítko. Jedná se o uzel typu *SoScale*.



Obr. 9: Ukázka scény s použitím 3D textury

5.2.6 Celkový pohled na scénu

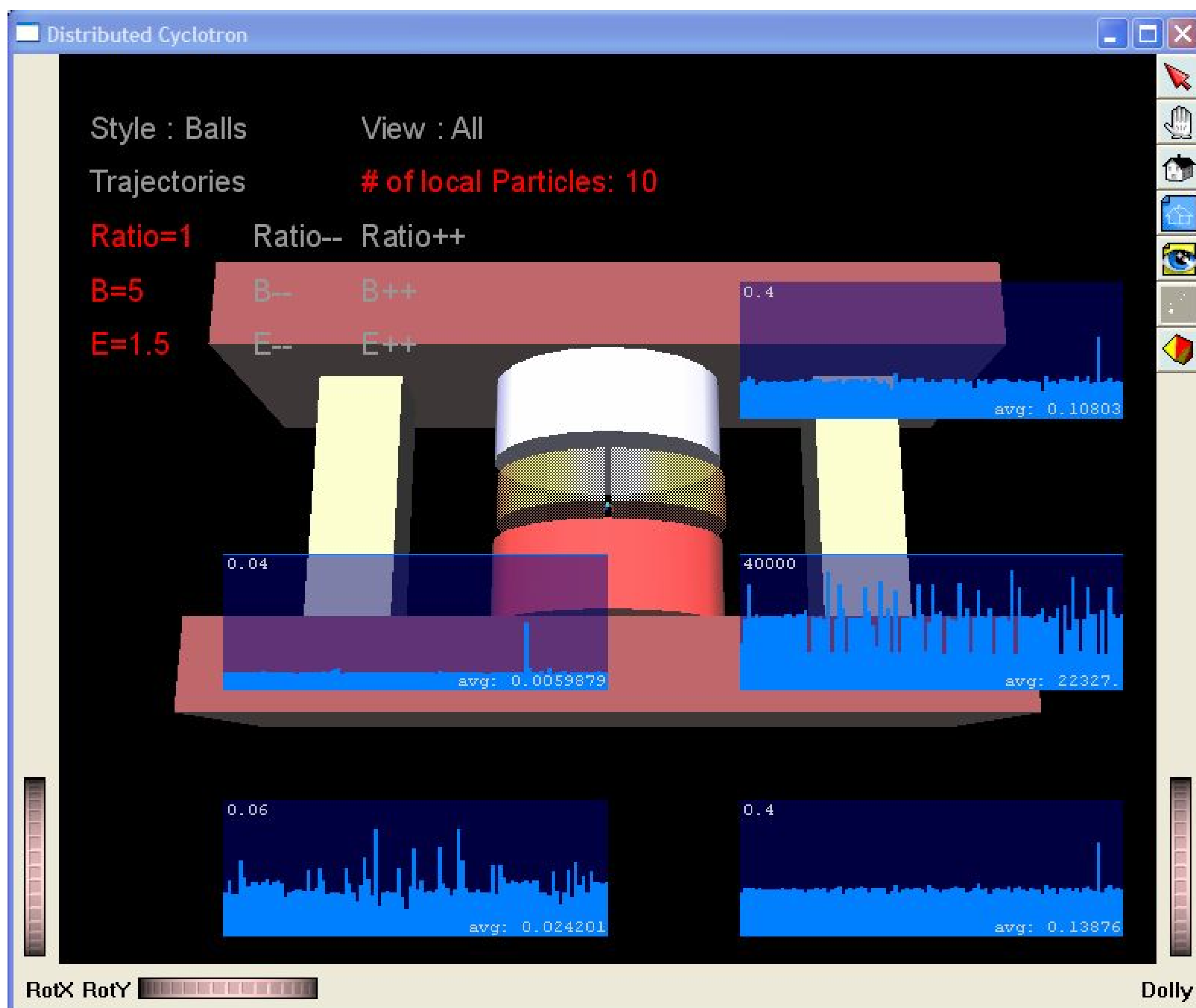
Scéna je snímána perspektivní kamerou, která je tvořena uzlem *SoPerspectiveCamera* umístěným přímo pod kořenem scény. Zde jsou také umístěny uzly nastavující prostředí a osvětlení *SoEnvironmen*t a *SoDirectional Light*. Kromě perspektivní kamery používám rovněž ortografickou kameru *SoOrtographicCamera*. Ta je umístěna pod separátorem tvořícím menu scény. Protože používám oboustranně viditelné objekty tvořené ručně z trojúhelníků je potřeba nastavit *SoShapeHints* na správné hodnoty. Dalším uzlem v grafu je *SoComplexity*, který nastavuje úroveň detailů předdefinovaných objektů. Complexitu mám ve scéně kvůli zvýšení rychlosti nastavenou na poměrně nízkou úroveň. Tuto nízkou hodnotu komplexity používám pouze pokud zobrazuji částice jako kuličky, snížím tak výrazně počet trojúhelníků. Open inventor automaticky zajišťuje úroveň detailů, takže částice které jsou dál od kamery jsou tvořeny menším počtem trojúhelníků. Pro 3D texturu již mám komplexitu nastavenu na běžné hodnoty. Protože používám několik různých druhů zobrazení, tak mám do grafu vloženy speciální uzly *SoSwitch*, které umožňují procházet jen určitými částmi grafu podle toho jak je nastavena jeho *whichChild* . Tím je dosaženo toho, že částice zobrazuji

buďto jako kuličky nebo jako oblak pomocí SimVoleonu nebo je nezobrazuji vůbec a pouze počítám jejich pozice. Toto nastavení se používá pokud nějaké počítače potřebujeme jen pro výpočet pozic částic a zobrazování by ubíralo výpočetní výkon. Pomocí několika dalších takovýchto přepínačů je rovněž možné nastavovat různé typy zobrazení urychlovače a vypínat nebo zapínat zobrazení trajektorií částic.

5.3 Prostředky pro měření časů

Pro měření časů v simulaci používám třídu *SbTime* a to tak, že vždy před a po měřené události zavolám metodu *SbTime::getTimeOfDay()* , která vrací aktuální čas. Jejich rozdílem získáme tuto dobu. Na zobrazování těchto hodnot se nejlépe osvědčily grafy třídy *SoPerfGraph,* protože zobrazují průběh těchto časů a počítají rovněž průměrnou hodnotu ze zobrazených hodnot. Další jejich výhodou je snadné použití. Tento graf je třeba vytvořit potom nastavit parametry jeho zobrazení a na závěr se opět přidá do grafu scény. Je potřeba uchovat si ukazatel na tento graf, aby bylo možno pomocí metody *perfGraphNet->appendValue(float)* přidávat do tohoto grafu nové hodnoty. Ukázka kódu pro měření doby výpočtu pozic elektronů.

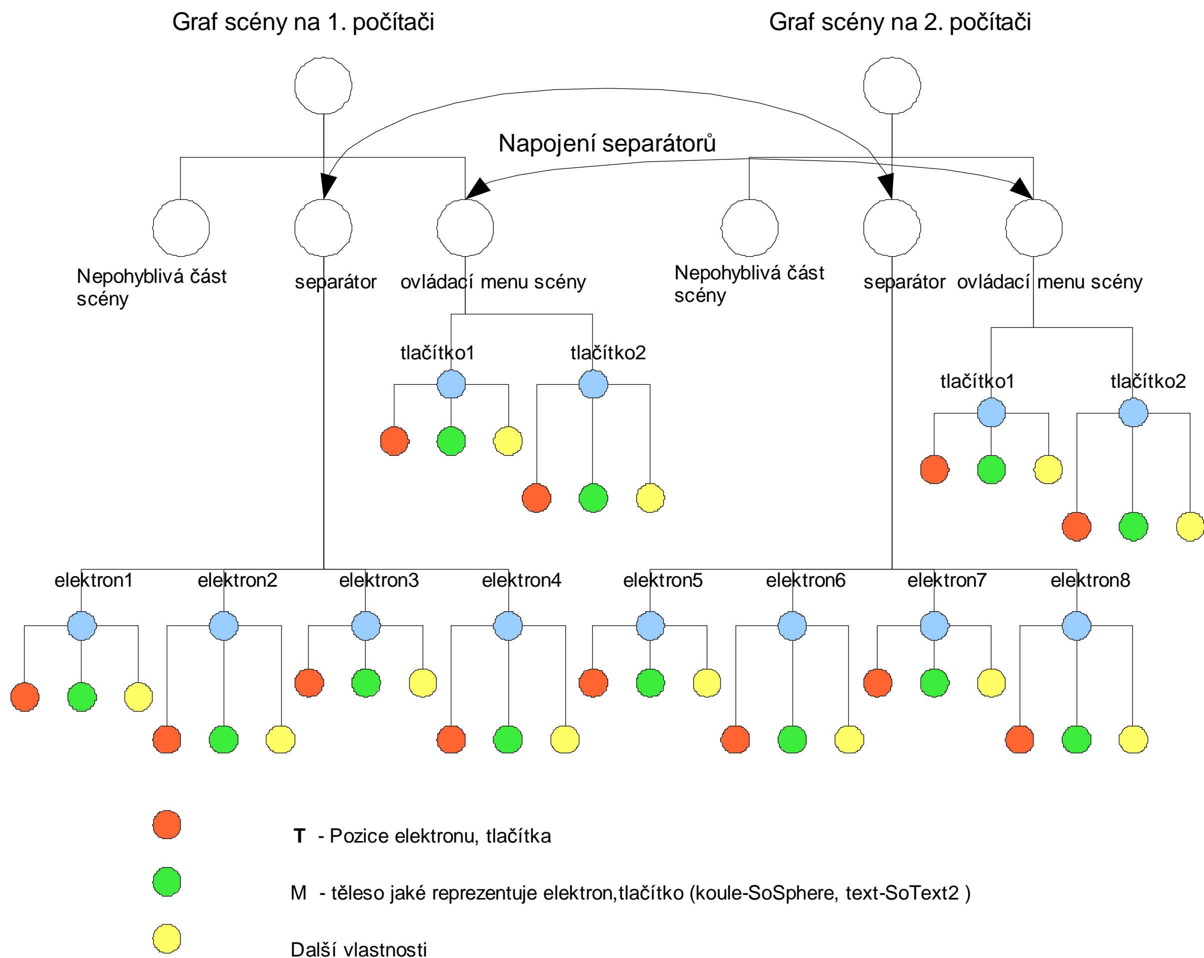
```
 SbTime ctVypocetLast = SbTime::getTimeOfDay();

E = E_max*cos(speed*frekvence*(lastTime.getValue()));
 for (int i=0; i<myBalls.getLength(); i++)
 if (myBalls[i]->aktivni) myBalls[i]->timeTick(dt,E); //provedení výpočtu
   SbTime ctVypocetC = SbTime::getTimeOfDay();
 dtVypocet=ctVypocetC-ctVypocetLast;
 perfGraphVypocet->appendValue(float(dtVypocet.getValue()));
```



Obr.10: Zobrazení scény v režimu měření

5.4 Od samostatné aplikace k CVE

Design Open Inventoru, jak jsem se výše zmínil, vycházející z konceptu grafu scény obsahuje takzvané separátory (uzly zastřešující určitou skupinu poduzlů stromu). Ing Jan Pečiva pozměnil knihovnu Open Inventor tak, aby umožňovala napojení těchto separátorů přes síť, čehož budeme využívat pro distribuci našeho výpočtu. V našem případě jsou tato napojení dvě, jedno je mezi separátory, které mají pod sebou všechny podstromy částic a druhé je mezi separátory ovládacího menu. Vše ukazuje následující obrázek.



Obr. 11: Zjednodušený graf scény

Toto napojení se provede v kódu poměrně snadno to tak, že po inicializaci sítě a vytvoření distribuční skupiny a grafu scény zavoláme na všech počítačích metodu *dg->ppendToDict(separator);* jako parametr volíme ukazatel na příslušný separátor, který chceme napojit. Inicializace sítě se provádí následovně. Nejprve zavoláme metodu *init();* a potom vytvoříme distribuční skupinu. Této distribuční skupině nastavíme jaký datový model chceme používat (*LAST_VALUE* nebo *COMITED_VALUE*) .Model *LAST_VALUE* znamená, že se použije poslední dostupná hodnota zatímco model *COMITED_VALUE* znamená, že se použije hodnota starší, která je již uznána všemi členy distribuční skupiny. Já používám model *LAST_VALUE.* Celý postup ukazuje následující příklad.

```
// Network initialization
  SoNetwork::init();
  tnSetGlobalSimulatedLatency(ms_simLatency);
  SoDistributionGroup *dg = SoNetwork::getDefaultDistributionGroup();
```

```
dg->executionTime /= slowDown;
// set update model
dg->updateGraphModel = SoDistributionGroup::LAST_VALUE;
```

Po inicializaci server nedělá nic a jen poslouchá. Klientské počítače se pomocí metody *dg->connect(addr);*připojí k distribuční skupině. Jako parametr se volí *SbInetAddr addr;* obsahující adresu a port. Až jsou počítače připojeny, libovolný z nich zahájí simulaci vysláním transakce startující simulaci.

```
SbTransaction *tr = new SbTransaction(dg);
tr->setCommitCallback(transactionStartSimName);
tr->schedule();
```

5.5 Funkčnost scény urychlovače

Máme vytvořenu třídu elektronů, kde budou uchovány všechny podstatné informace (poloha, hmotnost, vektor rychlosti a ukazatel na node *SoTranslation* zajišťující přesun koule na správné místo). Dále máme funkci, která zajistí přepočítání pozic elektronů a potřebujeme ji jednou za čas zavolat. Právě k tomu se hodí senzory. Senzory jsou objekty, které sledují určité události a v případě jejich výskytu zavolají programátorem vytvořenou funkci. Mezi možné sledované události patří změna hodnoty pole (*SoFieldSensor*), uplynutí časového kvanta (*SoAlarmSensor*, *SoTimerSensor*), nečinnost aplikace (*SoIdleSensor)* a další. Do scény je přidán *SoOneShotSensor*. Prvním parametrem v konstruktoru je ukazatel na callback funkci, druhým jsou data jí předávaná (typ *void* *).

5.5.1 Callback funkce

Ve funkci *sensorCallback* zjistíme čas uplynulý od jejího minulého volání. Aktualizujeme zde i všechny grafy používané pro testování a měření simulace, jako například graf ukazující počet vyrenderovaných snímků za sekundu, nebo graf sledující zatížení sítě. Z po síti sdílených proměnných ovládacího menu získáme hodnoty definující změny nastavení urychlovače pro další krok simulace. Zavolá se funkce pro emitování další částice. Tato funkce s určitou pravděpodobností emituje novou částici. Provede se zavolání funkce pro výpočet nových pozic částic (tučně ve zdrojovém kódu). Pokud je zvolena metoda zobrazování částic jen podle jejich hustoty provede se výpočet 3D-textury podle pozic částic. Metodou *schedule()* opět zařadíme senzor do fronty (v

proměnné *sensor* je ukazatel na senzor, který funkci zavolal). Tím máme zajištěno, že bude dosaženo maximálního možného počtu vyrenderovaných snímků za sekundu.

5.5.2 Funkce pro výpočet pozic a rychlostí elektronů

Tvoří nejdůležitější část testovacího programu. Počítá vektorově podle rovnic uvedených v druhé kapitole síly působící na elektron. Z těchto sil potom vypočítává zrychlení jednak dostředivé, udržující elektrony v duantech na kruhových drahách, které způsobuje magnetická síla, a také zrychlení způsobované elektrickým polem mezi duanty tvořící přírůstek absolutní hodnoty rychlosti. Důležitým parametrem této funkce je delta času tj. čas co uplynul od posledního volání callback funkce. Jelikož rovnice použité pro výpočet jsou v diferenciální a numerický výpočet platí jen pro malé delta je nutno počítat pozici s výrazně kratším integračním krokem než je volána callback funkce a to řeší cyklus for s proměnlivou délkou, v němž se pozice počítá s integračním krokem padesátkrát až pětsetkrát kratším v závislosti na době, kterou trvá vyrenderování a výpočet jednoho snímku. Tyto rovnice jsou v podstatě řešeny jednoduchou Eulerovou metodou. Jedná se o jednokrokovou metodu, kde nový stav lze stanovit na základě stavu předcházejícího podle vzorce:

$$x_{n+1} = x_n + f(t_n, x_n) \qquad (12)$$

Další možností bylo použít rovněž jednokrokovou metodu Runge-Kutta čtvrtého řádu, kde se následující krok počítá z předcházejícího podle rovnic:

$$k_1 = f(t_n, x_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, x_n + k_1 * \frac{h}{2}\right)$$

$$k_2 = f\left(t_n + \frac{h}{2}, x_n + k_2 * \frac{h}{2}\right)$$

$$k_2 = f(t_n + h, x_n + k_3 * h)$$

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (13)$$

Toto řešení jsem nakonec nepoužil, protože nepřineslo očekávané zlepšení v porovnání s eulerovou metodou.

Nejlepší známou a nejpřesnější metodou výpočtu nové hodnoty numerického řešení diferenciálních rovnic je zkonstruovat Taylorovu řadu ve formě:

$$y_{n+1} = y_n + h * f(t_n, y_n) + \frac{h^2}{2!} * f^{[1]}(t_n, y_n) + ... + \frac{h^p}{p!} * f^{[p-1]}(t_n, y_n) \quad (14)$$

kde h je integrační krok. Hlavní myšlenkou Moderní metody Taylorovy řady [3] je automatické nastavení řádu metody. Což znamená použít takový počet členů Taylorovy řady, abychom dosáhly požadované přesnosti. Tento nový přístup je implementován v simulačním jazyce TKSL/386 a v TKSL/C. Knihovna pro jazyk C však ještě není k dispozici a podrobný princip metody zatím není zveřejněn. Používáním této metody by bylo možné řešit rozsáhlé soustavy diferenciálních rovnic v přijatelně krátkém čase.

Další důležitá věc, kterou funkce pro výpočet pozic elektronů řeší, je kolize se stěnami urychlovače. Tyto kolize jsou detekovány pomocí sady podmínek, které popisují prostor kam se částice může dostat a kam již nikoliv. Po kolizi se stěnou se částice vrátí do neaktivního stavu.

5.5.3 Vytvoření transakce

K vlastnímu vytvoření transakce dochází v metodě *setPos,* která je volána pro každou částici po ukončení výpočtu její pozice. Nejprve tuto transakci vytvoříme potom je potřeba do proměnné typu *SbIntLis* vložit cestu průchodu stromem od separátoru, ve kterém je provedeno napojení. V našem případě se jedná o *SoSeparator ballRoot.* Pro nalezení indexu kořene podstromu pro jednu částici použijeme metodu *findChild(ukazatel na kořen)*. Dále již víme na jakých indexech se nachází uzel *SoTranslation* jehož data potřebujeme sdílet. Potom do transakce doplníme prvky čtení a prvky zápisu (writeSet a readSet). Podrobněji v kapitole 3.2.2 Aktivní transakce. Na závěr se provede naplánování transakce metodou *schedule().* Ukázka tohoto zdrojového kódu.

```
SbTransaction *t = new SbTransaction;
SbIntList list;
list.append(ballRoot->findChild(myGraph));
list.append(0);
```

```
list.append(0);

trans->translation.transactionRead(SoField::LATEST, t, ballRoot, list, TRUE);
trans->translation.transactionWrite(pos, t, ballRoot, list);

t->schedule();
```

6 Ovládání a nastavení aplikace

U každé aplikace je důležité a by byla přívětivá k uživateli. Koncový uživatel totiž dá často přednost programu s povedeným uživatelským rozhraním před programem s lepší funkčností. I tento fakt bylo nutné tedy u simulačního programu cyklotronu zohlednit. Důležitým rozhodnutím bylo zda naimplementovat uživatelské rozhraní pomocí Open Inventoru, nebo využít externích prostředků hostitelské platformy. Vzhledem k tomu, že program má být přenositelný a měl by demonstrovat možnosti grafických knihoven je volba poměrně jednoznačná. Open Inventor však nedisponuje žádnými prostředky určenými pro tvorbu menu, jako jsou tlačítka nebo textová pole. Bylo tedy nutné všechny prvky naimplementovat ručně pomocí standardních grafických prvků.

Ovládání urychlovače je tvořeno pouze tlačítky, což je pro náš účel dostačující, protože je možné jimi nahradit téměř všechny ovládací prvky typu posuvník nebo textové pole.

6.1 Vytvoření menu

Menu je tvořeno nápisy třídy *SoText2*, která je schopna ze zadaného řetězce vytvořit jeho grafickou podobu, a to dokonce za použití libovolného fontu a velikosti zvoleného písma nastavovaného modifikační třídou *SoFont*. Na obrazovku jsou promítnuty za použití ortografické kamery. Díky přepočtu souřadnic se menu drží na jednom místě a nedochází k jeho zkreslení.

Zdrojový kód pro vytvoření menu s jedním tlačítkem:

```
SoSeparator * MakeMenu(SoWinExaminerViewer * viewer){
SoSeparator * MyMenu = new SoSeparator;

SoCallback *overlayCB = new SoCallback;
```

```
overlayCB->setCallback(overlayViewportCB);
MyMenu->addChild(overlayCB);

camera = new SoOrthographicCamera;
MyMenu->addChild(camera);

SoEventCallback * ecb = new SoEventCallback; //callback udalosti pro mys
ecb->addEventCallback(SoMouseButtonEvent::getClassTypeId(), event_cb, viewer);
MyMenu->addChild(ecb);

SoMaterial *material0 = new SoMaterial;
material0->diffuseColor.setValue(SbColor(0.6f, 0.6f, 0.6f));
MyMenu->addChild(material0);

SoFont *myFont = new SoFont;
 myFont->name.setValue("Arial");
 myFont->size.setValue(20);
 MyMenu->addChild(myFont);

SoText2 * b0 = new SoText2();
b0->string = "B++";

MyMenu->addChild(b0);
```

Tímto máme vytvořenou grafickou reprezentaci menu. Tomuto menu však chybí nejdůležitější vlastnost a to funkčnost. Je tedy potřeba přidat mechanizmus na zachytávání událostí od myši a mechanizmus, který rozpozná na jaký objekt ve scéně myš právě ukazuje.

6.2 Callback funkce

Na zachytávání událostí z periferních zařízení (myš,klávesnice) se používají třídy Open Inventoru. Třída *SoEventCallback* je k tomu, aby zachytila jakoukoliv událost. Metodě *SoEventCallback::addEventCallback()* předává jako první parametr typ události, na který má callback funkce *event_cb()* reagovat. Jméno callback funkce je uvedeno jako druhý parametr. V callback

funkci nejdříve zjistíme jestli bylo stisknuto levé tlačítko (pravé nepoužijeme). Pro určení jaký objekt byl ve scéně myší vybrán se používá třída *SoRayPickAction*. Z informací o události zjistíme souřadnice kliknutí myši a ty předáme prostřednictvím metody *SoRayPickAction::setPoint()* instanci třídy *SoRayPickAction*. Projdeme s pomocí metody *SoRayPickAction::apply()* celý graf scény a zjistíme vybraný bod v prostoru. Jestliže byl takový bod nalezen, stačí jen ověřit, ke kterému objektu ve scéně patří.

Zdrojový kód *callback* funkce:

```
static void  event_cb(void * ud, SoEventCallback * n)
{
const SoMouseButtonEvent * mbe = (SoMouseButtonEvent *)n->getEvent();
if (mbe->getButton() == SoMouseButtonEvent::BUTTON1 &&
mbe->getState() == SoButtonEvent::DOWN) {
SoWinExaminerViewer * viewer = (SoWinExaminerViewer *)ud;

SoRayPickAction rp(viewer->getViewportRegion());
rp.setPoint(mbe->getPosition());
rp.apply(viewer->getSceneManager()->getSceneGraph());
SoPickedPoint * point = rp.getPickedPoint();
if (point == NULL) {    return;    }
 SoNode * MyNode = point->getPath()->getTail(); //jaky objekt se vybral
 SoText2  * pickedBtn; //sem si ulozim vybrane tlacitko/elektron
 n->setHandled();
 if (MyNode->getTypeId() == SoText2::getClassTypeId()){ //je to tlacitko
            pickedBtn = (SoText2 *) MyNode;
} else return;
```

6.3 Sdílení hodnot v menu

Po identifikaci stisku tlačítka dojde k určení o jaké tlačítko se jedná, jestliže se toto tlačítko stará pouze o lokální parametry jednoho počítače, tak dojde k jejich modifikaci dle stisknutého tlačítka, například změna zobrazení. Ve chvíli, kdy se zmodifikuje proměnná, kterou je potřeba sdílet dojde

k lokální změně hodnot a k vytvoření nové transakce s požadovanými hodnotami. Provede se její naplánování. Na tomto příkladě je vidět provedení změn intenzity elektrického pole:

```
if (!strcmp(pickedBtn->string.getValues(0)->getString(), "E++")){
        if (E_max < 5){E_max+=0.5f;}
        string s_b6="E="+stringify(E_max);
        SbString s=s_b6.c_str();
        b6->string = s;

        SoText2 *node = b6;

        SbTransaction *t = new SbTransaction;  //vytvoření transakce
        SbIntList list;
        int nodeIndex = menu->findChild(node);
        assert(nodeIndex != -1 && "Wrong child index.");
        list.append(nodeIndex);
        node->string.transactionNotifyRead(SoField::LATEST, t, menu, list, TRUE);
        node->string.transactionWrite(0, 1, &s, t, menu, list);

        t->schedule();
}
```

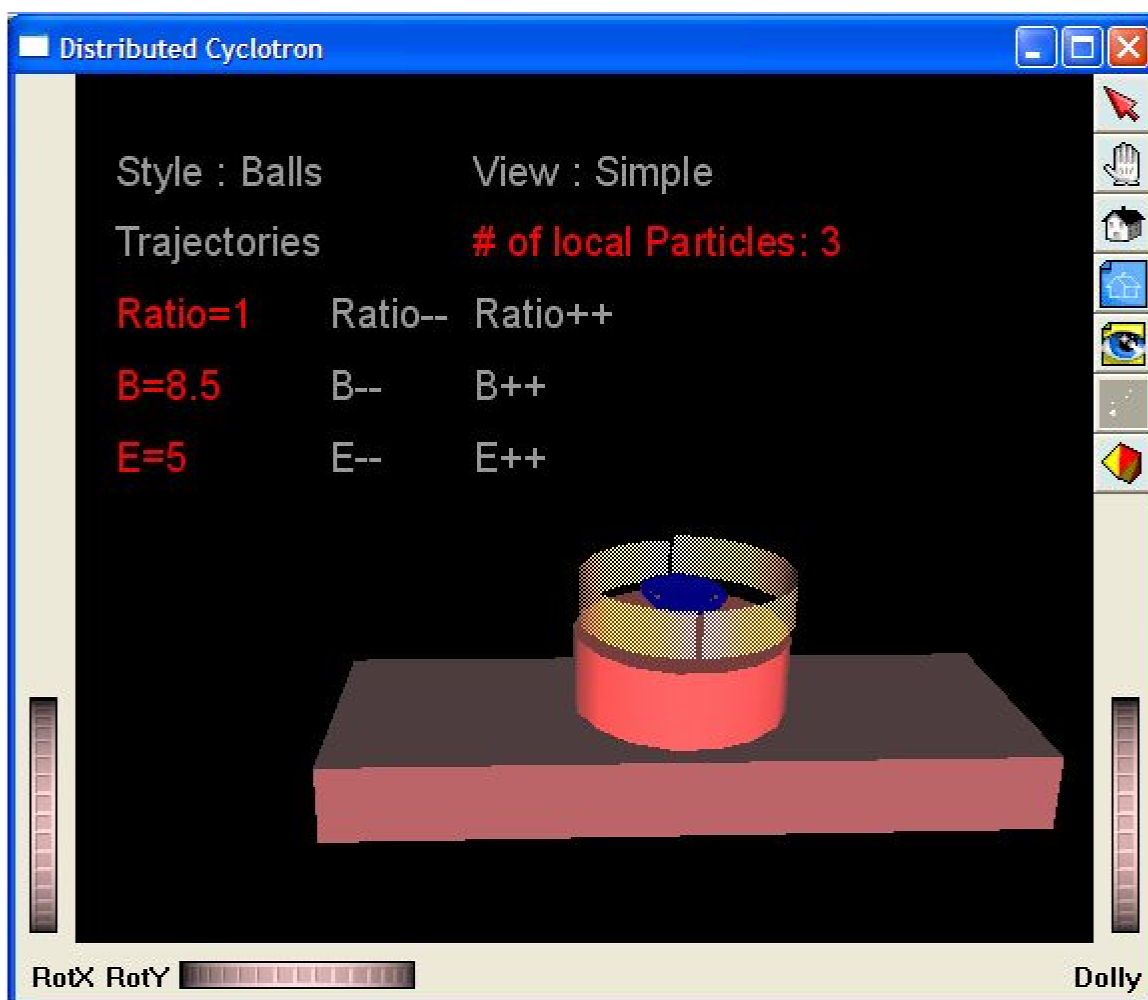
Po rozšíření těchto hodnot v distribuční skupině se při prvním volání callBack funkce pro animaci scény zajistí, že dojde k aktualizaci položek v menu na jednotlivých počítačích. V tuto chvíli se rovněž nastaví proměnné řídící simulaci na nové hodnoty. Získání požadované hodnoty ze zapouzdřené proměnné typu SoMFString uvádím zde na příkladu magnetické indukce B:

```
const char * str;
char * stopstr;
string pom_string;

pom_string=(b3->string.getValues(0))->getString();
str=pom_string.c_str();
B=(float)strtod(str+2,&stopstr);
```

6.4 Ovládání simulace

Tlačítka v menu jsou barevně odlišena a to tak, že tlačítka znázorněná červeně jsou jenom nápisy, se kterými není spojená žádná akce. Hodnoty pouze zobrazují. Zatímco při kliknutí na tlačítka znázorněná šedě je provedena nějaká akce.

Tlačítko *Style:Balls (3D-Texture, Nothing)* Je určeno pro změnu zobrazovacího stylu částic. Tlačítko *Wiew* odstraňuje ze scény část urychlovače popřípadě zachová jen částice, aby bylo možné lépe sledovat jejich trajektorie. Dalším tlačítkem je možno zapínat nebo vypínat zobrazení trajektorií částic. Ty je možno zobrazovat pouze při volbě stylu *Style:Balls.* Druhou kategorií tlačítek jsou ta co mění proměnné prostředí. Tyto hodnoty je potřeba sdílet v celé distribuční skupině. Mechanizmus je popsán v předcházející kapitole. Důležitá je hodnota ratio, která ukazuje shodu fáze elektrického pole a doby oběhu. K urychlení dochází nejlépe u ratio = 1 což znamená, že doba oběhu částice se rovná frekvenci elektrického pole. Zvětšením ratia se zvedá frekvence elektrického pole. B je magnetická indukce ta určuje poloměr kruhových drah částic a E je intenzita elektrického pole. Z elektrické intenzity je počítán přírůstek rychlosti částice. Zobrazení je řízeno i standardními prvky třídy *SoWinExaminerViewer* nebo *SoQtExaminerViewer*.



Obr. 12: Ovládací prvky scény s urychlovačem

6.5 Parametry příkazové řádky

Základní nastavení programu se provádí pomocí příkazové řádky. Jedná se zejména o číslo portu, na kterém má aplikace poslouchat. Pro první počítač (server) je nastaveno implicitně na hodnotu 1043, na tomto čísle portu čeká na připojení dalších počítačů do distribuční skupiny. Ostaní počítače si ho musí natavit parametrem *–l číslo portu.* To typicky volíme 1044,1045…Ještě předtím musí však jako parametr zadat IP adresu a listening port libovolného počítače, který už je do distribuční skupiny připojen. To vyplývá z toho, že počítače tvoří plně propojenou sít. Aby nebylo nutné dodržet pořadí připojování počítačů je vhodné volit adresu serveru a port 1043. Parametrem *–graphs* se povolí zobrazení všech grafů měřících jednotlivé hodnoty.

Dalšími parametry jsou rychlost simulace *–speed* , zpomalení simulace je jen převrácenou hodnotou zrychlení *--slow-down.* Parametrem *–particles* se nastaví počet částic implicitní hodnota jsou 3 částice. Parametrem *–delay* se nastavuje zpoždění sítě v milisekundách. Tyto parametry je nutné volit shodně pro všechny počítače v distribuční skupině.

Příklad spuštění pro server s IP adresou 192.168.1.100:
start.exe --delay 10 –particles 100

Příklad spuštění pro klienta1 s IP adresou 192.168.1.200:
start.exe 192.168.1.100:1043 -l 1044 --delay 10 –particles 100

Příklad spuštění pro klienta2 jsou již dvě možnosti:
start.exe 192.168.1.100:1043 -l 1045 --delay 10 –particles 100
nebo
start.exe 192.168.1.200:1044 -l 1045 --delay 10 –particles 100

7 Měření a testování

7.1 Na čem se testovalo

Pro testování byly použity čtyři počítače propojené standardní sítí LAN 100Mbps. Pro zjištění závislosti výkonu na počtu počítačů, respektive pro zjištění efektivity tohoto řešení bylo provedeno stejné měření i na třech, dvou a jednom počítači.

Konfigurace testovacích počítačů:

Hardware:

Intel(R) Celeron (R) CPU 2.66GHz

256MB RAM

Integrovaný grafický adaptér Intel(R) 82915G/GV/910GL Expres Chipset Family

Software:

OS Microsoft Windows XP Service Pack 2

Knihovny Coin verze 2.4.4

Knihovna SIMVoleon veze 2.0.1

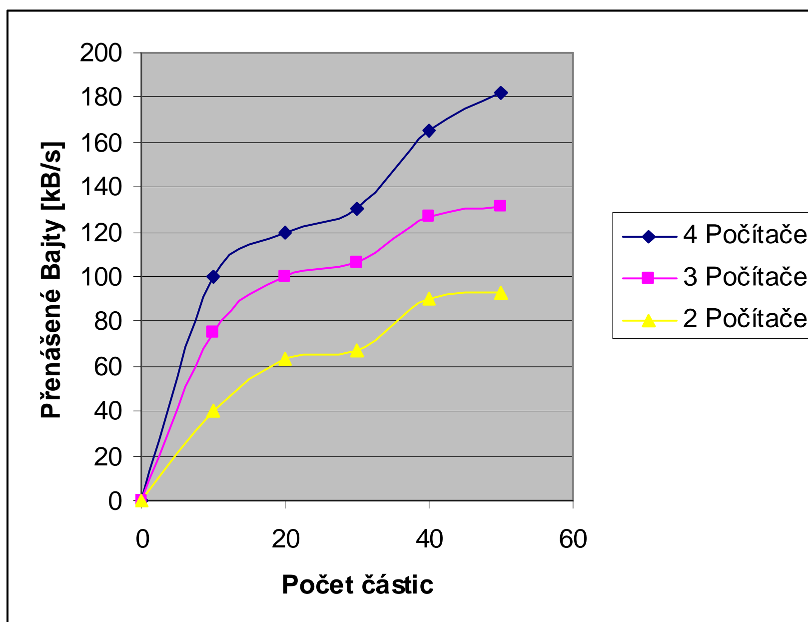
Knihovna CVE verze březen 2006

7.2 Množství přenášených dat

Množství přenášených dat zjistíme tak, že pomocí funkce *tnGet* zjískáme celkový počet odeslaných bajtů a jejím dalším voláním celkový počet přijatých bajtů ty sečteme a odečteme od této hodnoty sumu bajtů z minulého měření a podělíme časem, který uběhl od tohoto měření. Hodnotu určující počet přenášených bajtů za sekundu zobrazíme pomocí grafu *perfGraphNet.* Následuje ukázka kódu.

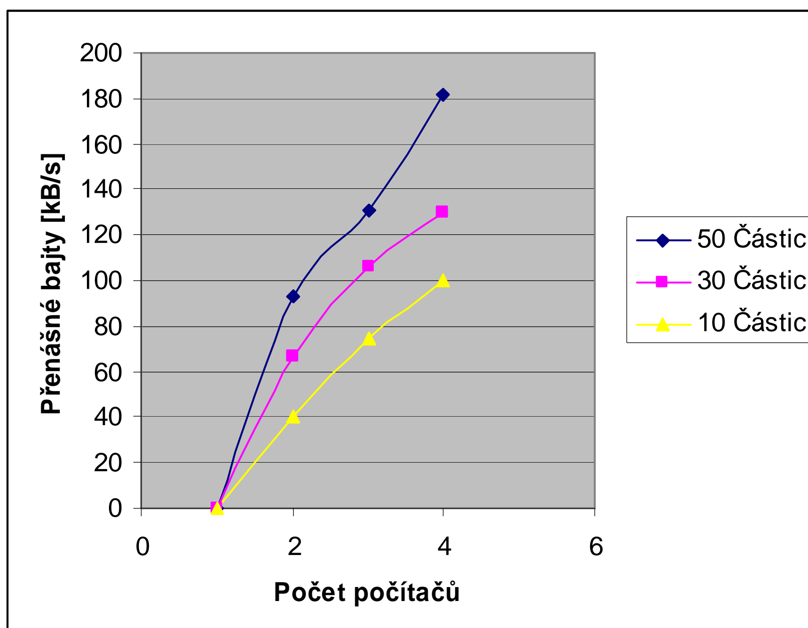
```
int newNetTotal = tnGet(TN_TOTAL_BYTES_SENT, 0)
+ tnGet(TN_TOTAL_BYTES_RECEIVED, 0);
perfGraphNet->appendValue(float(newNetTotal - lastNetTotal)/float(dt.getValue()));
lastNetTotal = newNetTotal;
```

Tato hodnota je přímo závislá na počtu vygenerovaných transakcí. Tudíž závisí na rychlosti s jakou jsme schopni počítat nové pozice elektronů na jednotlivých počítačích. Aby bylo zaručeno korektní měření, byly všechny testovací aplikace nastaveny do standardního zobrazení s vypnutým vykreslováním trajektorií. Měnil se pouze počet částic a počet počítačů v distribuční skupině.



Graf 1: Závislost množství přenášených dat na počtu částic

Tento graf lze interpolovat logaritmickou funkcí, což je dobré. Je to způsobeno tím, že počítače mají konstantní výkon, takže při rostoucím počtu částic jsou schopny jejich pozice počítat méně často a tudíž se generuje méně transakcí na jednu částici za sekundu. Zvlnění křivky v oblasti kolem 30 částic je způsobeno variabilní délkou integračního kroku výpočtu tj. v této oblasti dochází ke zmenšení integračního kroku.



Graf 2: Závislost množství přenášených dat na počtu počítačů v distribuční skupině

Tento graf ukazuje lineární závislost množství přenášených dat na počtu počítačů. Jedná se ovšem o sumu dat odeslanou a přijatou jedním počítačem. Pro celkový počet přenesených bajtů v síti

je potřeba rovnici této křivky ještě vynásobit počtem počítačů v distribuční skupině, takže celkové zatížení sítě vzhledem k počtu počítačů je kvadratické.

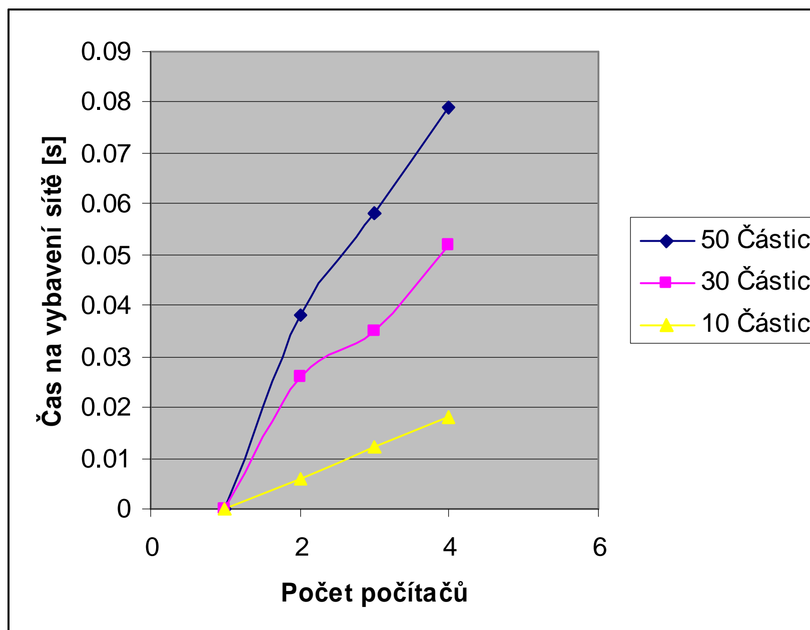
7.3 Výpočetní čas pro vybavení sítě

Jedná se vlastně o dobu trvání metody *SoDistributionGroup::Proces()*, která je volána na všech počítačích distribuční skupiny, vždy po přepočítání všech pozic částic na jednotlivých počítačích. Pokud nevrací hodnotu *RUNNIG* Ukončujeme aplikaci.

```
SoDistributionGroup *dg = SoNetwork::getDefaultDistributionGroup();
 if (dg->process() != SoDistributionGroup::RUNNING) {
   printf("SoDistributionGroup state changed to NOT_ACTIVE. Shutting down application.\n")
```

počítačů	50 částic	30 částic	10 částic
1	0	0	0
2	0.038	0.026	0.006
3	0.058	0.035	0.012
4	0.079	0.052	0.018

Tabulka 1: Čas potřebný pro vybavení sítě



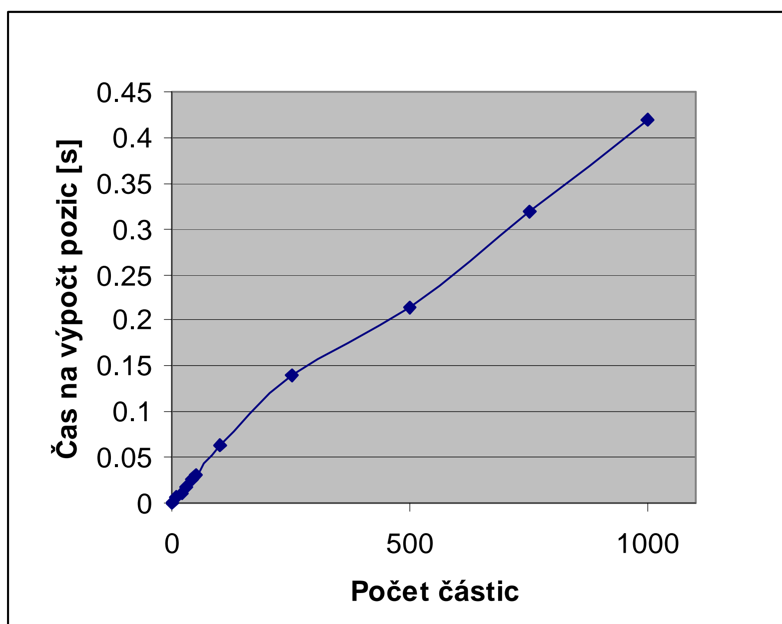
Graf 3: Čas potřebný pro vybavení sítě

Tímto grafem jsem si jen potvrdil to co, bylo řečeno Ing. Janem Pečivou o vlastnostech používané knihovny. A sice, že čas potřebný pro vybavení sítě je lineárně závislý na počtu počítačů v distribuční skupině.

7.4 Časová náročnost výpočtu pozic částic

Princip tohoto výpočtu je uveden v kapitole 5.2.2. Jeho časová složitost je závislá na požadované přesnosti řešení a na počtu částic. V tomto měření byla zjišťována celková doba potřebná k výpočtu nových pozic všech částic, což je doba potřebná k provedení tohoto kódu.

```
E = E_max*cos(speed*frekvence*(lastTime.getValue()));
for (int i=0; i<myBalls.getLength(); i++)
if (myBalls[i]->aktivni) myBalls[i]->timeTick(dt,E);
```

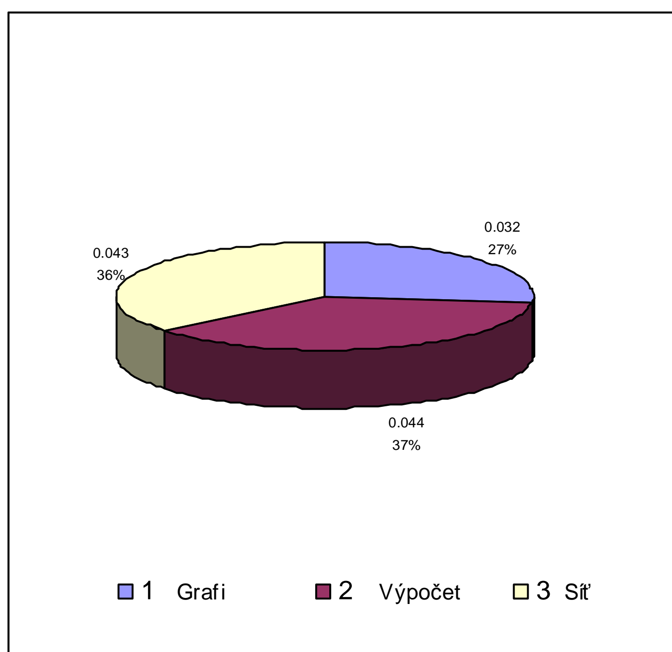


Graf 4: Výpočet pozic částic

Na tomto grafu je vidět, že v rozpětí od 30 do asi 250 částic dochází ještě k zmenšování integračního kroku, potom už integrační krok dosáhne minima a dochází ke snižování přesnosti. Toto rozpětí platí pro základní nastavení zobrazení. S vypnutým zobrazováním se posouvá směrem k vyšším počtům částic.

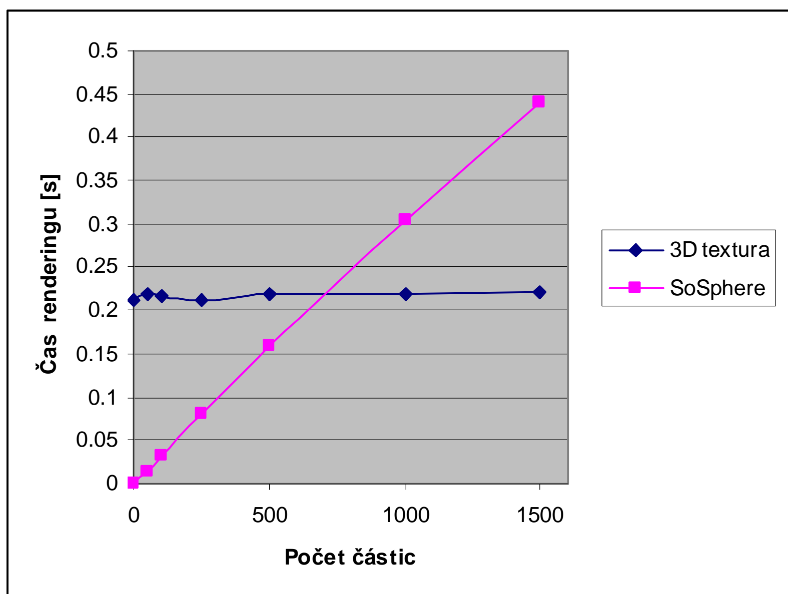
7.5 Rendering scény

Pomocí tohoto měření jsem chtěl zjistit jakou část řešení je možno paralelizovat. Jako další mě zajímalo jak moc je výhodné použití 3D textury vytvořené pomocí *SimVoleonu* pro zobrazování dynamických objemových dat. Respektive od jakého počtu částic je efektivnější než zobrazení pomocí *SoSphere* . Abychom zajistili lepší vypovídací hodnotu výsledů je třeba od času potřebného na vyrenderování celé scény odečíst čas, který zabere renderování pouze prázdné scény obsahující pouze grafu použité pro zobrazování hodnot. Ty nemusí být při praktickém použití zobrazeny a tudíž neubírají výkon.



Graf 5: Rozložení zátěže pro dva počítače a sto částic

Z tohoto grafu vyplývá, že parlelizovatelná část tvoří jen asi 37% výpočetních nároků, což by se mohlo jevit jako nedostatečné. Musíme však vzít v úvahu, že na ostatních počítačích je možno grafiku úplně vypnout a tím pádem se nám paralelizovatlná část na těchto počítačích zvětší na 50% kde zbylou část stojí vybavení sítě a jiné části aplikace. Další věcí, kterou je potřeba vzít v potaz je, že testovací počítače byly vybaveny pouze integrovanou grafickou kartou s velmi malým výkonem a bez podpory 3D textur. Při použití lepší grafické karty se čas potřebný na vyrenderování scény ještě výrazně sníží a paralelizovatelná část by pak dosáhla většího podílu. Z tohoto úhlu pohledu už je potom paralelní přístup významný a lze tak dosáhnou urychlení výpočtu.



Graf 6: Výhodnost 3D textury

Výhodnost použití 3D textur by se podle předpokladu měla projevit při vysokém počtu částic. Z grafu je evidentní, že konstantní náročnost 3D textur se začíná vyplácet již od 700 částic. Do této doby je z hlediska počtu vyrenderovaných snímků za sekundu (fps) vhodnější používat pro částice objekty třídy *SoSphere*.

7.6 Zpoždění změn datového modelu

Toto zpoždění lze chápat jako čas, který uplyne od vygenerování transakce na jednom počítači k jejímu přečtení na cílovém počítači v distribuční skupině. Lze změřit jako rozdíl mezi časovou známkou doručené transakce a nově vygenerovanou časovou známkou. Toto je možné, protože jednotlivé instance aplikace si mezi sebou udržují synchronizovaný čas, který se používá pro generování časových známek.

V praxi toto zpoždění odpovídá latenci sítě, která se může pohybovat od 100us na lokálních sítích do půl sekundy v internetu. V případě našeho testování byla latence menší než 1ms.

7.7 Jiné limitující faktory transakčního přístupu

Jedním z důležitých faktorů, které limitují použitelnost transakční přístupu pro distribuci výpočtu je to, že vyžadují plně propojenou síť počítačů. Což znamená to, že může každý počítač komunikovat

s každým. Toto řešení je pro paralelní architektury vhodné pouze pro malý počet počítačů. Většinou se uvádí 8 počítačů (procesorů) jako maximum, kdy je ještě toto řešení použitelné.

Z toho logicky vyplývá, že je potřeba soustředit velkou zátěž na malý počet počítačů. V našem případě tedy jednotlivé počítače počítají pozice velkému počtu částic. To ale přináší další problém ve formě obrovského počtu vygenerovaných transakcí za sekundu. Tento počet lze určit podle vzorce:

$$P_c = \frac{c}{T_r + T_n + T_c} \tag{15}$$

kde P je počet transakcí za sekundu, c je počet částic a T je celkový čas, který uplyne mezi jednotlivými voláními funkce pro výpočet nových pozic částic. Ten se spočítá jako součet času na rendering scény T_r, času na vybavení sítě T_n a času, který zabere samotná funkce pro výpočet pozic částic T_c.

Již pro padesát částic tedy dostaneme extrémně vysoké číslo:

$$P_{50} = \frac{50}{0.065 + 0.038 + 0.028} = \frac{50}{0.131}$$

$$P_{50} \cong 380\ transakcí/s$$

Tento počet je navíc generován každým počítačem v distribuční skupině. Abychom dostaly celkový počet transakcí není možné toto číslo pouze vynásobit počtem počítačů nebo částic, ale je třeba vzít v úvahu, že s jejich rostoucím počtem taky rostou časy T_r, T_n a T_c. Knihovny, které byly pro tvorbu diplomové práce k dispozici mají pravděpodobně nedostatečně velké zásobníky pro tyto transakce a tudíž může při velkém počtu částic dojí k jejich přetečení.

8 Závěr

Projekt, myslím, dosáhl svých cílů. Aplikace, která v rámci projektu vznikla je prezentací síly a možností knihovny Open Inventor a transakčního přístupu pro řešení sdílených grafických scén. Díky tomu, že aplikace byla koncipována tak, aby kladla vysoké nároky na knihovny umožňující sdílení grafických scén, tak pomohla odhalit slabiny a omezení tohoto přístupu. Některé z nich se podařilo odstranit již v průběhu řešení této diplomové práce. K nápravě dalších pravděpodobně dojde později.

Zároveň je tato aplikace prakticky použitelná. Rozhodně si nečinní ambice zařadit se mezi regulérní simulátory urychlení. Je však, myslím si, dostatečně názorná a přesná pro použití při demonstraci funkce kruhového urychlovače (například studentům středních škol).

Jsem si však vědom několika nedostatků, které by stály za vylepšení. Možným vylepšením je profesionálnější ztvárnění menu. Vytvoření některých dalších ovládacích prvků by jistě přineslo mnohem lepší ovládání než je stávající. Z hlediska možností zásahu do simulace je samozřejmě menu plně funkční. Každá další funkce už by byla nejspíš jen umělou a zbytečnou nadstavbou.

Cíle v případném pokračování projektu se poněkud rozcházejí dvěma směry. Jeden směřuje k tvorbě dokonalého simulátoru pohybu částic popsaných rozsáhlou soustavou diferenciálních rovnic. Toto pokračování by šlo pravděpodobně směrem k aplikaci moderních numerických metod jako například Moderní metody Taylorovy řady [3]. Případný paralelizmus by v tomto případě byl přímo na úrovni řešení soustavy diferenciálních rovnic.

Druhým možným směrem pokračování této práce je další spolupráce s Ing. Janem Pečivou na vývoji těchto knihoven a jejich aplikace v praxi. Ze zkušeností získaných tvorbou testovací aplikace vím, že použitelnost těchto knihoven je například ve hrách velmi dobrá a práce s nimi intuitivní.

Doufám, že práce neposlouží jen ke splnění diplomového projektu, ale že se uplatní i v praxi a dokumentace k aplikaci pomůže dalším programátorům v začátcích používání těchto knihoven.

Literatura

[1] Halliday, D. , Resnick, R., Walker, J.: Fyzika, Brno, Vutium, Prometheus 2000.

[2] The Inventor Mentor:Programming 3D Graphics with Open Inventor, Release 2

[3] Kunovský, J.: Modern Taylor Series Method. Habilitation theses. TU Brno, 1995.

[4] Pečiva, J.: Omnipresent Collaborative Virtual Environments for Open Inventor Applications. Springer LNCS, Volume 3814, 2005

[5] Pečiva, J.: Open Inventor, 2003. Dostupné z: URL http://root.cz/clanky/open-inventor (2005).

[6] Ullmann, V.: Jaderná a radiační fyzika, 2000. Dostupné z: URL http://astronuklfyzika.cz/JadRadFyzika5.htm (2005).

[7] Calvin, J., Dickens, A., Gaines, R., Metzger, P.,Miller, D., Owen, D.: The SIMNET Virtual World Architecture, Proc. of IEEE VRAIS'93, 1993.

[8] Kuhl, F., Weatherly, R., Dahmann, J. Creating Computer Simulation Systems, An Introduction to the High Level Architecture. Prentice Hall PTR, 2000.

[9] Pečiva, J.: Active Transaction Approach for Collaborative Virtual Environments, Proc. of VRCIA 2006 (to appear), Hong Kong, 2006.