EUDET-Memo-2008-31

A Modular TPC Endplate Description for GEAR

Martin Killenberg[*], Stephen Turnbull[†]

January 5, 2009

Abstract

We report on an extension of the `TPCParameters` class, which describes the end plate of a TPC in the **GE**ometry **A**PI for **R**econstruction toolkit (GEAR). The interface is now able do describe an end plate made up of several readout modules instead of only one monolithic pad plane.

[*]University of Bonn, Bonn, Germany
[†]Carleton University, Ottawa, ON, Canada

1 Introduction

Up to now the TPC description in GEAR [1] had one pad plane at the end plate of the TPC. For full detector simulations a circular geometry was used, being a full circle with inner and outer radius. For small R&D prototypes a rectangular layout with flexible pad sizes is available.

The large LC-TPC prototype, which has been set up in the context of EUDET, has a more realistic end plate design where the end cap is made up of several modules. Each module can have a different pad geometry. So GEAR had to be extended to be able to describe modules which are only sectors of a circle and the different modules being rotated and translated with respect to the global coordinate origin.

2 Design Considerations

In GEAR the TPC is described by the `TPCParameters` class. In the previous versions it contained the overall TPC properties like the length of the drift volume and a pointer to the pad layout. The pad layout is derived from the `PadRowLayout2D` base class, which allows to describe a layout with pads being arranged in rows. As it was the only pad plane it defined the global coordinate system. With the new modular approach each of the modules will have its own, local coordinate system.
There were two main requirements for the extension:

1. The code should be 100% backward compatible, so the existing programmes do not break.

2. All local coordinates of the modules are translated to the global coordinate system so the user code does not have to handle different coordinate systems and perform the coordinate transformations.

For this reason a new class called `TPCModule` was introduced. It is derived from the `PadRowLayout2D`, so it provides the same functionality as the pad layout implementations. It contains one of these implementations and knows how much the pad layout has been shifted and rotated against the global origin. The module's main functionality is to translate the pad layout's local coordinates to global coordinates. All other functionality concerning the arrangement of the pads is directly passed on to the pad layout implementation, so all the existing functionality is still available without having to be reimplemented.

3 The TPCModule

As already mentioned the `TPCModule` is derived from `PadRowLayout2D` to provide all its functionality and for backward compatibility. The user code does not have to care whether it directly uses an implementation of the pad layout, which only knows about one coordinate system, or if it uses a module. The module contains one of the palpable implementations (currently `RectangularPadRowLayout`, `FixedPadSizeDiskLayout` and `FixedPadAngleDiskLayout` are available).

Each module has a unique module ID. As the electronics in the prototype setup can vary from module to module, each module has its own readout frequency. The module can be rotated by an angle with respect to the global coordinate system. Its origin can by shifted with respect to the global origin, the offset is given in cartesian or polar coordinates, depending on the TPC's coordinate system. All these values can be queried using the respective `get` functions (see Table 1).

A border around the pad plane can be defined to enlarge the active area of the module. This is for instance needed for pad planes with resistive coating, where charge arriving on the resistive layer near the pads still produces a signal in the pads. Or in simulations, where the diffusion can make charges produced near the pad plane end up on one of the pads. `getModuleExtent()` returns this enlarged area, where `getDistanceToModule()` gets the closest distance to it and `isInsideModule()` tests whether the given global coordinates are within the module extent.

Although the interface usually provides only global coordinates, it sometimes can be useful to have access to the local information. Some calculations can easier be performed in local coordinates, and an event display has a better performance calculating the coordinate transformations on the GPU than on the main processor. For this reason the module allows access to the underlying pad layout (`getLocalPadLayout()`) and the transformation functions from global to local coordinates and vice versa (`globalToLocal()` and `localToGlobal()`). Also the extended module area can be retrieved in local coordinates using `getLocalModuleExtent()`.

The member functions derived from `PadRowLayout2D` can be divided into two sections. The first group directly queries the instance of the pad layout and returns its value. These are the geometry specific functions which do not contain coordinate information, like number of pads, shape of the pads etc.

The second group of derived functions handles coordinates and has to perform the coordinate transformations. All coordinates of the functions derived from `PadRowLayout2D` are global coordinates, which are translated to local coordinates and subsequently the required information is retrieved from the local pad layout. If the pad layout returns local coordinates they are tranformed into the global coordinate system before being passed on by the module.

A special case is the plane extent. It gives the maximum and minimum x- and y-coordinates (or r-φ-coordinates for polar coordinates) the module can occupy. This does not necessarily mean that the complete area is covered with pads but that there are no pads outside this area. For polar coordinate systems this is a sector of a circular ring, which is a good description of the geometry if the coordinate origin of the module is in the global origin. But if the module is rotated and translated, the global plane extent is no longer an adequate description of the module geometry. However, the plane extend limits the area in which the sensitive area can be. This is especially useful in simulation and digitisation because signals outside this area do not have to be processed at all, which improves performance. Figure 1 gives examples for a circular module in global polar and global cartesian coordinates.

TPCModule	
Module specific	
`int`	`getModuleID()`
`double`	`getReadoutFrequency()`
`Vector2D &`	`getOffset()`
`double`	`getAngle()`
`int`	`getTPCCoordinateType()`
`bool`	`isOverlapping()`
`double`	`getBorderWidth()`
`std::vector<double> &`	`getModuleExtent()`
`double`	`getDistanceToModule()`
`bool`	`isInsideModule()`
`Vector2D`	`globalToLocal(double c0, double c1)`
`Vector2D`	`localToGlobal(double c0, double c1)`
`PadRowLayout2D &`	`getLocalPadLayout()`
`std::vector<double> &`	`getLocalModuleExtent()`
Derived from `PadRowLAyout2D` (directly passed to instance)	
`int`	`getPadLayoutImplType()`
`int`	`getCoordinateType()`
`int`	`getPadShape()`
`int`	`getNPads()`
`int`	`getNRows()`
`double`	`getRowHeight(int rowNumber)`
`double`	`getPadWidth(int padIndex)`
`double`	`getPadHeight(int padIndex)`
`std::vector<int> &`	`getPadsInRow(int rowNumber)`
`int`	`getRowNumber(int padIndex)`
`int`	`getPadNumber(int padIndex)`
`int`	`getPadIndex(int rowNumber, int padNumber)`
`int`	`getRightNeigbour(int padIndex)`
`int`	`getLeftNeighbour(int padIndex)`
Derived from `PadRowLayout2D` (coordinate transformation)	
`Vector2D`	`getPadCenter(int padIndex)`
`std::vector<double> &`	`getPlaneExtent()`
`int`	`getNearestPad(double c0, double c1)`
`bool`	`isInsidePad(double c0, double c1, int padIndex)`
`bool`	`isInsidePad(double c0, double c1)`
`double`	`getDistanceToPad(double c0, double c1, int padIndex)`

Table 1: The member functions of TPCModule.

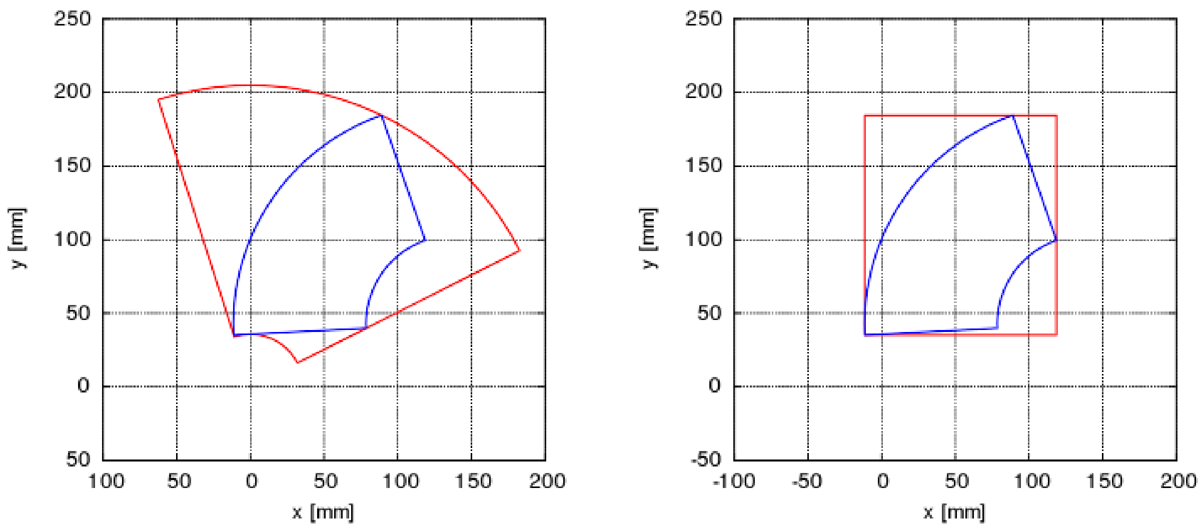

Figure 1: The global plane extents (red) of a wedge shaped module (blue) in global polar and global cartesian coordinates.

4 Changes in TPCParameters

`TPCParameters` now contains a vector with modules instead of a single pad layout. Its functionality has been extended to handle the modules. Table 2 shows the member functions of the `TPCParameters` class. `getMaxDriftLength()` and `getDriftVelocity()` work like in the previous version, they are global TPC parameters. Now there might be more than one pad layout, each with its own readout frequency. If there is only one module, `getPadLayout()` and `getReadoutFrequency()` return the module (which is a pad layout) and its readout frequency, respectively. As the module is returned it is ensured that only global coordinates are passed to the user code. In case there is more than one module an exception is thrown. This is to prevent the old code from producing inaccurate results by not taking into account all modules.

The TPC can have a cartesian or a polar coordinate system independently from the modules. This information can be retrieved with `getCoordinateType()`.

There is a block of member functions to handle the modules. One can either access the modules by their global module ID (`getModule(int ID)`), query the number of modules (`getNModules()`) or retrieve a `std::vector` with pointers to all modules (`getModules()`).

The `isInsideModule()`, `isInsidePad()`, `getNearestModule()`, `getNearestPad()`, and `getPlaneExtent()` functions are convenience functions which loop all modules and use their respective member functions to determine the requested information. As the pad index is only unique within one module, the `GlobalPadIndex` helper class has been introduced which contains the module ID and the local pad index.

TPCParameters	
Hitherto existing	
`double`	`getMaxDriftLength()`
`double`	`getDriftVelocity()`
`PadRowLayout2D &`	`getPadLayout()`
`double`	`getReadoutFrequency()`
New	
`int`	`getCoordinateType()`
`TPCModule &`	`getModule(int ID)`
`int`	`getNModules()`
`std::vector<TPCModule *> &`	`getModules()`
`bool`	`isInsideModule(double c1, double c2)`
`bool`	`isInsidePad(double c1, double c2)`
`TPCModule &`	`getNearestModule(double c1, double c2)`
`GlobalPadIndex`	`getNearestPad(double c1, double c2)`
`std::vector<double> &`	`getPlaneExtent()`

Table 2: The member functions of the `TPCParameters` class.

5 The XML Syntax

In order to describe the modules, the XML syntax had to be extended. Listing 1 shows a simple example with only one module. The `TPCParameters` detector type has the new tag <`coordinateType`> which can have the values `cartesian` or `polar` and describes the global coordinate system of the end plate.

The <`modules`> section takes `moduleIDStartCount` as an optional value. If it is given, module numbering starts with this value and is increased for every module, so the number does not have to be stated in every module. By explicitly specifying the module ID the default value can be overridden.

Each module has a separate <`module`> section. The <`readoutFrequency`> now is specified within the module instead of the detector itself. Afterwards a pad layout is defined. The module has the optional parameters of the <`angle`> the module is rotated with respect to the global coordinate system and the <`offset`> the local origin is shifted. `x_r` and `y_phi` are values in the global coordinate system. The active area can be extended using <`enlargeActiveAreaBy`>.

In case there are several similar modules which only vary in some of the parameters, one can define a <`default`> module as the first section within <`modules`>. All parameters that are common to all modules are specified in this section, only the individual parameters have to be specified for each module. This improves code readability and helps to avoid *copy and paste* errors. However, a default value can be overridden in each module. The usage of a default module is shown in the more verbose example in the appendix for eight identical sector modules which make up a full circle.

```
<gear>
  <detectors>

    <detector name="TPC" geartype="TPCParameters">

      <maxDriftLength  value="600.0" />
      <coordinateType  value="polar" />

      <modules  moduleIDStartCount="20">

        <module>

          <readoutFrequency value="2.0e+07" />

          <PadRowLayout2D type="FixedPadAngleDiskLayout"
                          rMin="60" rMax="150"
                          nRow="14"
                          phiMin="-0.3926990816987241548"
                          phiMax="0.8926990816987241548"
                          nPadsInRow="17">
          </PadRowLayout2D>

          <angle value="2.3" />
          <offset x_r="145" y_phi="0.3" />
          <enlargeActiveAreaBy value="1."/>

        </module>

      </modules>

    </detector>
  </detectors>
</gear>
```

Listing 1: A simple XML file describing one module.

The XML parser is 100% backward compatible with the old syntax where the pad layout is directly placed in the `TPCParameters` detector. In this case the parser creates a `TPCParameters` object with only one module. As described above this also is backward compatible so that no change in existing programme code is necessary. However, old and new XML syntax cannot be mixed.

6 Conclusion

The `TPCParameters` class has been extended to contain multiple modules. Each of the modules can have an offset to the global coordinate system and be rotated by an angle. The `TPCModule` contains an implementation of a pad layout and translates its local coordinates to the global coordinate system. As the `TPCModule` itself is an implementation of the abstract `PadRowLayout2D` class, it provides all the functionality of the hitherto direct usage of the pad layouts, is 100% backward compatible and the change is completely transparent for the user.

Acknowledgement

Thanks to Frank Gaede, who helped us to design the classes and integrate them consistently into the GEAR framework.

This work is partly supported by the Commission of the European Communities under the 6th Framework Programme "Structuring the European Research Area", contract number RII3-026126.

References

[1] The GEAR home page: http://ilcsoft.desy.de/portal/software_packages/gear/

A Example XML file

```
<gear>
  <!--
      Example XML file for GEAR describing a modular TPC.
      The end plate has eight wedge shaped modules, each being
      one eights of a full circle.
      In addition there are two rectangular modules within the
      circular ring.
    -->

  <detectors>

    <!-- The TPC has geartype set to TPCParameters -->
    <detector name="TPC" geartype="TPCParameters">

      <!-- First set the global parameters -->

      <!-- Drift length in mm -->
      <maxDriftLength  value="600.0" />
      <!-- Type pf coordinate system, "cartesian" or "polar" -->
      <coordinateType  value="cartesian" />

      <!-- The modules section
           The first module has ID 20, for all following
           modules the ID is increased by 1
        -->
      <modules moduleIDStartCount="20">

        <!-- First there is a default module.
             It contains all properties which are common
             for all modules
          -->
        <default>
          <!-- Each module has its own readout frequency -->
          <readoutFrequency value="2.0e+07" />

          <!-- There is a pad layout in each module -->
          <PadRowLayout2D type="FixedPadSizeDiskLayout"
                          rMin="386." rMax="1626."
                          padHeight="6." padWidth="2."
                          maxRow="206" padGap="0.">
      <!-- In this example phiMin corresponds to
           the angle of 1 mm  on a circle
           at a radius of 386. mm (rMin)
        -->
      <phiMin value="4.12318505419417984 4e-4"/>
      <!-- phi_max = pi/4 - phiMin -->
      <phiMax value="0.78498584489202886 1"/>
    </PadRowLayout2D>

    <!-- Enlarge the active area by 1 mm.
         The angle is enlarged such that the gap
         is 1 mm at rMin. Combinded with the
         values of phiMin and phiMax
         the Module is 1/8 of a full circle.
      -->
    <enlargeActiveAreaBy value="1."/>

  </default>

  <!-- The first module is not rotated. -->
  <module>
    <angle value="0." />
  </module>

  <!-- Rotate each of the following modules
       by 1/8 of a circle.
    -->
  <module>
    <angle value="0.785398163397448279" />
  </module>

  <module>
    <angle value="1.570796326794896558" />
  </module>

  <module>
    <angle value="2.356194490192344837" />
  </module>

  <module>
    <angle value="3.141592653589793116" />
  </module>

  <module>
    <angle value="3.926990816987241395" />
  </module>
```

```
<module>
  <angle value="4.71238898038468967 4" />
</module>

<module>
  <angle value="5.49778714378213795 3" />
</module>

<!-- The two rectangular modules overwrite the values
     set in the default module.
  -->

<module>
  <!-- Set a module number independent from
       the automatic numbering
    -->
  <moduleID value="100"/>

  <!-- Shift the origin -->
  <offset x_r="130.0" y_phi="0" />

  <!-- Do not rotate -->
  <angle value ="0" />

  <!-- Overwrite the default readout freuqency -->
  <readoutFrequency value="2.5e+07" />

  <!-- Define a rectangular pad plane -->
  <PadRowLayout2D  type="RectangularPadRowLayout"
                   xMin="-50." xMax="50."
                   yMin="-50." repeatRows="5">
    <row nPad="10"  leftOffset="4.5." padHeight="9.0"
         padWidth="9.0" rowHeight="10.0" />
    <row nPad="10" rightOffset="4.5." padHeight="9.0"
         padWidth="9.0" rowHeight="10.0" />
  </PadRowLayout2D>

  <!-- Enlarge the active area by 1 mm -->
  <enlargeActiveAreaBy value="1."/>
</module>
```

```
        <!-- Another modue with a different offset -->
        <module>
          <moduleID value="101"/>
          <offset x_r="-130.0" y_phi="0" />
          <angle value ="0" />
          <readoutFrequency value="4.0e+07" />

          <PadRowLayout2D  type="RectangularPadRowLayout"
                           xMin="-50." xMax="50."
                           yMin="-50." repeatRows="5">
            <row nPad="10"  leftOffset="4.5." padHeight="9.0"
                 padWidth="9.0" rowHeight="10.0" />
            <row nPad="10" rightOffset="4.5." padHeight="9.0"
                 padWidth="9.0" rowHeight="10.0" />
          </PadRowLayout2D>
          <enlargeActiveAreaBy value="1."/>
        </module>

      </modules>

    </detector>

  </detectors>

</gear>
```