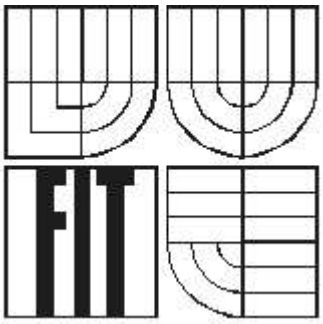
Vysoké učení technické v Brně
Fakulta informačních technologií



ROČNÍKOVÝ PROJEKT

Simulace vysoce náročného fyzikálního děje – urychlení elektronu pomocí vysokofrekvenčního lineárního urychlovače

Ondřej Peterka

BRNO 2005

Prohlášení:

Prohlašuji, že jsem předložený ročníkový projekt vypracoval samostatně za pomoci vedoucího diplomové práce, s použitím citované literatury, ostatních informačních zdrojů a výsledků vlastního šetření, které jsou uvedeny v této práci.

V Brně dne …… ….……...

Ondřej Peterka

Poděkování :

Považuji za svou milou povinnost poděkovat panu Ing. Janu Pečivovi, za odborné a organizační vedení a výbornou spolupráci při zpracování celého projektu.

ABSTRAKT

Jméno: Ondřej Peterka

Název ročníkového projektu: Simulace vysoce náročného fyzikálního děje

Abstrakt: Tato práce se zabývá simulací urychlení elektronu pomocí lineárního urychlovače. Pro implementaci byla použita nadstavbová knihovna nad openGL Open Inventor. Práce si klade za cíl vytvoření uživatelsky přívětivého programu, pomocí něhož se dá simulovat proces urychlení elektronu v lineárním urychlovači částic. Klade důraz na vizuální ztvárnění děje procesu urychlení a na jeho co možná největší názornost. Je možné si nechat vygenerovat lineární urychlovač libovolných parametrů a na něm pak provést simulaci urychlení. Aplikace je rozšířena o řadu prvků podporujících názornost fyzikálního dění uvnitř urychlovače. Vzhledem k tomu, že zde není zohledněna teorie relativity, je výsledná aplikace určená spíše široké veřejnosti než vědeckým kruhům.

Klíčová slova : lineární urychlovač, zrychlení elektronu, LINAC, Open Inventor

ABSTRACT

Name: Ondřej Peterka

Topic of Thesis: Simulation of a high-frequency linear accelerator

Abstract: This work deal with simulation of acceleration electron using linear accelerator. For implementation is used library Open Inventor. The object of this work is creating user-friendly application, which could be used for a simulation of LINAC. The main object is visual interpretation of electron's accelerating process and its plasticity. It can create LINAC according to requested parameters and undertake simulation using created LINAC. Application has several features, which can be used for better view and understanding of whole process. The work is not proposed for science usage, because it does not calculate with theory of relativity.

Key words : linear accelerator, LINAC, acceleration of electron, Open Inventor

OBSAH

SEZNAM POUŽITÝCH ZKRATEK

LINAC	lineární urychlovač (z angl. *Linear accelerator*)
GPL	General Public License
LGPL	Lesser General Public License

SEZNAM OBRÁZKŮ

1 Úvod

1.1 Implementační prostředky

Námětem práce je znázornění simulace urychlení elektronu pomocí vysokofrekvenčního lineárního urychlovače částic. Jedním z prvních kroků na cestě naplnění cílů tohoto projektu byla volba implementačního prostředí. Pro ztvárnění vizuální podstaty aplikace byla vybrána knihovna pro realtime grafiku jménem Open Inventor, což je nadstavbová knihovna nad známou openGL. Implementací Open Inventoru je celá řada, nicméně drtivá většina z nich není freewarového charakteru. Na druhou stranu ty implementace, které splňovaly podmínku freewarové licence, buď nejsou nadále vyvíjeny a jsou zastaralé anebo jednoduše nedosahují příliš vysoké kvality. Ale i zde se našla výjimka potvrzující pravidlo. Byla jí implementace s názvem Coin norské společnosti System In Motion. Ta je volně dostupná pod licencí GPL a je plně kompatibilní s Open Inventorem. Dřívější verze byla dokonce dostupná pod licencí LGPL. Jako implementační jazyk byl zvolen jazyk C++. Byl to vcelku přirozený výběr, jelikož pomocí C++ byla vytvořena i použitá knihovna od System In Motion.

1.2 Podstata lineárního urychlení

Dalším významným počinem, před samotnou implementací, bylo zjištění, jak vlastně urychlení částic obecně funguje, jaké jsou výhody a nevýhody jednotlivých typů a který z nich zvolit. Tomuto tématu se věnuje celá kapitola, popisující nejen typy urychlovačů a jejich specifika, ale i fyziku urychlení.

Jak se čtenář sám přesvědčí, už jen k sestavení nefunkčního modelu je nutno zapojit fyzikální a na ně navazující matematické znalosti. Další samostatné části textu se tak zabývají jednak problematikou modelovaní přístroje a jednak problematikou samotného urychlení jakožto fyzikálního děje. Zde se nevyhnu uvedení nezbytných vzorečků a jejich odvození, ale budu se snažit vše podat tak, aby látka byla prostá odborných termínu a pokud možno pochopitelná i pro úplného laika. Kromě vysvětlení urychlení, se zodpovím i otázku „Kde se částice vhodné pro urychlení vezmou a jak se emitují“. Těchto způsobů je samozřejmě celá řada, tento text se však bude zabývat pouze nejznámějšími. Poslední částí v souslednosti událostí, které se při urychlení částice odehrají, je kolize částice s jiným objektem. Jaké objekty to jsou a který byl použit v tomto projektu, se dozvíte v předposlední části kapitoly.

V závěru této kapitoly se pak zmíním i o praktických možnostech využití urychlovačů a to zejména ve světe vědy a medicíny.

Po přečtení této kapitoly by měl mít čtenář částečný přehled o urychlovačích , o tom proč se částice urychlují a hlavně jak se dosáhne zmiňované akcelerace, která je postavena na vcelku zajímavé myšlence. Také by si měl být vědom důsledků vzniku a použití urychlovačů v praxi . Každopádně by se měli vyvrátit možné první dojmy, že urychlovače jsou jen nákladnou hračkou k pobavení vědecké komunity.

1.3 Open Inventor pro vytvoření LINACu

Po nastudování nezbytné literatury o urychlovačích a jejich fyzice, jsem již mohl přistoupit k prvním krokům v návrhu programu. K tomu, abych mohl začít se samotnou implementací, bylo potřeba zjistit mnoho praktických vědomostí o grafice a následně konkrétních údajů o použité knihovně, tedy o Open Inventoru. Tato zpráva neobsahuje žádné obecné popisy použitých grafických algoritmů Open Inventoru. Zaměřil jsem se hlavně na mnou vytvořené postupy, které se týkají zejména dvou částí projektu. Za prvé to je vytvoření samotného modelu LINACu a za druhé modelování a simulace urychlení částice.
V první zmíněné části se rozepisuji o tom, jak byl model LINACu sestaven, jakých prostředků Open Inventoru jsem použil. Rovněž se zmíním o ztvárnění LINACu a jeho stylu. V podkapitolách podrobně popíši jednotlivé komponenty modelu LINACu. Nechybí samozřejmě ani zdůvodnění použitého postupu. Zmíním se také o použitém formátu souborů Open Inventoru sloužící k zobrazení modelů a jeho stručné charakteristiky.
V druhé části se budu věnovat, jak už jsem dříve naznačil, implementaci urychlení částice pomocí Open Inventoru. Samotný princip není nijak složitý, ale implementační a matematické pojetí je už zřejmě trochu více náročné. I zde si však čtenář bohatě vystačí se středoškolskou matematikou a fyzikou. Bude řeč i o implementaci znázornění kolize částice, která se objeví až na konci cesty částice.

1.4 Vytvoření GUI a interakce s uživatelem

Hlavní jádro aplikace, to jest simulace urychlení, bude tedy popsáno tak, jak se zmiňuje předcházející kapitola. Další, a to neméně důležitou částí aplikace, je kapitola zabývající se interakcí s uživatelem. Této problematice je věnována celá kapitola v tomto textu. V aplikaci je implementováno mnoho prvků usnadňujících komunikaci mezi programem a uživatelem. Každé skupině takových vlastnosti věnuji jednu podkapitolu, ve které se zaměřím nejen na to, jak byla implementována, ale i na vysvětlení její praktičnosti a použitelnosti. Vedle takových samozřejmostí jako je například vypisování rychlosti a energie částice nebo nastavení frekvence a amplitudy zdroje, uživatel jistě přivítá například možnost nahlédnutí do útrob urychlovače nebo nabídku kompletního návrhu tohoto přístroje a to vše pomocí pár poklikání myši.
Kapitola pojednávající o této stránce aplikace je rozdělena na tři stěžejní části. První částí je podkapitola zabývající se sestavením menu v Open Inventoru, další se bude zabývat prvky pro ovládání již probíhající simulace a poslední část se bude zabývat vytvořením nového modelu pomocí menu.

1.5 Co bude závěrem tohoto textu

V závěru písemné práce se rozepíši o možném praktickém využití výstupu tohoto projektu. Uvedu úvahu o tom zda byli původní cíle projektu naplněny, či nikoliv a prodiskutuji případné nedostatky. Rovněž věnuji odstavec možnému pokračování projektu v rámci diplomové práce .

2 Fyzika urychlovačů částic

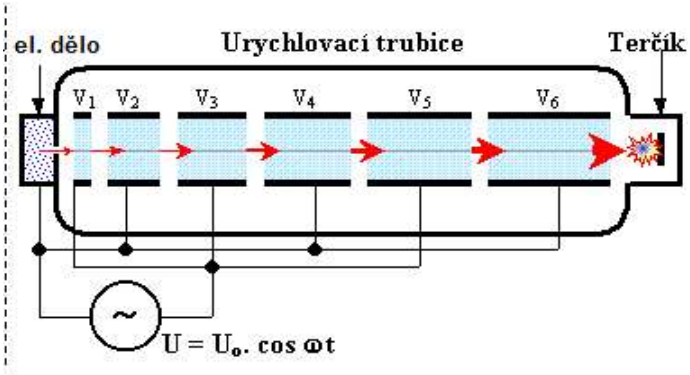
2.1 Obecně o urychlovačích

Jestliže se chceme zabývat studiem elementárních částic a zjistit tak něco o jejich struktuře a vlastnostech, musíme použít částice s vysokou kinetickou energií. Ty částice, které jsme schopni urychlit patří do skupiny tzv. *stabilních elektricky nabitých částic*. Jsou to známé elektrony, které mají záporný náboj a na druhé straně částice s kladným nábojem pozitrony, protony a deuterony. Tento výčet je samozřejmě neúplný, ale obsahuje přinejmenším všechny nejpoužívanější.
Zařízení, které má schopnost urychlit tyto částice, tedy zvýšit jejich kinetickou energii, se nazývají *urychlovače.* Urychlovač se skládá ze tří části. Jsou to: zařízení pro emitování částic, urychlovací systém a terčíku (či jiného objektu určeného ke kolizi). O všech těchto částech bude řeč později.
Urychlovačů částic je celá řada a obecně se dají rozdělit na dvě hlavní skupiny. Do první skupiny spadají urychlovací zařízení, které zapříčiní trajektorii urychlené částice ve tvaru přímky. Urychlovače, patřící do této skupiny nazýváme *lineární*. Do druhé skupiny patří urychlovače s kruhovou, či spirálovitou dráhou letu urychlené částice. Takovéto urychlovače se většinou nazývají *kruhové*. Každý urychlovač má jiné charakteristiky a je vhodný na něco jiného. Jeden z nejnázornějších, relativně jednoduchý ale netriviální, je vysokofrekvenční lineární urychlovač. Ten se také stal předmětem této práce a dále budu rozebírat pouze jeho vlastnosti a princip.

2.2 Jak získat částici vhodnou k urychlení

Ještě předtím než se dostanu k samotné fyzice urychlení, měl bych podat slíbenou odpověď na již položenou otázku, kde se vlastně částice, kterou chceme urychlit, vezme. Možností je opět celá řada. Nejjednodušší z nich je systém obsahující *ionizační trubici* v níž je přítomen plyn (např. vodík H), katoda a anoda. Při určitém napětí na katodě a anodě, které nepřesahuje několik stovek voltů, vznikají ionty (např. u již zmíněném vodíku by to byly protony). Ty se pak za pomocí odsávací elektrody dovedou až do urychlovacího zařízení. Pro urychlovače elektronů, což je případ této práce, stačí k emitování elektronů termoemise. Tedy žhavená katoda, která je vybavena usměrňujícími elektrodami. U tohoto způsobu emitování elektronů se můžeme setkat často s pojmem *elektronové dělo*. Na podobném principu funguje například monitor či televize.

2.3 Proces urychlení částice



Obrázek 1 :Schéma vysokofrekvenčního urychlovače částic.

Zcela vlevo na obrázku je znázorněno elektronové dělo spolu s komorou naplněnou plynem pro termoemisi elektronů. Na jednotlivé elektrody je přivedeno střídavé napětí. Je zde rovněž znázorněn níže popsaný princip sudých a lichých elektrod. Čtenář si může povšimnout postupně narůstající délky elektrod. Na konci urychlovací trubice je schématicky vykreslen terčík s právě probíhající kolizí.

Nyní se již můžeme začít věnovat samotnému urychlení částice. Jak jsem již předeslal, budu se zabývat pouze principem urychlení u vysokofrekvenčního lineárního urychlovače, jenž je použit v aplikaci.

Urychlovací systém u vysokofrekvenčního lineárního urychlovače, je tvořen prostou trubicí, která obsahuje válcové duté elektrody na něž je přivedeno vysokofrekvenční střídavé napětí, takovým způsobem, že na liché elektrody je přiveden jeden pól napětí a na sudé druhý pól napětí. Proč tomu tak je a jak to funguje, se dozví čtenář později. Napětí u velkých urychlovačů dosahuje až několik megavoltů a frekvence se pomocí dutinových rezonátorů může dostat na hranici několika gigahertzů. U těch nejmenších se napětí pohybuje okolo několika stovek kilovoltů. Dutým prostorem těchto elektrod částice procházejí setrvačností. Jakýkoliv vliv okolních polí, ať už magnetických či elektrických, je odstíněno (podobným principem na jakém funguje Faradayova klec).

K samotnému urychlení dochází tak, že nabité částice s nábojem *q* jsou akcelerovány pomocí elektrostatického pole, které je dané napětím na elektrodách mezi nimiž se částice nachází. K urychlení tak dochází v mezerách mezi jednotlivými elektrodami nikoliv uvnitř těchto elektrod! Částice, která dorazí k první elektrodě na níž je napětí U1 získá energii:

$$E = qu1 \quad \text{vzorec 2.3.1}$$

Jestliže dorazí k další elektrodě, kdy na ní bude správně orientované napětí U2, bude mít již energii :

$$E = q(u1 + u2) \quad \text{vzorec 2.3.2}$$

A tak dále.. . Celková konečná energie urychlené částice je přímo úměrná součtu napětí na jednotlivých elektrodách, kterými prošla. Tento fakt je popsán vzorcem:

$$E = q(u1 + u2 + \ldots + uN) \quad \text{vzorec 2.3.3}$$

Platí tedy že čím větší napětí a čím více elektrod tím je výsledná energie větší. Aby částice získala co největší energii ze strany napětí na elektrodě, musí se dostat do mezery mezi elektrodami, v době, kdy je na urychlující elektrodě amplituda se správnou orientací. Pro elektron, jakožto částici se záporným nábojem to bude kladná amplituda napětí. S výhodou tak můžeme využít faktu, že je napětí na elektrody rozvedeno, tak jak je popsáno výše. Tedy jeden pól napětí na lichých, druhý pól na sudých elektrodách.
Putující částice se tak může urychlit aniž by bylo nutné neustále napětí na elektrodách zvyšovat. Elektrostatické pole mezi elektrodami bude vždy stejné a největší možné, pokud se nám podaří vytvořit takový systém, který umožní částici dostat se do mezery v době, kdy na ni bude správně orientované maximální napětí (viz. výše). Musíme si ovšem uvědomit, že částice má značnou rychlost a nejen to, tato rychlost se zvětšuje. Závislost rychlosti částice na získané energii se dá získat následovně.

$$E = qU \qquad E = \frac{1}{2}mv^2 \qquad \text{vzorce 2.3.4}$$

Vezmeme v úvahu již zmíněný vzorec pro energii závislou na napětí a všeobecně známý vzorec pro kinetickou energii tělesa:

Prostým spojením těchto dvou vzorců získáme výraz určující rychlost částice:

$$v = \sqrt{\frac{2qU}{m}} \qquad \text{vzorec 2.3.5}$$

Změna rychlosti oproti rychlosti předcházejícího urychlení se dá vyjádřit vztahem:

$$v_x = \sqrt{\frac{x}{x-1}} * v_{x-1} \qquad \text{vzorec 2.3.6}$$

Z důvodu konstantní frekvence střídavého napětí, se musí zvětšovat délka elektrod ve směru dráhy letu elektronu. Tím bude zajištěna možnost maximálního urychlení částice. Jak se musí elektrody zvětšovat je odvozeno z vzorce předcházejícího a vzorce:

$$s = vt \qquad \text{vzorec 2.3.7}$$

Výsledný vzorec je pak tohoto tvaru:

$$l_x = \sqrt{\frac{x}{x-1}} * l_{x-1} \qquad \text{vzorec 2.3.8}$$

Jak je vidět už samotná konstrukce modelu LINACu není triviální záležitostí.
Tímto bych ukončil pojednání o procesu urychlení a zmínil některé další zajímavosti z principu urychlovačů z pohledu fyzikálního modelu.

Modelový příklad:
Uvažujme následující systém: délka první elektrody 10 metrů, zdroj, který má amplitudu 10 voltů a frekvenci 93773,66 Hz. Pak při prvním urychlení dosáhne elektron rychlosti 1875473,37311 metrů za sekundu. Při dalším maximálním urychlení to již bude 2652319,88013 metrů za sekundu. A tak dále... .

2.4 Co se s urychlenou částicí děje dál

Závěrečnou fází, jíž urychlovaná částice prochází, je kolize s nějakým objektem. V našem případě to je tzv. *vnitřní terčík*, ale může to být například i jiná částice s protichůdným směrem. Tak se dá mimochodem dosáhnout mnohem většího efektu srážky (systémy pracující na tomto principu se nazývají *collidery*). Vraťme se však k našemu případu použití vnitřního terčíku. Pojem „vnitřní“ má velmi jednoduché vysvětlení. Na rozdíl do vnějšího typu terčíků je prostě ukryt uvnitř urychlovacího systému, nikoliv mimo urychlovací systém.
Produktem srážky jsou částice, které pak bývají předmětem zkoumání. Jsou to například fotony, neutrony, mezony atd. Některé z nich ani jinak uměle získat nejde. Jejich produkce je také jeden z hlavních významů urychlovačů částic. Tyto částice jsou většinou vyvedeny do lapacích komor, kde jsou měřící přistroje.
Tím jak do terčíku částice narážejí se terčík značně zahřívat. Tak by mohlo dojít k deformaci nebo samotnému zničení materiálu terčíku. Proto je vybaven chladicí aparaturou.

2.5 Urychlovač - pouze hračka pro fyziky?

Ne, opravdu tomu tak není. Urychlených částic se využívá v celé řadě vědních oborů. Určitě každý již zaslechl pojem „elektronový mikroskop“. Princip tohoto přístroje je založen právě na urychlených elektronech, jejichž dostatečná energie je předurčuje k rozpoznání malých částic s malou vlnovou délkou ale malou velikostí. I ten nejlepší rastrový mikroskop selže už při zobrazení, z dnešního pohledu fyzika, velké částice atomu. Jeho uplatnění si jistě dovedeme představit nejen v molekulární biologii či chemii, ale i v praktické medicíně, tak jak ji známe.
Dalším možným využitím je například pomoc, při dokazování *velkého třesku*. Navozuje se totiž prostředí, které bylo přítomno velkému třesku a zkoumají se jeho možné důsledky. Tak by se dalo teoreticky tuto teorii vyvrátit nebo obohatit o další poznatky.
Dobře známé využití je i v medicíně. Jistě každý slyšel o využití přístrojů v boji s rakovinnými nádory. Princip léčení je takový, že rychle letící atom, který se zbaví nejlépe všech elektronů, se dostane skrz nepoškozenou tkáň lépe a s minimálním destruktivním účinkem nežli jakýkoliv skalpel. V místě nádoru, kde atom (z kterého po zbavení se elektronů zůstane pouze jádro-iont) ztrácí svoji energii, tím jak prochází hmotou, vyzáří svoji energii a ničí tak poškozenou tkáň.
Dalších využití je celá řada, uvedl jsem jen ty nejzajímavější ale i tak doufám, že se mě čtenáře aspoň částečně podařilo přesvědčit o praktickém využití urychlovačů částic.

3 Vytvořeni LINACu v Open Inventoru

Open Inventor klientskému programátorovi nabízí velmi mnoho podpůrných tříd. 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 válce, které můžeme pomocí dalšího objektu, sloužícímu k pohybu 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 válce a jeho umístění.

Společně s Open Inventorem je pod OS Windows hojně využívána knihovna *SoWin*, která je založená na stejném aplikačním modelu jako Open Inventor. Názorným příkladem je hned základní kámen jakékoliv nekonzolové aplikace a tou je okno aplikace. Pomocí propracované třídy `SoWinViever` 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`, který má těchto prvků ze všech možných potomků nejvíce.

3.1 Komponentový model LINACu

Za pomocí knihovny Open Inventor, je vytvořen celý model LINACu. Nebylo tedy potřeba sáhnout k menší úrovni abstrakce, například k přímému využití knihovny OpenGL. Celý model je vytvořen pomocí komponent definovaných pomocí popisných souborů. Struktura těchto souborů je vcelku jednoduchá a plně odráží princip Open Inventoru.
Jednotlivé části se dají prostým zakomentováním jednoho či dvou řádků v kódu odstranit a je tak možné zamezit zobrazení komponentě, která brání ve výhledu například při ladění grafické podoby aplikace (viz. Obrázek 2).
Model se skládá z části, které jsem fyzikálně popsal blíže v předcházející kapitole „2 *Fyzika urychlovačů částic“*. Jsou to systémy pro emitování elektronů, urychlovací systém a konzole s terčíkem. Lineární urychlovač je doplněn dále o soustavu nosných stojanů, jejichž hlavní část je opět načítána z *.iv souboru a doplněna dvojicí podpor. Ty jsou implementovány za pomocí tříd SoCylinder a SoSphere. Tedy skládají se z geometrických těles válce a koule. Systém emitování elektronů je spíše pro dekorativní učely, takže jeho složení není zas až tak důležité. Stejně je tomu i u konzole s terčíkem. Tedy jen pro úplnost pár poznámek.

3.1.1 Systém emitování elektronů - elektrické dělo a terčík

Systém emitování částic je tvořen vnějším a vnitřním pláštěm. Každý z těchto plášťů je definován v samostatném popisném *.iv souboru. Základní součástí systému je žhavící katoda, která je implementována pomocí třídy `SoCylinder`.
Část obsahující terčík je tvořena konzolou, vnitřním pláštěm a samotným terčíkem. Konzole je stejně jako vnitřní plášť opět definován pomocí samostatného *.iv souboru a nastavena pomocí uzlů (tříd) `SoScale` a `SoTransformation` na požadovanou pozici a tvar. Terčík je implementován opět pomocí třídy `SoCylinder.`

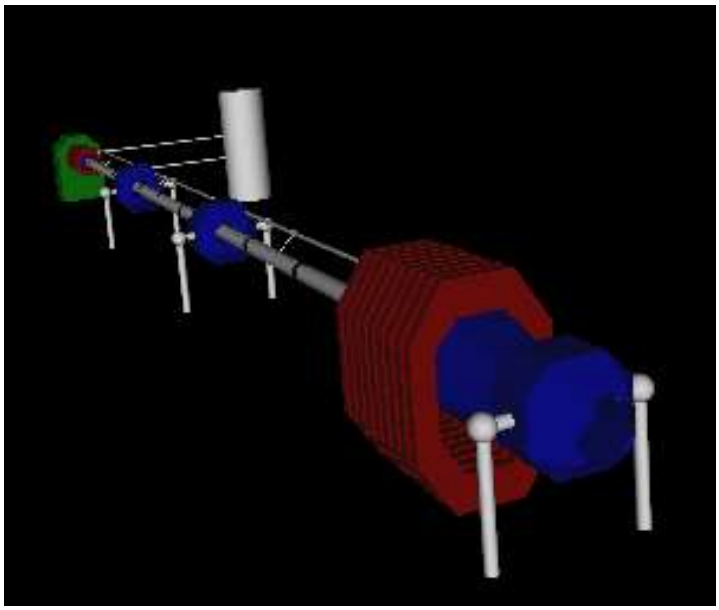
3.1.2 Urychlovací systém

Jádrem modelu je urychlovací trubice. Ta obsahuje na první pohled viditelný plášť, který je načítán ze stejného souboru jako vnitřní plášť konzoly s terčíkem . Opět pomocí tříd `SoScale` a `SoTransformation` upraven na požadovanou velikost, tvar, polohu a orientaci. Dále se urychlovací systém sestává ze zdroje napětí. Ze zdroje vychází vedení pro kladný a záporný pól. Vedení je pak napojeno na jednotlivé elektrody a to v souladu s ideou popsanou v kapitole „2 *Fyzika urychlovačů částic“*. Sudé elektrody na jeden pól a liché na druhý pól napětí. V závislosti na frekvenci se mění barva tohoto vedení, aby byl střídavý charakter proudu více názorný.
Elektrody jsou tvořeny pomocí dvojice válců (tedy opět `SoCylinder`) a stejně jako u některých předchozích využití třídy je ponechán pouze plášť. Délka elektrod je modifikována pomocí třídy `SoScale` algoritmem, o kterém bude řeč za malou chvíli. Mezera mezi elektrodami je volena dostatečně velká, aby se nejevil celý systém v přemrštěném měřítku jako monolitický objekt. I barva elektrod se mění v závislosti na frekvenci. Je tak jasnější, že napětí na elektrodě není konstantní.

3.1.3 Sestavení modelu

Rozhodoval jsem se mezi různými možnostmi jak vytvořit model LINACu. Nejdříve mě napadla možnost vytvoření jednoho vypracovaného modelu, na kterém by se prováděli veškeré simulace. Byla to možnost dávající větší prostor pro „uměleckou“ kreativitu. Není totiž tolik náročná jako druhá možnost a mohl bych se tak více věnovat vzhledu nebo fyzice urychlení. Zvolil jsem nicméně druhou uvažovanou možnost, kterou je možnost vytváření vzhledu a parametrů LINACu přímo v programu.
Aplikace je vystavena tak, aby si uživatel mohl sám nějaký vysokofrekvenční lineární urychlovač navrhnout a simulovat pomocí něho. Jak dosáhnout vytvoření nového modelu za pomocí implementovaného ovládaní aplikace bude diskutováno v kapitole *„4 Implementace interakce s uživatelem“*. Zde se pouze zaměřím na to, jak je docíleno jeho vypočítání ze zadaných parametrů. Co je potřeba, aby se mohl nový model vytvořit je délka první elektrody a celková maximální délka. Optimální frekvenci a napětí pro nejlepší využití urychlovače, si musí uživatel zjistit sám. Buď experimentálně nebo výpočtem.
Má-li program k dispozici délku první elektrody a celkovou maximální délku, dopočítá velikosti zbylých elektrod podle vzorce 2.3.8. Podle vypočtené délky se upraví rozměr elektrody pomoci třídy SoScale. Třídy SoRotation a SoTranslation se postarají o to, aby elektrody ležely tam kde mají, tj. v zákrytu za sebou v jedné řadě. Zadaná maximální délka se

většinou nenaplní celá protože poslední elektrodu nezkracuji, ale nechávám ji tak, jak by byla dlouhá v případě, že by za ní byla ještě následující. Zaokrouhluje se směrem dolů, protože se předpokládá že uživatel by již v praxi neměl rezervy na rozšíření lineárního urychlovače o dalších několik desítek metrů.
Podle takto vypočtené délky se vytvoří i již zmiňovaný plášť urychlovacího systému a vedení.



Obrázek 2 : Komponenty.

V pozadí je emitujicí systém (elektrické dělo), vpravo bílou barvou vyobrazený zdroj střídavého napětí, na něj napojené vedení spojené s jednotlivými elektrodami. V průběhu LINACu jsou modré stojany s podporami a v popředí je vidět konzole s terčíkem. Není zobrazen plášť.

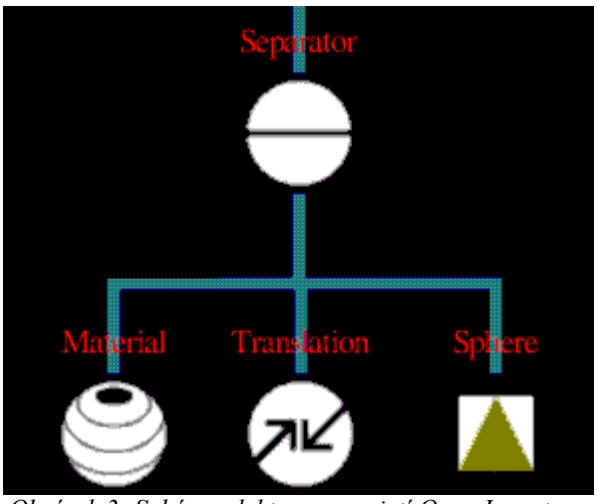
3.2 Simulace urychlení

Celá kapitola se opírá o poznatky uvedené v kapitole *„2 Fyzika urychlovačů částic“*. Jak se čtenář mohl v této kapitole přesvědčit, fyzikálně, matematicky pomocí vzorců není těžké dráhu elektronu popsat. Implementační detaily jsou věcí trošku jiné obtížnosti.
Open Inventor naštěstí poskytuje velmi dobrý mechanismus pro podporu pohybu a rozhodně nezapře své zaměření na realtime grafiku.

3.2.1 Schéma elektronu

Začněme, ale pěkně od začátku. Nejdříve ze všeho je potřeba vytvořit objekt, který bude reprezentovat elektron – částici, kterou budeme urychlovat. Jelikož máme k dispozici objektové programování, už samotná předchozí věta obsahuje obecnou definici schematu, který pro vyjádření elektronu budeme používat. Vytvoříme třídu `Elektron`, jejímiž instancemi budou

objekty reprezentující jednotlivé urychlované elektrony. Které základní členy bude třída mít? Určitě to bude rychlost, vektor pohybu, aktuální pozice. Už při samotném návrhu bylo jasné, že bude potřeba elektronů více a tedy že bude muset být navržena struktura, kde tyto elektrony uchovávat. Jako statický člen třídy bude struktura `Vector` ze standardní šablonové knihovny (*STL*) jazyka C++. V této struktuře se elektorny uchovávají. Jako argument šablony bude co jiného než třída `Elektron.`



Obrázek 3: Schéma elektronu v pojetí Open Inventoru

V grafu jsou patrné jednotlivé uzly z nichž se elektron skládá. Jsou instance tříd SoSeparator, SoMaterial, SoTranslation a SoSphere.

Vizuální stránku elektronu jako částice, bude mít na starosti Open Inventor. Nejzažitější podoba pro elektron je podle mého názoru kulička žluté barvy. Proč tedy nepoužít právě tuto grafickou reprezentaci. Pomocí dvou tříd Open Inventoru můžeme velmi jednoduše docílit právě tohoto efektu. Jsou to třídy `SoSphere`, vytvářející kouli o zadaném poloměru, a `SoMaterial`, kterou se dá nastavit nejen barva, ale i řada ostatních vlastností materiálu jako je například textura. Jistě bude potřeba mít k dispozici mechanismus pro určení polohy v prostoru. Pro tento účel je určena třída `SoTranslation`. Pomocí těchto tříd vytvoříme listy malého grafu kterému jako kořen dáme třídu `SoSeparator.` Ta nám zaručí, že žádný z použitých modifikačních objektů (instance tříd `SoTranslation` a `SoMaterial`), nebudou ovlivňovat žádné další objekty ve scéně. Schematicky by se dal tento graf vyjádřit jak je znázorněno na *Obrázku 3 - Schéma elektronu v pojetí Open Inventoru*. Na tyto modifikátory si každý elektron-objekt udržuje ukazatel. Každá změna stavu elektronu je tak zjednodušena na použití tohoto ukazatele.

3.2.2 Čas simulace

Pro každou simulaci je velmi důležité zachytit správně plynutí času a korektně s ním pracovat. Opne Inventor nabízí speciální třídu `SbTime`, která je, jak už z jejího názvu vyplývá, přímo určena pro práci s časem. Jednu její instanci jsem použil jako hlavní čas simulace. Tato

proměnná je nazvána **dt** a vyjadřuje časový rozdíl mezi jednotlivými voláními funkce pro pohyb částic.
První vlastností instance třídy `SbTime`, kterou jsem využil je fakt, že funguje jako kontejner pro čas vyjádřený v mikrosekundách. Na začátku porgramu je iniciována na nulu. Užitečnou vlastností je rovněž statická funkce `SbTime SbTime::getTimeOfDay()`, která vrací čas okamžiku jejího volání vyjádřený v mikrosekundách od času 00:00:00:00 dne 1. ledna 1970. Tato funkce je využita při jakémkoliv výpočtu pohybu částic právě pro zjištění časového rozdílu od posledního volání. Segment kódu, který má na starost správný výpočet tohoto časového úseku:

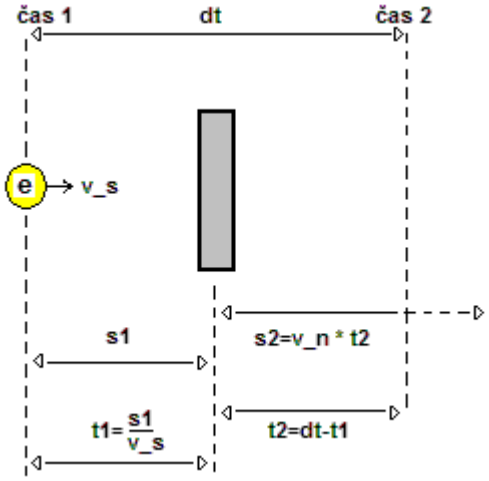
```
SbTime dt, ct = SbTime::getTimeOfDay();
if (lastTimeTick == SbTime::zero()) {
      lastTimeTick = ct;
      sensor->schedule();
      return;
}
dt = ct - lastTimeTick;
```

Proměnná **`dt`** již diskutována byla. Proměnná `ct` určuje čas, který je `v daném okamžiku volání. Vyjádřen je` v mikrosekundách od času 00:00:00:00 dne 1. ledna 1970.
Proměnná `lastTimeTick` je čas posledního volání tohoto kódu. Rozdíl mezi `ct`, tedy času který je, a `lastTimeTick`, času který byl při posledním volání, získáváme časový rozdíl `dt` používaný pro pohyb ve scéně.

3.2.3 Pohyb elektronu

Jak jsem zmínil v úvodu této kapitoly, Open Inventor skýtá mnohé prostředky pro vytvoření pohybu těles ve scéně. Těmito prostředky jsou třídy odvozené od bázové abstraktní třídy `SoSensor`. Má celou řadu potomků, kteří se používají u specifických problémů. V aplikaci byly použity pouze tři mutace třídy `SoSensor`. Jsou to `SoOneShotSensor`, `SoTimerSensor` a `SoFieldSensor`. O každém z těchto tří bude řeč, až se dostaneme k problému, u kterého byl použit. Na každý senzor je připojena tzv. callback funkce, která se provede jakmile je senzor sepnut.
`SoOneShotSensor` byl použit právě u simulace pohybu elektronu. Jeho poněkud složitý název je víceméně přesný a vystihuje jeho funkčnost. Tento senzor totiž funguje tak, že se naplánuje a pak spouští pokaždé, když je to jen trochu možné. Po každém spuštění se musí znovu říct, zda chceme, aby byl spuštěn i nadále. Musí se znovu naplánovat. Tímto způsobem se dá vytvořit nekonečný běh vyvolávání určitého segmentu kódu, tak často jak je to jen z technického hlediska možné. Pro vytvoření pseudoreálného pohybu je to ideální řešení.
V callback funkci se tedy v případě použití pro pohyb elektronu, spustí část programu, která se postará o výpočet příští pozice a rychlosti elektronu. K tomuto účelu byla vytvořena členská statická funkce třídy Elektron nazvaná `Elektron::updatePositions()`. Ta prochází vektorem všech elektronů a u každého zjistí, zda by za čas `t`, který uběhl mezi posledním voláním této funkce a aktuálním voláním, prošel místem urychlení či nikoliv. K tomuto zjištění je využita pozice jednotlivých elektrod. Ty jsou uloženy ve vektoru `VecBoost`, který se tvořil zároveň s vytvořením modelu lineárního urychlovače.

V případě že ne, je další postup jednoduchý. Ze zaznamenané rychlosti a pozice v objektu elektronu se vypočítá další pozice. Rychlost zůstává stejná, protože jak si čtenář snad vzpomene, ke změně rychlosti dochází jen mezi elektrodami a nikoliv uvnitř nich.
Jestliže se však zjistí, že elektron v průběhu dráhy, která by byla uražena za dobu t, projde elektrostatickým polem mezi elektrodami, musí se provést komplikovanější propočet. Nejlépe asi pro vysvětlení poslouží obrázek.



Obrázek 4 : Schema elektronu při urychlení

Šedý obdélník znázorňuje místo urychlení. Změna délky dráhy je znázorněna čárkovanou čárou u vzorce pro tuto změnu, dráhu s2.

Elektron má jistou počáteční rychlost, kterou pro jednoduchost nazvu **v_s.** Nebýt urychlení urazil by touto rychlostí za dobu **dt** dráhu **s**. Jelikož však ke změně rychlosti dojde někde v průběhu této potencionální dráhy musí být přepočítána. Dalším údajem, který je pro nás důležitý je dráha **s1**, což je vzdálenost mezi pozicí elektronu, na které se nachází na začátku přepočtu, a mezi místem, kde dojde k urychlení či zpomalení (v závislosti na napětí). Prostým odečtením jsme schopni vzdálenost **s1** získat. Posledním údajem, který známe a který je nezbytný pro zjištění další pozice elektronu je nová rychlost po urychlení **v_n**. Na základě rychlosti **v_s** a **s1** zjistíme čas, za jaký se elektron dostane na místo, kde dojde ke změně. Tento čas je na obrázku nazván **t1**.
Nyní se dostává do stavu, kdy se projeví změna rychlosti. Nejdřív si zjistíme čas, zbývající z celkové kapacity času **dt**. Opět pouhým odečtením časového údaje **t1** od **dt** dostáváme čas **t2**. Posledním krokem, ve kterém již zjistíme údaj který potřebujeme, je vynásobení nové rychlosti **v_n** a právě zjištěného času **t2**. Tím jsme získali délku dráhy **s2**, která nám určuje posun od posledního přepočtu polohy uvažovaného elektronu.
Poslední otázkou v pohybu elektronu zůstává problematika průběhu napětí na zdroji. Tomuto tématu je věnován následující odstavec.

3.2.3 Průběh napětí na zdroji a jeho vliv na změnu rychlosti elektronu

Zda se elektron zpomalí nebo zrychlí je závislé na orientaci napětí v místě urychlení. To je dále závislé na průběhu napětí na zdroji. Ve funkci `Elektron::updatePositions()` je zaveden globálně udržovaný čas, podle kterého se zdroj řídí. Pro časové údaje nám Open Inventor nabízí již zmiňovanou třídu `SbTime`. Její instance je také použitá pro uložení, inkrementaci a kontrolu času zdroje. Tato proměnná je pojmenována `VoltTime`. V každém volání funkce `Elektron::updatePositions()` je `VoltTime` navýšen o celou hodnotu **dt**, aby se udržela iluze přirozeného plynutí času. Zavádím tak nový čas, jehož průběh mám plně pod kontrolou, což je právě pro výpočet změny rychlosti elektronu v obecně různě dlouhém časovém úseku, nutné. Zjištění průběhu napětí a velikost napětí v okamžiku, kdy tento údaj ovlivní další vývoj putovaní elektronu, se tak stává méně komplikovaný. K času řídícího průběhu napětí z minulého volání `Elektron::updatePositions()` přičtu již diskutovaný čas **s1**. Tím dostávám čas zdroje v okamžiku urychlení a jsem z něj schopen odvodit napětí vyskytující se v místě změny rychlosti. Údaj už jen upravím potřebným znaménkem podle toho, zda je elektroda lichá či sudá.

3.3 Srážka s terčíkem

Pokud se elektron dostane na pozici terčíku dochází ke kolizi. Ta je v programu znázorněna tím, že elektron zmizí ze scény a na jeho místě se vytvoří shluk barevně odlišných částic s menším poloměrem než měl elektron.

3.3.1 Odstranění elektronu ze scény

Už jen samotné odstranění elektronu ze scény není triviální záležitostí. Nestačí totiž za pomocí příkazu C++ objektem alokovanou paměť uvolnit. Před tím, než se dostaneme k tomuto kroku, musíme odstranit odkaz na tento objekt z vektoru elektronů a odstranit jeho reprezentaci v grafu Open Inventoru. Ukazatel na elektron, u kterého se v metodě `Elektron::updatePositions()` zjistí, že má byt zrušen, se uloží do jiného vektoru nazvaného `vOut` a zároveň je vymazán z vektoru elektronů `elektrons`. Vektor `vOut` se prochází po volání `Elektron::updatePositions()` a pro každý uložený ukazatel se provedou dva kroky. Zjistí se pozice podle které určíme, zda má být vyvolána animace srážky s terčíkem, či elektron terčík nezasáhl. A vymaže se odkaz z grafu v rodičovi elektronu.
Celá věc je v průchodu vektorem `vOut` implementována následovně:

```
//vytvoření roztříštění elektronu na menší částice
float cur_pos[3];
(*ci)->getPosition(cur_pos);
if ((*ci)->v_z > 0 && !(bReset|| bRedraw))
      //narazil zřejmě do terčíku-vyvolá se animace
   createBurst(r,cur_pos[2],(*ci)->v_z);

SoGroup * parent = getParent((*ci)->Croot,r);
      //zrušení odkazu na elektron – tím se stává objekt
      //nereferencovaný a ruší se
if (parent!=NULL) {
       parent->removeChild((*ci)->Croot);
}
```

Po skončení průchodu vektorem se celý `vOut` vyčistí pro další použití voláním metody `vOut.clear()`.

3.3.2 Vytvoření animace srážky

Animace je vytvořena podobně jako animace pohybu elektronu. Pro částice vznikající srážkou je navržena třída `Micro`, která dědí po třídě Elektron. Při srážce se vytvoří několik instancí třídy `Micro`, které se uloží do vektoru `micros`. Každé z nich je již v konstruktoru přiřazen směr,rychlost a délka trajektorie. Hraje zde svoji roli i náhodná veličina, takže je při výpočtu použito funkce `rand()`. Kromě funkce `rand()` ve vzorci figuruje i rychlost elektronu, který srážku způsobil. Vektor `micros` se pak postupně prochází ve funkcí `Micro::updatePositionsMicro()`, která je, příbuzná funkci pohybu elektronu.

3.4 Animace průběhu napětí

Aby byl názornější střídavý charakter zdroje napětí, vybavil jsem jednotlivé elektrody i vedení senzorem, který mění jejich barvu v závislosti na frekvenci. Tento senzor je instancí třídy `SoTimerSensor`, u které se dá nastavit interval i začátek spouštění callback funkce. V callback funkci se prochází vektor ukazatelů na materiály jednotlivých elektrod a vedení, a v souladu s myšlenkou lichých a sudých elektrod se jim mění barva. Vektor samotný je získán již při tvorbě lineárního urychlovače.

4 Implementace interakce s uživatelem

Cílem každé aplikace je být co nejpřívětivější k uživateli. Podle tohoto hlediska se často koncový uživatel rozhoduje a mnohdy upřednostní program s hezkým GUI a ovládáním namísto programu s více možnostmi použití. I tento fakt byl při zpracování programu pro simulaci vysokofrekvenčního lineárního urychlovače zohledněn.
Prvním krokem tak bylo rozhodnout, zda bude program disponovat GUI implementovaným pomocí Open Inventoru, nebo se vytvoří GUI za pomocí externích prostředků hostitelské platformy. Jestliže si uvědomíme, že projekt má být z části reprezentací grafických možností Open Inventoru, byla volba jasná. Bohužel Open Inventor neposkytuje žádné prostředky přímo určené pro tvorbu menu. Neobsahuje žádná tlačítka, posuvníky či textová pole. Musel jsem tedy veškeré ovládání sestavit za pomocí standardních grafických metod a prvků.
Program typu simulace LINACu musí obsahovat dosti obsáhlé ovládání, které umožní nejen spuštění a pozastavení simulace, ale i vytvoření nového urychlovače podle pomocí menu zadaných parametrů. Ovládací prvky se tak dají rozdělit do dvou základních skupin. První skupina bude obsahovat ty prvky, které ovlivňují simulaci ihned, bezprostředně po jejich aktivování. Druhá skupina obsahuje prvky, potřebné k návrhu LINACu. Každé této množině bude věnována podkapitola. Začněme však samotným principem vytvoření menu.

4.1 Vytvoření menu a jeho ovládání

Omezil jsem se pouze na základní ovládací prvky menu, tedy na tlačítka. Pomocí nich se dá vytvořit obstojné ovládání pro jakýkoliv program a nahradit jimi všechny sekundární ovládací prvky typu posuvník či textového pole.

4.1.1 Znázornění menu pomocí Open Inventoru

Celé menu je do scény promítnuto za pomocí ortografické kamery. Ortografická kamera nám umožní, aby se menu nacházelo neustále ve výhledu uživatele (pomocí přepočtu souřadnic dokonce i na stejném místě). Celé menu je tvořeno needitovatelnými textovými poli, která reprezentují chybějící tlačítka s nápisem. K tomuto účelu jsem použil třídu `SoText2`, která je schopna ze zadaného řetězce vytvořit jeho grafickou podobu, a to dokonce za použití libovolného fontu a velikosti zvoleného písma (modifikační třída `SoFont`). Následující segment kódu znázorňuje vytvoření tlačítka „Stop animation“ za pomocí třídy `SoText2`:

```
//stop
SoTranslation * transBtn2 = new SoTranslation; //posunutí
transBtn2->translation.setValue(0.0,moveMenu,0.0);
MyMenu->addChild(transBtn2);                   //přidání do grafu menu

btnStopStart = new SoText2();
btnStopStart->string = "Stop animation";       //text tlačítka
MyMenu->addChild(btnStopStart);                //přidání do grafu menu
```

Volbu velikosti písma a fontu obstarává instance třídy `SoFont`:

```
SoFont *myFont = new SoFont;
      myFont->name.setValue("Helvetica"); //typ fontu
myFont->size.setValue(10);          //velikost
      MyMenu->addChild(myFont);           //přidání do grafu menu
```

4.1.2 Výběr položky menu

Menu je tedy vytvořené. Co mu však chybí je funkčnost. K tomu, aby bylo možné vybírat jednotlivá tlačítka a ty pak mohla vyvolávat nějakou odezvu byl použit jednak mechanismus, který je schopen zachytit událost poklikaní myši a jednak mechanismus, který je schopen na základě místa klinutí zjistit, jaký objekt ve scéně pod tímto místem leží (tedy který objekt byl myší vybrán).
Na zachytávání událostí z periferních zařízení typu myš nebo klávesnice dobře posloužily třídy Open Inventoru. Třída `SoEventCallback` je využita k tomu, aby zachytila jakoukoliv událost vyvolanou zvenčí. Protože nás však zajímají pouze události vyvolané poklikáním myši trochu funkčnost třídy `SoEventCallback` pozměníme. I na tuto možnost autoři Open Inventoru mysleli. Celé vytvoření mechanismu pro zachytávání pokliku myší je zachyceno v následujícím kódu:

```
SoEventCallback * ecb = new SoEventCallback;
ecb->addEventCallback(
        SoMouseButtonEvent::getClassTypeId(), event_cb, viewer
                    );
MyMenu->addChild(ecb);
```

Metodě `SoEventCallback::addEventCallback()` se předává jako první parametr typ události, na který má callback funkce `event_cb()` , jejíž jméno je uvedeno jako druhý parametr, reagovat. Poslední argument je volitelný a používá se pro strukturu dat, která je předána jako argument callback funkci.
V callback funkci nejdříve zjistíme, zda to, co událost myši vyvolalo, bylo stisknutí levého tlačítka (pravé tlačítko ignorujeme). Pro zjištění jaký objekt byl ve scéně prostřednictvím myši vybrán se používá třída `SoRayPickAction`. Z informací o události vyextrahujeme 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 se nám už jen zeptat, ke kterému objektu ve scéně patří. Celý výše popsaný proces je implementován následovně:

```
//zjistíme informace o údalosti
const SoMouseButtonEvent * mbe = (SoMouseButtonEvent *)n->getEvent();
//zjistíme zda se jedná o akci vyvolanou stisknutím levého tlačítka
if (mbe->getButton() == SoMouseButtonEvent::BUTTON1 &&
    mbe->getState() == SoButtonEvent::DOWN) {

    //kořen grafu
  SoWinExaminerViewer * viewer = (SoWinExaminerViewer *)ud;
```

```
    //zjistíme vybraný bod v prostoru z bodu kliknutí
  SoRayPickAction rp(viewer->getViewportRegion());
  rp.setPoint(mbe->getPosition()); //nastavení bodu kliknutí
  rp.apply(viewer->getSceneManager()->getSceneGraph()); //projdeme graf

  SoPickedPoint * point = rp.getPickedPoint(); //vyvoláme nalezený bod
  if (point == NULL) { //nebyl vybrán žádný objekt
    (void)fprintf(stderr, "\n** miss! **\n\n");
    return;
  } //end if point
  SoNode * MyNode = point->getPath()->getTail(); //konkrétní objekt
  ...
```

Nyní když víme, který objekt byl ve scéně vybrán, stačí se zeptat jestli jde o tlačítko, tedy konkrétně o instanci třídy SoText2. Toho docílíme takto:

```
          //porovnání id třídy SoText2 a třídy proměnné MyNode
    if (MyNode->getTypeId() == SoText2::getClassTypeId())
          pickedBtn = (SoText2 *) MyNode; //bezpečné přetypování
    else return;
```

Posledním krokem zůstává rozlišení mezi jednotlivými tlačítky. Tento problém je řešen, tak že ověříme nápis tlačítka.

```
//srovnání řetězce instance SoText2 s textem „Stop animation“
if (!strcmp(pickedBtn->string.getValues(0)->getString(), "Stop animation")){
          handleStopStart(true);
          btnStopStart->string = "Start animation";
          return;
}
```

Jestliže projde podmínkou srovnání textu, je provedena odpovídající akce a callback funkce je opuštěna.

4.2 Prvky pro ovládání simulace

Pod pojmem ovládacích prvků simulace rozumím akce, které vyvolají změnu v simulaci. Dobrým příkladem je už zmiňovaná volba pro zastavení a opětovné rozjetí simulace, nebo pro emitování nového elektronu. Ke každému uvedu pár zajímavých implementačních detailů. Jednotlivé prvky jsou probírány postupně - tak jak jsou v menu uvedeny.

4.2.1 InfoLine

Tento prvek je tvořen dvojicí Instancí třídy `SoText2`. Jedna slouží jako nadpis a druhá, která je na začátku inicializovaná na prázdný řetězec je určena jako stavová řádka. Na této řádce se objevují informace týkající se provedených akcí. Její změna na patřičný text je vyvolávána z různých částí kódu podle potřeby. Není to tedy ovládací prvek v pravém slova smyslu. Má podobnou funkci jako štítek známý z ostatních GUI. Nedá se přímo použít pro vyvolání změny stavu simulace.

4.2.2 Fire electron

Tento ovládací prvek, zajišťuje pouze jediné. Do simulace vpustí elektron (vyvolá jeho emitaci). V programu je pro tento účel vytvořena funkce `Create1Elektron()`, která vytváří novou instanci třídy elektron. Argumenty konstruktoru elektronu jsou jeho pozice, barva, rychlost atd.
Nadále se s tímto elektronem nakládá stejně, jako by byl vytvořen na začátku programu.

4.2.3 Stop/Start animation

Touto volbou docílíme kompletního rozjetí či zastavení simulace. V té části callback funkce, která má na starost zajištění reakce na tuto volbu, se nastavuje příznak `stopFlag`, který ostatní části programu používají k indikaci stavu animace. Dalším aktem, který se v reakci na zvolení tohoto tlačítka odehraje, je nastavení, lépe řečeno pozastavení, senzorů. Jsou to ony dva senzory zmíněné v kapitole „*3 Vytvořeni LINACu v Open Inventoru"*. Senzor řídící urychlení i senzor řídící animaci napětí se odplánuje *(unschedule)* . Tím je zaručeno ukončení simulace, tedy toho, že se nevolají funkce pro pohyb elektronů a částic z produkce srážky terčíku a elektronu. Zrovna tak se pozastaví animace změny napětí na elektrodách, které se jinak projevuje jako blikání elektrod. Vše co jsem zde uvedl samozřejmě platí i obráceně, tedy pro opětovné spuštění animace.
Na druhou stranu je však při pozastavení animace uživateli pořád umožněno scénou procházet, libovolně si ji natáčet nebo dokonce dopředu naplánovat nějakou akci (.například vypuštění nového elektronu). Tohoto výsledku bylo přesně potřeba dosáhnout.
Text ovládacího prvku se mění v závislosti na tom, jestli je animace pozastavena nebo není. Jestliže ano, jistě nikomu nebude chybět volba pro znovu pozastavení. Proto je při pozastavené animaci nastaven text na „Start animation" a opětovnou volbou se animace znovu rozběhne a u probíhající animace je text roven řetězci „Stop animation". Tento princip změny textu je aplikován i na dalších ovládacích prvcích, které budu zmiňovat. Znovu se už o nich nebudu nijak rozepisovat.

4.2.4 Reset

Nejsem si jist, jestli není pojmenování tohoto prvku zavádějcí, ale snad bude vše jasnější po přečtení tohoto odstavce.
Volba reset, stejně jako Fire electron vytvoří nový elektron. Avšak na rozdíl od volby Fire elektron vyresetuje všechny senzory, napětí a vymaže všechny již simulované procesy. Jednoduše řečeno ukončí předchozí simulaci a začne zcela novou. Tyto všechny akce vyvolává funkce `resetElektrons()`. Její kód je vcelku názorný, takže ho zde uvádím:

```
void resetElektrons(SoSeparator * root){

  deleteAllElektrons(&elektrons,root);  //smaže simulované částice
  deleteAllMicros(&micros,root);
  pickedElectron = NULL; // vynulování ukazatele na vybraný elektron
      //odplánování senzorů
  if (sensor !=NULL && !sensor->isScheduled()) sensor->unschedule();
```

```
  if (tSensor !=NULL && !tSensor->isScheduled()) tSensor->unschedule();
  create1Elektron(root); //vytvoření elektronů
  sensor->schedule(); //znovu rozjetí simulace

} //end restartElektrons
```

Pozn. K vysvětlení proměnné `pickedElectron` *se dostanu na konci této kapitoly.*

4.2.5 Zapnutí/vypnutí ořezávací roviny

Tato možnost v programu je velmi důležitá. Umožňuje nahlédnout dovnitř urychlovače a pozorovat tak simulaci v plném rozsahu. Při vypnuté volbě může sice uživatel obdivovat krásu lineárního urychlovače, ale ze samotné simulace (pohybu elektronů) nebude mít nic. Pro ořezávací roviny obecně Open Inventor nabízí třídu `SoClipPlane`. Ta může přebírat jako argument konstruktoru libovolně definovanou rovinu. Aby uživatel měl co nejlepší přehled o simulaci, zvolil jsem rovinu tak, aby procházela středem podél celé délky lineárního urychlovače. Rovina je zadána pomocí bodu a normály. Jelikož LINAC roste ve směru osy Z, bude normála určena osou X. Určujícím bodem bude počátek souřadnicového systému `nullPoint` - bod [0,0,0]. Pomocí třídy `SbPlane` tuto rovinu vytvořím. Jestli chci ořezávanou rovinu aplikovat nebo ne, můžu nastavit pomocí člene `on` objektu třídy `SoClipPlane`. Vytvoření a aktivování ořezávací roviny je tak možné implementovat takto:

```
            if (bClipPlane) {//chci ořezávat
                  pl.setValue(SbPlane(SbVec3f(1,0,0),nullPoint));
                  ((SoClipPlane*) clip)->on.setValue(true);

            } else { //nechci  ořezávací rovinu
                  ((SoClipPlane*) clip)->on.setValue(false);
            }
```

Jelikož ne každému vyhovuje pevně daná ořezávací rovina, pokusil jsem se najít i jinou možnost jak umožnit dobrý výhled na simulaci. Uvažoval jsem možnost vytvoření průhledného modelu, ale tuto možnost jsem zavrhl, protože se mi zdála nepřehledná. Druhou možností, již jsem nakonec i zvolil jako optimální řešení, byla rovina, která by se inteligentně upravovala podle pohledu kamery. Implementací se zabývám v následujícím odstavci.

4.2.6 Inteligentní ořezávací rovina

Pro vytvoření jakéhokoliv mechanismu, který reaguje na změnu některého údaje uvnitř Open Inventuru, nám knihovna nabízí senzor nazvaný `SoFieldSensor`. Tomu se dá přiřadit libovolná veličina, kterou bude hlídat a jakmile se údaj změní vyvolá se příslušná callback funkce. Ke správné reakci nastavení roviny bylo potřeba celkem dvou těchto senzorů. Jeden pro údaj pozice kamery a druhý pro údaj orientace kamery:

```
SoFieldSensor * sensor1 = new SoFieldSensor(sensorClipClb,clip);
sensor1->attach(&(viewer->getCamera()->position));

SoFieldSensor * sensor2 = new SoFieldSensor(sensorClipClb,clip);
sensor2->attach(&(viewer->getCamera()->orientation));
```

Při každém pohybu kamery se tak vyvolává stejná callbacková funkce `sensorClipClb()`. V té se zjistí vektor `lookAt`, určující směr pohledu kamery a vytvoří se jeho vektorový součin s vektorem `linVec` udávajícím směr růstu LINACu pomocí funkce `cross()` *(z angl. cross product)*. Výsledkem je normála k rovině, kterou určují vektory `linVec` a `lookAt`. Nazvěme ji `tmpNorm`. Vektor `tmpNorm` a vektor `lookAt` určují rovinu kterou hledáme. Můžeme buďto použít pro definici roviny tyto dva údaje, nebo z nich vytvořit vektorovým součtem normálu k této rovině a držet se tak již jednou použitého postupu z minulého odstavce.

```
SbVec3f lookat(0, 0, -1); // default směr kamery
camrot.multVec(lookat, lookat);  //zjištění aktuálního směru kamery
if (lookat.length == 0) return;  //nesmí být nula
SbVec3f linVec(0,0,1); //vektor LINACu
tmpNorm=lookat.cross(linVec); //lookat bude normalou jedne roviny
if (tmpNorm[0] == 0 && tmpNorm[1] == 0 && tmpNorm[2] == 0 ) return;
//vysledna normala pro použití při konstrukci IT ořezávací roviny
norm=linVec.cross(tmpNorm);
```

Pomocí tohoto kódu se přpočítává rovina pokaždé, když se změní některý z parametrů kamery. Výsledná rovina je vždy optimálně volena pro náhled z dané pozice.

4.2.7 Redraw – volba pro aplikování změn na modelu LINACu

Tato volba se uplatní zejména pokud jsme změnili rozměry LINACu (o tom, jak toho docílit bude řeč dále). V opačném případě se projeví úplně stejně jako volba restart. Je to dáno tím, co se vykoná po aktivaci Redraw. V samotné callback funkci se nastaví pouze příznak pro vykreslení nového LINACu, ten je v kódu pojmenován `bRedraw`. Podle tohoto příznaku se pak callback funkce senzoru, používaného pro pohyb, rozhodne jestli má ukončit simulaci a vytvořit nový LINAC nebo má dále pokračovat v simulaci.
Jestliže se rozhodne pro vytvoření nového LINACu, volá až na pár výjimek vše, co vyvolá volba Reset. Prvním krokem je tedy zastavit simulaci. Pak si funkce vynutí nové přestavení grafu scény a to voláním funkce `redrawLINAC()`. Ta zahodí celý dříve vytvořený graf určující tvar modelu LINACu a podle nových údajů vytvoří, za pomoci stejného kódu jako při inicializaci programu, graf nový. Ten dá na místo starého grafu. Je samozřejmostí, že v tomto čase nesmí docházet k žádnému průchodu grafu. Ten je totiž v té chvíli nekompletní a program by se zhroutil. Proto je také celá simulace pozastavena. Jakmile je graf znovu vytvořen může se simulace znovu rozběhnout. Znovu se tedy naplánují senzory a všechny proměnné se uvedou do stavu, jaký mají mít na začátku simulace. Důležitý fakt je ten, že se v grafu neruší nic jiného než model. Ořezávací rovina, menu, senzory... vše zůstává.

4.2.8 Rychlost , energie a selekce elektronu

Tyto dvě položky v menu mají stejný význam i konstrukci jako Infoline. Do nich se vypisují informace o rychlosti a energii vybraného elektronu. Nemá asi cenu rozebírat jejích konstrukci. Spíše se zde už zaměřím na možnost „selekce“ elektronů a její implementaci.
Selekce elektronu je, jak nejspíš čtenáře již napadlo, na stejném principu jako selekce položek v menu. Nebrání nám nic ve využití již jednou napsaného kódu pro obsluhu menu. Některým dodatečným úpravám se samozřejmě nevyhneme, ale ty jsou už záležitostí čistě mechanickou. Tedy na stejné místo a úroveň kódu, kde kontrolujeme zda se jedná o položku menu, umístíme podmínku, která zjistí zda místo menu nebylo kliknuto na elektron. Elektron, jak již bylo dříve zmíněno, je reprezentován grafickým prvek `SoSphere`. Po úpravě bude segment programu, který zjišťuje, zda bylo kliknuto na menu, vypadat následovně:

```
            //je to elektron a nebo položka v menu
      if (MyNode->getTypeId() == SoSphere::getClassTypeId()){
            //je to zřejmě elektron
            SoSphere * pickedSphere = (SoSphere *) MyNode;
            //najdeme kterému elektronu vybraná koule patří
            findElektron(pickedSphere);
            return; //opust9me callback funkci
      } else if (MyNode->getTypeId() == SoText2::getClassTypeId()){
            //bylo kliknuto na polo6ku v menu
            pickedBtn = (SoText2 *) MyNode;
      } else return; //nebylo kliknuto na nic co nás zajímá
      ...
```

Je však potřeba zvážit jednu věc. Ve scéně nejsou všechny koule (instance třídy SoSphere) reprezentací elektronu. Jednak to může být třeba částice třídy Micro, anebo to může být jakýkoliv jiný dekorativní prvek na LINACu, na který bylo použito objektu třídy `SoSphere`. Proto voláme funkci `findElektron()`, které jako argument předáváme ukazatel na vybranou kouli. Tato metoda projde celý vektor elektronů a u každého zjišťuje, zda se náhodou adresa koule nerovná adrese koule reprezentující daný elektron. Funkce `findElektron()` v podstatě tvoří všechen kód pro výběr elektronu

```
void findElektron(SoSphere * sph) {
            if (pickedElectron!=NULL) //je už nějaký vybrán?
              pickedElectron->material-> //ano->změníme jeho barvu do normálu
                   diffuseColor.setValue(SbColor(0.9f, 0.9f, 0.0f));

            //najdem nový...nemusí existovat!
            for (list<Elektron*>::iterator ci=elektrons.begin();
                                          ci != elektrons.end(); ++ci) {
                  if ((*ci)->sphere == sph) { //porovnání adres

                     (*ci)->material-> //změna materiálu
                        diffuseColor.setValue(SbColor(0.9f, 0.0f, 0.0f));
                      pickedElectron=(*ci); //uložíme ukazatel na vybraný
                  } //end if
            }     //end for
} //end findElektron
```

4.3 Změna podmínek simulace

Změnou podmínek simulace rozumím přenastavení zdroje nebo lineárního urychlovače. Do této podkapitoly také zahrnu řízení rychlosti simulace a systém nastavení jednotek pro změnu podmínek simulace. Začněme tímto systémem, protože jeho výstupy přímo ovlivňují všechny ostatní ovládací prvky v této skupině.

4.3.1 Nastavení rozměru změny

Zvolil jsem vcelku jednoduchý systém, který dovoluje umožnit rozsah změn pomocí jednotek až miliónů. Ovládací prvek tvoří skupina šesti přepínačů - tlačítek, fungujícím na principu rádiových tlačítek *(radio button)* známých ze standardního GUI. V daný okamžik může být vybrán jenom jeden. Ten, který je vybrán nastaví na daný počet (1,10,100,1000 ...) proměnou rozměr, podle kterého se pak mění další ovládací prvky o nichž bude za chvíli řeč. Celá záležitost je implementována stejně jak každé jiné tlačítko. Jediné co je zde unikátní, je zvýraznění ovládacího prvku, podle kterého je rozměr v dané chvíli nastaven. Kód snad nemá ani cenu uvádět.

4.3.2 Regulace rychlosti simulace

Vzhledem k faktu, že rychlosti pohybu elektronu se mohou řádově lišit v miliónech metrech za vteřinu, bylo potřeba implementovat možnost celou simulaci zpomalit či zrychlit. Ovládací prvek pro regulaci rychlosti simulace se sestává z tlačítka up, samotného číselného údaje určující zpomalení nebo zrychlení, a tlačítka down.
Všechny časy v simulaci jsou neustále korigovány proměnnou `velSim`. A právě tato proměnná je měněna za pomocí získaného údaje. Rychlost urychleného elektronu je totiž tak velká, že kdybychom nechali simulaci, byť s malými urychlovači, probíhat v reálném čase, asi by ze simulace moc nezbylo. Jednak by z technického hlediska nebylo možné na personálních počítačích takový pseudorealtime výpočet provést a jednak by lidské oko pohyb elektronu nepostřehlo. Proto se vše implicitně zpomaluje. Rozsah položky určující zrychlení a zpomalení je <-1000,9). Proč právě takovýto nesouměrný interval? Odpověď jsem již víceméně podal. Je to z toho důvodu, že uživatel spíše bude simulaci zpomalovat než zrychlovat. Na druhou stranu není dobré simulovat příliš velké rychlosti elektronu. Animace je pak trhaná a celkový dojem z aplikace klesá. Optimální je simulace malých urychlovačů u kterých stačí rozsah (9,-9). Nechtěl jsem však potencionálního uživatele ochudit o simulaci větších rychlostí.

4.3.3 Frekvence a napětí zdroje

I tento parametr můžeme nastavovat. Změny se projeví ihned. Schéma ovládacího prvku je stejné jako u regulace rychlosti simulace.

4.3.4 Rozměry LINACu

Nastavujeme dva údaje. Oba se projeví až po zavolání metody `redrawLINAC()` pomocí tlačítka `Redraw`. Hodnoty těchto údajů jsou před překreslením LINACu zkontrolovány funkcí `kontrolaOK()`.

5 Závěr

Projekt myslím dosáhl svých cílů. Aplikace, která v rámci projektu vznikla je prezentací síly a možností knihovny Open Inventor, zároveň je ale prakticky použitelná. Jak už jsem se zmiňoval v úvodu, rozhodně si nečinní ambice zařadit se mezi regulérní simulátory urychlení. Je však myslím dostatečně názorná pro použití při demonstraci funkce vysokofrekvenčního urychlovače (například studentům středních škol).

Jsem si však vědom několika nedostatků, které by stály za vylepšení. Jak už jsem se zmínil v úvodu této technické zprávy, je zcela opomenuta relativistická složka pohybu elektronu. Je tedy bohužel možné dosáhnout větší rychlosti než je rychlost světla., což je podle pojetí dnešní vědy nereálné.

Dalším 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.

Prvním cílem, v případě pokračování tohoto projektu v rámci diplomové práce, by bylo odstranění výše uvedených nedostatků. Dalším krokem by zřejmě byla optimalizace pro zlepšení výkonu aplikace. V tomto směru by stálo určitě za změnu vyjádření elektronů a částic.

Doufám, že tato práce neposlouží jen ke splnění ročníkového projektu, ale že se uplatní i v praxi.

SEZNAM POUŽITÉ LITERATURY

[1] Halliday, D. – Resnick, R. – Walker, J. (2000): Fyzika – Elektřina a magnetismus, Brno, Vutium, Prometheus.

[2] Wernecke, J.(1994): The Inventor Mentor:Programming ObjectÿOriented 3D Graphics with Open Inventor, Release 2, Silicon graphics

[3] Halliday, D. – Resnick, R. – Walker, J. (2000): Fyzika – Moderní fyzika, Brno, Vutium, Prometheus.

[4] http://graphics.idav.ucdavis.edu/graphics/GraphicsNotes/homepage.html

[5] http://www.lns.cornell.edu/~dugan/USPAS/

[6] http://sweb.cz/AstroNuklFyzika/JadRadFyzika5.htm#Urychlovace

[7] http://root.cz/clanky/open-inventor/

SEZNAM PŘÍLOH

Text 1 Menu – popis ovládání programu pro simulaci LINACu

PŘÍLOHY

Menu – popis ovládání aplikace pro simulaci LINACu

Infoline - slouží k vypisování informací o právě provedené činnosti. Využívá se například pro potvrzení vytvoření nového modelu lineárního urychlovače. Jestliže se vytvoření zdaří objeví se zpráva: „New LINAC created“. Jestliže se například kvůli nesprávnému poměru délky první elektrody a celkové maximální délce LINACu nepodaří vytvořit (neprojde kontrolní funkcí), program vypíše na Infoline zprávu: „"Length of LINAC has to be bigger than its first electrode."“

Fire electron - po vybrání této volby se emituje nový elektron a vstoupí co nejdřív do urychlovací trubice.

Start animation - tato volba slouží uživateli, jestliže potřebuje celou simulaci zastavit nebo opětovně spustit. Jestliže je v daném okamžiku simulace spuštěna a běží je na místě volby “Start animation“ volba „Stop animation“ pro lepší rozlišení stavu ve kterém se simulace nachází.

Reset - vytvoří stejně jako volba „Fire electron“ nový elektron a umístí ho do urychlovací trubice. Ale na rozdíl od volby „Fire electron“ vyresetuje napětí na zdroji tak aby bylo synchronní s pohybem elektronu. Tedy za předpokladu že máme nastavenu správně frekvenci, délky LINACu a amplitudu napětí, dochází k maximálnímu urychlení tohoto elektronu na právě zkonstruovaném urychlovači.

On clipPlane - vypíná/zapíná ořezávací rovinu. Jinými slovy umožňuje/znemožňuje uživateli nahlédnout dovnitř LINACu. Opět, stejně jako u volby „Start animation“, se přepisuje název na akci která bude provedena. Při již povolené ořezávací rovině bude text změněn na „Off clipPlane“.

On IT clipPlane - vypíná/zapíná inteligenci nastavování ořezávací roviny. Při zapnuté volbě se rovina natáčí tak, aby bylo umožněno uživateli nahlédnout dovnitř LINACu y kterékoliv pozice kamery.

Redraw - podle zadaných parametrů (celkové délky, délky první elektrody) vypočte a vytvoří nový LINAC. Všechny proměnné simulace se vyresetují. Zdroj je pozastaven. K opětnému spuštění zdroje slouží volba „Restart“ nebo volba „Fire electron“.

Velocity - zde se vypisuje údaj o rychlosti v metrech za sekundu.

Energy - zde se vypisuje údaj o energii elektronu v elektronvoltech.

Nastavování parametrů simulace

Všechny následující položky mají stejný sytém nastavování. Volba před číslem („jménoPoložky up“) slouží k navýšení parametru, volba za číslem („down“) pak ke snížení údaje parametru. U každé položky je nastaven určitý rozsah omezující parametr. Rovněž některé kombinace, jako výše zmiňovaný poměr délek, nejsou přípustné. Při nastavování parametrů je potřeba zohlednit nastavený rozměr přes, který se bude veličina modifikovat. Ten se dá nastavit pomocí přepínačů (viz. Níže).

Sim up X down - slouží k nastavování rychlosti simulace. Rozsah je od -200 do 9. Je tak možné regulovat rychlost letícího elektronu. Projeví se ihned.

Linac length up X down
- nastavuje se celková délka lineárního urychlovače. Rozsah je 200 až 10000 metrů. Projeví se až po volbě „Redraw“.

First length up X down
- nastavuje délku první elektrody. Podle tohoto a předchozího se vypočítá a vytvoří LINAC. Projeví se až po volbě „Redraw“. Rozsah je 10 až 1000 metrů.

Frequency up X down
-nastavuje frekvenci zdroje. Rozsah je 1000 až 10000000 Hz. Projeví se ihned.

Voltage up X down
-nastavuje amplitudu zdroje. Rozsah je 10 až 50000 voltů. Projeví se ihned.

Přepínače rozměrů

Vždy je vybrán pouze jeden z šesti možných. Implicitně je nastaven přepínač „P“. Jestliže je daný přepínač aktivní je znázorněn velkým písmenem, jestliže není pak písmenem malým.

Implementováno je následujících šest možností:

P – jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu jednotek.

D – jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu desítek.

H– jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu stovek.

T – jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu tisíců.

DT – jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu deseti tisíců.

HT – jestliže je aktivován tento přepínač, bude každá změna parametru na úrovni řádu sta tisíců.