toyExperiment Geometry Service

Geometry/Geometry.h

```
#include "toyExperiment/Geometry/LuminousRegion.h"
#include "toyExperiment/Geometry/Tracker.h"
...
namespace tex {
class Geometry {
public:
  Geometry(const fhicl::ParameterSet&, art::ActivityRegistry&);
    // Accesors
    Tracker const&  tracker()  const { return _tracker; }
    double          bz()       const { return _bz;      }
    LuminousRegion const&  luminousRegion() const { return _luminousRegion; }

    // true if the given point is inside the world box.
    bool insideWorld( CLHEP::Hep3Vector const&);

    // Called by art.
    void preBeginRun( art::Run const &run);

private:
    std::string         _geometryFile;
    Tracker             _tracker;
    LuminousRegion      _luminousRegion;
    // The magnetic field is uniform in the z direction; this is the B_z component of that field.
    double  _bz;
    ...
};
}
```

Geometry "Database"

databaseFiles/geom01.fcl

```
...

tracker : {
  inner : {
    nShells       :   5      // Number of shells
    r0            :  20      // Radius at radial center of the innermost layer, mm
    dr            :  10      // Radial step from layer to layer, mm
    halfLength    : 100      // Half length in z, mm
    halfThickness :   0.150  // Half thickness radially, mm
    sigma         :   0.005  // Resolution in tangent plane, mm
  }

  outer : {
    nShells       :  10
    ...
  }
}

bfield : {
  bz : 1.5  // z component of uniform field, in Tesla.
}

luminousRegion : {
   center : [ 0., 0., 0.]
   sigma  : [ 1.0, 0.1, 5.0 ]
}
```

Geometry Service Implementation

Geometry/Geometry_service.cc

```
tex::Geometry::Geometry( fhicl::ParameterSet const& pset,
                         art::ActivityRegistry&iRegistry) :
  _verbosity(pset.get<int> ("verbosity",0)),
  _geometryFile(pset.get<std::string> ("geometryFile")),
  _tracker()
{
  iRegistry.sPreBeginRun.watch(this, &Geometry::preBeginRun);
}

void tex::Geometry::preBeginRun(art::Run const & run) {
  // Parse the geometry .fcl file into a parameter set.
  ParameterSetFromFile pSetFile(_geometryFile);

  // Construct the tracker.
  fhicl::ParameterSet trackerPSet = pSetFile.pSet().get<fhicl::ParameterSet>("tracker");
  makeTracker( trackerPSet );
  ...
}

void tex::Geometry::makeTracker( fhicl::ParameterSet const& pSet ) {
  for ( auto const& component : TrackerComponent::knownNames() ) {
    fhicl::ParameterSet p = pSet.get<fhicl::ParameterSet>( component.second );
    size_t nShells  = p.get<size_t>("nShells");
    ...
    _tracker.addShell( Shell( id, component.first, r, hlen, hthick, sigma ) );
  }
}
```

Ideas, Comments

- Don't hardcode detectors:
  - Detector base class
  - Retrieve detectors by name
  - For simulations, provide `Detector::GetGDML` (constructs GDML on demand)
  - For digitization and reconstruction, provide similar on-demand construction Get methods
- Time dependence handled via geometry service database (can be made to update geometry for every run, as needed)