

입자핵물리학 실험을 전공하는  
대학원생을 위한  
ROOT 와 GEANT4 집중강좌

2009 Nuclear Physics School @ Pohang

Inkyu Park

Dept. of Physics, University of Seoul

# 강의 계획

- 2009 년 7 월 1 일 수요일 (9:30-12:30)
  - ROOT 의 소개 및 설치 (Introduction, Installation)
  - ROOT 의 클래스
  - ROOT 연습 : Histogram & Ntuple
- 2009 년 7 월 2 일 목요일 (9:30-12:30)
  - GEANT4 의 소개 (Introduction)
  - GEANT4 의 설치 (Installation)
  - GEANT4/CLHEP 의 문법
  - GEANT4 Getting started
- 2009 년 7 월 3 일 목요일 (14:00-17:00)
  - Example: Detector construction
  - Example: Physics List, Particles
  - Visualization
  - Histogram

# **Lecture 1:**

# **ROOT**

# Prerequisites

- **ROOT와 GEANT4를 각각 3시간만에 소개하고, 간단한 예제를 실행해 본다**는 것은 →불가능한 이야기 이다
  - 서울시립대 물리학과 3학년 : ROOT 1학기, GEANT4 2학기
- **Prerequisite 1: math + physics**
  - math: histograms, functions, statistics
  - physics: mechanics, modern physics
- **Prerequisite 2: OS (LINUX)**
  - commands: ls, cd, cp, mv, rm, mkdir, more, grep
  - editors: vi, vim, emacs, pico, etc.,...
- **Prerequisite 3: Programming (C++)**
  - compile: gcc, g++, make
  - C++ syntax: types, keywords
  - OO concept: class, inheritance

# What you can do with ROOT?

- Draw text, LaTeX, 2D/3D shapes
- Draw functions, graphs, histograms
- Fit histograms, estimate statistical errors
- Data handling with Ntuple, Tree
- File I/O
- Network programming
- Parallel, GRID programming with Proof
- Geometry, virtual detector construction
- Interfacing to Physics generators

**Framework!**

# ROOT Installation

- Visit the web site <http://root.cern.ch> and get the source
  - get the pro (production) version 5.22/00
  - download `root_v5.22.00.source.tar.gz`
  - `tar xvfz root_v5.22.00.source.tar.gz`
- Setting an environmental variable ROOTSYS
  - `export ROOTSYS=/home/icpark/root`
- Installation
  - `cd $ROOTSYS`
  - `./configure`
  - `make`
  - `make install`
- Setting some more environmental variables
  - `export PATH+=:$ROOTSYS/bin`
  - `export LD_LIBRARY_PATH+=:$ROOTSYS/lib`

# Launching ROOT

- “root” is the command to call ROOT
  - root ↵
- A pop-up and a welcome message will appear

```
Terminal — root.exe — 80x24
On this machine the XAWFLAGS=
On this machine the XAWLIBS=
On this machine the G4LIB_BUILD_SHARED=1
On this machine the G4LIB_BUILD_STATIC=1
On this machine the G4LIB_USE_GRANULAR=1
On this machine the G4UI_USE_TCSH=1
icpark@localhost:~> root
*****
*
*   W E L C O M E  t o  R O O T   *
*
*   Version   5.22/00  17 December 2008 *
*
*   You are welcome to visit our Web site *
*   http://root.cern.ch *
*
*****

ROOT 5.22/00 (trunk@26997, De 18 2008, 10:17:00 on macosx)

CINT/ROOT C/C++ Interpreter version 5.16.29, Jan 08, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```



# ROOT command

- “How to quit?” is a top ranked FAQ
  - `.q`
- ROOT commands start with “.”, otherwise ROOT understands as “C++” statements.
- Some basic ROOT commands
  - `.q` (quit)
  - `.x` (execute a file)
  - `.L` (load a file)
  - `.ls` (ls on current directory)
  - `.pwd` (print working directory)
  - `.! [shell]` (execute a OS command)
  - `.T` (trace mode toggle)

# ROOT as a C++ interpreter

- Thanks to CINT, ROOT is a command line C++ interpreter

- **example 1**

```
- root [0] Int_t a;  
- root [1] a = 3 * 5;  
- root [2] cout << a << endl;  
- 15
```

- **example 2**

```
- root [0] Double_t x=1.3, y;  
- root [1] y = TMath::Erfc(x)  
- (Double_t) 6.59920503287388940e-02  
- root [2] TMath::Sin(TMath::Pi())  
- (Double_t) 1.22464679914735320e-16
```

# Call functions

## • Load functions from file

```
-root [0] .L mymult.C
-root [1] mult(3,4)
-(Double_t)1.200000000e+01
-root [2] mult()
-enter a number:4
-enter a second number:8
-(Double_t)3.200000000e+01
-
```

## • ROOT internal functions

```
-root [0] TMath::Pi()
-3.14159265358979312e+00
-root [1] gRandom->Rndm()
-9.99741748906672001e-01
```

```
> more mymult.C
```

```
double mult(double a,double b){
// multiply a and b
    return a*b;
}

double mult() {
    double_t a, b;
    cout << "enter a number:";
    cin >> a;
    cout << "enter a second number:";
    cin >> b;
    return a*b;
}
```

# **Lecture 2:**

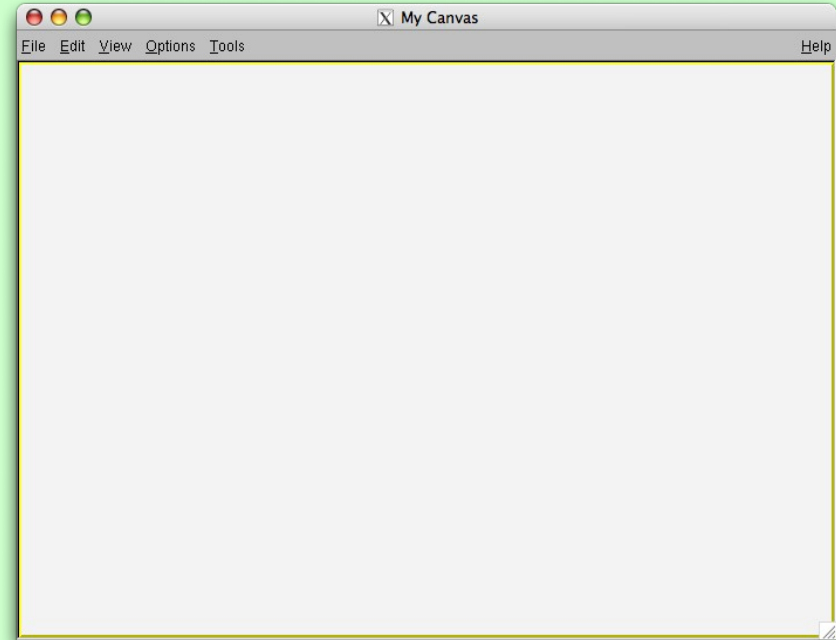
# **ROOT Classes & Practice**

# ROOT classes

- ROOT 클래스는 모두 “T”로 시작한다.
  - TCanvas (Canvas window)
  - TF1 (1D function class)
  - TF2 (2D function class)
  - TH1F (1D float histogram class)
  - TH1D (1D double precision histogram class)
  - TNtuple (Ntuple class)
  - TFile (ROOT File object class)
  - TGeo\* (Geometry classes)

# TCanvas

- ROOT 로 그림을 그리려면 우선 창이 필요하다.
- TCanvas 는 가장 많이 쓰이는 output 창이다.
  - `TCanvas *canvas = new TCanvas ("can", "My Canvas");`
- TCanvas 창의 크기와 위치를 정의 할 수 있다.
  - `TCanvas ("can", "My Canvas");`
    - create default widow
  - `TCanvas ("can", "My Canvas", 300, 400);`
    - window size = 300x400
  - `TCanvas ("can", "My Canvas", 100, 200, 300, 400);`
    - upper left corner starts at (100,200) & size of 300x400



# Functions

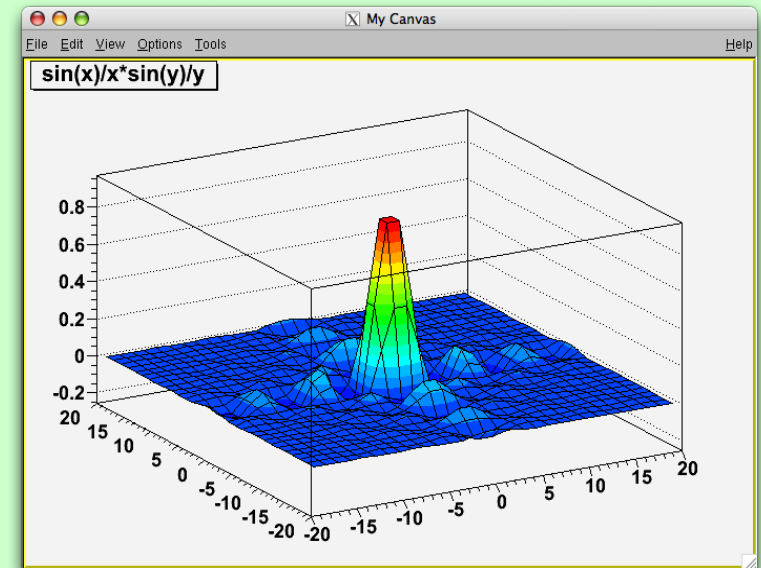
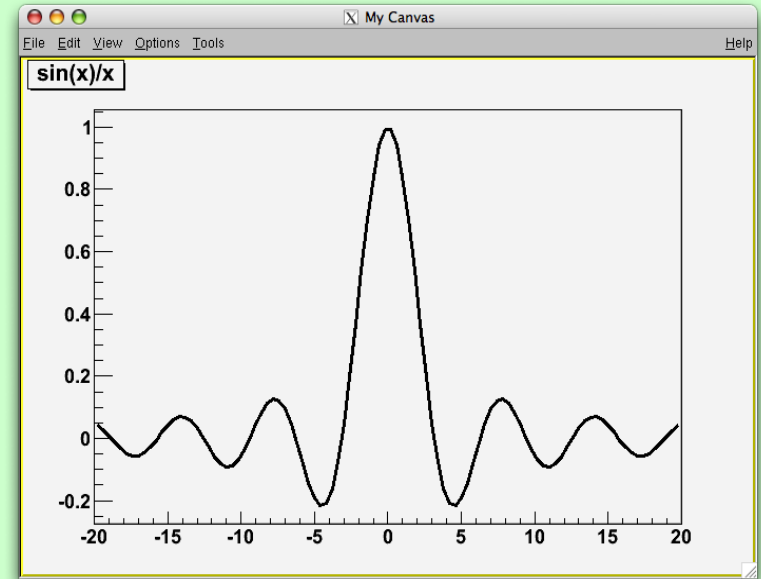
- ROOT 에서 함수를 그리는 것은 매우 쉽다.

- 1D function

- `TF1 *f1 = new TF1("f1", "sin(x)/x", -20,20);`
- `fun1->Draw();`

- 2D function

- `TF2 f2 = new TF2("f2", "sin(x)/x*sin(y)/y", -20,20,-20,20);`
- `f2->Draw("surf1")`



# Histograms

- Histogram 을 정의하고

- `TH1F *h1 = new TH1F("h1", "My hist", 100, -5, 5);`

- Histogram 을 채우고

- `h1->Fill(x)`

- Histogram 을 그린다

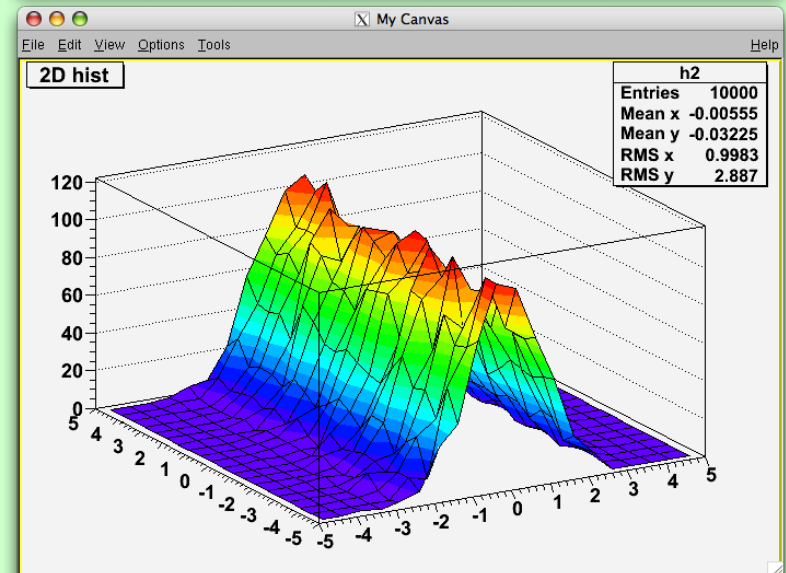
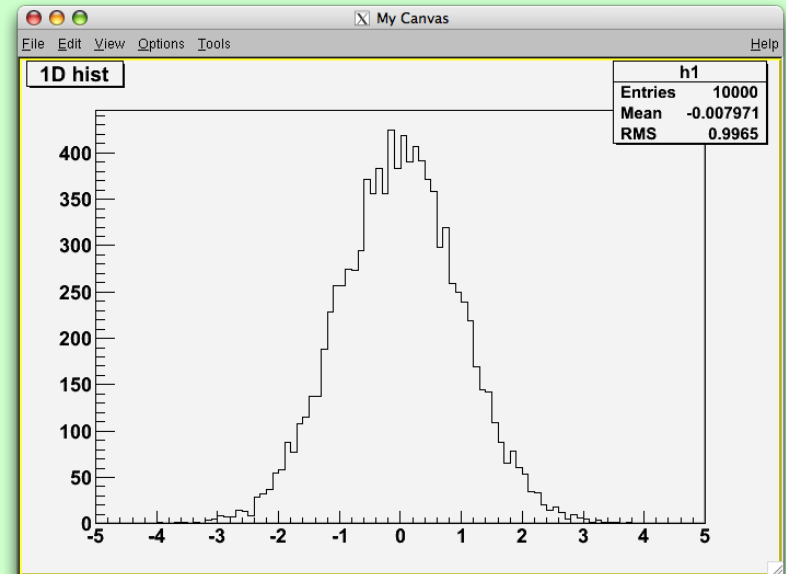
- `h1->Draw();`

- 2D 의 경우

- `TH2F *h2 = new TH2F("h2", "My hist", 100, -5, 5, 100, -5, 5);`

- `h2->Fill(x, y);`

- `h2->Draw("lego");`



# Tree

- Tree 는 많은 양의 다양한 데이터를 저장하기 위해서 만든 개념이다. 트리생성은 매우 쉽다.
  - TTree \*tree=new TTree("tree","My tree");
- 어떤 Object 도 그 주소를 branch(가지)로 등록할 수 있다. 예를 보자.

```
typedef struct {float x,y,z;} POINT; POINT point;
TH1F *h1= new TH1F("h1","h1",100,-4,4);

TTree *tree = new TTree("tree","My tree");
tree->Branch("point",&point,"x:y:z");
tree->Branch("hist1","TH1F",&h1,128000,0);

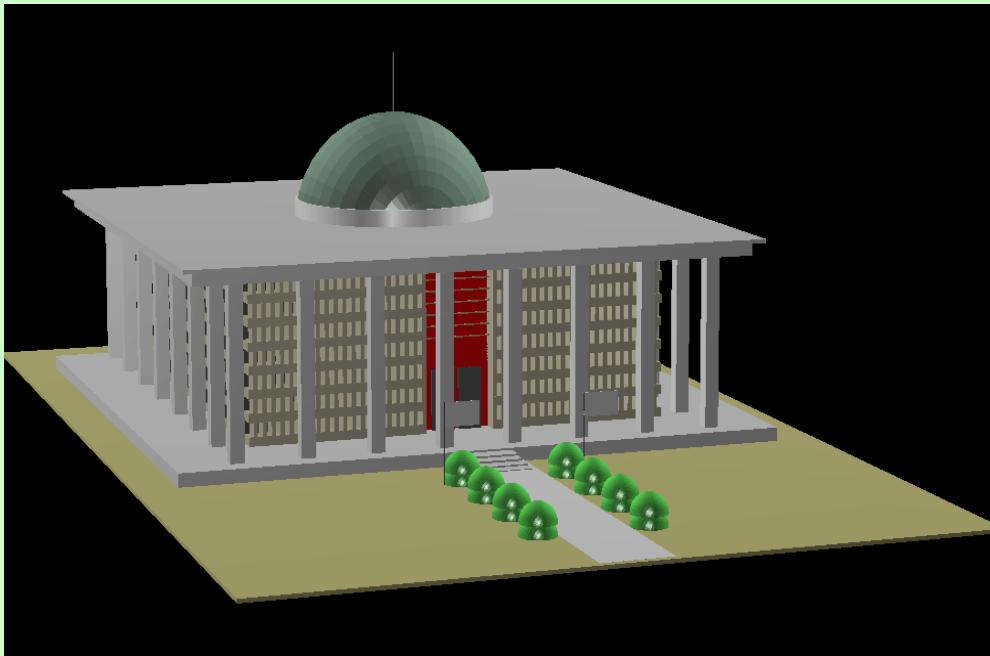
for (int i=0;i<1000;i++)
{ ...
  h1->Fill(v1);
  point.x = hx; point.y = hy; point.z = hz;
  tree->Fill();
}
```

# TFile

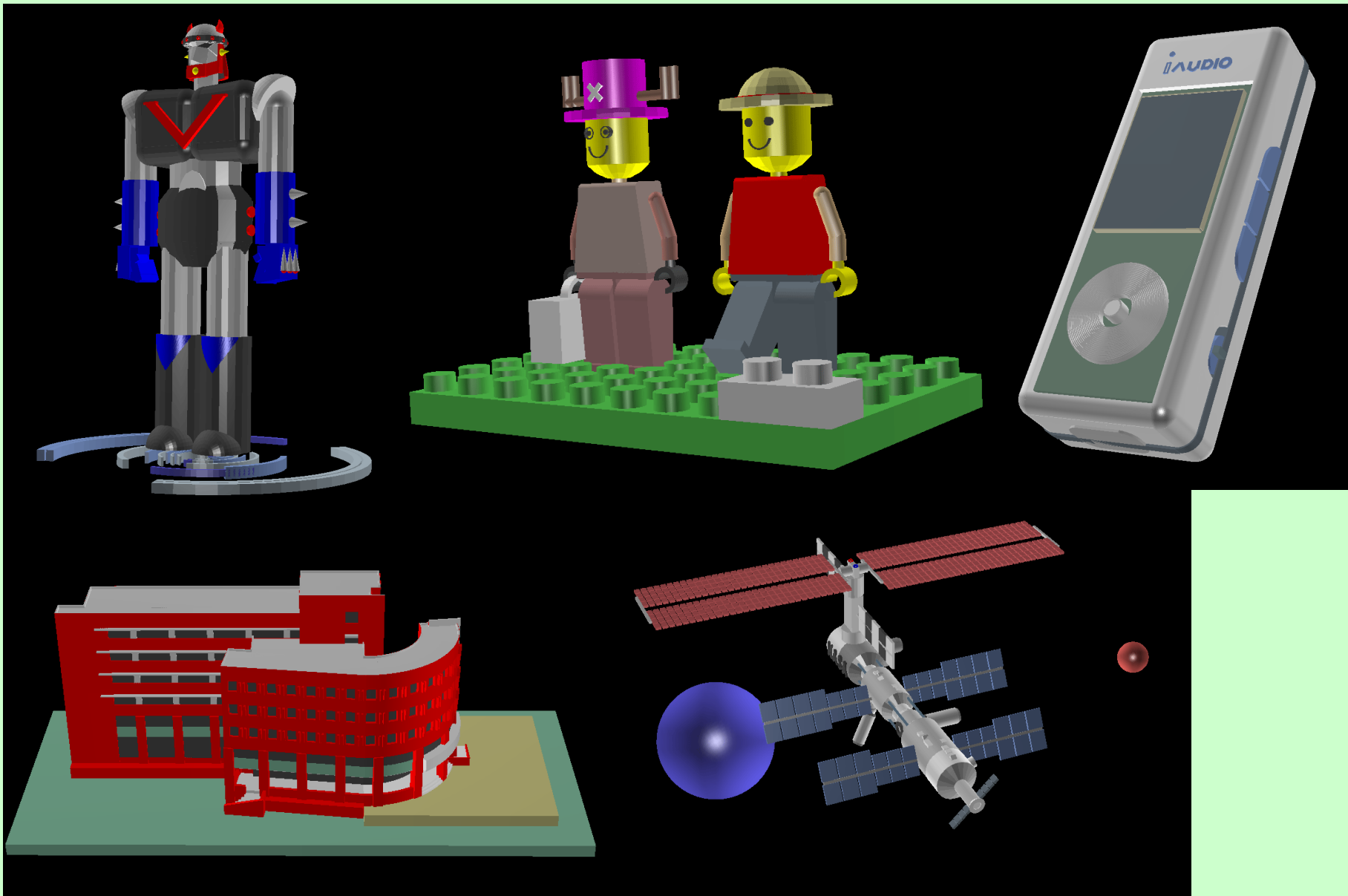
- ROOT 의 다양한 오브젝트들을 파일에 담아보자
  - `TFile *file = new TFile("file.root", "RECREATE");`
- 파일을 연 후 다양한 오브젝트들을 `Write()` 한다.
  - `hist->Write();`
  - `function->Write();`
  - `tree->Write();`
- 파일을 닫으면 끝!
  - `file->Close();`

# Geometry

- 2D Shape 그래픽 오브젝트들
  - TBox, TArrow, TArc
- 3D Shape 그래픽 오브젝트들
  - TBRIK, TTUBE
- TGeo 클래스등으로 각종 그림을 그릴 수 있다.
- 몇가지 그동안 학생들이 했던일을 보자!



# 물리학과 3학년 학생들의 작품



# **Lecture 3:**

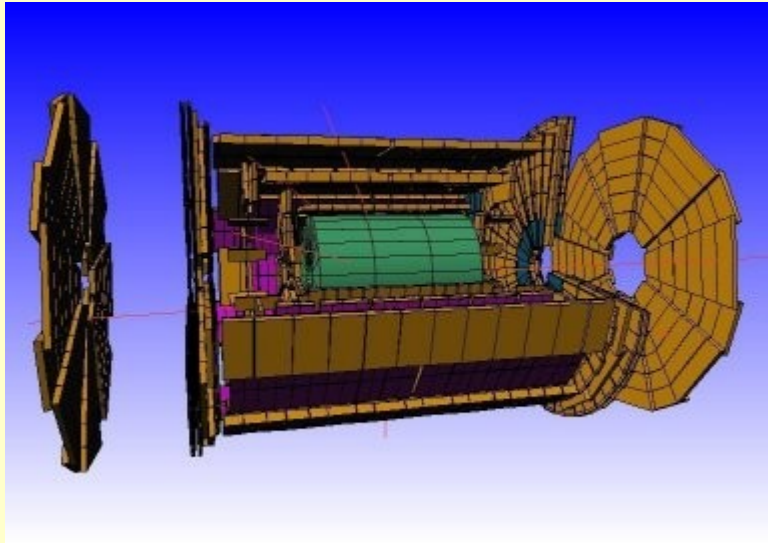
## **GEANT4**

### **Installation**

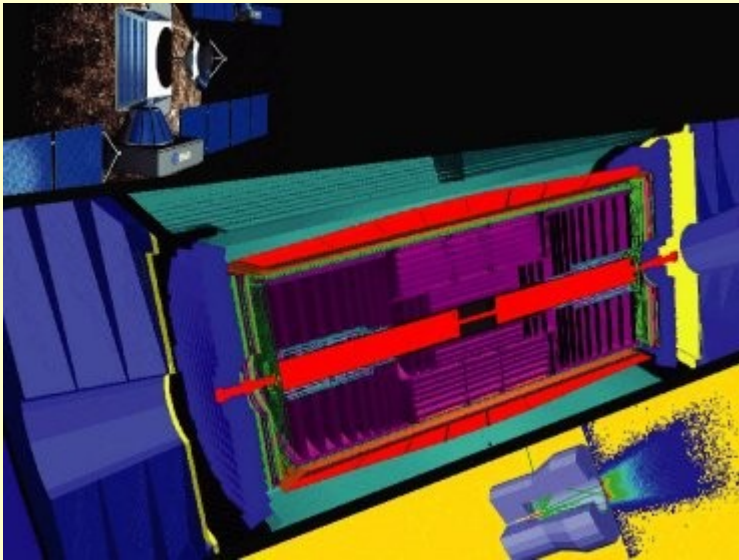
# 검출기 시뮬레이터란 ?

- 전산모사란 실제 물리실험에서 벌어지는 상황을 수치적으로 흉내내기를 하는 일을 말한다.
- 우주선이나 가속기에서 나오는 입자들을 조사하기 위해 만든 검출기를 컴퓨터상에서 전산모사하는 프로그램을 검출기 시뮬레이터라 부른다.
- 실제 검출기를 제작하듯 같은 물질과 같은ジオ메트리를 사용하여 시뮬레이터를 제작한다.
- 검출기 시뮬레이터는 입자들의 생성과정 (kinematics) 과 입자들이 검출기를 지나가는 과정을 모사한다.
- 입자의 종류, 에너지, 전하 등등에 따른 물질과의 상호작용 정도를 계산에 고려해야 한다.
- 전산모사과정에서 발생하는 각종 정보를 출력하는 기능이 필요하고, 눈으로 볼 수 있는 그림출력도 제공하면 좋다.

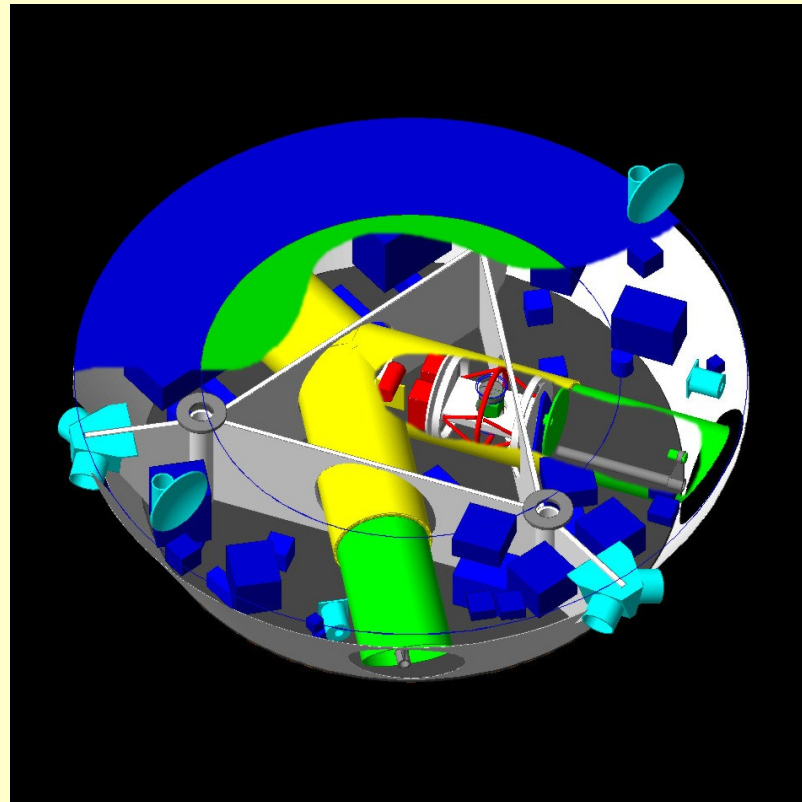
# Geant4 – Picture gallery



ATLAS Muon Chamber



CMS



LISA

# Geant3 vs Geant4

- Geant3 는 80 년대에 FORTRAN 을 바탕으로 LEP 실험 과 함께 지속적으로 발달하였다 .
  - 복잡한 지오메트리를 설치하기 힘들었고 , CAD 와의 연동불가능
  - Electromagnetic, Hadronic physics 프로세스들을 업그레이드하기가 수월하지 않았다 . 100MeV-100GeV 까지만 잘 작동하였다 .
  - 트래킹 코드등 코드의 관리와 업그레이드가 아주 복잡하였다 .
- Geant4 는 LHC 실험을 위해 90 년대 후반부터 준비해 온 OO 기반의 검출기 시뮬레이터이다 .
  - OO 이므로 OO 의 장점을 다 따른다 .
  - 시작부터 CAD 지오메트리 형식을 따라 , 복잡한 지오메트리를 다룰 수 있고 , CAD 와 연동가능
  - Physics 프로세스의 업그레이드가 용이하다 .



# Geant4 documentation

- **GEANT4 의 공식 홈페이지**
  - `http://geant4.cern.ch`
- **GEANT4 를 온라인에서 배울 수 있는 곳**
  - `http://www.wlap.org`
- **GEANT4 를 기술한 논문**
  - Nuclear Instruments and Methods,  
A 506 (2003) 250-303
  - IEEE Transactions on Nuclear Science,  
53 No. 1 (2006) 270-278

# Geant4 의 설치 : 준비운동

- 본 강좌는 Linux 와 g++ 을 바탕으로 한다.
  - 지금 이 강좌는 Mac OSX 와 g++ 에서 하고 있다 .
- Windows 와 cygwin, VC++ 로 설치하는 것은 본 강좌에서는 다루지 않는다.
  -
- 가급적 root 계정으로 설치하자 .
- 물리에 관련된 모든 패키지를 독립된 디스크에 몰아서 설치하자 . ( 후일 다른 머신에 network mount 할 것에 대비하자 .)
- 보통 /usr/local, /opt, /cern, /home/geant4 등을 많이 쓰지만 , 이 강좌에서는 설치할 디렉터리를 \$DSKPHY 라 가정한다 .
- 우선 CLHEP 과 GEANT4 를 위한 방을 만들자 .
  - mkdir \$DSKPHY/CLHEP
  - mkdir \$DSKPHY/GEANT4

# Geant4 의 설치 : 잠깐만 !

- 무작정 다운받고 설치하지 전에... 몇 가지를 확인하자 .
- 반드시 있어야 하는 것들
  - C++ 컴파일러 ( g++ )
  - CLHEP 라이브러리
  - GNU make
  - GEANT4 source
- 본 강좌에서는 아래의 버전을 가지고 진행한다 .
  - gcc 4.0.1

# Geant4 의 설치 : CLHEP 의 설치 (1/2)

- CLHEP 이란 Class Library for High Energy Physics 를 딴 이름으로, 고에너지물리를 위한 c++ 클래스 라이브러리이다.
- 난수발생, 벡터, 행렬, 선형대수, 지오메트리 등 물리에 필요한 많은 유틸리티를 c++ class 로 제작
- 현재 버전 1.8, 1.9, 2.0 등으로 발전 중이나 GEANT4 v8.0 은 2.0 과 잘 어울린다. (Remember, serious consequences happen when you upgrade your wife!)
  - <http://proj-clhep.web.cern.ch/proj-clhep>
  - `/afs/cern.ch/atlas/project/geant4/4.2.0/CLHEP`, or `/afs/cern.ch/sw/lhcxx`
  - `clhep-2.0.2.2.tgz` 을 풀면 `2.0.2.2/CLHEP` 이 생성됨
  - `$DSKPHY/CLHEP/2.0.2.2/CLHEP`

# Geant4 의 설치 : CLHEP 의 설치 (2/2)

- 본 강좌에서는 2.0.2.2 을 사용했다 (clhep-2.0.2.2.tgz)
- `$DSKPHY/2.0.2.2/CLHEP`
  - `$DSKPHY/clhep-2.0.2.2.tgz` (5MB)
  - `tar xvfz clhep-2.0.2.2.tgz`
  - `cd $DSKPHY/2.0.2.2/CLHEP/`
  - `./configure`
  - `make`
  - `make install`
- 기본설정은 `CLHEP_BASE_DIR=/usr/local` 이다.
  - `/usr/local/lib`
  - `/usr/local/include/CLHEP`
- 이것으로 CLHEP 설치는 끝이다.

# Geant4 의 설치 : 다운로드 & 설치

- 본 강좌에서 다루는 버전은 9.2 이고, CLHEP 은 2.0 과 맞는다.
- **download**
  - `http://geant4.web.cern.ch/geant4`
  - `wget`  
`http://geant4.cern.ch/support/source/geant4.9.2.p01.tar.gz (18MB)`
- **installation**
  - `tar xvfz geant4.9.2.p01.tar.gz`
  - `cd geant4.9.2.p01.tar.gz`
  - `export G4INSTALL=`pwd``
  - `./Configure -build`
- 질문에 차근차근 답변을 하는 것이 최상의 길이다.

# Geant4 의 설치 : 컴파일

- 머신에 따라 다르지만 보통 1-2 시간 정도가 흐른다.
- 아래의 메시지가 맨 끝에 뜨면 일단 성공이라 여길 수 있다.

```
#####  
# Your Geant4 installation seems to be successful!  
# To be sure please have a look into the log file:  
# $G4INSTALL/.config/bin/$G4SYSTEM/g4make.log  
#####
```

# Geant4 의 설치 : Physics data 의 설치

- 몇몇 물리프로세스는 특정 크로스섹션 데이터를 필요로 한다.
  - G4NDL.3.8: thermal cross section 을 포함한 중성자데이터
  - G4NDL.0.2: thermal cross section 을 제외한 중성자데이터
  - G4EMLOW.3.0: Low energy EM process
  - PhotonEvaporation.2.0: Photon evaporation
  - RadiativeDecay.3.0: Radioactive decay
  - G4ELASTIC.1.0: High energy elastic scattering
- 이들을 `$G4INSTALL/data` 라는 디렉터리에 넣자
  - `mkdir $G4INSTALL/data`
  - `wget http://.../G4ELASTIC.1.1.tar.gz`
  - `wget http://.../G4NDL.3.8.tar.gz`
  - ...
- `unzip untar` 하여 풀어놓으면 된다.

# Geant4 의 설치 : visualization (1/2)

- GEANT4 는 기본설치로는 어떠한 그래픽도 제공되지 않는다.
- 검출기의 그림과 입자들의 가상궤적등을 보고 싶다면 , **graphic library** 를 설치해야 한다.
  - X
  - OpenGL
  - DAWN
  - Open Inventor
  - Open Scientist
  - WIRED3 HepRep browser
  - WIRED4 JAS Plug-in
- 정말 만족할 만한 **graphic library** 는 아직 없다고 본다...

# Geant4 의 설치 : visualization (2/2)

- 설치시에 visualization 을 설치하지 않았을 경우는 직접 매뉴얼작업을 통해 추가 설치할 수 있다.
  - `cd $G4INSTALL/source/visualization`
- README 파일을 잘 읽어보고 어떤 그래픽 툴이 필요한지를 잘 결정해 환경변수를 셋팅한다.
  - `export G4VIS_BUILD_DAWN_DRIVER=1`
  - `export G4VIS_BUILD_DAWNFILE_DRIVER=1`
  - `export G4VIS_BUILD_VRML_DRIVER =1`
  - `export G4VIS_BUILD_VRMLFILE_DRIVER=1`
  - `make`
- granular 로 라이브러리를 구성할 땐 libname.map 을 재구성
  - `make libmap`

# Geant4의 사용 : 환경변수

- 이제 사용자계정으로 로그인하여 Geant4를 사용하자.
- 몇 가지 환경변수의 셋팅은 아주 중요하다.
  - `export CLHEP_BASE_DIR=/usr/local`
  - `export G4INSTALL=$DSKPHY/GEANT4/geant4.8.0.p01`
  - `export G4SYSTEM=Linux-g++`
  - `export LD_LIBRARY_PATH=$G4INSTALL/lib/$G4SYSTEM:  
$LD_LIBRARY_PATH`
  - `export G4WORKDIR=./`
- 위의 셋팅을 `.bashrc`에 넣어 놓으면 편리하다.
- `source .bashrc` 하면 Geant4의 사용준비가 끝났다.

# Geant4 make 의 이해 (1/2)

- `$G4INSTALL/config` 에 들어있는 아래의 스크립트들이 `make` 과정에 사용된다.
  - `architecture.gmk`
    - `architecture` 에 관련된 셋팅
  - `common.gmk`
    - 라이브러리등의 제작에 필요한 공통적이 셋팅
  - `globlib.gmk`
    - `compound libraries` 제작을 위한 셋팅
  - `binmake.gmk`
    - 실행파일을 만들기 위한 규칙
  - `GNUmakefiles`
    - `make` 파일

# Geant4 make 의 이해 (2/2)

- 사용자가 제작한 검출기 libraries 는 `$G4INSTALL/lib/$G4SYSTEM` 에 생성된다.
- 실행파일은 `$G4WORKDIR/bin/$G4SYSTEM` 에 들어간다.
- 컴파일 과정 중 생긴 잡다한 임시파일들 (object files, dependency files, data products of the compilation process) 은 `$G4WORKDIR/tmp/$G4SYSTEM` 에 생긴다.
- `G4WORKDIR` 을 셋팅하지 않으면 `$G4INSTALL` 을 사용하므로 `$G4INSTALL` 디렉터리가 지저분해진다. 가끔적 자신의 계정을 사용하자.

# 오늘의 숙제

- 각자에게 부여된 계정에 **CLHEP** 을 설치해보아라 .
  - `CLHEP_BASE_DIR=$HOME/lib` 으로 하자
  -
- **Geant4** 를 다운로드 하여 자신의 `$HOME` 에 설치해보자 .
  - 
  - `G4INSTALL=$HOME/GEANT4`
  -
- `$G4INSTALL/examples` 의 `novice` 에 들어 있는 **N01** 부터 **N07** 까지 컴파일을 해보자 .
- **N01** 안에 들어있는 **README** 파일을 읽어보아라 .

# **Lecture 4:**

**CLHEP/GEANT4**

**Syntax**

# G4의 문법을 알아보자.

- 때론 규정이 맘에 안들면 프로그래밍을 하기 싫을 때도 있다. 허나 어찌랴... Geant4를 다시 쓸 수는 없지 않는가?
- 모든 Geant4 소스파일은 .cc의 확장자를 갖는다.
- 모든 Geant4 헤더파일은 .hh의 확장자를 쓴다.
- 모든 Geant4 클래스는 G4라는 두 글자로 시작한다.
  - G4RunManager, G4Step, G4LogicalVolume
- Abstract 클래스는 V라는 한 글자를 더 넣어 표시한다.
  - G4VHit, G4VPhysicalVolume
- 각 단어의 시작을 대문자로 하여 구별한다.
  - G4UserAction, G4VPVParameterisation
- 함수들도 같은 이름규칙을 사용한다.
  - G4RunManager::SetUserAction(),  
G4LogicalVolume::GetName()

# G4 문법 : 쓸데없는 일을 만들어 한다 ?

- CPU의 종류와 컴파일러에 따라 같은 c++의 변수도 다른 정밀도를 갖는다.
- 서로 다른 머신과 컴파일러에서도 같은 결과를 만들기 위해 Geant4는 기본 c/c++ 변수를 다시 정의하여 쓴다.
  - `int` → `G4int`
  - `long` → `G4long`
  - `float` → `G4float`
  - `double` → `G4double`
  - `bool` → `G4bool`
  - `string` → `G4String`
  -
- 이렇게 하면 다른 프로그램에 포팅할때 유리한점도 있다.
- 이들의 정의는 **globals.hh**에 들어있다.

# Geant4 and CLHEP

- GEANT4 는 앞의 경우처럼 여러 CLHEP 클래스를 typedef 해서 사용한다 .
  - **System of units**
  - **Vector classes and matrices**
    - G4ThreeVector (typedef to Hep3Vector)
    - G4RotationMatrix (typedef to HepRotation)
    - G4LorentzVector (typedef to HepLorentzVector)
    - G4LorentzRotation (typedef to HepLorentzRotation)
  - **Geometrical classes**
    - G4Plane3D (typedef to HepPlane3D)
    - G4Transform3D (typedef to HepTransform3D)
    - G4Normal3D (typedef to HepNormal3D)
    - G4Point3D (typedef to HepPoint3D)
    - G4Vector3D (typedef to HepVector3D)

# System of units (1/4)

- Geant4 에서는 여러 단위계를 사용할 수 있다 .
- Geant4 에서 물리량은 숫자 뒤에 단위를 곱해 표현한다 .
  - millimeter (mm)
  - nanosecond (ns)
  - Mega electron Volt (MeV)
  - Positron charge (eplus)
  - Degree Kelvin (kelvin)
  - Amount of substance (mole)
  - Luminosity intensity (candela)
  - Radian (radian)
  - steradian (steradian)

# System of units (2/4)

- 같은 단위도 여러 방식으로 표현 가능하다.  
    `millimeter=mm=1;`  
    `meter = m = 1000*mm;`  
    .....  
    `m3 = m*m*m;`  
    .....
- 자세한 정의를 보려면 아래의 파일을 참조하자
  - `source/global/management/include/SystemOfUnits`
- 장점 : 단위계를 마음대로 바꾸어써도 된다.
  - `30*cm == 0.3*m`

# System of units (3/4)

- 출력을 할 때 단위를 고려해야 한다. 즉 단위로 나누어라!  
예를 들면
  - `cout << KineticEnergy/KeV << " KeV " << endl;`
- **G4BestUnit** 이란 함수를 쓰면, G4가 적당한 단위계로 표현해준다. (length, time, energy, 등등을 명시해야된다.)
  - `cout << G4BestUnit(StepSize, "Length") << endl;`
- **G4UnitDefinition::PrintUnitsTable** 을 부르면 G4가 쓰는 모든 단위를 볼 수 있다.

# System of units (4/4)

- 새로운 단위계를 쓰고 싶다면 **SystemOfUnits.hh** 을 개조하거나,
  - `#include "SystemOfUnits.hh"`
  - `static const G4double inch = 2.54*cm;`
- **G4UnitDefinition** 클래스를 사용해 새로운 단위계를 만들면 된다.
  - `G4UnitDefinition (name, symbol, category, value)`
  - `G4UnitDefinition ("inch", "in", "Length", 25.4*mm) ;`

# 3-Vectors (1/2)

- Geant4 는 따로 벡터클래스를 만들어 쓰기보다는 CLHEP 의 `HepVector3D`, `Hep3Vector` 를 typedef 해서 쓴다
  - 중복투자는 바보들만 한다 !
- 예를 들면 ,
  - `G4ThreeVector *p=new G4ThreeVector(10,20,100);`
- CLHEP 에서 정의한 멤버함수를 부르면 된다.
  - `G4double px=p->x();`
  - `p->setZ(50);`
- 구면좌표계로 표현하고 변환할 수 있다.
  - `p->phi(); p->theta(); p->mag();`
  - `p->setPhi(); p->setTheta(); p->setMag();`
- 단점은 **Coding convention** 이 G4 와 맞지 않는다.
  - 물론 `function overloading` 을 하면 되겠지만 ...

# 3-Vectors (2/2)

- 벡터를 규격화할 땐 ,
  - `p->unit()` ;
- Y 축을 중심으로 2.73 라디안을 회전하면 ,
  - `p->rotateY(2.73)` ;
- 또는 어떤 벡터를 중심으로 1.57 라디안을 돌리면 ,
  - `p->rotate(1.57, G4ThreeVector(10, 20, 30))` ;
- 더 자세히 공부하려면 벡터클래스설명을 참조하자 .
  - `$CLHEP_BASE_DIR/include/CLHEP/Vector/ThreeVector.h`
- See the CLHEP manual
  - `wwwinfo.cern.ch/asd/lhc+`  
`+/clhep/manual/RefGuide/Vector/Hep3Vector.html`

# Rotation Matrices (1/2)

- 검출기설계에 가장 많이 쓰이는 회전매트릭스 역시 CLHEP 에서 가져왔다.
- **G4RotationMatrix = HepRotation**
  - `#include "G4RotationMatrix.hh"`
  - `G4RotationMatrix *rm = new G4RotationMatrix;`
- **x 축을 중심으로 45 도 돌리면,**
  - `rm->rotateX(45*deg);`
- **어떤 벡터를 중심으로 회전시키려면,**
  - `rm->rotate(45*deg, Hep3Vector(1., 1., .3));`
- **회전 매트릭스의 Inversion 은**
  - `rm->invert();`

# Rotation Matrices (2/2)

- 회전후의 변화된 각은 다음의 함수들을 불러 알 수 있다.
  - `phiX()` , `phiY()` , `phiZ()` , `thetaX()` , `thetaY()` , `thetaZ()`
- 어떤 벡터와의 각을 특정 함수를 사용하여 알 수 있다.
  - `G4double angle;`
  - `G4ThreeVector axis;`
  - `rm->getAngleAxis(angle, axis);`
- 더 자세한 사항은
  - `$CLHEP_BASE_DIR/include/CLHEP/Vector/Rotation.h`
  - `wwinfo.cern.ch/asd/lhc+`  
`+/clhep/manual/RefGuide/Vector/HepRotation.html`

# **Lecture 5:**

## **GEANT4**

### **Getting Started**

# GEANT4 예제의 소개

- G4 는 다양한 예제를 제공하고 있다 .
- 우선 초보자 예제를 공략하도록 하자 .

```
Terminal — bash — 81x28
icpark@localhost:~> ls $G4INSTALL/examples/*
/Users/icpark/GEANT4/geant4.9.2.p01/examples/GNUMakefile
/Users/icpark/GEANT4/geant4.9.2.p01/examples/History
/Users/icpark/GEANT4/geant4.9.2.p01/examples/README

/Users/icpark/GEANT4/geant4.9.2.p01/examples/advanced:
GNUMakefile      gammaray_telescope      purging_magnet
README           hadrontherapy           radiation_monitor
Rich             human_phantom           radioprotection
Tiara            lAr_calorimeter        underground_physics
air_shower       medical_linac           xray_fluorescence
brachytherapy    microbeam               xray_telescope
composite_calorimeter  microdosimetry
cosmicray_charging  nanobeam

/Users/icpark/GEANT4/geant4.9.2.p01/examples/extended:
analysis         g3tog4                  persistency
biasing          geometry                polarisation
electromagnetic  hadronic                radioactivedecay
errorpropagation  medical                runAndEvent
eventgenerator   optical                 visualization
exoticphysics    parallel
field            parameterisations

/Users/icpark/GEANT4/geant4.9.2.p01/examples/novice:
GNUMakefile      N02                     N04                     N06                     README
N01              N03                     N05                     N07
icpark@localhost:~>
```

# 작업공간 구성과 Makefile 의 제작

- 작업 디렉터리를 만들고, include 와 src 만든다.
  - mkdir nps09 ; cd nps09
  - mkdir include ; mkdir src
- Makefile 을 만든다.
  - vi Makefile
    - ROOT 와 같이 컴파일하도록 Flag 를 셋팅한다.

```
> more Makefile

name      := nps09
G4TARGET := $(name)
G4EXLIB   := true

.PHONY: all
all: lib bin

include $(G4INSTALL)/config/binmake.gmk
# to compile with ROOT
CPPFLAGS += $(shell $(ROOTSYS)/bin/root-config --cflags)
LDFLAGS  += $(shell $(ROOTSYS)/bin/root-config --glibs)
```

# main() 함수 만들기

- Geant4 는 검출기 전산모사용 프로그램 라이브러리로 그 자체가 실행되는 어플리케이션이 아니다!
  - 사용자가 main() 을 만들어 실행파일을 만들어 사용해야 한다
- 사용자의 메인에 **G4RunManager** 를 도입하면 된다.
- 가장 기초적인 main() 함수의 모습
  - vi nps09.cc

```
> more nps09.cc

#include "G4RunManager.hh"
int main()
{
    G4RunManager *rm=new G4RunManager();

    delete rm;
    return 0;
}
```

# G4의 핵심 G4RunManager

- **G4RunManager**는 Geant4 hierarchy의 가장 밑바탕이 되는 클래스이다.
- **G4RunManager**가 하는 일 1: 검출기를 설정한다.
  - `SetUserInitialization()`
- **G4RunManager**가 하는 일 2: 시뮬레이션에 쓰일 입자를 등록하고 어떤 물리과정을 시뮬레이션 하는가를 설정한다.
  - `SetUserInitialization()`
- **G4RunManager**가 하는 일 3: 초기 입자의 생성과 생성지점 설정
  - `SetUserAction()`
- **G4RunManager**가 하는 일 4: event loop를 돌린다.
  - `BeamOn()`

# 반드시 있어야 할 3 개의 사용자 클래스

- 실험의 삼요소 :

- 1: 검출기 → Initialization (Detector Construction)
- 2: 입자빔 → Initialization (Physics List)
- 3: 충돌 이벤트 → Action (Primary Generation)

- Geant4 는 이들 세 개의 필수적인 클래스의 기초 구성을 abstract 클래스로 설정해 놓았다.

- 사용자는 이들 abstract class 를 상속해 사용하면 된다 .

- **DetectorConstruction**

- G4VUserDetectorConstuction

- **PhysicsList**

- G4VUserPhysicsList

- **PrimaryGeneratorAction**

- G4VUserPrimaryGeneratorAction

G4RunManager 가  
Initialize() 와  
BeamOn() 이 불릴 때 3  
개의 클래스의 지정을  
점검한다 .

# 두 개의 필수적인 **Initialization** 클래스

- **G4UserDetectorConstruction (initialization)**

- 검출기의 구성물질 , 지오메트리 , 센서들 , 데이터 readout 등을 정의한다 .
  - Materials
  - Geometry of the detector
  - Definition of sensitive detectors
  - Readout schemes

- **G4UserPhysicsList (initialization)**

- 전산모사에 사용될 입자들의 출석을 부른다 .
- 또한 입자들의 붕괴되고 생성되는 과정들 (Process 라고 부른다 ) 을 등록한다 .
- 각 입자들의 시뮬레이션을 언제 끝내는가를 설정한다 . `cutoff parameters`

# 한 개의 필수적인 **action** 클래스

- **G4VUserPrimaryGeneratorAction** (필수)
  - Primary event kinematics
- **G4UserRunAction** (optional)
  - run by run
- **G4UserEventAction** (optional)
  - event by event
- **G4UserStackingAction** (optional)
  - to control the order with which particles are propagated through the detector
- **G4UserTrackingAction** (optional)
  - Actions to be undertaken at each end of the step
- **G4UserSteppingAction** (optional)
  - Actions to be undertaken at the end of every step

# 가장 단순한 **main()** 작업

```
#include "G4RunManager.hh"
#include "MyDetectorConstruction.hh"
#include "MyPhysicsList.hh"
#include "MyPrimaryGeneratorAction.hh"

int main ()
{
    G4RunManager* runMan=new G4RunManager();
    runMan->SetUserInitialization(new MyDetectorConstruction);
    runMan->SetUserInitialization(new MyPhysicsList);
    runMan->SetUserAction(new MyPrimaryGeneratorAction);
    runMan->Initialize();

    runMan->BeamOn(1);
    delete runMan;
    return 0;
}
```

# 매크로 파일을 이용한 **Batch** 작업 (1/2)

```
int main(int argc, char** argv)
{
    G4RunManager* runMan = new G4RunManager;
    runMan->SetUserInitialization(new MyDetectorConstruction);
    runMan->SetUserInitialization(new MyPhysicsList)
    runMan->SetUserAction(new MyPrimaryGeneratorAction);
    runMan->Initialize();

    // read a macro file
    G4UImanager* UI = G4UImanager::GetUIpointer();
    G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UI->ApplyCommand(command+fileName);

    // 여기 BeamOn() 대신 위의 Uimanager 가 매크로파일을 불러 들인다 .
    // 그 매크로안에 BeamOn 이 들어있다 .

    delete runMan;
    return 0;
}
```

# 매크로 파일을 이용한 **Batch** 작업 (1/2)

- 실행파일에 매크로 파일을 인자로 넣고 돌리면 된다.
  - `nps09 run.mac`
- 매크로 파일은 보통 아래와 같이 생겼다.

```
> more run.mac

# set verbose level for this run
/run/verbose 2
/event/verbose 0
/tracking/verbose 2
# 100 electrons of 1GeV Energy
/gun/particle e-
/gun/energy 1 GeV
/run/beamOn 100
```

# Interactive 모드 (1/2)

- 인터랙티브 모드로 컴파일 하면 대화형으로 GEANT4 를 다룰 수 있다.

```
int main()
{
    G4RunManager* rm = new G4RunManager();
    ...
    rm->Initialize();
    // interactive terminal
    G4UIsession* session = new G4UIterminal();
    session->SessionStart();
    delete session;
    //
    delete rm;
    return 0;
}
```

# Interactive 모드 (2/2)

- Interactive 모드일 때는 그냥 실행을 시키고,
  - nps09 ↵
- Initialization 이 끝나면 아래의 프롬프트가 나타난다.

Idle>

- 이때부터 사용자 명령어를 써넣으면 된다.

```
Idle> /vis/create_view/new_graphics_system DAWN
```

```
Idle> /vis/draw/current
```

```
Idle> /run/verbose 1
```

```
Idle> /event/verbose 1
```

```
Idle> /gun/particle mu+
```

```
Idle> /tracking/verbose 1
```

```
Idle> /gun/energy 10 GeV
```

```
Idle> /run/beamOn 1
```

```
Idle> /vis/show/view
```

# G4UItcsh 의 설정

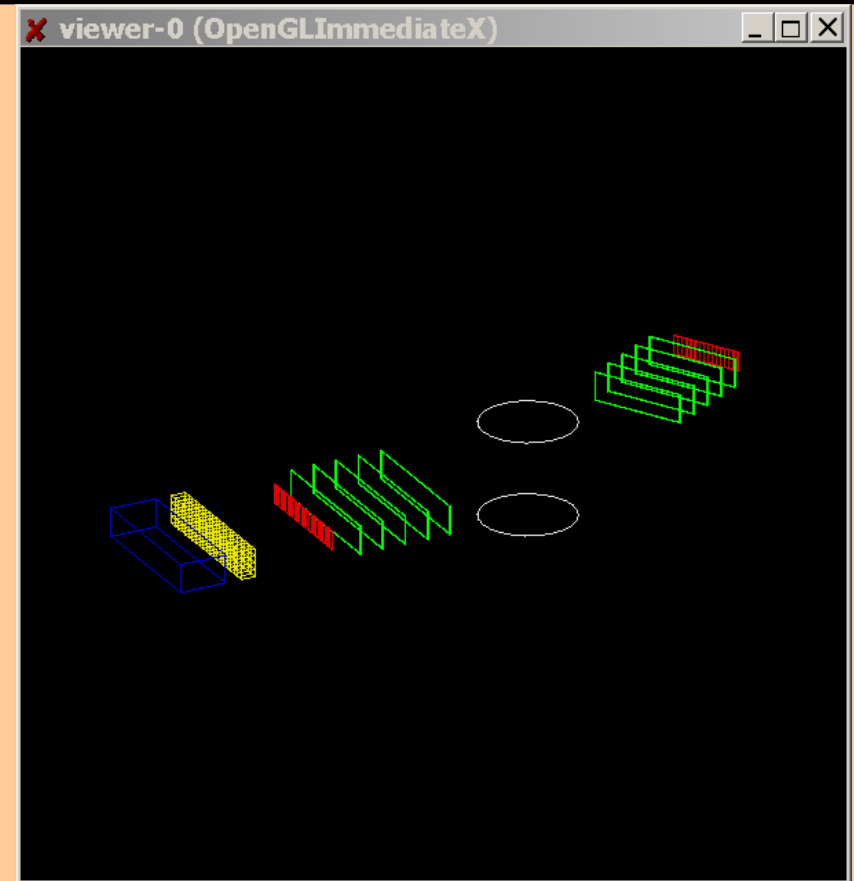
- Interactive 모드에서 명령어 리콜 기능을 사용하려면 G4UItcsh 을 설정해주면 된다.

```
#include "G4UItcsh.h"
#include "G4UItcsh.hh"

int main(int argc, char** argv)
{
...
//
G4UItcsh* term = new G4UItcsh(new G4UItcsh);
term->SessionStart();
delete term;
...
}
```

# OpenGL 을 사용한 시각화

- 가장 간단한 시각화 방법
- /vis/open OGLX 또는 OGLXm 등을 사용
- \$G4INSTALL/source/visualization 의 OpenGL 이 설치 되어야함
- GLX 문제가 Nvidia 쪽에 나타나나, X 를 사용하고 DISPLAY 를 설정하면 잘 작동 된다.
- 다양한 시각화가 제공된다.
  - DAWN, WIRED, VRML, etc.



**GEANT4:**

**Detector  
Construction**

# G4VUserDetectorConstruction

- **G4VUserDetectorConstruction** 는 GEANT4 가 제공하는 abstract 클래스로 사용자가 반드시 이를 상속해 사용자클래스를 제공해야만 한다 .
- 만들어진 **MyDetectorConstruction** 는 **G4RunManager** 가 **SetUserInitialization()** 을 호출 할 때 , 생성된다 .
- **G4VUserDetectorConstruction** 의 멤버함수 **Construct()** 가 **RunManager** 가 **Initialize()** 를 할 때 불려진다 .
- 따라서 , 사용자는 **Construct()** 을 **overload** 하여 자신의 검출기를 설계해 넣어야 한다 .
- 완성된 **Construct()** 는 **World Physical Volume** 의 포인터를 리턴 값으로 돌려준다 .

# 검출기만들기 1 단계 : Materials 설정

- 가상의 검출기를 만들기 위해서는 우선 **G4Material** 이라는 클래스를 알아야 한다.
- **G4Material** 클래스는 내부에 원소의 원자번호, 질량, 핵자수, 궤도에너지 등 원소의 특성을 다루는 **G4Element** 클래스와 동위원소를 다루기 위한 **G4Isotope** 를 가지고 있는데, GEANT4 사용자의 입장에서 **G4Material** 만 알면 된다.
- **G4Material** 클래스는 원소의 기본 성격 외에, density, state, temperature, pressure, radiation length, mean free path, dE/dx, 등 다양한 성질을 다룬다.

# 검출기만들기 1 단계 : 단순한 물질 정의하기

- 단순한 물질은 이름과 원자번호, 원자질량, density 만 주면 Material 이 만들어진다.

```
#include "G4Material.hh"
...
G4double z=18.;
G4double a=39.95*g/mole;
G4double density=1.390*g/cm3;
G4String name;
...
G4Material* LAr = new G4Material(name="Liquid
Argon",z,a,density);
```

- G4Material 포인터는 Logical volume 을 만들 때 사용된다.

# 검출기만들기 1 단계 : 분자만들기

- 원자들을 합쳐서 분자를 만들수도 있다.

```
#include "G4Element.hh"
#include "G4Material.hh"

G4double z,a;
G4String name,symbol;
G4Element* H=new G4Element
(name="Hydrogen",symbol="H",z=1,a=1.01*g/mole);
G4Element* O = new G4Element
(name="Oxygen",symbol="O",z=8.,a=16.0*g/mole);

G4int ncomponent,natoms;
G4Material* H2O = new G4Material
(name="Water",density=1.000*g/cm3,ncomponents=2);
H2O->AddElement(H,natoms=2);
H2O->AddElement(O,natoms=1);
```

# 검출기만들기 1 단계 : 복합물질 만들기

- 질소와 산소를 합쳐 공기를 만들 수 있다.

```
#include "G4Element.hh"
#include "G4Material.hh"
...
G4double z,a;
G4String name,symbol;
G4Element* N = new G4Element(name="Nitrogen",symbol="N",
z=7.,a=14.01*g/mole);
G4Element* O = new G4Element(name="Oygen",symbol="O",
z=8.,a=16.0*g/mole);

G4double fractionmass,density=1.290*mg/cm3;
G4int ncomponent,natoms;
G4Material* Air = new G4Material(name="Air",density,
ncomponents=2);
Air->AddElement(N,fractionmass=70*percent);
Air->AddElement(O,fractionmass=30*percent);
```

# 검출기만들기 1 단계 : 물질과 원소를 섞기

- Element 와 Material 을 섞어 혼합물도 만들수 있다.

...

```
G4double a,z,fractionmass,density;
```

```
G4String name,symbol;
```

```
G4int ncomponents,natoms;
```

```
G4Element* Ar = new G4Element(name="Argon",symbol="Ar",  
z=18.,a=39.95*g/mole);
```

```
G4Element* C = new G4Element(name="Carbon",symbol="C", z=6.,  
a=12.00*g/mole);
```

```
G4Element* O = new G4Element(name="Oxygen",symbol="O",  
z=8.,a=16.00*g/mole);
```

```
G4Material* CO2 = new G4Material(name="CO2",  
density=1.977*mg/cm3, ncomponents=2);
```

```
CO2->AddElement(C,natoms=1);
```

```
CO2->AddElement(O,natoms=2);
```

```
G4Material* ArCO2=new G4Material(name="ArCO2",  
density=1.8*mg/cm3, ncomponents=2);
```

```
ArCO2-> AddElement(Ar,fractionmass=93*percent);
```

```
ArCO2-> AddMaterial(CO2,fractionmass=7*percent);
```

...

# 검출기만들기 1 단계 : 온도와 압력의 고려

```
...
G4double a,z,fractionmass,density;
G4String name,symbol,ncomponent,natoms;
G4Element* Ar = new G4Element
    (name="Argon", symbol="Ar",z=18.,a=39.95*g/mole);
G4Element* C = new G4Element
    (name="Carbon", symbol="C", z=6., a=12.00*g/mole);
G4Element* O = new G4Element
    (name="Oxygen", symbol="O",z=8.,a=16.00*g/mole);

G4double T=300.*kelvin;
G4double P=2*atmosphere;
G4Material* CO2 = new G4Material
    (name="CO2",density=1.977*mg/cm3,ncomponents=2,kStateGas,T,P);
CO2->AddElement(C,natoms=1);
CO2->AddElement(O,natoms=2);

G4Material* ArCO2=new G4Material
    (name="ArCO2",density=1.8*mg/cm3,ncomponents=2 kStateGas,T,P);
ArCO2-> AddElement(Ar,fractionmass=93*percent);
ArCO2-> AddMaterial(CO2,fractionmass=7*percent);
...
```

# 검출기만들기 1 단계 : 물질과 원소 출력하기

- **Materials** 과 **elements** 은 스탠더드 출력에 바로 출력할 수 있다.

```
#include "G4Element.hh"
#include "G4Material.hh"
...
cout<<ArCO2;
cout<<* (G4Element::GetElementTable());
cout<<* (G4Material::GetMaterialTable());
```

# 검출기만들기 2 단계 : volume 이란 ?

- 검출기를 구성하는 각각의 입체를 volume 이라 부른다.
- 검출기를 설치할 가상의 실험실 공간을 World volume 이라 부른다.
- 각각의 volume 은 그 모양과 물리적 성질로 나타낼 수 있고, 어느 volume 안에 위치하게 된다. (World volume 빼고)
- 어느 volume 이 모 volume 에 놓이게 될 때, 모 volume 의 좌표계를 따른다.
- GEANT4 에서는 volume 이 오버랩 될 수 없다. 반드시 한 볼륨은 다른 어떤 볼륨에 속해 있게 된다.

# 지오메트리 : **Volume** 을 만드는 3 단계

1. 모양과 크기를 갖는 **Solid** 를 구성한다.
2. 위의 **Solid** 에 물리적인 특성 (**G4Material**) 을 부여하여 **Logical volume** 을 만든다.
3. 이 **Logical volume** 을 모 **volume** 의 적당한 위치에 삽입하여 **Physical volume** 을 만든다.

# 검출기만들기 2 단계 : Solids 의 이해

- 3D 입체를 표현하는 기법은 여러 개가 있다.
  - Constructive Solid Geometry (CSG)
  - SWEEP solids
  - Boundary Represented solids (BREPs)
- CSG 는 최소한의 파라미터로 입체를 기술하여 간단하고 효율이 좋으나 CAD 와 연동이 안 되는 단점이 있다. BREPs 는 아주 복잡한 입체도 그릴 수 있고 CAD 와도 연동이 되나 효율이 떨어진다.
- CSG solids 에는 Box, Tube, Cone, Spheres, Wedges, Torus 등 다양한 모양이 준비되어 있다.

# 검출기만들기 2 단계 : 가상실험실 만들기

- 네모난 박스를 가정하여 실험실을 만들자

```
#include "G4Box.hh"
```

```
...
```

```
G4double labx=3.0*m;
```

```
G4double laby=1.0*m;
```

```
G4double labz=1.0*m;
```

```
...
```

```
G4Box* labbox = new G4Box("Labbox", labx, laby, labz);
```

# 검출기만들기 3 단계 : Logical volume

- Material 과 Solid 가 준비되었다면 Logical volume 을 만들 수 있다.

```
#include "G4LogicalVolume.hh"
#include "G4Box.hh"
#include "G4Material.hh"
...
G4Box* a_box = new G4Box("A box", dx, dy, dz);
G4double a=39.95*g/mole;
G4double density=1.390*g/cm3;
G4Material* LAr = new G4Material(name="Liquid
Argon", z=18., a, density);
G4LogicalVolume* a_box_log = new G4LogicalVolume
(a_box, LAr, "a simple box");
```

# 검출기만들기 4 단계 : Logical volume 삽입

- 만들어진 Logical volume 을 모 volume 에 어디에 위치시키고 어떤 각도로 넣을지를 결정한다 .
- Physical volume 은 logical volume 에 위치정보 더한 instance 이다 .

```
#include "G4VPhysicalVolume.hh"
#include "G4PVPlacement.hh"
...
G4RotationMatrix *rot=new G4RotationMatrix;
rot->rotateX(30*deg);
G4Double aboxPosX=-1.0*m;
G4VPhysicalVolume* a_box_phys = new G4PVPlacement (rot,
G4ThreeVector(aboxPosX,0,0), a_box_log, "a box",
experimentalHall_log, false, 1);
```

# 검출기만들기 4 단계 : World volume 설정

- 유일한 예외로 World volume 은 최상위 volume 이라 어디 놓일 수 없다. 따라서 G4PVPlacement 에 null 포인터로 정의되며, rotation 도 정의해선 안 된다.

```
#include "G4PVPlacement.hh"
```

```
...
```

```
G4VPhysicalVolume* hall_phys = new G4PVPlacement(  
    0,          // No rotation  
    G4ThreeVector(0,0,0), // No Translation  
    hall_log,  // logical volume  
    "Experimental Hall", // its name  
    0,          // No mother volume  
    false,     // no boolean operations  
    0          // copy number  
);
```

# 추가작업 : G4VisAttributes 셋팅하기

- 각각의 Logical volume 은 visualization 을 위해 시각화 속성을 셋팅할 수 있다.

- void G4LogicalVolume::SetVisAttributes (const G4VisAttributes\* );

- void G4LogicalVolume::SetVisAttributes (const G4VisAttributes& );

- 사용자의 가상실험실을 wireframe 으로 나타내고, 하늘색으로 채우고 싶다면,

```
lab_log=new G4LogicalVolume (lab_box,Air,"lab");
```

```
...
```

```
G4VisAttributes *lab_vis=new  
G4VisAttributes (G4Colour (0.,1.,1.));
```

```
lab_vis->SetForceWireframe (true);
```

```
lab_log->SetVisAttributes (lab_viz);
```

**GEANT4:**

**Particles and  
physics  
processes**

# G4VUserPhysicsList

- G4VUserPhysicsList 클래스를 상속받아 전산모사에 관계된 입자들과 물리프로세스를 설정해 주어야만 한다.
  - 반드시 사용자가 제공해야 할 3 가지 코드 중 한 개임을 명심하자 .
- G4VUserPhysicsList 의 3 개의 함수를 오버로딩으로 제공 해야 한다.
  - ConstructParticle () // 입자들을 설정
  - ConstructProcess () // 프로세스를 설정
  - SetCuts () // 전산모사를 마칠 최소 단위

# Particles

- Geant4 내에 기본적인 입자들은 준비되어 있다.
  - electron, proton, gamma etc.
- 각각의 입자는 **G4ParticleDefinition** 라는 기본 클래스를 상속하여 각각의 클래스로 정의된다.
- **G4ParticleDefinition** 은 입자의 이름, 질량, 전하량, 스핀 등을 데이터 멤버로 가지고 있고 이들 값은 바꿀 수 없다. (평균수명, 붕괴율등은 셋팅가능)
  - 물론 소스코드레벨에서 바꾸고 다시 컴파일하면 강제로 바꿀 수는 있겠다.
- 주요한 입자클래스들로는, **leptons, bosons, mesons, shortlived, baryons, ions** 등이 있다.

# G4ParticleTable 의 소개

- 한 입자는 **singleton** 으로 **static object** 로 정의된다.
  - `G4Electron` // 전자를 기술하는 클래스
  - `G4Electron::theElectron` // 전자
  - `G4Electron::ElectronDefinition()` // pointer to 전자
- 입자는 **static** 이므로 프로그램 어디서나 쓸 수 있다.
- 입자의 클래스를 **G4ParticleTable** 에 모아두었다. 이를 이용해 **G4** 내에서 입자들을 꺼낼 수 있다.
  - `FindParticle(G4String name)` // 이름으로 찾기
  - `FindParticle(G4int PDGencoding)` // PDG 코드로 찾기
- **G4ParticleTable** 역시 **singleton** 으로 정의된다.
  - `G4ParticleTable::GetParticleTable()`
    - returns the pointer to the Particle table

# Particle construction

- 함수 `G4VUserPhysicsList::ConstructParticle()` 는 전사 모사에 사용될 모든 입자들을 만들기 위해 사용자가 반드시 제공해야 한다.

```
#include "G4Geantino.hh"
#include "G4Electron.hh"
#include "MyPhysicsList.hh"
void MyPhysicsList::ConstructParticle()
{
    G4Proton::ProtonDefinition();
    G4Electron::ElectronDefinition();
}
```

# ConstructProcess()

- ConstructProcess() 는 버추얼 함수이고 사용자가 프로세스를 만들어 사용하는 입자에 등록해 주어야 한다.
- ConstructParticle() 를 통해 선언된 입자들마다 **G4ProcessManager** 를 가져야 하고, **AddProcess(G4VProcess \*)** 를 통해 physics process 들을 등록해야 한다.
- 크게 7가지 physics process 로 나눌 수 있다.
  - transportation
  - electromagnetic (EM processes)
  - hadronic
  - decay
  - optical
  - photolepton\_hadron
  - parametrization

# AddTransportation()

- Geant4에서는 입자의 진행을 프로세스로 본다.
  - 모든 입자는 Transportation 프로세스를 반드시 거쳐야만 한다.
- G4VUserPhysicsList의 AddTransportation() 라는 함수를 ConstructPhysics() 에서 불러주면 된다.

```
void G4VUserPhysicsList::AddTransportation()
{
    G4Transportation* transport=new G4Transportation;
    PartIterator->reset();
    while ((*PartIterator)()) {
        G4ParticleDefinition *particle=PartIterator->value();
        G4ProcessManager* pman=particle->GetProcessManager();
        if (!particle->IsShortLived()) {
            pman->AddProcess(transportation);
            pman->SetProcessOrderingToFirst(transport,istep);
        }
    }
}
```

# G4ProcessManager

- **G4ProcessManager** 은 **G4ParticleDefinition** 의 데이터 멤버다 .
- **G4ProcessManager** 는 각 입자들이 격어야 할 **Physics process** 들의 리스트를 가지고 있고 , 어떤 프로세스부터 처리 하는가 하는 처리순서를 관장한다 .
- 각 입자에 대해 **G4ProcessManager** 의 **AddProcess()** 와 **SetProcessOrdering()** 를 통해 **Physics process** 를 등록해주어야한다 .
  - **AddAtRestProcess()** , **AddContinuousProcess()** , **AddDiscreteProcess()** 등도 사용할 수 있다 .
- **ActivateProcess()**, **InActivateProcess()** 를 사용하여 프로세스들을 켜다 켜다 할 수 있다 .

# ConstructProcess() for gammas

- 감마입자에 관련된 물리 프로세스를 정의 해보자.
  - photo electric effect
  - Compton scattering
  - gamma conversion:  $\gamma \rightarrow e^- + e^+$

```
void MyPhysicsList::ConstructProcess()
{
    AddTransportation();
    ConstructEM();
}

void MyPhysicsList::ConstructEM()
{
    G4ParticleDefinition* gamma = G4Gamma::GammaDefinition();
    G4ProcessManager *pman= gamma->GetProcessManager();

    G4PhotoElectricEffect *thePhotoElectricEffect = new G4PhotoElectricEffect;
    G4ComptonScattering *theComptonScattering = new G4ComptonScattering;
    G4GammaConversion *theGammaConversion = new G4GammaConversion;

    pman->AddDiscreteProcess(thePhotoElectricEffect);
    pman->AddDiscreteProcess(theComptonScattering);
    pman->AddDiscreteProcess(theGammaConversion);
}
```

# G4의 물리 프로세스들

- GEANT4 내장 물리 프로세스들

- G4ComptonScattering
- G4GammaConversion
- G4PhotoElectricEffect
- G4eMultipleScattering, G4hMultipleScattering
- G4eIonisation, G4MuIonisation, G4hIonisation, G4ionIonisation
- G4eBremsstrahlung, G4MuBremsstrahlung, G4hBremsstrahlung
- G4eplusAnnihilation
- G4MuPairProduction, G4hPairProduction

- 그리고 입자들의 붕괴프로세스

- G4Decay

# Setting cuts (1/3)

- **SetCuts()** 은 G4VUserPhysicsList 의 virtual 멤버함수이다.
- 지오메트리의 설정, 입자들의 등록, 물리프로세스의 설정이 끝난 뒤에 **SetCuts()** 을 불러야 한다.
- **SetCuts()** 을 길이로 설정하면 지오메트리에 설정된 물질 안에서 cut-off 에너지로 바뀌어, 입자들이 이 에너지보다 작은 경우가 되면 그 입자에 대한 전산모사를 마친다.
- G4VUserPhysicsList 에는 기본 Cut 값이 정해져 있다. G4RunManager 가 **SetCutsWithDefault()** 를 호출하면, 이 기본값이 **SetCuts()** 의 값으로 사용된다.
  - 기본 cut 값은 1mm 이다 .

# Setting cuts (2/3)

- `GetLengthCuts()` 을 호출하여 현재 cut 값을 알 수있다.
- 각 물질에서의 최소 에너지를 얻으려면
  - `GetEnergyThreshold(G4Material *)`
- 각각의 입자들이 서로 다른 cut 값을 사용하게 하려면, 입자들의 상관관계를 잘 따져보아야 한다.
  - gamma, electron/positron, proton/antiproton, others
- cut 을 정의하기 위해 `SetCuts()` 함수 외에 다양한 함수들도 제공된다.
  - `SetCutValue(G4double cut, G4String name)`
  - `SetCutValueForOthers(G4double cut)`
  - `SetCutValueForOtherThan(G4double cut, G4ParticleDefinition* particle)`

# Setting cuts (3/3)

- SetCuts() 의 예를 들면

```
void MyPhysicsList::SetCuts ()
{
    const G4double cut=0.1*mm;
    SetCutValue (cut, "gamma" );
    SetCutValue (cut, "e+" );
    SetCutValue (cut, "e-" );
    SetCutValue (cut, "proton" );
    SetCutValueForOthers (cut) ;
}
```

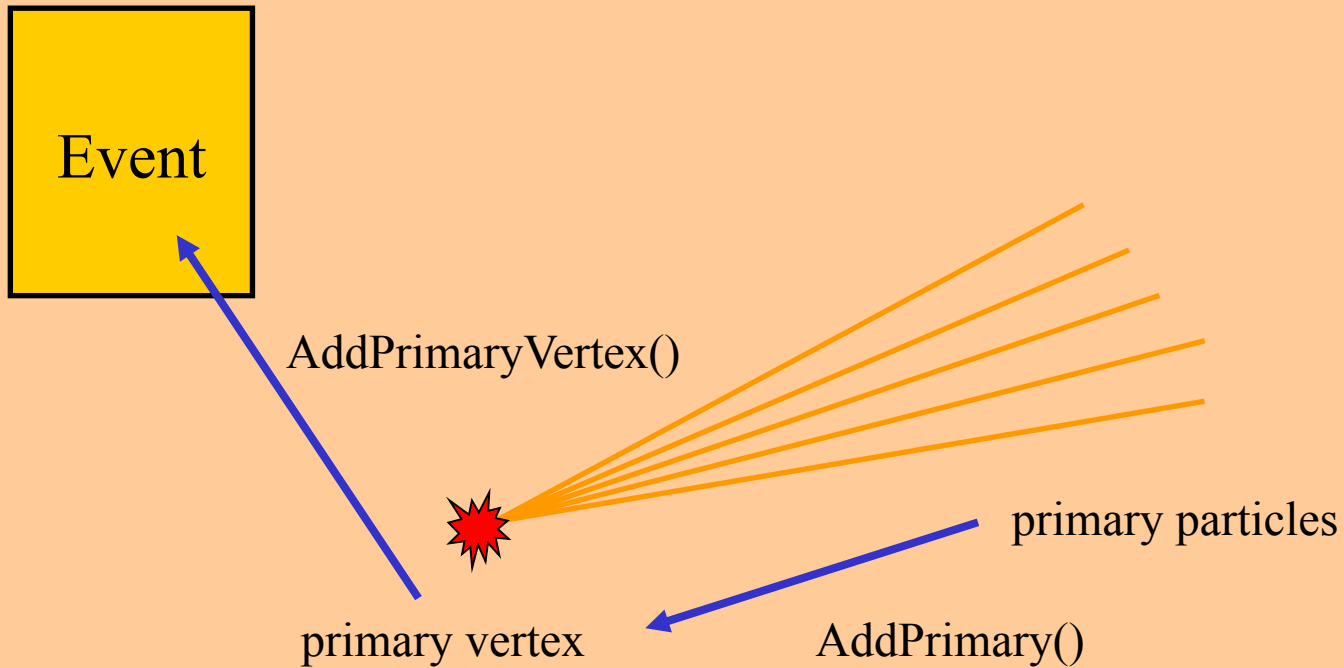
# 연습문제

- MyPhysicsList 의 ConstructParticle() 안에 electron, positron, mu+ 와 mu- 를 등록해 보아라.
- 등록된 입자들에 아래의 물리프로세스를 등록하여라.
  - electrons: multiple scattering, ionization, Bremsstrahlung
  - positron: + annihilation
  - muons: multiple scattering, ionization, Bremsstrahlung, pair production
- SetCuts() 으로 위 입자들의 적당한 cut 을 도입하여라.
- Particle iterator 와 FindParticle() 의 사용하여 원하는 입자를 끌어내 보아라.
- protone 을 쏘고, calorimeter 에서의 shower develop 을 관찰해 보아라.

**GEANT4:**

**Primary  
Generation  
(Kinematics)**

# Event 의 구성



# Event 의 구성

- Event 는 많은 Primary particle 들로 구성된다.
  - `G4PrimaryParticle (PDGcode , Px , Py , Pz)`
  - `G4PrimaryParticle (G4ParticleDefinition * , Px , Py , Pz)`
- Event 에는 Primary vertex 가 있다.
  - `G4PrimaryVertex (Vx , Vy , Vz , T0)`
  - `G4PrimaryVertex (G4ThreeVector , T0)`
- `G4PrimaryVertex` 의 멤버함수인 `AddPrimary` 를 이용해 primary particle 들을 vertex 에 등록한다.
  - `AddPrimary (G4PrimaryParticle* aParticle)`
- `G4Event` 의 `AddPrimaryVertex` 를 통해 Vertex 를 등록한다.
  - `AddPrimaryVertex (G4PrimaryVertex *aVertex)`

# Event 의 구성

- Event 는 많은 Primary particle 들로 구성된다.
  - `G4PrimaryParticle (PDGcode , Px , Py , Pz)`
  - `G4PrimaryParticle (G4ParticleDefinition * , Px , Py , Pz)`
- Event 에는 Primary vertex 가 있다.
  - `G4PrimaryVertex (Vx , Vy , Vz , T0)`
  - `G4PrimaryVertex (G4ThreeVector , T0)`
- `G4PrimaryVertex` 의 멤버함수인 `AddPrimary` 를 이용해 primary particle 들을 vertex 에 등록한다.
  - `AddPrimary (G4PrimaryParticle* aParticle)`
- `G4Event` 의 `AddPrimaryVertex` 를 통해 Vertex 를 등록한다.
  - `AddPrimaryVertex (G4PrimaryVertex *aVertex)`

# G4VUserPrimaryGeneratorAction

- **G4VUserPrimaryGeneratorAction** 는 사용자가 제공해야 하는 마지막 필수적인 함수로 이를 상속해 자신만의 **Primary Generator Action** 을 만들면 된다.
- **primary particle** 들이 어떻게 생성되는가를 이 클래스에 넣는다.
- **G4VUserPrimaryGeneratorAction** 는 **GeneratePrimaries()** 라는 **virtual** 함수를 가지고 있다. 이 함수는 매 **event** 때 마다 한번씩 자동으로 불린다. 이 함수의 인자로 **event** 포인터를 넘겨주므로, **event** 에 **primary particle** 들을 등록 시킬 수 있다.
  - `GeneratePrimaries (G4Event* anEvent) ;`

# G4VPrimaryGenerator

- **G4VPrimaryGeneratorAction::GeneratePrimaries()** 가 event 에 입자들을 채워 넣을 때 다양한 kinematics 를 사용할 수 있게 하도록 G4VPrimaryGenerator 라는 abstract 클래스를 상속해 사용자만의 Primary Generator 를 만들 수 있다.
- 여기서 Primary event 를 구성하면 되겠다.
  - `GeneratePrimaryEvent(G4Event* an Event)`
- 사용자는 PrimaryGeneratorAction 에서 구비되어 있는 다양한 PrimaryGenerator 들 중 한 개를 골라 쓰면 된다.
- G4VUserPrimaryGeneratorAction 는 매 event 의 프로세스 때 virtual 멤버 함수인 **GeneratePrimaries()** 를 호출한다. 이때 G4VPrimaryGenerator 의 **GeneratePrimaryVertex()** 를 호출하면 된다.

# MyPrimaryGeneratorAction.hh 의 예

```
#include "G4VUserPrimaryGeneratorAction.hh"
```

```
class MyPrimaryGenerator;
```

```
class G4Event;
```

```
class MyPrimaryGeneratorAction : public  
    G4VUserPrimaryGeneratorAction
```

```
{
```

```
    public:
```

```
        MyPrimaryGeneratorAction();
```

```
        ~MyPrimaryGeneratorAction();
```

```
    public:
```

```
        void GeneratePrimaries(G4Event* anEvent);
```

```
    private:
```

```
        MyPrimaryGenerator *TestBeam;
```

```
};
```

# MyPrimaryGeneratorAction.cc 의 예

```
#include "MyPrimaryGeneratorAction.hh"
#include "MyPrimaryGenerator.hh"

MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    TestBeam=new MyPrimaryGenerator();
}

MyPrimaryGeneratorAction::~MyPrimaryGeneratorAction()
{
    delete TestBeam;
}

void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event*
    anEvent)
{
    TestBeam->GeneratePrimaryVertex(anEvent);
}
```

# MyPrimaryGenerator.hh 의 예

```
#include "G4VPrimaryGenerator.hh"
#include "G4ThreeVector.hh"
#include "G4PrimaryVertex.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleMomentum.hh"
#include "G4Event.hh"

class MyPrimaryGenerator: public G4VPrimaryGenerator
{
public:
    MyPrimaryGenerator();
    void GeneratePrimaryVertex(G4Event *evt);
private:
    G4ParticleDefinition *beam;
    G4ParticleMomentum    beamP;
    G4double               beamE;
    G4ThreeVector         beamV;
    G4double               beamT;
};
```

# MyPrimaryGenerator.cc 의 예

```
#include "MyTestBeam.hh"
MyPrimaryGenerator::MyPrimaryGenerator()
{
    beam=G4Electron::ElectronDefinition();
    beamE=10*GeV;
    beamP=G4ParticleMomentum(1.0,0.0,0.0);
    beamV=G4ThreeVector(0.0,0.0,-100.0);
    beamT=0.0;
}
MyPrimaryGenerator::GeneratePrimaryVertex(G4Event *evt)
{
    G4PrimaryVertex *vtx = new G4PrimaryVertex(beamV,beamT);
    G4double px=beamE*beamP.x();
    G4double py=beamE*beamP.y();
    G4double pz=beamE*beamP.z();
    G4PrimaryParticle *part=new
G4PrimaryParticle(beam,px,py,pz);
    vtx->AddPrimary(part);
    evt->AddPrimaryVertex(vtx);
}
```

# G4ParticleGun

- **G4ParticleGun** 은 Geant4 가 제공하는 아주 기본적인 Primary generator 이다 .
- **G4ParticleGun** 는 아래의 다양한 함수를 사용해 Primary 입자들을 만든다 .

```
void SetParticleDefinition(G4ParticleDefinition*)  
void SetParticleMomentum(G4ParticleMomentum)  
void SetParticleMomentumDirection(G4ThreeVector)  
void SetParticleEnergy(G4double)  
void SetParticleTime(G4double)  
void SetParticlePosition(G4ThreeVector)  
void SetParticlePolarization(G4ThreeVector)  
void SetNumberOfParticles(G4int)
```

**GEANT4:**

**Run and Event**

- Run 은 실제 가속기 실험처럼 여러 개의 event 를 모은 단위이고, Geant4 에서 G4Run 클래스로 표현된다.
- G4RunManager 가 BeamOn() 함수를 부를 때 새로운 Run 이 생성된다. 한 개의 Run 내에서는 검출기의 geometry, Physics 프로세스 등을 바꿀 수 없다.
- G4Run 은 다음의 중요한 정보들을 가지고 있다.
  - 고유의 run 번호 ( 실제 Geant4 는 이수를 사용하지는 않는다 . )
  - run 안에 들어 있는 사건의 개수
  - Sensitive Detectors 에 설정된 G4VHitsCollection 테이블
  - Digitizers 에 설정에 따른 G4VDigiCollection 테이블

# G4RunManager (1/2)

- G4RunManager 는 Geant4 의 모든 시뮬레이션 과정을 컨트롤하는 가장 중요한 클래스다 .
- G4RunManager 는 *singleton* 이다 . 즉 프로그램 수행 중 오로지 한 개의 Run Manager 만 존재한다 .
- 사용자의 initialization 클래스들과 사용자의 action 클래스들은 **SetUserInitialization()** 과 **SetUserAction()** 의 의해 G4RunManager 에 등록되어야 한다 .
- **Initialize()**
  - `UserDetectorConstruction` 을 불러 검출기를 만든다 .
  - `UserPhysicsList` 를 불러 사용 할 입자들과 `physics processes` 를 설정한다 .
  - `cross-section tables` 을 계산한다 .

# G4RunManager (2/2)

- **BeamOn(G4int NumberOfEvents)**
  - 새로운 Run 을 만들고 , NumberOfEvents 만큼 event loop 을 돌린다 .
- **GetRunManager()**
  - singleton 인 G4RunManager object 의 포인터를 찾아준다 .
- **GetCurrentEvent()**
  - 현재 시뮬레이션 중인 event 의 포인터를 넘겨준다 .
- **SetNumberOfEventsToBeStored(G4int n)**
  - 여러 개의 사건을 모아서 저장할 때 쓴다 . BeamOn () 이 불려지기 전에 설정을 해야 한다 .
- **GetPreviousEvent(G4int i\_thPrevious)**
  - i 번째 전 사건의 포인터를 넘겨준다 .

# G4UserRunAction

- **G4UserRunAction** 는 사용자 **action** 클래스로 , 사용자가 이를 상속해 다음의 두개의 함수를 **overload** 하여 자신만의 **run** 셋팅을 할 수 있다.
  - **BeginOfRunAction ()**
    - **BeamOn()** 이 불리자마자 실행되고 , 이를 통해 **run** 고유번호와 **run** 정보를 담은 히스토그램을 설치하고 , 기타 **run** 에 관련된 셋팅을 한다 .
  - **EndOfRunAction ()**
    - **BeamOn()** 의 제일 마지막 단계에서 불린다 . 히스토그램을 닫고 , **run summary** 파일을 만들고 하는 작업을 넣을 수 있다 .

# Geant4 가 어떤 상태인지를 알라 !

- Geant4 가 현재 무엇을 하고 있는 상태 (state) 인지를 아는 것은 매우 중요하다. Geant4 의 상태는 G4RunManager 가 결정한다.
  - 초기화전단계 (PreInit state)
    - RunManager 가 초기화를 아직 부르지 않은 상태 . 지오메트리나 프로세스 , 컷값들이 바뀌면 이 상태가 된다 .
  - 초기화단계 (Init state)
    - G4RunManager 의 initialize() 가 수행되고 있는 단계 .
  - 휴식상태 (Idle state)
    - Geant4 가 무언가를 기다리며 작업을 쉬고 있는 단계
  - 빔조사단계 (GeomClosed state)
    - BeamOn() has been invoked
  - 사건진행상태 (EventProc state)
    - GetCurrentEvent(), GetPreviousEvent() 만 호출 가능하다 .
  - 종료단계 (Quit state)
    - G4RunManager 의 destructor 가 불려졌을 때 .

# Run 의 취소

- 문제가 발생하거나 원하지 않는 Run 을 시뮬레이션 중에 취소하려면 G4RunManager 의 **AbortRun()** 을 부르면 된다.
- **AbortRun()** 은 GeomClosed 상태나 EventProc 상태에서 만 부를 수 있다. (가만 생각해보면 당연한 소리다)
- 어떤 사건을 시뮬레이션 하는 도중 AbortRun() 이 불러서 Run 이 끝나게 되면, 그 Run 의 마지막 event 는 corrupted 됐었을 확률이 높으므로 분석에 사용하지 말자.

# Customizing G4RunManager

- G4RunManager의 모든 멤버 함수는 virtual 이므로 이를 상속해 자신만의 RunManager를 만들 수 있다.

```
- public:  
- virtual void Initialize();  
- virtual void DefineWorldVolume(G4VPhysicalVolume *);  
- virtual void AbortRun();  
- virtual void BeamOn(G4int n_events);  
- protected:  
- virtual void InitializeGeometry();  
- virtual void InitializePhysics();  
- virtual void InitializeCutOff();  
- virtual G4bool ConfirmBeamOnCondition();  
- virtual void RunInitialization();  
- virtual void DoEventLoop(G4int n_events);  
- virtual G4Event* GenerateEvent(G4int i_event);  
- virtual void AnalyzeEvent(G4Event* anEvent);  
- virtual void RunTermination();
```

# 검출기 **geometry** 의 변환

- Run 과 Run 사이에서는 검출기에 작은 변환을 가할 수 있다. ( 예를 들면 검출기를 조금씩 돌려 놓는다거나, 또는 검출기 특정부분의 두께를 조절한다거나 )
- 또는 Run 과 Run 사이에서 아예 기존의 검출기를 아예 없애고, 새로운 검출기를 놓을 수도 있다. 이 경우에는 World volume 을 G4RunManager 에 다시 등록 할 필요가 있다.

```
G4RunManager* runManager = G4RunManager::GetRunManager();  
MyNewGeometry newGeometry;  
G4VPhysicalVolume* newWorldPhys=newGeometry.Construct();  
runManager->DefineWorldVolume(newWorldPhys);
```

- 두 경우 모두, 지오메트리가 바뀐 사실을 G4RunManager 에게 알려야 한다.

```
- runManager->GeometryHasBeenModified()
```

- Geant4 에서 한 개의 물리사건은 G4Event 클래스로 기술된다 .
- G4Event object 는 G4RunManager 에 의해 생성되고 , 생성된 뒤에는 G4EventManager 의 관리를 받는다 .
- 시뮬레이션 중인 현재의 사건은 G4RunManager 의 **GetCurrentEvent()** 함수를 호출함으로써 얻을 수 있다 .
- G4Event 는 물리사건의 중요한 4 가지 정보를 가지고 있다 .
  - Primary vertexes 와 primary particles
  - Trajectories (G4TrajectoryContainer)
  - Hits collections (G4HCofThisEvent)
    - Sensitive detector 에 의해 생성
  - Digits collections (G4DCofThisEvent)
    - Digitizer 에 의해 생성

# G4EventManager

- **G4EventManager** 는 사건을 다루는 매니저 클래스이다.
  - G4PrimaryVertex 와 G4PrimaryParticle object 들을 현재의 G4Event object 와 G4Track object 들에 연결시킨다 . 그리고 , 모든 G4Track object 들을 G4StackManager 에 보낸다 .
  - G4StackManager 로 부터 한 개의 트랙 object 을 꺼내 , G4TrackingManager 에게 보낸다 . G4TrackingManager 가 트랙킹을 시도해보고 , 실패하면 “killed” 라고 표시한다 . ( 나중에 G4EventManager 가 killed 라 된 모든 트랙 object 들을 지운다 .)
  - In case the primary track is “suspended” or “postponed to the next event”, it is sent back to the G4StackManager. Secondary G4Track object returned by G4TrackingManager are also sent to G4StackManager
  - G4StackManager 가 더 이상 보내줄 것이 없으면 , G4EventManager 가 현재의 event processing 을 끝낸다 .

# G4UserEventAction

- **G4UserEventAction** 은 사용자 **action** 클래스로써 , 두 개의 **virtual** 함수를 사용자가 오버로드하여 물리사건을 사건 단위로 심층 조절할 수 있게 해준다 .
  - **BeginOfEventAction()**
    - **primary particles** 이 **G4Track objects** 로 등록되기 전에 불린다 .  
Primary particle 들을 조절하거나 , **event-by-event** 히스토그램을 셋팅할 때 쓰인다 .
  - **EndOfEventAction()**
    - **event** 프로세스의 마지막 단계에서 불린다 . **event-by-event analysis** 가 필요할 때 불린다 .

# G4UserSteppingAction

- **G4UserSteppingAction** 는 event 의 시작전과 마지막 단계에서의 action 이 아닌, 시뮬레이션 한 스텝 한 스텝 단계에 어떠한 action 이 필요할 때 쓰인다.
- **G4UserSteppingAction** 을 상속해 사용자만의 SteppingAction 을 만들고, 이를 RunManager 에 등록한다.
  - `void UserSteppingAction(const G4Step*)`
- 이 사용자 SteppingAction 함수는 트래킹 과정 중 매 스텝마다 불린다. 각 스텝에 있어서 필요한 정보는 G4Step 포인터를 통해 얻을 수 있다.

**GEANT 4:**

**Tracks & Hits**

# G4Step (1/2)

- G4Step 이란 사건을 작은 시간단위로 쪼갠 한 순간이라 보면 된다. G4Step 은 다음의 정보들을 갖는다.
  - pointers to PreStep and PostStepPoint
  - Geometrical step length
  - True step length (MS 을 고려 )
  - delta of position/time between PreStep and PostStepPoint
  - Delta of momentum/energy between PreStep and PostStepPoint
  - pointer to a G4Track
  - Total energy deposited during the step. This is the sum of:
    - Energy deposited by energy loss process
    - Energy loss by secondaries which have not been generated because their energies were below the cut threshold

- **G4StepPoint (PreStep and PostStepPoint) 의 정보들**
  - $(x, y, z, t)$
  - $(p_x, p_y, p_z, E_k)$
  - Pointers to the physical volumes
  - Safety
  - Beta, Gamma
  - Polarization
  - Step status
  - Pointer to the physics process for the current step
  - Pointer to the physics process for PreStep
  - Total track length
  - Global time
  - Local time
  - Proper time

# G4VHit

- Hit 이란 검출기의 센서영역에 트랙이 통과할 때 남기는 자취의 최소단위이다.
- G4 에서는 **G4VHit** 이란 베이스클래스를 제공하며, 사용자가 이를 상속해 자기의 실험에 맞게 **MyHit** 을 정의해 쓰면 된다.
- G4VHit 의 **Draw()** 와 **Print()** 라는 버추얼 함수를 오버로 드해 사용자에게 맞게 설정한다.
- event 내에서는 **G4VHitsCollection** 을 통해 Hit 들을 저장한다.
- G4THitsCollection 는 G4VHitsCollection 를 바탕으로 한 template 클래스로, 사용자의 **MyHit** 을 담을 수 있다.

# MyHit 의 예

```
- #include "G4VHit.hh"
- #include "G4THitsCollection.hh"
- class MyDetHit: public G4VHit {
  private:
    G4double edep;
    G4ThreeVector pos;
  public:
    MyDetHit();
    void Draw() const;
    void Print() const;
    inline void SetEdep(G4double de) {edep=de;}
    inline G4double GetEdep() const{return edep;}
    inline void SetPos(G4ThreeVector v) {pos=v;}
    inline G4ThreeVector GetPos() {return pos;}
};
```

# G4VSensitiveDetector

- **G4VSensitiveDetector** 는 실제 검출을 하는 검출기 부분을 기술하는 베이스 클래스이다.
- G4VSensitiveDetector 는 G4Step 의 지시에 따라 Hit 들을 만들다 . → ProcessHits()
- 하나의 sensitive detector class 는 반드시 고유명을 가져야 한다 . 다수의 sensitive 검출기를 갖는 경우에 대비해 , 고유명들이 디렉토리 구조를 가지도록 설계하는 것이 좋겠다 .
  - `myEMcal = new myEMcal ("/myDet/myCal/myEMcal") ;`
- 이 sensitive detector 의 포인터를 대응되는 G4LogicalVolume object 에 셋팅해주고 , G4SDManager 에 등록을 해야 한다 .
  - `scin_log->SetSensitiveDetector (myEMcal) ;`

# G4VSensitiveDetector (2)

- **G4VSensitiveDetector** 는 세 개의 중요한 virtual 함수가 있다.
  - **Initialize()**
    - 매 사건의 시작 때 불리어 지면 , 함수 인자로써 **G4HCofThisEvent** 를 받는다 . 이 Hit Collection 에 생성되는 Hit 들을 담는다 .
  - **ProcessHits()**
    - sensitive detector 를 포인팅하는 **G4LogicalVolume** 에 Step 이 벌어지면 **G4SteppingManager** 가 이 함수를 부른다 . 이때 **G4Step** object 과 ReadOut geometry 의 **G4TouchableHistory** object 가 인자로써 쓰인다 .
  - **EndOfEvent()**
    - 매 사건의 마지막에 불리어 , 각각의 Hit collection 을 **G4HCofThisEvent** 에 등록시킨다 .

# G4SDManager

- Sensitive 검출기에 관한 일은 G4SDManager 가 한다.
- SDM 은 singleton 으로 static 함수를 불러 얻을 수 있다

▪

– `G4SDManager::GetSDMpointer()`

- 모든 Sensitive Detector 들을 반드시 G4SDManager 에 등록시켜야 제대로 작동한다.

```
G4SDManager *sdman=G4SDManager::GetSDMpointer()  
sdman->AddSensitiveDetector (this);
```

- SDM 이 Hit collection ID 를 넘겨준다.

– `sdman->GetCollectionID("My Collection");`

# Hits Collections

- Hit collection 은 SD 생성 때 고유이름을 주어 만들 수 있다.  
`collectionName.insert("CalorimeterCollection");`
- 또는 SD 의 초기화 때 만들어도 된다.  
`caloHitsCollection=new CalorimeterHitsCollection  
(SensitiveDetectorName,collectionName[0]);`
- Hit 이 생성될 때 마다 collection 에 넣으면 된다.  
`int icell=caloHitsCollection->insert(caloHit);`
- 각각의 Hit collection 은 나중에 event 안에 등록시켜줘야 한다. either in SD's Initialize() or SD's EndOfEvent()  

```
static G4int HCID=-1;  
if (HCID<0) HCID=GetCollectionID(0);  
HCE->AddHitsCollection(HCID,caloHitsCollection);
```

# Access to the hits collections

- Hits collections are accessed for various purposes
  - Digitization
  - Event Filtering in G4VUserStackingAction
  - "End of Event" simple analysis
  - Drawing/printing hits
- SD Manager 를 통해 Hit Collection 을 access 한다.

```
G4SDManager* SDM=G4SDManager::GetSDMpointer();
G4RunManager* RM=G4RunManager::GetRunManager();
G4int cID=SDM->GetCollectionID("cName");
const G4Event* event=RM->GetCurrentEvent();
G4HCofThisEvent *HCofEvent=event->
GetHCofThisEvent();
MyHitsCollection *myCollection=
(MyHitsCollection *) (HCofEvent->GetHC(cID));
```