Architectural Specification for High-Performance Geometric Kernels in Panel Bender CAM: Semantic Feature Recognition via OpenCASCADE

1. Introduction: The Geometric Imperative in Panel Bending

The evolution of Computer-Aided Manufacturing (CAM) for sheet metal fabrication has transitioned from manual G-code programming to automated, semantic-driven process planning. In the specific domain of Panel Bending, this transition presents unique computational challenges distinct from traditional Press Brake operations. While a press brake deforms material along a linear axis using a vertical punch and die, a panel bender employs a complex manipulator to hold the sheet horizontally and auxiliary blades to fold flanges positively or negatively. This kinematic difference fundamentally alters the data requirements: the machine controller requires not merely a geometric description of the finished part, but a topological understanding of the bending sequence, collision volumes, and material flow.

The "Geometric Kernel" of a Panel Bender CAM software serves as the translation engine between the "dumb" geometry of a standard exchange format (such as STEP AP203/214/242) and the "smart" semantics required by the machine. A raw 3D model contains only boundaries—surfaces, curves, and vertices—without explicit knowledge of what constitutes a "flange," a "bend," or a "hem." The kernel's primary responsibility is Feature Recognition: the algorithmic reconstruction of the design intent. This involves parsing the Boundary Representation (B-Rep) to construct a high-level **Face-Adjacency Graph (FAG)**, where nodes represent planar flanges and edges represent the deformation zones (bends) connecting them.

This report provides an exhaustive technical analysis of implementing such a kernel using **OpenCASCADE Technology (OCCT)**. It addresses the critical engineering constraints of the project: robustness against "dirty" geometry common in industrial CAD exports, and algorithmic efficiency to avoid combinatorial explosion ($O(N^2)$ complexity) when processing complex chassis with hundreds of faces. We will explore the theoretical underpinnings of B-Rep topology, detailed traversal strategies using TopExp::MapShapesAndAncestors, robust C++ data structures for graph representation, and advanced filtering techniques for topological artifacts like seam edges.[1]

2. Theoretical Framework: Topology and Geometry in OpenCASCADE

To engineer a robust solution, one must first possess a rigorous understanding of the underlying data structures provided by OpenCASCADE (OCCT). OCCT adheres to the ISO 10303-42 standard for Boundary Representation, distinguishing strictly between *Topology* (connectivity and orientation) and *Geometry* (mathematical definition).

2.1 The TopoDS Hierarchy and Orientation

The TopoDS package manages the relationship between entities. In a sheet metal part, the hierarchy of interest is Solid $\rightarrow$ Shell $\rightarrow$ Face $\rightarrow$ Wire $\rightarrow$ Edge $\rightarrow$ Vertex. A critical concept often overlooked in naive implementations is **Orientation** (TopAbs_FORWARD, TopAbs_REVERSED, TopAbs_INTERNAL, TopAbs_EXTERNAL).[3]

In a valid Manifold Solid B-Rep, the orientation defines the material side. A Face restricts an underlying infinite surface (Geometry) to a finite region. The orientation of the Face relative to the Surface normal determines the "outside" of the solid. Typically, OCCT solids are constructed such that the Face normal points away from the material. However, when extracting features, we must account for the fact that a "Bend" (Cylindrical Face) shares edges with "Flanges" (Planar Faces). The orientation of the shared edge will be FORWARD in one face and REVERSED in the other. This reciprocal relationship is the fundamental mechanism by which topological adjacency is mathematically defined and detected.[5]

2.2 The Geometric/Topological Duality

The Face-Adjacency Graph (FAG) is essentially a dual graph of the topological mesh, but with semantic filtering. In the raw topology, a "Bend" is a Face, just like a "Flange." In the FAG, a "Bend" is reduced to an attribute of an Edge connecting two Node-Flanges. This dimensional reduction—collapsing a 2D surface (bend) into a 1D logical link—requires precise geometric classification.

We categorize faces based on their underlying Geom_Surface:

- **Nodes (Flanges):** Defined by Geom_Plane. These represent the rigid bodies of the sheet metal that are manipulated.
- **Edges (Bends):** Defined by Geom_CylindricalSurface (or Geom_ConicalSurface for tapered bends). These represent the zones of plastic deformation.
- **Artifacts:** Faces that do not fit these categories (e.g., B-Splines representing stamped forms or louvers) require special handling or flagging as "non-unfoldable" features.

The relationship between the Topological Face and the Geometric Surface is managed via the BRep_Tool class. It is imperative to handle Geom_RectangularTrimmedSurface and

Geom_OffsetSurface wrappers, which OCCT frequently adds to the basis surfaces during Boolean operations or import.[6]

3. Data Ingestion and Sanitation Strategy

Real-world STEP files are rarely mathematically perfect. They originate from diverse CAD systems (SolidWorks, Catia, Inventor), each with different tolerance settings and export kernels. A robust CAM kernel must therefore implement a "Sanitation Phase" before any graph construction begins.

3.1 Handling "Dirty" Geometry with ShapeFix

Imported shapes often suffer from tolerance issues, such as micro-gaps between faces or edges that are effectively disjoint but visually connected. Using the ShapeFix package is mandatory to normalize the topology.[8] The ShapeFix_Shape class provides a comprehensive healing pipeline:

1. **Small Edge Removal:** Converting edges shorter than the manufacturing tolerance (e.g., 0.01mm) into vertices.
2. **Wire Closure:** Ensuring all wires bounding a face are topologically closed loops.
3. **3D/2D Consistency:** Synchronizing the 3D curves of edges with their 2D parametric curves (p-curves) on the faces.

A specifically challenging artifact in panel bending is the "split face." Designers often model a single flange as multiple adjacent rectangular faces, or the export process tessellates a plane into triangles. If left untreated, these split faces will appear as distinct nodes in the FAG, creating "phantom bends" with zero degrees and zero radius.

3.2 Unifying Same Domain Faces

To resolve split faces, we employ the ShapeUpgrade_UnifySameDomain algorithm.[10] This tool detects adjacent faces that share the same underlying geometric surface and eliminates the internal boundary edges, fusing them into a single topological face.

The integration of this step is critical for the semantic validity of the FAG. By unifying same-domain faces, we ensure a one-to-one mapping between a "Logical Flange" and a "Graph Node." The C++ implementation must configure this tool to unify both faces and edges while maintaining the shape's history if downstream operations require tracking back to the original entity IDs.

4. Efficient Traversal Strategy: The Linear Time Solution

The core computational challenge in constructing the FAG is determining adjacency. A naive

algorithm iterates through every face F_i and compares it against every other face F_j to check for shared edges. This yields a time complexity of $O(N^2)$, where N is the number of faces. While acceptable for simple brackets ($N < 50$), complex server chassis or electrical enclosures can typically contain $N > 500$ faces. An $O(N^2)$ approach becomes a performance bottleneck, potentially freezing the UI during import.

4.1 The TopExp::MapShapesAndAncestors Advantage

To achieve $O(N)$ linear complexity, we must invert the traversal direction. Instead of asking "What faces are near Face A?", we ask "Which faces share Edge E?". OpenCASCADE provides the TopExp package for this exact purpose, specifically the MapShapesAndAncestors function.[1]

This function iterates through the topology once and populates a hash map (TopTools_IndexedDataMapOfShapeListOfShape).

- **Key:** The sub-shape (in our case, TopAbs_EDGE).
- **Value:** A list of ancestors (in our case, TopAbs_FACE).

The internal mechanism uses the HashCode of the topological pointers to ensure $O(1)$ average access time for lookups.[1][2] By building this map upfront, we transform the adjacency query from a geometric search into a fast memory lookup.

4.2 Algorithm Complexity Analysis

The table below compares the naive approach with the map-based approach:

Feature	Naive Iteration	Map-Based Approach
Complexity	$O(N^2 \cdot$	$O(N \cdot$
Mechanism	Geometric intersection or exhaustive pointer comparison	Hash Map Lookup
Scalability	Poor (Degrades exponentially with part complexity)	Excellent (Linear scaling)

Memory Usage	Minimal	Moderate (Requires storage for the Map)
Implementation	Simple loops	Requires TopTools data structures

Given that modern workstations have ample RAM, the memory trade-off for the Map-based approach is negligible compared to the CPU time savings.

4.3 Traversal Logic

The optimal traversal logic follows these steps:

1. **Map Construction:** Call TopExp::MapShapesAndAncestors(Shape, TopAbs_EDGE, TopAbs_FACE, Map).
2. **Face Iteration:** Iterate over all faces in the shape to identify "Bend Candidates" (Cylindrical Faces).
3. **Edge Query:** For each edge of a Bend Candidate, lookup the Map.
4. **Adjacency Detection:** If the Map returns a list containing exactly two faces—the Bend Candidate itself and a Planar Face—we have established a link.

This logic avoids ever comparing two non-adjacent faces, strictly limiting processing to actual physical connections.

5. Architectural Design of the Face-Adjacency Graph

The FAG is an abstraction layer that sits above the B-Rep. It requires a dedicated C++ data structure that holds references to the OCCT objects while augmenting them with manufacturing data (Bend Angle, Radius, K-Factor).

5.1 C++ Data Structures

We propose using a struct-based architecture with smart pointers or handles to manage the graph. The use of raw pointers to TopoDS_Shape objects is safe only if the underlying TopoDS_Shape is kept in scope; however, it is safer to store the TopoDS_Face and TopoDS_Edge objects directly, as they are essentially handles themselves (smart pointers to an internal TShape).

5.1.1 Graph Node: The Planar Face

The Node represents a localized region of the sheet that remains undeformed.

C++

```
#include <TopoDS_Face.hxx>
#include <gp_Pln.hxx>
#include <gp_Dir.hxx>

// Represents a Node in the FAG (A Planar Flange)
struct FAG_Node {
    int NodeID;             // Unique identifier for graph algorithms
    TopoDS_Face Face;       // The underlying topological face
    gp_Pln Geometry;        // The canonical geometric plane
    gp_Dir Normal;          // Outward pointing normal
    double Area;            // Surface area for weight estimation
    bool IsGrounded;        // Flag for the stationary face (base)

    FAG_Node(int id, const TopoDS_Face& face, const gp_Pln& plane)
        : NodeID(id), Face(face), Geometry(plane), IsGrounded(false) {
        // Normal extraction logic would go here
    }
};
```

5.1.2 Graph Edge: The Connection

The Edge represents the relationship between two Nodes. This can be a physical bend (Cylinder) or a sharp connection (Shared Edge).

C++

```
#include <TopoDS_Face.hxx>
#include <TopoDS_Edge.hxx>

enum ConnectionType {
    CONN_CYLINDRICAL_BEND, // Standard bend with radius
    CONN_SHARP,            // Zero-radius bend (hems or simplified geometry)
    CONN_OTHER             // Unknown/Complex connection
};

// Represents an Edge in the FAG (A Bend or Connection)
struct FAG_Edge {
    int EdgeID;
```

```
    int NodeID_1;       // ID of the first connected flange
    int NodeID_2;       // ID of the second connected flange
    ConnectionType Type;

    // Attributes for Cylindrical Bends
    TopoDS_Face BendFace;  // The cylindrical face geometry
    double Radius;      // Bend radius
    double Angle;       // Bend angle in radians (supplementary)
    bool IsConvex;      // Bend direction relative to Z-up

    // Attributes for Sharp Connections
    TopoDS_Edge CommonEdge;// The shared linear edge

    FAG_Edge() : EdgeID(-1), Type(CONN_OTHER), Radius(0.0), Angle(0.0) {}
};
```

5.1.3 The Graph Container

The graph itself can be modeled using an adjacency list for efficient traversal during the sequencing phase.

C++

```
#include <vector>
#include <map>
#include <memory>

class FaceAdjacencyGraph {
public:
    std::vector<std::shared_ptr<FAG_Node>> Nodes;
    std::vector<std::shared_ptr<FAG_Edge>> Edges;

    // Adjacency List: NodeID -> List of Connected EdgeIDs
    std::map<int, std::vector<int>> AdjacencyMap;

    void AddNode(const std::shared_ptr<FAG_Node>& node) {
        Nodes.push_back(node);
    }

    void AddEdge(const std::shared_ptr<FAG_Edge>& edge) {
        Edges.push_back(edge);
```

```
        AdjacencyMap.push_back(edge->EdgeID);
        AdjacencyMap.push_back(edge->EdgeID);
    }
};
```

6. Detailed Implementation Strategy

The construction of the graph proceeds in distinct phases: Geometry Classification, Map Construction, and Linkage.

6.1 Phase 1: Geometry Classification

The first pass iterates over all faces in the healed shape to categorize them. We utilize BRep_Tool::Surface to access the geometry. It is crucial to handle Geom_RectangularTrimmedSurface, which OCCT often wraps around the fundamental geometry. We must downcast this wrapper to access the BasisSurface.[5]

C++

```
// Pseudo-code for classification
std::vector<TopoDS_Face> planarFaces;
std::vector<TopoDS_Face> cylindricalFaces;

TopExp_Explorer expl(healedShape, TopAbs_FACE);
for (; expl.More(); expl.Next()) {
    TopoDS_Face F = TopoDS::Face(expl.Current());
    TopLoc_Location L;
    Handle(Geom_Surface) S = BRep_Tool::Surface(F, L);

    // Unwrap Trimmed Surfaces
    if (S->DynamicType() == STANDARD_TYPE(Geom_RectangularTrimmedSurface)) {
        S = Handle(Geom_RectangularTrimmedSurface)::DownCast(S)->BasisSurface();
    }

    if (S->DynamicType() == STANDARD_TYPE(Geom_Plane)) {
        planarFaces.push_back(F);
        // Create FAG_Node and add to graph
    } else if (S->DynamicType() == STANDARD_TYPE(Geom_CylindricalSurface)) {
        cylindricalFaces.push_back(F);
    }
```

```
}
```

6.2 Phase 2: The Seam Edge Filter

A critical nuance in OCCT is the handling of periodic surfaces. A Geom_CylindricalSurface is periodic in U (0 to 2π). When a face represents a full 360-degree cylinder (or even a partial segment in some kernels), it may contain a **Seam Edge**. This edge connects the face to itself.

If we blindly traverse the MapShapesAndAncestors map, a seam edge will return the *same* face twice (or once, depending on the map implementation logic regarding orientation). This creates a self-loop in the topology which must be filtered out, as it does not represent a connection to another flange.[13]

Filtration Logic:

When processing edges of a Cylindrical Face:

1. Check BRep_Tool::IsClosed(edge, face).[6] If true, it is a seam. Skip it.
2. Alternatively, check the ancestor map. If the ancestor list contains only the current Cylindrical Face (and size is 1 or 2 duplicates), it is an internal edge or seam. Skip it.
3. We are strictly interested in edges where the ancestor list contains exactly **two different** faces: the Cylinder and a Plane.

6.3 Phase 3: Linkage and Graph Population

This phase "collapses" the B-Rep topology into the Graph semantics.

1. **Iterate Cylindrical Faces (Bend Candidates):**
   For each cylinder, we analyze its edges. We look for exactly two valid "Connection Edges" that link to Planar Faces.
   - *Edge 1:* Connects Cylinder to Planar Face A.
   - *Edge 2:* Connects Cylinder to Planar Face B.
   - *Action:* Create a FAG_Edge of type CONN_CYLINDRICAL_BEND. Store the Cylinder Geometry, Face A ID, and Face B ID.
2. **Iterate Planar Faces (Sharp Connection Candidates):**
   Some bends have zero radius (sharp folds). These do not have an intermediate Cylindrical Face.
   For each Planar Face node, iterate its edges.
   - Query Map: If an edge connects to *another* Planar Face.
   - *Action:* Create a FAG_Edge of type CONN_SHARP.
   - *De-duplication:* Since this check is bidirectional, ensure edges are not created twice (e.g., check ID_A < ID_B).

7. Geometric Attribute Extraction

The CAM software requires more than just connectivity; it needs the physics of the bend.

7.1 Calculating Bend Radius

For CONN_CYLINDRICAL_BEND edges, the radius is extracted from the Geom_CylindricalSurface.

C++

```
Handle(Geom_CylindricalSurface) cyl = Handle(Geom_CylindricalSurface)::DownCast(surf);
double radius = cyl->Radius();
```

Note: This radius is relative to the specific face surface. In sheet metal, the "inner radius" is the critical manufacturing parameter. We must determine if the face is the "inner" (compressed) or "outer" (stretched) side. This is done by comparing the surface normal with the curvature vector. If the normal points *towards* the axis of the cylinder, it is the inner face.

7.2 Calculating Bend Angle and Convexity

The bend angle is the supplementary angle between the normals of the two connected flanges. However, a simple dot product ($\arccos(N_1 \cdot N_2)$) yields an unsigned angle, which is insufficient to distinguish between a "Mountain" (Up) and "Valley" (Down) bend.

To obtain a **Signed Angle**, we utilize the gp_Dir::AngleWithRef method or cross-product analysis.[15]

1. **Reference Axis:** The axis of the Cylinder (cyl->Axis().Direction()) provides a robust reference vector V_{ref}.
2. **Cross Product:** Calculate $V_{cross} = N_1 \times N_2$.
3. **Sign Detection:** Compare V_{cross} with V_{ref}. If they are parallel ($V_{cross} \cdot V_{ref} > 0$), the bend is convex. If anti-parallel, it is concave.

7.3 Thickness Detection

Thickness is a global parameter required for K-factor compensation.[2] It can be calculated by finding a pair of parallel planar faces with opposite normals and measuring the distance using BRepExtrema_DistShapeShape.[18] This value should be consistent across the entire part; variations imply a non-sheet-metal part or a formed feature.

8. Robustness and Error Handling

8.1 Topological Integrity Checks

The input STEP file may contain "manifold errors," such as non-manifold edges (shared by 3+ faces) often seen in T-junction welds. The FAG construction must flag these. If an edge in the ancestor map returns > 2 faces, it represents a non-sheet-metal junction. The kernel should throw a TopologyError or flag the part as requiring manual intervention.

8.2 Floating Point Precision

Computational geometry is plagued by floating-point inaccuracies. OCCT uses Precision::Confusion() (typically 10^{-7}) for equality checks. When comparing angles or points, explicit tolerance checks must be used (IsEqual(val1, val2, Tol)). Comparing geometric pointers directly is safe for topological identity, but comparing derived values (like plane coefficients) requires tolerances.

9. Conclusion

The construction of a Face-Adjacency Graph from a raw STEP file is a deterministic process when leveraging the architectural strengths of OpenCASCADE. By treating the problem as a topological traversal utilizing TopExp::MapShapesAndAncestors, we reduce the algorithmic complexity to linear time $O(N)$, ensuring scalability for complex industrial parts. The rigorous classification of geometry into "Nodes" (Planes) and "Edges" (Cylinders), combined with the careful filtering of seam edges and artifacts, yields a semantic graph rich enough to drive a Panel Bender's motion controller.

This kernel serves as the foundational layer. Upon this graph, subsequent algorithms for unfolding (flattening the graph), bend sequencing (finding a collision-free traversal path), and tool selection can be built, transforming a static 3D model into a dynamic manufacturing process.

10. References

This report synthesizes technical documentation and best practices from the OpenCASCADE technology stack. Specific function behaviors and class definitions are referenced from the official documentation and developer guides:

- [1] : TopExp::MapShapesAndAncestors usage and efficiency.
- [6] : Handling of periodic surfaces, seam edges, and BRep_Tool::IsClosed.
- [8]

: ShapeFix package for healing topology.

- [10]
  : ShapeUpgrade_UnifySameDomain for merging split faces.
- [2]
  : Sheet metal feature recognition principles and thickness calculation.
- [15]
  : Vector mathematics for signed angle calculation using gp_Dir.

Works cited

1. TopExp Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_top_exp.html
2. Sheet Metal Operations Component - Open Cascade, accessed February 2, 2026, https://www.opencascade.com/components/sheet-metal-operations/
3. TopAbs_Orientation.hxx File Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/_top_abs___orientation_8hxx.html
4. Modeling Data - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_data.html
5. Topology and Geometry in Open CASCADE. Part 6, accessed February 2, 2026, https://opencascade.blogspot.com/2009/03/topology-and-geometry-in-open-cascade.html
6. BRep_Tool Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep___tool.html
7. Correct face normal calculation - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/correct-face-normal-calculation
8. ShapeFix Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_shape_fix.html
9. 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
10. ShapeUpgrade_UnifySameDomain Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://old.opencascade.com/doc/occt-7.0.0/refman/html/class_shape_upgrade__unify_same_domain.html
11. ShapeUpgrade_UnifySameDomain Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_shape_upgrade___unify_same_domain.html
12. TopTools_IndexedDataMapOfSh, accessed February 2, 2026, https://dev.opencascade.org/content/toptoolsindexeddatamapofshapelistofshape
13. Shape Healing - Open CASCADE Technology, accessed February 2, 2026,

https://dev.opencascade.org/doc/occt-7.7.0/overview/html/occt_user_guides__shape_healing.html
14. Get Normal of an edge - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/get-normal-edge
15. gp_Dir Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/classgp___dir.html
16. gp_Dir Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/occt-6.9.1/refman/html/classgp___dir.html
17. On sheet metal unfolding (Part 4) | by Analysis Situs - Medium, accessed February 2, 2026, https://analysis-situs.medium.com/on-sheet-metal-unfolding-part-4-f9eda305c32e
18. 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