TPC Flash ADC Simulation/Reconstruction
In Marlin and LCIO

Jim Hunt
and Dan Peterson

Cornell University

LCWS: May 2007

LEPP
LABORATORY FOR ELEMENTARY-PARTICLE PHYSICS
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit

SimTrackerHit

SimTrackerHit
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit → SimTrackerHit → SimTrackerHit

? → ? → ?

Reconstructed Hit → Reconstructed Hit → Reconstructed Hit

Question: Can We Directly Make Reconstructed Hits?
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit

SimTrackerHit

SimTrackerHit

No! TPCs Have Pads!

TPC Hits are not one-to-one with Reconstructed Hits.
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit
SimTrackerHit
SimTrackerHit

No! TPCs Have Pads!

TPC Hits are not one-to-one with Reconstructed Hits.

(Silicon Hits are closer to one-to-one with Reconstructed Hits)
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit

SimTrackerHit

SimTrackerHit

Pad

Pad

Pad

Pad
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit → SimTrackerHit → SimTrackerHit

Pad → Pad → Pad → Pad
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit → Pad
SimTrackerHit → Pad
SimTrackerHit → Pad

Pad → Reconstructed Hit
Pad → Reconstructed Hit
Pad → Reconstructed Hit

Jim Hunt (Cornell University)
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit

Pad

Reconstructed Hit

Jim Hunt (Cornell University)

TPC Flash ADC Sim/Reco

LCWS: May 2007
What Are We Trying To Do?

Charged Particle Track (From Mokka)

SimTrackerHit → SimTrackerHit → SimTrackerHit

Pad → Pad → Pad → Pad

Reconstructed Hit → Reconstructed Hit → Reconstructed Hit

Goal: Simulate this structure in Marlin.
Custom LCIO Objects: IonizationCenters and ChargedPads

Charged Particle Track (From Mokka)

SimTrackerHit
SimTrackerHit
SimTrackerHit
New Custom LCIO Object: IonizationCenter
Custom LCIO Objects: IonizationCenters and ChargedPads

Charged Particle Track (From Mokka)

SimTrackerHit → IonizationCenter → ChargedPad
SimTrackerHit → IonizationCenter → ChargedPad
SimTrackerHit → IonizationCenter → ChargedPad

New Custom LCIO Object: ChargedPad
Custom LCIO Objects: IonizationCenters and ChargedPads

IonizationCenters are One-To-One with SimTrackerHits
IonizationCenters hold the One-To-Many Relationship.
ChargedPads hold the Many-To-One Relationship.
An IonizationCenter points back to the lcio::SimTrackerHit and holds an array of the ordered pairs (pad_number, charge).

class IonizationCenter : public CustomLCIOObject {
    ...
    class Charge {
        ...
        PadNumber pad_number;
        double charge;
    };
    fast_data_structures::array<Charge> m_charges;
    SimTrackerHit m_sim_tracker_hit;
}
A ChargedPad points to a PadGeometry::Pad (see later slides), holds Flash ADC Values and holds a list of IonizationCenters.

class ChargedPad : public CustomLCIOObject{
    ...
    fast_data_structures::sllist<IonizationCenter*> m_ionization_centers;
    fast_data_structures::array<unsigned int> m_adc_values;
    PadGeometry::Pad* m_pad;
}
The Algorithm for creating IonizationCenters and ChargedPads in pseudo-code is:

```python
for sim_hit in SimTrackerHits:
    ion = new IonizationCenter
    ion.set_sim_tracker_hit(sim_hit)
    ion.generate the (pad_number,charge) ordered pairs

for pad in Pads:
    charged_pad = new ChargedPad

for ion in IonizationCenters:
    for pairs in ion:
        charged_pad = get_charged_pad(pair.pad_number)
        charged_pad.add_ion(ion)
```
The user controls how charge is placed on pads with an

```cpp
class IonizationCenterChargeDistribution {
    virtual double get_max_xy_radius(Vector &ionization_center)=0;
    virtual double get_relative_charge(Vector &ionization_center,
                                         PadGeometry::Pad* pad )=0;
    virtual double get_total_charge()=0;
};
```
The user controls how charge is placed on pads with an

```cpp
class IonizationCenterChargeDistribution {
    virtual double get_max_xy_radius(Vector &ionization_center)=0;
    virtual double get_relative_charge(Vector &ionization_center,
                                           PadGeometry::Pad* pad )=0;
    virtual double get_total_charge()=0;
};
```

The Algorithm in pseudo-code for each ion:

```plaintext
dist = get_user_defined_charged_distribution()
xy_radius = dist.get_max_xy_radius(ion.get_xy())
pads = get_pads_near(ion.get_xy(), xy_radius)
total_charge = 0
for pad in pads
    charge = dist.get_relative_charge(ion.get_xy(), pad)
    total_charge += charge
    pad.charge = charge
scale_correction = dist.get_total_charge()/total_charge
for pad in pads
    pad.charge *= scale_correction
```
 IonizationCenter charge vs time.

Below is an example of a highly populated pad (pad number 27584 from a Kaon event).
**ChargedPadViewer (charge on one pad)**

**Generated Flash ADC values vs time**

(generated dynamically by looping over the ChargedPad’s IonizationCenters and putting in an exponential decay)
For the rest of the talk I’m going to show tools that allow the user to inspect events. The point is to figure out what made the time structure for this pad. All plots that follow are from this event.
We could not use

The GEometry Api for Reconstruction package

for Pad Geometry because it lacked the crucial
get_pads_near(xy,xy_radius) method.¹

¹the closest match was PadRowLayout2D::getNearestPad
A New PadGeometry

We could not use

The GEometry Api for Reconstruction package

for Pad Geometry because it lacked the crucial get_pads_near(xy,xy_radius) method.\(^1\)

So instead I wrote a new PadGeometry package

- that is based on arbitrarily placed convex polygons,
- and optimized for speed by using aggressively inlined custom data structures.

\(^1\)the closest match was PadRowLayout2D::getNearestPad
Here’s our 2D event display:
PadGeometry Details

With Tracks From an Event (SimTrackerHits are red):

![TPC Pad Viewer](image)

- `left = -1516`
- `right = 1516`
- `bottom = -1516`
- `top = 1516`
- `x=-1516 y=1516`
- `current_pad = 0`
- `current_pad_block = 0`
The intensity of the yellow color scales with integrated charge on each pad:
PadGeometry Details

Now zoom in:

```
left  = 918.789
right = 1234.3
bottom = 5455979
top   = 320.973
x=1099.98  y=320.973
current_pad = 35631
current_pad_block = 8658
```
With integrated charged coloring:

```
left  = 918.789
right = 1234.3
bottom = 5.45979
top   = 820.978
x=1092.77 y=319.103
current_pad = 35631
current_pad_block = 8658
```
How does this package work? For example, suppose you want to project charge from and IonizationCenter at $z = 0$ to the position of the mouse:
PadGeometry Details

Jim Hunt (Cornell University)

TPC Flash ADC Sim/Reco

LCWS: May 2007
Pad Geometry Details

Jim Hunt (Cornell University)

TPC Flash ADC Sim/Reco

LCWS: May 2007
A simpler example is how the package finds the pad at some xy-location:
It has an xy-grid index:
PadGeometry Details

And scans pads within one grid spacing:

![TPC Pad Viewer](image)

- left = 913.789
- right = 1234.3
- bottom = 5.45979
- top = 320.973
- x = 1232.95, y = 132.431
- current pad = 37784
- current pad block = 8759

Jim Hunt (Cornell University)  TPC Flash ADC Sim/Reco  LCWS: May 2007
Here's the pad whose flash adc was displayed earlier:
A Transition...

That was a new 2D EventDisplay.

But for good diagnostics we really need 3D ...
Run the new tpc python program:

~/tpc_tracking> tpc

A TPC Tracking Environment

http://www.lns.cornell.edu/~jmh263/tpc

tpc version: trunk (svnversion: 195M)

select an lcio file from the list:

[1] kaon_example.slcio

enter # 1
opening ~/tpc_tracking/data_files/kaon_example.slcio ...
executing ~/tpc_tracking/data_files/kaon_example.py ...

tpc python>
tpc python> e=tpc.create_event_display()
The 3D Event Display
tpc python> tpc.event.read_lcevent_from_file()
The 3D EventDisplay

Now tracks with labels are visible:
Put charge on the pads:

```python
tpc python> tpc.event.ionization_centers.create()
tpc python> tpc.event.ionization_centers.generate_charge_information()
tpc python> tpc.event.charged_pads.create()
tpc python> tpc.event.ionization_centers.put_charge_on_pads()
tpc python> tpc.event.refresh Gui()
```
The 3D Event Display

Charge
The 3D Event Display

TPC Tracking Event Display

Window  View  Camera
Now highlight the Many-To-One Relationship between IonizationCenters and one Pad.

```
tpc python> e.set_current_charged_pad(
...  tpc.event.charged_pads.get_charged_pad(27584))
tpc python> tpc.event.refresh_gui()
```
The 3D Event Display

Many-To-One
Another example: highlight the IonizationCenters-To-ChargedPads relationships for one Track.

```
tpc python> positron=tpc.event.mc_particles.get_particle(7)
tpc python> track=tpc.event.sim_tracks.get_track(positron)
tpc python> e.set_current_sim_track(track)
tpc python> tpc.event.refresh_gui()
```
The 3D Event Display

Ionization Centers To Charged Pads (Many-To-Many) for one Track
And the One-To-Many relationship between One IonizationCenter and Many ChargedPads:

tpc python> tpc.event.ionization_centers.get_nionization_centers()
19104
tpc python> e.set_current_ionization_center(
... tpc.event.ionization_centers.get_ionization_center(19000))
tpc python> tpc.event.refresh_gui()
The 3D EventDisplay

One-To-Many.

TPC Tracking Event Display
The 3D EventDisplay

other views

TPC Tracking Event Display

Y axis

Z axis
The 3D Event Display

other views

TPC Tracking Event Display

Window View Camera
So what’s in the event?

```python
tpc python> tpc.event.mc_particles.get_particle(1).print_decays(True)
K+  ->  pi0  nu_mu  mu+  e-  e-
   pi0  ->  gamma  gamma
   mu+  ->  e+  nu_e  nu_mu~
   e+  ->  gamma
```
So what’s in the event?

```
tpc python> tpc.event.mc_particles.get_particle(1).print_decays(True)
K+  ->  pi0 nu_mu mu+ e- e-
pi0  ->  gamma gamma
mu+  ->  e+ nu_e nu_mu~
e+   ->  gamma
```
So what’s in the event?

```python
tpc python> tpc.event.mc_particles.get_particle(1).print_decays(True)
K+    -> pi0 nu_mu mu+ e- e-
pi0   -> gamma gamma
mu+   -> e+ nu_e nu_mu~
e+    -> gamma
```
Here is our working example of a Marlin Processor.
CustomMarlinProcessor is a thin wrapper around marlin::Processor.

class PutChargeOnPadsProcessor :
    public tpc_tracking::CustomMarlinProcessor {

        CUSTOM_MARLIN_PROCESSOR_IMPL(PutChargeOnPadsProcessor)

    public:

        void processEvent(lcio::LCEvent *lc_event);
        // etc...

};
The user no longer has to supply a newProcessor() call.
Event::Sync synchronizes lcio::LCEvent with tpc_tracking::Event.

```cpp
void PutChargeOnPadsProcessor::processEvent(lcio::LCEvent *lc_event) {
    using tpc_tracking::Event;
    using tpc_tracking::globals::event;

    Event::Sync sync(lc_event);

    event->ionization_centers.create();
    event->charged_pads.create();
    event->ionization_centers.generate_charge_information();
    event->ionization_centers.put_charge_on_pads();
}
```
We now have

- a new Event Display (3D and 2D),
- a new PadGeometry package,
- a mechanism to add custom objects to LCIO\(^2\)
- marlin processors that move charge from IonizationCenters to ChargedPads,
- and a python interface that glues everything together.

\(^2\)for details see http://www.lns.cornell.edu/~jmh263/tpc
Conclusions

Future plans:

- Allow the user to control the amplifier characteristics (Polymorphism).
- Implement Flash ADC Time Reconstruction and store the result in LCIO (Probably a CustomLCIOObject).
- Plan how to integrate this project into the Marlin/MarlinTPC/LCIO/GEAR code base.
For Details, Tutorials, and More visit

http://www.lns.cornell.edu/~jmh263/tpc
Extra Slides
tpc.event is a wrapper around lcio::LCEvent.

namespace tpc_tracking {
    class Event {

        lcio::LCEvent *m_lc_current_event;
    };
}
- tpc.event.*'s objects represent Collections.
- the collections hold objects that may wrap lcio objects.

```cpp
namespace tpc_tracking {
    class MCParticle; // lcio::MCParticle
    class SimTrackerHit; // lcio::SimTrackerHit
    class SimTrack; // [ uses tpc_tracking::SimTrackerHit ]
    class IonizationCenters; // custom lcio object
    class ChargedPads; // custom lcio object

    class Event {
        MCParticles mc_particles
        SimTrackerHits sim_tracker_hits
        SimTracks sim_tracks;
        IonizationCenters ionization_centers;
        ChargePads charged_pads;
    };
};
```
light-weight object:

namespace tpc_tracking {
    class MCParticle {

        lcio::MCParticle* m_pMCParticle;
    };
}

heavy-weight object:

namespace tpc_tracking {
    class SimTrack {

        std::vector<SimTrackerHitNumber> m_sim_tracker_hits;
    };
}
**EventObject and CustomLCIOCollection<T>**

- **EventObject**
  - SimTracks
  - CustomLCIOCollection<T>
  - MCParticles
  - SimTrackerHits
  - IonizationCenters
  - ChargedPads
namespace tpc_tracking {

    // LCIO Collections:

class EventObject;
class SimTracks : public EventObject;

template<typename T>
class CustomLCIOCollection<T> : public EventObject;

class MCParticles : public CustomLCIOCollection<MCParticle>;
class SimTrackerHits : public CustomLCIOCollection<SimTrackerHit>;
class IonizationCenters:public CustomLCIOCollection<IonizationCenter>;
class ChargedPads : public CustomLCIOCollection<ChargedPad>;
};
CustomLCIOObject: Icio::SimTrackerHits details

```
lcrtre::LCRTRelations
```

```
LCEEVENT::LCOBJECT
```

```
EVENT::SimTrackerHit
```

```
IMPL::SimTrackerHitImpl
```

```
IOIMPL::SimTrackerHitIOImpl
```

```
IMPL::AccessChecked
```

```
SIO::SIOObjectHandler
```

```
SIO::SIOSimTrackerHitHandler
```
The preprocessor generates an SIOObjectHandler.
SIO is hidden in CustomLCIOObject::Stream.

namespace tpc_tracking {
  class IonizationCenter : public CustomLCIOObject {

    bool read(Stream &stream);
    bool write(Stream &stream);
  };

  DECLARE_CUSTOM_LCIO_OBJECT(IonizationCenter);
};
The preprocessor generates an SIOObjectHandler. SIO is hidden in CustomLCIOObject::Stream.

```cpp
namespace tpc_tracking {
    DEFINE_CUSTOM_LCIO_OBJECT(IonizationCenter);
    bool IonizationCenter::read(Stream &stream) {
        stream >> m_sim_tracker_hit;
        stream >> m_bGeneratedChargeInformation;
        if (m_bGeneratedChargeInformation) {
            stream >> m_charges;
        }
        return true;
    }
    bool IonizationCenter::write(Stream &stream) {
        stream << m_sim_tracker_hit;
        stream << m_bGeneratedChargeInformation;
        if (m_bGeneratedChargeInformation) {
            stream << m_charges;
        }
        return true;
    }
};
```
What happened to SIO_PTR and SIO_PTAG?

unsigned int CustomLCIOObject::read(SIO_stream* sio_stream)
{
    Stream stream(sio_stream);
    stream. declare_pointer_target(this) ;
    if (read(stream))
        return SIO_BLOCK_SUCCESS;
    else
        return SIO_BLOCK_NOTFOUND; // see FAQ in SIO manual;
}

unsigned int CustomLCIOObject::write(SIO_stream* sio_stream)
{
    Stream stream(sio_stream);
    stream. declare_pointer_target(this);
    if (write(stream))
        return SIO_BLOCK_SUCCESS;
    else
        return SIO_BLOCK_NOTFOUND; // see FAQ in SIO manual;
}
As explained earlier, I did not use Gear.

General Settings are stored in a `Settings` object.

They are initialized at start-up; remember `kaon_example.py`:

```python
tpc.settings.set_tpc_sim_tracker_hits_collection_name('tpc07_TPC')
tpc.settings.set_tpc_half_length(2037.5);
tpc.settings.set_tpc_min_radius(371);
tpc.settings.set_tpc_max_radius(1516);

tpc.settings.set_time_start(0.25);
tpc.settings.set_time_per_bin(0.25);

tpc.settings.set_nbins(1000);
...

tpc.pad_geometry.create_circular_pattern2(10,10,0.9)

tpc.settings.
create_diffusion_ionization_center_charge_distribution(10000,20,30)
```