Univerzita Palackého v Olomouci
Přírodovědecká fakulta

Modelování fyzikálních procesů pomocí trasovacího nástroje Geant4

Libor Nožka

Olomouc 2012

Oponent: Mgr. Petr Hamal

Publikace byla připravena v rámci projektu Investice do rozvoje vzdělávání

Tento projekt je spolufinancován Evropským sociálním fondem
a státním rozpočtem České republiky

1. vydání

ISBN 978-80-244-3078-2

NEPRODEJNÉ

Obsah

.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vzdělávání výzkumných pracovníků v Regionálním centru pokročilých technologií a materiálů. CZ.1.07/2.3.00/09.0042

Modelování fyzikálních procesů pomocí trasovacího nástroje Geant4

Libor Nožka

Abstrakt. V aplikační oblasti se při návrzích detekčních systémů naplno začaly používat softwarové nástroje pro modelování jejich účinnosti a optimalizaci geometrie. V studiích proveditelnosti se provádí srovnávací/výkonnostní testy (tzv. benchmark) uspořádání jednotlivých částí detektoru a podle toho se volí řešení. Vedle toho nachází tyto nástroje v rekonstrukcích měřených dat, kdy je třeba experimentálně získaná data aproximovat do oblastí mimo možnosti měření popř. dojít k výsledkům nepřímo měřených veličin.

Nástroj Geant4 zaujímá přední místo v této oblasti. I když byl přednostně navržen pro nasazení v částicové fyzice, díky jeho univerzálnosti se používá také v kosmickém výzkumu a medicíně (známá nadstavba GATE – Geant4 Application for Tomographic Emission). Přesnost modelování silně závisí na věrohodnosti jak modelů popisujících fyzikální procesy, tak popisu geometrie a materiálového složení prvků účastnících se modelování. V Geant4 jsou tyto aspekty řešeny do velké hloubky a především způsobem, který umožňuje snadné rozšíření do širokého spektra oblastí.

Tento text je koncipován tak, aby seznámil čtenáře se strukturou nástroje a způsobu implementace vlastního problému.

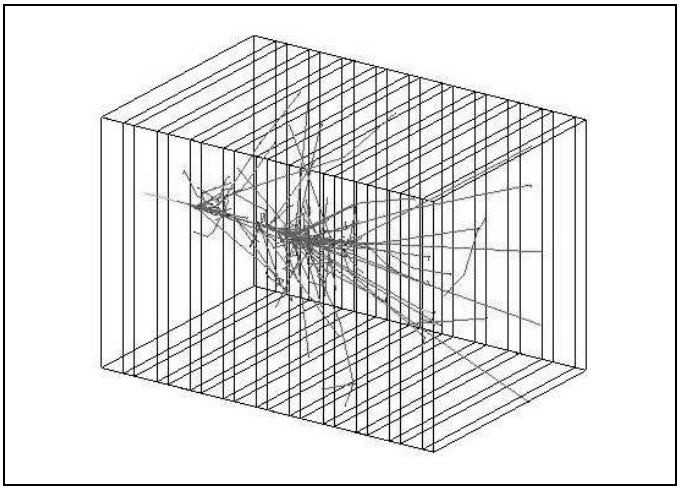
Projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

1 Úvod

K potřebě simulace procesů na úrovni částicové fyziky je v současné době používán nejpropracovanější softwarový nástroj Geant4 [1]. Geant4 umí simulovat velkou množinu interakcí jak na úrovni vysokoenergetické a nízkoenergetické částicové fyziky, tak také na úrovni vlnové optiky, elektřiny a magnetismu.

Nástroj Geant4 byl přednostně vyvinut pro potřeby v částicové fyzice, příklad viz Obrázek 1, díky jeho univerzálnosti se používá také v kosmickém výzkumu a medicíně (známá nadstavba GATE – Geant4 Application for Tomographic Emission). Geant4 lze podle potřeby upravovat jak v oblasti upřesnění modelů fyzikálních procesů, např. [2], tak v oblasti charakteristiky materiálů.

Cílem této práce je ozřejmit způsob použití nástroje Geant4.



Obrázek 1 – Příklad vizuálního výstupu simulace - absorpce elektronu o 500 MeV v kalorimetru.

2 Geant4 - základní charakteristika

2.1 Co je Geant4?

Softwarový nástroj (též toolkit) Geant je softwarový produkt vyvíjený mnoha spolupracujícími institucemi v čele s CERN, ESA (European Space Agency), SLAC a Helsinky Institute of Physics. V současné době je hojně používán vědeckými pracovišti zabývajícími se problematikou

fyziky vysokých energií (CERN, SLAC, Fermilab apod.). Umožňuje simulovat rozsáhlé experimenty, které tato pracoviště realizují (např. projekt ATLAS, DELPHI) a pomáhat tak při jejich úpravách a odstraňování některých problémů.
Geant4, v současné době ve verzi 9.5, je v podstatě knihovna tříd (objektový typ) v jazyce C++, jejichž použitím vytváříme simulační program. Každá z těchto tříd implementuje určitou specifickou část kódu simulace a jejich vhodným ”poskládáním” vytváříme v jazyce C++ program určený pro danou simulaci.
Tyto třídy lze rozdělit do následujících logických kategorií:

1. **Run a Event**: tato kategorie tříd se vztahuje k řízení běhu simulace (tzv. run) a vytváření událostních kroků (tzv. eventů) simulace, např. tvorba sekundárních částic.
2. **Řízení trasování**: jedná se o kategorii tříd, které řídí trasování (krokování) běhu simulace. Sledují šíření všech částic existujících v daném kroku simulace a uplatňují na nich relevantní fyzikální procesy.
3. **Geometrie a CAD rozhraní**: třídy v této kategorii spravují geometrii detektorů a vzájemnou polohu jejich komponent. Geant4 obsahuje návrhář geometrických objektů založený na standardu ISO STEP, a je tak plně schopen spolupracovat s CAD systémy.
4. **Magnetické pole**: pomocí tříd této kategorie je možně začlenit do detektorů magnetické pole. Objektový návrh, na kterém je Geant4 založen, umožňuje změnou numerických algoritmů v odvozených třídách definovat i pole odlišná od magnetického.
5. **Definice částic a materiálu**: pomocí tříd této kategorie definujeme částice a materiál pro použití v detektorech.
6. **Fyzikální procesy**: rozsáhlá kategorie tříd spravujících všechny fyzikální procesy, které se účastní interakcí mezi částicemi.
7. **Snímání interakcí a digitalizace**: dvě úzce propojené kategorie, které spravují informace o interakcích (pozice, čas vzniku, energie, hybnost a geometrická informace). Tyto informace jsou určené především pro modely AD převodníků, trigrovací logiky nebo čítač impulsů.
8. **Vizualizace a intercom**: kategorie tříd spravující vizualizaci detektoru a trajektorií částic a zadávání příkazů pro řízení simulace. Geant4 podporuje v podstatě jakýkoliv typ vizualizace neboť objektově orientovaný návrh tohoto nástroje umožňuje implementaci konkrétních vizualizačních nástrojů plně oddělit od zbytku

kódu standardním rozhraním. Nejpoužívanějšími vizualizačními technikami jsou OpenGL, DAWN a WRML.

9. **Rozhraní s ostatním softwarem**: tato kategorie zahrnuje třídy, pomocí kterých můžeme vytvářet různá uživatelská rozhraní (GUI) a komunikaci s ostatními softwarovými technologiemi, např. databázovými systémy OODBMS, MySQL apod.

2.2 Fyzikální procesy modelované Geantem

2.2.1 Rozsah modelování

Geant4 je určen pro modelování procesů, které spadají především do oblasti částicové fyziky. Jedná se o procesy, které se týkají interakce částic s hmotou:

- rozpad částic,
- elektromagnetické interakce při vysokých energiích: fotoelektrický jev, Comptonův rozptyl, gama konverze na pár e+e− nebo μ+μ−, tranzitivní radiace, scintilace, Čerenkovův jev, fotoabsorpce, ionizace, bremsstrahlung (ztráta energie elektronů a pozitronů emisí fotonů v poli atomového jádra), anihilace e+e− na gama záření nebo na pár μ+μ−, synchrontronní radiace,
- elektromagnetické interakce při nízkých energiích: Comptonův rozptyl (včetně lineárně polarizovanými gama paprsky), Rayleighův rozptyl, elektronová ionizace, fluorescence, hadronová a iontová ionizace, energetické fluktuace v absorpčním prostředí,
- hadronové interakce: interakce lepton-hadron, fotojaderné a elektrojaderné interakce, elastický rozptyl nukleon-nukleon a hadron-nukleon při středních a vysokých energiích, neelastická interakce hadron-nukleon, anihilace nukleon-antinukleon.

Podrobnější výčet včetně krátkého popisu interakcí modelované Geantem je v [3]. Každý fyzikální proces je v Geantu reprezentován třídou, která proces popisuje a určuje za jakých podmínek nastává. Vedle uvedených fyzikálních procesů, umožňuje objektově orientovaný návrh tohoto nástroje definovat uživatelské procesy (i zcela umělé). Detailněji je způsob implementace fyzikálních procesů popsán v kapitole 3.4). Optické procesy jsou rozebrány v kapitole 4 z důvodů použití v ukázkovém simulačním programu WaterTank.

2.2.2 Modelování procesů při nízkých energiích

Nízkoenergetické procesy v Geantu reprezentují elektromagnetické interakce částic s elektronovým obalem materiálu detektoru. Jsou jimi fotoelektrický jev, Comptonův rozptyl při nízkých energiích, Rayleighův rozptyl, bremsstrahlung, ionizace a fluorescence. Současná implementace těchto procesů je věrohodná pro energie od 250 eV do 100 GeV a týká se prvků s atomovým číslem od 1 do 99. Simulace těchto procesů zahrnuje dvě fáze výpočtu:

- výpočet celkového účinného průřezu fyzikálního procesu,
- vytvoření konečného stavu částice v trasovacím kroku simulace, sledováním stavových veličin částice a sekundárních částic vzniklých procesem (emise fotonů).

Obě fáze jsou založeny na teoretických modelech procesů a na použití relevantních empiricky získaných dat, které jsou uloženy v databázi Geantu.

Databáze Geantu

Pro výpočet účinných průřezů nízkoenergetických fyzikálních procesů a konečného stavu částice po trasovacím kroku je použito databáze empiricky získaných dat uložených v databázi Geantu. Tato data byla získána z volně přístupných datových knihoven:

- EPDL97 (Evaluated Photons Data Library) [4, 5],
- EEDL (Evaluated Electrons Data Library) [6, 5],
- EADL (Evaluated Atomic Data Library) [7, 5],
- Stopping Power Data [8, 9, 10, 11],
- hodnoty vazebných energií od Scofielda [12].

Tato databáze poskytuje data pro výpočet:

- celkových účinných průřezů pro fotoelektrický jev, Comptonův rozptyl, Rayleighův rozptyl a bremsstrahlung,
- celkových účinných průřezů elektronových podslupek atomů pro fotoelektrický jev a ionizaci,
- energetických spekter sekundárních částic při interakci s elektrony,
- Hubbelova FF faktoru pro Rayleighův rozptyl,
- vazebných energií elektronů ve všech podslupkách,
- pravděpodobností přechodů mezi podslupkami pro fluorescenci.

Data pokrývají energetický rozsah od 1 eV do 100 GeV pro Rayleighův a Comptonův rozptyl, do nejnižších vazebních energií pro každý prvek pro fotoelektrický jev a ionizaci, a do 10 eV pro bremsstrahlung.

Výpočet celkového účinného průřezu
Celkový účinný průřez σ je v závislosti na energii E odvozen z příslušných hodnot v databázi. Databáze poskytuje hodnoty účinných průřezů pro diskrétní hodnoty energie. Celkový účinný průřez se při simulaci počítá následující interpolací [3]

$$\log\big(\sigma(E)\big) = \frac{\log(\sigma_1)\log(E_2/E) + \log(\sigma_2)\log(E/E_1)}{\log(E_2/E_1)}, \qquad (1)$$

kde E_1 a E_2 jsou nejbližší nižší a vyšší energie, pro kterou jsou v databázi známé hodnoty (σ_1, σ_2).
Pro výpočet střední volné dráhy λ částice, která interaguje s materiálem přes daný fyzikální proces, je třeba započítat účinné průřezy všech prvků tvořící materiál

$$\lambda = \frac{1}{\sum_i \sigma_i(E) n_i}, \qquad (2)$$

kde σ_i je účinný průřez fyzikálního procesu a n_i je atomová hustota i-tého prvku.

3 Realizace simulace s nástrojem Geant4

Práce s nástrojem Geant představuje realizovat především tyto následující kroky:

- definice detektoru - geometrie a materiálové složení komponent detektoru a jejich umístění, popř. specifikace magnetického pole detektoru,
- specifikace částic zúčastňujících se všech kroků simulace (tedy včetně sekundárních částic), popřípadě i jejich definice,
- specifikace fyzikálních procesů, které budou zahrnuty do simulace, popř. i jejich definice,

- vytvoření primární události (eventu), nejčastěji vystřelení částice do prostoru, kde je umístěn detektor.

Při vytváření simulačního programu tyto kroky znamenají odvodit uživatelem definované třídy od příslušných základních tříd Geantu (tzv. Mandatory User Classes) a přepsat některé jejich virtuální funkce. Ostatní kód těchto základních tříd, který je vývojáři ukryt (jedna z charakteristik objektového programování), je součástí jádra Geantu, které spravuje celkové řízení simulace. Jinak řečeno, odvození a přepsání virtuálních metod některých základních tříd Geantu představuje specifikaci modelovaného problému a kód jádra Geantu, který nelze měnit, tuto simulaci řídí.
V samotném běhu programu jsou vytvořeny instance (objekty) uživatelem definovaných tříd. Tyto objekty jsou následně zaregistrovány správci (manažerovi) simulace, který následně řídí simulaci. Do běhu simulace může uživatel zasahovat prostřednictvím instancí speciálních tříd (tzv. User Action Classes), které je opět třeba odvodit od příslušných základních tříd Geantu a zaregistrovat správci simulace. Během běhu programu pak správce simulace volá v určitých krocích určité, uživatelem přepsané, virtuální funkce. To uživateli umožňuje simulaci sledovat, ukládat výsledky mezi jednotlivými kroky pro následnou analýzu simulace nebo dokonce měnit běh simulace. Samotnou simulaci uživatel řídí pomocí sady příkazů, které může zadávat několika způsoby, detailněji v kapitole 3.6.1.
Ve zbytku této kapitoly podrobněji rozebereme všechny tyto uvedené aspekty při práci s Geantem. Napřed si však ukážeme, jak vypadá jednoduchý příklad zdrojového kódu funkce main(), ve které celý simulační program probíhá.

3.1 Funkce main() - vstupní bod programu

Obsah funkce main(), jako vstupního bodu programu, se liší podle potřeb simulace a je plně implementovan uživatelem; Geant4 tuto funkci neposkytuje. Zde je nejjednodušší příklad obsahu funkce main() potřebný pro běh simulace:

```
//zahrnutí základních balíčků
#include "G4RunManager.hh"
#include "G4UImanager.hh"
//hlavičkové soubory uživatelem definovaných tříd
#include "MyDetectorConstruction.hh"
```

```
#include "MyPhysicsList.hh"
#include "MyPrimaryGeneratorAction.hh"
int main()
{
//vytvoření implicitního správce simulace (run manager)
G4RunManager* pRunManager=new G4RunManager;
//registrace konstrukce detektoru a fyzikálních procesů
pRunManager->SetUserInitialization(new
CMyDetectorConstruction);
pRunManager->SetUserInitialization(new CMyPhysicsList);
//registrace primární události (generátoru primární
částice)
pRunManager->SetUserAction(new CMyPrimaryGeneratorAction);
//inicializace jádra Geantu po registraci
pRunManager->initialize();
//získání ukazatele na objekt správce uživatelského
rozhraní (UI manager)
//zapsání příkazů k~nastavení úrovně výpisu výsledků běhu
simulace
G4UIManager* pUI=G4UImanager::GetUIpointer();
pUI->ApplyCommand("/run/verbose 1");
pUI->ApplyCommand("/event/verbose 1");
pUI->ApplyCommand("/tracking/verbose 1");

//start simulace vysláním primární částice (zde 3x)
int nNumOfEvents=3;
pRunManager->beamOn(nNumOfEvents);
//po ukončení simulace dealokujeme paměť správce simulace
//ostatní námi alokované objekty jsou dealokovány správcem
delete pRunManager;
return 0;
}
```

Program tohoto výpisu používá instance dvou tříd poskytnutých Geantem, G4RunManager a G4UImanager, a tří tříd, CMyDetector-Construction, CMyPhysicsList a CMyPrimaryGeneratorAction, odvozených od základních tříd Geantu.

První věcí, kterou musí funkce main() udělat je vytvoření instance (objektu) třídy G4RunManager, který řídí běh programu (správce běhu simulace). Před spuštěním samotné simulace je potřeba správci poskytnout všechny potřebné informace, které se týkají:

1. geometrické konstrukce a materiálového složení detektoru,
2. seznamu všech částic a fyzikálních procesů uvažovaných v simulaci,

3. vytváření primárních částic,
4. dalších dodatečných požadavků pro simulaci.

V uvedeném příkladu, příkazové řádky:

```
pRunManager->SetUserInitialization(new
CMyDetectorConstruction);
pRunManager->SetUserInitialization(new CMyPhysicsList);
pRunManager->SetUserAction(new CMyPrimaryGeneratorAction);
```

vytváří objekty, které specifikují geometrii detektoru, fyzikální procesy a počáteční stav vytvořením primární částice. Následující instrukce:

```
pRunManager->initialize();
```

provede konstrukci detektoru, vytvoření modelů fyzikálních procesů a vytvoří objekty jádra Geantu potřebných k běhu simulace. Poslední funkce správce ve funkci main():

```
int nNumOfEvents=3;
pRunManager->beamOn(nNumOfEvents);
```

zahájí simulaci. Funkce beamOn může být volána vícekrát, každé volání pak představuje samostatnou simulaci. Jakmile je simulace zahájena, není možné měnit uspořádání detektoru ani zaregistrované fyzikální procesy. Je však možné je měnit mezi jednotlivými simulacemi.

3.2 Konstrukce detektoru

3.2.1 Geometrie

Detektor se skládá z množiny základních částí, objemů. Největší objem se nazývá svět (ang. world) a obsahuje veškeré části detektoru. Každý objem může v sobě obsahovat další objem, který se tak nazývá dceřiný objem. Souřadnice umístění dceřiného objemu se udávají vzhledem k souřadnicovému systému matečního objemu. Při konstrukci detektoru rozeznáváme tři úrovně objemů:

- geometrický objem - popsán svými geometrickými vlastnostmi (výška, šířka, poloměr válce apod.),

- logický objem - geometrický objem, kterému jsou přiřazeny fyzikální vlastnosti (materiálové složení, magnetické pole),
- fyzický objem - umístění logického objemu v matečním objemu (přiřazení polohy).

Následující příklad ukazuje vytvoření jednoduchého kvádrového detektoru naplněného vzduchem:

```
//vytvoření geometrického objemu - kvádr 100x50x20 cm
G4Box* pBox=new G4Box("ExpHallBox",50*cm,25*cm,10*cm);
//vytvoření logického objemu - naplnění vzduchem (ukazatel
pAir)
pExpHallLog=new G4LogicalVolume(pBox,pAir,
"ExpHallLog",NULL,NULL,NULL);
//umístění středu logického objemu na pozici (50,-20,0) cm
vzhledem ke světu
pExpHallPhys=new G4PVPlacement(NULL,G4ThreeVector(50*cm,-
20*cm,0*cm),m_pExpHallLog,"ExpHall",NULL,false,0);
```

Kvádr je v Geantu popsán třídou G4Box. Dalšími základními objemy jsou válec (třída G4Tubs), jehlan (G4Cons), hranol (G4Para), různoběžník (G4Trd,G4Trap), koule (G4Sphere) a torus (G4Torus). Bližší informace o vytváření těchto objemů lze nalézt v [13].

3.2.2 Materiál

Materiál je ve své podstatě tvořen jednotlivými prvky, které jsou tvořeny izotopy prvku. Geant4 pro popis materiálu analogicky zavádí pro tři základní třídy materiálového složení logického objemu:

- G4Isotope - třída definuje atom podle jeho atomového čísla, počtu nukleonů, hmotnosti na jeden mol atd.,
- G4Element - třída definuje atom podle jeho efektivního atomového čísla, efektivního počtu nukleonů, efektivní hmotnosti na jeden mol, počtu izotopů, valenční energie,účinného průřezu apod.,
- G4Material - třída popisuje materiál podle jeho složení z prvků, hustoty, teploty, tlaku, střední volné dráhy apod.

Následující příklad ukazuje, jak vytvořit materiál tvořený vodní párou:

```
//definice prvků
```

```
G4Element* pElHydrogen=new
G4Element(name="Hydrogen",symbol="H",z=1.0,a=1.01*g/mole);
G4Element* pElOxygen=new
G4Element(name="Oxygen",symbol="O",z=8.0,a=16.0*g/mole);
//definice molekuly vody
density=1.000*g/cm3;
G4Material* pH2O=new
G4Material(name="Water",density,ncomponents=2);
pH20->AddElement(pElHydrogen,natoms=2);
pH20->AddElement(pElOxygen,natoms=1);
//definice vodní páry
density=0.3*mg/cm3;
pressure=2.0*atmosphere;
temperature=500.0*kelvin;
G4Material* pSteam=new G4Material(name="Water
steam",density,ncomponents=1,kStateGas,
temperature,pressure);
pSteam->AddMaterial(pH2O,fractionmass=1.0);
```

Optické vlastnosti materiálu

V některých případech je žádoucí zahrnout do procesu simulace vlnové vlastnosti fotonů, například při emisi fotonů fluorescencí nebo Čerenkovským jevem. Optické vlastnosti materiálu, které jsou používány v optických procesech, jsou uloženy v tabulce vlastností, kterou spravuje třída G4MaterialProperitiesTable, která je propojena s příslušným objektem třídy G4Material. Tyto vlastnosti mohou být konstantami nebo jsou vyjádřeny jako funkce vlnové délky fotonu ve formě tabulky. Tabulka vlastností je implementována jako tzv. hašovací tabulka, kde každá položka je tvořena klíčem a hodnotou. Pro registraci nové vlastnosti materiálu je třeba:

1. vytvořit novou instanci třídy G4MaterialProperitiesTable,
2. přidat novou vlastnost voláním členské funkce AddProperty,
3. zaregistrovat novou tabulku vlastností pro materiál voláním G4Material::SetMaterialProperitiesTable.

Vše názorně ukazuje následující příklad, ve kterém nadefinujeme spektrální index lomu světla ve vodě:

```
const G4int nEntries = 32;
//pole energií fotonu
G4double PhotonEnergy[nEntries] =
{ 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV, 2.177*eV,
2.216*eV, 2.256*eV,
```

```
2.298*eV, 2.341*eV, 2.386*eV, 2.433*eV, 2.481*eV, 2.532*eV,
2.585*eV,
2.640*eV, 2.697*eV, 2.757*eV, 2.820*eV, 2.885*eV, 2.954*eV,
3.026*eV,
3.102*eV, 3.181*eV, 3.265*eV, 3.353*eV, 3.446*eV, 3.545*eV,
3.649*eV,
3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
//index lomu podle energie fotonu
G4double RefractiveIndex[nEntries] =
{ 1.3435, 1.3440, 1.3445, 1.3450, 1.3455, 1.3460, 1.3465,
1.3470, 1.3475, 1.3480, 1.3485, 1.3492, 1.3500, 1.3505,
1.3510, 1.3518, 1.3522, 1.3530, 1.3535, 1.3540, 1.3545,
1.3550, 1.3555, 1.3560, 1.3568, 1.3572, 1.3580, 1.3585,
1.3590, 1.3595, 1.3600, 1.3608};
//vytvoření nové tabulky vlastností
G4MaterialPropertiesTable* pWaterMPT = new
G4MaterialPropertiesTable();
pWaterMPT->AddProperty(key="RINDEX", PhotonEnergy,
RefractiveIndex, nEntries);
//zapsání tabulky vlastností
pWater->SetMaterialPropertiesTable(pWaterMPT);
```

3.2.3 Třída G4VUserDetectorConstruction

V příkladě na straně 8 jsme vytvořili objekt třídy CMyDetectorConstruction, který nese všechny informace o konstrukci a materiálovém složení detektoru a předali jeho ukazatel správci simulace k registraci. Třída CMyDetectorConstruction je odvozena od základní třídy G4VUserDetectorConstruction, která poskytuje virtuální členskou funkci Construct(). Při odvození v této funkci vytvoříme geometrii a materiálové složení detektoru, tak jak to bylo uvedeno výše. Funkce Construct() je v programu volána během inicializace správce simulace.

3.3 Specifikace částic

3.3.1 Reprezentace částic

Geant4 poskytuje pro simulaci několik typů částic z těchto kategorií:

- leptony,
- mesony,
- baryony,
- bosony,
- částice s krátkou dobou rozpadu,
- ionty.

Každý typ částice je reprezentován třídou odvozenou od základní třídy G4ParticleDefinition. Tato třída sdružuje informace o vlastnostech částice jako její jméno, hmotnost, spin, poločas rozpadu částice a rozpadové schéma. Částice, která se účastní interakce s hmotou, je odvozena od třídy G4DynamicParticle (odvozená od G4ParticleDefinition), která navíc obsahuje informace o energii částice, hybnosti a polarizaci. Nejvyšším stupněm popisu částice je třída G4Track, odvozená od třídy G4DynamicParticle, která zahrnuje vedle zmíněných vlastností také informace o pohybu částice v prostoru a čase. Objekt třídy G4StackManager jádra Geantu během simulace udržuje seznam objektů třídy G4Track a v každém kroku řídí jejich účast v procesu simulace.
Každá třída reprezentující částici má pouze jeden statický objekt (singleton). Například elektron je popsán třídou G4Electron a její singleton je G4Electron::theElectron. Ukazatel k tomuto objektu získáme voláním statické metody G4Electron::ElectronDefinition(). Ukazatel na objekt částice potom můžeme získat ze statické tabulky reprezentované třídou G4ParticleTable, která je spravována jádrem Geantu. Ukazatel na konkrétní částici získáme voláním její statické metody FindParticle:

```
G4ParticleTable* pParticleTable =
G4ParticleTable::GetParticleTable();
G4ParticleDefinition* pParticle = pParticleTable->
FindParticle("e-");
```

3.3.2 Registrace částic, třída G4VUserPhysicsList

Částice, které zahrnujeme do simulace, je potřeba zaregistrovat ve třídě odvozené od základní třídy G4VUserPhysicsList. Registrace se provádí ve virtuální členské funkci ConstructParticle získáním reference na singleton částice, jak ukazuje následující implementace třídy CMyPhysicsList :

```
CMyPhysicsList::ConstructParticle()
{
...
//požadavkem na singletony theElectron a theProton je
kompilátor zahrne
//do datové sekce programu, což zajistí alokaci paměti pro
oba singletony
G4Electron::ElectronDefinition();
G4Proton::ProtonDefinition();
```

```
...
}
```

3.4 Specifikace fyzikálních procesů

Fyzikální procesy popisují, jak částice interagují s materiálem. Geant rozlišuje sedm hlavních kategorií fyzikálních procesů:

1. elektromagnetické procesy,
2. hadronové procesy,
3. rozpad částic,
4. fotolepton-hadronové interakce,
5. optické procesy,
6. parametrizace,
7. transport částic.

Každý fyzikální proces je popsán třídou odvozenou od základní třídy G4VProcess. Procesy jsou během simulace zpracovávány stejným způsobem na základě trasování. Tato abstrakce umožňuje uživatelsky definovat zcela nové procesy a zapojit je nenásilně do simulace. K tomu je potřeba ve třídě odvozené od G4VProcess přepsat potřebné virtuální členské funkce XXXDoIt popisující stav částic před a po interakci a funkce XXXGetPhysicalInteractionLength definující pravděpodobnost informace na základě účinného průřezu procesu. Detailnější popis lze najít v [13].
Geant4 poskytuje třídy pro širokou škálu procesů vyskytujících se ve fyzice vysokých energií. Například Comptonův rozptyl je popsán třídou G4ComptonScattering, Čerenkovův jev třídou G4Cerenkov apod. Kompletnější výčet lze nalézt opět v [13]. Proto, aby se částice mohla při simulaci účastnit fyzikálního procesu, je třeba tento proces pro částici zaregistrovat. Každá částice odvozená od třídy G4ParticleDefinition dědí od této třídy ukazatel na objekt třídy G4ProcessManager, tzv. správce procesů, který obsahuje seznam všech procesů zaregistrovaných pro tuto částici. Objekt obsahuje informace o pořadí volání procesů a rozhoduje o volání členských funkcí XXXDoIt každého procesu. Registrace fyzikálního procesu se provádí ve virtuální členské funkci ConstructProcess() třídy odvozené od základní třídy G4VUserPhysicsList. Vlastní registrace se provádí členskou funkcí AddProcess třídy G4ProcessManager. Následující část kódu ukazuje způsob zaregistrování některých fyzikálních procesů pro optický foton, elektron a pozitron ve třídě CMyPhysicsList :

```
CMyPhysicsList::ConstructProcess()
{
pCerenkovProcess = new G4Cerenkov("Cerenkov");
pScintillationProcess = new
G4Scintillation("Scintillation");
pAbsorptionProcess = new G4OpAbsorption();
pRayleighScatteringProcess = new G4OpRayleigh();
//procesy na rozhraní dvou prostředí, použití UNIFIED
modelu
pBoundaryProcess = new G4OpBoundaryProcess();
G4OpticalSurfaceModel theModel = unified;
pBoundaryProcess->SetModel(theModel);
//iterátor na seznam všech částic registrovaných dříve pro
simulaci
//theParticleIterator poskytuje základní třída
G4VUserPhysicsList
theParticleIterator->reset();
while((*theParticleIterator)()){
//získání ukazatele na částici a jejího správce procesů
G4ParticleDefinition* pParticle = theParticleIterator->
value();
G4ProcessManager* pProcManager = pParticle->
GetProcessManager();
G4String strParticleName = pParticle->GetParticleName();
//může se částice účastnit procesu?
if(pCerenkovProcess->IsApplicable(*pParticle)) {
//jestliže ano, pak jej pro částici zaregistrujeme
pProcManager->AddContinuousProcess(pCerenkovProcess);
}
if(pScintillationProcess->IsApplicable(*pParticle)) {
pProcManager->AddProcess(pScintillationProcess);
pProcManager-
>SetProcessOrderingToLast(pScintillationProcess,
idxAtRest);
pProcManager-
>SetProcessOrderingToLast(pScintillationProcess,
idxPostStep);
}
//speciálně pro optický foton zaregistrujeme absorpci a
Rayleighův rozptyl
if(strParticleName == "opticalphoton") {
G4cout << " AddDiscreteProcess to OpticalPhoton " <<
G4endl;
pProcManager->AddDiscreteProcess(pAbsorptionProcess);
pProcManager-
>AddDiscreteProcess(pRayleighScatteringProcess);
```

```
pProcManager->AddDiscreteProcess(pBoundaryProcess);
}
} else if(strParticleName == "e-") { //registrace pro
elektron
pProcManager->AddProcess(new G4MultipleScattering(),-1, 1,
1); //násobný rozptyl
pProcManager->AddProcess(new G4eIonisation(), -1, 2, 2);
//ionizace
pProcManager->AddProcess(new G4eBremsstrahlung(), -1, 3,
3); //Bremsstrahlung
} else if (strParticleName == "e+") { //registrace pro
pozitron
pProcManager->AddProcess(new G4MultipleScattering(),-1, 1,
1); //násobný rozptyl
pProcManager->AddProcess(new G4eIonisation(), -1, 2, 2);
//ionizace
pProcManager->AddProcess(new G4eBremsstrahlung(), -1, 3,
3); //Bremsstrahlung
pProcManager->AddProcess(new G4eplusAnnihilation(), 0,-1,
4); //anihilace
}
}
}
```

3.5 Vytvoření primární události

Základní třída G4VUserPrimaryGeneratorAction je určena pro odvození uživatelské třídy, která specifikuje primární událost simulace, nejčastěji vystřelení primární částice do světa (prostoru) detektoru. Zveřejňuje virtuální členskou funkci generatePrimaries, jejímž přepsáním vytvoříme prvotní událost. Tato funkce je volána ve funkci beamOn správce simulace. Při tvorbě prvotní události postupujeme tak, že vytvoříme objekt, tzv. generátor primární události, třídy odvozené ze základní třídy G4VPrimaryGenerator. Geant poskytuje odvozenou třídu G4ParticleGun pro specifikaci primární částice - druh částice, její hybnost, směr pohybu a energii. K tomu slouží veřejné členské funkce třídy G4ParticleGun. Následující část ukázka demonstruje použití této třídy k vytvoření primární částice:

```
//odvození třídy MyPrimaryGeneratorAction od
G4VUserPrimaryGeneratorAction
class CMyPrimaryGeneratorAction : public
G4VUserPrimaryGeneratorAction
```

```
{
...
private:
//členská proměnná - ukazatel na objekt třídy G4ParticleGun
G4ParticleGun* m_pParticleGun;
...
}
//v konstruktoru vytvoříme objekt třídy G4ParticleGun
CMyPrimaryGeneratorAction::CMyPrimaryGeneratorAction()
{
G4int nParticleCnt = 1; //počet částic v~děle
m_pParticleGun = new G4ParticleGun(nParticleCnt);
//v tabulce zaregistrovaných částic vyhledáme elektron
G4ParticleTable* pParticleTable =
G4ParticleTable::GetParticleTable();
G4ParticleDefinition* pParticle = pParticleTable-
>FindParticle("e-");
//nastavíme potřebné vlastnosti primární částice
m_pParticleGun->SetParticleDefinition(pParticle);
m_pParticleGun->SetParticleTime(0.0*ns);
m_pParticleGun-
>SetParticlePosition(G4ThreeVector(0.0*cm,0.0*cm,0.0*cm));
m_pParticleGun-
>SetParticleMomentumDirection(G4ThreeVector(1.,0.,0.));
m_pParticleGun->SetParticleEnergy(500.0*keV);
...
}
//v destruktoru dealokujeme paměť objektu třídy
G4ParticleGun
CMyPrimaryGeneratorAction::~CMyPrimaryGeneratorAction()
{
delete m_pParticleGun;
}
//vystřelení částice
CMyPrimaryGeneratorAction::generatePrimaries(G4Event*
pEvent)
{
//pro každou událost specifikujeme odlišný směr pohybu
primární částice
G4int nEventMode=pEvent->get_eventID()%3;
switch(nEventMode){
case 0:
m_pParticleGun->SetParticleMomentumDirection
(G4ThreeVector(1.0,0.0,0.0));
break;
case 1:
```

```
m_pParticleGun->SetParticleMomentumDirection
(G4ThreeVector(1.0,0.1,0.0));
break;
12
case 2:
m_pParticleGun->SetParticleMomentumDirection
(G4ThreeVector(1.0,0.0,0.1));
break;
}
//vystřelení částice
m_pParticleGun->generatePrimaryVertex(pEvent);
}
```

3.6 Směrování a vizualizace simulace

3.6.1 Směrování simulace

Po inicializaci správce běhu simulace můžeme ovlivnit směrování simulace celou množinou příkazů. Některé příkazy můžeme vidět v kódu na straně 6. Zadávání příkazů realizujeme pomocí objektu třídy G4UImanager. Třída poskytuje snadno rozšiřitelný příkazový interpret. Příkazy lze předat příkazovému interpretu několika způsoby:

1. přímým začleněním příkazů do kódu (tak jak je tomu v uvedeném příkladě),
2. předáním dávkového souboru se seznamem příkazů vstupní funkci main() programu,
3. znakovým terminálem typu bash, thcs nebo DOS interpretr,
4. dialogovým terminálem typu Xm, Xaw, Motif, Qt nebo Win32 dialogy,
5. pomocí GAG - plně grafické uživatelské rozhraní a jeho rozšíření GainServer typu klient/server,
6. použitím OPACS/Wo dialogovým manažerem ve spojení s vizuálu-začním systémem OPACS.

Mimo prvních dvou způsobů se pro předání příkazů interpretru užívá tzv. sezení. Sezení vytvoříme vytvořením instance třídy G4Session. Tento objekt je potřeba spojit s objektem třídy reprezentující dané uživatelské rozhraní. Například třída G4UItcsh spravuje znakové terminálové rozhraní typu tcsh, třída G4UIXm implementuje dialogové rozhraní typu Xm. Abychom mohli použít konkrétní uživatelské rozhraní, je třeba mít

nainstalované příslušné programové balíčky. S programovým balíkem Geantu verze 4 dostáváme k dispozici vedle znakových terminálů také rozhraní Xm a Xaw. Následující ukázkový kód definuje uživatelské rozhraní Xm nebo tcsh terminál pokud jeden z nich je nainstalovaný. Proměnné prostředí G4UI USE TCSH a G4UI USE XM definujeme při instalaci Geantu jsou-li příslušné balíčky těchto terminálu k dispozici.

```
int main(int argc, char* argv[])
{
//získáme ukazatel na statický objekt příkazového
interpretu
G4UImanager* pUI = G4UImanager::GetUIpointer();
if(argc==1){ //nepředáváme dávkový soubor
G4UIsession* psession=NULL;
#ifdef G4UI_USE_XM //je k~dispozici Xm rozhraní
psession=new G4UIXm(argc,argv);
#else
#ifdef G4UI_USE_TCSH // je k~dispozici tcsh terminál
psession=new G4UIterminal(new G4UItcsh);
#else
psession=new G4UIterminal(); // je k~dispozici pouze
základní terminál
#endif //G4UI_USE_TCSH
#endif //G4UI_USE_XM
//start sezení
psession->SessionStart();
delete psession;
}
else{ //předáváme dávkový soubor
13
G4String command="/control/execute ";
G4String filename=argv[1];
pUI->ApplyCommand(command+filename);
}
}
```

3.6.2 Často používané příkazy

`/vis/open`
Příkaz otevře ovladač okna vizualizace (viz. níže), do kterého bude vykreslen detektor a trajektorie částic. Například příkaz /vis/open OGLIX otevře okno systému OpenGL.

`/vis/viewer/set/viewpointThetaPhi`

Tímto příkazem nastavíme směr pohledu na svět detektoru, například /vis/viewer/set/viewpointThetaPhi 35 45.

`/vis/viewer/zoom`
Příkaz znásobí zvětšení světa detektoru, například /vis/viewer/zoom 0.7.

`/vis/drawVolume`
Příkaz vykreslí detektor do vizualizačního okna.

`/tracking/storeTrajectory`
Příkaz nastaví příznak pro uložení trajektorií částic.

`/run/beamOn`
Příkaz spustí simulaci, například /run/beamOn 1.

`/gun/energy`
Tento příkaz nastaví energii primární částice, například /gun/energy 500*MeV.

3.6.3 Vizualizace simulace

Průběh simulace lze vykreslit do speciálního okna, které je spravováno svým vizualizačním ovladačem. Ovladač je prezentován objektem příslušné třídy, který je potřeba zaregistrovat ve správci vizualizace. Správce vytváří uživatel odvozením třídy od základní třídy G4VisManager a implementací virtuální členské funkce RegisterGraphicsSystems(). Vlastní registrace se provádí voláním funkce RegisterGraphicsSystem, jak ukazuje následující kód. Pomocí příslušných proměnných prostředí předem otestujeme, zda je ovladač k dispozici.

```
//zahrneme příslušné hlavičkové soubory
#ifdef G4VIS_USE_DAWN
#include "G4FukuiRenderer.hh"
#endif
#ifdef G4VIS_USE_OPENGLX
#include "G4OpenGLImmediateX.hh"
#include "G4OpenGLStoredX.hh"
#endif
#ifdef G4VIS_USE_VRML
#include "G4VRML1.hh"
#include "G4VRML2.hh"
#endif
void TPVisManager::RegisterGraphicsSystems()
```

```
{
#ifdef G4VIS_USE_DAWN
RegisterGraphicsSystem (new G4FukuiRenderer);
#endif
14
#ifdef G4VIS_USE_OPENGLX
RegisterGraphicsSystem (new G4OpenGLImmediateX);
RegisterGraphicsSystem (new G4OpenGLStoredX);
#endif
#ifdef G4VIS_USE_VRML
RegisterGraphicsSystem (new G4VRML1);
RegisterGraphicsSystem (new G4VRML2);
#endif
if (fVerbose > 0) {
G4cout <<
"\nYou have successfully chosen to use the following
graphics systems."
<< G4endl;
//vytiskneme úspěšně zaregistrované vizualizační ovladače
PrintAvailableGraphicsSystems ();
}
}
```

4 Optické procesy

Foton je při simulaci považován za vlnu, pokud jeho vlnová délka je mnohem menší než typický rozměr atomu. Optické fotony jsou v Geantu popisovány odlišně než energetické gama fotony a vztahují se na ně optické procesy jako absorpce, Čerenkovská emise, scintilace, Rayleighův rozptyl, lom na rozhraní dvou prostředí apod. Optické vlastnosti materiálu jsou uloženy v jeho tabulce vlastností, detailněji viz kapitola 3.2.2.

4.1 Čerenkovův jev

4.1.1 Fyzikální základ

Čerenkovské záření nastává v případě, kdy elektricky nabitá částice prochází opticky disperzním prostředím rychleji, než je rychlost světla v tomto prostředí. Opticky disperzní prostředí je takové optické prostředí, jehož optický index lomu n roste s energií fotonu ($dn/d\varepsilon \geq 0$). Fotony jsou emitovány v kuželu pod úhlem θ vzhledem ke směru pohybu částice, pro který platí vztah

$$\cos\theta = \frac{1}{\beta n}, \tag{3}$$

kde $\beta = v/c$ a v je rychlost částice. S klesající energií částice úhel vyzařování θ klesá, frekvence emitovaných fotonů roste, ale klesá jejich počet. Jakmile rychlost částice klesne na rychlost světla v prostředí, úhel vyzařování klesne na nulu a Čerenkovský jev vymizí. Odtud dostaneme dolní mez energie (tedy i frekvence) Čerenkovského fotonu

$$n(\varepsilon_{\min}) = n(\hbar\omega_{\min}) = \frac{1}{\beta}, \tag{4}$$

na druhou stranu emitované fotony s energií větší než je určitá hodnota ε_{max} jsou okamžitě absorbovány materiálem. Interval (ε_{min}, ε_{max}) tvoří spektrální okno, ve kterém dochází k Čerenkovskému vyzařování. Maximální vyzařovaný úhel je

$$\cos\theta_{\max} = \frac{1}{\beta n(\varepsilon_{\max})}. \tag{5}$$

Následkem toho jsou všechny fotony emitovány v kuželu s vrcholovým úhlem θ. Počet fotonů vytvořených na jednotku délky je [3]

$$\frac{dN}{dx} \approx 370 z^2 \left[\varepsilon_{\max} - \varepsilon_{\min} - \frac{1}{\beta^2} \int_{\varepsilon_{\min}}^{\varepsilon_{\max}} \frac{d\varepsilon}{n^2(\varepsilon)} \right]. \tag{6}$$

Pro počet emitovaných fotonů platí Poissonovo rozdělení s průměrem <n>= DelkaKroku·d*N*/d*x*. Statistické rozdělení energie fotonů je pak dáno funkcí

$$f(\varepsilon) = \left[1 - \frac{1}{n^2(\varepsilon)\beta^2} \right]. \tag{7}$$

Polarizace emitovaných fotonů je kolmá k plášti vyzařovacího kužele.

4.1.2 Třída G4Cerenkov

Zářivý tok, spektrum, polarizace a emise Čerenkovova záření je počítána ve funkci AlongStepDoIt objektu třídy G4Cerenkov, která reprezentuje proces Čerenkovské emise. Čas a pozice emitovaného fotonu jsou počítány z veličin známých na začátku kroku trasování nabité částice. V tomto kroku je pohyb částice považován za přímočarý i přes eventuální přítomnost magnetického pole detektoru. Uživatel může určit maximální průměrný počet emitovaných fotonů během kroku použitím funkce G4Cerenkov::SetMaxPhotonsPerStep. Skutečný počet emitovaných fotonů během kroku simulace bude poněkud odlišný ze statistické podstaty emise podle Poissonova rozdělení. Během kroku je hustota emise produkovaných fotonů rozložena rovnoměrně po celém trasovacím kroku pohybu částice, i kdyby částice v tomto intervalu významně měnila svou energii.

Protože je během jednoho kroku emitováno relativně mnoho fotonů (asi 300 fotonů na 1 cm dráhy ve vodě), bylo z důvodu správy paměti v předešlých verzích Geantu nutné zdržet trasování primární částice dokud všechny jí emitované fotony nebyly zcela vytrasovány. Protože Geant od verze 4 používá dynamickou alokaci paměti, není tento krok nutný.

Nicméně, nové verze tuto funkcionalitu podporují použitím funkce G4Cerenkov::SetTrackSecondariesFirst. Následující ukázka demonstruje zaregistrování procesu Čerenkovské emise do simulace:

```
void CMyPhysicsList::ConstructProcess()
{
...
//vytvoříme objekt popisující proces čerenkovské emise
pCerenkovProcess = new G4Cerenkov("Cerenkov");
//nastavíme maximální průměrný počet emitovaných fotonů
pCerenkovProcess->SetMaxNumPhotonsPerStep(300);
//povolíme pozdržení trasování částice, dokud nevytrasujeme
emitované fotony
pCerenkovProcess->SetTrackSecondariesFirst(true);
//iterátor na seznam všech částic registrovaných dříve pro
simulaci
//theParticleIterator poskytuje základní třída
G4VUserPhysicsList
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* pParticle = theParticleIterator-
>value();
G4ProcessManager* pProcManager = pParticle-
>GetProcessManager();
G4String strParticleName = pParticle->GetParticleName();
//může částice vyvolat proces emise čerenkovských fotonů
if(pCerenkovProcess->IsApplicable(*pParticle)) {
//jestliže ano, pak jej pro tuto částici zaregistrujeme
pProcManager->AddContinuousProcess(pCerenkovProcess);
}
}
...
}
```

4.2 Scintilace

Při scintilaci materiálu dochází k izotropní emisi fotonů s náhodně rozloženou lineární polarizací v rovině kolmé ke směru jejich šíření. Důležitou charakteristikou scintilačního materiálu je tzv. scintilační zisk, tedy počet emitovaných fotonů na jednotku energie, a vlastní faktor rozšíření spektra (anglicky resolution scale) scintilátoru, který obecně rozšiřuje statistické rozdělení počtu emitovaných fotonů. Skutečný počet emitovaných fotonů během trasovacího kroku simulace kolísá kolem střední hodnoty v intervalu o šířce

$$\delta n = FaktorRozsireniSpektra \cdot \sqrt{StredniPocetFotonu}\,, \qquad (8)$$

přičemž uvažujeme Poissonovo rozdělení četnosti emitovaných fotonů. Scintilační materiál je dále charakterizován svým fotoemisním spektrem a spektrálním rozložením vyhasínání emise. V Geantu je scintilace rozložena na rychlou a pomalou scintilaci s příslušnou časovou konstantou exponenciálního vyhasínání. Relativní zastoupení rychlé scintilace je dáno jejím ziskovým zastoupením.

Modelování scintilace, Třída G4Scintillation

Abychom korektně mohli zahrnout proces scintilace materiálu do simulace, musíme do tabulky vlastností materiálu zadat empirické hodnoty parametrů vztahujících se ke scintilaci:

- SCINTILATIONYIELD - scintilační zisk,
- RESOLUTIONSCALE - faktor rozšíření spektra,
- FASTTIMECONSTANT - časová konstanta exponenciálního vyhasínání rychlé složky scintilace,
- SLOWTIMECONSTANT - časová konstanta exponenciálního vyhasínání pomalé složky scintilace,
- YIELDRATIO - ziskové zastoupení rychlé složky scintilace,
- FASTCOMPONENT - fotoemisní spektrum rychlé složky scintilace,
- SLOWCOMPONENT - fotoemisní spektrum pomalé složky scintilace.

Následující kód ukazuje implementaci scintilačních vlastností vody:

```
CMyDetectorConstruction::Construct()
{
...
const G4int nEntries = 32;
G4double PhotonEnergy[nEntries] =
{ 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV, 2.177*eV,
2.216*eV, 2.256*eV,
2.298*eV, 2.341*eV, 2.386*eV, 2.433*eV, 2.481*eV, 2.532*eV,
2.585*eV,
2.640*eV, 2.697*eV, 2.757*eV, 2.820*eV, 2.885*eV, 2.954*eV,
3.026*eV,
3.102*eV, 3.181*eV, 3.265*eV, 3.353*eV, 3.446*eV, 3.545*eV,
3.649*eV,
3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
//emisní spektrum rychlé složky scintilace
```

```
G4double ScintilFast[nEntries] =
{ 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00 };
//emisní spektrum pomalé složky scintilace
G4double ScintilSlow[nEntries] =
{ 0.01, 1.00, 2.00, 3.00, 4.00, 5.00, 6.00,
7.00, 8.00, 9.00, 8.00, 7.00, 6.00, 4.00,
3.00, 2.00, 1.00, 0.01, 1.00, 2.00, 3.00,
4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 8.00,
7.00, 6.00, 5.00, 4.00 };
G4MaterialPropertiesTable* pWaterPMT = new
G4MaterialPropertiesTable();
pWaterPMT->AddProperty("FASTCOMPONENT", PhotonEnergy,
ScintilFast, nEntries);
pWaterPMT->AddProperty("SLOWCOMPONENT",PhotonEnergy,
ScintilSlow, nEntries);
pWaterPMT->AddConstProperty("SCINTILLATIONYIELD",50./MeV);
pWaterPMT->AddConstProperty("RESOLUTIONSCALE",1.0);
pWaterPMT->AddConstProperty("FASTTIMECONSTANT", 1.*ns);
pWaterPMT->AddConstProperty("SLOWTIMECONSTANT",10.*ns);
pWaterPMT->AddConstProperty("YIELDRATIO",0.8);
pWater->SetMaterialPropertiesTable(pWaterPMT);
...
}
```

Proces scintilace je řízen objektem třídy G4Scintillation. Konstrukci tohoto objektu provedeme ve členské funkci ConstructProcess třídy G4VUserPhysicsList :

```
void CMyPhysicsList::ConstructProcess()
{
...
//vytvoříme objekt popisující proces Scintilace
pScintillationProcess = new
G4Scintillation("Scintillation");
////povolíme pozdržení ostatní trasování, dokud
nevytrasujeme emitované fotony
pScintillationProcess->SetTrackSecondariesFirst(true);
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* pParticle = theParticleIterator-
>value();
```

```
G4ProcessManager* pProcManager = particle-
>GetProcessManager();
18
G4String strParticleName = pParticle->GetParticleName();
if (pScintillationProcess->IsApplicable(*pParticle)) {
pProcManager->AddProcess(pScintillationProcess);
pProcManager-
>SetProcessOrderingToLast(pScintillationProcess,
idxAtRest);
pProcManager-
>SetProcessOrderingToLast(pScintillationProcess,
idxPostStep);
}
}
...
}
```

4.3 Absorpce fotonu

Implementace absorpce fotonu ve třídě G4OpAbsorption je založena na vyřazení fotonu během trasování na základě známé absorpční délky fotonu v materiálu. Tu je třeba zapsat do proměnné ABSLENGTH tabulky vlastností. Následující ukázka demonstruje zapsání spektrálního rozložení absorpční délky fotonu ve vodě:

```
CMyDetectorConstruction::Construct()
{
...
const G4int nEntries = 32;
G4double PhotonEnergy[nEntries] =
{ 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV, 2.177*eV,
2.216*eV, 2.256*eV,
2.298*eV, 2.341*eV, 2.386*eV, 2.433*eV, 2.481*eV, 2.532*eV,
2.585*eV,
2.640*eV, 2.697*eV, 2.757*eV, 2.820*eV, 2.885*eV, 2.954*eV,
3.026*eV,
3.102*eV, 3.181*eV, 3.265*eV, 3.353*eV, 3.446*eV, 3.545*eV,
3.649*eV,
3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
//spektrální závislost absorpční délky na energii fotonu
G4double Absorption[nEntries] =
{ 3.448*m, 4.082*m, 6.329*m, 9.174*m, 12.346*m, 13.889*m,
15.152*m,
17.241*m, 18.868*m, 20.000*m, 26.316*m, 35.714*m, 45.455*m,
47.619*m,
```

```
52.632*m, 52.632*m, 55.556*m, 52.632*m, 52.632*m, 47.619*m,
45.455*m,
41.667*m, 37.037*m, 33.333*m, 30.000*m, 28.500*m, 27.000*m,
24.500*m,
22.000*m, 19.500*m, 17.500*m, 14.500*m };
G4MaterialPropertiesTable* pWaterPMT = new
G4MaterialPropertiesTable();
pWaterPMT->AddProperty("ABSLENGTH", PhotonEnergy,
Absorption, nEntries);
pWater->SetMaterialPropertiesTable(pWaterPMT);
...
}
```

Proces absorpce fotonů registrujeme samozřejmě pouze pro foton:

```
void CMyPhysicsList::ConstructProcess()
{
...
//vytvoříme objekt popisující proces absorpce fotonu
G4OpAbsorption* pAbsorptionProcess = new G4OpAbsorption();
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* pParticle = theParticleIterator-
>value();
G4ProcessManager* pProcManager = particle-
>GetProcessManager();
G4String strParticleName = pParticle->GetParticleName();
//jestliže je částice optický foton zaregistrujeme proces
jeho absorpce
if (strParticleName == "opticalphoton") {
pmanager->AddDiscreteProcess(pAbsorptionProcess);
}
}
...
}
```

4.4 Rayleighův rozptyl

4.4.1 Fyzikální základ

Energetická závislost účinného průřezu Rayleighova rozptylu je, stejně jako u ostatních relevantních nízkoenergetických fyzikálních procesů, odvozena z empiricky získaných dat uložených v databázi Geantu (viz kapitola 2.2.2). Celkový účinný průřez je vypočítán podle aproximační rov-

nice (1). Úhel θ koherentně rozptýleného fotonu je trasován podle rozdělení polarizace fotonu

$$\Phi(E,\theta) = \left(1+\cos^2\theta\right)\sin\theta \cdot FF^2(q), \qquad (9)$$

kde *FF* je Hubbelův faktor a $q=2E\sin(\theta/2)$ je změna hybnosti částice. Hubbelův faktor, jehož empirické hodnoty jsou uloženy v databázi, zahrnuje do výpočtu počáteční energii fotonu, která není zahrnuta do Rayleighova vztahu. Při nízkých energiích je Hubbelův faktor izotropický a neovlivňuje úhlové rozdělení rozptýleného fotonu, zatímco při vysokých energiích formuje úhlovou distribuci do několika význačných směrů (píků). Proto se tento faktor nazývá formovací faktor (odtud označení FF).

4.4.2 Konečný stav fotonu, třída G4OpRayleigh

Třída G4OpRayleigh implementuje proces Rayleighova rozptylu fotonu v materiálu. Při výpočtu dráhy fotonu trasuje jeho úhel θ s podmínkou, že nový směr pohybu je kolmý k nově vypočítané polarizaci fotonu a nový směr pohybu, počáteční a koncová polarizace leží v jedné rovině. Aby se foton mohl účastnit procesu Rayleighova rozptylu, musí mu být přiřazen polarizační stav (spin). Polarizace fotonu je datovým členem třídy G4DynamicParticle a nastavuje se voláním její členské funkce SetPolarization (například při modelování emise Čerenkovským procesem nebo scintilací) nebo nepřímo funkcí G4ParticleGun:: SetParticlePolarization pokud je foton primární částicí simulace. Pro simulaci je nutné znát střední volnou dráhu fotonu pro Rayleighův rozptyl. Potřebný údaj můžeme uživatelsky vložit do tabulky vlastností pod proměnnou RAYLEIGH. V opačném případě bude použito Einstein-Smolochowského vztahu pro výpočet tohoto parametru použitím členské funkce RayleighAttenuationLengthGenerator třídy G4OpRayleigh. Následuje ukázka registrace procesu Rayleighova rozptylu fotonu:

```
void CMyPhysicsList::ConstructProcess()
{
...
//vytvoříme objekt popisující proces Rayleighova rozptylu
G4OpRayleigh* pRayleighProcess = new G4OpRayleigh();
theParticleIterator->reset();
while( (*theParticleIterator)() ){
```

```
G4ParticleDefinition* pParticle = theParticleIterator-
>value();
G4ProcessManager* pProcManager = particle-
>GetProcessManager();
G4String strParticleName = pParticle->GetParticleName();
//proces Rayleighova rozptylu zaregistrujeme pouze pro
foton
if (strParticleName == "opticalphoton") {
pmanager->AddDiscreteProcess(pRayleighProcess);
}
}
...
}
```

4.5 Optické procesy na rozhraní dvou prostředí

Průchodem světla rozhraním dvou prostředí s různým indexem lomu, dochází k lomu, odrazu nebo absorpci světla. Pro účely modelování rozlišujeme tři druhy přechodu fotonů přes rozhraní:

- přechod dielektrikum → dielektrikum - foton může být odražen zpět nebo prochází do druhého prostředí,
- přechod dielektrikum → kov - foton může být pohlcen kovem nebo odražen zpět,
- přechod dielektrikum → černé prostředí - černé prostředí je v Geantu definováno jako prostředí bez definovaných optických vlastností; v takovém případě je foton pohlcen bez jakékoliv detekce nebo vyvolání fyzikálního procesu.

Při modelování průchodu fotonu rozhraním dielektrikum → kov se vychází z odraznosti povrchu kovu, kterou je třeba zadat uživatelsky. Odtud se vypočítá pravděpodobnost absorpce fotonu materiálem.

4.5.1 Průchod fotonu rozhraním dvou dielektrik

Při průchodu světla mezi rozhraním dvou dielektrik dochází k částečnému lomu a odrazu světla. Označme E elektrickou složku dopadající vlny, E' elektrickou složku lomené vlny a E'' elektrickou složku odražené vlny. Jestliže uvažujeme rovinné vlnění, platí

$$\vec{E} = \vec{A}\exp\left(i\vec{k}\vec{x} - i\omega t\right), \tag{10}$$

$$\vec{E}' = \vec{A}'\exp\left(i\vec{k}'\vec{x} - i\omega t\right), \tag{11}$$

$$\vec{E}'' = \vec{A}'' \exp\left(i\vec{k}''\vec{x} - i\omega t\right), \tag{12}$$

kde *k*, *k*‘ a *k*‘‘ jsou příslušné vlnové vektory. Pro všechny tři vlny platí na rozhraní Fresnelův zákon

$$\vec{k}\vec{x} = \vec{k}'\vec{x} = \vec{k}''\vec{x}\,, \tag{13}$$

nebo-li

$$\vec{k}\sin\alpha = \vec{k}'\sin\alpha = \vec{k}''\sin\alpha\,, \tag{14}$$

kde α, α‘ a α‘‘ jsou úhly dopadu, lomu a odrazu. Odtud plynou známé zákony odrazu a lomu

$$\alpha'' = \alpha \wedge n\sin\alpha = n'\sin\alpha'\,. \tag{15}$$

Pro velikost amplitud platí odvozením z Maxwellových rovnic známe Fresnelovy vzorce:

- šíření vlny v sagitální rovině

$$\frac{A'_{sag}}{A} = \frac{2n\cos\alpha}{n\cos\alpha + \frac{\mu}{\mu'}n'\cos\alpha'}, \tag{16}$$

$$\frac{A''_{sag}}{A} = \frac{n\cos\alpha - \frac{\mu}{\mu'}n'\cos\alpha'}{n\cos\alpha + \frac{\mu}{\mu'}n'\cos\alpha'}, \tag{17}$$

- šíření vlny v tangenciální rovině

$$\frac{A'_{\tan}}{A} = \frac{2n\cos\alpha}{\frac{\mu}{\mu'}n'\cos\alpha + n\cos\alpha'}, \tag{18}$$

$$\frac{A''_{\tan}}{A} = \frac{\frac{\mu}{\mu'}n'\cos\alpha - n\cos\alpha'}{\frac{\mu}{\mu'}n'\cos\alpha + n\cos\alpha'}, \tag{19}$$

kde μ a μ‘ jsou magnetické permeability obou prostředí.
Protože Geant popisuje šíření světla částicově pomocí fotonů, musí se vypočítat pravděpodobnost lomu nebo odrazu fotonu. Pravděpodobnost, že foton projde do druhého prostředí je rovna propustnosti rozhraní:

$$T = \frac{\mu'}{\mu}\left(\frac{A'}{A}\right)^2 \frac{n'\cos\alpha'}{n\cos\alpha}, \tag{20}$$

a odpovídající pravděpodobnost, že bude foton odražen zpět je $R = 1 - T$. Označme *u* normálový vektor k povrchu rozhraní. Pro polarizační vektory *e*, *e*‘ a *e*‘‘ přicházejícího, průchozího a odraženého fotonu platí:

- pro dopadající foton

$$\vec{q} = \vec{k} \times \vec{q}, \tag{21}$$

$$\vec{e}_{sag} = \frac{\vec{e}\vec{q}}{|\vec{q}|}\frac{\vec{q}}{|\vec{q}|}, \tag{22}$$

$$\vec{e}_{\tan} = \vec{e} - \vec{e}_{sag}, \tag{23}$$

- pro průchozí foton

$$\vec{e}'_{sag} = \vec{e}_{sag}\frac{2n\cos\alpha}{n\cos\alpha + \frac{\mu}{\mu'}n'\cos\alpha'}, \tag{24}$$

$$\vec{e}'_{\tan} = \vec{e}_{\tan}\frac{2n\cos\alpha}{\frac{\mu}{\mu'}n'\cos\alpha + n\cos\alpha'}, \tag{25}$$

- pro odražený foton

$$\vec{e}''_{sag} = \vec{e}'_{sag} - \vec{e}_{sag}, \tag{26}$$

$$\vec{e}''_{\tan} = \frac{\mu}{\mu'}\frac{n'}{n}\vec{e}'_{\tan} - \vec{e}_{\tan}. \tag{27}$$

4.5.2 Model povrchu UNIFIED, koncept povrchu materiálu

Geant4 poskytuje pro optické procesy koncept povrchu materiálu označený surfaces. V tomto konceptu jsou informace potřebné pro popis povrchu rozděleny do dvou kategorií: informace o materiálových vlastnostech samotného povrchu a informace o odpovídajících logických a fyzických objemech mezi nimiž se povrch nachází. Koncept zavádí tzv. hraniční povrch (border surface) a plášťový povrch (skin surface).

Hraniční povrch, reprezentovaný třídou G4LogicalBorderSurface, je specifikovaný párem fyzických objemů dotýkajících se povrchu, přičemž se rozlišuje pořadí mezi objemy, takže uživatel může specifikovat odlišné optické vlastnosti povrchu pro fotony přicházející z opačné strany. Plášťový povrch, reprezentovaný třídou G4LogicalSkinSurface, je charakterizovaný logickým objemem, který je zcela obklopený tímto povrchem. Tento typ povrchu je použitelný v situacích, kdy je daný objem obklopen odrazným materiálem a je vložen do mnoha různých matečních objemů. Příkladem je logický objem naplněný vzduchem.
Objekt vytvořený třídou G4LogicalBorderSurface nebo G4LogicalSkinSurface se nazývá objekt fyzického povrchu. Pro specifikaci povrchu je nutné tomuto objektu předat ukazatel na objekt třídy typu G4SurfaceProperty, který implementuje potřebný model povrchu. Pro optické procesy předáváme objektu fyzického povrchu ukazatel na objekt třídy G4OpticalSurface odvozené od G4SurfaceProperty.

Třída G4OpticalSurface implementuje model optického povrchu. V této části se zmíníme o modelu UNIFIED [14]. Model UNIFIED se aplikuje na rozhraní dielektrikum-dielektrikum a snaží se poskytnout realistický model optického povrchu. Můžeme jím modelovat jemné povrchy pokryté filmem z kovu (zrcadlový povrch), nebo povrchy pokryté difúzně odrazným materiálem a můžeme též modelovat povrch tvořený mikrofazetami s normálovými vektory, které sledují různé statistické rozložení směru kolem normály v bodě dopadu. Dále model UNIFIED umožňuje modelovat hrubé povrchy, kde odražený foton může opět dopadnout na tentýž povrch. Model zavádí některé parametry, které specifikují typ povrchu. V Geantu mají následující označení pro zadání do tabulky vlastností materiálu:

- SPECULARLOBECONSTANT - pravděpodobnost odrazu ve směru normály mikrofazety,
- SPECULARSPIKECONSTANT - pravděpodobnost odrazu ve směru střední normály celého povrchu,
- BACKSCATTERCONSTANT - pravděpodobnost odrazu uvnitř hluboké rýhy se zpětným rozptylem (back-scattering),
- DIFFUSELOBECONSTANT - pravděpodobnost odrazu na difúzně odrazném povrchu.

Pro uplatnění modelu UNIFIED v Geantu je třeba zadat první tři parametry, poslední parametr je možné nechat implicitně nastavený Geantem. Následuje ukázka vytvoření povrchu vody:

```
CMyDetectorConstruction::Construct()
{
...
//vytvoření optického povrchu vody
G4OpticalSurface* pOpWaterSurface = new
G4OpticalSurface("WaterSurface");
pOpWaterSurface->SetType(dielectric_dielectric);
pOpWaterSurface->SetFinish(polished);
pOpWaterSurface->SetModel(unified);
//vytvoření hraničního povrchu
G4LogicalBorderSurface* pWaterSurface =
new G4LogicalBorderSurface("WaterSurface",
waterTank_phys,expHall_phys,pOpWaterSurface);
//zadaní parametrů pro model UNIFIED
const G4int num = 2;
G4double Ephoton[num] = {2.038*eV, 4.144*eV};
G4double SpecularLobe[num] = {0.3, 0.3};
G4double SpecularSpike[num] = {0.2, 0.2};
G4double Backscatter[num] = {0.2, 0.2};
//zápis do tabulky materiálových vlastností vody
G4MaterialPropertiesTable* pWaterMST = new
G4MaterialPropertiesTable();
pWaterMST->AddProperty("SPECULARLOBECONSTANT", Ephoton,
SpecularLobe, num);
pWaterMST->AddProperty("SPECULARSPIKECONSTANT", Ephoton,
SpecularSpike, num);
pWaterMST->AddProperty("BACKSCATTERCONSTANT", Ephoton,
Backscatter, num);
pOpWaterSurface->SetMaterialPropertiesTable(pWaterMST);
...
}
```

4.5.3 Třída G4BoundaryProcess

Během trasování pohybu fotonu je sledován jeho dopad na rozhraní dvou prostředí. Pokud k tomu dojde, objekt třídy G4BoundaryProcess vyhledá v tabulce povrchů, kterou spravuje jádro Geantu, příslušný objekt fyzického povrchu a podle jeho informací aplikuje optický proces lomu, odrazu nebo absorpce. Objekt třídy G4BoundaryProcess opět registrujeme pouze pro foton:

```
void CMyPhysicsList::ConstructProcess()
{
...
//vytvoříme objekt popisující optické procesy na rozhraní
dvou materiálů
pBoundaryProcess = new G4OpBoundaryProcess();
//nastavíme model UNIFIED
G4OpticalSurfaceModel pSurfaceModel = unified;
pBoundaryProcess->SetModel(pSurfacemodel);
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* pParticle = theParticleIterator-
>value();
G4ProcessManager* pProcManager = particle-
>GetProcessManager();
G4String strParticleName = pParticle->GetParticleName();
//pBoundaryProcess zaregistrujeme pouze pro foton
if (strParticleName == "opticalphoton") {
pmanager->AddDiscreteProcess(pBoundaryProcess);
}
}
...
}
```

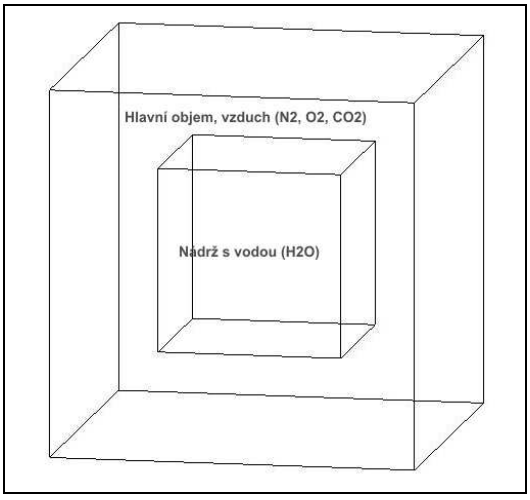
5 Příklad modelování Geantem - program WaterTank

Program WaterTank, jehož výpis zdrojového kódu nalezneme v příloze, představuje jednoduchý simulační program vytvořený pomocí knihoven Geantu, který modeluje proces emise Čerenkovského a scintilačního záření průchodem elektronu nádrží naplněnou demineralizovanou vodou. Modelování bylo zaměřeno na výpočet závislosti zisku obou typů emise na energii elektronu a rozložení směru pohybu emitovaných fotonů a rozdělení jejich vektorů polarizace.

5.1 Příprava simulace

5.1.1 Konstrukce detektoru

Uspořádání detektoru je jednoduché, Obrázek 2. Ve funkci CWTDetectorConstruction::Construct() definujeme hlavní objem (svět) vyplněný vzduchem, do něhož je umístěna krychlová nádrž o rozměru strany 5 m naplněná demineralizovanou vodou. Při konstrukci bylo potřeba zadat do tabulky vlastností vody a vzduchu patřičné hodnoty parametrů pro účely simulace absorpce fotonu, scintilace a lomu světla na rozhraní voda-vzduch (seznam parametrů pro každý proces lze nalézt v příslušné části předchozí kapitoly).



Obrázek 2 – Konstrukce detektoru v programu WaterTank.

5.1.2 Modelované fyzikální procesy a zúčastněné částice

Do modelování byly zahrnuty standardně tyto elementární částice:

- leptony: e−, e+, ve (elektronové neutrino), ˜ve, μ+ (mion), μ−, νμ, ˜νμ,
- bozony: π+ (pion), π−, π0,
- baryony: p, ˜p, n, ˜n.
- bosony: γ, optický foton (nízkoenergetické γ).

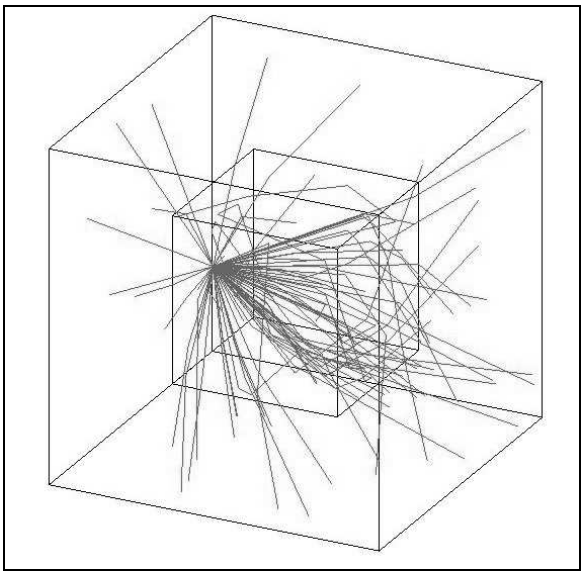
Částice jsou registrovány ve funkci CWTPhysicsList::ConstructParticle. Pro každou částici jsou přiřazeny fyzikální procesy, kterých se daná částice může zúčastnit. Vedle elektromagnetických interakcí registrovaných ve funkci CWTPhysicsList::ConstructEM a rozpadových procesů ve funkci CWTPhysicsList::ConstructGeneral jsou ve funkci CWTPhysicsList::ConstructOp registrovány procesy emise Čerenkovského a scintilačního záření. Pro nízkoenergetické fotony (optické fotony) jsou registrovány procesy absorpce, Rayleighova rozptylu a procesy na rozhraní dvou dielektrik.

5.1.3 Primární částice

Primární částice, elektron, je definována v konstruktoru třídy CWTPrimaryGeneratorAction. Počátek trajektorie částice je na souřadnici (-5 m, 0 m, 0 m) tedy na jedné ze stěn nádrže a počáteční směr je (1, 0, 0) tedy v ose x ve směru do středu detektoru. Energie elektronu byla během simulace od 100 keV do 1 Mev po 50 keV.

5.2 Výsledky simulace

Cílem modelování v programu WaterTank bylo vypočítat zisk (výtěžnost) emise Čerenkovského a scintilačního záření při průchodu vysokoenergetického elektronu demineralizovanou vodou. K zajištění dostatečné statistiky dat byla simulace pro každou zadanou energii elektronu opakována 500x a výsledky simulace byly postupně uloženy v datových souborech pro další zpracování. Pro doplnění byla vytvořena vizualizace procesu modelování při průchodu elektronu o energii 500 keV, Obrázek 3. Z obrázku, ale především ze záznamu simulace, lze zjistit, že elektron ztratil značnou část své počáteční energie již v prvním kroku simulace a dostal se tak pod prahovou hodnotu energie (kterou lze uživatelsky nastavit), pod kterou už není uvažován v dalších krocích simulace (je ”zničen”).

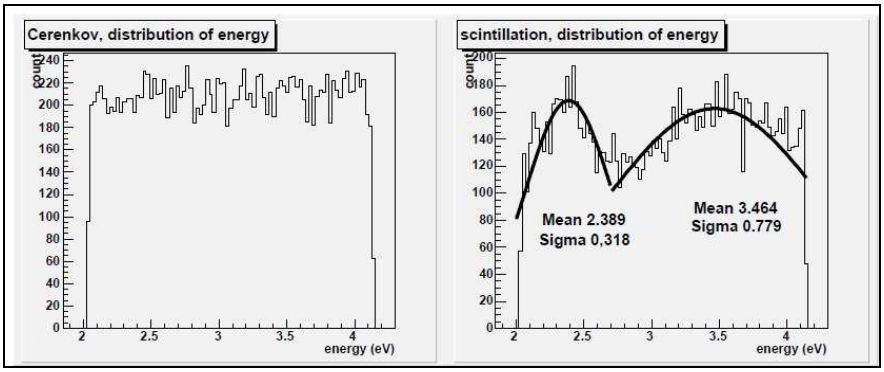


Obrázek 3 – Vizualizace emise záření při průchodu elektronu o 500 keV demineralizovanou vodou.

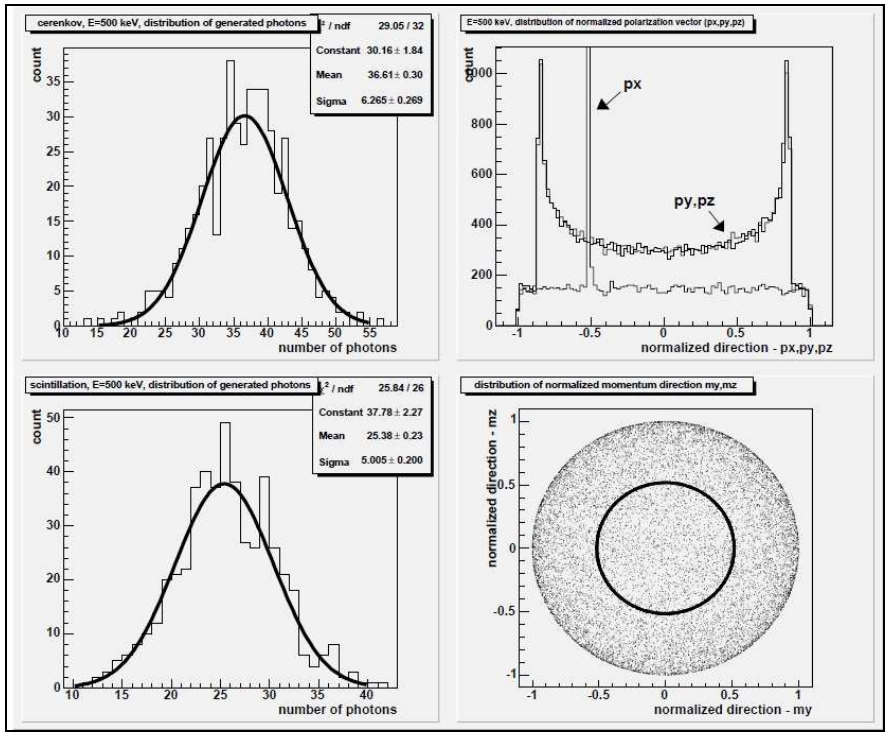
Na Obrázcích 4 a 5 jsou znázorněny výsledky simulace při energii elektronu 500 keV. Rozdělení energie Čerenkovských fotonů, obrázek 4, je rovnoměrné v intervalu od 2 eV do 4,15 eV. Rozdělení energie scintilačně emitovaných fotonů na stejném intervalu vykazuje 2 maxima pro energii 2,39 eV a 3,64 eV. Na druhém obrázku je pak znázorněno statistické rozdělení četnosti emitovaných fotonů při Čerenkovské a scintilační emisi, rozdělení polarizace všech emitovaných fotonů a rozdělení jejich hybnosti. Čerenkovské fotony vykazují velkou anizotropii hybnosti a polarizačního vektoru ve směru osy x, tedy ve směru pohybu elektronu (složka p_x polarizačního vektoru). Scintilační emise je naproti tomu izotropní. Kombinovaný histogram rozdělení složek *my* a *mz* hybnosti fotonů názorně ukazuje izotropní charakter scintilační emise a prstenec odpovídá vyzařovacímu kuželu Čerenkovské emise.

Na Obrázku 6 jsou pro srovnání vykresleny výsledky simulace pro energie elektronu 500 keV, 700 keV a 950 keV. S rostoucí energií roste počet Čerenkovsky a scintilačně emitovaných fotonů, ale zároveň se v obou případech rozšiřuje interval tohoto počtu (šířka rozdělení). Pravý horní histogram ukazuje kombinované rozdělení složek hybnosti *my* a *mz* pro Čerenkovskou emisi. V histogramu jsou zřetelně vidět charakteristické prstence pro každou energii. Izotropně rozdělený ”oblak” odpovídá

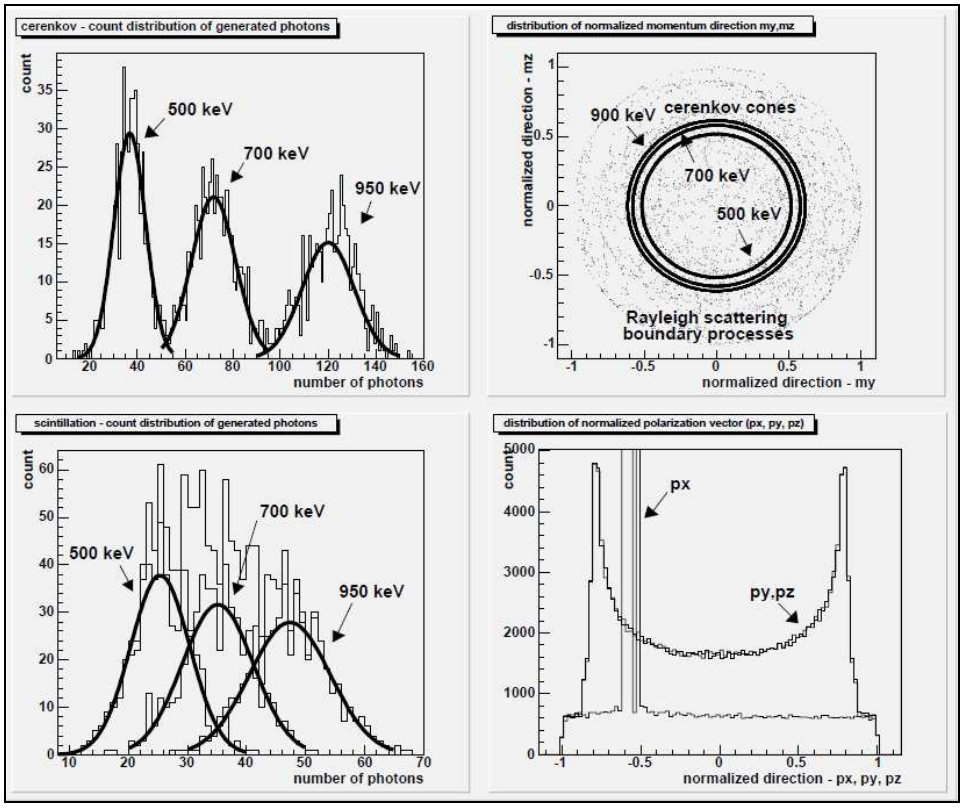
fotonům, které jsou v simulačním kroku jejich emise navíc podrobeny Rayleighově rozptylu.
Poslední obrázek 7 ukazuje energetickou závislost zisku (výtěžnosti) Čerenkovské a scintilační emise. Čerenkovská emise nastala až při energii elektronu 250 keV.



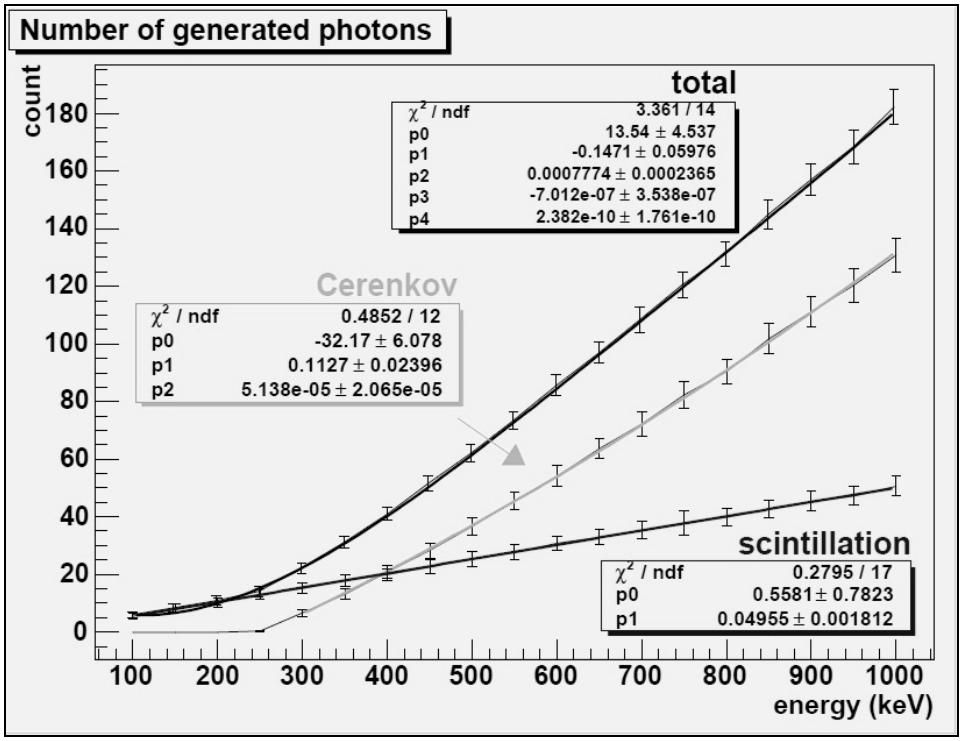
Obrázek 4 – Rozdělení energie emitovaného záření při energii elektronu 500 keV.



Obrázek 5 – Rozdělení četnosti, hybnosti a polarizace pro energii elektronu 500 keV.



Obrázek 6 – Srovnání výsledků simulace pro energii elektronu 500 keV, 700 keV a 950 keV



Obrázek 7 – Srovnání výsledků simulace pro energii elektronu 500 keV, 700 keV a 950 keV.

Literatura

[1] J. Allison for Geant4 Collaboration, "Geant4 - A Simulation Toolkit," Nucl. Instrum. Meth. A, 506, 250-303 (2003).

[2] NOZKA, L. – PECH, M. – HIKLOVA, H. – MANDAT, M. – HRABOVSKY, M. – SCHOVANEK, P. – PALATKA, M. BRDF profile of Tyvek and its implementation in the Geant4 simulation toolkit, *Optics Express*, 19 (5), 2011.

[3] Physics ReferenceManual, http://geant4.web.cern.ch/geant4/G4Users-Documents/UsersGuides/PhysicsReferenceManual/html/PhysicsReferenceManual.html

[4] CULLEN D., HUBBELL J.H., KISSEL L.: EPDL97: the Evaluated Photon Data Library, '97 version, UCRL 50400, Vol.6, Rev.5

[5] http://www.nea.fr/html/dbdata/nds evaluated.htm

[6] PERKINS S.T., CULLEN D.E., SELTZER S.M.: Tables and Graphs of Electron-Interaction Cross-Sections from 10 eV to 100 GeV Derived from the LLNL Evaluated Electron Data Library (EEDL), Z=1-100, UCRL-50400, Vol.31

[7] PERKINS S.T., CULLEN D.E., CHEN M.H., HUBBELL J.H., RATHKOPF J., SCOFIELD J.: Tables and Graphs of Atomic Subshell and Relaxation Data Derived from the LLNL Evaluated Atomic Data Library (EADL), Z=1-100, UCRL-50400, Vol.30

[8] ANDERSEN H.H., ZIEGLER J.F.: The Stopping and Ranges of Ions in Matter. Vol.3, Pergamon Press, 1977.

[9] ZIEGLER J.F.: The Stopping and Ranges of Ions in Matter. Vol.4, Pergamon Press, 1977.

[10] ZIEGLER J.F., BIERSACK J.P., LITTMAN U.: The Stopping and Ranges of Ions in Solids. Vol.1, Pergamon Press, 1985.

[11] ALLISY A. et al (ICRU): Stopping Powers and Ranges for Protons and Alpha Particles, ICRU Report 49, 1993.

[12] SCOFIELD J.H.: Radiative Transitions, in Atomic Inner-Shell Processes, B.Crasemann ed., Academic Press, New York, 1975, pp.265-292.

[13] Geant4 User's Guide For Applications Developers, http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/-ForApplicationDeveloper/html/

[14] MOISAN C., LEVIN A.: A More Physical Approach to Model the Surface Treatment of Scintillation Counters and its Implementation into DETECT, TRIUMF Preprint TRI-PP-96-64, Oct. 1996.

PŘÍLOHA

Zdrojový kód programu *WaterTank*

watertank.cpp

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "G4UIterminal.hh"
#include "G4UItcsh.hh"
#include "G4ios.hh"
#include <stdlib.h>
#include "WTDetectorConstruction.hh"
#include "WTPhysicsList.hh"
#include "WTPrimaryGeneratorAction.hh"
#include "WTRunAction.hh"
#include "WTEventAction.hh"
#include "WTStackingAction.hh"
#include "WTSteppingVerbose.hh"
#ifdef G4UI_USE_XM
#include "G4UIXm.hh"
#endif //G4UI_USE_XM
#ifdef G4VIS_USE
#include "WTVisManager.hh"
#endif
#define WORKDIR "/home/libor/documents/data/geantsim/electron/scan_100-
1000keV"
G4VProcess* g_pCerenkov;
G4VProcess* g_pScintillation;
G4String g_strDataFile;
G4String g_strDataFileCer;
G4String g_strDataFileSci;
G4String g_strProcStatFile;
int main(int argc,char** argv) {
if(argc==3){
char szbuff[512];
sprintf(szbuff,"%s/E%s_tot.txt",WORKDIR,argv[2]);
g_strDataFile=szbuff;
sprintf(szbuff,"%s/E%s_cer.txt",WORKDIR,argv[2]);
g_strDataFileCer=szbuff;
sprintf(szbuff,"%s/E%s_sci.txt",WORKDIR,argv[2]);
g_strDataFileSci=szbuff;
sprintf(szbuff,"%s/E%s_ps.txt",WORKDIR,argv[2]);
g_strProcStatFile=szbuff;
}
else{
g_strDataFile="/home/libor/documents/temp/ex06data_tot.txt";
g_strDataFileCer="/home/libor/documents/temp/ex06data_cer.txt";
g_strDataFileSci="/home/libor/documents/temp/ex06data_sci.txt";
g_strProcStatFile="/home/libor/documents/temp/ex06data_ps.txt";
}
G4cout<<"data output: "<<g_strDataFile<<G4endl;
G4cout<<"proc statistics: "<<g_strProcStatFile<<G4endl;
// Seed the random number generator manually
G4long myseed = 345354;
```

```
HepRandom::setTheSeed(myseed);
g_pCerenkov=g_pScintillation=NULL;
//my Verbose output class
G4VSteppingVerbose::SetInstance(new CWTSteppingVerbose);
// Run manager
G4RunManager* runManager = new G4RunManager;
// UserInitialization classes - mandatory
runManager-> SetUserInitialization(new CWTDetectorConstruction);
runManager-> SetUserInitialization(new CWTPhysicsList);
#ifdef G4VIS_USE
watertank.cpp 2
// visualization manager
G4VisManager* visManager = new CWTVisManager;
visManager->Initialize();
#endif
// UserAction classes
runManager->SetUserAction(new CWTRunAction);
runManager->SetUserAction(new CWTPrimaryGeneratorAction);
runManager->SetUserAction(new CWTEventAction);
runManager->SetUserAction(new CWTStackingAction);
//Initialize G4 kernel
runManager->Initialize();
// get the pointer to the User Interface manager
G4UImanager* UI = G4UImanager::GetUIpointer();
if (argc==1) //define UI session for interactive mode
{
G4UIsession* session = 0;
#ifdef G4UI_USE_XM
session=new G4UIXm(argc,argv);
#else
#ifdef G4UI_USE_TCSH
session = new G4UIterminal(new G4UItcsh);
#else
session = new G4UIterminal();
#endif
#endif
UI->ApplyCommand("/control/execute vis.mac");
session->SessionStart();
delete session;
}
else // Batch mode
{
G4String command = "/control/execute ";
G4String fileName = argv[1];
UI->ApplyCommand(command+fileName);
}
// job termination
#ifdef G4VIS_USE
delete visManager;
#endif
delete runManager;
return 0;
}
```

WTDetectorConstruction.h

```
#ifndef WTDetectorConstruction_h
#define WTDetectorConstruction_h 1
#include "globals.hh"
```

```
#include "G4VUserDetectorConstruction.hh"
class CWTDetectorConstruction : public G4VUserDetectorConstruction
{
public:
CWTDetectorConstruction();
~CWTDetectorConstruction();
public:
G4VPhysicalVolume* Construct();
private:
G4double expHall_x;
G4double expHall_y;
G4double expHall_z;
G4double tank_x;
G4double tank_y;
G4double tank_z;
G4double bubble_x;
G4double bubble_y;
G4double bubble_z;
};
#endif /*WTDetectorConstruction_h*/
```

WTDetectorConstruction.cpp

```
#include "CWTDetectorConstruction.hh"
#include "G4Material.hh"
#include "G4MaterialTable.hh"
#include "G4Element.hh"
#include "G4ElementTable.hh"
#include "G4LogicalBorderSurface.hh"
#include "G4LogicalSkinSurface.hh"
#include "G4Box.hh"
#include "G4LogicalVolume.hh"
#include "G4RotationMatrix.hh"
#include "G4ThreeVector.hh"
#include "G4Transform3D.hh"
#include "G4PVPlacement.hh"
#include "G4OpBoundaryProcess.hh"
#include "G4Sphere.hh"
CWTDetectorConstruction::CWTDetectorConstruction()
{
expHall_x = expHall_y = expHall_z = 10*m;
tank_x = tank_y = tank_z = 5*m;
bubble_x = bubble_y = bubble_z = 0.5*m;
}
CWTDetectorConstruction::~CWTDetectorConstruction(){;}
G4VPhysicalVolume* CWTDetectorConstruction::Construct()
{
// ------------- Materials -------------
G4double a, z, density;
G4int nelements;
// Air
//
G4Element* N = new G4Element("Nitrogen", "N", z=7 , a=14.01*g/mole);
G4Element* O = new G4Element("Oxygen" , "O", z=8 , a=16.00*g/mole);
G4Material* Air = new G4Material("Air", density=1.29*mg/cm3, nelements=2);
Air->AddElement(N, 70.*perCent);
Air->AddElement(O, 30.*perCent);
// Water
```

```
//
G4Element* H = new G4Element("Hydrogen", "H", z=1 , a=1.01*g/mole);
G4Material* Water = new G4Material("Water", density= 1.0*g/cm3,
nelements=2);
Water->AddElement(H, 2);
Water->AddElement(O, 1);
//
// ------------ Generate & Add Material Properties Table ------------
//
const G4int nEntries = 32;
G4double PhotonEnergy[nEntries] =
{ 2.034*eV, 2.068*eV, 2.103*eV, 2.139*eV,
2.177*eV, 2.216*eV, 2.256*eV, 2.298*eV,
2.341*eV, 2.386*eV, 2.433*eV, 2.481*eV,
2.532*eV, 2.585*eV, 2.640*eV, 2.697*eV,
2.757*eV, 2.820*eV, 2.885*eV, 2.954*eV,
3.026*eV, 3.102*eV, 3.181*eV, 3.265*eV,
3.353*eV, 3.446*eV, 3.545*eV, 3.649*eV,
3.760*eV, 3.877*eV, 4.002*eV, 4.136*eV };
//
// Water
//
G4double RefractiveIndex1[nEntries] =
{ 1.3435, 1.344, 1.3445, 1.345, 1.3455,
1.346, 1.3465, 1.347, 1.3475, 1.348,
1.3485, 1.3492, 1.35, 1.3505, 1.351,
1.3518, 1.3522, 1.3530, 1.3535, 1.354,
1.3545, 1.355, 1.3555, 1.356, 1.3568,
1.3572, 1.358, 1.3585, 1.359, 1.3595,
1.36, 1.3608};

G4double Absorption1[nEntries] =
{3.448*m, 4.082*m, 6.329*m, 9.174*m, 12.346*m, 13.889*m,
15.152*m, 17.241*m, 18.868*m, 20.000*m, 26.316*m, 35.714*m,
45.455*m, 47.619*m, 52.632*m, 52.632*m, 55.556*m, 52.632*m,
52.632*m, 47.619*m, 45.455*m, 41.667*m, 37.037*m, 33.333*m,
30.000*m, 28.500*m, 27.000*m, 24.500*m, 22.000*m, 19.500*m,
17.500*m, 14.500*m };
G4double ScintilFast[nEntries] =
{ 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00 };
G4double ScintilSlow[nEntries] =
{ 0.01, 1.00, 2.00, 3.00, 4.00, 5.00, 6.00,
7.00, 8.00, 9.00, 8.00, 7.00, 6.00, 4.00,
3.00, 2.00, 1.00, 0.01, 1.00, 2.00, 3.00,
4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 8.00,
7.00, 6.00, 5.00, 4.00 };
G4MaterialPropertiesTable* myMPT1 = new G4MaterialPropertiesTable();
myMPT1->AddProperty("RINDEX", PhotonEnergy, RefractiveIndex1,nEntries);
myMPT1->AddProperty("ABSLENGTH", PhotonEnergy, Absorption1, nEntries);
myMPT1->AddProperty("FASTCOMPONENT",PhotonEnergy, ScintilFast, nEntries);
myMPT1->AddProperty("SLOWCOMPONENT",PhotonEnergy, ScintilSlow, nEntries);
myMPT1->AddConstProperty("SCINTILLATIONYIELD",50./MeV);
myMPT1->AddConstProperty("RESOLUTIONSCALE",1.0);
myMPT1->AddConstProperty("FASTTIMECONSTANT", 1.*ns);
myMPT1->AddConstProperty("SLOWTIMECONSTANT",10.*ns);
```

```
myMPT1->AddConstProperty("YIELDRATIO",0.8);
Water->SetMaterialPropertiesTable(myMPT1);
//
// Air
//
G4double RefractiveIndex2[nEntries] =
{ 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00,
1.00, 1.00, 1.00, 1.00 };
G4MaterialPropertiesTable* myMPT2 = new G4MaterialPropertiesTable();
myMPT2->AddProperty("RINDEX", PhotonEnergy, RefractiveIndex2, nEntries);
Air->SetMaterialPropertiesTable(myMPT2);
//
// ------------- Volumes --------------
// The experimental Hall
//
G4Box* expHall_box = new G4Box("World",expHall_x,expHall_y,expHall_z);
G4LogicalVolume* expHall_log
= new G4LogicalVolume(expHall_box,Air,"World",0,0,0);
G4VPhysicalVolume* expHall_phys
= new G4PVPlacement(0,G4ThreeVector(),expHall_log,"World",0,false,0);
// The Water Tank
//
G4Box* waterTank_box = new G4Box("Tank",tank_x,tank_y,tank_z);
G4LogicalVolume* waterTank_log
= new G4LogicalVolume(waterTank_box,Water,"Tank",0,0,0);
G4VPhysicalVolume* waterTank_phys
= new G4PVPlacement(0,G4ThreeVector(),waterTank_log,"Tank",
expHall_log,false,0);
// ------------- Surfaces --------------
//
// Water Tank
//
G4OpticalSurface* OpWaterSurface = new G4OpticalSurface("WaterSurface");
OpWaterSurface->SetType(dielectric_dielectric);
OpWaterSurface->SetFinish(polished);
OpWaterSurface->SetModel(glisur);
G4LogicalBorderSurface* WaterSurface =
new G4LogicalBorderSurface("WaterSurface",
waterTank_phys,expHall_phys,OpWaterSurface);
if(WaterSurface->GetVolume1() == waterTank_phys) G4cout << "Equal" <<
G4endl;
if(WaterSurface->GetVolume2() == expHall_phys ) G4cout << "Equal" <<
G4endl;
//
// Generate & Add Material Properties Table attached to the optical
surfaces
//
const G4int num = 2;
G4double Ephoton[num] = {2.038*eV, 4.144*eV};
//OpticalWaterSurface
G4double RefractiveIndex[num] = {1.35, 1.40};
G4double SpecularLobe[num] = {0.3, 0.3};
G4double SpecularSpike[num] = {0.2, 0.2};
G4double Backscatter[num] = {0.2, 0.2};
G4MaterialPropertiesTable* myST1 = new G4MaterialPropertiesTable();
myST1->AddProperty("RINDEX", Ephoton, RefractiveIndex, num);
```

```
myST1->AddProperty("SPECULARLOBECONSTANT", Ephoton, SpecularLobe, num);
myST1->AddProperty("SPECULARSPIKECONSTANT", Ephoton, SpecularSpike, num);
myST1->AddProperty("BACKSCATTERCONSTANT", Ephoton, Backscatter, num);
OpWaterSurface->SetMaterialPropertiesTable(myST1);
//always return the physical World
return expHall_phys;
}
```

WTEventAction.h

```
#ifndef WTEventAction_h
#define WTEventAction_h 1
#include "G4UserEventAction.hh"
class G4Event;
class CWTEventAction : public G4UserEventAction
{
public:
CWTEventAction();
~CWTEventAction();
public:
void BeginOfEventAction(const G4Event*);
void EndOfEventAction(const G4Event*);
};
#endif
```

WTEventAction.cpp

```
#include "CWTEventAction.hh"
#include "G4Event.hh"
#include "G4EventManager.hh"
#include "G4TrajectoryContainer.hh"
#include "G4Trajectory.hh"
#include "G4VVisManager.hh"
#include "G4ios.hh"
CWTEventAction::CWTEventAction()
{}
CWTEventAction::~CWTEventAction()
{}
void CWTEventAction::BeginOfEventAction(const G4Event*)
{}
void CWTEventAction::EndOfEventAction(const G4Event* evt)
{
// get number of stored trajectories
//
G4TrajectoryContainer* trajectoryContainer = evt-
>GetTrajectoryContainer();
G4int n_trajectories = 0;
if (trajectoryContainer) n_trajectories = trajectoryContainer->entries();
// extract the trajectories and draw them
//
if (G4VVisManager::GetConcreteInstance())
{
for (G4int i=0; i<n_trajectories; i++)
{ G4Trajectory* trj = (G4Trajectory*)
((*(evt->GetTrajectoryContainer()))[i]);
trj->DrawTrajectory(50);
}
}
}
```

WTPhysicsList.h

```
#ifndef WTPhysicsList_h
#define WTPhysicsList_h 1
#include "globals.hh"
#include "G4VUserPhysicsList.hh"
class G4Cerenkov;
class G4Scintillation;
class G4OpAbsorption;
class G4OpRayleigh;
class G4OpBoundaryProcess;
class CWTPhysicsListMessenger;
class CWTPhysicsList : public G4VUserPhysicsList
{
public:
CWTPhysicsList();
~CWTPhysicsList();
public:
void ConstructParticle();
void ConstructProcess();
void SetCuts();
//these methods Construct particles
void ConstructBosons();
void ConstructLeptons();
void ConstructMesons();
void ConstructBaryons();
//these methods Construct physics processes and register them
void ConstructGeneral();
void ConstructEM();
void ConstructOp();
//for the Messenger
void SetVerbose(G4int);
void SetNbOfPhotonsCerenkov(G4int);
private:
G4Cerenkov* theCerenkovProcess;
G4Scintillation* theScintillationProcess;
G4OpAbsorption* theAbsorptionProcess;
G4OpRayleigh* theRayleighScatteringProcess;
G4OpBoundaryProcess* theBoundaryProcess;
CWTPhysicsListMessenger* pMessenger;
};
#endif /* WTPhysicsList_h */
```

WTPhysicsList.cpp

```
#include "G4ios.hh"
#include <iomanip>
#include "globals.hh"
#include "CWTPhysicsList.hh"
#include "CWTPhysicsListMessenger.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4ParticleTable.hh"
#include "G4Material.hh"
#include "G4MaterialTable.hh"
#include "G4ProcessManager.hh"
#include "G4ProcessVector.hh"
#include "G4Cerenkov.hh"
```

```
#include "G4Scintillation.hh"
#include "G4OpAbsorption.hh"
#include "G4OpRayleigh.hh"
#include "G4OpBoundaryProcess.hh"
extern G4VProcess* g_pCerenkov;
extern G4VProcess* g_pScintillation;
CWTPhysicsList::CWTPhysicsList() : G4VUserPhysicsList()
{
theCerenkovProcess = 0;
theScintillationProcess = 0;
theAbsorptionProcess = 0;
theRayleighScatteringProcess = 0;
theBoundaryProcess = 0;
pMessenger = new CWTPhysicsListMessenger(this);
//defaultCutValue = 1.0*mm;
SetVerboseLevel(0);
}
CWTPhysicsList::~CWTPhysicsList() { delete pMessenger;}
void CWTPhysicsList::ConstructParticle()
{
// In this method, static member functions should be called
// for all particles which you want to use.
// This ensures that objects of these particle types will be
// created in the program.
ConstructBosons();
ConstructLeptons();
ConstructMesons();
ConstructBaryons();
}
void CWTPhysicsList::ConstructBosons()
{
// pseudo-particles
G4Geantino::GeantinoDefinition();
G4ChargedGeantino::ChargedGeantinoDefinition();
// gamma
G4Gamma::GammaDefinition();
// optical photon
G4OpticalPhoton::OpticalPhotonDefinition();
}
void CWTPhysicsList::ConstructLeptons()
{
// leptons
G4Electron::ElectronDefinition();
G4Positron::PositronDefinition();
G4NeutrinoE::NeutrinoEDefinition();
G4AntiNeutrinoE::AntiNeutrinoEDefinition();
G4MuonPlus::MuonPlusDefinition();
G4MuonMinus::MuonMinusDefinition();
G4NeutrinoMu::NeutrinoMuDefinition();
G4AntiNeutrinoMu::AntiNeutrinoMuDefinition();
}
void CWTPhysicsList::ConstructMesons()
{
// mesons
G4PionPlus::PionPlusDefinition();
G4PionMinus::PionMinusDefinition();
G4PionZero::PionZeroDefinition();
}
void CWTPhysicsList::ConstructBaryons()
```

```
{
// barions
G4Proton::ProtonDefinition();
G4AntiProton::AntiProtonDefinition();
G4Neutron::NeutronDefinition();
G4AntiNeutron::AntiNeutronDefinition();
}
void CWTPhysicsList::ConstructProcess()
{
AddTransportation();
ConstructGeneral();
ConstructEM();
ConstructOp();
}
#include "G4Decay.hh"
void CWTPhysicsList::ConstructGeneral()
{
G4Decay* theDecayProcess = new G4Decay();
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* particle = theParticleIterator->value();
G4ProcessManager* pmanager = particle->GetProcessManager();
if (theDecayProcess->IsApplicable(*particle)) {
pmanager->AddDiscreteProcess(theDecayProcess);
}
}
}
#include "G4ComptonScattering.hh"
#include "G4GammaConversion.hh"
#include "G4PhotoElectricEffect.hh"
#include "G4MultipleScattering.hh"
#include "G4eIonisation.hh"
#include "G4eBremsstrahlung.hh"
#include "G4eplusAnnihilation.hh"
#include "G4MuIonisation.hh"
#include "G4MuBremsstrahlung.hh"
#include "G4MuPairProduction.hh"
#include "G4hIonisation.hh"
void CWTPhysicsList::ConstructEM()
{
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* particle = theParticleIterator->value();
G4ProcessManager* pmanager = particle->GetProcessManager();
G4String particleName = particle->GetParticleName();
if (particleName == "gamma") {
// gamma
// Construct processes for gamma
// pmanager->AddDiscreteProcess(new G4GammaConversion());
// pmanager->AddDiscreteProcess(new G4ComptonScattering());
// pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());
} else if (particleName == "e-") {
//electron

// Construct processes for electron
pmanager->AddProcess(new G4MultipleScattering(),-1, 1, 1);
pmanager->AddProcess(new G4eIonisation(), -1, 2, 2);
pmanager->AddProcess(new G4eBremsstrahlung(), -1, 3, 3);
} else if (particleName == "e+") {
```

```
//positron
// Construct processes for positron
pmanager->AddProcess(new G4MultipleScattering(),-1, 1, 1);
pmanager->AddProcess(new G4eIonisation(), -1, 2, 2);
pmanager->AddProcess(new G4eBremsstrahlung(), -1, 3, 3);
pmanager->AddProcess(new G4eplusAnnihilation(), 0,-1, 4);
} else if( particleName == "mu+" ||
particleName == "mu-" ) {
//muon
// Construct processes for muon
pmanager->AddProcess(new G4MultipleScattering(),-1, 1, 1);
pmanager->AddProcess(new G4MuIonisation(), -1, 2, 2);
pmanager->AddProcess(new G4MuBremsstrahlung(), -1, 3, 3);
pmanager->AddProcess(new G4MuPairProduction(), -1, 4, 4);
} else {
if ((particle->GetPDGCharge() != 0.0) &&
(particle->GetParticleName() != "chargedgeantino")) {
// all others charged particles except geantino
pmanager->AddProcess(new G4MultipleScattering(),-1,1,1);
pmanager->AddProcess(new G4hIonisation(), -1,2,2);
}
}
}
}
void CWTPhysicsList::ConstructOp()
{
theCerenkovProcess = new G4Cerenkov("Cerenkov");
theScintillationProcess = new G4Scintillation("Scintillation");
theAbsorptionProcess = new G4OpAbsorption();
theRayleighScatteringProcess = new G4OpRayleigh();
theBoundaryProcess = new G4OpBoundaryProcess();
g_pCerenkov=theCerenkovProcess;
g_pScintillation=theScintillationProcess;
SetVerbose(1);
theCerenkovProcess->SetMaxNumPhotonsPerStep(300);
theCerenkovProcess->SetTrackSecondariesFirst(true);
theScintillationProcess->SetScintillationYieldFactor(1.);
theScintillationProcess->SetTrackSecondariesFirst(true);
G4OpticalSurfaceModel themodel = unified;
theBoundaryProcess->SetModel(themodel);
theParticleIterator->reset();
while( (*theParticleIterator)() ){
G4ParticleDefinition* particle = theParticleIterator->value();
G4ProcessManager* pmanager = particle->GetProcessManager();
G4String particleName = particle->GetParticleName();
if (theCerenkovProcess->IsApplicable(*particle)) {
pmanager->AddContinuousProcess(theCerenkovProcess);
}
if (theScintillationProcess->IsApplicable(*particle)) {
pmanager->AddProcess(theScintillationProcess);
pmanager->SetProcessOrderingToLast(theScintillationProcess, idxAtRest);
pmanager->SetProcessOrderingToLast(theScintillationProcess, idxPostStep);
}
if (particleName == "opticalphoton") {
G4cout << " AddDiscreteProcess to OpticalPhoton " << G4endl;
pmanager->AddDiscreteProcess(theAbsorptionProcess);
pmanager->AddDiscreteProcess(theRayleighScatteringProcess);
pmanager->AddDiscreteProcess(theBoundaryProcess);
}
```

```
}
}

void CWTPhysicsList::SetVerbose(G4int verbose)
{
theCerenkovProcess->SetVerboseLevel(verbose);
theScintillationProcess->SetVerboseLevel(verbose);
theAbsorptionProcess->SetVerboseLevel(verbose);
theRayleighScatteringProcess->SetVerboseLevel(verbose);
theBoundaryProcess->SetVerboseLevel(verbose);
}
void CWTPhysicsList::SetNbOfPhotonsCerenkov(G4int MaxNumber)
{
theCerenkovProcess->SetMaxNumPhotonsPerStep(MaxNumber);
}
void CWTPhysicsList::SetCuts()
{
// " G4VUserPhysicsList::SetCutsWithDefault" method sets
// the default cut value for all particle types
//
SetCutsWithDefault();
//SetCutValue(defaultCutValue, "gamma");
SetCutValue(0.00001*mm, "e-");
if (verboseLevel>0) DumpCutValuesTable();
}
```

WTPhysicsListMessenger.h

```
#ifndef WTPhysicsListMessenger_h
#define WTPhysicsListMessenger_h 1
#include "globals.hh"
#include "G4UImessenger.hh"
class WTPhysicsList;
class G4UIdirectory;
class G4UIcmdWithAnInteger;
class CWTPhysicsListMessenger: public G4UImessenger
{
public:
CWTPhysicsListMessenger(CWTPhysicsList* );
~CWTPhysicsListMessenger();
void SetNewValue(G4UIcommand*, G4String);
private:
CWTPhysicsList* pPhysicsList;
G4UIdirectory* N06Dir;
G4UIdirectory* physDir;
G4UIcmdWithAnInteger* verboseCmd;
G4UIcmdWithAnInteger* cerenkovCmd;
};
#endif
```

WTPhysicsListMessenger.cpp

```
#include "CWTPhysicsListMessenger.hh"
#include "CWTPhysicsList.hh"
#include "G4UIdirectory.hh"
#include "G4UIcmdWithAnInteger.hh"
CWTPhysicsListMessenger::CWTPhysicsListMessenger(CWTPhysicsList* pPhys)
:pPhysicsList(pPhys)
{
N06Dir = new G4UIdirectory("/N06/");
```

```
N06Dir->SetGuidance("UI commands of this example");
physDir = new G4UIdirectory("/N06/phys/");
physDir->SetGuidance("PhysicsList control");
verboseCmd = new G4UIcmdWithAnInteger("/N06/phys/verbose",this);
verboseCmd->SetGuidance("set verbose for physics processes");
verboseCmd->SetParameterName("verbose",true);
verboseCmd->SetDefaultValue(1);
verboseCmd->SetRange("verbose>=0");
verboseCmd->AvailableForStates(G4State_PreInit,G4State_Idle);
cerenkovCmd = new
G4UIcmdWithAnInteger("/N06/phys/cerenkovMaxPhotons",this);
cerenkovCmd->SetGuidance("set max nb of photons per step");
cerenkovCmd->SetParameterName("MaxNumber",false);
cerenkovCmd->SetRange("MaxNumber>=0");
cerenkovCmd->AvailableForStates(G4State_PreInit,G4State_Idle);
}
CWTPhysicsListMessenger::~CWTPhysicsListMessenger()
{
delete verboseCmd;
delete cerenkovCmd;
delete physDir;
delete N06Dir;
}
void CWTPhysicsListMessenger::SetNewValue(G4UIcommand* command,
G4String newValue)
{
if( command == verboseCmd )
{ pPhysicsList->SetVerbose(verboseCmd->GetNewIntValue(newValue));}
if( command == cerenkovCmd )
{pPhysicsList->SetNbOfPhotonsCerenkov(cerenkovCmd-
>GetNewIntValue(newValue));}
}
```

WTPrimaryGeneratorAction.h

```
#ifndef WTPrimaryGeneratorAction_h
#define WTPrimaryGeneratorAction_h 1
#include "G4VUserPrimaryGeneratorAction.hh"
#include "globals.hh"
class G4ParticleGun;
class G4Event;
class CWTPrimaryGeneratorMessenger;
class CWTPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
public:
CWTPrimaryGeneratorAction();
~CWTPrimaryGeneratorAction();
public:
void GeneratePrimaries(G4Event*);
void SetOptPhotonPolar(G4double);
private:
G4ParticleGun* particleGun;
CWTPrimaryGeneratorMessenger* gunMessenger;
};
#endif /*WTPrimaryGeneratorAction_h*/
```

WTPrimaryGeneratorAction.cpp

```
#include "CWTPrimaryGeneratorAction.hh"
#include "CWTPrimaryGeneratorMessenger.hh"
#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
CWTPrimaryGeneratorAction::CWTPrimaryGeneratorAction()
{
G4int n_particle = 1;
particleGun = new G4ParticleGun(n_particle);
//create a messenger for this class
gunMessenger = new CWTPrimaryGeneratorMessenger(this);
//default kinematic
//
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* particle = particleTable->FindParticle("e-");
particleGun->SetParticleDefinition(particle);
particleGun->SetParticleTime(0.0*ns);
particleGun->SetParticlePosition(G4ThreeVector(-5.0*m,0.0*cm,0.0*cm));
particleGun->SetParticleMomentumDirection(G4ThreeVector(1.,0.,0.));
particleGun->SetParticleEnergy(500.0*keV);
}
CWTPrimaryGeneratorAction::~CWTPrimaryGeneratorAction()
{
delete particleGun;
delete gunMessenger;
}
void CWTPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
particleGun->GeneratePrimaryVertex(anEvent);
}
void CWTPrimaryGeneratorAction::SetOptPhotonPolar(G4double angle)
{
if (particleGun->GetParticleDefinition()->GetParticleName() !=
"opticalphoton")
{
G4cout << "--> warning from PrimaryGeneratorAction::SetOptPhotonPolar() :"
"the particleGun is not an opticalphoton" << G4endl;
return;
}
G4ThreeVector normal (1., 0., 0.);
G4ThreeVector kphoton = particleGun->GetParticleMomentumDirection();
G4ThreeVector product = normal.cross(kphoton);
G4double modul2 = product*product;
G4ThreeVector e_perpend (0., 0., 1.);
if (modul2 > 0.) e_perpend = (1./sqrt(modul2))*product;
G4ThreeVector e_paralle = e_perpend.cross(kphoton);
G4ThreeVector polar = cos(angle)*e_paralle + sin(angle)*e_perpend;
particleGun->SetParticlePolarization(polar);
}
```

WTPrimaryGeneratorMessenger.h

```
#ifndef WTPrimaryGeneratorMessenger_h
#define WTPrimaryGeneratorMessenger_h 1
#include "G4UImessenger.hh"
#include "globals.hh"
```

```
class CWTPrimaryGeneratorAction;
class G4UIdirectory;
class G4UIcmdWithADoubleAndUnit;
class CWTPrimaryGeneratorMessenger: public G4UImessenger
{
public:
CWTPrimaryGeneratorMessenger(CWTPrimaryGeneratorAction*);
~CWTPrimaryGeneratorMessenger();
void SetNewValue(G4UIcommand*, G4String);
private:
CWTPrimaryGeneratorAction* WTAction;
G4UIdirectory* gunDir;
G4UIcmdWithADoubleAndUnit* polarCmd;
};
#endif
WTPrimaryGeneratorMessenger.cpp 1
#include "CWTPrimaryGeneratorMessenger.hh"
#include "CWTPrimaryGeneratorAction.hh"
#include "G4UIdirectory.hh"
#include "G4UIcmdWithADoubleAndUnit.hh"
CWTPrimaryGeneratorMessenger::CWTPrimaryGeneratorMessenger(
CWTPrimaryGeneratorAction* CWTGun)
:CWTAction(CWTGun)
{
gunDir = new G4UIdirectory("/N06/gun/");
gunDir->SetGuidance("PrimaryGenerator control");
polarCmd = new G4UIcmdWithADoubleAndUnit("/N06/gun/optPhotonPolar",this);
polarCmd->SetGuidance("Set linear polarization");
polarCmd->SetGuidance(" angle w.r.t. (k,n) plane");
polarCmd->SetParameterName("angle",true);
polarCmd->SetUnitCategory("Angle");
polarCmd->SetDefaultValue(0.);
polarCmd->AvailableForStates(G4State_Idle);
}
CWTPrimaryGeneratorMessenger::~CWTPrimaryGeneratorMessenger()
{
delete polarCmd;
delete gunDir;
}
void CWTPrimaryGeneratorMessenger::SetNewValue(
G4UIcommand* command, G4String newValue)
{
if( command == polarCmd )
{ CWTAction->SetOptPhotonPolar(polarCmd->GetNewDoubleValue(newValue));}
}
```

WTRunAction.h

```
#ifndef WTRunAction_h
#define WTRunAction_h 1
#include "globals.hh"
#include "G4UserRunAction.hh"
class G4Timer;
class G4Run;
class CWTRunAction : public G4UserRunAction
{
public:
CWTRunAction();
~CWTRunAction();
public:
```

```
void BeginOfRunAction(const G4Run* aRun);
void EndOfRunAction(const G4Run* aRun);
private:
G4Timer* timer;
};
#endif /*WTRunAction_h*/
```

WTRunAction.cpp

```
#include "G4Timer.hh"
#include "CWTRunAction.hh"
#include "G4ios.hh"
#include "G4Run.hh"
#include "G4UImanager.hh"
#include "G4VVisManager.hh"
CWTRunAction::CWTRunAction()
{
timer = new G4Timer;
}
CWTRunAction::~CWTRunAction()
{
delete timer;
}
void CWTRunAction::BeginOfRunAction(const G4Run* aRun)
{
G4cout << "### Run " << aRun->GetRunID() << " start." << G4endl;
timer->Start();
if (G4VVisManager::GetConcreteInstance())
{
G4UImanager::GetUIpointer()->ApplyCommand("/vis/scene/notifyHandlers");
}
}
void CWTRunAction::EndOfRunAction(const G4Run* aRun)
{
if (G4VVisManager::GetConcreteInstance())
{
G4UImanager::GetUIpointer()->ApplyCommand("/vis/viewer/update");
}
timer->Stop();
G4cout << "number of event = " << aRun->GetNumberOfEvent()
<< " " << *timer << G4endl;
}
```

WTStackingAction.h

```
#ifndef WTStackingAction_H
#define WTStackingAction_H 1
#include "globals.hh"
#include "G4UserStackingAction.hh"
class CWTStackingAction : public G4UserStackingAction
{
public:
CWTStackingAction();
~CWTStackingAction();
public:
G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack);
void NewStage();
void PrepareNewEvent();
```

```
private:
G4int gammaCounter;
FILE* m_pFileData;
FILE* m_pFileDataCer;
FILE* m_pFileDataSci;
FILE* m_pFileProcStat;
G4int m_nCerCnt,m_nSciCnt;
};
#endif
```

WTStackingAction.cpp

```
#include "CWTStackingAction.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4Track.hh"
#include "G4ios.hh"
extern G4VProcess* g_pCerenkov;
extern G4VProcess* g_pScintillation;
extern G4String g_strDataFile;
extern G4String g_strDataFileCer;
extern G4String g_strDataFileSci;
extern G4String g_strProcStatFile;
CWTStackingAction::CWTStackingAction()
: gammaCounter(0)
{
m_nCerCnt=m_nSciCnt=0;
m_pFileData=fopen(g_strDataFile,"w");
m_pFileDataCer=fopen(g_strDataFileCer,"w");
m_pFileDataSci=fopen(g_strDataFileSci,"w");
m_pFileProcStat=fopen(g_strProcStatFile,"w");
}
CWTStackingAction::~CWTStackingAction()
{
if(m_pFileData){
fflush(m_pFileData);
fclose(m_pFileData);
}
if(m_pFileDataCer){
fflush(m_pFileDataCer);
fclose(m_pFileDataCer);
}
if(m_pFileDataSci){
fflush(m_pFileDataSci);
fclose(m_pFileDataSci);
}
if(m_pFileProcStat){
fflush(m_pFileProcStat);
fclose(m_pFileProcStat);
}
}
G4ClassificationOfNewTrack
CWTStackingAction::ClassifyNewTrack(const G4Track * aTrack)
{
if(aTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition())
{ // particle is optical photon
if(aTrack->GetParentID()>0)
{ // particle is secondary
gammaCounter++;
//--------------------------------------------------------------------------
```

```
G4double fEnergy;
G4ThreeVector vecMomentum, vecPolarization;
G4ParticleDefinition* pParDef=aTrack->GetDefinition();
const G4VProcess* pProcess=aTrack->GetCreatorProcess();
if(pParDef->GetParticleName()=="opticalphoton"){
fEnergy=aTrack->GetKineticEnergy()*1000000;
vecMomentum=aTrack->GetMomentumDirection();
vecPolarization=aTrack->GetPolarization();
if(pProcess==g_pCerenkov){
m_nCerCnt++;
if(m_pFileDataCer){
fprintf(m_pFileDataCer,"%f:%f:%f:%f:%f:%f:%f\n",fEnergy,
vecMomentum.x(),vecMomentum.y(),vecMomentum.z(),
vecPolarization.x(),vecPolarization.y(),vecPolarization.z());
}
}
else if(pProcess==g_pScintillation){
m_nSciCnt++;
if(m_pFileDataSci){
fprintf(m_pFileDataSci,"%f:%f:%f:%f:%f:%f:%f\n",fEnergy,
vecMomentum.x(),vecMomentum.y(),vecMomentum.z(),
vecPolarization.x(),vecPolarization.y(),vecPolarization.z());
}
}
if(m_pFileData){
fprintf(m_pFileData,"%f:%f:%f:%f:%f:%f:%f\n",fEnergy,
vecMomentum.x(),vecMomentum.y(),vecMomentum.z(),
vecPolarization.x(),vecPolarization.y(),vecPolarization.z());
}
}
//--------------------------------------------------------------------------
}
}
return fUrgent;
}
void CWTStackingAction::NewStage()
{
G4cout << "Number of optical photons produces in this event : "
<< gammaCounter <<"( C="<<m_nCerCnt<<", S="<<m_nSciCnt<<" )"<<G4endl;
if(m_pFileProcStat)
fprintf(m_pFileProcStat,"%i:%i\n",m_nCerCnt,m_nSciCnt);
}
void CWTStackingAction::PrepareNewEvent()
{ gammaCounter = 0;
m_nCerCnt=m_nSciCnt=0;
}
```

WTSteppingVerbose.h

```
class CWTSteppingVerbose;
#ifndef WTSteppingVerbose_h
#define WTSteppingVerbose_h 1
#include "G4SteppingVerbose.hh"
class CWTSteppingVerbose : public G4SteppingVerbose
{
public:
CWTSteppingVerbose();
~CWTSteppingVerbose();
void StepInfo();
void TrackingStarted();
```

```
private:
FILE* m_pFile;
};
#endif
```

WTSteppingVerbose.cpp

```
#include "stdio.h"
#include "CWTSteppingVerbose.hh"
#include "G4SteppingManager.hh"
#include "G4UnitsTable.hh"
CWTSteppingVerbose::CWTSteppingVerbose()
{
m_pFile=fopen("/home/libor/documents/temp/ex06output.txt","w");
}
CWTSteppingVerbose::~CWTSteppingVerbose()
{
if(m_pFile){
fflush(m_pFile);
fclose(m_pFile);
}
}
void CWTSteppingVerbose::StepInfo()
{
CopyState();
G4int prec = G4cout.precision(3);
if( verboseLevel >= 1 ){
if( verboseLevel >= 4 ) VerboseTrack();
if( verboseLevel >= 3 ){
G4cout << G4endl;
G4cout << std::setw( 5) << "#Step#" << " "
<< std::setw( 6) << "X" << " "
<< std::setw( 6) << "Y" << " "
<< std::setw( 6) << "Z" << " "
<< std::setw( 9) << "KineE" << " "
<< std::setw( 9) << "dEStep" << " "
<< std::setw(10) << "StepLeng"
<< std::setw(10) << "TrakLeng"
<< std::setw(10) << "Volume" << " "
<< std::setw(10) << "Process" << G4endl;
}
G4cout << std::setw(5) << fTrack->GetCurrentStepNumber() << " "
<< std::setw(6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
<< std::setw(6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
<< std::setw(6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
<< std::setw(6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
<< std::setw(6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy")
<< std::setw(6) << G4BestUnit(fStep->GetStepLength(),"Length")
<< std::setw(6) << G4BestUnit(fTrack->GetTrackLength(),"Length")
<< " ";
// if( fStepStatus != fWorldBoundary){
if( fTrack->GetNextVolume() != 0 ) {
G4cout << std::setw(10) << fTrack->GetVolume()->GetName();
} else {
G4cout << std::setw(10) << "OutOfWorld";
}
if(fStep->GetPostStepPoint()->GetProcessDefinedStep() != 0){
G4cout << " "
<< std::setw(10)
<< fStep->GetPostStepPoint()->GetProcessDefinedStep()
```

```
->GetProcessName();
} else {
G4cout << " UserLimit";
}
G4cout << G4endl;
if( verboseLevel == 2 ){
G4int tN2ndariesTot = fN2ndariesAtRestDoIt +
fN2ndariesAlongStepDoIt +
fN2ndariesPostStepDoIt;
if(tN2ndariesTot>0){
G4cout << " :----- List of 2ndaries - "
<< "#SpawnInStep=" << std::setw(3) << tN2ndariesTot
<< "(Rest=" << std::setw(2) << fN2ndariesAtRestDoIt
<< ",Along=" << std::setw(2) << fN2ndariesAlongStepDoIt
<< ",Post=" << std::setw(2) << fN2ndariesPostStepDoIt
<< "), "
<< "#SpawnTotal=" << std::setw(3) << (*fSecondary).size()
<< " ---------------"
<< G4endl;
for(size_t lp1=(*fSecondary).size()-tN2ndariesTot;
lp1<(*fSecondary).size(); lp1++){
G4cout << " : "
<< std::setw(6)
<< G4BestUnit((*fSecondary)[lp1]->GetPosition().x(),"Length")
<< std::setw(6)
<< G4BestUnit((*fSecondary)[lp1]->GetPosition().y(),"Length")
<< std::setw(6)
<< G4BestUnit((*fSecondary)[lp1]->GetPosition().z(),"Length")
<< std::setw(6)
<< G4BestUnit((*fSecondary)[lp1]->GetKineticEnergy(),"Energy")
<< std::setw(10)
<< (*fSecondary)[lp1]->GetDefinition()->GetParticleName();
G4cout << G4endl;
}
G4cout << " :-----------------------------"
<< "---------------------------------"
<< "-- EndOf2ndaries Info ---------------"
<< G4endl;
}
}
}
G4cout.precision(prec);
}
void CWTSteppingVerbose::TrackingStarted()
{
CopyState();
G4int prec = G4cout.precision(3);
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
if( verboseLevel > 0 ){
G4cout << std::setw( 5) << "Step#" << " "
<< std::setw( 6) << "X" << " "
<< std::setw( 6) << "Y" << " "
<< std::setw( 6) << "Z" << " "
<< std::setw( 9) << "KineE" << " "
<< std::setw( 9) << "dEStep" << " "
<< std::setw(10) << "StepLeng"
<< std::setw(10) << "TrakLeng"
<< std::setw(10) << "Volume" << " "
```

```
<< std::setw(10) << "Process" << G4endl;
G4cout << std::setw( 5) << fTrack->GetCurrentStepNumber() << " "
<< std::setw( 6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
<< std::setw( 6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
<< std::setw( 6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
<< std::setw( 6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
<< std::setw( 6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy")
<< std::setw( 6) << G4BestUnit(fStep->GetStepLength(),"Length")
<< std::setw( 6) << G4BestUnit(fTrack->GetTrackLength(),"Length")
<< " ";
if(fTrack->GetNextVolume()){
G4cout << std::setw(10) << fTrack->GetVolume()->GetName();
} else {
G4cout << "OutOfWorld";
}
G4cout << " initStep" << G4endl;
}
G4cout.precision(prec);
}
```

WTVisManager.h

```
#ifndef WTVisManager_h
#define WTVisManager_h 1
#ifdef G4VIS_USE
#include "G4VisManager.hh"
class CWTVisManager: public G4VisManager {
public:
CWTVisManager ();
private:
void RegisterGraphicsSystems ();
};
#endif
#endif
```

WTVisManager.cpp

```
#ifdef G4VIS_USE
#include "CWTVisManager.hh"
// Supported drivers...
// Not needing external packages or libraries...
#include "G4ASCIITree.hh"
#include "G4DAWNFILE.hh"
#include "G4GAGTree.hh"
#include "G4HepRepFile.hh"
#include "G4HepRep.hh"
#include "G4RayTracer.hh"
#include "G4VRML1File.hh"
#include "G4VRML2File.hh"
// Needing external packages or libraries...
#ifdef G4VIS_USE_DAWN
#include "G4FukuiRenderer.hh"
#endif
#ifdef G4VIS_USE_OPENGLX
#include "G4OpenGLImmediateX.hh"
#include "G4OpenGLStoredX.hh"
#endif
#ifdef G4VIS_USE_OPENGLWIN32
#include "G4OpenGLImmediateWin32.hh"
#include "G4OpenGLStoredWin32.hh"
```

```
#endif
#ifdef G4VIS_USE_OPENGLXM
#include "G4OpenGLImmediateXm.hh"
#include "G4OpenGLStoredXm.hh"
#endif
#ifdef G4VIS_USE_OIX
#include "G4OpenInventorX.hh"
#endif
#ifdef G4VIS_USE_OIWIN32
#include "G4OpenInventorWin32.hh"
#endif
#ifdef G4VIS_USE_VRML
#include "G4VRML1.hh"
#include "G4VRML2.hh"
#endif
CWTVisManager::CWTVisManager () {}
void CWTVisManager::RegisterGraphicsSystems () {
// Graphics Systems not needing external packages or libraries...
RegisterGraphicsSystem (new G4ASCIITree);
RegisterGraphicsSystem (new G4DAWNFILE);
RegisterGraphicsSystem (new G4GAGTree);
RegisterGraphicsSystem (new G4HepRepFile);
RegisterGraphicsSystem (new G4HepRep);
RegisterGraphicsSystem (new G4RayTracer);
RegisterGraphicsSystem (new G4VRML1File);
RegisterGraphicsSystem (new G4VRML2File);
// Graphics systems needing external packages or libraries...
#ifdef G4VIS_USE_DAWN
RegisterGraphicsSystem (new G4FukuiRenderer);
#endif
#ifdef G4VIS_USE_OPENGLX
RegisterGraphicsSystem (new G4OpenGLImmediateX);
RegisterGraphicsSystem (new G4OpenGLStoredX);
#endif
#ifdef G4VIS_USE_OPENGLWIN32
RegisterGraphicsSystem (new G4OpenGLImmediateWin32);
RegisterGraphicsSystem (new G4OpenGLStoredWin32);
#endif
#ifdef G4VIS_USE_OPENGLXM
RegisterGraphicsSystem (new G4OpenGLImmediateXm);
RegisterGraphicsSystem (new G4OpenGLStoredXm);
#endif
#ifdef G4VIS_USE_OIX
RegisterGraphicsSystem (new G4OpenInventorX);
#endif
#ifdef G4VIS_USE_OIWIN32
RegisterGraphicsSystem (new G4OpenInventorWin32);
#endif
#ifdef G4VIS_USE_VRML
RegisterGraphicsSystem (new G4VRML1);
RegisterGraphicsSystem (new G4VRML2);
#endif
if (fVerbose > 0) {
G4cout <<
"\nYou have successfully chosen to use the following graphics systems."
<< G4endl;
PrintAvailableGraphicsSystems ();
}}
#endif
```

Mgr. Libor Nožka, Ph.D.

Modelování fyzikálních procesů pomocí trasovacího nástroje Geant4

Výkonný redaktor: prof. RNDr. Tomáš Opatrný, Dr.
Odpovědná redaktorka: Vendula Drozdová
Návrh a grafické zpracování obálky: Jiří K. Jurečka

Vydala a vytiskla Univerzita Palackého v Olomouci
Křížkovského 8, 771 47 Olomouc
www.upol.cz/vup

Olomouc 2012

1. vydání

ISBN 978-80-244-3078-2