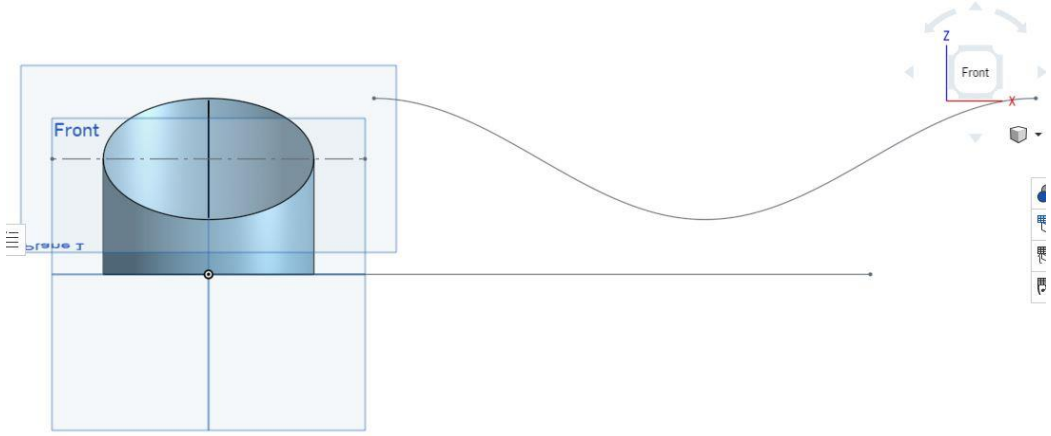
ACTS Bevelled Cylinder Implementation Update

Sakib Rahman

7 July, 2022

Issue

- The unwrapped bevelled cylinder surface is currently modeled as a polygon in ACTS.
- The boundary is actually sinusoidal.
- https://github.com/acts-project/acts/issues/1238



Solution

- It is possible to find analytic form of the 6 boundaries.

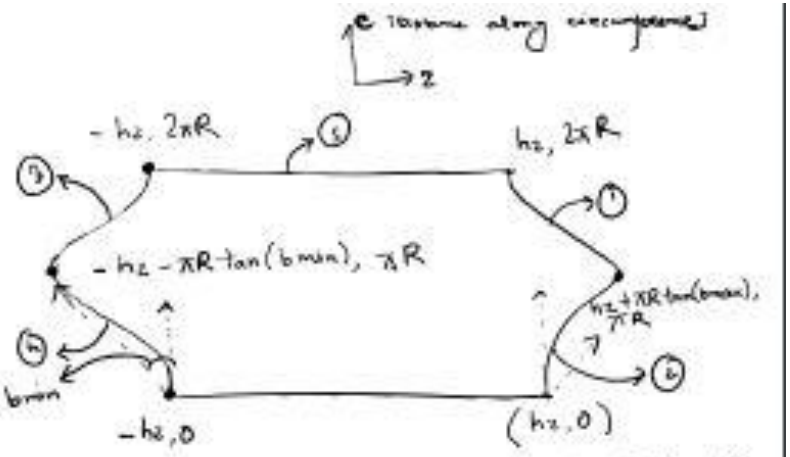
- Replace the two cases highlighted with checks on the value of coordinates evaluated on the boundary using the analytic form.

```
if (bevelMinZ != 0. && bevelMaxZ != 0.) {
    double radius = get(eR);
    // Beleved sides will unwrap to a trapezoid
    ////////////////////////////////////
    //  ________
    // /| .  . |\ r/phi
    // \|______|/ r/phi
    // -Z   0  Z
    ////////////////////////////////////
    float localx =
        lposition[0] > M_PI * radius ? 2 * M_PI * 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);
    }
```

- Multiple probable cases. For example,

  If lposition[1]>hz and lposition[0]<= piR, check distance along z to boundary 2.

  If lposition[1]<=hz and Lposition[0]> piR, check distance along c to boundary 5.



Compute the distance to all of the 6 boundaries and check if any one of them are within tolerance

$$Z = \frac{\pi R \tan(b_{max})}{2} \cos\frac{(C-\pi R)}{R} + hz + \frac{\pi R \tan(b_{max})}{2} \quad \ldots (1)$$

$$Z = -\frac{\pi R \tan(b_{max})}{2} \cos\frac{C}{R} + hz + \frac{\pi R \tan(b_{max})}{2} \quad \ldots (2)$$

$$Z = -\frac{\pi R \tan(b_{min})}{2} \cos\left(\frac{(-\pi R)}{R}\right) - hz + \frac{\pi R \tan(b_{min})}{2}$$

$$Z = \frac{\pi R \tan(b_{min})}{2} \cos\frac{C}{R} - hz - \frac{\pi R \tan(b_{min})}{2}$$

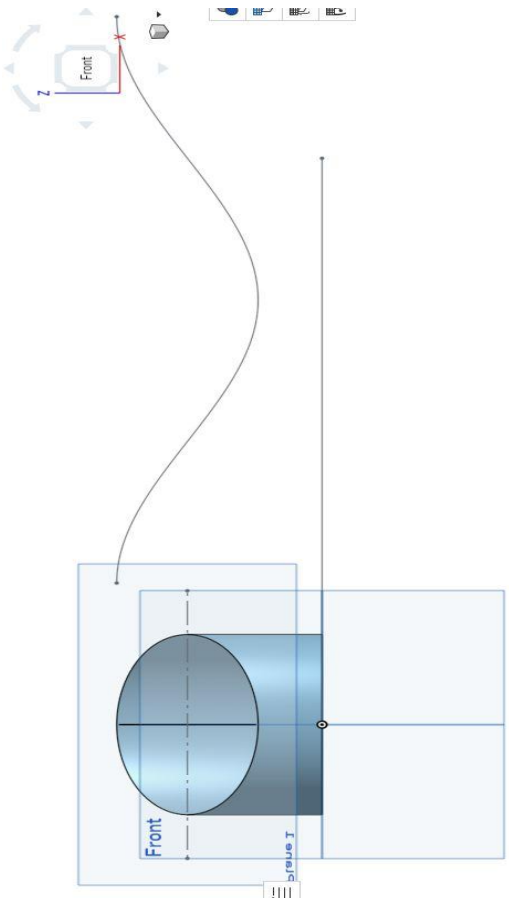
$C = 2\pi R \quad (5)$

$C = 0 \quad (6)$

Implemented New Boundary Condition for Bevelled Cylinders

```
bool Acts::CylinderBounds::inside(const Vector2& lposition,
                                  const BoundaryCheck& bcheck) const {
  double bevelMinZ = get(eBevelMinZ);
  double bevelMaxZ = get(eBevelMaxZ);

  double halfLengthZ = get(eHalfLengthZ);
  double halfPhi = get(eHalfPhiSector);
  if (bevelMinZ != 0. || bevelMaxZ != 0.) {
    double radius = get(eR);
    Vector2 shiftedlposition = shifted(lposition);
    if (std::fabs(shiftedlposition[Acts::eBoundLoc0]) <= halfPhi &&
        std::fabs(shiftedlposition[Acts::eBoundLoc1]) <= halfLengthZ)
      return true;
    else {
      // check within tolerance
      auto boundaryCheck = bcheck.transformed(jacobian());

      double distanceToBoundary = 0;
      if (std::fabs(shiftedlposition[Acts::eBoundLoc0]) > halfPhi &&
          std::fabs(shiftedlposition[Acts::eBoundLoc1]) <= halfLengthZ) {
        distanceToBoundary = std::fabs(shiftedlposition[Acts::eBoundLoc0]) - halfPhi;
        return boundaryCheck.isTolerated({distanceToBoundary,0.0});
      } else {
        if (lposition[Acts::eBoundLoc0] >= M_PI*radius && lposition[Acts::eBoundLoc1]>halfLengthZ)
          distanceToBoundary = lposition[Acts::eBoundLoc1] \
                               -
(M_PI*radius*std::tan(bevelMaxZ)*std::cos(lposition[Acts::eBoundLoc0]/radius-M_PI)/2.0 \
                               +halfLengthZ+M_PI*radius*std::tan(bevelMaxZ)/2.0);
        else if (lposition[Acts::eBoundLoc0] < M_PI*radius && lposition[Acts::eBoundLoc1]>halfLengthZ)
          distanceToBoundary = lposition[Acts::eBoundLoc1] \
                               -(-
M_PI*radius*std::tan(bevelMaxZ)*std::cos(lposition[Acts::eBoundLoc0]/radius)/2.0 \
                               +halfLengthZ+M_PI*radius*std::tan(bevelMaxZ)/2.0);
        else if (lposition[Acts::eBoundLoc0] >= M_PI*radius && lposition[Acts::eBoundLoc1]< -
halfLengthZ)
          distanceToBoundary = lposition[Acts::eBoundLoc1] \
                               -(-
M_PI*radius*std::tan(bevelMinZ)*std::cos(lposition[Acts::eBoundLoc0]/radius-M_PI)/2.0 \
                               -halfLengthZ-M_PI*radius*std::tan(bevelMinZ)/2.0);
        else
          distanceToBoundary = lposition[Acts::eBoundLoc1] \
                               -
(M_PI*radius*std::tan(bevelMinZ)*std::cos(lposition[Acts::eBoundLoc0]/radius)/2.0 \
                               -halfLengthZ-M_PI*radius*std::tan(bevelMinZ)/2.0);
        return boundaryCheck.isTolerated({0.0,distanceToBoundary});
      }
    }
  } else {
    return bcheck.transformed(jacobian())
        .isInside(shifted(lposition), Vector2(-halfPhi, -halfLengthZ),
                  Vector2(halfPhi, halfLengthZ));
  }
}
```

Currently updating unit tests




```
BOOST_AUTO_TEST_CASE(CylinderBoundsProperties) {
  // CylinderBounds object of radius 0.5 and halfz 20
  double nominalRadius{0.5};
  double nominalHalfLength{20.};
  double halfphi(M_PI / 4.0);
  double averagePhi(0.0);
  double bevelMinZ(M_PI / 4);
  double bevelMaxZ(M_PI / 6);
  CylinderBounds cylinderBoundsObject(nominalRadius, nominalHalfLength);
  CylinderBounds cylinderBoundsSegment(nominalRadius, nominalHalfLength,
                                       halfphi, averagePhi);
  CylinderBounds cylinderBoundsBeveledObject(nominalRadius, nominalHalfLength,
                                             M_PI, 0., bevelMinZ, bevelMaxZ);

  /// test for type()
  BOOST_CHECK_EQUAL(cylinderBoundsObject.type(), SurfaceBounds::eCylinder);

  /// test for inside(), 2D coords are r or phi ,z? : needs clarification
  const Vector2 origin{0., 0.};
  const Vector2 atPiBy2{M_PI / 2., 0.0};
  const Vector2 atPi{M_PI, 0.0};
  const Vector2 beyondEnd{0, 30.0};
  const Vector2 unitZ{0.0, 1.0};
  const Vector2 unitPhi{1.0, 0.0};
  const Vector2 withinBevelMin{0.5, -20.012};
  const Vector2 outsideBevelMin{0.5, -40.};
  const BoundaryCheck trueBoundaryCheckWithTolerance(true, true, 0.1, 0.1);
  const BoundaryCheck trueBoundaryCheckWithLessTolerance(true, true, 0.01,
                                                         0.01);
  BOOST_CHECK(
      cylinderBoundsObject.inside(atPiBy2, trueBoundaryCheckWithTolerance));
  BOOST_CHECK(
      !cylinderBoundsSegment.inside(unitPhi, trueBoundaryCheckWithTolerance));
  BOOST_CHECK(
      cylinderBoundsObject.inside(origin, trueBoundaryCheckWithTolerance));

  BOOST_CHECK(!cylinderBoundsObject.inside(withinBevelMin,
                                           trueBoundaryCheckWithLessTolerance));
  BOOST_CHECK(cylinderBoundsBeveledObject.inside(
      withinBevelMin, trueBoundaryCheckWithLessTolerance));
  BOOST_CHECK(!cylinderBoundsBeveledObject.inside(
      outsideBevelMin, trueBoundaryCheckWithLessTolerance));
```

Checking validity of changes

Need two things:

1) Boundary checks work for ACTS bevelled surface

2) Conversion from DD4hep/TGeo cut tubes to ACTS bevelled surface works

*Create a standalone DD4hep example based on ODD for quick check