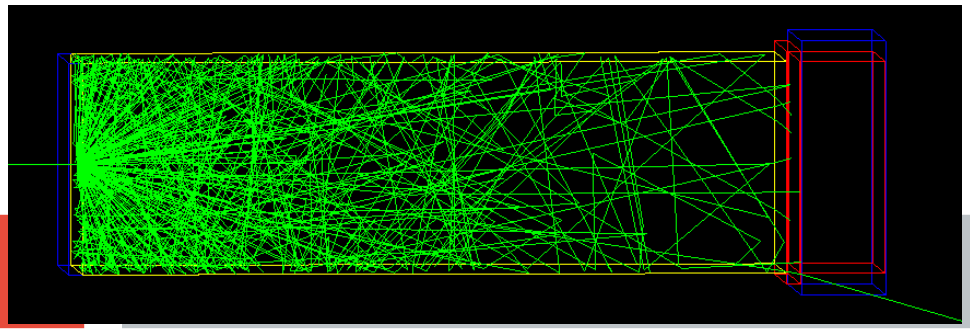
GATE developments at CPPM

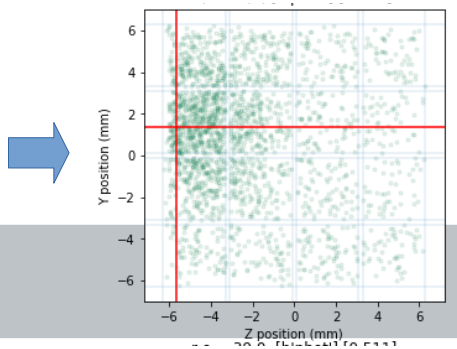
OpenGATE Meeting - Lyon 04th july

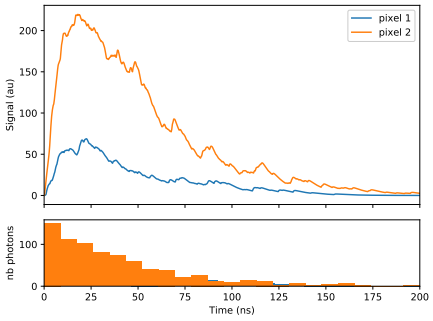
Mathieu Dupont, mdupont@cppm.in2p3.fr
Aix-Marseille Univ, CNRS/IN2P3, CPPM, Marseille, France

CPPM

- **CPPM: Centre de Physique des Particules de Marseille**
  - Particle physics, Astroparticle physics, Observational cosmology
- **The X and Gamma Imaging team (imXgam) lead by Christian Morel :**
  - Technology transfer between Particle physics and Medical or societal application :
  - Main field where GATE is used:
    - µCT spectral development with XPAD3 detecor
    - Calorimetry : We are currently developing SiPM digitizer for GATE







Outline

Python output

Optical simulation in GPU

Python output in GATE ?

- **Why ?**
  - Python widely used for data analysis
  - We don't like ROOT
  - Compile root_numpy can be cumbersome (especially when dealing with updates)
  - Can be good idea to make GATE not so dependant on ROOT
    - We want to keep ROOT output but be able to compile GATE without ROOT
    - Working group created during Hackaton
- **Occasion to rewrite GATE output machinery ?**
  - To make easier to write new output format
  - Make easier to choose

Python output

Add new dependancy to GATE ?

Not necessarily, numpy array can be written without library,

- **Numpy file =**
  - Header = 6 magic bytes "\\x93NUMPY"
    - + the major version number of the file format, e.g. \\x01.
    - + 1 byte is an unsigned byte: the minor version number of the file format, e.g. \\x00
    - + 2 bytes form a little-endian unsigned short int: the length of the
    - + header data HEADER_LEN
    - + array's format as dictionnay :
      - Descr + fortran_order + shape
  - + Data

Example array format:
File with 10’000 elements
One element = one string, one uint32_t and one double :

```
{'descr':
[   ('name', '|S10'),
    ('trackID', '<u4'),
('double_number','<f8')
],
'fortran_order': False,
'shape': (10000,),
}
```

API for Python Output in GATE :

```
class GateOutputNumpyTreeFile : public GateNumpyTree,
public GateOutputTreeFile
{
public:
  GateOutputNumpyTreeFile();
  void open(const char* s) override;
  void fill() override ;

  void close() override ;
  void write_header() override ;


  void write_variable(std::string name, const void
*p, std::type_index t_index) override;
  void write_variable(const std::string name, const
char *p, size_t nb_char) override;
  void write_variable(const std::string name, const
std::string *p, size_t nb_char) override;


  template<typename T>
  void write_variable(const std::string name, const
T *p)
  {
    write_variable(name, p, typeid(T));
  }

}
```

```
GateOuputNumpyTreeFile h;
h.open("/tmp/z.npy");

uint32_t trackID = 0;
h.write_variable("trackID", &trackID);

double gauss = 0.;
h.write_variable("double_number", &gauss);

string str = "wxcvb";
h.write_variable("name", &str, 10);

h.write_header();

std::default_random_engine generator;
std::normal_distribution<double> distribution(5.0,2.0);

for (int i = 0; i < 10000; ++i)
{
  gauss = distribution(generator);
  str = "plop" ;

  trackID++;
  h.fill();
}

h.close()
```

More generic ?

```
class GateOutputTreeFileManager
{
  GateOutputTreeFileManager();


  void add_file(const std::string &file_path,
const std::string &kind);

  template<typename T>
  void write_variable(const std::string name,
const T *p)
  {
    for(auto&& f : m_listOfTreeFile)
    {
      f->register_variable(name, p, typeid(T)
    }
  }


  std::vector<TreeFile*> m_listOfTreeFile;

};
```

```
GateOutputTreeFileManager m;


m.add_file("/tmp/z.npy", "npy");
m.add_file("/tmp/z.txt", "txt");
m.add_file("/tmp/z.root",
"root");


uint32_t trackID = 0;
m.write_variable("trackID",
&trackID);

m.write_header();

m.fill() ;

m.close();
```

Already implemented, wait to be integrated

Available here : https://github.com/wrzof/Gate/tree/npy_hits

You can already use it :

/gate/output/tree/enable

/gate/output/tree/addFileName /tmp/p.npy

/gate/output/tree/addFileName /tmp/p.root

/gate/output/tree/hits/enable *# will produce p.hits.npy*

/gate/output/tree/addCollection Singles *# will produce p.Singles.npy*

Which branches to save ?

- Old manner:

Mask : `/gate/output/ascii(binary)/setCoincidenceMask 1 0 1 0 1 1`

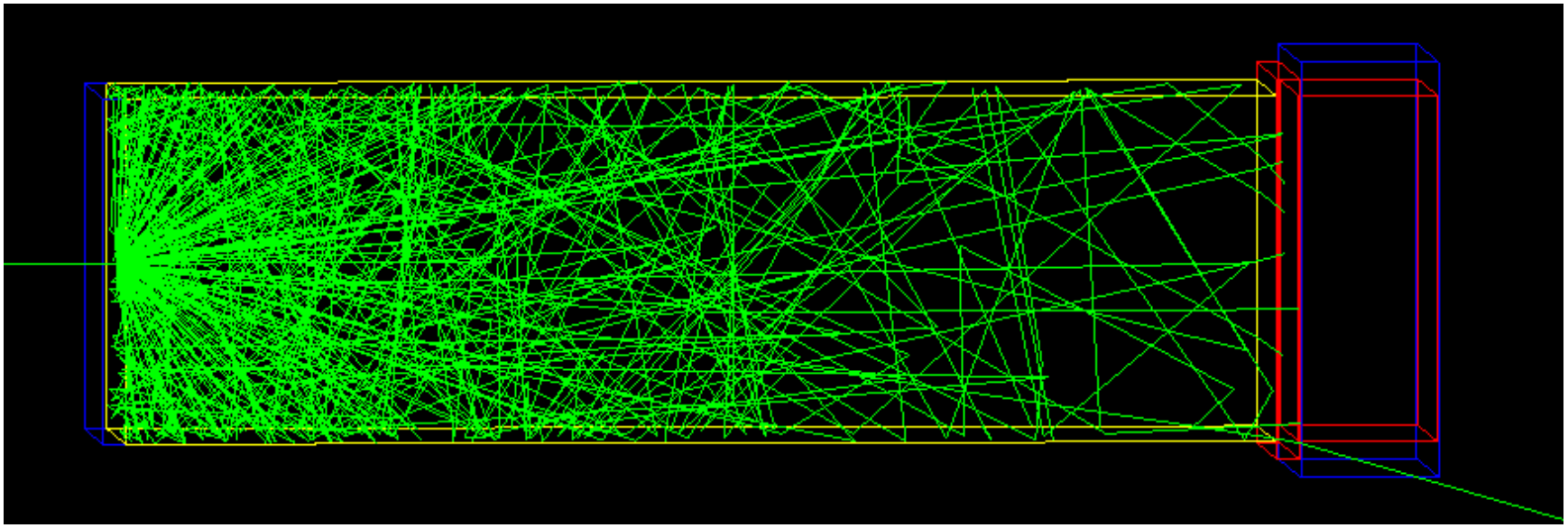
Not very clear nor intuitive

- I propose:

```
/gate/output/tree/hits/enable
/gate/output/tree/hits/trackLocalTime/disable
#for volumeID[0], volumeID[1], ...
/gate/output/tree/hits/volumeIDs/disable
/gate/output/tree/addCollection Singles
/gate/output/tree/Singles/comptVolName/disable
```

Optical simulation on GPU (1/2)

At CPPM, working on calorimetry:



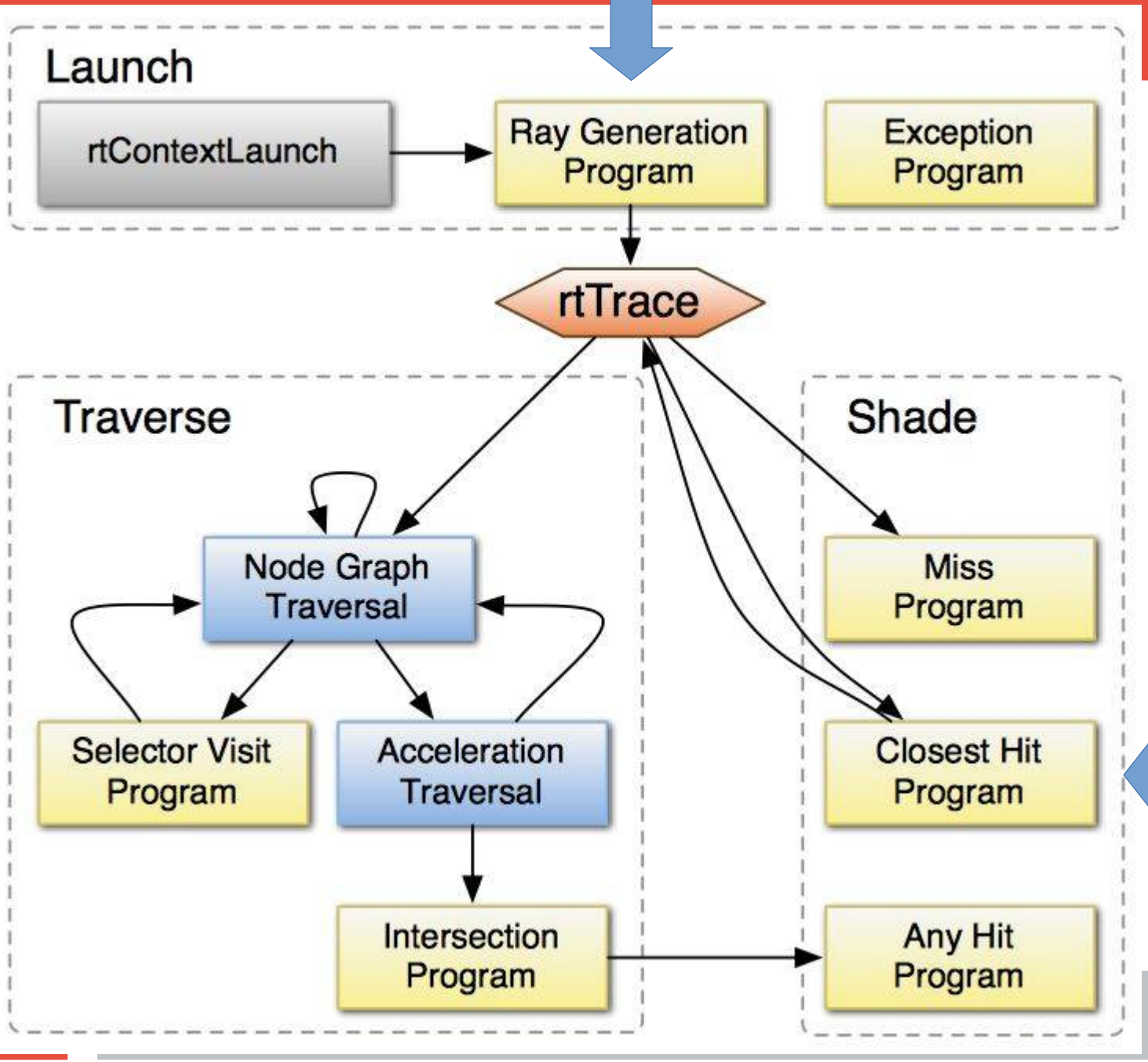
Lot of time spent in optical simulation. Can we use GPU to accelerate it ?

Optical simulation on GPU (2/2)

- **Problem :**
  - Hard to implement (with cuda) and hard to maintain (bad experience with cuda in GATE)
- **Solution :**
  - Use library : NVIDIA® OptiX™ Ray Tracing Engine
  - Not a new idea : *Opticks : GPU Optical Photon Simulation for Particle Physics using NVIDIA® OptiX™* For experiment Junon
    - But optiks very difficult to use and seems to be very coupled to application
- **Here we experiment (M1 internship) OptiX for a very simple task : Scintillation in boxes**
- **OptiX does:**
  - Raytracing
  - Parallelization on GPU

OptiX

To implement : scintillation



To implement : refraction, reflection, etc

Work in progress

- **Currently trying to reproduce Geant4 Unified Model**
- **Geometry accepted : only boxes (for now)**

- **Plan for GATE :**
  - Develop independant tool
  - Read hits file and make scintillation/optical photons tracking in GPU

Thanks !

How works register_variable ?

With `typeid` function and `type_index`:

```
m_tmapOfDefinition[typeid(double)] = get_endianness_carater() + "f8";
m_tmapOfDefinition[typeid(float)] = get_endianness_carater() + "f4";
m_tmapOfDefinition[typeid(uint8_t)] = get_endianness_carater() + "u1";
m_tmapOfDefinition[typeid(uint16_t)] = get_endianness_carater() + "u2";
m_tmapOfDefinition[typeid(uint32_t)] = get_endianness_carater() + "u4";
m_tmapOfDefinition[typeid(uint64_t)] = get_endianness_carater() + "u8";
m_tmapOfDefinition[typeid(int8_t)] = get_endianness_carater() + "i1";
m_tmapOfDefinition[typeid(int16_t)] = get_endianness_carater() + "i2";
m_tmapOfDefinition[typeid(int32_t)] = get_endianness_carater() + "i4";
m_tmapOfDefinition[typeid(int32_t)] = get_endianness_carater() + "i4";
m_tmapOfDefinition[typeid(int64_t)] = get_endianness_carater() + "i8";

m_tmapOfDefinition[typeid(bool)] = get_endianness_carater() + "?";
m_tmapOfDefinition[typeid(char)] = get_endianness_carater() + "b";
```

For ROOT :

```
m_tmapOfDefinition[typeid(Char_t)] =  "B";
m_tmapOfDefinition[typeid(UChar_t)] ="b";
m_tmapOfDefinition[typeid(Short_t)] = "S";
m_tmapOfDefinition[typeid(UShort_t)] = "s";
```