C++11 code 3.3.12: Sub-problem (3-3.j): Implementation of `DeltaLocalVectorAssembler::Eval()` ↬ GitLab

```
Eigen::VectorXd DeltaLocalVectorAssembler::Eval(const lf::mesh::Entity &cell) {
  Eigen::VectorXd result;
  // get the coordinates of the corners of this cell
  const lf::geometry::Geometry *geo_ptr = cell.Geometry();
  auto vertices = lf::geometry::Corners(*geo_ptr);
  Eigen::Vector2d x_hat;
  // the margin we allow when we determine wether a point is inside an
  // element
  const double margin = 1e-10;
  if (lf::base::RefEl::kTria() == cell.RefEl()) {
    x_hat = PointEvaluationRhs::GlobalInverseTria(vertices, x_0);
    result.resize(3);
    result.setZero();
    if (x_hat(0) <= 1 + margin && x_hat(0) >= 0 - margin &&
        x_hat(1) <= 1 + margin && x_hat(1) >= 0 - margin &&
        x_hat(1) + x_hat(0) <= 1 + margin) {
      already_found = true;
      // Barycentric coordinates on reference triangle
      result[0] = 1.0 - x_hat(0) - x_hat(1);
      result[1] = x_hat(0);
      result[2] = x_hat(1);
    }
  } else if (lf::base::RefEl::kQuad() == cell.RefEl()) {
    x_hat = PointEvaluationRhs::GlobalInverseQuad(vertices, x_0);
    result.resize(4);
    result.setZero();
    if (x_hat(0) <= 1 + margin && x_hat(0) >= 0 - margin &&
        x_hat(1) <= 1 + margin && x_hat(1) >= 0 - margin) {
      already_found = true;
      // Local shape functions on unit square
      result[0] = (1.0 - x_hat(0)) * (1.0 - x_hat(1));
      result[1] = x_hat(0) * (1.0 - x_hat(1));
      result[2] = x_hat(0) * x_hat(1);
      result[3] = (1.0 - x_hat(0)) * x_hat(1);
    }
  } else {
    LF_ASSERT_MSG(
        false, "Function only defined for triangular or quadrilateral cells");
  }
  return result;
}
```