검출기 시뮬레이션 입문

2021/11/09

KCMS Lectures on Collider Physics

박인규
서울시립대학교

1

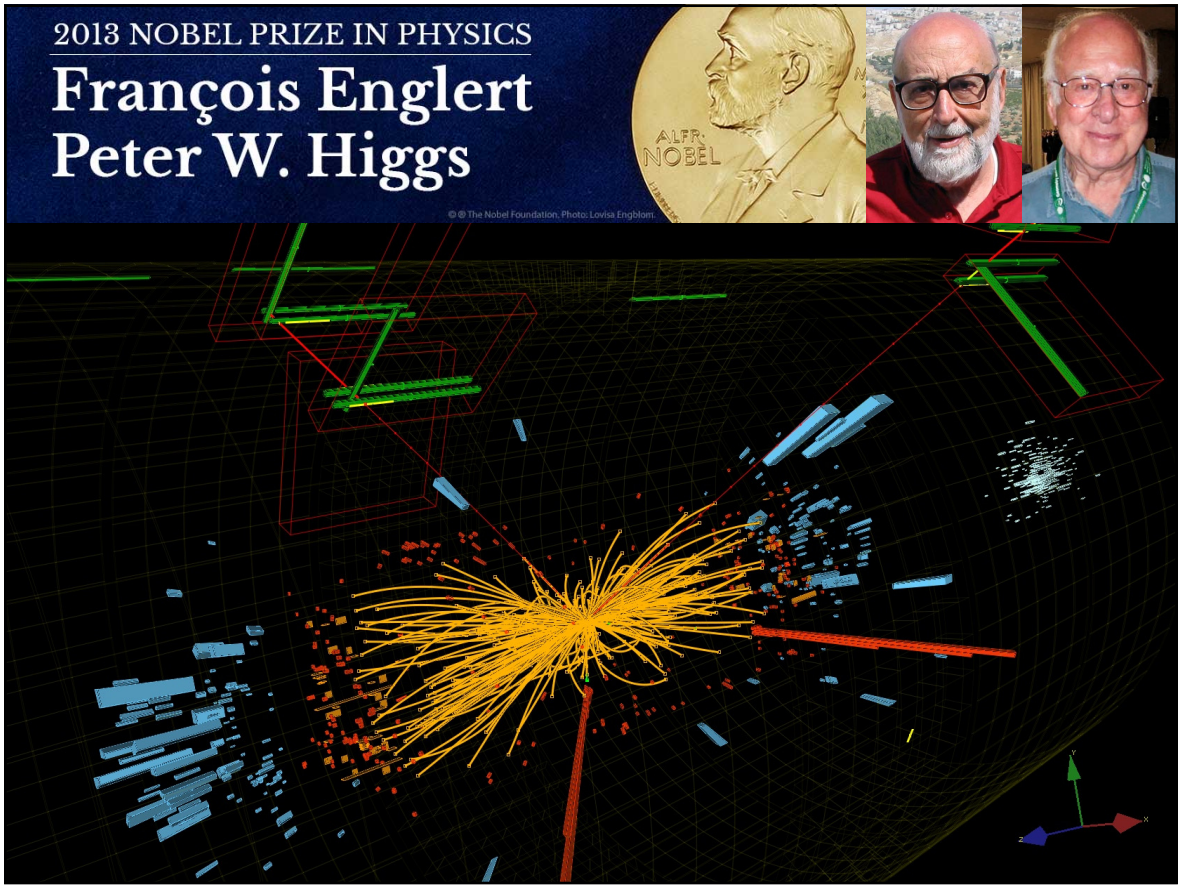
강의계획

- 제1부
  - 1강: GEANT4로의 초대
  - 2강: 물리학 기초지식: 물질과 상호작용
  - 3강: 몬테카를로 시뮬레이션
    - Case study
- 제2부
  - 0강: Geant4의 설치
  - 1강: GEANT4 문법과 CLHEP
  - 2강: Getting started
  - 3강: Visualization
  - 4강: Detector construction
  - 5강: Particles & Physics Processes
  - 6강: Kinematics
  - 7강: Run & Event
  - 8강: Tracks & Hits

2

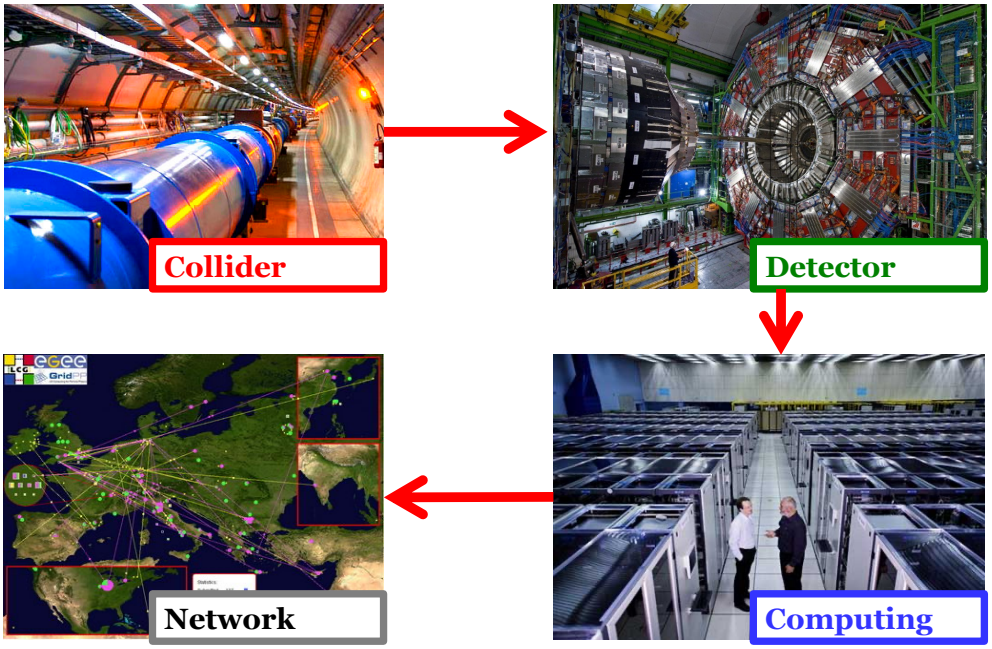
Geant4로의 초대

3



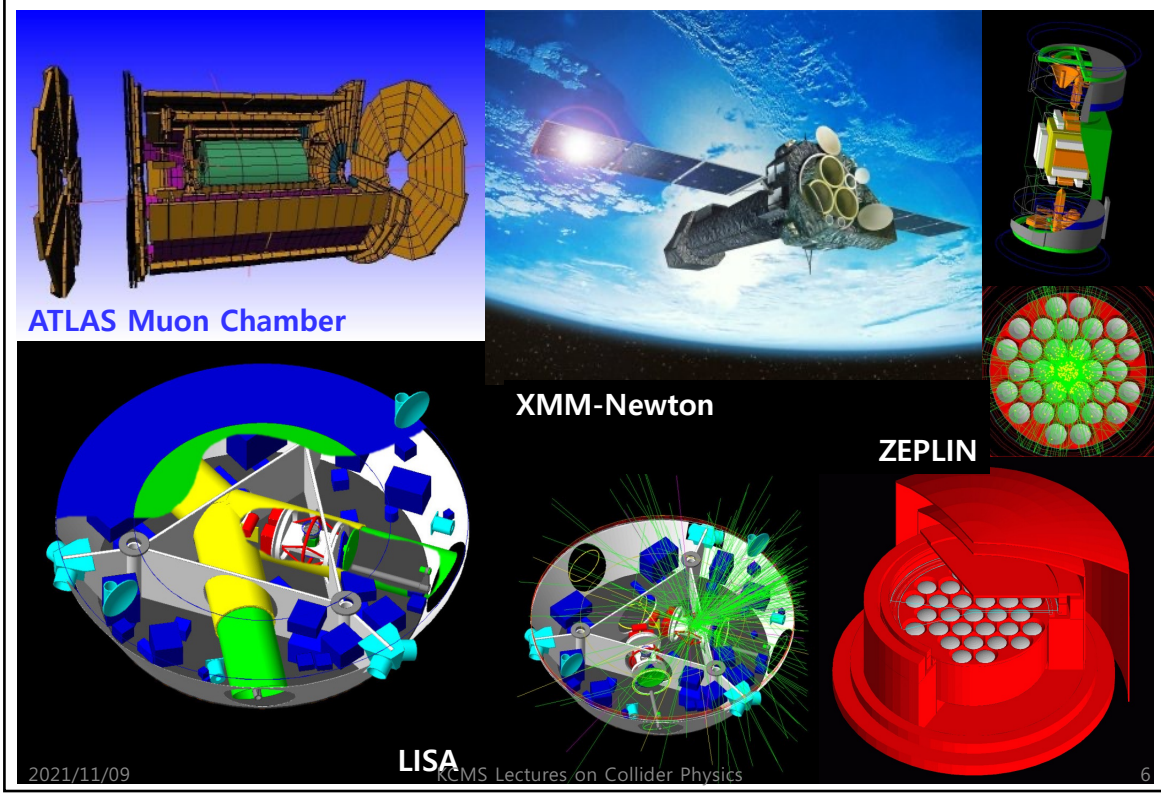
4

신의 입자를 찾는 과정.



5

검출기: 보이지 않는 것을 보이게 해주는 눈!

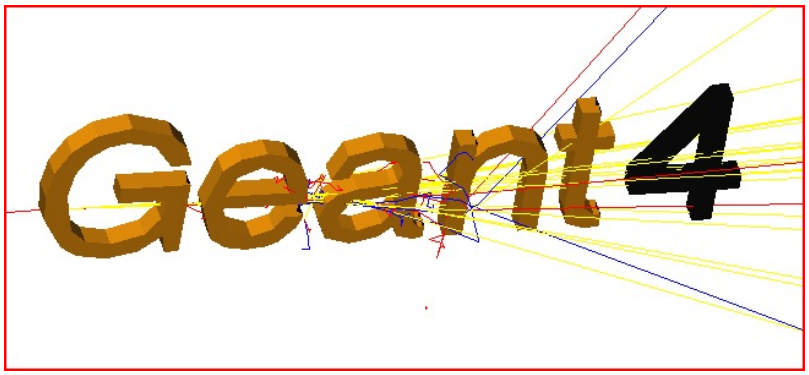


6

검출기 시뮬레이션이란?

- 전산모사란 실제 물리실험에서 나타나는 현상을 수치적으로 흉내내기를 하는 일을 말한다.
  - 우주선이나 가속기에서 나오는 입자들이 검출기와 어떻게 반응하는 가를 컴퓨터상에서 전산모사하는 것을 검출기 시뮬레이션이라 한다.
    - 검출기 성능을 최적화 하여 제작할 수 있다.
    - 검출기 성능을 미리 파악할 수 있다.
    - 실제 검출기에서 얻어진 신호의 성질을 이해할 수 있다.
- 검출기 시뮬레이터가 하는 일
  - 입자들의 생성과정과 붕괴과정을 모사
  - 입자들의 물질과 상호작용하는 정도를 계산
  - 물리반응에 관련된 정보를 출력하고, 시각화 자료를 제공

검출기 시뮬레이터의 표준!



GEANT

- GEANT(GEometry ANd Tracking)
  - 입자들이 매질 속을 지나갈 때 발생하는 물리현상을 몬테카를로 방법으로 계산하여 흉내내기를 하는 프로그램
  - 이름에 관하여
    - 보통 영어로 읽어 "지앤트"라고 발음하나, Géant는 불어로 giant 즉 거인을 뜻하고, "제앙"이라 발음하면 좀 개발자 느낌이 난다.
      - 물론 중요하지 않다!
- 역사
  - 최초의 프로그램은 1974년경 개발되어 사용되기 시작
  - GEANT3는 1982년 CERN라이브러리 패키지로 제공
    - FORTRAN으로 작성됨
    - 최종버전 GEANT3.21 (GNU GPL로 현재도 사용가능)
  - 1990년대 중반부터 CERN의 RD44팀에 의해 C++버전으로 다시 쓰여 Geant4로 계승됨.
  - Geant5 또는 GeantV 등이 개발 중이라 하지만 아직 비공식

GEANT3에서 Geant4로

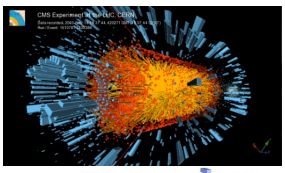
- GEANT3 (대표적 활용: CERN LEP실험)
  - 복잡한 지오메트리의 설치가 힘듦
    - CAD와의 연동 불가능
  - 전자기, 강한상호작용등의 물리 프로세스 업그레이드 불편
    - 100MeV-100GeV 정도까지만 작동
  - 복잡한 코드의 관리와 업그레이드가 불편
    - CVS, SVN 활용 이전
- Geant4 (대표적 활용: CERN LHC실험)
  - Object Oriented이므로 OO의 장점을 모두 갖는다.
    - 메모리관리, 클라스, 상속 등을 통한 체계적 프로그램
  - CAD 호환으로 복잡한 지오메트리를 다룰 수 있다.
    - CAD 프로그램 파일과 연동 가능
  - Physics프로세스의 업그레이드가 용이하다.
    - 확장된 영역의 물리프로세스 제공

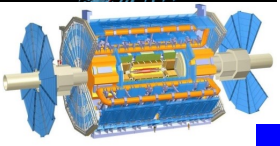
Geant4가 쓰이는 주요 분야

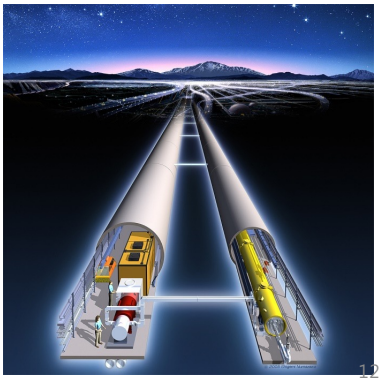
- 고에너지물리(High Energy Physics)
  - 힉스입자의 탐색, 중성미자, 암흑물질 탐색
- 우주과학(Space Science)
  - 우주환경, 우주선방호설계
- 방사선기술(Radiation Technology)
  - 내방사선반도체 설계, 감마선투시장비, 비파괴검사
- 의학(Medical Science)
  - PET설계, 생체피폭량 조사, 방사선치료 선량계산
- 산업응용(Industrial Applications)
  - 방사선 계측기, 검출기 제작

Geant4가 쓰이는 곳 (1/4)

- 고에너지물리(HEP) 연구
  - CERN(유럽핵입자물리연구소)
    - LHC: ATLAS, CMS, ALICE, LHCb 실험
  - FNAL(미국페르미국립연구소)
    - Tevatron: CDF, D0 실험
  - KEK(일본고에너지물리연구소)
    - KEK B-factory: BELLE실험
  - SLAC(스탠포드선형가속기연구소)
    - BaBar 실험
  - ILC(국제선형가속기 연구)
- 거의 모든 핵/입자실험의 표준

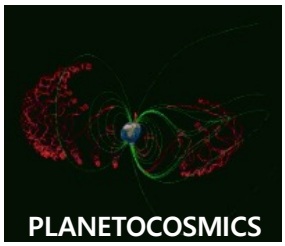


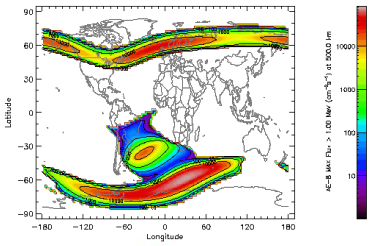


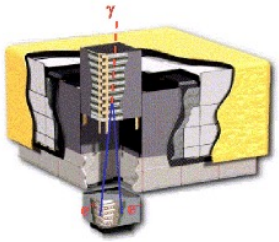


Geant4가 쓰이는 곳 (2/4)

- 우주과학(Space science)
  - SUG: Space Users Group
    - Geant4를 우주연구에 활용하는 연구자모임
    - PLAETOCOSMICS: 수성, 지구, 화성 등의 행성에 부딪히는 우주선(Cosmic ray)의 전자기, 강상호작용을 분석
  - Space Environment Information System (SPENVIS)
    - 우주환경정보시스템
  - GLAST: Gamma Ray Large Area Space Telescope
    - 광역 감마선 망원경

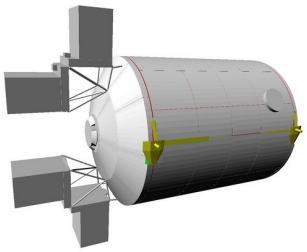


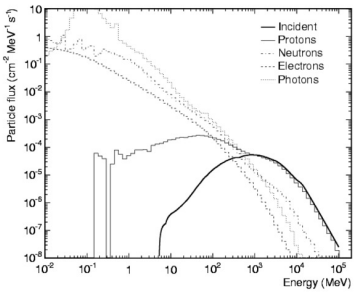


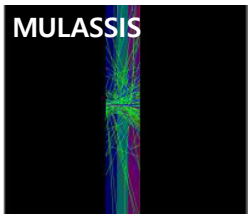


Geant4가 쓰이는 곳 (3/4)

- 방사선기술(Radiation technology)
  - Dose Estimation by Simulation of the ISS Radiation Environment (DESIRE)
    - 콜럼버스ISS내의 방사선피폭량을 계산
  - Physics Models for Biological Effects of Radiation and Shielding
    - 방사선이 생체에 미치는 영향을 조사
  - MUlti-LAyered Shielding Simulation Software (MULASSIS)
    - QinetiQ사의 우주선 방호 차폐 설계

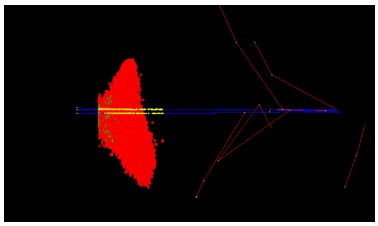




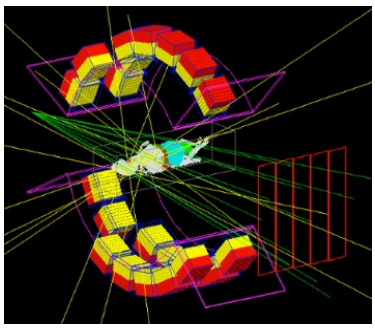


Geant4가 쓰이는 곳 (4/4)

- 의학(Medical science)
  - G4DNA: Geant4 DNA 프로젝트
    - 방사선에 의한 DNA변형을 연구
  - Geant4 의학연구자 집단
    - G4EMU: 유럽 Geant4 의학연구자 모임
    - G4MED: 일본 Geant4 의학물리 연구자모임
    - G4NAMU: 북미 Geant4 의학연구자모임
  - GAMOS: Geant4-based Architecture for Medicine-Oriented Simulations
  - GATE: Geant4 Application for Tomographic Emission



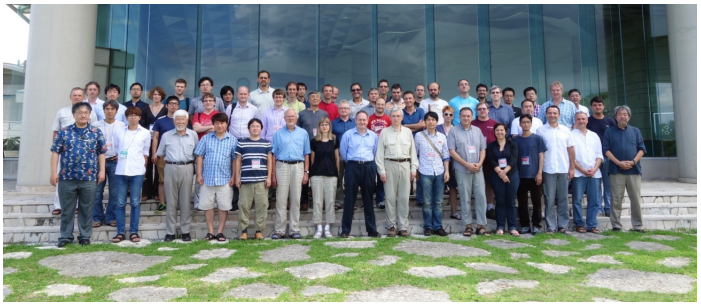
G4DNA project



Geant4 PET Simulation

Geant4 사람들

- Geant4 개발 및 지원: 과학자+SW 엔지니어
  - 연구단 규모 150여명의 개발자 및 회원
    - 한국 KISTI 참여
  - 조직: 16개 Working groups
  - 전문가회의
    - 26th Geant4 Collaboration Meeting
      - Virtual (Sep.13-24, 2021)



20th Anniversary Symposium & 19th Collaboration Meeting,
Okinawa (Japan), 29 September - 4 October 2014.

- Run, Event and Detector Responses
- Tracking
- Particles and Track
- Geometry and Transport
- Generic Processes and Materials
- Electromagnetic Physics
- Low Energy Electromagnetic Physics
- Hadronic Physics
- Persistency
- User and Category Interfaces
- Visualization
- Physics Lists and Validation Tools
- Testing and Quality Assurance
- Software Management
- Documentation Management
- Novice and Extended Examples
- Advanced Examples

Geant4 documentation

- GEANT4의 공식 홈페이지
  - http://geant4.web.cern.ch
- GEANT4를 온라인에서 배울 수 있는 곳
  - http://www.wlap.org
- GEANT4를 기술한 논문
  - Nuclear Instruments and Methods, A 506 (2003) 250-303
  - IEEE Transactions on Nuclear Science, 53 No. 1 (2006) 270-278
  - Nuclear Instruments and Methods, A 835 (2016) 186-225

•[Virtual] 5th LPCC Detector Simulation Workshop, CERN (Geneva), **22-23 November 2021**.
•[Virtual] Geant4 Advanced Course @ CERN, **4-8 October 2021**.

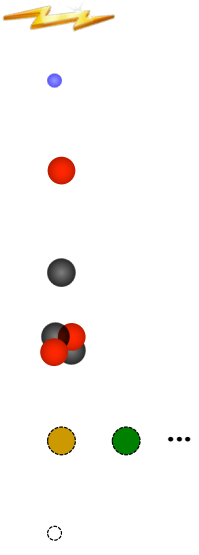
한국에서의 Geant4 교육

- 핵/입자/천체 물리분야, 방사선 산업, 핵의학 분야에 수요가 많으나 전문가는 절대적으로 부족
- 원인
  - 핵/입자 물리학의 기초지식이 필수
    - 선무당이 사람 잡는 결과를 만든다.
  - 상당한 지식의 소프트웨어 엔지니어링 지식이 필수
    - OO 프로그램밍, Linux OS 경험이 필요.
  - 체계적인 Geant4 교육 시스템 및 전문가 부재
    - 우리뿐 아니고 전세계적으로 G4의 전문가가 많지 않은 실정
- 해결책
  - 대학원과정내 Geant4 특강을 개설하고 공동으로 학습
    - 전체 국내 수요를 묶어 한두 군데에서 개설
    - KISTI의 GEANT4 워크숍, 서울시립대 물리학과 특성화 대학원과정
    - KCMS Lectures on Collider Physics

물리학 기초지식: 물질과 상호작용

우리주변의 입자들

- 우리 주변을 돌아다니는 입자는 알고 보면 그다지 많지 않다.
  - 광자
    - 전자기파, X선, 감마선
  - 전자
    - 브라운관 전자총, 베타선
  - 양성자
    - 수소원자의 핵, 양성자 가속기
  - 중성자
    - 핵발전소
  - 알파입자
    - 헬륨의 핵, 핵붕괴
  - 뮤온, 파이온, …
    - 우주선과 대기와의 충돌로 생성
  - 중성미자, …
    - 태양, 원자력 발전소

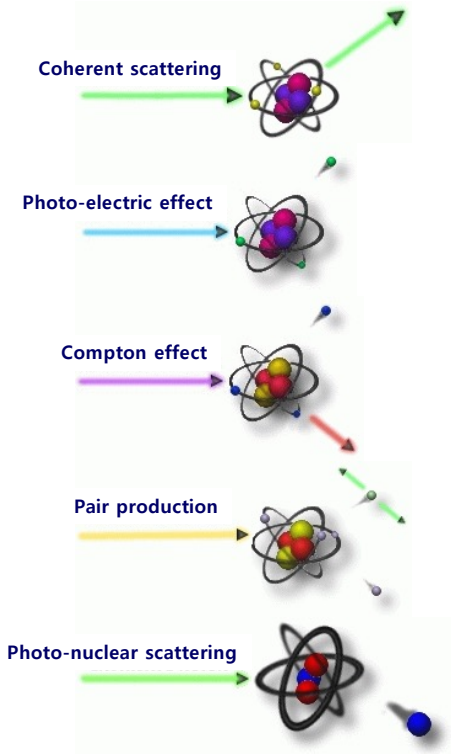


물질과 입자의 상호작용

- 광자 (전자기력)
  - 광전효과
  - 콤프튼산란
  - 쌍생성
- 전자 (전자기력)
  - 이온화
  - 다중산란
  - 제동복사
- 양성자 (전자기력, 핵력)
  - 이온화
  - 핵반응
- 중성자 (핵력)
  - 핵반응

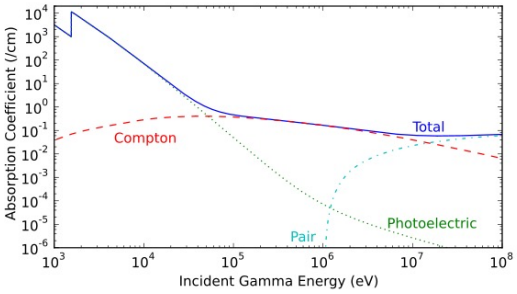
빛과 물질과의 상호작용

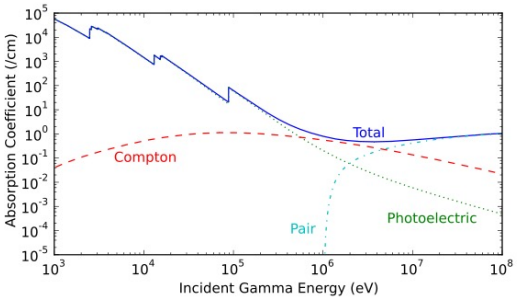
- 전자기파, 가시광선, 자외선의 흡수
  - 주로 원자의 전자구름과 상호작용
- 광전효과 (100keV)
  - 광자가 원자에 흡수되어 전자가 방출
    - $\sigma \approx Z^5/E \ (E_\gamma > m_e c^2)$
    - $\sigma \approx Z^5/E^{7/2} \ (E_\gamma < m_e c^2)$
- 콤프턴효과 (0.1-10MeV)
  - 광자와 전자의 산란
    - $\sigma \approx Z/E$
- 쌍생성 (>1.022MeV)
  - 광자의 에너지가 전자쌍을 만듦
    - $\sigma \propto Z^2$
- 광핵반응
  - 알파, 감마붕괴의 역반응



빛과 물질과의 상호작용

- **Absorption coefficient μ**
  - 위치 x에서의 빛의 세기 $I(x)$
    - $dI = I(x+dx) - I(x) = -\mu\, I(x)dx$
    - $I(x) = I_0 e^{-\mu x}$
  - 반두께: 빛의 세기가 반으로 줄어드는데 까지의 거리
    - $x_{1/2} = \frac{ln2}{\mu} = \frac{0.693}{\mu}$
  - μ는 특성: 물질의 고유성질, 빛의 파장의 함수
    - 핵 및 입자물리가 다루는 에너지는 MeV정도로 감마선을 주로 다룸
    - $\mu_\gamma = \mu_{pe} + \mu_{Comp} + \mu_{pair}$
  - μ는 산란단면적에 비례한다.
    - $\mu = n\sigma$



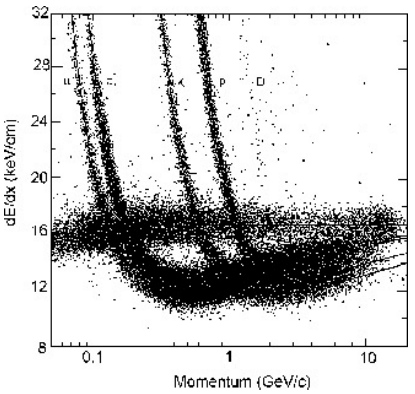


하전입자의 물질과의 상호작용

- 하전입자는 원자의 전자구름을 통해 주로 에너지를 잃게 된다.
  - 원자핵은 산란을 일으켜 방향을 꺾으나 에너지는 소량만 잃는다.
- **Stopping Power**
  - $S(E) = -\frac{dE}{dx} = n\bar{I}$
    - n: 이온화 된 입자수
    - $\bar{I}$: 평균이온화 에너지
- **Bethe-Bloch formula**

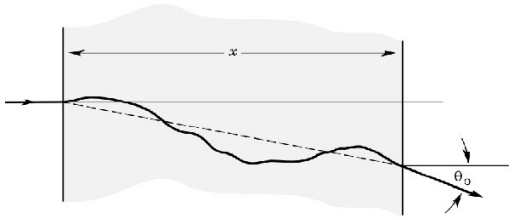
$$-\frac{dE}{dx} = \frac{4\pi}{m_e c^2}\frac{nZ^2}{\beta^2}\left(\frac{e^2}{4\pi\epsilon_0}\right)^2\left[\ln\left(\frac{2m_e c^2\gamma^2\beta^2}{\bar{I}}\right) - \beta^2\right]$$

  - $\gamma\beta \approx 3$일 때 최소값을 갖는다.

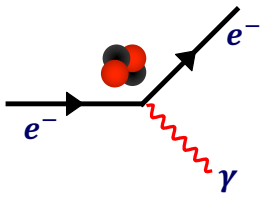


하전입자의 물질과의 상호작용

- **Staggering, Multiple scattering**
  - Bethe-Bloch 공식은 평균값은 잘 맞히나 입자 개별로는 분산이 크다.
    - 알파입자는 분산이 작고, 전자는 크다!
    - 통계적 특성 → Staggeing
  - 여러 번 산란을 걸치는 경우도 분산을 만든다
    - $\theta_{rms} \approx \frac{20MeV}{\beta pc} z \sqrt{\frac{L}{X_0}}$

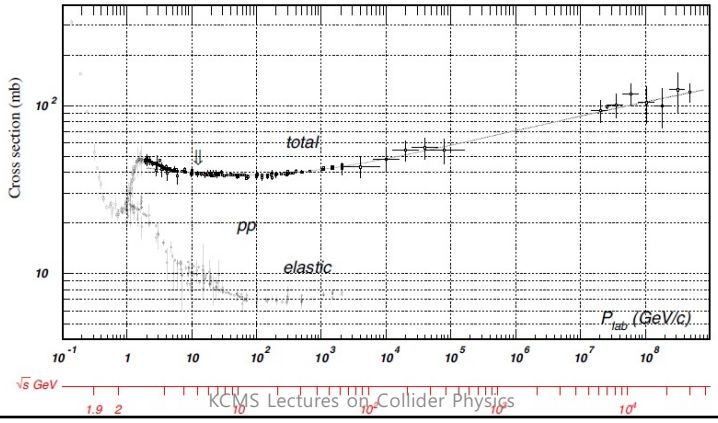


- **Bremsstrahlung (제동복사)**
  - 가벼운 입자는 속도의 변화에 제동복사를 일으킴
    - $(\frac{dE}{dx})_{brem} = -\frac{E}{X_0}, (X_0 \approx 170 \frac{A}{Z^2})$
    - $E = E_0 e^{-x/X_0}$



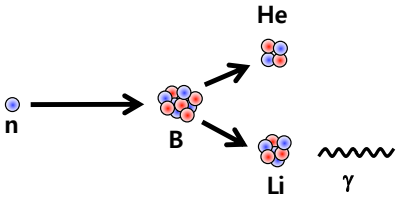
강입자와 물질과의 상호작용

- **강입자 (hadron) : 강한 핵력을 통하여 상호작용하는 입자들**
  - 높은 에너지로 핵들과 충돌하여 다른 강입자들을 다수 만들어 낸다.
    - 중성자, 양성자, 파이온, 케이온
  - 양성자
    - ~ 2GeV : 파이온, 케이온 등을 만들어 냄. 산란단면적의 요동이 심함
    - > 5GeV : 산란단면적이 안정적으로 감소
    - ~70-100GeV: 산란단면적이 최소에 달함. 20-40mb
    - > 100GeV: 로그 형태로 증가



중성자와 물질과의 상호작용

- 중성자는 전기적으로 중성이어서 전자기 상호작용을 하지 않는다.
  - 핵력만 영향을 미칠 수 있다!
- Low energy neutron
  - 중성자 + 핵 → 들뜬 핵
    - 감마선의 방출

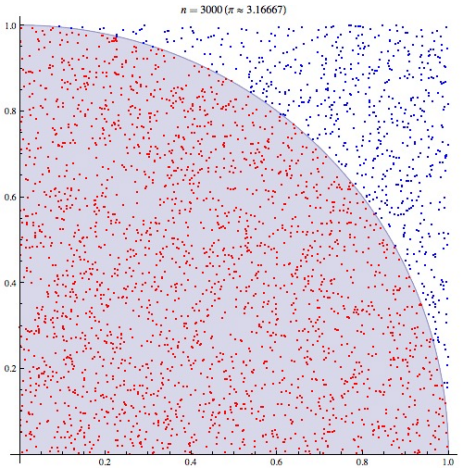


- Medium energy neutron
  - 탄성산란
    - 핵의 전달받는 에너지는 $q^2/2M$ 로, 핵이 무거울 수록 작아진다. (운동량보존)
      - 수소가 풍부한 물질이 좋은 중성자 감속제가 됨 (파라핀)
- High energy neutron (~MeV)
  - 반응하지 않고 통과
    - 가벼운 물질과의 산란으로 검출 (예: 베릴륨, 보론)
      - $^{10}_{5}B + n \rightarrow ^{7}_{3}Li + \alpha$

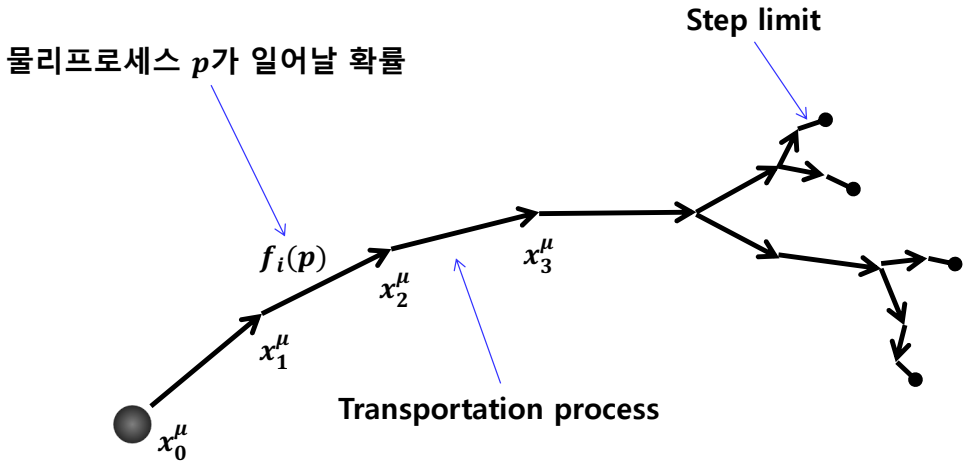
몬테카를로 시뮬레이션

몬테카를로 방법

- 컴퓨터로 파이 값을 구하려면?
  - 전개식을 계산한다.
    - $\pi = 4\times(\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots)$ [Gregory-Leibniz series]
    - $\frac{1}{\pi}=\frac{2\sqrt{2}}{9801}\sum_{k=0}^{\infty}\frac{(4k)!(1103+26390k)}{k!^4(396^{4k})}$ [Ramanujan's series]
  - 몬테카를로 방법을 사용한다.
    - 난수발생을 통한 화살쏘기
      - 사각형의 면적 = 1
      - 원의 면적 = $\pi/4$
      - $\pi = 4\times\frac{n(inside\ circle)}{n(total)}$



Transportation



초기 입자의 위치와 시간

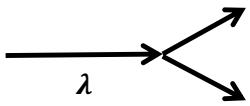
True step length

- Geant4의 작동원리
  - 매질 속 입자들의 평균자유행로(mean free path)을 계산하여, 매번 난수 발생을 통해 입자들을 다음 스텝으로 이동시킨다.

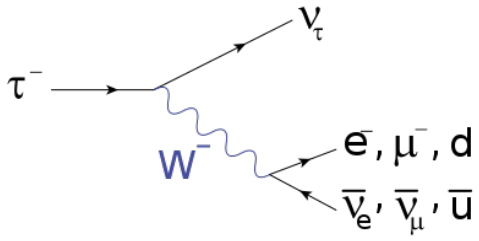
- 평균자유행로의 계산
  - 매질 속 원자의 수: $n = \frac{N_A \rho}{A}$
    - N_A: 아보가드로 수, ρ: 매질의 밀도, A: 원자량
  - 평균자유행로: $\lambda(E) = (\sum_i [n_i \cdot \sigma(Z_i, E)])^{-1}$
    - $\sum_i [n_i \cdot \sigma(Z_i, E)]$: macroscopic cross section
    - $\sigma(Z_i, E)$: total cross section per atom

입자의 붕괴

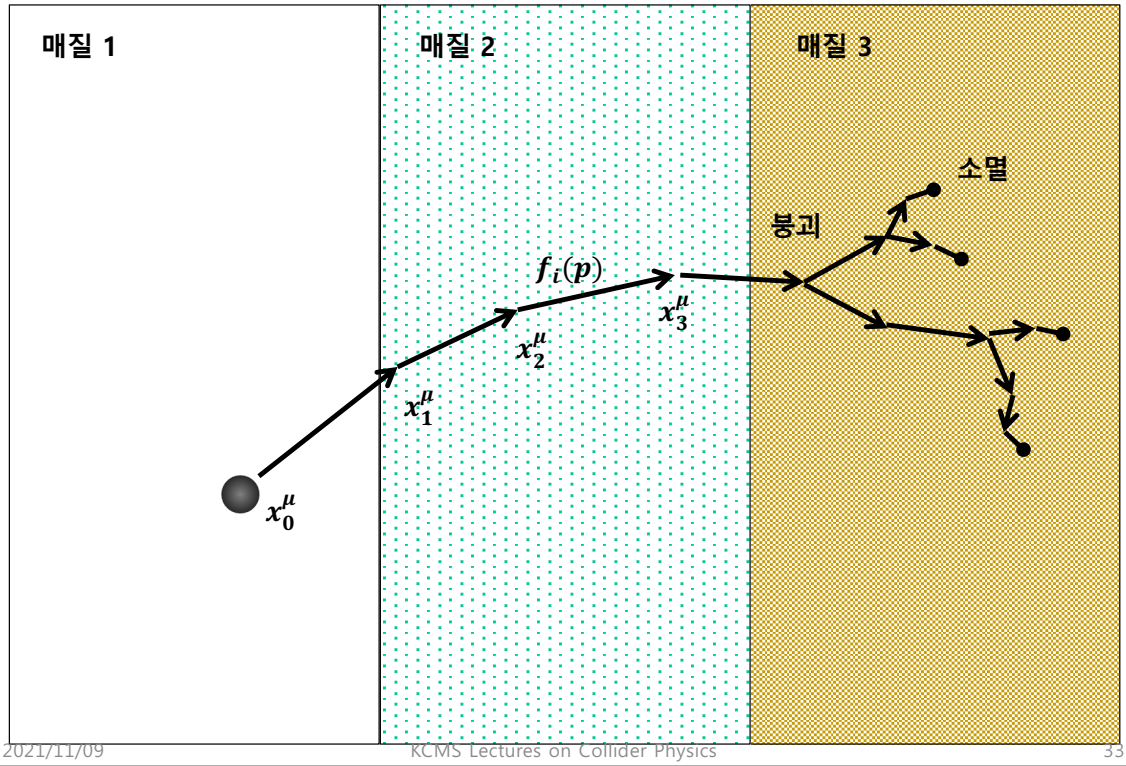
- 평균수명이 τ인 입자의 평균비행거리 λ를 날아가고 붕괴하여 다른 입자로 깨진다.
  - $\lambda = \gamma\beta c\tau$
    - $\gamma = \frac{1}{\sqrt{1-\beta^2}}$, $\beta = \frac{v}{c}$
    - v = particle velocity, c = speed of light, τ = lifetime



- 붕괴율(Branching ratio)
  - 어미 입자가 붕괴하여 각각의 딸 입자로 가는 확률
    - 예) 타우입자의 붕괴
      - $\tau \to e + \bar{\nu}_e + \nu_\tau$: 18%
      - $\tau \to \mu + \bar{\nu}_\mu + \nu_\tau$: 17%
      - $\tau \to \pi + (n)\pi^0 + \nu_\tau$: 45%
      - $\tau \to 3\pi + (n)\pi^0 + \nu_\tau$: 12%



Geant4 안에서 벌어지는 일



Geant4의 물리데이터

- 기본적인 물리 프로세스는 모두 Geant4에 내장되어 있다.
  - 광전효과
    - G4PhotoElectricEffect
  - 콤프턴산란
    - G4ComptonScattering
  - 전자-반전자 쌍생성
    - G4GammaConversion
  - …

Geant4의 물리데이터

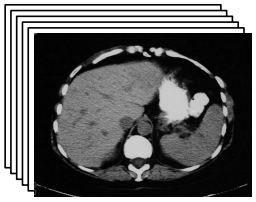
- 특별한 영역에서의 물리데이터는 필요한 경우에만 국한되므로 따로 데이터파일을 분리하여 놓았다.
  - Geant4가 제공하는 특정영역의 물리 데이터 파일
    - G4NDL4.5: thermal neutron cross-sections
    - G4EMLOW6.41: low energy electromagnetic processes
    - G4PhotonEvaporation3.1: photon evaporation
    - G4RadioactiveDecay4.2: radioactive decay hadronic processes
    - G4SAIDDATA1.1: evaluated cross-sections in SAID data-base
    - G4NEUTRONXS1.4: evaluated neutron cross-sections on natural composition of elements
    - G4ABLA3.0: nuclear shell effects in INCL/ABLA hadronic mode
    - G4PII1.3: shell ionization cross-sections
    - RealSurface1.0: measured optical surface reflectance
    - G4ENSDFSTATE1.0: nuclides properties

Geant4 case study

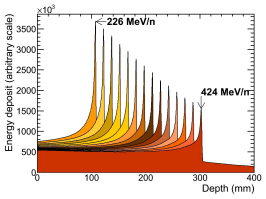
(Univ. of Seoul)

중입자 치료 최적화 선량 계산 프로그램의 제작

DICOM image



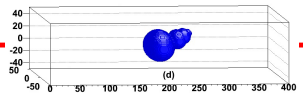
Input Bragg-peak DB



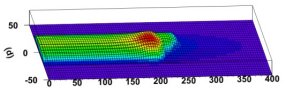
유/무선 네트워크

Marking

3D Tumor shape sending



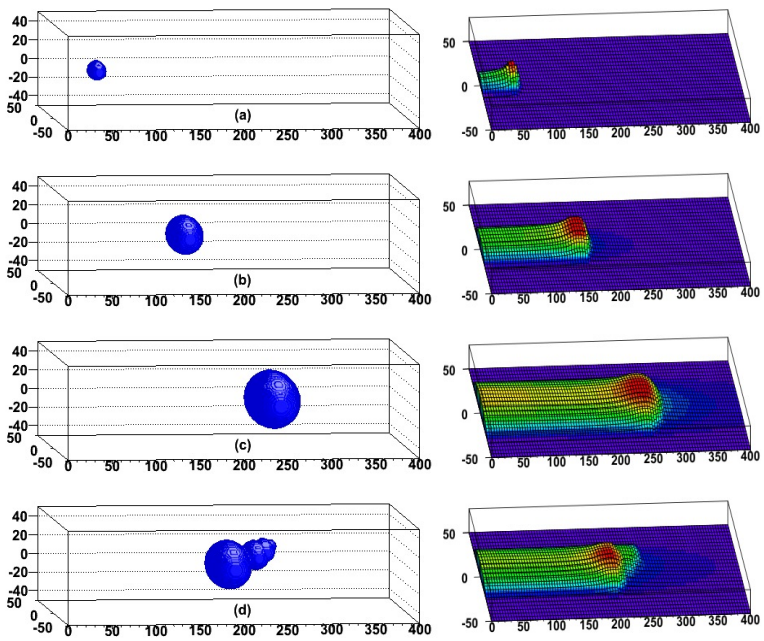
Planning
w(E,x,y)

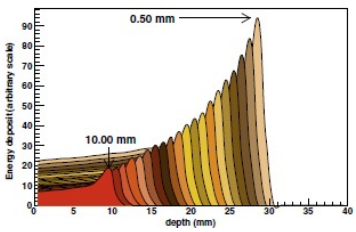


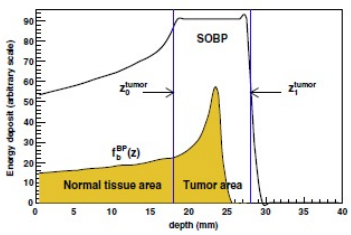
Treatment simulation

- 3D-SOBP MINUIT
- Fast G4 Simulator

중입자 치료률 계산

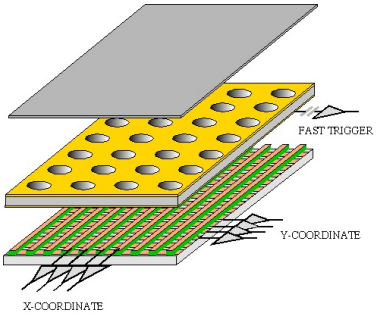


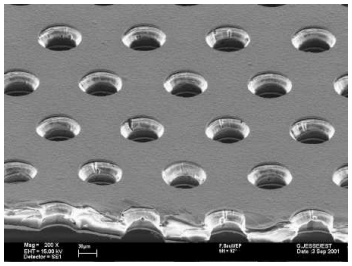


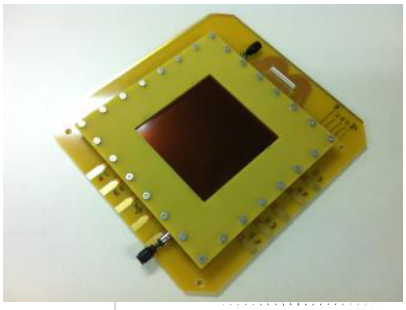


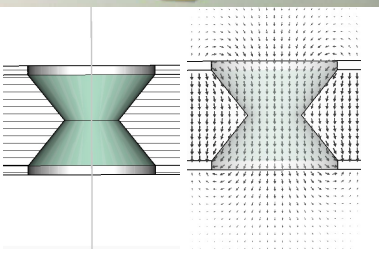
Geant4 시뮬레이션 실전 예

- GEM (Gas Electron Multiplier): 가스전자증배기



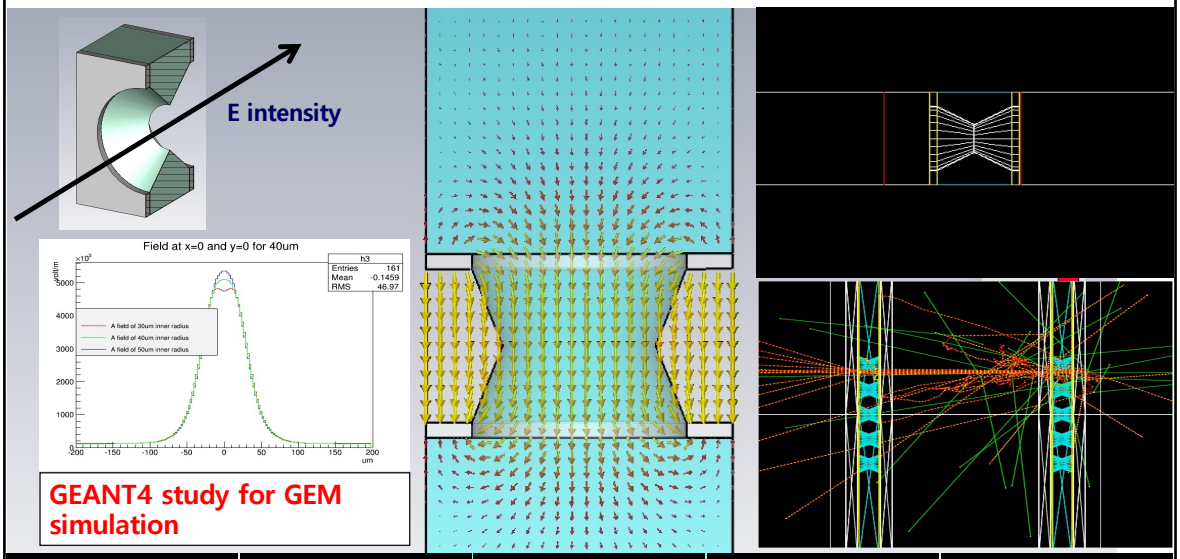


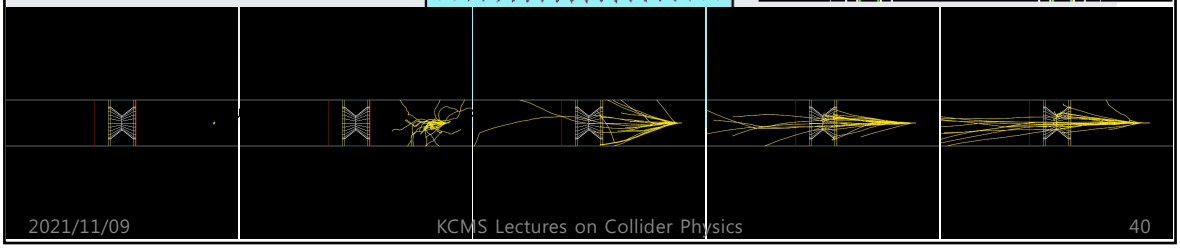




- GEM검출기를 실제작하고 증폭 실험을 통해 구하고 Geant4로 확인
  - Geant4 결과로 석사학위 논문
    - 입자: 전자 (에너지 10eV ~ 100keV)
    - 매질: 진공 + 아르곤 1기압 + 전기장 + GEM
      - http://www.cms-kr.org/twiki/bin/view/Computing/Tier3atUOS/GEMgeant4sim

GEM검출기 시뮬레이션 결과





Geant4를 시작하면 겪게 되는 고충들

- 설치가 안돼요!
  - 윈도우 기반 상용소프트웨어에 익숙한 우리 환경에서 일반적인 문제
    - 대량의 batch job등을 위해서는 linux환경에 익숙해질 필요가 있음
- 실행파일이 어디 있나요?
  - 실행파일은 없다. 라이브러리만 제공하고 실행프로그램은 각자 작성.
    - 예제파일을 컴파일 해서 실행해보고 떠라 하기가 최선.
- 아무 사건도 일어나지 않습니다.
  - 당연합니다. Geant4는 최초에 입자도 없고, 매질도 없습니다.
    - 최소 기본 셋팅은 Geantino와 진공: 반응을 하지 않는 입자와 진공!
- 입자를 넣어도 반응이 일어나지 않습니다.
  - 당연합니다. Physics가 꺼져 있으니까요. Geant4에는 왕도가 없다.
    - Geant4의 철학은 물리공부 없이는 사용할 수 없다.
    - 선무당이 사람잡지 못하게!
- 결론:
  - 물리 + 컴퓨터 프로그램밍을 위한 기초 지식 필수!

결론

- Geant4는 방사선입자와 매질과의 상호작용을 계산해 주는 프로그램
  - 매우 방대한 물리데이터를 포함하고 있다.
- 고에너지물리학, 우주과학, 방사선의학, 방사선 산업관련 Geant4의 수요가 크게 늘고 있다.
  - 반면 국내 전문가가 부족하고, 인력양성을 위한 교육과정도 부재
    - Geant4의 전문가가 되기 위해서는 물리학과 컴퓨터를 동시에 잘해야 함
- 우리나라가 집중할 수 있는 분야: 전문소프트웨어
  - Geant3는 포트란시대의 표준 검출기 시뮬레이터
  - Geant4는 OO형 프로그램으로 LHC실험을 통해 표준이 됨
  - Geant5의 계획이 CERN을 통해 제안되고 있음
    - 단순한 참여를 넘어, 주도적 개발을 할 수 있는 분야

Installation

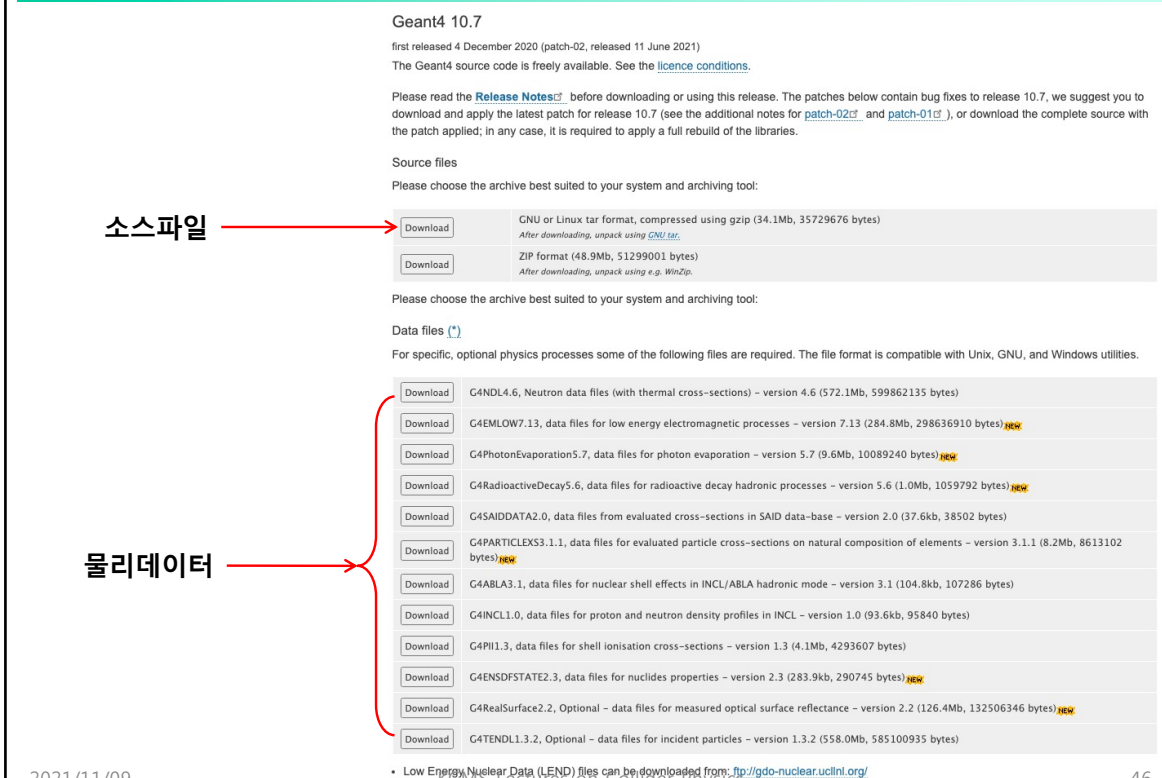
Geant4의 설치: 준비운동

- 본 강좌는 Linux/Mac과 g++을 바탕으로 한다.
  - Windows도 가능
- Linux root 계정으로 설치하면 system wide 설치 가능
  - 개인 머신이면 사용자 계정으로 설치
- 모든 패키지를 독립된 디스크에 몰아서 설치
  - 후일 다른 머신에 network mount할 것에 대비
- 보통 /usr/local, /opt, /cern, /home/geant4 등에 설치
  - 이 강좌에서는 ~/geant4 에 설치
- 우선 GEANT4를 다운로드 받는다.
  - https://geant4.web.cern.ch/support/download

Geant4의 설치: 잠깐만!

- 무작정 다운받고 설치하지 전에… 몇 가지를 확인하자.
- 반드시 있어야 하는 것들
  - Geant4 source: geant4.10.07.p02.tar.gz
  - gcc-4.9.3 (Mac Clang-12.0 Xcode)
  - CMake 3.8
  - CMake
- Optional
  - CLHEP, Expat, zlib
  - GDML XML Geometry
  - User Interface, Visualizations
    - QT, X11, Open Inventor, Motif
- 유용한 소프트웨어들
  - DAWN, WIRED4, OpenScientist, AIDA …

다운로드



Geant4의 설치: 다운로드 & 설치

- 본 강좌에서 다루는 버전은 10.7 (36MB)
- 패키지를 다운로드 하고, 언팩한다.

```
cd ~/geant
wget http://geant4.cern.ch/support/source/geant4.10.07. p02.tar.gz
tar xvfz geant4.10.07.p02.tar.gz
```

- 컴파일할 폴더와 G4를 설치할 폴더를 정한다.

```
mkdir build
mkdir install
cd build
cmake -DCMAKE_INSTALL_PREFIX=~/geant4/install ../geant4.10.07.p02
```

  - 필요에 따라 다시 cmake를 돌릴 수 있다.

```
cmake -DGEANT4_INSTALL_DATA=ON .
```

- 데이터 파일은 나중에 따로 다운받아 언팩해도 된다.

```
cd ~geant4/install/share/Geant4-10.7.2/data
```


cmake

- 이전 방식: make
  - ./configure ; make ; make install
    - 실행 파일 ./configure 가 Makefile을 생성
- 현재 방식: cmake
  - cmake ../source ; make ; make install
    - cmake가 CMakeList.txt를 읽어 Makefile을 생성
      - cmake가 없으면, sudo apt install cmake
- 본 강의록에서의 폴더 설정
  - ~/geant4 : Geant4 폴더
  - ~/geant4/geant4.10.07.p02 : Geant4 소스 폴더
  - ~/geant4/build : Geant4를 컴파일 할 폴더
  - ~/geant4/install : Geant4 라이브러리, 예제 등을 설치 할 폴더

Geant4의 Build 옵션

- CMake 옵션은 –D와 함께 셋팅한다.
  - 옵션을 바꿔 cmake를 실행할 때마다 새로운 Makefile이 만들어진다
    - GDML 지원

      cmake -DCMAKE_INSTALL_PREFIX=~/geant4/install -DGEANT4_USE_GDML=ON ~/geant4/source
  - 표준옵션
    - CMAKE_INSTALL_PREFIX=~/geant4/install (default: /usr/local)
    - GEANT4_INSTALL_DATA=ON (default: OFF)
    - GEANT4_USE_OPENGL_X11=ON (default : OFF)
    - …
  - 전문가옵션: 사실상 건드릴 필요는 없다.

Geant4의 설치: Physics data의 설치

- 몇몇 물리프로세스는 특정 크로스섹션 데이터를 필요로 한다.
  - G4설치시 자동 다운로드 되기도 하나, 후작업으로 가능하다.

    cd ~geant4/install/share/Geant4-10.7.2/data

    wget https://cern.ch/geant4-data/datasets/G4*****.tar.gz

    tar xvfz G4*****.tar.gz
  - 필요한 물리에 따라 데이터들을 설치한다.
    - G4NDL.4.6.tar.gz: thermal cross section을 포함한 neutron데이
    - G4EMLOW.7.13.tar.gz: 저 에너지 전자기 프로세스
    - G4PhotonEvaporation.5.7.tar.gz
    - G4RadioactiveDecay.5.6.tar.gz: 방사성 붕괴 데이터
    - G4PARTICLEXS.3.1.tar.gz
    - G4PII.1.3.tar.gz
    - G4RealSurface.2.2.tar.gz
    - G4SAIDDATA.2.0.tar.gz
    - G4ABLA.3.1.tar.gz
    - G4ENSDFSTATE.2.3.tar.gz

Geant4의 설치: 컴파일

- 컴파일
  - 가급적 코어수를 충분히 잡아 컴파일한다.

    make -jN

    make install
- 에러 메시지가 뜨지 않으면 일단 성공이라 여길 수 있다.
  - install 폴더 안을 보자

    > cd ~/geant4/install

    > ls

    bin include lib share

    > ls bin

    geant4-config geant4.csh geant4.sh

    > ls share/Geant4-10.7.2/

    data examples geant4make tools.license

환경변수 잡아주기

- G4를 실행하기 위해서는 환경변수를 잡아줘야 한다
  - ~/geant4/install/bin 안에 들어있는 geant4.sh에 다 있다.

    source ~/geant4/install/bin/gean4.sh

  - 만약 물리데이터 파일을 특정한 곳에 저장했다면
    - 아래의 환경변수를 통해 물리데이터가 있는 path를 알려줘야 한다.
      - **G4ABLADATA**
      - **G4ENSDFSTATEDATA**
      - **G4INCLDATA**
      - **G4LEDATA**
      - **G4LEVELGAMMADATA**
      - **G4NEUTRONHPDATA**
      - **G4PARTICLEXSDATA**
      - **G4PIIDATA**
      - **G4RADIOACTIVEDATA**
      - **G4REALSURFACEDATA**
      - **G4SAIDXSDATA**

오늘의 숙제

1. 각자 자기 계정에 Geant4가 설치될 폴더를 생성하여라.
   - Geant4 홈페이지에서 G4를 다운로드하고 언팩하자.
   - build와 install 폴더를 생성하고, 컴파일 환경을 체크한다.

2. cmake로 컴파일을 한다.
   - make - jN

3. ~/geant4/install 폴더 안의 내용을 출력해보아라.

4. examples 폴더 안에 어떤 예제들이 들어 있는지 알아보자.

Chapter 1: Geant4 문법

G4의 문법을 알아보자.

- 때론 규정이 맘에 않들면 프로그래밍을 하기 싫을 때도 있다. 허나 어쩌랴... 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.
  - G4UserRunAction, G4VPVParameterisation
- 함수들도 같은 이름규칙을 사용한다.
  - G4RunManager::SetUserAction(), G4LogicalVolume::GetName()

G4문법: 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

- 이들의 정의는 globals.hh에 들어있다.

Geant4와 CLHEP

- GEANT4는 앞의 경우처럼 여러 CLHEP클래스를 typedef해서 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)

G4 단위계 (1/4)

- Geant4에서는 여러 단위계를 사용할 수 있다.

- Geant4에서 물리량은 숫자 뒤에 단위를 곱해 표현한다.
  - millimeter 2*mm
  - nanosecond 2*ns
  - Mega electron Volt 2*MeV
  - Positron charge eplus
  - Degree Kelvin kelvin
  - Amount of substance mole
  - Luminosity intensity candela
  - Radian radian
  - steradian steradian

G4 단위계 (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/G4SystemOfUnits.hh
- 장점 → 단위계를 마음대로 바꾸어 써도 된다.
  - 30*cm == 0.3*m

G4 단위계 (3/4)

- 출력을 할때 단위를 고려해야 한다. 즉 단위로 나누어라! 예를 들면
  - cout << KineticEnergy/KeV << " KeV " << endl;
- G4BestUnit이란 함수를 쓰면, G4가 적당한 단위계로 표현해 준다. (length, time, energy, 등등을 명시해야된다.)
  - cout << G4BestUnit(StepSize, "Length") << endl;
- G4UnitDefinition::PrintUnitsTable을 부르면 G4가 쓰는 모든 단위를 볼 수 있다.

G4 단위계 (4/4)

- 새로운 단위계를 쓰고 싶다면
  - G4SystemOfUnits.hh을 개조할 수 있다 (비추천)

```
#include <CLHEP/Units/SystemOfUnits.h>
static const G4double inch = 2.54*cm;
```

  - G4UnitDefinition클래스를 사용해 새로운 단위계를 만들면 된다.

```
G4UnitDefinition (name, symbol, category, value)
G4UnitDefinition ("inch","in","Length",25.4*mm);
```


3-Vectors (1/2)

- Geant4는 따로 벡터클래스를 만들어 쓰기보다는 CLHEP의 HepVector3D, Hep3Vector를 typedef해서 쓴다. (중복투자는 바보들만 한다!).
- 예를 들면,
  - G4ThreeVector *p=new G4ThreeVector(10,20,100);
- CLHEP에서 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();
- 한가지 … 괴로운 점은 Coding convention이 G4와 맞지 않는다. (물론 function overloading을 하면 되겠지만)

3-Vectors (2/2)

- 벡터를 규격화할 땐,
  - p->unit();
- Y축을 중심으로 2.73라디안을 회전하면,
  - p->rotateY(2.73);
- 또는 어떤 벡터를 중심으로 1.57라디안을 돌리면,
  - p->rotate(1.57,G4ThreeVector(10,20,30));
- 더 자세히 벡터클래스를 공부하려면 클래스설명을 참조하자.
  - source/externals/clhep/include/CLHEP/Vector/ThreeVector.h

Rotation Matrices (1/2)

- 검출기설계에 가장 많이 쓰이는 회전매트릭스 역시 CLHEP에서 가져왔다. G4RotationMatrix = HepRotation
  - #include "G4RotationMatrix.hh"
  - G4RotationMatrix *rm = new G4RotationMatrix;
- x축을 중심으로 45도 돌리면,
  - rm->rotateX(45*deg);
- 어떤 벡터를 중심으로 회전시키려면,
  - rm->rotate(45*deg,Hep3Vector(1.,1.,.3));
- 회전 매트릭스의 Inversion은
  - 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
  - source/externals/clhep/include/CLHEP/Vector/Rotation.h

Chapter 2: Getting Started

예제

- Geant4가 제대로 설치 됐는지를 알 수 있는 가장 좋은 방법은 예제를 실행해보는 것이다.
- Geant4를 가장 빠르게 배우는 법도 예제를 따라하는 것이다.
- 예제는 install/share/Geant4-10.7.2/examples 안에 있다.

```
> cd ~/geant4/install/share/Geant4-10.7.2/examples
> ls
advanced basic extended CMakeLists.txt GNUmakefile History novice README
README.HowToRun README.HowToRunMT
```

  - basic: B1 B2 B3 B4 B5, 5개의 초급용 예제들
  - advanced : 실질적인 실험에 적용할 정도의 예제들
  - WIRED4 JAS Plug-in

디렉터리 구성과 Makefile의 제작

- 작업 폴더를 설정한다.
  - include와 src 서브 디렉터리를 설정

```
> mkdir work ; cd work
> mkdir include
> mkdir src
```

- CMakeLists.txt 파일을 만든다.
  - 예제에서 복사해서 수정해서 쓰는 것이 최고!

```
> cp ~/geant4/install/share/Geant4-10.7.2/examples/basic/B1/CMakeLists.txt .
```

- 작업 폴더 구성은 다음과 같다.

```
> ls
include src CMakeLists.txt
```

CMakeLists.txt의 이해 (1/2)

- CMakeLists.txt
  - 스텝 1: 프로젝트 이름 정하기

```
# (1)
cmake_minimum_required(VERSION 3.8...3.18)
if(${CMAKE_VERSION} VERSION_LESS 3.12)
    cmake_policy(VERSION ${CMAKE_MAJOR_VERSION}.${CMAKE_MINOR_VERSION})
endif()
project(mysim)
```

  - 스텝2: 배치 작업처럼 그래픽이 필요 없을 경우는 OFF

```
# (2) WITH_GEANT4_UIVIS OFF
option(WITH_GEANT4_UIVIS "Build example with Geant4 UI and Vis drivers" ON)
if(WITH_GEANT4_UIVIS)
    find_package(Geant4 REQUIRED ui_all vis_all)
else()
    find_package(Geant4 REQUIRED)
endif()
```

  - 스텝3: 헤더 파일 위치를 알려준다.

```
# (3)
include(${Geant4_USE_FILE})
    include_directories(${PROJECT_SOURCE_DIR}/include)
```

  - 스텝4: 컴파일에 필요한 소스와 헤더 파일들의 목록을 만든다.

```
# (4)
file(GLOB sources ${PROJECT_SOURCE_DIR}/src/*.cc)
file(GLOB headers ${PROJECT_SOURCE_DIR}/include/*.hh)
```


CMakeLists.txt의 이해 (2/2)

- ...
  - 스텝5: 실행파일의 이름과 메인 소스, G4 라이브러리를 링크해준다.

```
# (5)
add_executable(exampleB1 exampleB1.cc ${sources} ${headers})
target_link_libraries(exampleB1 ${Geant4_LIBRARIES})
```

  - 스텝6:

```
# (6)
set(MYSIM_SCRIPTS mysim.in mysim.out init_vis.mac run1.mac run2.mac vis.mac )
foreach(_script ${MYSIM_SCRIPTS})
    configure_file(
        ${PROJECT_SOURCE_DIR}/${_script}
        ${PROJECT_BINARY_DIR}/${_script}
        COPYONLY
    )
endforeach()
```

  - 스텝7: 실행파일의 위치를 정한다

```
# (7)
install(TARGETS mysim DESTINATION bin)
```

main() 함수

- Geant4는 검출기 전산모사용 프로그램 라이브러리로 그 자체가 실행되는 어플리케이션이 아니다!
  - 사용자가 main()을 만들어 실행파일을 만들어 사용해야 한다.
  - 사용자의 메인에 G4RunManagerFactory 클래스만 도입하면 된다.
- 가장 기초적인 main() 함수의 모습

```
> vi mysim.cc
#include "G4RunManagerFactory.hh"
int main()
{
  auto g4man = G4RunManagerFactory::CreateRunManager();
  // user initialization
  ...
  g4man->Initialize();
  g4man->BeamOn(100);
  delete g4man;
  return 0;
}
```


G4의 핵심 G4RunManagerFactory

- G4RunManagerFactory는 Geant4 hierarchy의 가장 밑바탕이 되는 클래스이다.
- RunManager가 하는 일
  - 매니저가 하는 일 1: 검출기를 설정한다.
    - g4man->SetUserInitialization(new MyDetectorComstruction)
  - 매니저가 하는 일 2: 사용될 입자와 물리 프로세스를 설정한다.
    - g4man->SetUserInitialization(new MyPhysicsList)
  - 매니저가 하는 일 3: 초기 입자의 생성과 생성 지점 설정
    - g4man->SetUserInitialization(new MyActionInitialization)
  - 매니저가 하는 일 4: event loop를 돌린다.
    - BeamOn()

반드시 있어야 할 3개의 사용자 클래스

- 현실이든 시뮬레이션이든 반드시 3 가지가 필요하다.
  - 검출기
  - 입자
  - 액션 (입자를 검출기에 어떻게 얼마나 쏠 지)
- Geant4는 이들 세 개의 필수 클래스의 기초 구성을 abstract 클래스로 설정해 놓았다.
  - 사용자는 abstract base 클래스를 상속해 사용자 클래스를 만들면 된다.
    - DetectorConstruction → G4VUserDetectorConstuction
    - PhysicsList → G4VUserPhysicsList
    - ActionInitialization → G4VUserActionInitialization
      - PrimaryGeneratorAction
  - Initialize()와 BeamOn()이 불릴 때 마다 3 개의 User 클래스를 부른다.

DetectorConstruction.hh

- G4VUserDetectorConstruction
  - 검출기의 구성물질, 지오메트리, 센서, 데이터 readout 등을 정의한다.
- MyDetectorConstruction.hh

```
#include "G4VUserDetectorConstruction.hh"
#include "globals.hh"
class G4VPhysicalVolume;
class G4LogicalVolume;
class MyDetectorConstruction : public G4VUserDetectorConstruction
{
    public:
        MyDetectorConstruction();
        virtual ~MyDetectorConstruction();
        virtual G4VPhysicalVolume* Construct();
        G4LogicalVolume* GetScoringVolume() const { return fScoringVolume; }
    protected:
        G4LogicalVolume*  fScoringVolume;
};
```

DetectorConstruction.cc (1/2)

- MyDetectorConstruction.cc

```
G4VPhysicalVolume* MyDetectorConstruction::Construct()
{
  // Get nist material manager
  G4NistManager* nist = G4NistManager::Instance();

  // Lab
  G4Material* labMat = nist->FindOrBuildMaterial("G4_AIR");
  G4Box* labShape = new G4Box("Lab",12*cm,12*cm,24*cm);
  G4LogicalVolume* labLV = new G4LogicalVolume(labShape,labMat,"Lab");
  G4VPhysicalVolume* labPV = new G4PVPlacement(0,G4ThreeVector(),labLV,"la
b",0,false,0, true);
  // (rotation,position,logical volume, volume name, mother volume, boolean, c
opy number, overlap check)

  ...
```


DetectorConstruction.cc (2/2)

- MyDetectorConstruction.cc

```
) ...

  // Detector
  G4Material* detMat = nist->FindOrBuildMaterial("G4_Galactic"); // G4_WATER
  G4Box* detShape = new G4Box("Detector",8*cm,8*cm,16*cm);
  G4LogicalVolume* detLV = new G4LogicalVolume(detShape,detMat,"Detector"
);
  new G4PVPlacement(0,G4ThreeVector(),detLV,"Detector",labLV,false,0,true);

  // Target
  G4Material* tarMat = nist->FindOrBuildMaterial("G4_Al");
  G4Box* tarShape = new G4Box("Target",4*cm,4*cm,0.01*cm);
  G4LogicalVolume* tarLV = new G4LogicalVolume(tarShape,tarMat,"Target");
  new G4PVPlacement(0,G4ThreeVector(0,0,4*cm),tarLV,"Target",detLV,false,0,tr
ue);

  // Set Target as scoring volume
  fScoringVolume = tarLV;
  return labPV;
}
```

PhysicsList.hh

- G4VModularPhysicsList
  - 전산모사에 사용될 입자들의 출석을 부른다.
  - 입자들의 붕괴되고 생성되는 과정(Physics Process)을 등록한다.
  - 각 입자들의 시뮬레이션을 언제 끝내는 지 Cutoff를 설정한다.
- MyPhysicsList.hh

```
#include "G4VModularPhysicsList.hh"
class MyPhysicsList: public G4VModularPhysicsList
{
public:
  MyPhysicsList();
  virtual ~MyPhysicsList();
};
```


PhysicsList.cc

- MyPhysicsList.cc

```
MyPhysicsList::MyPhysicsList() : G4VModularPhysicsList()
{
  //SetVerboseLevel(1);
  RegisterPhysics(new G4EmStandardPhysics());          // EM Physics
  //RegisterPhysics(new G4EmExtraPhysics());           // synchrotron rad.
  RegisterPhysics(new G4DecayPhysics());               // Decays
  //RegisterPhysics(new G4HadronElasticPhysicsXS());   // Hadron Physics
  //RegisterPhysics(new G4StoppingPhysics());
  //RegisterPhysics(new G4IonPhysicsXS());
  //RegisterPhysics(new G4IonElasticPhysics());
  //RegisterPhysics(new G4HadronInelasticQBBC());
  //RegisterPhysics(new G4NeutronTrackingCut());       // Neutron tracking cut
}

MyPhysicsList::~MyPhysicsList(){}
```

가장 중요한 Action: Primary generator

- 필수적인 Action
  - G4VUserPrimaryGeneratorAction (필수)
    - Primary event kinematics
- 필요시에 취할 수 있는 Action
  - 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 UserActionInitialization

- MyActionInitialization.hh

```
#include "G4VUserActionInitialization.hh"
class MyActionInitialization : public G4VUserActionInitialization {
  public:
    MyActionInitialization();
    virtual ~MyActionInitialization();
    virtual void Build() const;
};
```

- MyActionInitialization.cc

```
MyActionInitialization::MyActionInitialization():G4VUserActionInitialization(){}
MyActionInitialization::~MyActionInitialization() {}

void MyActionInitialization::Build() const {
  SetUserAction(new MyPrimaryGeneratorAction);
}
```

가장 단순한 PrimayGeneratorAction.hh

- **MyPrimaryGeneratorAction.hh**

```
class MyPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  public:
    MyPrimaryGeneratorAction();
    virtual ~MyPrimaryGeneratorAction();
    virtual void GeneratePrimaries(G4Event*);

  private:
    G4ParticleGun* fParticleGun;
};
```


가장 단순한 PrimayGeneratorAction.cc

- **MyPrimaryGeneratorAction.cc**

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction():G4VUserPrimaryGeneratorAction()
{
  fParticleGun  = new G4ParticleGun(1);
  G4ParticleDefinition* particle = G4ParticleTable::GetParticleTable()->FindParticle("e-");
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(1.*MeV);
}

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

void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  G4double x0 = 1*cm * (G4UniformRand()-0.5);
  G4double y0 = 1*cm * (G4UniformRand()-0.5);
  G4double z0 = -15*cm;
  fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));
  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

매크로 파일을 이용한 Batch 작업

- Uimanager를 고용한다.

```
int main(int argc, char** argv)
{
  G4RunManager* g4man = new G4RunManager;
  g4man->SetUserInitialization(new MyDetectorConstruction);
  g4man->SetUserInitialization(new MyPhysicsList)
  g4man->SetUserAction(new MyPrimaryGeneratorAction);
  g4man->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 Uimanager가 매크로파일을 불러 들인다.
  // 그 매크로안에 BeamOn이 들어있다.
  delete g4man;
  return 0;
}
```


매크로 파일을 이용한 Batch 작업 (1/2)

- 실행파일에 매크로 파일을 인자로 넣고 돌리면 된다.

```
> ./mysim run.mac
```

- 매크로 파일은 보통 아래와 같이 생겼다.

```
# 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)

- 인터렉티브 모드: 대화형으로 GEANT4를 다룰 수 있다.

```
int main()
{
  G4RunManager* g4man = new G4RunManager();
  ...
  g4man->Initialize();
  G4UIsession* session = new G4UIterminal();
  session->SessionStart();
  delete session;
  delete g4man;
  return 0;
}
```


Interactive 모드 (2/2)

- Interactive모드일 때는 그냥 실행을 시키고,

```
> ./bin/Linux-g++/mydetsim
```

- Initialization이 끝나면 Geant4가 아래의 프롬프트를 준다.

```
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
```

G4UItcsh의 설정

- Interactive모드에서 명령어 리콜 기능을 사용하려면 G4UItcsh을 설정해주면 된다.

```
#include "G4UIterminal.hh"
#include "G4UItcsh.hh"

int main(int argc, char** argv)
{
  ...
  G4UIsession* session = new G4UIterminal(new G4UItcsh);
  session->SessionStart();
  delete session;
  ...


}
```


Chapter 3: Detector Construction

G4VUserDetectorConstruction

- G4VUserDetectorConstruction 는 GEANT4가 제공하는 abstract 클래스로 사용자가 반드시 이를 상속해 사용자클래스를 제공해야만 한다.
- 만들어진 MyDetectorConstruction는 G4RunManager가 SetUserInitialization()을 호출 할 때, 생성된다.
- G4VUserDetectorConstruction의 멤버함수 Construct() 가 RunManager가 Initialize()를 할 때 불려진다.
- 따라서, 사용자는 Construct()을 overload하여 자신의 검출기를 설계해 넣어야 한다.
- 완성된 Construct()는 World Physical Volume의 포인터를 리턴 값으로 돌려준다.

검출기만들기 1단계: 물질(Materials)이란?

- 가상의 검출기를 만들기 위해서는 우선 G4Material이라는 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 입장에서는 G4Material만 알면 된다.
- G4Material 클래스는 원소의 기본 성격 외에, density, state, temperature, pressure, radiation length, mean free path, dE/dx, 등 다양한 성질을 다룬다.

검출기만들기 1단계: 단순한 물질 정의하기

- 단순한 물질은 이름과 원자번호, 원자질량, density만 주면 Material이 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 volume을 만들 때 사용된다.

검출기만들기 1단계: 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 1단계: 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=“Oxygen”,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 1단계: 물질과 원소를 섞기

- Element와 Material을 섞어 Material을 만들어도 된다.

```
...
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 1단계: 온도와 압력이 다른 물질

```
…
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 1단계: 물질과 원소 출력하기

- Materials과 elements은 스탠더드 출력에 바로 출력할 수 있다.

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

검출기만들기 2단계: volume이란?

- 검출기를 구성하는 각각의 입체를 volume이라 부른다.
- 검출기를 설치할 가상의 실험실 공간을 World volume이라 부른다.
- 각각의 volume은 그 모양과 물리적 성질로 나타낼 수 있고, 어느 volume안에 위치하게 된다. (World volume빼고)
- 어느 volume이 모 volume에 놓이게 될 때, 모 volume의 좌표계를 따른다.
- GEANT4에서는 volume이 오버랩 될 수 없다. 반드시 한 볼륨은 다른 어떤 볼륨에 속해 있게 된다.

지오메트리: Volume을 만드는 3단계

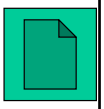
1. 모양과 크기를 갖는 Solid를 구성한다.
2. 위의 Solid에 물리적인 특성(G4Material)을 부여하여 Logical volume을 만든다.
3. 이 Logical volume을 모 volume의 적당한 위치에 삽입하여 Physical volume을 만든다.

검출기만들기 2단계: Solids의 이해

- 3D 입체를 표현하는 기법은 여러 개가 있다.
  - Constructive Solid Geometry (CSG)
  - SWEPT solids
  - Boundary Represented solids (BREPs)

- CSG는 최소한의 파라메터로 입체를 기술하여 간단하고 효율이 좋으나 CAD와 연동이 안 되는 단점이 있다. BREPs는 아주 복잡한 입체도 그릴 수 있고 CAD와도 연동이 되나 효율이 떨어진다.

- CSG solids에는 Box, Tube, Cone, Spheres, Wedges, Torus 등 다양한 모양이 준비되어 있다.



검출기만들기 2단계: 가상실험실 만들기

- 네모난 박스를 가정하여 실험실을 만들자

```
#include “G4Box.hh”
....
G4double lab_x=3.0*m;
G4double lab_y=1.0*m;
G4double lab_z=1.0*m;
...
G4Box* lab_box =
      new G4Box(“Lab box”,lab_x,lab_y,lab_z);
```

검출기만들기 3단계: Logical volume 만들기

- Material과 Solid가 준비되었다면 Logical volume을 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 4단계: Logical volume 집어넣기

- 만들어진 Logical volume을 모 volume에 어디에 위치시키고 어떤 각도로 넣을지를 결정한다.
- Physical volume은 logical volume에 위치정보 더한 instance이다.

```
#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 4단계: World volume 위치하기

- 유일한 예외로 World volume은 최상위 volume이라 어디 놓일 수 없다. 따라서 G4PVPlacement에 null 포인터로 정의되며, rotation도 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 volume은 visualization을 위해 시각화 속성을 셋팅할 수 있다.
  - void G4LogicalVolume::SetVisAttributes(const G4VisAttributes*);
  - void G4LogicalVolume::SetVisAttributes(const G4VisAttributes&);
- 사용자의 가상실험실을 wireframe으로 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);
```

Chap. 4: Particles and physics processes

G4VUserPhysicsList

- G4VUserPhysicsList 클래스를 상속받아 전산모사에 관계된 입자들과 물리프로세스를 설정해 주어야만 한다.
  - 반드시 사용자가 제공해야 할 3가지 코드 중 한 개임을 명심하자.
- G4VUserPhysicsList의 3개의 함수를 오버로딩으로 제공 해야 한다.
  - ConstructParticle() // 입자들을 설정
  - ConstructProcess() // 프로세스를 설정
  - SetCuts() // 전산모사를 마칠 최소 단위

Particles

- Geant4내에 기본적인 입자들은 준비되어 있다.
  - electron, proton, gamma etc.
- 각각의 입자는 G4ParticleDefinition라는 기본 클래스를 상속하여 각각의 클래스로 정의된다.
- G4ParticleDefinition은 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.

G4ParticleTable의 소개

- 한 입자는 singleton으로 static object로 정의된다.
  - G4Electron // 전자를 기술하는 클래스
  - G4Electron::theElectron // 전자
  - G4Electron::ElectronDefinition() // pointer to 전자
- 입자들이 static이므로 프로그램 어디서나 불러 쓸 수 있다.
- 입자의 클래스를 G4ParticleTable에 모아두었다. 이를 이용해 G4내에서 입자들을 꺼낼 수 있다.
  - FindParticle(G4String name) // 이름으로 찾기
  - FindParticle(G4int PDGencoding) // PDG코드로 찾기
- G4ParticleTable 역시 singleton으로 정의된다.
  - G4ParticleTable::GetParticleTable() // pointer to table

Particle construction

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

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


Setting cuts (1/3)

- SetCuts()은 G4VUserPhisicsList의 virtual 멤버함수이다.
- 지오메트리의 설정, 입자들의 등록, 물리프로세스의 설정이 끝난 뒤에 SetCuts()을 불러야 한다.
- SetCuts()을 길이로 설정하면 지오메트리에 설정된 물질 안에서 cut-off에너지로 바뀌어, 입자들이 이 에너지보다 작은 경우가 되면 그 입자에 대한 전산모사를 마친다.
- G4VUserPhysicsList에는 기본 Cut값이 정해져 있다. G4RunManager가 SetCutsWithDefault()를 호출하면, 이 기본값이 SetCuts()의 값으로 사용된다.
- 기본 cut값은 1mm이다.

Setting cuts (2/3)

- 현재의 cut값을 알아보려면, GetLengthCuts()을 호출한다.
- 각 물질에서의 최소 에너지를 얻으려면
  - GetEnergyThreshold(G4Material *)
- 각각의 입자들이 서로 다른 cut값을 사용하게 하려면, 입자들의 상관관계를 잘 따져보아야 한다.
  - gamma → electron → prositron → proton/antiproton → others
- cut을 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=.1*mm;
  SetCutValue(cut,"gamma");
  SetCutValue(cut,"e+");
  SetCutValue(cut,"e-");
  SetCutValue(cut,"proton");
  SetCutValueForOthers(cut);
}
```

Production threshold vs tracking cut

- 각각의 프로세스가 개개의 입자들이 붕괴되어 2차 입자들로 나누어지는 과정을 맡는다.
- 따라서,
  - 각각의 프로세스는 2차 입자를 만드는 데 고유의 한계가 있다.
  - 생성된 입자는 모두 추적된다
  - 각각의 입자들에 cut값을 줄 수 있다. (에너지로 환산된 cut도 가능)
- 각각의 입자에 적용된 SetCut에 따라 2차 입자가 생성되는가가 결정된다.

Production below threshold

- Cut 이하의 threshold이더라도 경우에 따라 2차 입자를 만들도록 하는 경우가 있다.
  - if, by checking the range of the secondary produced against quantities like safety (~the distance to the next boundary), it turns out that the particle, even below threshold, might reach a sensitive part of the detector
  - when mass-to-energy conversion can occur, to conserve the energy. For instance, in gamma conversion, the positron is always produced, even at 0 energy, for further annihilation

Physics Processes (1/2)

- Physics process는 물질과 입자의 상호작용을 말한다.
- Geant4는 크게 7가지 종류로 physics process를 나눈다.
  - electromagnetic
  - hadronic
  - transportation
  - decay
  - optical
  - photolepton_hadron
  - parameterisation

Physics Processes (2/2)

- G4VProcess가 가장 밑바탕이 되는 클래스이다.
- 모든 physics processes는 다음의 3개의 함수로 구성된다.
  - AtRestDoIt()
  - AlongStepDoIt()
  - PostStepDoIt()
- 각각의 함수가 불려질 때 프로세스를 진행할 아래의 클래스들이 사용된다.
  - G4VAtRestProcess — Process with AtRestDoIt
  - G4VContinuousProcess — Process with AlongStepDoIt
  - G4VDiscreteProcess — Process with PostStepDoIt
- 이외에도 조금 더 복잡한 프로세스용 클래스들이 있으나, 여기서는 넘어가자. (예 G4VContinuousDiscreteProcess)

G4ProcessManager

- G4ProcessManager은 G4ParticleDefinition의 데이터멤버다.
- G4ProcessManager는 각 입자들이 겪어야 할 Physics process들의 리스트를 가지고 있고, 어떤 프로세스부터 처리하는가 하는 처리순서를 관장한다.
- 각 입자에 대해 G4ProcessManager의 AddProcess()와 SetProcessOrdering()를 통해 Physics process를 등록해주어야한다.
- AddAtRestProcess(), AddContinuousProcess(), AddDiscreteProcess() 등을 사용하여 입자에 프로세스를 등록할 수 있다.
- ActivateProcess() and InActivateProcess()를 사용하여 프로세스들을 껐다 켰다 할 수 있다.

AddTransportation()

- Geant4에서는 입자의 진행을 프로세스로 본다. (따라서 모든 입자는 Transpotation프로세스를 반드시 가져야만 한다.)
- G4VUserPhysicsList는 AddTransportation()라는 함수를 제공하고 이를 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,iAlongStep);
    }
  }
}
```

ConstructProcess()

- ConstructProcess()는 버츄얼 함수이고 사용자가 프로세스를 만들어 사용하는 입자에 등록해 주어야 한다.
- ConstructParticle()를 통해 선언된 입자들마다 G4ProcessManager 를 가져야 하고, AddProcess(G4VProcess *)를 통해 physics process들을 등록해야 한다.
- 예를 들면, geantino는 오로지 transportation process만 갖는 입자이다.

```
Void ExN01PhysicsList::ConstructProcess()
{
   AddTransportation();
}
```


ConstructProcess() for gammas

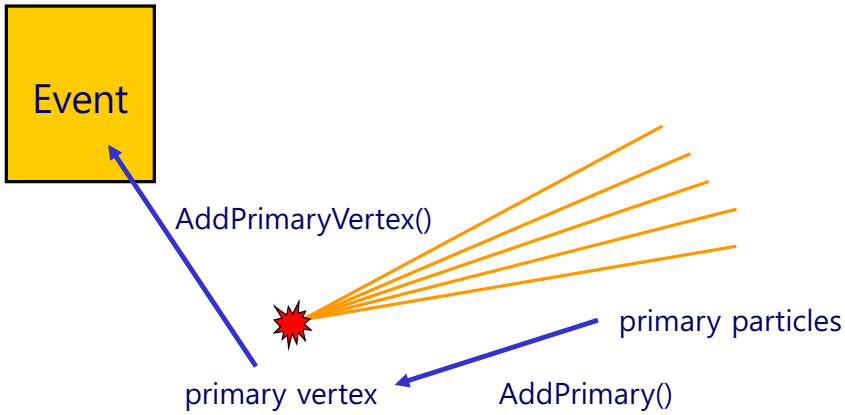
```
Void MyPhysicsList::ConstructProcess()
{
  AddTransportation();
  ConstructEM();
}

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);
}
```

오늘의 숙제

- MyPhysicsList의 ConstructParticle() 안에 geantino, electron, positron, mu+와 mu- 를 등록해 보아라.
- 정의한 입자들에 아래의 Physics processes를 추가하여라.
  - geantinos: transportation
  - electrons: G4MultipleScattering,G4eIonisation, G4eBremsstrahlung
  - positron: G4eplusAnnihilation
  - muons:G4MultipleScattering, G4MuIonisation, G4MuBremsstrahlung, G4MuPairProduction
- SetCuts()에 위 입자들의 적당한 cut을 도입하여라.
- exampleN02를 보고, Particle iterator와 FindParticle()의 사용법을 익혀 원하는 입자를 끌어내 보아라.
- electron을 쏘고, calorimeter에서의 shower develop을 관찰해 보아라.

Chap. 5: Kinematics

Event의 구성



Event의 구성

- Event는 많은 Primary particle들로 구성된다.
  - G4PrimaryParticle(PDGcode,Px,Py,Pz)
  - G4PrimaryParticle(G4ParticleDefinition *,Px,Py,Pz)
- Event에는 Primary vertex가 있다.
  - G4PrimaryVertex(Vx,Vy,Vz,T0)
  - G4PrimaryVertex(G4ThreeVector,T0)
- G4PrimaryVertex의 멤버함수인 AddPrimary를 이용해 primary particle들을 vertex에 등록한다.
  - AddPrimary(G4PrimaryParticle* aParticle)
- G4Event의 AddPrimaryVertex를 통해 Vertex를 등록한다.
  - AddPrimaryVertex(G4PrimaryVertex *aVertex)

G4VUserPrimaryGeneratorAction

- G4VUserPrimaryGeneratorAction는 사용자가 제공해야 하는 마지막 필수적인 함수로 이를 상속해 자신만의 Primary Generator Action을 만들면 된다.
- primary particle들이 어떻게 생성되는가를 이 클래스에 넣는다.
- G4VUserPrimaryGeneratoAction는 GeneratePrimaries()라는 virtual 함수를 가지고 있다. 이 함수는 매 event때 마다 한 번씩 자동으로 불린다. 이 함수의 인자로 event 포인터를 넘겨주므로, event에 primary particle들을 등록 시킬 수 있다.
  - GeneratePrimaries(G4Event* anEvent);

G4VPrimaryGenerator

- G4VPrimaryGeneratorAction::GeneratePrimaries()가 event에 입자들을 채워 넣을 때 다양한 kinematics를 사용할 수 있게 하도록 G4VPrimaryGenerator라는 abstract 클래스를 상속해 사용자만의 Primary Generator를 만들 수 있다.
- 여기서 Primary event를 구성하면 되겠다.
  - GeneratePrimaryEvent(G4Event* an Event)
- 사용자는 PrimaryGeneratorAction에서 구비되어 있는 다양한 PrimaryGenerator들 중 한 개를 골라 쓰면 된다.
- G4VUserPrimaryGeneratoAction는 매 event의 프로세스 때 virtual 멤버 함수인 GeneratePrimaries()를 호출한다. 이때 G4VPrimaryGenerator의 GeneratePrimaryVertex()를 호출하면 된다.

MyPrimaryGeneratorAction.hh의 예

```
#include "G4VUserPrimaryGeneratorAction.hh"

class MyPrimaryGenerator;
class G4Event;

class MyPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  public:
   MyPrimaryGeneratorAction();
  ~MyPrimaryGeneratorAction();
  public:
   void GeneratePrimaries(G4Event* anEvent);
  private:
   MyPrimaryGenerator *TestBeam;
};
```


MyPrimaryGeneratorAction.cc의 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.hh의 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.cc의 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

- G4ParticleGun은 Geant4가 제공하는 아주 기본적인 Primary generator이다.
- G4ParticleGun는 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)
```


오늘의 숙제

- 최고로 간단한 MyDetSim이란 프로그램을 만들려고 한다. 검출기가 존재하지 않는 world volume만 등록한 DectectorConstruction 클래스를 완성하라.
- geantino만 등록하고, Transportation 프로세스를 설정하여라. Cut은 default값을 쓰도록 설정하여라.
- G4ParticleGun에 위에서 등록한 geantino를 걸어서 1GeV의 에너지로 x축에 평행한 방향으로 (-1.0m,0m,0m)에서 테스트 빔을 쏘아라.
- run과 event의 verbose를 설정하여 출력물을 얻어 제출하여라.

Chapter 6: Visualization

시각화 툴의 설치

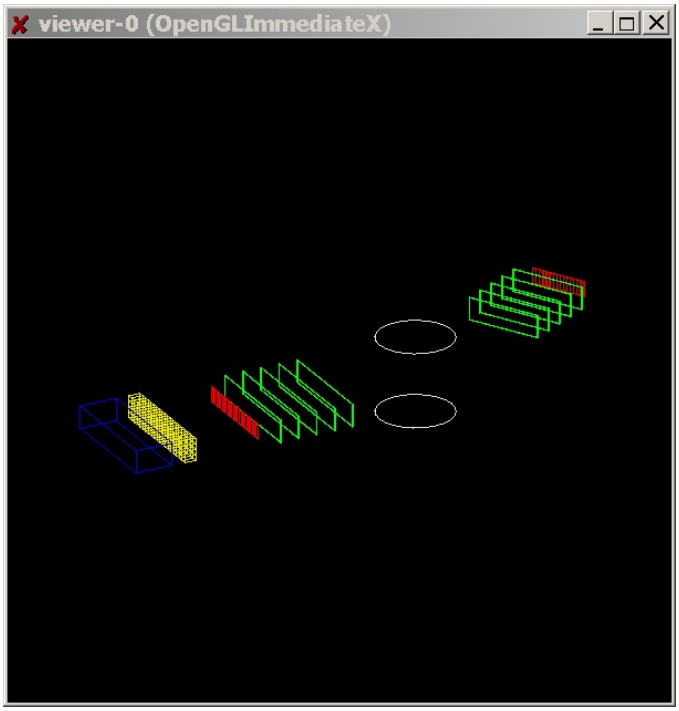
- 설치시에 visualization을 설치하지 않았을 경우는 cmake를 다시 돌려 필요한 visualization 라이브러리를 설치한다.
  - –D 옵션을 사용한다.
    - 예를 들어 OpenGL을 사용하려면

```
cmake –DGEANT4_USE_OPENGL_X11=ON   ~/geant4/source
make -jN
```

- 시각화 관련 옵션
  - Default는 모두 꺼져 있다.
    - GEANT4_USE_INVENTOR (OFF)
    - GEANT4_USE_INVENTOR_QT (OFF)
    - GEANT4_USE_QT (OFF)
    - GEANT4_USE_OPENGL_X11 (OFF) : Unix only
    - GEANT4_USE_RAYTRACER_X11 (OFF) : Unix only
    - GEANT4_USE_XM (OFF) : Unix only
    - GEANT4_USE_OPENGL_WIN32 (OFF) : Windows only

OpenGL

- 가장 간단한 방법이다.
- /vis/open OGLX 또는 OGLXm 등을 사용
- $G4INSTALL/source/visualization의 OpenGL이 설치되어야합
- GLX문제가 Nvidia쪽에 나타나나, X를 사용하고 DISPLAY를 설정하면 잘 작동된다.
- /vis/viewer/set/viewpointThetaPhi 40 40



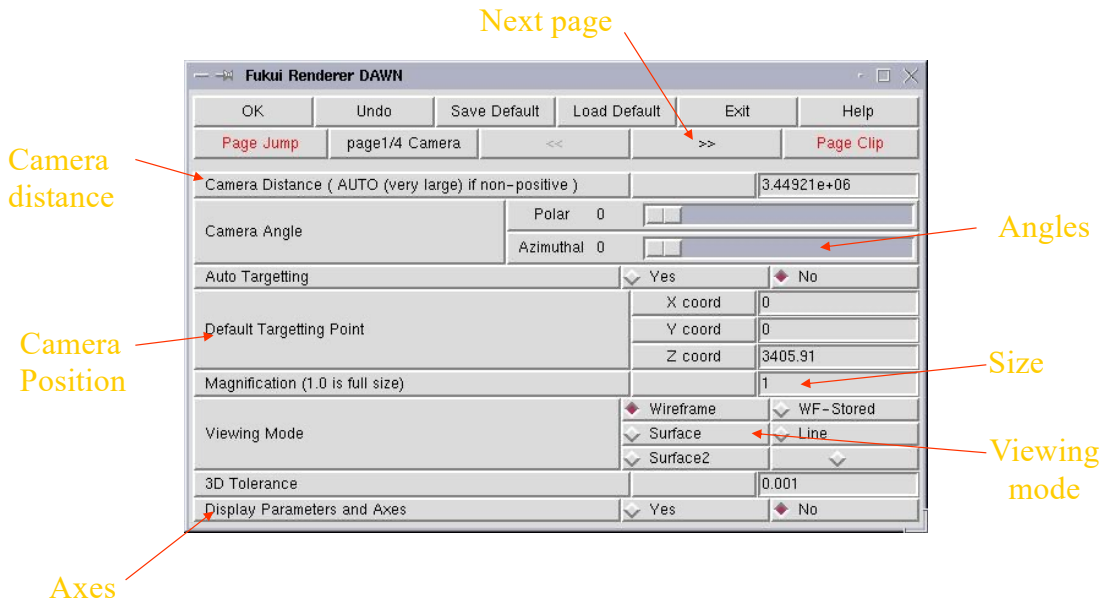
DAWN

- DAWN은 일본의 Fukui University 대학에서 만든 벡터로 기술되는 3D PostScript 전처리기이다.
- http://geant4.kek.jp/~tanaka 에서 3.88a버젼을 받을 수 있다.
  - RedHat9.0에는 3.85e, Fedora rel3은 3.88a 이 맞다.
- 벡터그림을 만들므로 확대해도 아름답다.
- 검출기의 그림을 확인하고, 문서화에 주로 쓰인다.
- Remote visualization, cut view, off-line re-visualization등 다양한 기능을 제공한다.
- visualization 환경변수를 셋팅하면 Geant4실행 시 prim 데이터를 만든다.

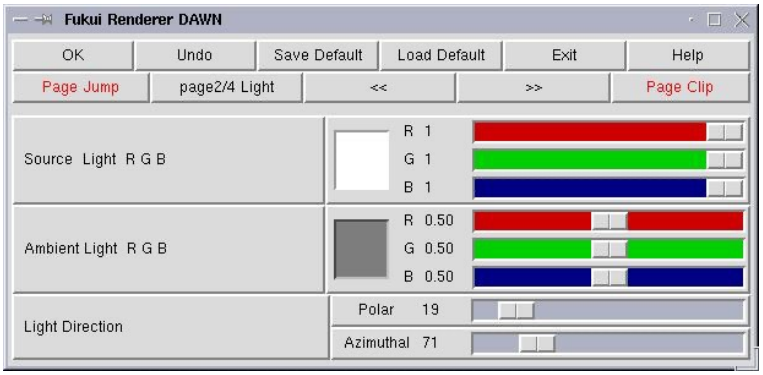
DAWN

- 두 종류의 DAWN driver가 있다.
  - DAWN-File
  - DAWN-Network
- DAWN-File driver는 3D data를 g4.prim 이라는 이름의 파일로 출력한다. Geant4 세션이 끝난 후, DAWN을 시작해 PostScript출력물을 얻을 수 있다.
  - dawn g4.prim
- DAWN-Network driver는 3D data를 TCP/IP socket을 통해 보내고 원격에서 DAWN을 실행해 볼 수 있게 해준다.
- /vis/open DAWNFILE의 명령을 쓴다.

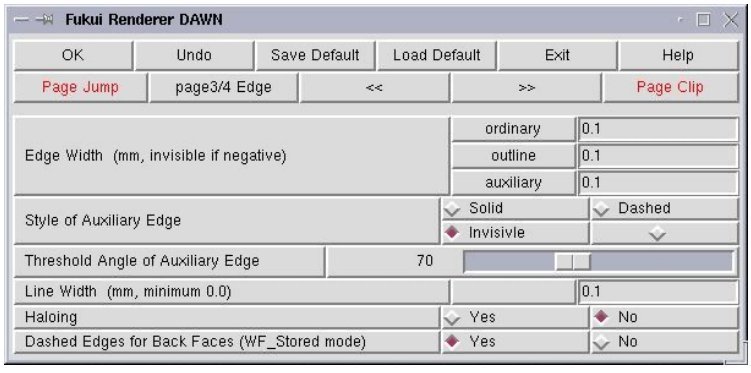
DAWN GUI (page 1)



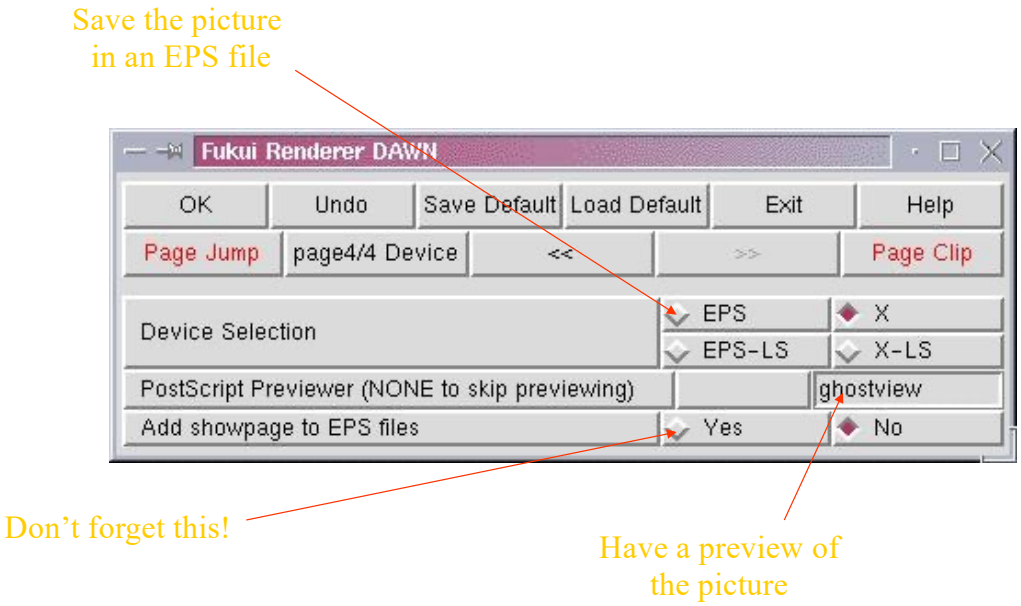
DAWN GUI (page 2)



DAWN GUI (page 3)



DAWN GUI (page 4)

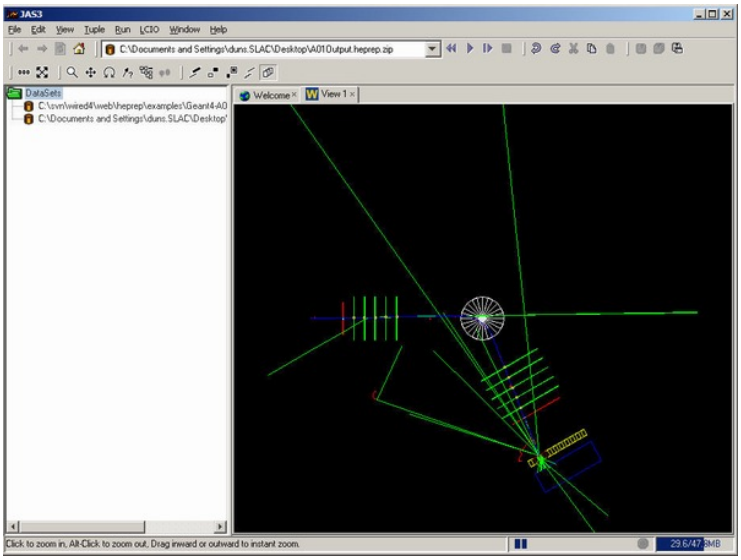


WIRED: Www Interactive Remote Event Display

- JAS 3 를 우선 설치한다
  - http://jas.freehep.org/jas3
- JAS3를 실행하고 View→Plugin Manager를 선택하고 WIRED가 설치되어 있는지 확인
- 없으면 Available탭에서 WIRE4와 BaseLibrary를 선택하고 설치.
- /vis/open HepRepXML을 선택하여 테스트
- So far the best option
- http://wired4.freehep.org를 참조하자

WIRED

- JAS3를 실행하고 WIRE4를 실행
- Open file에서 heprep파일을 선택
- GUI로 되어있어 툴바에 있는 아이콘들에 마우스를 갖다대면 설명이 나온다.



시각화 속성: Visualization Attributes

- G4VisAttributes는 시각화 속성을 저장하는 클래스이다.
- 시각화 속성(Visualization attributes)이란, 시각화 하고자 하는 대상, 즉, 검출기의 volume, 입자의 궤적, 등등이 표현될 때 쓰여질, 색깔, 선의 굵기 등등의 성격을 말한다.
- 각각의 대상 (예를 들면 검출기 volume을 나타내는 G4Box등)에는 크기, 모양들이 규정되는 반면, Visualization attributes 오직 시각화에만 쓰이는 정보만 담고 있다.
- VisAttributes의 생성은 다양한 방법을 통할 수 있다.
  - G4VisAttributes();
  - G4VisAttributes(G4bool visibility);
  - G4VisAttributes(const G4Colour& colour);
  - G4VisAttributes(G4bool visibility,const G4Colour&);

시각화 속성: Visibility (Boolean)

- Visibility속성은 스위치처럼 켰다 껐다 하여 Visualization Manager가 시각화 과정을 처리할 때 그 대상물을 고려할 지 아니면 그냥 넘어갈지를 정한다.
  - G4VisAttributes::SetVisibility(G4bool visibility);
- 예를 들면, SetVisibility(visibility=false)로 설정하면 Visualization Manager가 그대상은 무시하고 다른 대상의 시각화로 넘어간다.
- 참고로 Invisible이란 정적상수가 정의되어 있어, 대상물에서 바로 셋팅할 수 있다.
  - static const G4VisAttributes Invisible
  - object->SetVisAttributes(G4VisAttributes::Invisible);

시각화 속성: Color

- G4VisAttributes는 색깔을 G4Colour의 object로 저장한다.
- G4Colour는 RGBA로 불리는 3개의 색깔(Red, Green, Blue)과 1개의 투명도(Alpha, opacity)를 나타내는 4개의 0부터 1까지의 숫자로 구성된다.
  - G4Color(G4double red=1.0, G4double green=1.0, G4double blue=1.0, G4double alpha=1.0);
- G4VisAttributes (생성자, 멤버함수)
  - G4VisAttributes::G4VisAttributes
  - (const G4Colour& color);
  - void G4VisAttributes::SetColour
  - (G4double red, G4double green,
  - G4double blue,G4double alpha);

```
G4Colour white ();
G4Colour white (1.,1.,1.);
G4Colour gray  (.5,.5,.5);
G4Colour black (0.,0.,0.);
G4Colour red   (1.,0.,0.);
G4Colour green (0.,1.,0.);
G4Colour blue  (0.,0.,1.);
G4Colour cyan  (0.,1.,1.);
G4Colour magenta (1.,0.,1.);
G4Colour yellow (1.,1.,0.);
```

시각화 속성: wireframe , solid styles

- volume들을 그릴 때 선으로만 표현하는 가 (wireframe), 아니면 면을 그리는 가(surfaces)에 따라 여러가지 시각화 속성이 나타난다.
- Wireframe으로그리면 검출기가 투명하게 보이므로, 속안에 들어있는 검출기 내용도 볼 수 있다.
- Surface로그리면 실제와 같이 그림자효과나 조명효과도 나타낼 수 있다.
- Wireframe모드와 Solid모드를 섞어 사용할 수 도 있다.
- Forced wireframe와 forced solid는 아래와 같이 정의한다.
  - void G4VisAttributes::SetForceWireframe(G4bool force);
  - void G4VisAttributes::SetForceSolid(G4bool force);

Chap. 7: Run and Event

G4Run

- Run은 실제 가속기 실험처럼 여러 개의 event를 모은 단위이고, Geant4에서 G4Run 클래스로 표현된다.
- G4RunManager가 BeamOn() 함수를 부를 때 새로운 Run이 생성된다. 한 개의 Run내에서는 검출기의 geometry, Physics 프로세스 등을 바꿀 수 없다.
- G4Run은 다음의 중요한 정보들을 가지고 있다.
  - 고유의 run번호 (실제 Geant4는 이수를 사용하지는 않는다.)
  - run안에 들어 있는 사건의 개수
  - Sensitive Detectors에 설정된 G4VHitsCollection 테이블
  - Digitizers에 설정에 따른 G4VDigiCollection 테이블

G4RunManager (1/2)

- G4RunManager는 Geant4의 모든 시뮬레이션 과정을 콘트롤하는 가장 중요한 클래스다.
- G4RunManager는 singleton 이다. 즉 프로그램 수행 중 오로지 한 개의 Run Manager만 존재한다.
- 사용자의 initialization 클래스들과 사용자의 action 클래스들은 SetUserInitialization() 과 SetUserAction() 의 의해 G4RunManager에 등록되어야 한다.
- Initialize()
  - UserDetectorConstruction을 불러 검출기를 만든다.
  - UserPhysicsList를 불러 사용 할 입자들과 physics processes를 설정한다.
  - cross-section tables을 계산한다.

G4RunManager (2/2)

- BeamOn(G4int NumberOfEvents)
  - 새로운 Run을 만들고, NumberOfEvents만큼 event loop을 돌린다.
- GetRunManager()
  - singleton인 G4RunManager object의 포인터를 찾아준다.
- GetCurrentEvent()
  - 현재 시뮬레이션중인 event의 포인터를 넘겨준다.
- SetNumberOfEventsToBeStored(G4int n)
  - 여러 개의 사건을 모아서 저장할 때 쓴다. BeamOn()이 불려지기 전에 설정을 해야 한다.
- GetPreviousEvent(G4int i_thPrevious)
  - i번째 전 사건의 포인터를 넘겨준다.

G4UserRunAction

- G4UserRunAction는 사용자 action 클래스로, 사용자가 이를 상속해 다음의 두개의 함수를 overload하여 자신만의 run 셋팅을 할 수 있다.
  - BeginOfRunAction()
    - BeamOn()이 불리자 마자 실행되고, 이를 통해 run 고유번호와 run 정보를 담는 히스토그램을 설치하고, 기타 run 에 관련된 셋팅을 한다.
  - EndOfRunAction()
    - BeamOn()의 제일 마지막 단계에서 불린다. 히스토그램을 닫고, run summary파일을 만들고 하는 작업을 넣을 수 있다.

Geant4가 어떤 상태인지를 알라!

- Geant4가 현재 무엇을 하고 있는 상태(state)인지를 아는 것은 매우 중요하다. Geant4의 상태는 G4RunManager가 결정한다.
  - 초기화전단계(PreInit state)
    - RunManager가 초기화를 아직 부르지 않은 상태. 지오메트리나 프로세스, 컷 값들이 바뀌면 이 상태가 된다.
  - 초기화단계 (Init state)
    - G4RunManager의 initialize()가 수행되고 있는 단계.
  - 휴식상태 (Idle state)
    - Geant4가 무언가를 기다리며 작업을 쉬고 있는 단계
  - 빔조사단계 (GeomClosed state)
    - BeamOn() has been invoked
  - 사건진행상태 (EventProc state)
    - GetCurrentEvent(), GetPreviousEvent()만 호출 가능하다.
  - 종료단계 (Quit state)
    - G4RunManager의 destructor가 불려졌을 때.

Run의 취소

- 문제가 발생하거나 원하지 않는 Run을 시뮬레이션 중에 취소하려면 G4RunManager의 **AbortRun()**을 부르면 된다.
- **AbortRun()**은 GeomClosed상태나 EventProc상태에서만 부를 수 있다. (가만 생각해보면 당연한 소리다)
- 어떤 사건을 시뮬레이션 하는 도중 AbortRun()이 불려서 Run이 끝나게 되면, 그 Run의 마지막 event는 corrupted됐었을 확률이 높으므로 분석에 사용하지 말자.

Customizing G4RunManager

- G4RunManager의 모든 멤버 함수는 virtual이므로 사용자가 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 geometry의 변환

- Run과 Run사이에서는 검출기에 작은 변환을 가할 수 있다. (예를 들면 검출기를 조금씩 돌려 놓는다거나, 또는 검출기 특정부분의 두께를 조절한다거나)

- 또는 Run과 Run사이에서 아예 기존의 검출기를 아예 없애고, 새로운 검출기를 놓을 수도 있다. 이 경우에는 World volume을 G4RunManager에 다시 등록 할 필요가 있다.

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

- 두 경우 모두, 지오메트리가 바뀐 사실을 G4RunManager에게 알려야 한다.

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

G4Event

- Geant4에서 한 개의 물리사건은 G4Event클래스로 기술된다.
- G4Event object는 G4RunManager에 의해 생성되고, 생성된 뒤에는 G4EventManager의 관리를 받는다.
- 시뮬레이션 중인 현재의 사건은 G4RunManager의 GetCurrentEvent() 함수를 호출함으로써 얻을 수 있다.
- G4Event는 물리사건의 중요한 4가지 정보를 가지고 있다.
  - Primary vertexes와 primary particles
  - Trajectories (G4TrajectoryContainer)
  - Hits collections (G4HCofThisEvent)
    - Sensitive detector에 의해 생성
  - Digits collections (G4DCofThisEvent)
    - Digitizer에 의해 생성

G4EventManager

- G4EventManager 는 사건을 다루는 매니저 클래스이다.
  - G4PrimaryVertex와 G4PrimaryParticle object들을 현재의 G4Event object와 G4Track object들에 연결시킨다. 그리고, 모든 G4Track object들을 G4StackManager에 보낸다.
  - G4StackManager로 부터 한 개의 트랙object 을 꺼내, G4TrackingManager에게 보낸다. G4TrackingManager가 트랙킹을 시도해보고, 실패하면 “killed”라고 표시한다. (나중에 G4EventManager가 killed라 된 모든 트랙object들을 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
  - G4StackManager가 더 이상 보내줄 것이 없으면, G4EventManager가 현재의 event processing을 끝낸다.

G4UserEventAction

- G4UserEventAction은 사용자 action 클래스로써, 두 개의 virtual 함수를 사용자가 오버로드하여 물리사건을 사건단위로 심층 조절할 수 있게 해준다.
  - BeginOfEventAction()
    - primary particles이 G4Track objects로 등록되기 전에 불린다. Primary particle들을 조절하거나, event-by-event 히스토그램을 셋팅할 때 쓰인다.
  - EndOfEventAction()
    - event 프로세스의 마지막 단계에서 불린다. event-by-event analysis가 필요할 때 불린다.

G4UserSteppingAction

- G4UserSteppingAction는 event의 시작전과 마지막 단계에서의 action이 아닌, 시뮬레이션 한 스텝 한 스텝 단계에 어떠한 action이 필요할 때 쓰인다.
- G4UserSteppingAction을 상속해 사용자만의 SteppingAction을 만들고, 이를 RunManager에 등록한다.
  - void UserSteppingAction(const G4Step*)
- 이 사용자 SteppingAction 함수는 트랭킹 과정 중 매 스텝마다 불린다. 각 스텝에 있어서 필요한 정보는 G4Step포인터를 통해 얻을 수 있다.

Chap. 8: Tracks & Hits

G4Step (1/2)

- G4Step이란 사건을 작은 시간단위로 쪼갠 한 순간이라 보면 된다. G4Step은 다음의 정보들을 갖는다.
  - pointers to PreStep and PostStepPoint
  - Geometrical step length
  - True step length (MS을 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

G4Step (2/2)

- G4StepPoint (PreStep and PostStepPoint)의 정보들
  - (x,y,z,t)
  - (px,py,pz,Ek)
  - 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

- Hit이란 검출기의 센서영역에 트랙이 통과할 때 남기는 자취의 최소단위이다.
- G4에서는 G4VHit이란 베이스클래스를 제공하며, 사용자가 이를 상속해 자기의 실험에 맞게 MyHit을 정의해 쓰면 된다.
- G4VHit의 Draw()와 Print()라는 버츄얼 함수를 오버로드해 사용자에 맞게 설정한다.
- event내에서는 G4VHitsCollection을 통해 Hit들을 저장한다.
- G4THitsCollection는 G4VHitsCollection를 바탕으로 한 template 클래스로, 사용자의 MyHit을 담을 수 있다.

MyHit의 예

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


G4VSensitiveDetector (1/2)

- G4VSensitiveDetector 는 실제 검출을 하는 검출기 부분을 기술하는 베이스 클래스이다.
- G4VSensitiveDetector는 G4Step의 지시에 따라 Hit들을 만들다. → ProcessHits()
- 하나의 sensitive detector class는 반드시 고유명을 가져야 한다. 다수의 sensitive검출기를 갖는 경우에 대비해, 고유명들이 디렉토리 구조를 가지도록 설계하는 것이 좋겠다.
  - myEMcal = new myEMcal("/myDet/myCal/myEMcal");
- 이 sensitive detector의 포인터를 대응되는 G4LogicalVolume object에 셋팅해주고, G4SDManager에 등록을 해야 한다.
  - scin_log->SetSensitiveDetector(myEMcal);

G4VSensitiveDetector (2/2)

- G4VSensitiveDetector는 세 개의 중요한 virtual함수가 있다.
  - Initialize()
    - 매 사건의 시작 때 불리어 지면, 함수 인자로써 G4HCofThisEvent 를 받는다. 이 Hit Collection에 생성되는 Hit들을 담는다.
  - ProcessHits()
    - sensitive detector를 포인팅하는 G4LogicalVolume에 Step이 벌어지면 G4SteppingManager가 이 함수를 부른다. 이때 G4Step object과 ReadOut geometry의 G4TouchableHistory object가 인자로써 쓰인다.
  - EndOfEvent()
    - 매 사건의 마지막에 불리어, 각각의 Hit collection을 G4HCofThisEvent 에 등록시킨다.

G4SDManager

- Sensitive검출기에 관한 일은 G4SDManager가 한다.
- SDM은 singleton으로 static함수를 불러 얻을 수 있다.
  - G4SDManager::GetSDMpointer()
- 모든 Sensitive Detector들을 반드시 G4SDManager에 등록시켜야 제대로 작동한다.
  - G4SDManager *sdman=G4SDManager::GetSDMpointer()
  - sdman->AddSensitiveDetector (this);
- SDM이 Hit collection ID를 넘겨준다.
  - sdman->GetCollectionID("My Collection");

Hits Collections

- Hit collection은 SD 생성 때 고유이름을 주어 만들 수 있다.
  - collectionName.insert("CalorimeterCollection");
- 또는 SD의 초기화 때 만들어도 된다.
  - caloHitsCollection=new CalorimeterHitsCollection
  - (SensitiveDetectorName,collectionName[0]);
- Hit이 생성될 때 마다 collection에 넣으면 된다.
  - int icell=caloHitsCollection->insert(caloHit);
- 각각의 Hit collection은 나중에 event안에 등록시켜줘야 한다. either in SD's Initialize() or SD's EnfOfEvent()
  - 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 Manager를 통해 Hit Collection을 access한다.
  - 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));