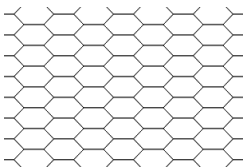
Developing new detector model geometries in Allpix^2

3rd Allpix^2 User Workshop

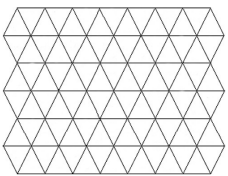
Radek Privara
Palacky University Olomouc
(radek.privara@cern.ch)

May 9–11, 2022

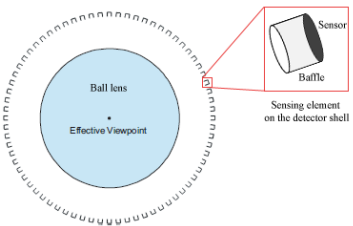
- Allpix^2 can currently simulate rectangular pixel detectors and radial strip detectors.
  - Hexagonal pixel detectors are being implemented as well, work-in-progress.
- Other detector geometries possible by implementing a new `DetectorModel` class.
  - Complete implementation in one class, almost no need to touch the rest of the framework.
  - Contribute to expand the framework's usability.
  - Other people can base their implementations on yours.



Hexagonal pixel detector.



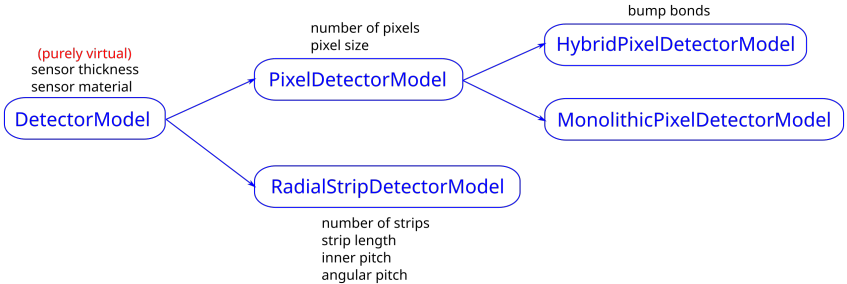
Triangular pixel detector.



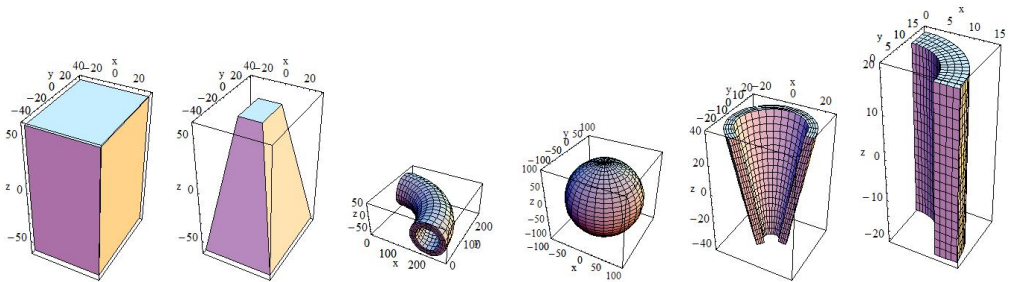
CAVE Spherical camera.

Detector model classes

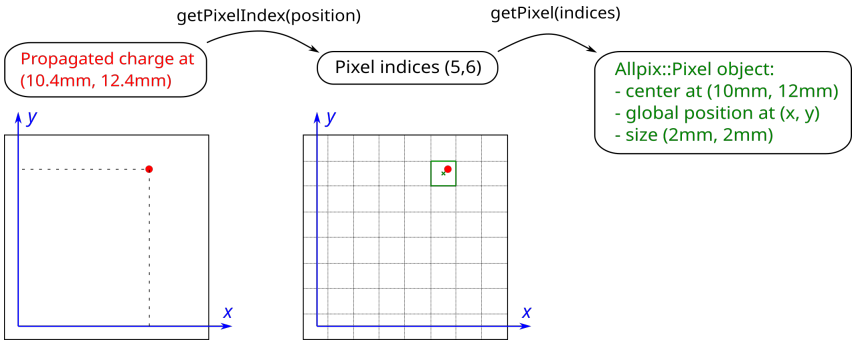
- When creating your detector model class, base it on the existing ones.
  - Choose depending on what parameters you need.
  - If no derived class works for you, inherit from the base `DetectorModel` class.
- Define mandatory and optional parameters to construct your detector model.



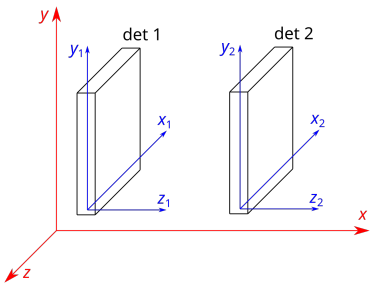
- Allpix^2 uses the Geant4 toolkit for detector creation, passage of particles, energy deposition, ...
- Geant4 can construct volumes of many shapes (*solids*):
  - Boxes, cones, tubes, ...
  - Boolean operations – volume union, intersection, subtraction.
  - Full list and documentation in the Geant4 Guide for Application Developers.
- In Allpix^2, you build only the sensor volume (wrapper), without inner segmentation.

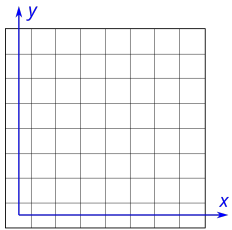


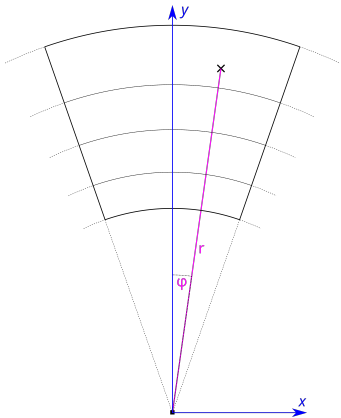
- Every detector is represented by a single volume with no segmentation.
- How do you assign propagated charge to a pixel?
  - Pass charge position to `getPixelIndex()` function, calculate pixel indices.
  - Pass pixel indices to `getPixel()` function, obtain `Pixel` object.
  - Bind the charge to the pixel.



- One global coordinate system - not tied to any detector.
- Local coordinate systems - specific to each detector.
  - Choose the local coordinate center for convenience.
- Transition to other coordinate systems if it's beneficial (e.g. polar, spherical).







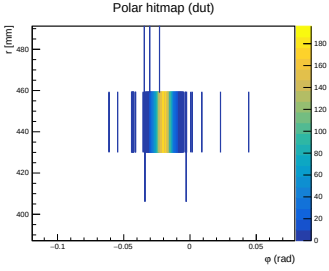
Unit tests

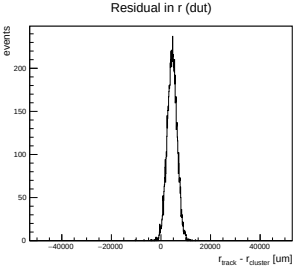
- Good practice to add unit tests with your implementation.
  - Ensures the framework is robust.
  - Protects your implementation from unintended changes (by your or someone else's hand).
- Create a simple simulation with a fixed seed:
  1. Construct your detector. [geometry creation]
  2. Move it, rotate it. [translations and rotations]
  3. Shoot a particle at it. [charge deposition and propagation]
  4. Observe which pixel receives the charge. [coordinate transformations, charge transfer to pixels]
- Define the expected outcome – a string in the output, for example:

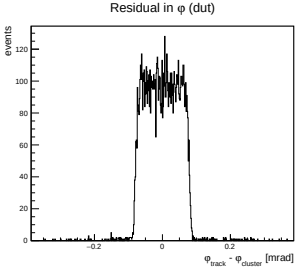
```
#PASS Set of 20 charges combined at (910,2)
#PASS Total charge induced on all pixels:  20e
```

Testing your implementation

- `DetectorHistogrammer` module plots detector hitmaps, charge distributions, residuals, efficiency.
- Very useful during development/testing of detector models.
- Some adjustments might be necessary to generate correct outputs.
- Add outputs relevant to your detector model (e.g. polar hitmap for radial detectors).







Summary

- Allpix^2 can simulate standard sensor shapes and additional layouts are being implemented.
- Possibility for creative detector geometries by implementing a new `DetectorModel` class.
- Many built-in tools to test your implementation during development.