The Telescope Array Fluorescence Detector Simulation on GPUs

Tareq AbuZayyad
University of Utah

CHEP 2013
Amsterdam, The Netherlands
10/14/2013

Overview

- Introduction to the Telescope Array (TA) Experiment.
- Fluorescence Detector (FD) Response Simulation:
  - Overview: We simulate the development of an extensive air shower in the atmosphere. The light production and propagation to the detector, and the collection and processing of this light by the detector to produce an "event".
- GPU Based Simulation:
  - How the simulation is done using CUDA. Design goals, code implementation and kernel launch configuration.
- GPU Performance Analysis:
  - Results on accuracy and speedup are shown for a few test cases. **More details if time allows**. See backup slides otherwise.
- Summary and Outlook.

The Telescope Array Experiment

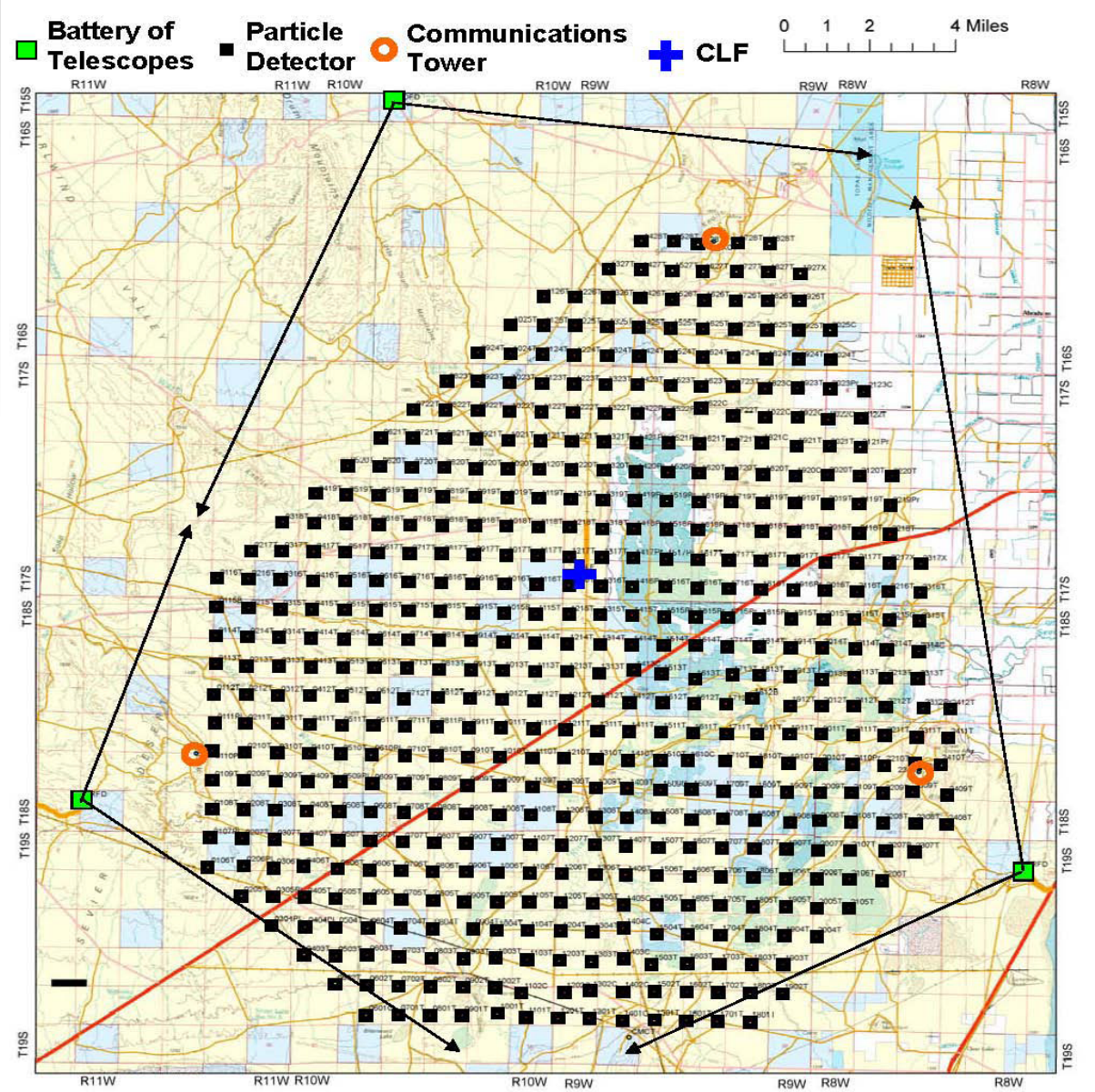
The TA Collaboration

T Abu-Zayyad[1], R Aida[2], M Allen[1], R Azuma[3], E Barcikowski[1], JW Belz[1], T Benno[4], DR Bergman[1], SA Blake[1], O Brusova[1], R Cady[1], BG Cheon[6], J Chiba[7], M Chikawa[4], EJ Cho[6], LS Cho[8], WR Cho[8], F Cohen[9], K Doura[4], C Ebeling[1], H Fujii[10], T Fujii[11], T Fukuda[3], M Fukushima[9,22], D Gorbunov[12], W Hanlon[1], K Hayashi[3], Y Hayashi[11], N Hayashida[9], K Hibino[13], K Hiyama[9], K Honda[2], G Hughes[5], T Iguchi[3], D Ikeda[9], K Ikuta[2], SJJ Innemee[5], N Inoue[14], T Ishii[2], R Ishimori[3], D Ivanov[5], S Iwamoto[2], CCH Jui[1], K Kadota[15], F Kakimoto[3], O Kalashev[12], T Kanbe[2], H Kang[16], K Kasahara[17], H Kawai[18], S Kawakami[11], S Kawana[14], E Kido[9], BG Kim[19], HB Kim[6], JH Kim[6], JH Kim[20], A Kitsugi[9], K Kobayashi[7], H Koers[21], Y Kondo[9], V Kuzmin[12], YJ Kwon[8], JH Lim[16], SI Lim[19], S Machida[3], K Martens[22], J Martineau[1], T Matsuda[10], T Matsuyama[11], JN Matthews[1], M Minamino[11], K Miyata[7], H Miyauchi[11], Y Murano[3], T Nakamura[23], SW Nam[19], T Nonaka[9], S Ogio[11], M Ohnishi[9], H Ohoka[9], T Okuda[11], A Oshima[11], S Ozawa[17], IH Park[19], D Rodriguez[1], SY Roh[20], G Rubtsov[12], D Ryu[20], H Sagawa[9], N Sakurai[9], LM Scott[5], PD Shah[1], T Shibata[9], H Shimodaira[9], BK Shin[6], JD Smith[1], P Sokolsky[1], TJ Sonley[1], RW Springer[1], BT Stokes[5], SR Stratton[5], S Suzuki[10], Y Takahashi[9], M Takeda[9], A Taketa[9], M Takita[9], Y Tameda[3], H Tanaka[11], K Tanaka[24], M Tanaka[10], JR Thomas[1], SB Thomas[1], GB Thomson[1], P Tinyakov[12,21], I Tkachev[12], H Tokuno[9], T Tomida[2], R Torii[9], S Troitsky[12], Y Tsunesada[3], Y Tsuyuguchi[2], Y Uchihori[25], S Udo[13], H Ukai[2], B Van Klaveren[1], Y Wada[14], M Wood[1], T Yamakawa[9], Y Yamakawa[9], H Yamaoka[10], J Yang[19], S Yoshida[18], H Yoshii[26], Z Zundel[1]

[1]University of Utah, [2]University of Yamanashi, [3]Tokyo Institute of Technology, [4]Kinki University, [5]Rutgers University, [6]Hanyang University, [7]Tokyo University of Science, [8]Yonsei University, [9]Institute for Cosmic Ray Research, University of Tokyo, [10]Institute of Particle and Nuclear Studies, KEK, [11]Osaka City University, [12]Institute for Nuclear Research of the Russian Academy of Sciences, [13]Kanagawa University, [14]Saitama University, [15]Tokyo City University, [16]Pusan National University, [17]Waseda University, [18]Chiba University [19]Ewha Womans University, [20]Chungnam National University, [21]University Libre de Bruxelles, [22]University of Tokyo, [23]Kochi University, [24]Hiroshima City University, [25]National Institute of Radiological Science, Japan, [26]Ehime University

The TA Detectors

- TA is located in Millard County, Utah, ~200 km southwest of Salt Lake City.
- Surface Detector: 507 scintillation counters 1.2 km spacing. (***run 24/7***)
- Three Fluorescence Detectors overlooking SD (***run only during moonless nights***):
  - Middle Drum (MD)
  - Black Rock (BR)
  - Long Ridge (LR)



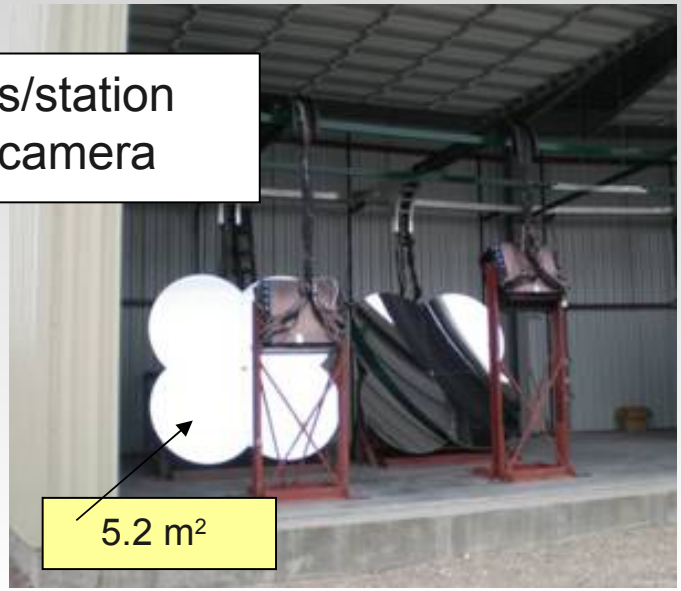
TA Fluorescence Detectors

Refurbished from HiRes

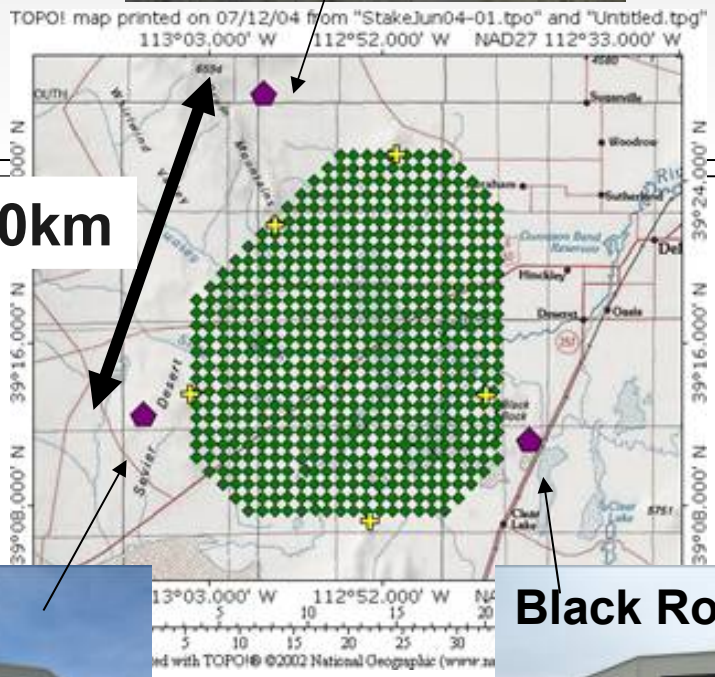
Observation started Dec. 2007

Middle Drum

14 cameras/station
256 PMTs/camera



5.2 m²



~30km

Observation started Nov. 2007

Long Ridge

Observation started Jun. 2007

Black Rock Mesa

New FDs

256 PMTs/camera
HAMAMATSU R9508
FOV~15x18deg
12 cameras/station

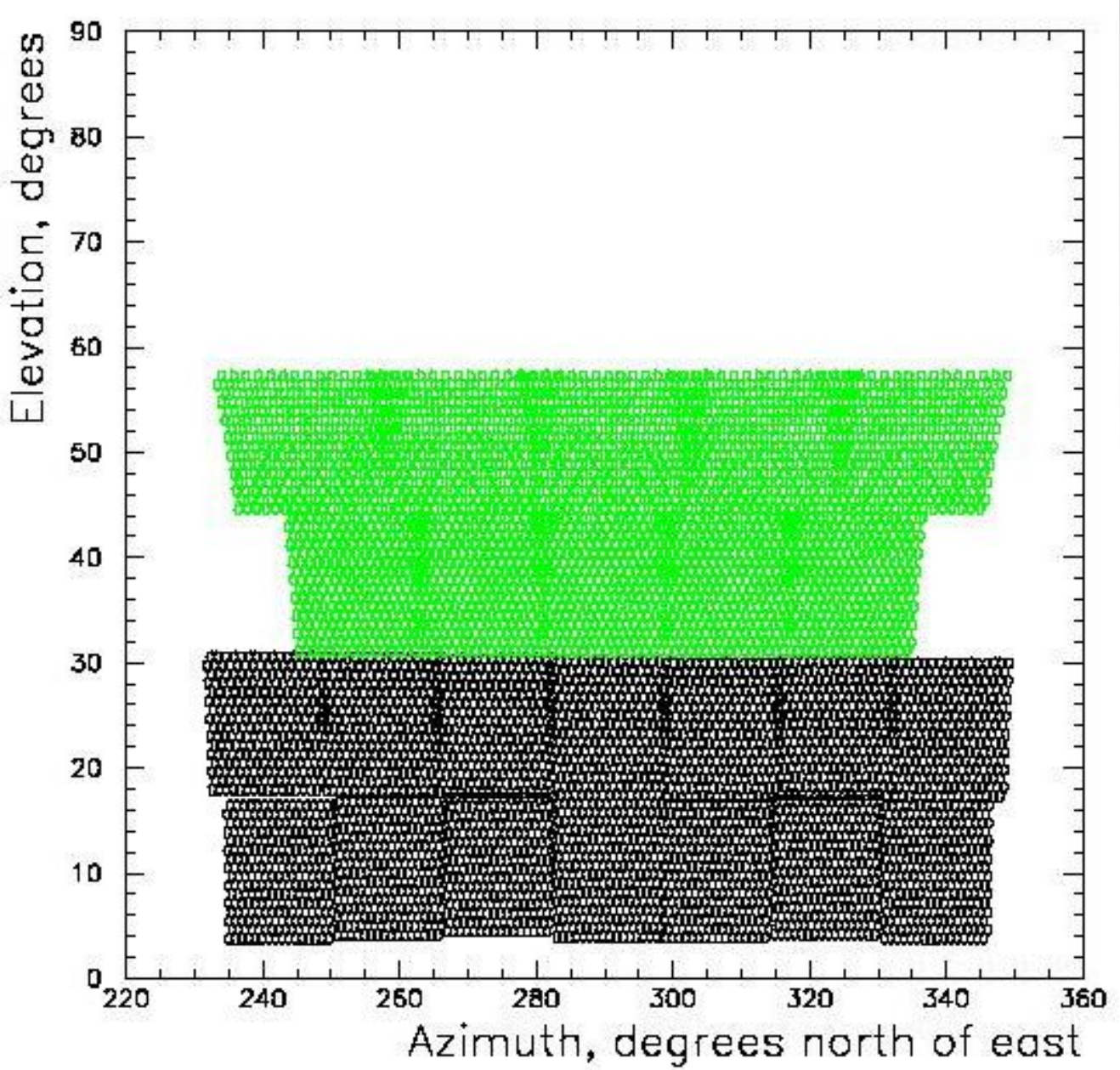
~1 m²

6.8 m²

Middle Drum TA/TALE FD Observatory Site (14 + 10 Telescopes)

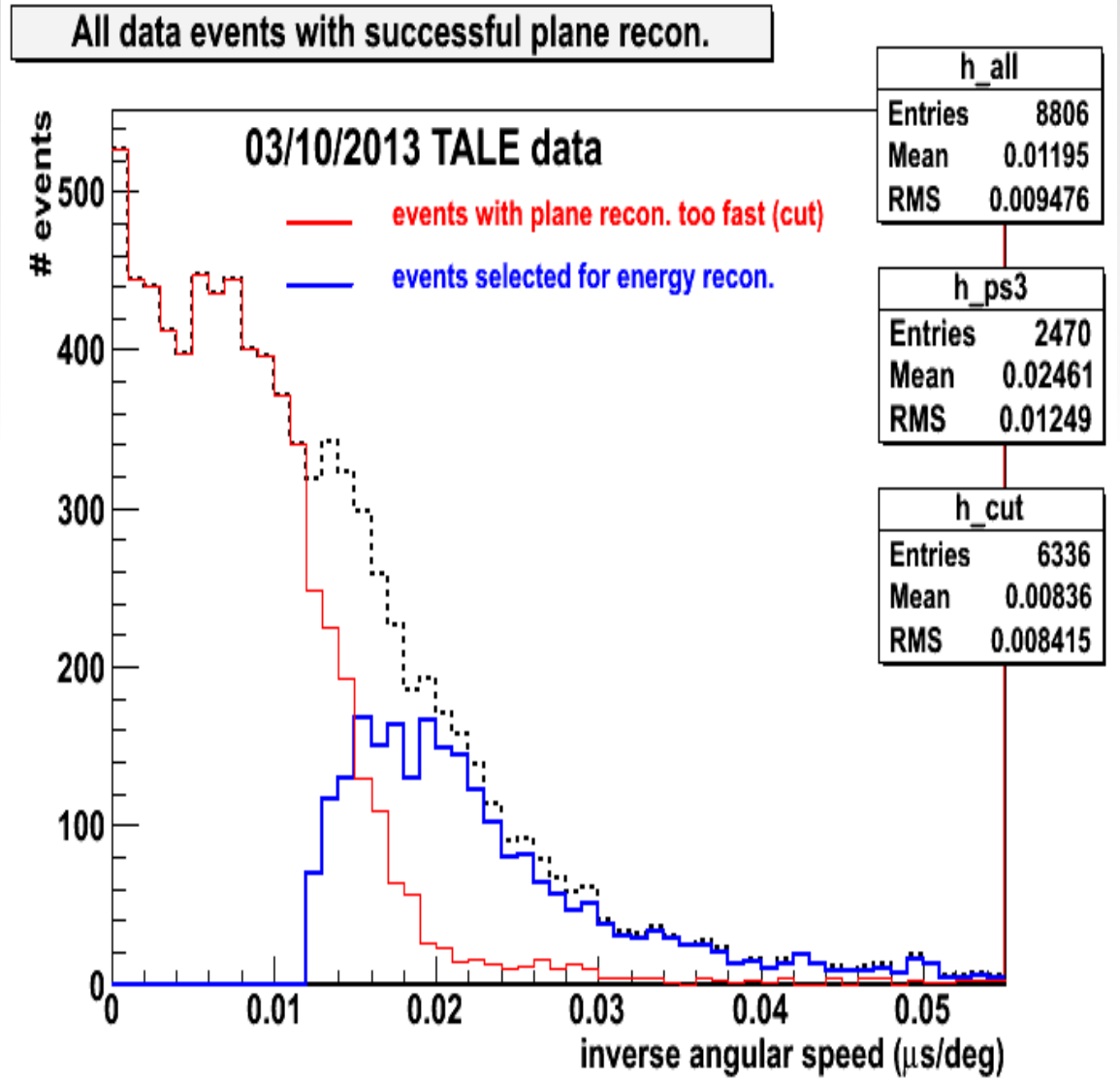
Middle Drum TA/TALE Viewing Range

- TAMD + TALE
- 14 lower telescopes make up TA (Middle Drum) Detector.
- 10 higher telescope (new addition) make up the TA-Low Energy extension Detector.



TALE Cerenkov events data rate

- Vast majority of triggered events are Cerenkov (lower energy).
- Still dominant, even after removing "Cerenkov Blasts"
- Fluorescence events make up the ***distribution tail***.
- Many more events to process than initially thought.



Fluorescence Detector Response Simulation

Detector Response Simulation

- Initialize simulation: given user input.
- Generate shower geometry / profile.
- Generate shower photons (Fluorescence / Ckov)
- Calculate Atmospheric Attenuation, account for detector collection efficiency (including Q.E.)
- Run electronics simulation: signal filtering and test trigger condition for individual PMT's. Follow with detector trigger logic simulation.
- ***Details can be found under back up slides***.

GPU Based Simulation

Opportunities for Parallelism

- Simulate a large number O(10^6) of independent events.
    - 1 GB of GPU card memory can accommodate ~500 showers simultaneously. (when using floats)
- A shower is treated as an extended light source.
    - divided into ~1000 point sources.
    - 32 wavelength bins.
- Ray-trace collected photons independently.
- 256 PMT's (pixels) per telescope; multiple telescopes per event.
- Signal processing: ~10000 time bins per pixel

Design Goals and Constraints

- CUDA Capable GPU may or may not be available on a given system, so CUDA support has to be *optional*.
- To insure the program produces the same results with or without acceleration, *as much code as possible* is written to run on either: using __host__ __device__ qualifier.
    - This is needed most if/when someone updates the code in the future; … chances are they will update one path and not the other! (speaking from experience)
- To get best performance on GeForce GPU's we also require that *as much as code as possible* is written using C++ templates to support both single and double precision calculations.

Basic Code structure

- Can compile on a machine with/without CUDA environment.
- Can compile to run with/without acceleration.
- Essentially all code can run on CPU and GPU.
- As float or double.
- **Aside:** Can also build a Root dictionary and use interactively in a root session(http://root.cern.ch/)

```
#ifdef __CUDACC__
#  define __HOST_DEVICE__  __host__ __device__
#else
#  define __HOST_DEVICE__
#endif
namespace utafd
{
 // Vector3 class definition ____________________________________
 template <class real_t>
 class Vector3 {
 public:
  __HOST_DEVICE__ Vector3() : x_(0), y_(0), z_(0) { }
  //...
  __HOST_DEVICE__ inline real_t dot(const Vector3& v2) const;
  //...
 protected:
  real_t x_, y_, z_;
 };
 // Vector3 class Implementation ________________________________
 //...
 template <class real_t>
 __HOST_DEVICE__ real_t Vector3<real_t>::dot(const Vector3& v2) const
 {
  return x_*v2.x_ + y_*v2.y_ + z_*v2.z_;
 }
 //...
}
```

Hide Separate Code Path

- Create wrapper functions for cases where CPU/GPU require different functions:
- CPU code calls as *random_uniform()*
- GPU code calls as *random_uniform(x,a)*
- *x, a* unique per thread

```
namespace utafd
{
 template <typename real_t>
 __HOST_DEVICE__ real_t random_uniform(unsigned long long* rng_x=0,
      unsigned int* rng_a=0)
 {
#if !defined(__CUDA_ARCH__)    //  host (cpu) path
   double r;
   rangen_(r);          // numerical recepies ran2()
   return (real_t) r;
#else                                   // device (gpu) path
   return (real_t) rand_MWC_co(rng_x, rng_a);  // from "CUDA MCML"
#endif
 }
}
```

Service Classes

- Simulation Parameters:
  - Coordinates
  - Atmosphere
  - Physics
  - Detector
- Single initialization point (cpu double); ***copy to*** cpu float, gpu float / double.
- Accessed through a common interface with a dummy argument to resolve data type.

```
// public ( under namespace utafd::service::atmos:: )
__HOST_DEVICE__ const AtmosService<double>& utafd_atmos(double dummy);
__HOST_DEVICE__ const AtmosService<float>&    utafd_atmos(float  dummy);

//private ( under namespace utafd::service::atmos:: )
AtmosService<double> utafd_atmos_d;
AtmosService<float>  utafd_atmos_f;
#ifdef __CUDACC__
  __device__ AtmosService<double>* utafd_atmos_d_dev = 0;
  __device__ AtmosService<float>*  utafd_atmos_f_dev = 0;
#endif

__HOST_DEVICE__ const AtmosService<double>& utafd_atmos(double dummy)
{
#if !defined(__CUDA_ARCH__)
  return utafd_atmos_d;
#else
  return *utafd_atmos_d_dev;
#endif
    }
// + 1 float version
```

Memory Layout

- PMT signal processing and trigger logic has to be done sequentially in time.
- To achieve efficient memory access pattern on the gpu:
  - Reverse memory layout for 2D arrays used for electronics signal processing
  - Use macro to select the appropriate class in __host__ __device__ code.

```
#ifdef __CUDACC__
#  define SAHTubeSignal SAHTubeSignalDevice
#  define SAH_SIGNAL(i,j) sah_tube.signal_[j][i]
#else
#  define SAHTubeSignal SAHTubeSignalHost
#  define SAH_SIGNAL(i,j) sah_tube.signal_[i][j]
#endif

namespace utafd
{
  class SAHTubeSignalHost {
  public:
    float  signal_[256][MAX_TIME_NBIN]; // PMT signal
    //...
  };

  class SAHTubeSignalDevice {
  public:
    float  signal_[MAX_TIME_NBIN][256];  // PMT signal
    //...
  };
  //...
}
```

Kernel Launch Configuration

- Table shows kernel launch configuration on Fermi card with 1GB of memory.

Kernel	tpb	nb
generate track segments	32	NE/tpb
shower profile	64	$NE \times NS/tpb$
shower photons (step 1)	32×32	$NE \times NS/tpb.x$
shower photons (step 2)	32	NE
mirror photo-electrons	64	$NME \times NS/tpb$
prep electronics	32	NME/tpb
init electronics	256	MLM
ray tracing	64	$MLM \times 4$
electronics	256	MLM

Table 1: Kernel configuration: tpb = threads per block, nb = number of blocks per launch, NE = total number of shower events, NS = number of track segments, NME = total number of mirror-events, MLM = maximum number of mirrors per launch.

Kernel Launch Configuration

- All kernels are launched using a similar setup.
- All prefer L1 Cache over User managed Shared Memory.

```
#define LAUNCH_CUDA_KERNEL(KERNEL,dimGrid,dimBlock,...)        \
{                                                               \
  cudaEvent_t start, stop;                                      \
  cudaEventCreate(&start);                                      \
  cudaEventCreate(&stop);                                       \
                                                                \
  cudaEventRecord(start, NULL);                                 \
                                                                \
  cudaFuncSetCacheConfig(KERNEL, cudaFuncCachePreferL1);        \
                                                                \
  KERNEL<<<dimGrid,dimBlock>>>(__VA_ARGS__);                    \
                                                                \
  cudaEventRecord(stop, NULL);                                  \
  while ( cudaEventQuery(stop) == cudaErrorNotReady ) { usleep(100); }  \
                                                                \
  cudaEventDestroy(start);                                      \
  cudaEventDestroy(stop);                                       \
  cudaThreadSynchronize();                                      \
  utafd_check_cudastat(__FILE__,__LINE__);  /* exits if error */ \
}                                                               \
```

GPU Performance Analysis

Performance comparison – running time

- Desktop CPU vs. Desktop GPU

E (eV)	100 evt (cpu) t in sec.	100 evt (gpu) t in sec.	1000 evt (cpu) t in sec.	1000 evt (gpu) t in sec.
10^{17}	13.91	0.759 (18x)	140.97	2.860 (49x)
10^{18}	18.76	0.875 (21x)	164.59	3.348 (49x)
10^{19}	24.64	1.006 (24x)	196.99	4.230 (47x)
10^{20}	40.72	1.297 (31x)	364.44	6.834 (53x)

Nvidia Virtual Profiler results

- Fraction of time spent in each function.

```
[0] GeForce GT 420M
  Context 1 (CUDA)
    MemCpy (HtoD)
    MemCpy (DtoH)
    Compute
      40.5% [9] utafd::device_raytrace_mirror_npe(MemStruct, MemStructShower, int)
      21.4% [9] utafd::device_sah_add_sky_noise(MemStruct, MemStructShower, int)
      14.9% [9] utafd::device_sah_electr(MemStruct, MemStructShower, int)
      3.8% [1] utafd::device_generate_mirror_npe(MemStructShower)
      2.0% [1] utafd::device_generate_track_segments(MemStructShower)
      1.3% [1] utafd::device_generate_shower_photons_seg(MemStructShower)
      0.3% [1] utafd::device_generate_shower_photons_beam(MemStructShower)
      0.3% [1] utafd::device_generate_shower_profile(MemStructShower)
      0.1% [1] utafd::device_calculate_event_duration(MemStructShower)
      0.0% [9] memset (16843009)
      0.0% [51] memset (0)
```

Nvidia Virtual Profiler results

utafd::device_raytrace_mirror_npe(MemStruct, MemStructShower, int)

Name	Value
Start	884.765 ms
Duration	69.888 ms
Grid Size	[416,1,1]
Block Size	[64,1,1]
Registers/Thread	58
Shared Memory/Block	8 bytes
Memory	
Global Load Efficiency	36.4%
Global Store Efficiency	0.4%
DRAM Utilization	5.1% (1.24 GB/s)
Instruction	
Branch Divergence Overhead	73.5%
Total Replay Overhead	0.9%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	0.1%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0.7%
Occupancy	
Achieved	23.2%
Theoretical	33.3%
Limiter	Block Size

Nvidia Virtual Profiler results

utafd::device_sah_add_sky_noise(MemStruct, MemStructShower, int)

Name	Value
Start	323.647 ms
Duration	43.577 ms
Grid Size	[32,1,1]
Block Size	[256,1,1]
Registers/Thread	16
Shared Memory/Block	0 bytes
▽ Memory	
Global Load Efficiency	94.5%
Global Store Efficiency	82.1%
DRAM Utilization	13.5% (3.3 GB/s)
▽ Instruction	
Branch Divergence Overhead	⚠ 48.8%
Total Replay Overhead	1.4%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	1.4%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0%
▽ Occupancy	
Achieved	91.6%
Theoretical	100%

Nvidia Virtual Profiler results

utafd::device_generate_track_segments(MemStructShower)

Name	Value
Start	180.226 ms
Duration	33.701 ms
Grid Size	[3,1,1]
Block Size	[32,1,1]
Registers/Thread	46
Shared Memory/Block	0 bytes
Memory	
Global Load Efficiency	14.6%
Global Store Efficiency	12.5%
DRAM Utilization	2.1% (535.13 MB/s)
Instruction	
Branch Divergence Overhead	26.4%
Total Replay Overhead	28.6%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	26.8%
Global Cache Replay Overhead	1.8%
Local Cache Replay Overhead	0%
Occupancy	
Achieved	2.7%
Theoretical	16.7%
Limiter	Block Size

Summary and Outlook

- An entire software suite used for the detector simulation of a real astrophysics experiment is ported to CUDA.
- Verified float precision is adequate for the MC
- GPU provides >10x performance of a quad core CPU.
- Next is to do event reconstruction, inverse Monte Carlo, on the GPU. This would be big!
    - Need to implement minimization routine on GPU.
- Looking forward for running on Kepler as well.

Backup Slides

floats vs. double precision

- NOTE: in the following comparisons I mention the function xyz2h()
- This function calculates the height of a point in (x,y,z) coordinates above sea level. The calculation involves the "Earth Radius":
  - $R_E >> z$ causes the calculation to be inaccurate when using ***floats***.

Quick Test Set

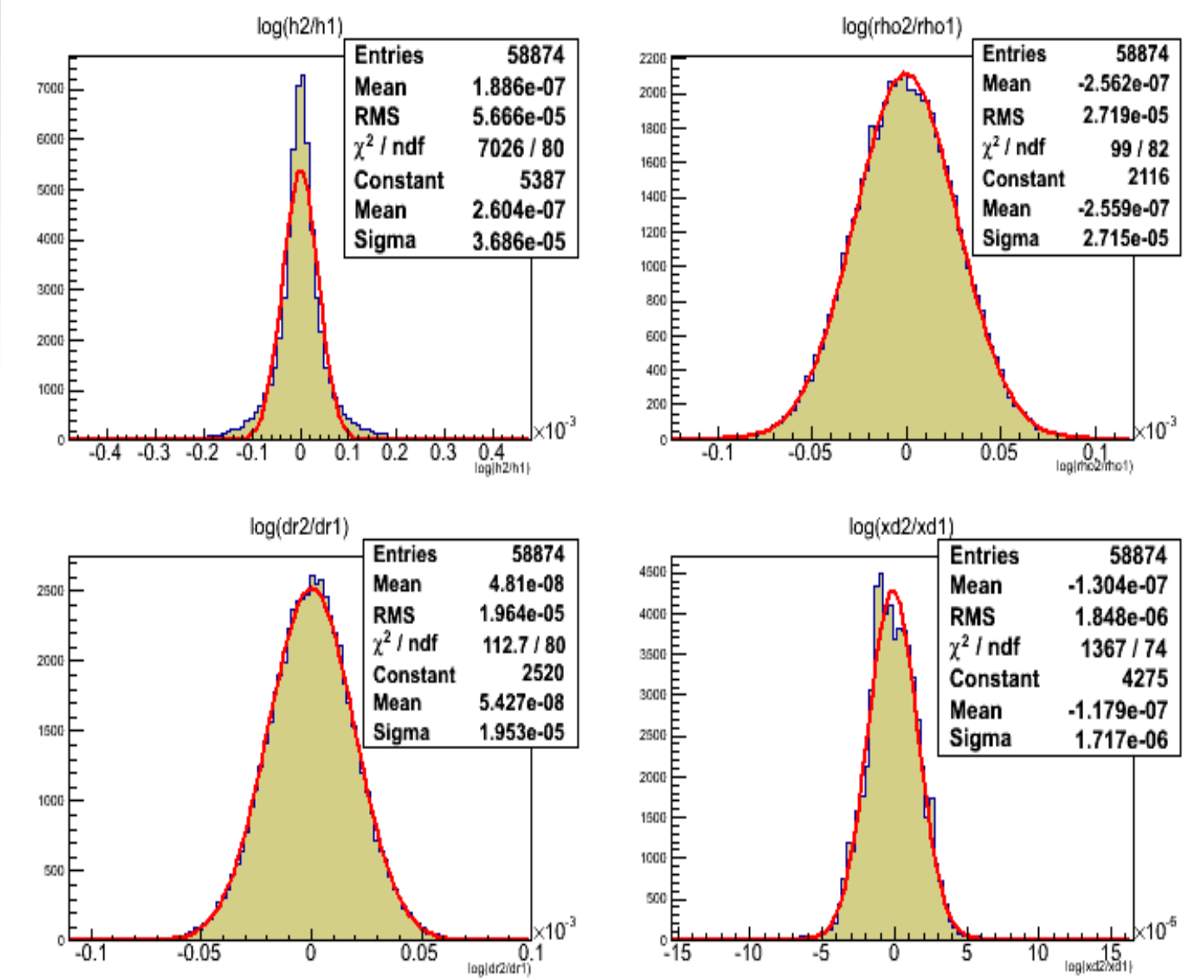
- 150 Thrown showers: 10^{17} eV proton.
- Run once without GPU acceleration, using double precision.
- Run once with GPU acceleration, using single precision math on the GPU.
- Write out intermediate calculation results for comparison.
- Plot results

Shower Simulation Step 1: Generate track segments

- Subdivide shower track to 832 segments; (~1-2 g/cm^2.
- For each segment calculate:
  - Height at center of segment.
  - Atmospheric Density.
  - Segment length (meters & g/cm^2)
  - Slant depth at center of segment (g/cm^2)
  - ...
- Comparison follows.

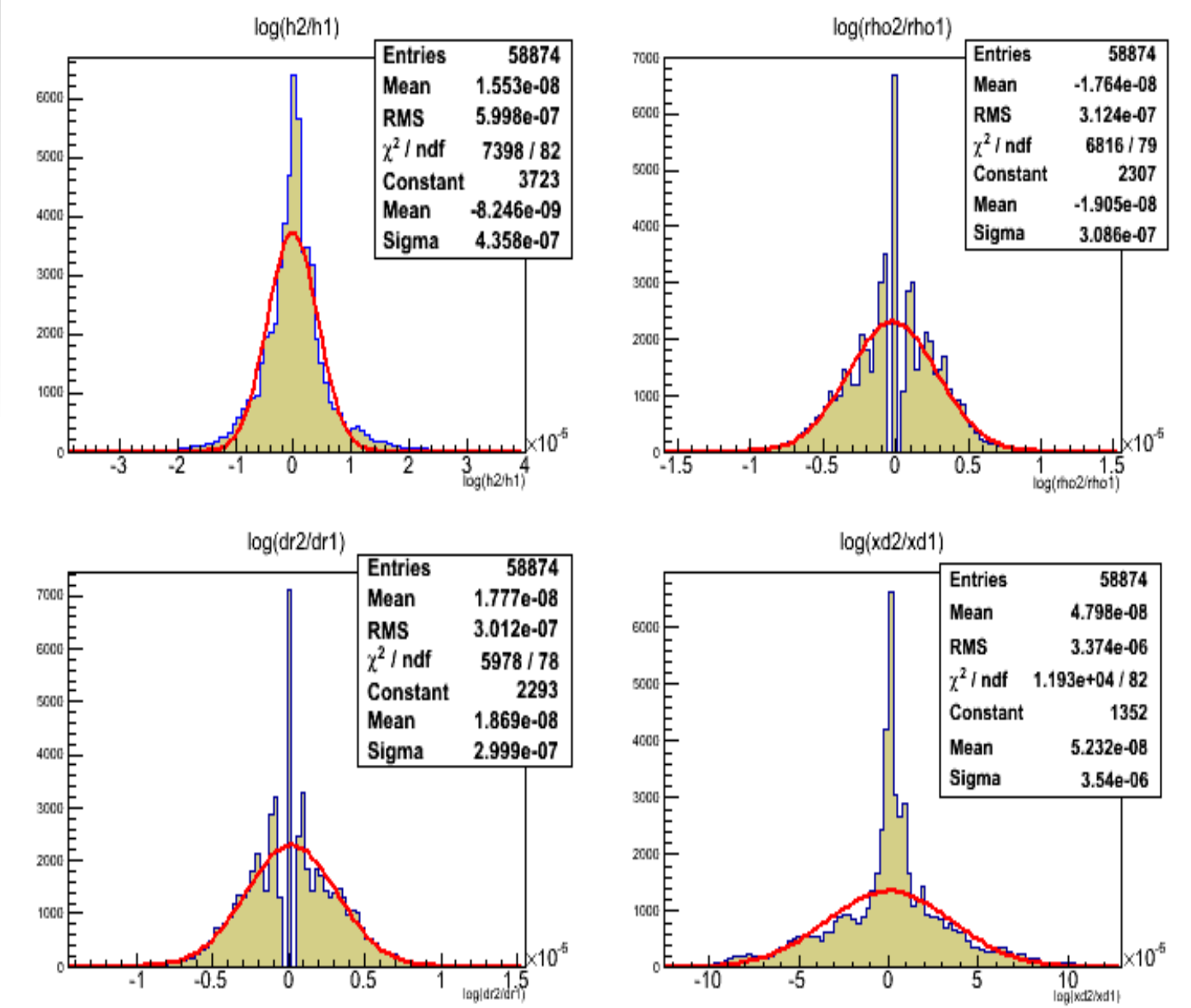
Track Segments (All floats)

- Each data point correspond s to a track segment in one of the simulated showers which was in MD FOV.



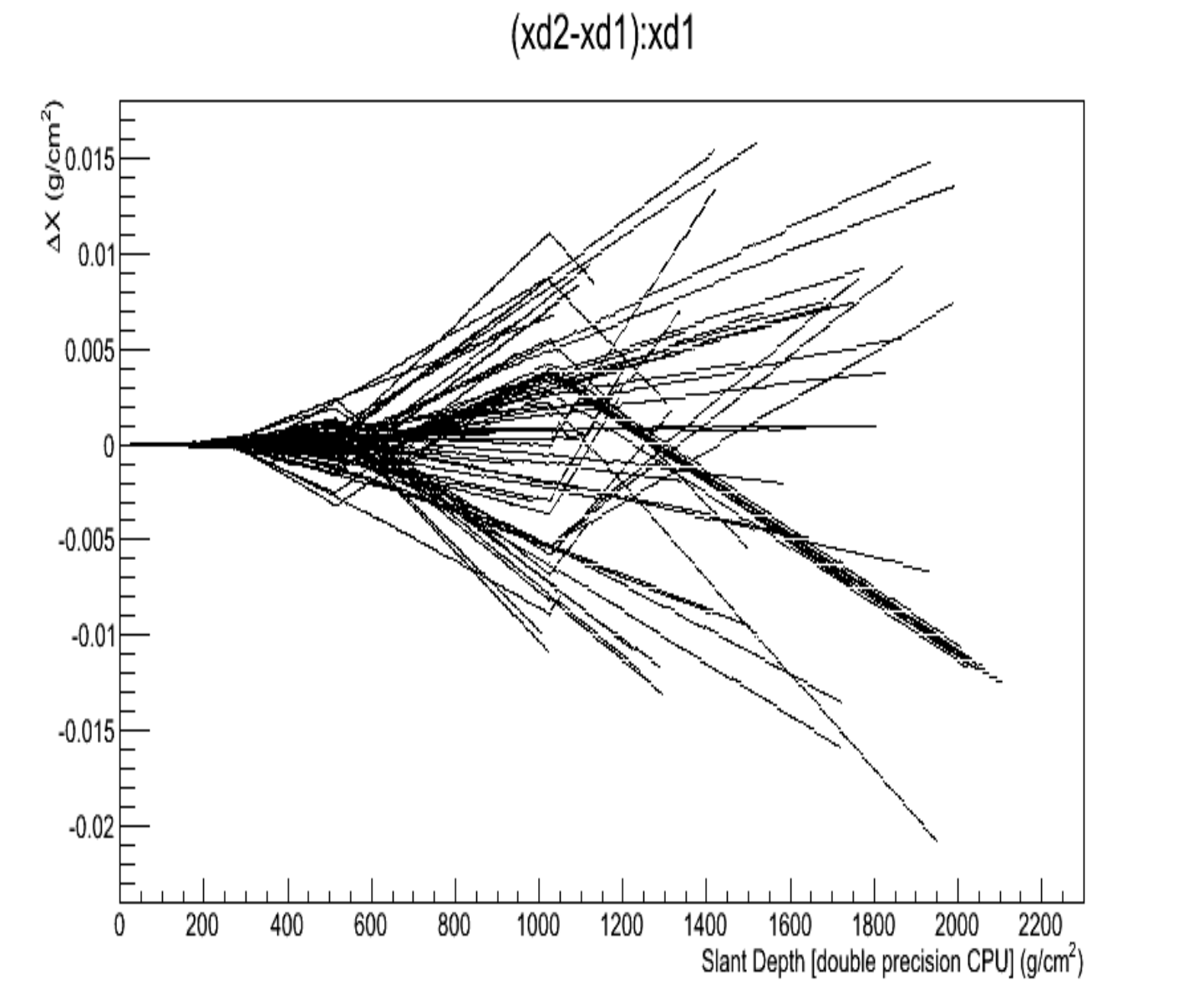
Track Segments (double xyz2h)

- Each data point correspond s to a track segment in one of the simulated showers which was in MD FOV.



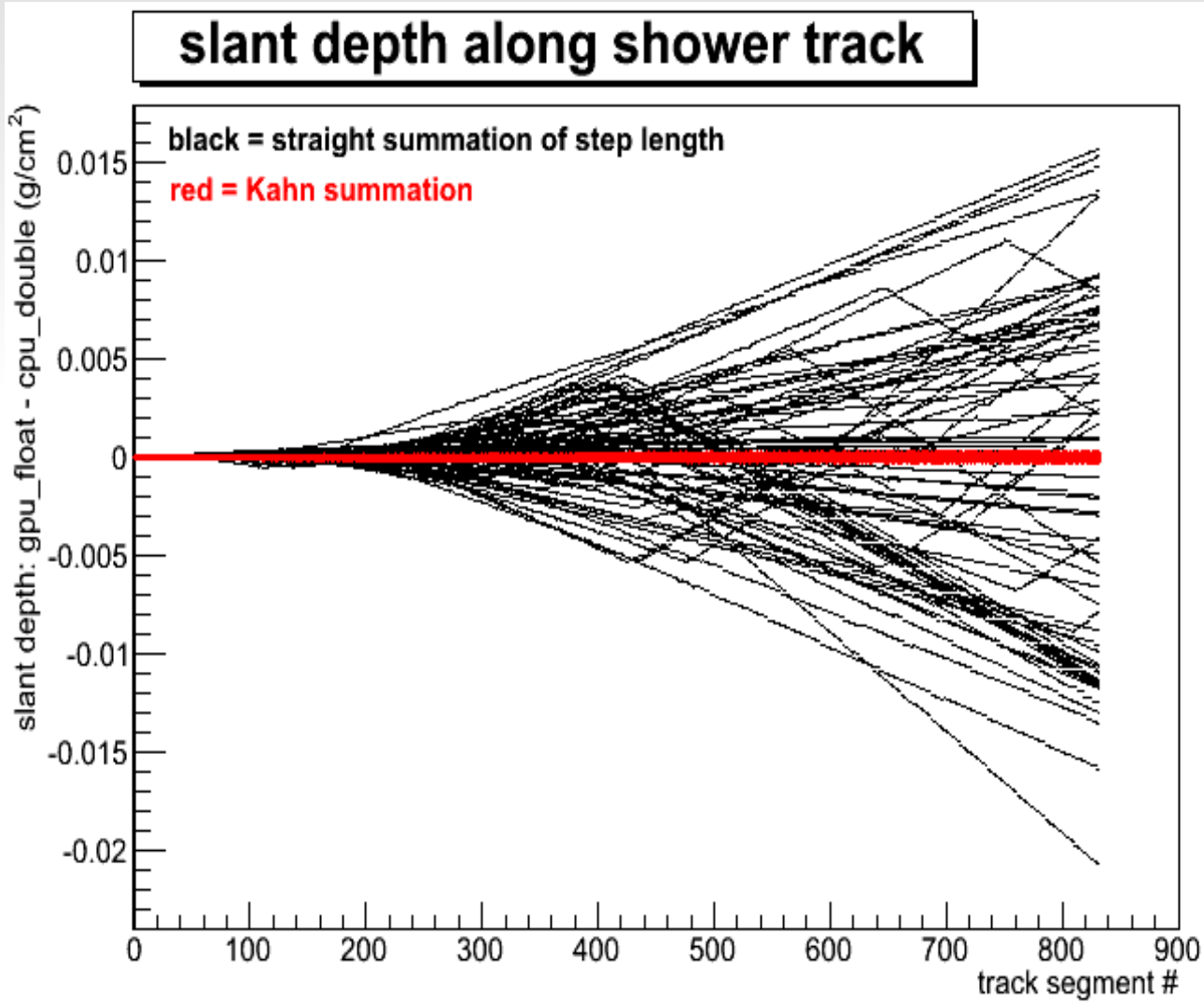
Track Segments (double xyz2h)

- Error on Slant Depth Calculation (floats) vs Slant Depth calculated with double precision.



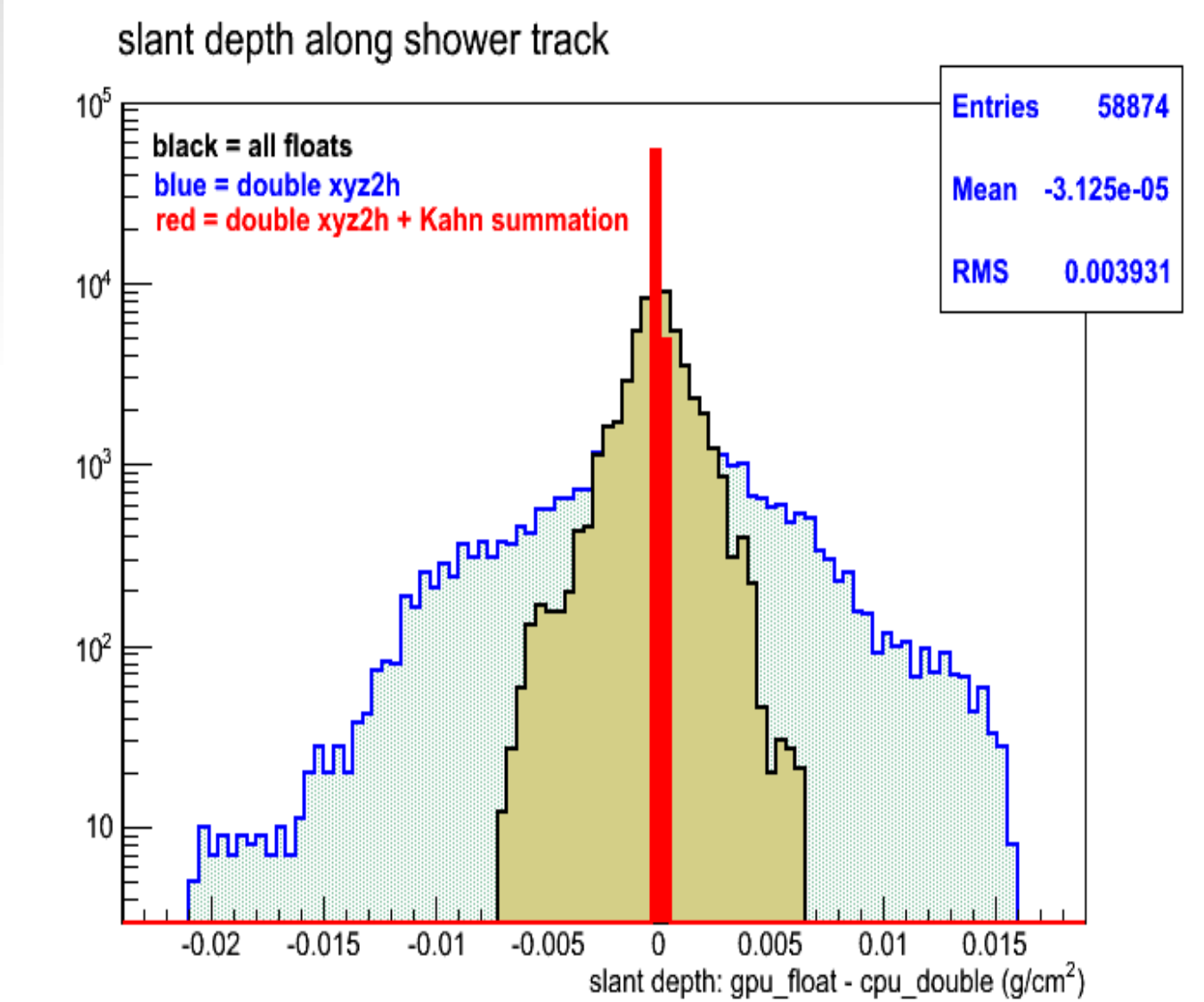
Track Segments (Kahn Summation)

- Error on Slant Depth Calculation (floats) after use of Kahn Summation to sum up individual track



Track Segments (Kahn Summation)

- Error on Slant Depth Calculation (floats) after use of Kahn Summation to sum up individual track

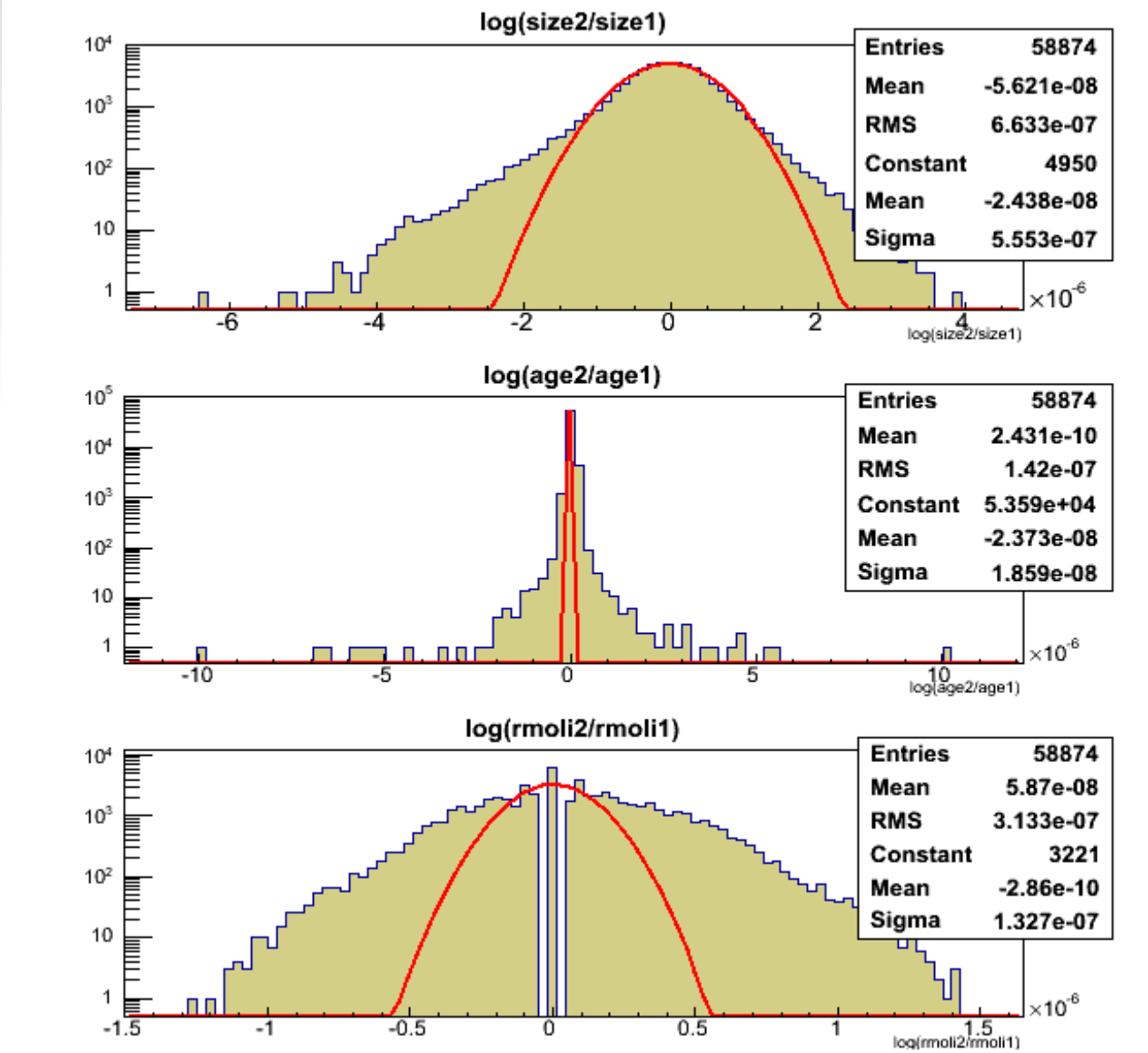


Shower Simulation Step 2: Generate shower profile

- For each track segment calculate:
  - Shower Size $N_e(x)$.
  - Shower age.
  - Moliere radius. (used for lateral spread calculation)
  - Total amount of Fluorescence light production.
  - Total amount of Cerenkov light production.
- Comparison follows.

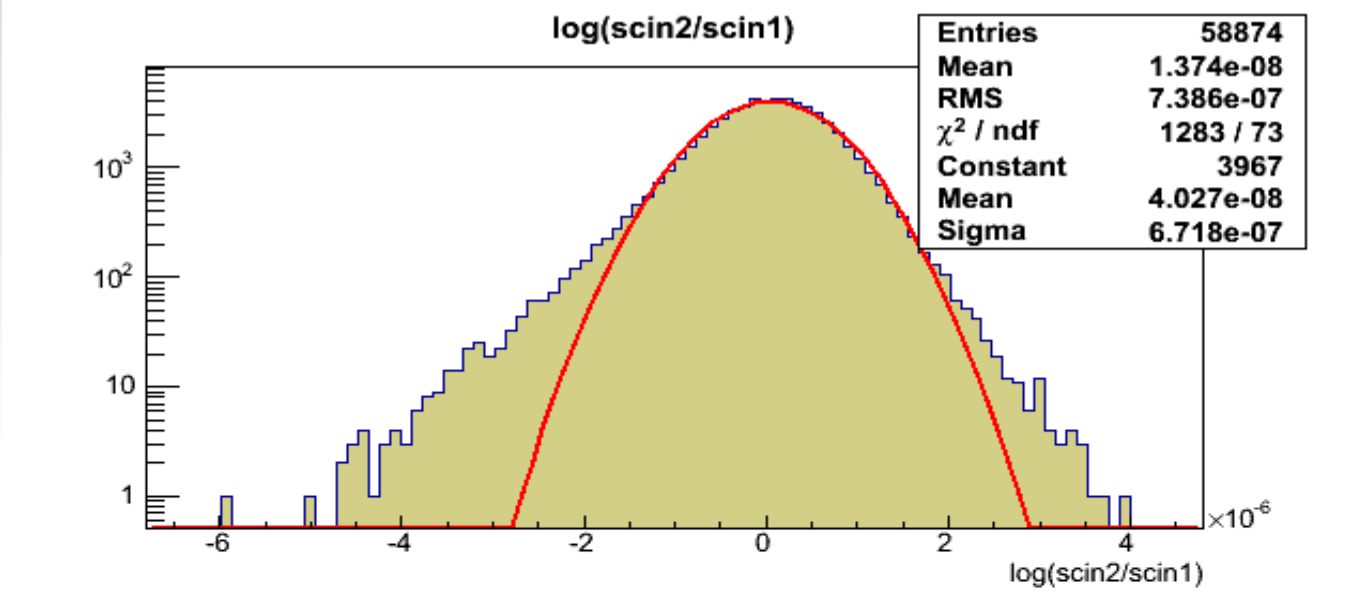
Shower profile

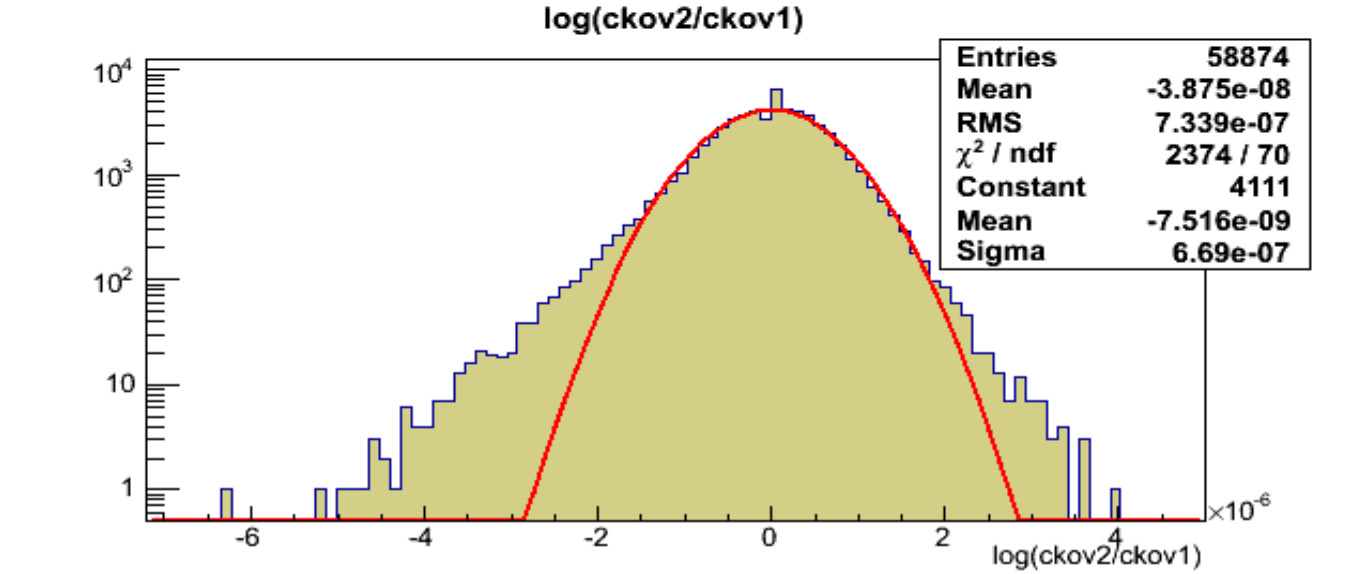
- One data point for each track segment:
  - Shower size
  - Shower age
  - Moliere Radius



Shower profile (cont'd)

- One data point for each track segment:
  - Total Fluorescence
  - Total Cerenkov



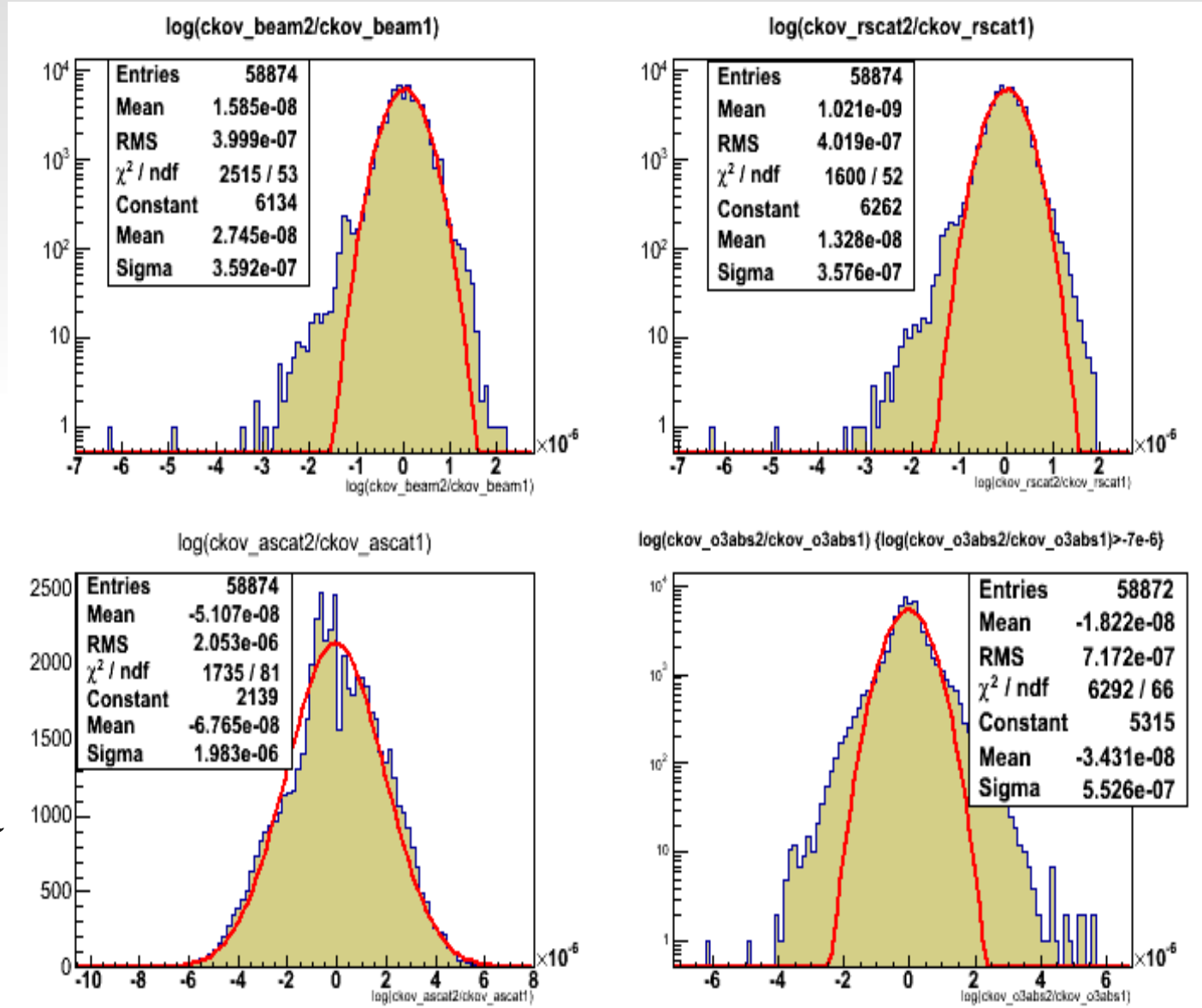


Shower Simulation Step 3: Generate shower photons

- For each track segment calculate:
  - Total amount of Fluorescence light; divided into wavelength bins … currently using FLASH.
  - Total amount of Cerenkov light; divided into wavelength bins.
  - Rayleigh & Aerosols scattered Cerenkov.
  - Total Cerenkov (Accumulates along shower development)
- Comparison follows.

Shower photons (Cerenkov)

- One data point for each track segment (summed over wavelength) :
  - Accumulated C'kov
  - Rayleigh

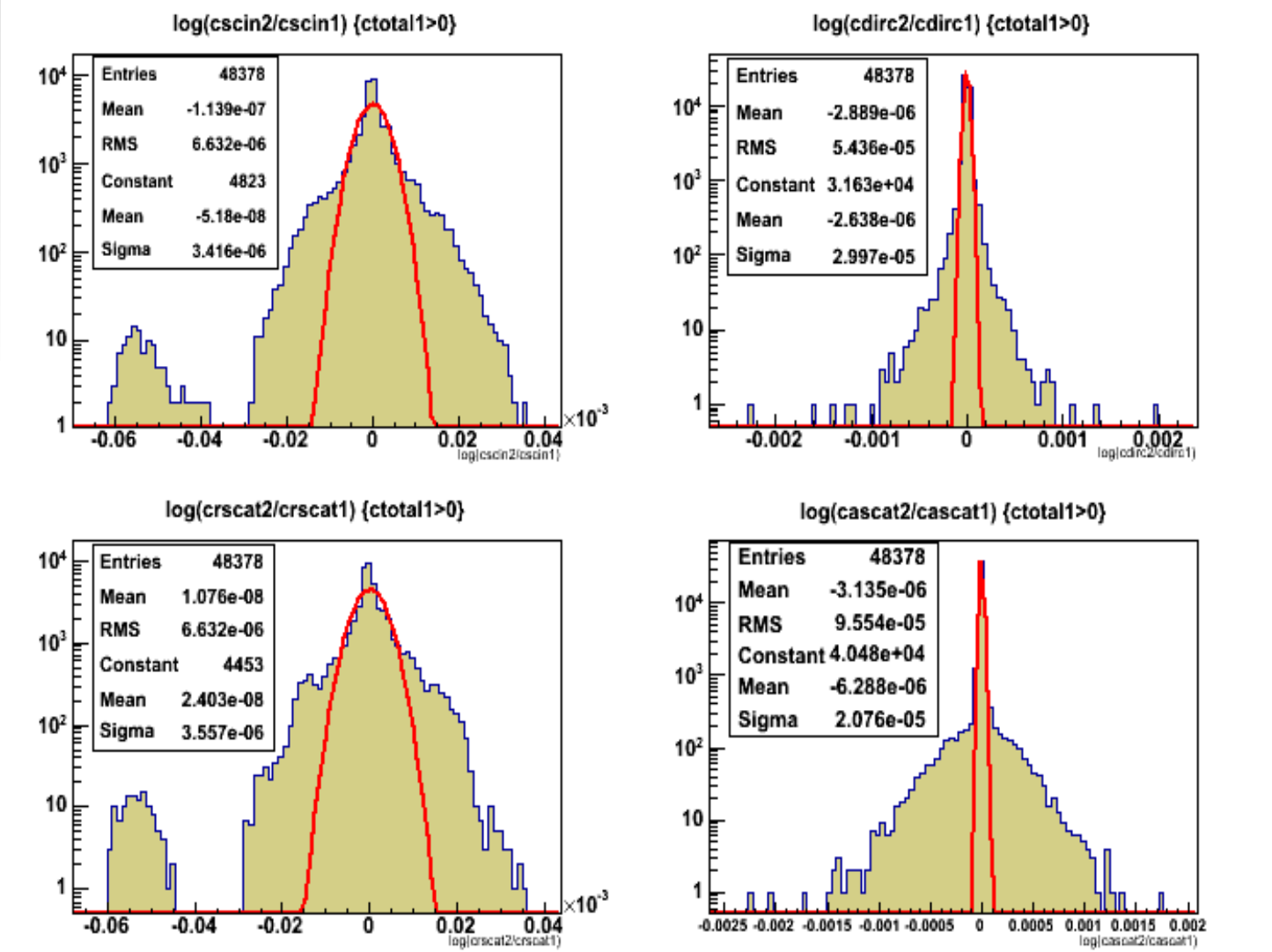


Shower Simulation Step 4: Generate Mirror photons (pe's)

- Each event in the FOV of the detector can be seen by one or more mirrors (telescopes).
- A telescope will see a portion of the shower (A subset of the shower track segments).
- For each track segment (in telescope FOV) calculate:
  - Total amount of Fluorescence pe's (*flux* x *mirror_area*)
  - Total amount of Cerenkov pe's including
    - Rayleigh & Aerosols scattered Cerenkov.
    - Direct Cerenkov

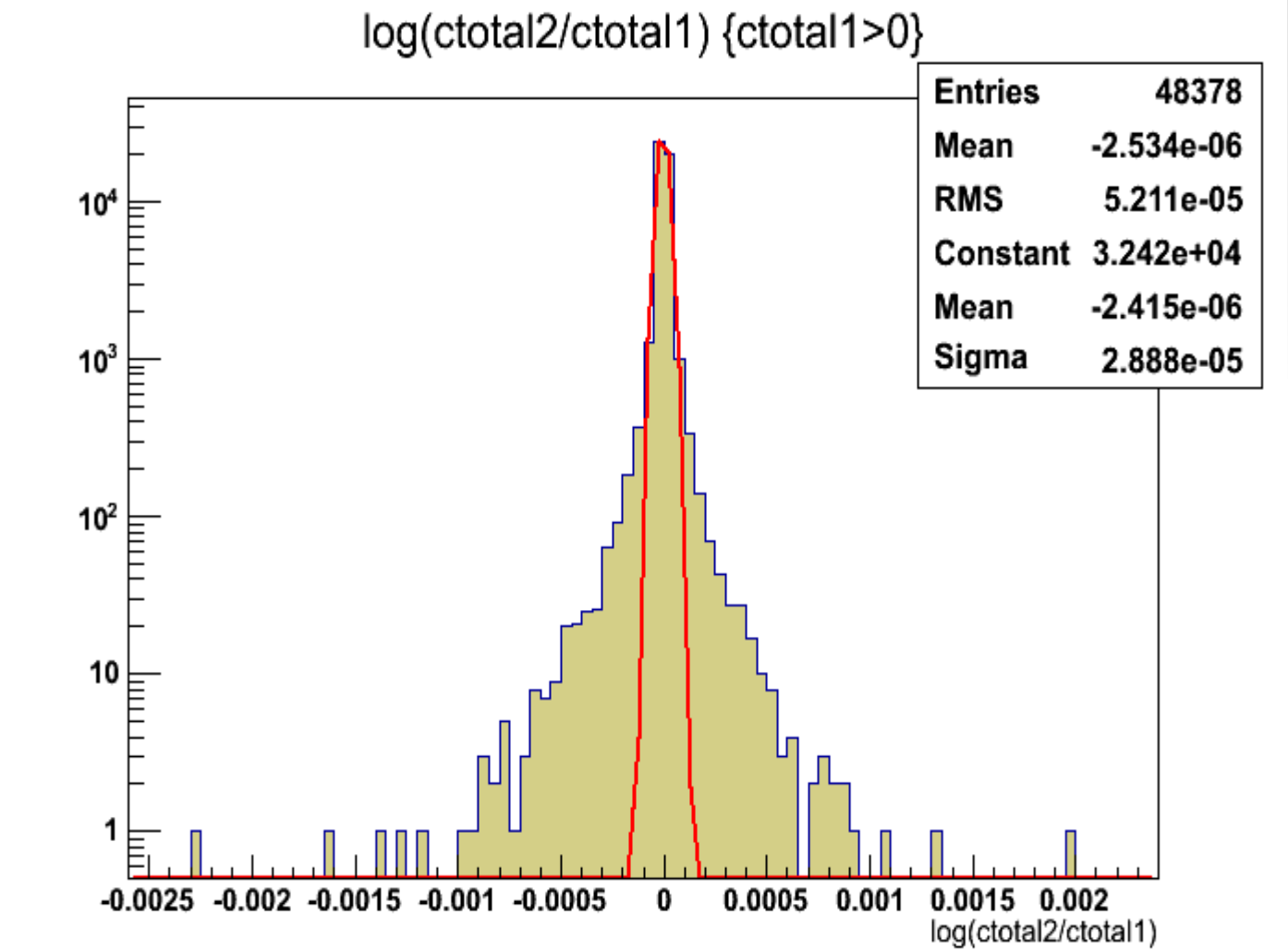
Mirror photo-electrons

- One data point for each track segment:
  - Fluorescence pe's



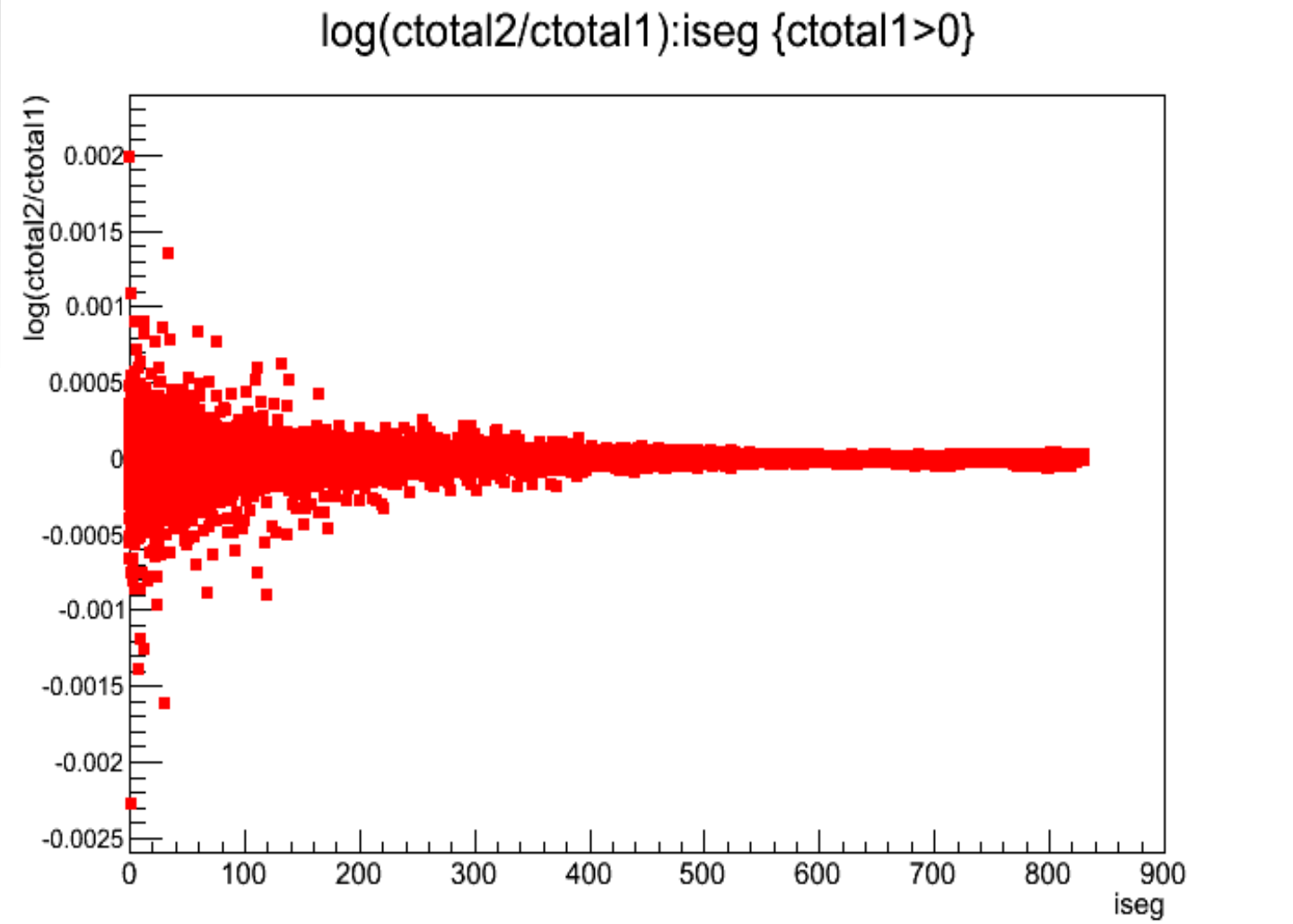
Mirror photo-electrons

- One data point for each track segment:
  - Total # pe's
- End of Physics.



Mirror photo-electrons

- One data point for each track segment:
  - Total # pe's



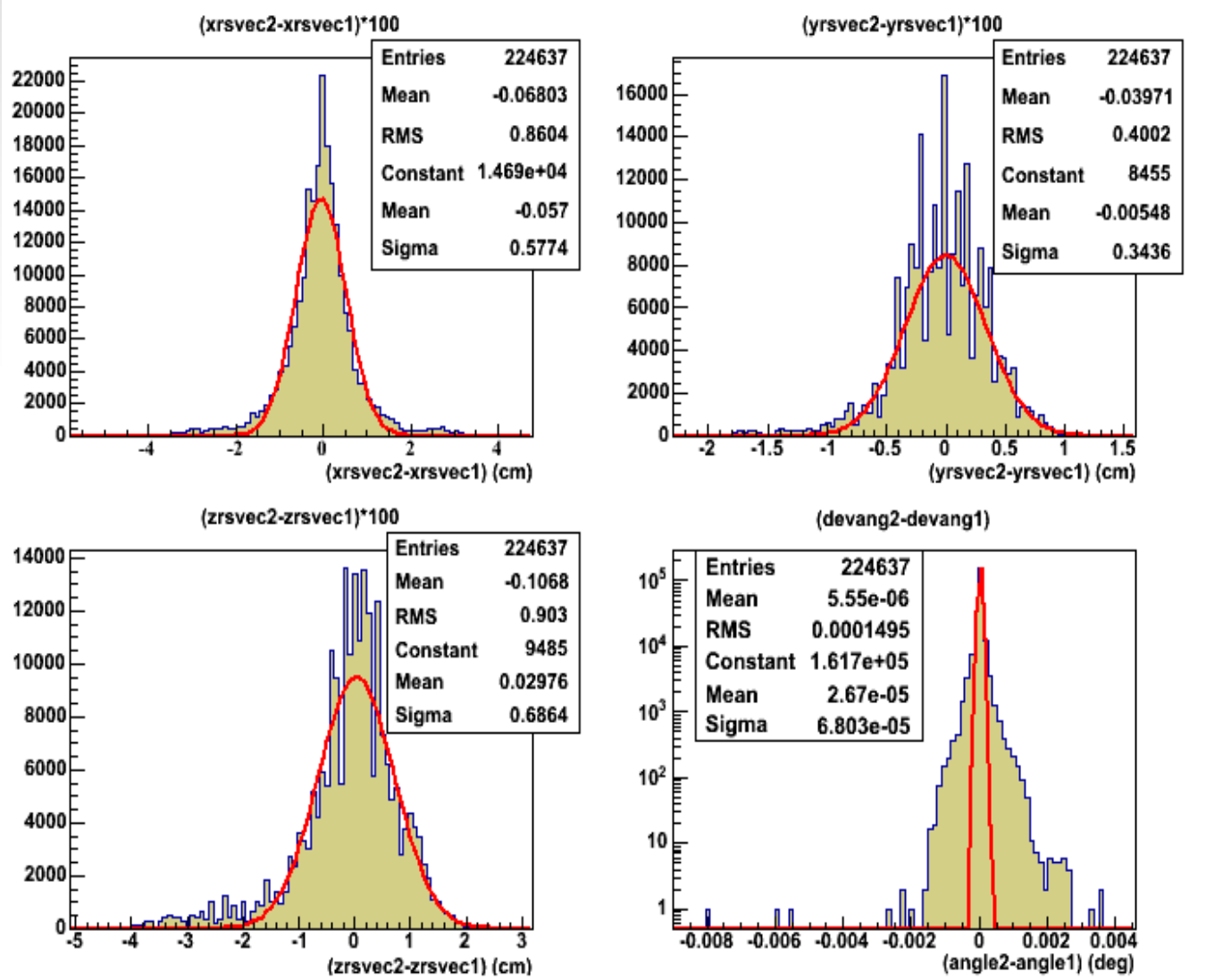
- Largest differen

Mirror photo-electrons

- How much error can we tolerate?
  - Next step in the simulation is ray-tracing. The variable "ctotal" is used as the mean of a Poisson distribution:
  - int raytrace_npe = random_poisson(ctotal);
- As long as we are within 0.1% (10^{-3}), I highly doubt we can tell the difference.
- We get $O(10^{-5})$

Mirror photo-electrons

- Needed for ray tracing:
- Transformed Coordinates:
  - rvec_x, rvec_y, rvec_z
  - angle made with mirror axis.



More comparisons ...

- Still need to verify
  - Ray tracing accuracy, including sampling of shower lateral spread.
  - Random number generator works as expected.
  - Electronics simulation is correct.
- Harder to do because of the use of different random number generators and therefore different random sequences.
- Can be done by using statistical averages.
- **The end result of the simulation (triggered event data) has been checked and the cpu/gpu are in excellent agreement. Still, would be nice to have the Individual checks listed above.**

Fluorescence Detector Response Simulation

Main components of Simulation

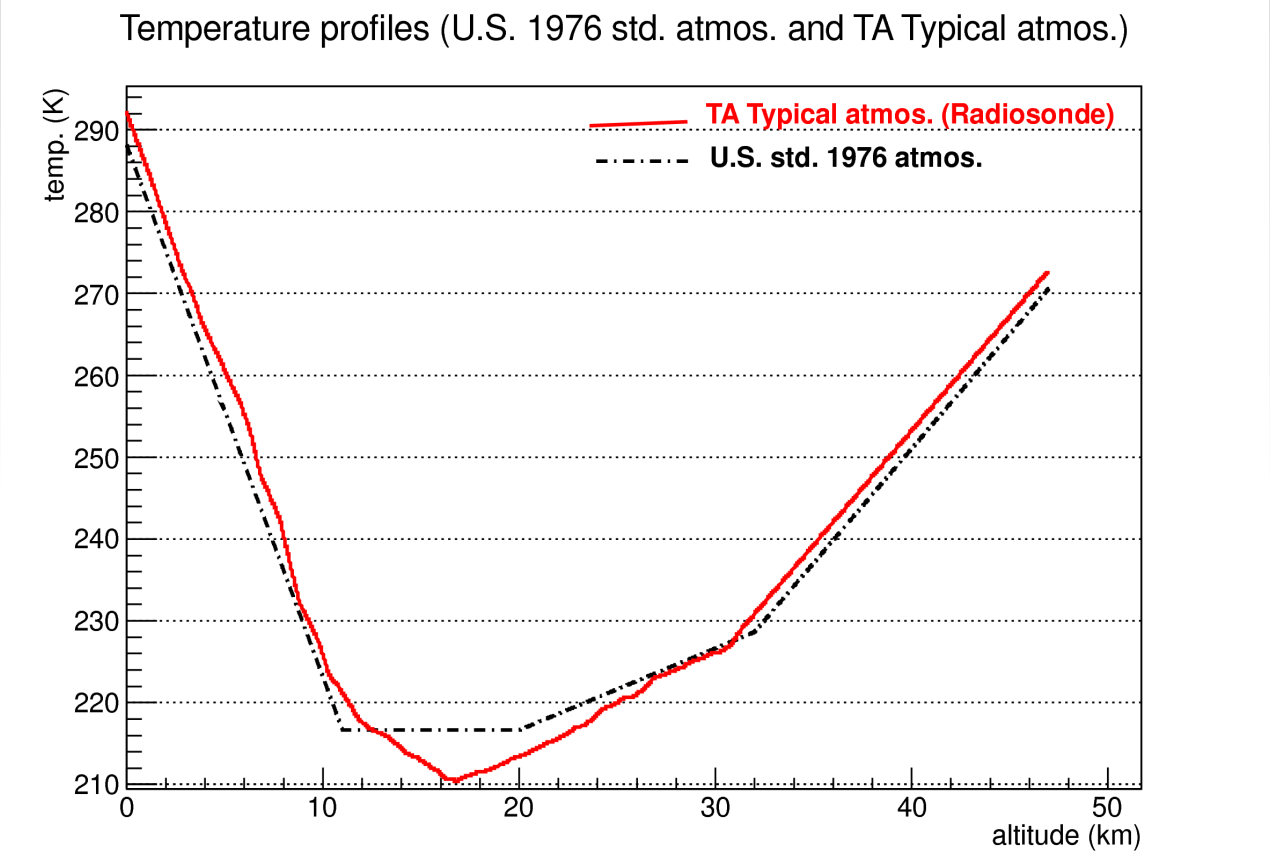
- Atmosphere (Medium)
- Extensive Air Shower (Light Source)
- Detector (Instrument)
- Physics (Interactions)
- Auxiliary
    - RNG
    - Coordinates System
    - Event Management

Coordinates System

- Defines the origin of a Cartesian system (x, y, z), as a point on the surface of the Earth (in *geodetic coordinates*).
- Provides a function to convert (x,y,z) to height (a.s.l.) using a spherical earth approximation.
  - Also (x,y,z) to height above ground (for aerosols density calculation)
- Coordinate transformations to/from alternative systems.
- Single instance per simulation (a service class).

The Atmosphere

- Medium in which shower develops and deposits it's energy, and where light production occurs.
- Single instance per simulation (a service class).
- Pressure, Density, and Temperature profiles:
  - All functions of altitude.
- Also Contains an Ozone Layer and Aerosols near the ground:
  - Altitude dependent as well.



Physics

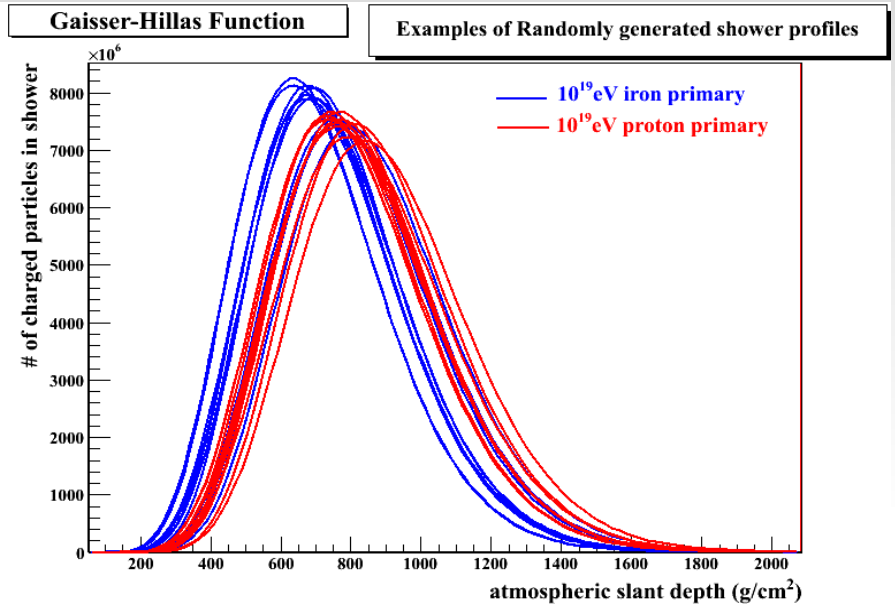
- Different models (experimental measurements) of Fluorescence yield, wavelength spectrum and shower *dE/dx*.
  - User selectable.
- Functions / data to calculate Cerenkov light production and wavelength spectrum.
- Single instance per simulation (a service class).

Extensive Air Shower Parameters

- Primary particle energy:
  - Either explicitly set or randomly sampled from a power law.
  - Determines total # of particles in the shower.
- Shower Geometry (Randomly Sampled):
  - Shower Track can be at any distance from the detector, and can have any downward direction.
- Shower Longitudinal Development Curve (Randomly Sampled):
  - Parametrized; parameters selected randomly from shower library.
- Shower Lateral extent:
  - Parameterized; average shower behavior.

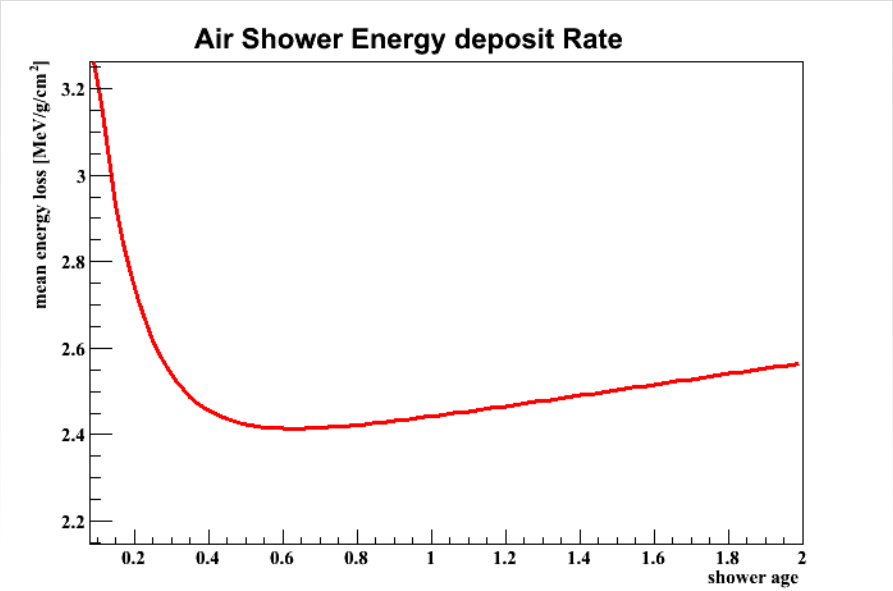
EAS Parameters (cont'd)

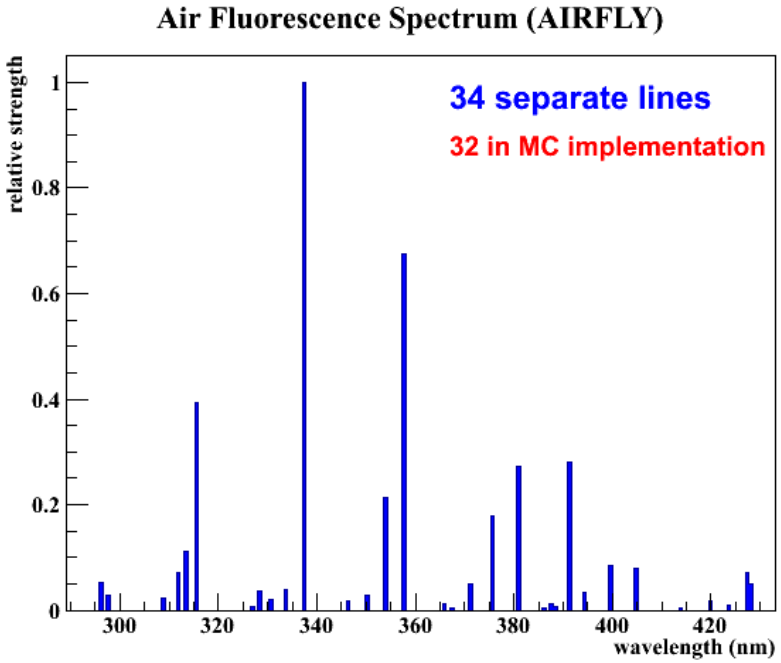
- Shower Longitudinal Profile:
  - Parametrization given by a Gaisser-Hillas function with four parameters
- Lateral Electron Distribution:
  - is described by a parameterization based on the NKG function



Light Production

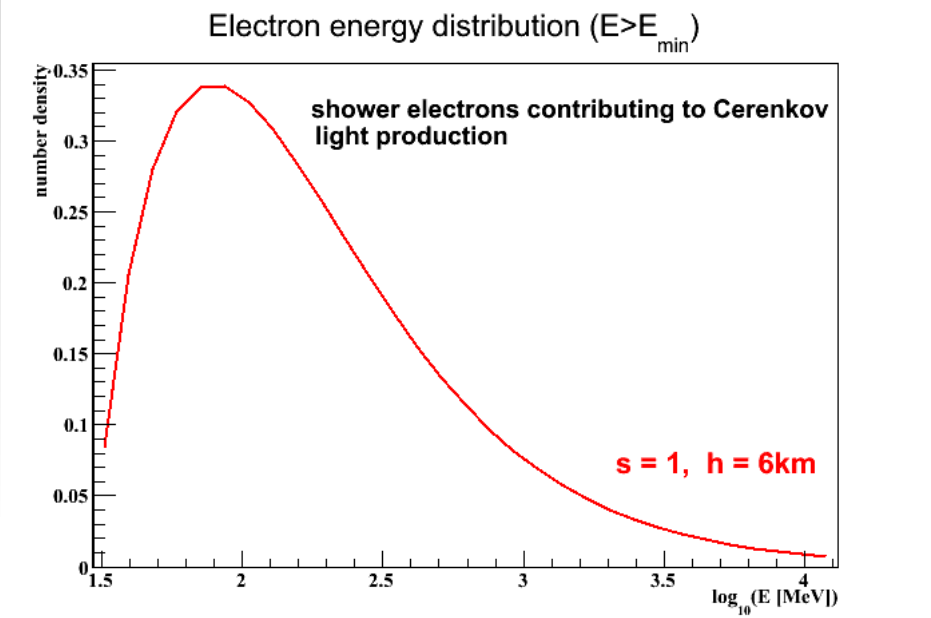
- Air Fluorescence
  - Shower Particles excite Nitrogen Molecules in the atmosphere
  - the molecules give back the energy in the form of UV light.

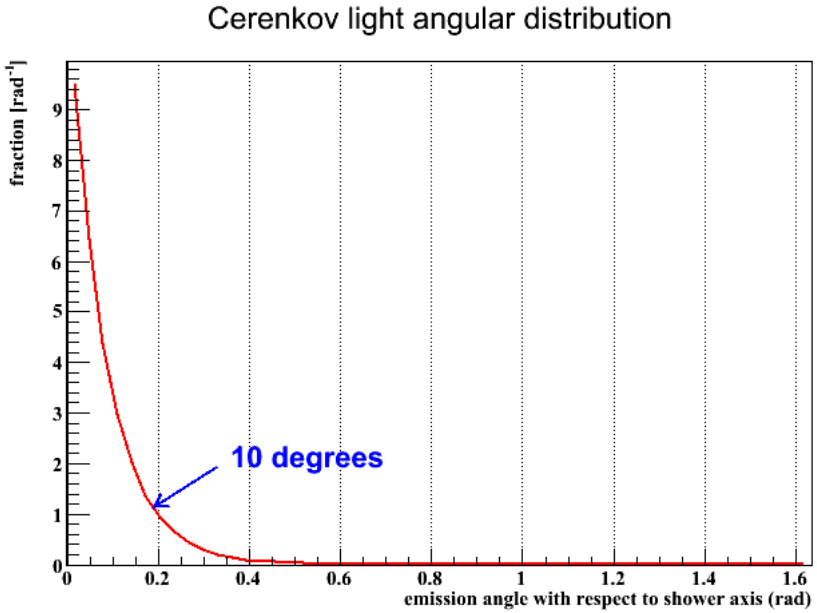




Light Production (cont'd)

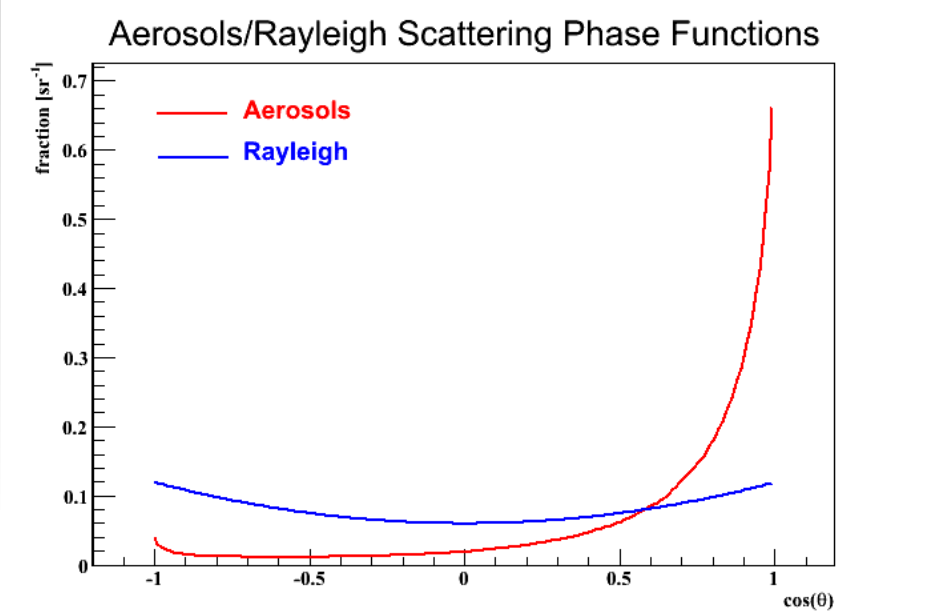
- Cerenkov Radiation.
- Shower particles traveling at speeds faster than the speed of light in air emit Cerenkov radiation.
- We observe direct Cerenkov light.
- We also see scattered light
  - Scattering by air molecules

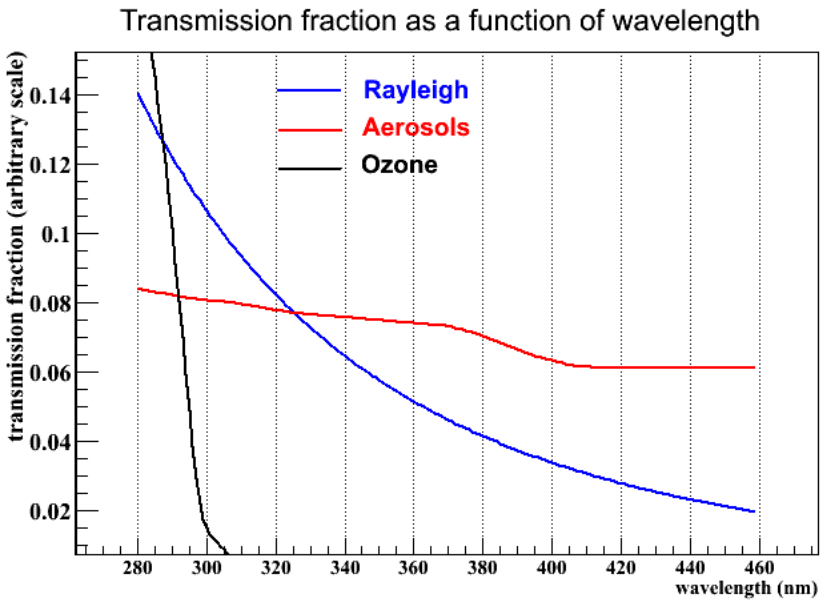




Light Propagation

- Light Scattering
  - Air Molecules (Rayleigh)
  - Aerosols (Empirical Model)
- Ozone Absorption.
- Function of:
  - Wavelength
  - Altitude (density)



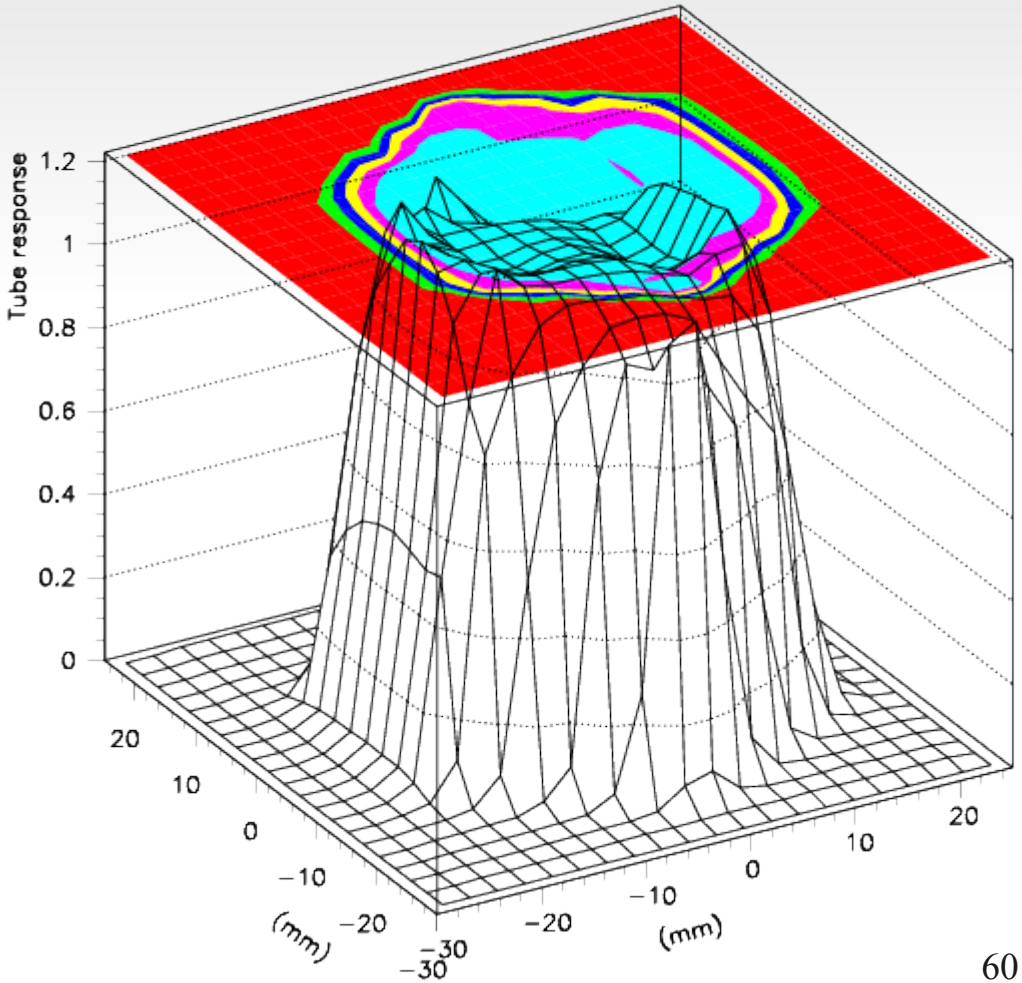


Light Detection

- Spherical mirror collects photons and focuses them to a PMT cluster.
- Photo-Multiplier Tubes convert photons to photoelectrons (electrical signal)

Detector Efficiency

- Mirror Reflectivity
- UV Filter Transmission
- PMT Quantum efficiency
  - are all wavelength dependent.
- PMT efficiency profile is also accounted for in the simulation.



Electronics

- Signal processing for PMT trigger:
  - RC filter
  - LC filter + threshold test
  - Integrate pulse if trigger.
- Record data.
- "Mirror Trigger Logic" for event formation done on CPU.