# Architecture and Implementation of a Semantic Geometric Kernel for Panel Bending CAM: A Feature-Based Approach using OpenCASCADE Technology

## 1. Executive Overview: Bridging the Semantic Gap in CAM

The development of automated Computer-Aided Manufacturing (CAM) software for panel bending machinery—such as the Salvagnini STREAMBEND or Prima Power systems—presents a unique set of challenges distinct from traditional milling or turning CAM. While subtractive manufacturing relies heavily on geometric boundaries to define toolpaths, sheet metal bending is fundamentally a topological transformation process. The machine does not merely traverse a surface; it reconfigures the topological adjacencies of the part itself. Consequently, the input data—typically a "dumb" boundary representation (B-Rep) provided via STEP AP203/AP214 protocols—is insufficient for direct machine control. The raw geometric data describes *what* the shape looks like, but fails to encode *how* it is constructed or *how* it behaves physically.[1]

This report details the architectural design and algorithmic implementation of a specialized Geometric Kernel tailored for this purpose. The primary objective of this kernel is to ingest raw, potentially "dirty" 3D geometry and transmute it into a semantic Face-Adjacency Graph (FAG). This graph serves as the "digital twin" of the sheet metal part, where nodes represent stable flanges and edges represent deformation zones (bends). By abstracting the complex B-Rep data into this semantic structure, we enable the downstream automation of unfolding, collision detection, and tool selection.

We analyze the critical technical constraints inherent to this domain: the necessity for robustness against non-manifold geometry common in neutral exchange formats, the mathematical heuristics required to distinguish topologically similar features (e.g., distinguishing a cylindrical bend from a cylindrical hole), and the automated inference of physical properties such as material thickness and optimal clamping bases. The proposed solution leverages the OpenCASCADE Technology (OCCT) library, utilizing its advanced topological traversal and shape healing capabilities to build a robust, industrial-grade kernel.[3]

# 2. Computational Foundations and Data Structures

To engineer a kernel capable of semantic feature recognition, one must first establish a rigorous understanding of the underlying data structures. The OpenCASCADE Technology (OCCT) library employs a hierarchy that separates the abstract definition of topology from the mathematical definition of geometry. This separation is the cornerstone upon which our feature recognition algorithms are built.

## 2.1 The Topological Hierarchy in Boundary Representation

In the context of a STEP file import, the geometric model is deserialized into a TopoDS_Shape. This generic container acts as the root of a topological tree. For a sheet metal part, the hierarchy typically resolves to a single TopoDS_Solid or a TopoDS_Shell representing the thin-walled volume. The kernel must parse this hierarchy to access the atomic elements of the model.[5]

The hierarchy is defined by the TopAbs_ShapeEnum and includes:

- **TopoDS_Face**: The primary unit of analysis. In our semantic model, faces are candidates for classification as either "Flanges" (nodes in the FAG) or "Bends" (edges in the FAG). A face is a topological boundary defined by a mathematical surface and bounded by wires.
- **TopoDS_Wire**: A collection of connected edges that form a loop. The distinction between the *outer wire* and *inner wires* of a face is critical for distinguishing between the boundary of a flange and internal cutouts (holes).[7]
- **TopoDS_Edge**: The 1D boundary curve. Edges carry the topological connectivity information; two faces are adjacent if and only if they share a common edge.
- **TopoDS_Vertex**: The 0D points bounding the edges.

## 2.2 Geometry vs. Topology: The Role of Adaptors

A frequent pitfall in computational geometry is conflating the topological entity (Face) with its geometric definition (Surface). A TopoDS_Face may contain a reference to a Geom_Surface (e.g., a B-Spline) that is geometrically cylindrical but topologically trimmed. To robustly classify features, the kernel must analyze the intrinsic properties of the surface regardless of its parametric representation.

The BRepAdaptor_Surface class serves as the interface between the topological and geometric domains. It allows the kernel to query surface properties—such as curvature, continuity, and type—without manually handling the transformation matrices (TopLoc_Location) or the specific subclass of the underlying geometry (e.g., converting a NURBS approximation of a cylinder back into analytical terms for radius extraction).[9] This abstraction is vital for the heuristic algorithms described in Section 6, as it normalizes the diverse inputs from different CAD exporters into a standard analytical framework.

# 3. Ingestion and Sanitization: Handling "Dirty" Geometry

The industrial reality of receiving CAD data from external clients is that the models are rarely mathematically perfect. "Dirty" geometry—characterized by gaps, self-intersections, inverted normals, and non-manifold edges—is pervasive. A semantic kernel that assumes perfect input will fail in production. Therefore, the first stage of the pipeline is a robust Sanitization and Shape Healing layer.[10]

## 3.1 The Necessity of Shape Healing

STEP files often lose precision during translation. A "closed" solid in the source CAD system (e.g., SolidWorks) may arrive in OCCT as a collection of disjoint faces with microscopic gaps between edges. If the kernel attempts to build the FAG on this data, the adjacency checks will fail; faces that should be connected via a bend will appear as separate islands, breaking the graph traversal.

The ShapeFix package in OCCT provides a suite of tools to rectify these anomalies. The ShapeFix_Shape class acts as a high-level manager, orchestrating specific fixers for wires, faces, and shells.[10]

## 3.2 Heuristic Healing Strategies

We propose a standardized healing routine to be executed immediately upon import:

1. **Tolerance Unification**: The ShapeFix_Shape::SetPrecision method should be configured with a tolerance slightly larger than the export precision (e.g., 1e-4 to 1e-3 mm). This allows ShapeFix_Wire to bridge micro-gaps and snap "floating" vertices to a common location, effectively "sewing" the topology back together.
2. **Orientation Normalization**: For the FAG to function, the notion of "material side" vs. "void side" must be consistent. The ShapeFix_Solid tool ensures that all face normals point outward from the material volume. This is critical for the "Ray Casting" thickness inference (Section 8), which relies on normal vectors to determine ray direction.
3. **Small Edge Removal**: Sheet metal models often contain artifacts—tiny edges or sliver faces generated by boolean operations (e.g., near corner reliefs). These artifacts confuse feature recognition. ShapeFix_Wire::FixSmallEdges eliminates edges shorter than the specified precision, simplifying the topology without altering the functional geometry.[11]

## 3.3 Canonicalization of Geometry

Feature recognition algorithms are significantly more efficient when operating on analytical surfaces (Planes, Cylinders) rather than freeform NURBS. Some CAD exporters approximate cylindrical bends as B-Splines. The healing layer must detect these approximations and convert them. Using ShapeCustom_SweptToElementary, the kernel can attempt to replace

Geom_BSplineSurface entities that conform to cylindrical definitions with actual Geom_CylindricalSurface objects. This allows downstream logic to simply check surface.GetType() == GeomAbs_Cylinder rather than performing expensive curvature sampling on splines.[12]

## 4. Feature Recognition Architecture: The "Smart" Transformation

The core value proposition of the kernel is the transformation of "dumb" geometry into "smart" features. This requires a classification engine that iterates through every topological entity and assigns it a semantic role: Flange, Bend, Hole, or Fillet.

### 4.1 Surface Classification Primitives

The classification process begins with a geometric pass using BRepAdaptor_Surface. We categorize faces into three primary buckets:

- **Planar Candidates**: Surfaces where GetType() == GeomAbs_Plane. These are potential Flanges or Webs.
- **Cylindrical Candidates**: Surfaces where GetType() == GeomAbs_Cylinder. These are potential Bends, Holes, or Fillets.
- **Complex Surfaces**: GeomAbs_Torus, GeomAbs_Cone, etc. These typically represent formed features (dimples, louvers) or chamfers, which are treated as attributes of the flange rather than topological connectors in the simplified FAG.

### 4.2 The Challenge of Ambiguity

The primary algorithmic challenge, as highlighted in the user query, is distinguishing between **Bends**, **Holes**, and **Fillets**. Geometrically, all three can be partial cylinders. A "dumb" step file does not label them. The distinction is purely topological and contextual.[3]

The kernel must employ a multi-stage heuristic filter to resolve this ambiguity. This filter analyzes the connectivity of the candidate cylinder to its neighbors and the geometric properties of those neighbors.

## 5. Heuristic Algorithms for Feature Classification

### 5.1 Distinguishing Bends, Holes, and Fillets

The classification logic is summarized in the following decision matrix. We assume the candidate face is a **Cylinder** with Radius $R$.

| **Feature Type** | **Connectivity** | **Neighbor** | **Thickness** |
|---|---|---|---|

| | Pattern | Characteristics | Consistency |
|---|---|---|---|
| **Bend** | Connects 2 distinct Faces ($F_1, F_2$) | $F_1, F_2$ are Planar and **Non-Parallel** (usually) | Distance between Inner/Outer Cylinder = $T$ |
| **Hole** | Connects 1 Face to itself (Internal Loop) OR Connects 2 Parallel Faces | Neighbors are Parallel (Top/Bottom of sheet) | Cylinder Length = $T$ |
| **Fillet** | Connects 2 Faces ($F_1, F_2$) | $F_1, F_2$ are Orthogonal or Tangent | Variable (often additive or subtractive) |

### 5.1.1 The Bend Detection Algorithm

A Bend is a deformation zone that provides structural continuity.

1. **Radius Check**: Verify $R$ is within manufacturable limits ($R_{min} < R < R_{max}$).
2. **Adjacency Check**: Retrieve the list of adjacent faces using TopExp::MapShapesAndAncestors (mapping Edges to Faces). A standard bend must share edges with exactly two planar faces (the flanges).[16]
3. **Planarity Check**: Verify that both neighbors are GeomAbs_Plane.
4. **Material Flow Check**: This is the critical differentiator from a fillet. In a bend, the material "flows" from Flange A, through the Bend, to Flange B. The normal vectors of the flanges typically diverge (convex) or converge (concave). If the flanges are coplanar, the cylinder is likely a stiffening rib, not a bend.
5. **Curvature Alignment**: The axis of the cylinder must be parallel to the shared edges (bend lines).

### 5.1.2 The Round Hole Detection Algorithm

A Hole is a topological void.

1. **Loop Containment**: Inspect the wires of the adjacent planar face. Using BRepTools::OuterWire, identify the boundary of the neighbor flange. If the edge shared with the cylinder belongs to an **internal wire** (a hole inside the face), the cylinder is a Hole.[8]
2. **Through-Hole Topology**: A through-hole often consists of two semi-cylindrical faces (due to STEP export splitting) connecting the Top Face to the Bottom Face. If the cylinder

connects two parallel planar faces separated by the material thickness $T$, it is a Hole.

### 5.1.3 The Fillet Detection Algorithm

A Fillet is a smoothing feature.

1. **Corner Fillet**: If the cylinder axis is parallel to the normal of the adjacent planar face, it is a corner radius (rounding the sharp corner of a flange), not a bend.
2. **Surface Fillet**: If the cylinder connects two orthogonal faces but does not maintain the constant wall thickness constraint (i.e., no corresponding inner/outer radius pair), it is likely a machined fillet on a block, which might be flagged as non-unfoldable in sheet metal context.[15]

## 5.2 Pseudo-Code for Classifier

C++

```
FeatureType ClassifyCylinder(const TopoDS_Face& cylFace, const
TopTools_IndexedDataMapOfShapeListOfShape& mapEF) {
    BRepAdaptor_Surface adapt(cylFace);
    if (adapt.GetType()!= GeomAbs_Cylinder) return Feature_Unknown;

    // Get Neighbors
    TopTools_ListOfShape neighbors;
    //... Iterate edges of cylFace, check mapEF to find adjacent faces...

    if (neighbors.Extent() == 1) return Feature_Hem; // Rolled edge
    if (neighbors.Extent() > 2) return Feature_Complex; // T-junction

    TopoDS_Face F1 = TopoDS::Face(neighbors.First());
    TopoDS_Face F2 = TopoDS::Face(neighbors.Last());

    // Check if neighbors are parallel (Hole check)
    if (AreFacesParallel(F1, F2)) {
        if (Distance(F1, F2) == MaterialThickness) return Feature_Hole;
    }

    // Check wire containment (Hole check)
    if (IsEdgeInternal(cylFace, F1)) return Feature_Hole;

    // Check geometric continuity (Bend check)
```

```
  if (!AreFacesCoplanar(F1, F2)) return Feature_Bend;

  return Feature_Fillet;
}
```

## 6. The Face-Adjacency Graph (FAG): Data Structure Implementation

The output of the feature recognition phase is the Face-Adjacency Graph (FAG). This data structure abstracts the B-Rep into a form suitable for manufacturing logic.

### 6.1 Graph Definition

- **Nodes ($N$)**: Represent **Planar Flanges**.
  - *Attributes*: ID (Persistent Index), SurfaceArea ($mm^2$), Normal ($n$), CenterOfGravity ($P_{xyz}$), Thickness ($T$).
- **Edges ($E$)**: Represent **Connections**.
  - *Type*: Bend (Cylindrical), Weld (Sharp), or Virtual (for disconnected parts).
  - *Attributes*: BendAngle ($\theta$), InnerRadius ($R$), KFactor ($K$), BendLength ($L$).

### 6.2 C++ Implementation Strategy

The FAG should be implemented as a dedicated C++ class, distinct from the OCCT topological structures. We recommend using an adjacency list representation for efficiency in sparse graphs (which sheet metal parts typically are).

C++

```
struct FAGNode {
  Standard_Integer TopoIndex; // Index in TopTools_IndexedMapOfShape
  gp_Dir Normal;
  Standard_Real Area;
  Standard_Boolean IsBase;
};

struct FAGEdge {
  Standard_Integer TargetNodeIndex;
```

```
    Standard_Real Angle;
    Standard_Real Radius;
    Standard_Real KFactor;
};

class FaceAdjacencyGraph {
    std::vector<FAGNode> myNodes;
    std::vector<std::vector<FAGEdge>> myAdjacencyList;

public:
    void Build(const TopoDS_Shape& shape);
    //... Traversal methods...
};
```

The TopoIndex provides the link back to the TopoDS_Shape for visualization or further geometric query. This decoupling allows the graph algorithms (unfolding, centrality) to run without the overhead of B-Rep traversal.[19]

## 7. Base Face Identification: A Weighted Scoring Algorithm

In Panel Bending, the "Base Face" is the segment of the part that remains clamped by the manipulator while the rest of the part is bent. Selecting the optimal base face is critical for process stability and minimizing robot movements. We propose a weighted scoring algorithm to automate this selection.

### 7.1 The Scoring Function

For every node (Flange) $i$ in the FAG, we calculate a score $S_i$:

$$S_i = w_1 \cdot \hat{A}_i + w_2 \cdot \hat{C}_i + w_3 \cdot \hat{O}_i$$

Where:

- $\hat{A}_i$ is the Normalized Surface Area.
- $\hat{C}_i$ is the Normalized Graph Centrality.
- $\hat{O}_i$ is the Orientation Alignment Score.
- $w_1, w_2, w_3$ are weighting coefficients (e.g., 0.5, 0.3, 0.2).

### 7.2 Metric 1: Surface Area ($\hat{A}_i$)

A larger surface area provides better vacuum/magnetic clamping and stability.

- **Calculation**: Use GProp_GProps on the TopoDS_Face geometry to compute the area.
- **Normalization**: $\hat{A}_i = \frac{Area_i}{\max(Area)}$.

### 7.3 Metric 2: Graph Centrality ($\hat{C}_i$)

A base face should be topologically central. If a leaf node (a small flange at the tip of the part) is chosen as the base, the manipulator must swing the entire mass of the part to perform bends at the other end, leading to inertia issues and potential collisions.

- **Algorithm**: We utilize **Closeness Centrality**.
  - Compute the shortest path distance $d(i, j)$ (number of bends) between node $i$ and all other nodes $j$ using Breadth-First Search (BFS) on the FAG.
  - Closeness $C_i = \frac{1}{\sum_{j \neq i} d(i,j)}$.
  - This metric favors faces that are "close" to all other faces in the bending sequence.[21]

### 7.4 Metric 3: Orientation Alignment ($\hat{O}_i$)

Panel benders typically operate with a horizontal reference table. We prefer a base face that aligns with the Global XY plane of the imported model, as this matches the likely design intent.

- **Calculation**: Compute the dot product of the Face Normal $n_i$ and the Global Z axis $Z = (0, 0, 1)$.
- **Score**: $\hat{O}_i = |n_i \cdot Z|$. A score of 1.0 indicates perfect horizontal alignment.

## 8. Automatic Material Thickness Inference

To apply the correct K-Factor and calculate bend deductions, the kernel must know the material thickness $T$. Relying on user input is error-prone; the kernel should infer this from the geometry.

### 8.1 The Ray Casting Approach

This method is robust for solids. It simulates measuring the part with a caliper.[23]

1. **Selection**: Pick a candidate planar face $F_{start}$ (ideally the Base Face).

2. **Ray Construction**: Define a ray starting at the center of $F_{start}$ with direction $-n$ (pointing into the material).
3. **Intersection**: Use BRepIntCurveSurface_Inter to find intersection points between the ray and the entire TopoDS_Solid.
4. **Analysis**: The ray will pierce the solid at multiple points.
   - $t = 0$: The start point on $F_{start}$.
   - $t_1$: The exit point on the opposite face.
   - $t_2$: Entry into a subsequent fold (if the part is folded over).
5. **Result**: The thickness $T = t_1$.

## 8.2 Parallel Face Distance Method

An alternative approach uses BRepExtrema_DistShapeShape.[24]

1. **Clustering**: Group all planar faces by their normal vectors.
2. **Pairing**: Identify pairs of faces with anti-parallel normals ($n_1 \approx -n_2$).
3. **Measurement**: Calculate the minimum distance between these pairs. The most frequent non-zero distance across the model is the thickness $T$.
   - *Advantage*: Can handle cases where ray casting might miss (e.g., holes at the center).
   - *Disadvantage*: computationally more expensive ($O(N^2)$) compared to $O(1)$ ray casting.

## 8.3 K-Factor Association

Once $T$ is known, the **K-Factor** (Neutral Axis offset) can be assigned. In "dumb" geometry, the K-Factor is implicit in the relationship between the flat layout and the folded model. However, for a *folded* input, we typically assign a standard K-Factor based on material tables (e.g., $K = 0.33$ for $R < T$, $K = 0.5$ for $R \geq T$) or derive it if a flat pattern is also provided. This attribute is stored on the FAG Edges.[1]

# 9. Conclusion

The architecture presented in this report transforms the "dumb" data of a STEP file into a "smart," manufacturing-ready Face-Adjacency Graph. By leveraging OpenCASCADE's powerful topological tools (BRepAdaptor, ShapeFix) and applying domain-specific heuristics for feature recognition, we can distinguish complex features like bends, holes, and fillets with high reliability.

The integration of a weighted scoring system for Base Face selection and ray-casting algorithms for thickness inference ensures that the kernel is not just a geometry viewer, but an intelligent CAM engine. This system lays the groundwork for fully automated generation of bending sequences, collision-free toolpaths, and accurate flat patterns, directly addressing the needs of modern Panel Bender software.

Future work should focus on the refinement of the "Hem" detection logic (0-radius bends) and the integration of the Boost Graph Library to optimize the centrality calculations for extremely complex parts with hundreds of bends.

## Works cited

1. Sheet Metal Operations Component - Open Cascade, accessed February 2, 2026, https://www.opencascade.com/components/sheet-metal-operations/
2. On sheet metal recognition and unfolding (Part 1), accessed February 2, 2026, https://quaoar.su/blog/page/on-sheet-metal-unfolding-part-1
3. Modeling Algorithms - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_algos.html
4. CAD Processor - Open Cascade, accessed February 2, 2026, https://www.opencascade.com/products/cad-processor/
5. Modeling Data - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_data.html
6. Modeling Data - Open CASCADE Technology, accessed February 2, 2026, https://old.opencascade.com/doc/occt-6.7.0/overview/html/user_guides__modeling_data.html
7. Tutorial - Open CASCADE Technology, accessed February 2, 2026, https://old.opencascade.com/doc/occt-7.0.0/overview/html/occt__tutorial.html
8. How to get external and internal boundary (holes) edges for mesh face - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/how-get-external-and-internal-boundary-holes-edges-mesh-face
9. BRepAdaptor_Surface Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_adaptor___surface.html
10. ShapeFix_Shape Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_shape_fix___shape.html
11. I need some help with the Shape Healing functions - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/i-need-some-help-shape-healing-functions
12. ShapeFix Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_shape_fix.html

13. Classifying TopoDS_Face face - opencascade - Stack Overflow, accessed February 2, 2026, https://stackoverflow.com/questions/55571103/classifying-topods-face-face
14. How to recognize a circle(cylinder) hole or rectangle hole on a TopoDS_Shape by C++ code? - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/content/how-recognize-circlecylinder-hole-or-rectangle-hole-topodsshape-c-code
15. Recognize Fillet - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/recognize-fillet
16. Get belonging face to an edge? - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/get-belonging-face-edge
17. Inner and Outer Wire - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/inner-and-outer-wire
18. Let's talk about fillets - OpenCASCADE's recipies from NEWe, accessed February 2, 2026, https://neweopencascade.wordpress.com/2018/10/17/lets-talk-about-fillets/
19. On sheet metal unfolding. Part 12: searching for feasible bends | by ..., accessed February 2, 2026, https://analysis-situs.medium.com/on-sheet-metal-unfolding-part-12-searching-for-feasible-bends-2a91a8976431
20. Indexing the topology of OpenCascade - Manifold Geometry // Многообразная Геометрия, accessed February 2, 2026, https://quaoar.su/blog/page/indexing-the-topology-of-opencascade
21. (Python) Function betweenness_centrality_clustering - Boost, accessed February 2, 2026, https://www.boost.org/doc/libs/latest/libs/graph/doc/bc_clustering.html
22. closeness_centrality.cpp - Boost, accessed February 2, 2026, https://www.boost.org/doc/libs/1_66_0/libs/graph/example/closeness_centrality.cpp
23. Ray casting in code (not in 3D viewer) - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/ray-casting-code-not-3d-viewer
24. BRepExtrema_DistShapeShape Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_extrema___dist_shape_shape.html
25. BRepExtrema_DistShapeShape doesn't return all shapes - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/content/brepextremadistshapeshape-doesnt-return-all-shapes