Imperial College London

Current status of Charge Analysis in HPTPC

Edward Atkin

Imperial College London

Introduction

What's in a raw tpc file?

```
*.....................................................................*
Br    3 :scopeData : vector<Waveform*>                                *
Entries :        5 : Total  Size=        648 bytes  File Size  =        122 *
Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.26     *
*.....................................................................*
```

- Each channel is read by the digitiser
- Channels consist of 3 anodes + beam trigger
- When a channel goes triggers all channels saved
- Written as a vector<Dmtpc::core::Waveform*> in the raw files
- i.e. get a vector of waveforms for each dmtpc event (exposure/bias in essence)

Imperial College London

The gist of it...

- The basics of what the charge analysis currently looks like is very simple

```
 1 rawfile->open()
 2
 3 for(loop over events){
 4
 5         rawfile->getevent()
 6
 7         for(all waveforms in event){
 8
 9                 waveform = event->get_waveform()
10
11                 waveform = calculate_baseline()
12
13                 analysed_waveform = waveform->analyse()
14
15                 peak_height = analysed_waveform->get_peak_height()
16
17                 peak_height->write();
18
19         }
20 }
```

Imperial College London

The gist of it...

- Immediately can build on this with slightly more complexity
- Clean events + some noise cuts + <insert function>

```
rawfile->open()

for(loop over events){

        rawfile->getevent()

        for(all waveforms in event){

                waveform = event->get_waveform()

                waveform = calculate_baseline()

                waveform_clean = waveform->clean()

                analysed_waveform = waveform_clean->analyse()

                peak_height = analysed_waveform->get_peak_height()

                if(peak_height passes){

                        peak_height->write();
                }
        }
}
```

Imperial College London

The Basic Function

- The actual basic code is not much more complicated!

```
for(int i = 0; i < nevents ; i++){
        d->getEvent(i);

        for (int scp=0; scp < d->event()->nScopeData() ; scp++){

                // Channel 4 is anode 3
                if (d->event()->getTrace(scp)->getInfo()->channel == 4){

                        // Create some variables to set
                        double TraceRMS;
                        double gaus_std = 5;

                        //Make a trace object
                        TH1D* myTrace = (TH1D*)d->event()->getTrace(scp)->getPhysical();

                        // Calculate the baseline for a range of bins
                        baselineValue = dmtpc::waveform::analysis::baseline(myTrace,TraceRMS, 1,2000);

                        // Make CSP waveform
                        dmtpc::waveform::CspWaveform cspwf;
                        dmtpc::waveform::analysis::analyzeCSP( myTrace , cspwf , gaus_std);

                        // Look at the first entry of cspwf
                        dmtpc::waveform::CspPulse pulse = cspwf.at(0);

                        peak = pulse.getPeak();

                        if(peak>2*TraceRMS){
                                hpeak->Fill(peak);
                        }
                }
        }
}
```

Imperial College London

The Basic Function

- There are three key parts: baseline calculation, CspWaveform object, CspWaveform analysis

```
1  for(int i = 0; i < nevents ; i++){
2          d->getEvent(i);
3
4          for (int scp=0; scp < d->event()->nScopeData() ; scp++){
5
6                  // Channel 4 is anode 3
7                  if (d->event()->getTrace(scp)->getInfo()->channel == 4){
8
9                          // Create some variables to set
10                         double TraceRMS;
11                         double gaus_std = 5;
12
13                         //Make a trace object
14                         TH1D* myTrace = (TH1D*)d->event()->getTrace(scp)->getPhysical();
15
16                         // Calculate the baseline for a range of bins
17                         baselineValue = dmtpc::waveform::analysis::baseline(myTrace,TraceRMS, 1,2000);
18
19                         // Make CSP waveform
20                         dmtpc::waveform::CspWaveform cspwf;
21                         dmtpc::waveform::analysis::analyzeCSP( myTrace , cspwf , gaus_std);
22
23                         // Look at the first entry of cspwf
24                         dmtpc::waveform::CspPulse pulse = cspwf.at(0);
25
26                         peak = pulse.getPeak();
27
28                         if(peak>2*TraceRMS){
29                                 hpeak->Fill(peak);
30                         }
31                 }
32         }
33 }
```

Imperial College London

Baseline Calculation

- Nice and simple!
- Loop through histogram
- Get values for each bin in range → calculate RMS

```
21 double
22 dmtpc::waveform::analysis::baseline(const TH1* hist, double& rms, int binMin, int binMax)
23 {
24
25   double base=0;
26   rms=0;
27   int n = binMax-binMin+1;
28   for (int i = binMin; i<=binMax; i++)
29   {
30     double v = hget(hist,i);
31     base+=v;
32     rms+=v*v;
33   }
34
35   base/=n;
36   rms/=n;
37   rms -= base*base;
38   rms = sqrt(rms);
39   return base;
40
41 }
```

Imperial College London

AnalyseCSP function

- More complicated… (too long to put on slide!)
- The basic idea:
  - Smooths waveform with a gaussian kernel
  - Calculates baseline (see earlier)
  - Sets various members of CspWaveform (baseline, rms, max time, min time)
  - Finds bin with maximum value. Bin = peak time, value = peak voltage
  - Finds voltages and times at {0%, 10%, 25%, 5%, 75% and 90%}*peak + baseline
  - Calculate rise time and fall time
  - Set various CspPulse members (peak voltage, peak time, start time, end time , integral and “fractional” voltages and times)

Conclusion

- Analysis is pretty simple! (for now…)
- Key part of the analysis is contained in dmtpc::waveform::analysis::analyzeCSP
- Key parameters needed for gain calculation: peak value, peak time, baseline
- Other useful parameters: rise time, fall time, integral
- The DMTPC objects most relevant to this are Waveform, CspWaveform, CspPulse (and Pulse)

Imperial College London

BACKUPS

Imperial College London

Dmtpc::core::waveform


```
#ifndef DMTPC_CORE_WAVEFORM_HH
#define DMTPC_CORE_WAVEFORM_HH

#include "ScopeInfo.hh"
#include "TObject.h"
#include <stdint.h>
#include "TH1.h"

namespace dmtpc
{
  namespace core
  {
    class Waveform : public TObject
    {

      public:

        Waveform();
        virtual ~Waveform();
        Waveform(const char * name, const char * title, const void * raw_data, const ScopeChannelInfo * info,  uint32_t secs, uint32_t nsecs);

        uint32_t GetBinContent(int i) const;
        double GetPhysicalBinContent(int i) const;

        // seamlessly convert to TH1
        operator TH1*() { return physical(); }  //non-const
        operator TH1() { return *physical(); }  //non-const
    //    operator const *TH1() const { return getPhysical(); }  //const
    //    operator const TH1() const { return *getPhysical(); }  //const

        void Draw(const char * opt = 0) const { physical()->Draw(opt); }
        const TH1D* getPhysical() const { return (const TH1D*) physical(); }
        const TH1* getRaw() const { return data; }
        void destroyPhysical() const;
        char getType() const { return type; }
        const ScopeTraceInfo * getInfo() const { return &trace_info; }


      protected:
        //protected since non-const return
        TH1D* physical() const;
        TH1* data;
        mutable TH1D * physical_data; //!
        ScopeTraceInfo trace_info;
        double vmin;
        double vmax;
        char type; //C,S,I;
        char nbits;

        ClassDef(Waveform,1);
    };
  }
}

#endif
```

Imperial College London

Waveform


```
11  namespace core
12  {
13    class Waveform : public TObject
14    {
15
16      public:
17
18        Waveform();
19        virtual ~Waveform();
20        Waveform(const char * name, const char * title, const void * raw_data, const ScopeChannelInfo * info,  uint32_t secs, uint32_t nsecs);
21
22        uint32_t GetBinContent(int i) const;
23        double GetPhysicalBinContent(int i) const;
24
25        // seamlessly convert to TH1
26        operator TH1*() { return physical(); }  //non-const
27        operator TH1() { return *physical(); }  //non-const
28    //    operator const *TH1() const { return getPhysical(); }  //const
29    //    operator const TH1() const { return *getPhysical(); }  //const
30
31        void Draw(const char * opt = 0) const { physical()->Draw(opt); }
32        const TH1D* getPhysical() const { return (const TH1D*) physical(); }
33        const TH1* getRaw() const { return data; }
34        void destroyPhysical() const;
35        char getType() const { return type; }
36        const ScopeTraceInfo * getInfo() const { return &trace_info; }
37
38
39      protected:
40        //protected since non-const return
41        TH1D* physical() const;
42        TH1* data;
43        mutable TH1D * physical_data; //!
44        ScopeTraceInfo trace_info;
45        double vmin;
46        double vmax;
47        char type; //C,S,I;
48        char nbits;
49
50        ClassDef(Waveform,1);
51    };
52  }
53 }
```

Imperial College London

Waveform constructor


```
28 dmtpc::core::Waveform::Waveform(const char * name, const char * title, const void * raw_data, const ScopeChannelInfo * info, uint32_t secs, uint32_t nsecs)
29 {
30
31
32   nbits = info->nbits;
33   vmin = info->vmin;
34   vmax = info->vmax;
35
36   double tmin = -double(info->nsamples_pretrigger)/info->sample_rate*1e6; //ms->ns
37   double tmax = double(info->nsamples - info->nsamples_pretrigger - 1)/info->sample_rate*1e6;
38
39   if (info->nbits <= 8)
40   {
41     type = 'C';
42     data = new TH1C(name,title, info->nsamples, tmin, tmax);
43
44     if (info->nbytes_raw == 1)
45     {
46       memcpy(&(((TH1C*)data)->fArray[1]), raw_data, info->nsamples);
47     }
48
49     else if (info->nbytes_raw == 2)
50     {
51       const uint16_t * short_data = (const uint16_t *) raw_data;
52       for(size_t i = 0; i< info->nsamples;i++) { ((TH1C*)data)->fArray[i+1] = (uint8_t) short_data[i]; }
53     }
54     else if (info->nbytes_raw == 4)
55     {
56         const uint32_t * int_data = (const uint32_t *) raw_data;
57         for(size_t i = 0; i< info->nsamples;i++) { ((TH1C*)data)->fArray[i+1] = (uint8_t) int_data[i]; }
58     }
59     else unsupported(info);
60   }
```

Imperial College London

CspWaveform

- Private members

```
64   protected:
65     int N;
66     std::vector<CspPulse> pulse;
67     uint32_t secs;
68     uint32_t nsecs;
69     double base;
70     double rms;
71     double wfMax;
72     double wfMaxTime;
73     int wfMaxBin;
74     double wfMin;
75     double wfMinTime;
76     int wfMinBin;
77
78     ClassDef(CspWaveform,1)
79 };
```

Imperial College London

CspPulse


```
18   public:
19
20     /** Default constructor.  Sets rise and fall time variables to -1,
21     all else to 0.
22     */
23     CspPulse() :
24       Pulse(),
25       R0(-1),R10(-1),R25(-1),R50(-1),R75(-1),R90(-1),
26       F0(-1),F10(-1),F25(-1),F50(-1),F75(-1),F90(-1){;}
27
28
29     CspPulse(int nb) : Pulse(nb),
30       R0(-1),R10(-1),R25(-1),R50(-1),R75(-1),R90(-1),
31       F0(-1),F10(-1),F25(-1),F50(-1),F75(-1),F90(-1){;}
32
33     CspPulse(const Pulse& p) : Pulse(p),
34       R0(-1),R10(-1),R25(-1),R50(-1),R75(-1),R90(-1),
35       F0(-1),F10(-1),F25(-1),F50(-1),F75(-1),F90(-1){;}
36
```

Imperial College London

Pulse


```
 94      protected:
 95
 96        bool init;
 97
 98        int nbin;
 99        double peak;
100        double peakTime;
101        double startTime;
102        double startBin;
103        double endTime;
104        double endBin;
105        double integral;
106
107      ClassDef(Pulse,1);
```

Imperial College London

GetRise, getFall


```
129 void
130 dmtpc::waveform::analysis::riseTime(const TH1* hist, const vector<double>& list,
131                   vector<double>& values, double startTime, double endTime,
132                   bool fromStart)
133 {
134
135   double binWidth = hist->GetXaxis()->GetBinWidth(1);
136   int n = list.size();
137
138   values.clear();
139   values.resize(n,-1);
140
141   int startBin = hist->GetXaxis()->FindBin(startTime);
142   int endBin = hist->GetXaxis()->FindBin(endTime);
143   startBin = startBin==1?startBin:startBin-1;
144   double v;
145   if (fromStart){
146     double lastV=hget(hist,startBin);
147     for (int i = startBin; i<=endBin ;i++)
148     {
149       v = hget(hist, i);
150
151       for (int j = 0; j<n ; j++){
152
153         if (values[j]==-1&&lastV<list[j]&&v>=list[j]){
154           double binFrac = (list[j] - lastV) / (v-lastV);
155           values[j] = binFrac*binWidth + hcenter(hist,i-1);
156
157           if (j==n-1) break;
158
159         }//if reaches value
160
161
162       }//values
163       lastV=v;
164     }//bins
165   }else{
166     double lastV=hget(hist,endBin);
167     for (int i = endBin; i>=startBin; i--)
168     {
169        v = hget(hist,i);
170        for (int j=n-1; j>=0; j++){
171
172          if (values[j]==-1&&lastV>list[j]&&v<=list[j]){
173
174            double binFrac = (list[j]-lastV)/(v-lastV);
175            values[j] = hcenter(hist,i+1)-binFrac*binWidth;
176
177            if (j==0) break;
178          }
179
180
181        }//values
182        lastV=v;
183     }
184   }//which side to start at
185
186
187 }
188
```