Differential Geometry as a Computational Foundation for Computer-Aided Engineering

Geometric Quantities, Discrete Operators, and Their Role in Simulation Workflows

This article examines differential geometry as a mathematical and computational foundation for modern Computer-Aided Engineering. It concentrates on the geometric quantities most relevant to analysis and implementation—tangent vectors, surface normals, metric tensors, curvature measures, and their discrete analogues on triangulated and quadrilateral meshes—and on the ways these quantities enter meshing, shell formulations, contact formulations, surface operators, post-processing, and shape optimization. The aim is to establish a concise technical framework that links geometric representation to numerical treatment and, in turn, to engineering outcomes, with particular emphasis on model fidelity, formulation consistency, and solver robustness.

1. Relevance of Differential Geometry to Computer-Aided Engineering

In Computer-Aided Engineering, geometry is not merely a passive representation of form; it actively shapes simulation quality, numerical stability, and engineering judgment.
Whether the objective is meshing, contact resolution, shell analysis, or shape optimization, reliable computation depends on the ability to quantify how a surface bends, stretches, and changes orientation.

Differential geometry provides the mathematical and computational foundation for:

- mesh generation and refinement
- shell formulations
- contact formulations
- CAD-to-mesh interoperability
- surface parameterization
- shape optimization
- curvature-aware visualization

For most engineering teams, the priority is not theoretical completeness but operational relevance.
In CAE, the value of differential geometry lies in its **computational subset**: the concepts that directly influence model fidelity, element behavior, solver robustness, and downstream interpretation.

This article isolates that practical core and relates it directly to the CAE workflow.

2. Surface Representation as a Parametric Mapping

A productive way to represent a surface is as a **mapping**:

$$\boldsymbol{x}(u,v)\colon \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$$

This formulation encompasses:

- NURBS patches
- subdivision surfaces
- implicit surfaces converted to parametric form
- triangulated meshes (piecewise linear surface patches)

From this representation, tangent vectors, surface normals, and curvature follow naturally. The next step is to examine the local differential quantities that make this representation computationally useful.

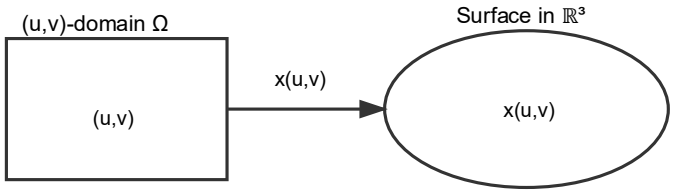

Figure 1: Parametric Surface

3. Tangent Fields and the First Fundamental Form

3.1 Tangent Vectors and the Tangent Plane

$$\boldsymbol{x_u} = \frac{\partial \boldsymbol{x}}{\partial u}, \qquad \boldsymbol{x_v} = \frac{\partial \boldsymbol{x}}{\partial v}$$

Together, these vectors define the **tangent plane** at the point of interest.

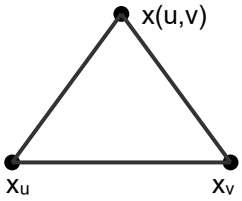

Figure 2: Tangent Plane

3.2 The First Fundamental Form and Metric Tensor

$$E = \boldsymbol{x_u} \cdot \boldsymbol{x_u}, \quad F = \boldsymbol{x_u} \cdot \boldsymbol{x_v}, \quad G = \boldsymbol{x_v} \cdot \boldsymbol{x_v}$$

It determines how distances and angles are measured locally on the surface. Once that local metric structure is established, the analysis can turn from measurement to bending.

In CAE, the metric tensor is central to:

- element distortion
- surface Jacobians
- evaluation of shell membrane strains

- mapping of integration points

Example 1. Computing Tangents and the Metric Tensor (C++)

```
template <typename T>
struct SurfacePoint
{
    basepoint3<T> x; // position
    basevec3<T> xu;  // ∂x/∂u
    basevec3<T> xv;  // ∂x/∂v
    basevec3<T> n;   // unit normal
};

template <typename T>
struct MetricTensor
{
    T E, F, G;
};
template <typename T>
inline MetricTensor<T> computeMetric(const SurfacePoint<T> &p)
{
    MetricTensor<T> m;
    m.E = dot(p.xu, p.xu);
    m.F = dot(p.xu, p.xv);
    m.G = dot(p.xv, p.xv);
    return m;
}
```

4. Surface Normals and Curvature Measures

4.1 Surface Normal Definition and Use

$$\boldsymbol{n} = \frac{\boldsymbol{x_u} \times \boldsymbol{x_v}}{\|\boldsymbol{x_u} \times \boldsymbol{x_v}\|}$$

Applications include:

- contact formulations
- shell formulations
- boundary-condition specification
- CAD-to-mesh consistency

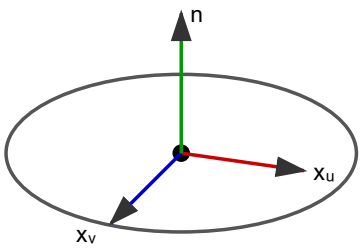

Figure 3: Surface Normal Vector

4.2 Curvature Measures and Differential Characterization

Curvature quantifies local surface bending.

Its evaluation, in turn, requires second derivatives:

$$\boldsymbol{x}_{\mathrm{uu}},\ \boldsymbol{x}_{\mathrm{uv}},\ \boldsymbol{x}_{vv}$$

These terms define the **shape operator**, from which we obtain the principal curvature measures used in analysis. In practice, however, CAE workflows usually require discrete approximations of these continuous quantities.

- principal curvatures k_1, k_2
- principal directions
- mean curvature H
- Gaussian curvature K

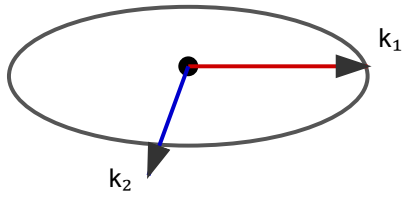

Figure 4: Principal Curvatures and Directions

Example 2. Principal Curvature Computation (Simplified, C++)

```
template <typename T>
struct CurvatureTensor
{
    T k1, k2;
    basevec3<T> d1, d2; // principal directions
};

template <typename T>
CurvatureTensor<T> computeCurvature(
    const SurfacePoint<T>& p,
    const basevec3<T>& xuu,
    const basevec3<T>& xuv,
    const basevec3<T>& xvv)
{
    // Build first and second fundamental forms
    T E = dot(p.xu, p.xu);
    T F = dot(p.xu, p.xv);
    T G = dot(p.xv, p.xv);

    T L = dot(xuu, p.n);
    T M = dot(xuv, p.n);
    T N = dot(xvv, p.n);
    // Solve generalized eigenproblem |II - k I| = 0
    // (implementation omitted for brevity)

    CurvatureTensor<T> K;
    // fill K.k1, K.k2, K.d1, K.d2
```

```
    return K;
}
```

5. Discrete Differential Geometry on Computational Meshes

Most CAE workflows operate on **triangulated** or **quadrilateral** meshes.
As a result, smooth geometric quantities must be approximated in discrete form.

5.1 Discrete Normal Construction

Common discrete normal definitions include:

- face normals
- area-weighted vertex normals
- angle-weighted vertex normals

Example 3. Angle-Weighted Vertex Normal (C++)

```
template <typename T>
basevec3<T> computeVertexNormal(int v, const Mesh<T>& mesh) {
    basevec3<T> n(0.0);
    for (auto f : mesh.facesAroundVertex(v)) {
        basevec3<T> fn = mesh.faceNormal(f);
        T angle = mesh.cornerAngle(f, v);
        n += angle * fn;
    }
    return normalize(n);
}
```

5.2 Discrete Curvature Estimation

Widely used discrete curvature estimators include:

- Meyer et al. mean curvature estimator
- normal variation
- quadric error metrics
- umbrella operators

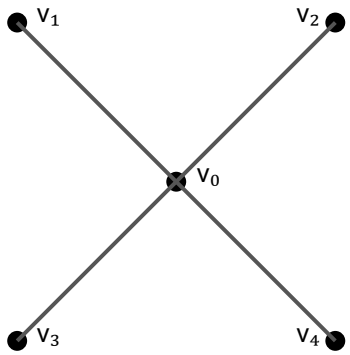

Figure 5: Discrete Curvature Neighborhood

5.3 The Discrete Laplace–Beltrami Operator

$$\Delta \boldsymbol{x} = \sum_j w_{ij}(\boldsymbol{x_j} - \boldsymbol{x_i})$$

This operator is used for:

- smoothing
- curvature flow
- parameterization
- shape optimization

Taken together, these discrete operators translate differential geometry into forms that can be applied directly within engineering workflows. The next section considers where those quantities enter the CAE process in practice.

Example 4. Cotangent Laplacian (C++)

```
template <typename T>
inline basevec3<T> laplacian(int v, const Mesh<T> &mesh)
{
    basevec3<T> sum(0.0f);
    T wsum = 0.0;

    for (auto e : mesh.edgesAroundVertex(v))
    {
        int j = mesh.otherVertex(e, v);
        T w = cotangentWeight(mesh, v, j);
        sum += w * (mesh.position(j) - mesh.position(v));
        wsum += w;
    }

    return (wsum > 0.0) ? sum / wsum : basevec3<T>(0.0f);
}
```

6. Role of Differential Geometry in the CAE Workflow

6.1 From CAD to Mesh Generation

In the CAD-to-mesh stage, curvature informs decisions that:

- detect sharp features
- identify developable regions
- guide anisotropic refinement
- control boundary layer thickness

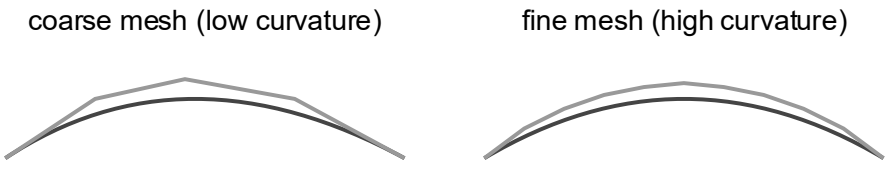

Figure 6: Curvature-Driven Refinement

6.2 From Mesh Representation to Solver Formulation

In solver preparation, these differential quantities are used to:

- compute surface Jacobians
- construct local frames
- evaluate shell bending and membrane strains
- ensure consistent normals for contact

Example 5. Local Tangent Frame for Shell Formulations (C++)

```
template <typename T>
struct TangentFrame
{
    basevec3<T> t1, t2, n;
};
template <typename T>
inline TangentFrame<T> buildFrame(const SurfacePoint<T> &p)
{
    TangentFrame<T> f;
    f.t1 = normalize(p.xu);
    f.n = normalize(cross(p.xu, p.xv));
    f.t2 = normalize(cross(f.n, f.t1));
    return f;
}
```

6.3 From Solver Output to Post-Processing Interpretation

In post-processing, curvature also supports:

- stress-field visualization on curved surfaces
- alignment between principal stress directions and principal curvature directions
- curvature-based filtering of noisy fields

These applications show that differential geometry is not confined to isolated algorithms; it spans the full simulation pipeline. That breadth, in turn, motivates a coherent implementation strategy.

7. Implementation Considerations for TheMeshProject

7.1 Core Framework Requirements

- unified interface for differential quantities
- support for CAD and discrete meshes
- curvature-estimation modules
- tangent frame utilities
- Laplace–Beltrami operators
- post-processing and visualization utilities

7.2 API Structure

- btm::geometry::MetricTensor
- btm::geometry::CurvatureTensor
- btm::geometry::TangentFrame
- btm::mesh::DiscreteLaplacian
- btm::mesh::NormalEstimator

7.3 Directions for Future Integration

This article establishes the foundation for subsequent work on the following capabilities, each of which extends the same geometric framework into implementation detail:

- shell formulations
- mesh-quality metrics
- geometry-aware refinement
- parameterization
- geometric feature detection

8. Concluding Remarks

Taken as a whole, the preceding sections show that differential geometry is not a peripheral mathematical topic in CAE; it is part of the discipline's operational foundation.
It provides the language and structure needed to connect geometric representation to numerical treatment and, ultimately, to engineering outcomes. In practical terms, it clarifies:

- how surfaces bend
- how distances distort
- how normals behave
- how curvature guides refinement
- how discrete meshes approximate smooth shapes

For TheMeshProject, the strategic implication is clear: differential geometry should be treated not as background theory, but as core engineering infrastructure linking geometric representation, numerical treatment, and practical simulation outcomes.