ACTS integration for B0 tracker

Sakib Rahman

TGeo Plugin and Unit Tests

TGeoCtub -> DiscSurface

In TGeoTubeConversionTests:

- ❑ Initialize TGeoCtub.
- ❑ Throw exception for any axes assignment in case of TGeoCtub.

In TGeoSurfaceConverter:

- ❑ Add exception for TGeoCtub->Disc/Endcap.

```
double rmin = 5.;
double rmax = 25;
double hz = 2.;
double phimin = -45.;
double phimax = 45.;
double nxlow = 0;
double nylow = 0.707;
double nzlow = -0.707;
double nxhi = 0.;
double nyhi = 0.707;
double nzhi = 0.707;
```

```
TGeoVolume *vol = gGeoManager->MakeTube("Tube", med, rmin, rmax, hz);
TGeoVolume *vols =
    gGeoManager->MakeTubs("Tube", med, rmin, rmax, hz, phimin, phimax);
TGeoVolume *volc =
    gGeoManager->MakeCtub("Tube", med, rmin, rmax, hz, phimin, phimax,
                          nxlow, nylow, nzlow, nxhi, nyhi, nzhi);
```

```
auto tubeCut = dynamic_cast<const TGeoCtub*>(tube);
if (tubeCut) {
  throw std::invalid_argument(
      "TGeoCtub -> CylinderSurface is not allowed for discs."
      "Please use TGeoTube or TGeoTubeSeg.");
}
```

```
for (const auto &axes : allowedAxes) {
...
  BOOST_CHECK_THROW(TGeoSurfaceConverter::toSurface(*volc->GetShape(),
                                                    *gGeoIdentity, axes, 1),
                    std::invalid_argument);
...
}
```

TGeoCtub -> CylinderSurface

In TGeoTubeConversionTests:

- ❑ Initialize TGeoCtub.
- ❑ Throw exception for invalid axes assignment.

In TGeoSurfaceConverter:

- ❑ Restrict the x-component of normals to be 0 at all times. Add an exception otherwise.
- ❑ Calculate bevelMinZ and bevelMaxZ based on normals of the boundary surfaces. The unit test currently fails otherwise.

```
../Tests/UnitTests/Plugins/TGeo/TGeoTubeConversionTests.cpp(156): error: in
"TGeoTube_to_CylinderSurface": check Acts::Test::checkCloseAbs((bevelminz),
(0.25*3.14159265358979323846), (s_epsilon)) has failed. The floating point value 0
is not within absolute tolerance 6.66134e-16 of reference 0.785398.
```

```
if (tubeSeg) {
        double phi1 = toRadian(tubeSeg->GetPhi1());
        double phi2 = toRadian(tubeSeg->GetPhi2());
        if (not boost::starts_with(axes, "X")) {
          throw std::invalid_argument(
              "TGeoShape -> CylinderSurface (sectorial): can only be converted "
              "with "
              "'(X)(y/Y)(*)' axes.");
        }
        halfPhi = 0.5 * (std::max(phi1, phi2) - std::min(phi1, phi2));
        avgPhi = 0.5 * (phi1 + phi2);
        auto tubeCut = dynamic_cast<const TGeoCtub*>(tube);
        if (tubeCut) {
           double nDirlow = tubeCut->GetNlow();
           double nDirhigh = tubeCut->GetNhigh();
           ...
        }
      }
...
}
```

DD4hep Plugin and Unit Tests

In DD4hepLayerBuilder

- ❑ Allow handling of TGeoTube

And TGeoCtub similar to TGeoSurfaceConverter

- ❑ Add angle information?

```
ProtoLayer pl(gctx, layerSurfaces);
if (detExtension->hasValue("r_min", "envelope") &&
    detExtension->hasValue("r_max", "envelope") &&
    detExtension->hasValue("z_min", "envelope") &&
    detExtension->hasValue("z_max", "envelope")) {
  // set the values of the proto layer in case enevelopes are handed over
  pl.envelope[Acts::binR] = {detExtension->getValue("r_min", "envelope"),
                             detExtension->getValue("r_max", "envelope")};
  pl.envelope[Acts::binZ] = {detExtension->getValue("z_min", "envelope"),
                             detExtension->getValue("z_max", "envelope")};
} else if (geoShape != nullptr) {
  TGeoTubeSeg* tube = dynamic_cast<TGeoTubeSeg*>(geoShape);
  if (tube == nullptr) {
    ACTS_ERROR(" Disc layer has wrong shape - needs to be TGeoTubeSeg!");
  }
  ...
}
```

```
struct Extent {
  /// Possible maximal value
  static constexpr double maxval = std::numeric_limits<double>::max();
  /// Start value
  static constexpr Range maxrange = {maxval, -maxval};
  // The different ranges
  std::vector<Range> ranges{(int)binValues, maxrange};
  ...
```

```
double ProtoLayer::min(BinningValue bval, bool addenv) const {
  if (addenv) {
    return extent.min(bval) - envelope[bval].first;
  }
  return extent.min(bval);
}
```

```
// check if layer has surfaces
if (layerSurfaces.empty()) {
  ACTS_VERBOSE(" Disc layer has no senstive surfaces.");
  // in case no surfaces are handed over the layer thickness will be set
  // to a default value to allow attaching material layers
  double z = (zMin + zMax) * 0.5;
  // create layer without surfaces
  // manually create a proto layer
  double eiz = (z != 0.) ? z - m_cfg.defaultThickness : 0.;
  double eoz = (z != 0.) ? z + m_cfg.defaultThickness : 0.;
  pl.extent.ranges[Acts::binZ] = {eiz, eoz};
  pl.extent.ranges[Acts::binR] = {rMin, rMax};
  pl.envelope[Acts::binR] = {0., 0.};
  pl.envelope[Acts::binZ] = {0., 0.};
} else {
  ACTS_VERBOSE(" Disc layer has " << layerSurfaces.size()
                                  << " senstive surfaces.");
  // geometry
  pl.envelope[Acts::binZ] = {std::abs(zMin - pl.min(Acts::binZ)),
                             std::abs(zMax - pl.max(Acts::binZ))};
  pl.envelope[Acts::binR] = {std::abs(rMin - pl.min(Acts::binR)),
                             std::abs(rMax - pl.max(Acts::binR))};
  pl.extent.ranges[Acts::binR] = {rMin, rMax};
}
```

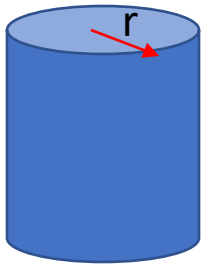
```
// create the share disc bounds
auto dBounds = std::make_shared<const RadialBounds>(pl.min(Acts::binR),
                                                    pl.max(Acts::binR));
double thickness = std::fabs(pl.max(Acts::binZ) - pl.min(Acts::binZ));
```

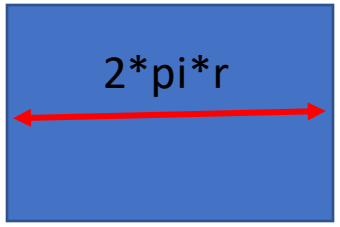
- What is the difference between proto-layer extent and envelope?
- When the layer is empty, assigning zero to envelope and finite values to extent?
- When the layer has surfaces, assigning large value to envelope and but this is cancelled out later when creating disc bounds for sensitive surfaces?
- What's the rationale behind this?

Need some clarification on Bevelled Cylinder Bounds

- Variable domains:
  lposition = [-pi*R, pi*R] ?
  Shiftedlposition = [-pi, pi] ?

- Should the middle left and middle right have factors of pi*radius instead of just radius?





A cylinder unwraps into a rectangle with breadth equal to its circumference

```
bool Acts::CylinderBounds::inside(const Vector2& lposition,
                                  const BoundaryCheck& bcheck) const {
...
  if (bevelMinZ != 0. && bevelMaxZ != 0.) {
    double radius = get(eR);
    float localx =
        lposition[0] > radius ? 2 * radius - lposition[0] : lposition[0];
    Vector2 shiftedlposition = shifted(lposition);
    if ((std::fabs(shiftedlposition[0]) <= halfPhi &&
         std::fabs(shiftedlposition[1]) <= halfLengthZ))
      return true;
    else if ((lposition[1] >= -(localx * std::tan(bevelMinZ) + halfLengthZ)) &&
             (lposition[1] <= (localx * std::tan(bevelMaxZ) + halfLengthZ)))
      return true;
    else {
      // check within tolerance
      auto boundaryCheck = bcheck.transformed(jacobian());

      Vector2 lowerLeft = {-radius, -halfLengthZ};
      Vector2 middleLeft = {0., -(halfLengthZ + radius * std::tan(bevelMinZ))};
      Vector2 upperLeft = {radius, -halfLengthZ};
      Vector2 upperRight = {radius, halfLengthZ};
      Vector2 middleRight = {0., (halfLengthZ + radius * std::tan(bevelMaxZ))};
      Vector2 lowerRight = {-radius, halfLengthZ};
      Vector2 vertices[] = {lowerLeft,  middleLeft,  upperLeft,
                            upperRight, middleRight, lowerRight};
      Vector2 closestPoint =
          boundaryCheck.computeClosestPointOnPolygon(lposition, vertices);

      return boundaryCheck.isTolerated(closestPoint - lposition);
    }
  } else {
    return bcheck.transformed(jacobian())
        .isInside(shifted(lposition), Vector2(-halfPhi, -halfLengthZ),
                  Vector2(halfPhi, halfLengthZ));
  }
...
}
```

Need some clarification on Bevelled Cylinder Bounds

- There are two listed checks. Does not seem like the detailed check deals with anything related to bevelled cylinders. Is that intentional?
- The quick check only deals with closed cylindrical bounds.
- Probable typo
- Would like to understand the rational behind the return value in the fast check.

```
bool Acts::CylinderBounds::inside3D(const Vector3& position,
                                    const BoundaryCheck& bcheck) const {
  // additional tolerance from the boundary check if configred
  bool checkAbsolute = bcheck.m_type == BoundaryCheck::Type::eAbsolute;

  // this fast check only applies to closed cylindrical bounds
  double addToleranceR =
      (checkAbsolute && m_closed) ? bcheck.m_tolerance[0] : 0.;
  double addToleranceZ = checkAbsolute ? bcheck.m_tolerance[1] : 0.;
  // check if the position compatible with the radius
  if ((s_onSurfaceTolerance + addToleranceR) <=
      std::abs(perp(position) - get(eR))) {
    return false;
  } else if (checkAbsolute && m_closed) {
    double bevelMinZ = get(eBevelMinZ);
    double bevelMaxZ = get(eBevelMaxZ);

    double addedMinZ =
        bevelMinZ != 0. ? position.y() * std::sin(bevelMinZ) : 0.;
    double addedMaxZ =
        bevelMinZ != 0. ? position.y() * std::sin(bevelMaxZ) : 0.;

    return ((s_onSurfaceTolerance + addToleranceZ + get(eHalfLengthZ) +
             addedMinZ) >= position.z()) &&
           ((s_onSurfaceTolerance + addToleranceZ + get(eHalfLengthZ) +
             addedMaxZ) <= position.z());
  }
  // detailed, but slower check
  Vector2 lpos(detail::radian_sym(phi(position) - get(eAveragePhi)),
               position.z());
  return bcheck.transformed(jacobian())
      .isInside(lpos, Vector2(-get(eHalfPhiSector), -get(eHalfLengthZ)),
                Vector2(get(eHalfPhiSector), get(eHalfLengthZ)));
}
```