Advanced Computational Geometry Architectures for Panel Bender CAM Kernels: A Semantic Feature Recognition Framework Using OpenCASCADE Technology

1. The Paradigm of Feature Recognition in Automated Sheet Metal Fabrication

The industrial transition from manual path planning to automated Computer-Aided Manufacturing (CAM) represents one of the most significant shifts in modern fabrication. In the specific domain of Panel Bending—a technology distinct from traditional Press Braking due to its kinematic reliance on manipulating the workpiece via a blankholder and bending blades—the demand for robust, autonomous software kernels is acute. The central challenge in developing such a kernel lies in the translation of "dumb" geometric data into semantic manufacturing intelligence. This report details the architectural and algorithmic methodologies required to construct a "Geometric Kernel" capable of ingesting raw STEP files (AP203/AP214), healing topological defects, and extracting a semantic **Face-Adjacency Graph (FAG)**. This graph serves as the mathematical foundation for subsequent unfolding, collision detection, and sequence optimization algorithms.

The scope of this analysis is rigorously confined to the use of **OpenCASCADE Technology (OCCT)** as the underlying geometric modeling engine. As an open-source C++ library, OCCT provides the necessary boundary representation (B-Rep) primitives—vertices, edges, wires, faces, shells, and solids—required to interrogate and manipulate 3D CAD data. However, the raw utility of OCCT must be augmented with sophisticated high-level algorithms to handle the "dirty" nature of real-world industrial CAD models. These models, often originating from diverse upstream systems such as SolidWorks, CATIA, or Inventor, frequently exhibit topological incoherencies: microscopic gaps, split faces resulting from imperfect Boolean operations, and ambiguous geometric definitions like zero-radius bends.[1]

This report addresses three critical technical pillars:

1. **Topological Healing:** The management of gap tolerances and the mathematical "sewing" of disjointed surfaces.
2. **Canonical Form Restoration:** The merging of split logical flanges into unified topological entities.
3. **Virtual Feature Extraction:** The detection and semantic classification of zero-radius

bends (sharp edges) as functional bending nodes within the graph.

1.1 The Disconnect Between B-Rep and Manufacturing Intent

A fundamental dichotomy exists between the Boundary Representation (B-Rep) of a part and its manufacturing reality. In a B-Rep scheme, a sheet metal part is defined by its boundaries: surfaces bounded by curves, curves bounded by points. A "bend" in this representation is merely a cylindrical surface (or a sharp edge in simplified models) connected to two planar surfaces. The B-Rep contains no intrinsic knowledge that this cylinder represents a plastic deformation zone, nor does it know that the two adjacent planes are flanges that must be manipulated by the machine's grippers.[1]

For a Panel Bender, this lack of semantic data is catastrophic. The machine requires a structured understanding of the part:

- **Flanges:** Rigid zones used for gripping and referencing.
- **Bends:** Zones of deformation with specific attributes (angle, radius, k-factor, deduction).
- **Topological Connectivity:** The sequence in which flanges are connected, determining the unfolding tree.

The "Geometric Kernel" acts as the translation layer. It must ingest the geometric "syntax" (B-Rep) and output the manufacturing "semantics" (FAG). This process is complicated by the quality of input data. A STEP file is a snapshot of geometry, stripped of the parametric history tree that created it. Consequently, the kernel must perform "Feature Recognition" (FR) to reconstruct the design intent from the bottom up.[1]

1.2 The Face-Adjacency Graph (FAG) as a Semantic Structure

The target output of the kernel is the Face-Adjacency Graph, formally defined as a graph $G = (V, E)$.

- V (Vertices) represents the logical flanges of the part.
- E (Edges) represents the bends (physical or virtual) connecting these flanges.

This graph structure is chosen because it is invariant to the spatial orientation of the part and provides a direct mapping to the unfolding process. Unfolding a 3D part into a 2D flat pattern is mathematically equivalent to computing a spanning tree of the FAG and applying coordinate transformations along the edges of the tree.[3] However, constructing this graph from a raw STEP file is not a trivial mapping operation. It requires a pipeline of geometric sanitization and analysis to ensure that V truly represents logical flanges (not fragmented faces) and E accurately captures all bending features (not just explicit cylinders).

---

2. Topological Remediation: The Mathematics of Sewing and Healing

The first hurdle in processing industrial CAD data is the "Face Soup" phenomenon. A valid solid model in a CAD system is mathematically "watertight" or "manifold," meaning it separates 3D space into a distinct "inside" and "outside." However, during translation to neutral formats like STEP, numerical precision errors often introduce gaps between faces.

These gaps, often in the range of 10^{-4} to 10^{-2} millimeters, break the topological connectivity of the model.

If the kernel were to attempt Feature Recognition on a model with gaps, graph traversal would fail. The algorithm would interpret the gap as a boundary of the sheet, seeing two disconnected parts instead of one continuous bent sheet. Therefore, the kernel must implement a robust "Sewing" or "Healing" phase using BRepBuilderAPI_Sewing.[4]

2.1 Algorithmic Mechanics of BRepBuilderAPI_Sewing

The BRepBuilderAPI_Sewing algorithm is not a boolean operation; it is a topological reconciliation process. It operates by identifying "free edges"—edges that belong to only one face—and attempting to pair them with other free edges that are geometrically coincident within a specified tolerance.[5]

The sewing process proceeds in distinct stages:

1. **Vertex Analysis:** The algorithm tessellates the edges and compares their vertices. If vertices from different edges lie within the tolerance sphere, they are merged into a single vertex.
2. **Edge Analysis:** Edges connecting the merged vertices are compared. If their underlying curves are within tolerance along their entire length, the edges are merged.
3. **Topology Update:** The faces are updated to reference the new, shared edges, effectively "stitching" the surface together.

2.1.1 Tolerance Management and Strategy

The single most critical parameter in this phase is the **tolerance**. A naive approach might set a global tolerance of 0.1mm to ensure all gaps are closed. However, this poses severe risks in sheet metal processing.[4]

Tolerance Strategy	Consequence	Risk Level

Micro (10^{-6} mm)	Fails to close common translation gaps. Model remains a disjoint "soup."	High (Process Failure)
Standard (10^{-3} mm)	Closes typical STEP artifacts. Preserves manufacturing features.	Optimal
Macro ($>$ mm)	Closes gaps but also collapses valid relief cuts, laser slits, and narrow slots.	High (Geometry Corruption)

The recommended implementation strategy involves a "progressive sewing" approach or a carefully selected baseline tolerance derived from the model's bounding box or expected precision.[5]

C++

```
// C++ Implementation Strategy using BRepBuilderAPI_Sewing
// Define a tolerance suitable for manufacturing (e.g., 0.01mm)
Standard_Real sewingTolerance = 1.0e-02;

BRepBuilderAPI_Sewing sewer(sewingTolerance);

// Configuration for robustness
sewer.SetNonManifoldMode(Standard_False); // Sheet metal should be manifold
sewer.SetOption1(Standard_True); // Enable connexity
sewer.SetOption2(Standard_True); // Enable analysis of degenerated shapes
sewer.SetOption3(Standard_True); // Cutting of free edges (Use with caution!)

// Load the dirty shape
sewer.Load(rawStepShape);

// Execute the algorithm
sewer.Perform();

// Retrieve the result
TopoDS_Shape sewnShape = sewer.SewedShape();
```

2.2 Manifold vs. Non-Manifold Anomalies

While sheet metal is physically a manifold object (locally homeomorphic to a disk), "dirty" sewing can introduce non-manifold artifacts. For instance, if three faces are sewn to a single edge (a T-junction), the topology becomes non-manifold. This is mathematically invalid for a solid body and will cause downstream algorithms (like offset or unfold) to crash.[5]

The kernel must rigorously validate the output of the sewing process. The BRepAlgoAPI_Check or BRepCheck_Analyzer classes should be employed to verify the orientation and connectivity of the resulting shell.[7] If the sewing results in a TopoDS_Shell that is not closed (i.e., has free boundaries other than the sheet periphery), the kernel must decide whether to attempt a second pass with a higher tolerance or flag the part for manual intervention.

2.3 Handling Free Edges and "Cutting"

The BRepBuilderAPI_Sewing class includes an option (option3) for "cutting of free edges".[4] This feature attempts to trim edges that protrude beyond the sewing boundary. In the context of Panel Bending, this feature must be used with extreme caution. A "protruding edge" might actually be a small tab or a valid feature of the flat pattern. Indiscriminate cutting can alter the dimensions of the part. It is generally safer to disable this option and rely on ShapeFix_Wire to heal the individual face boundaries post-sewing.[8]

---

3. Surface Unification and Canonical Form Restoration

Once the model is topologically connected, the next challenge is "Topological Noise." A common issue in STEP files exported from systems like Pro/Engineer or SolidWorks is the fragmentation of logical faces. A single planar flange might be represented as two or three adjacent triangular or rectangular faces. Similarly, a cylindrical bend might be split into two 180-degree segments or multiple strips.[10]

This fragmentation is problematic for the FAG. If a single physical flange is represented by three faces, the graph will contain three nodes where there should be one. This complicates the bending logic, as the algorithm might attempt to find a bend between two coplanar fragments of the same flange. Therefore, a "Normalization" or "Unification" phase is required.

3.1 The Unification Algorithm: ShapeUpgrade_UnifySameDomain

OCCT provides a specialized tool for this purpose: ShapeUpgrade_UnifySameDomain. This algorithm detects adjacent faces that share the same underlying geometric definition and fuses them into a single topological face.[11]

The unification process involves:

1. **Geometric Analysis:** Checking if the underlying surfaces of adjacent faces are identical (e.g., coplanar with the same normal and offset, or coaxial cylinders with the same radius).
2. **Edge Removal:** If the surfaces match, the shared "seam" edges are redundant. The algorithm removes these edges.
3. **Wire Reconstruction:** The outer boundaries of the merged faces are connected to form a new, single wire for the unified face.

3.1.1 Implementation and Configuration

To ensure robust handling of dirty geometry, the unifier must be configured to handle not just faces, but also edges and BSplines.

C++

```
// merging split faces
ShapeUpgrade_UnifySameDomain unifier;
// Initialize with the sewn shape and enable all unification modes
// UnifyFaces = True
// UnifyEdges = True
// ConcatBSplines = True
unifier.Initialize(sewnShape, Standard_True, Standard_True, Standard_True);

// Allow internal edges to be removed to clean up the face
unifier.AllowInternalEdges(Standard_False);

// Set tolerances for linear and angular comparisons
unifier.SetLinearTolerance(1.0e-4);
unifier.SetAngularTolerance(1.0e-5);

// Build the unified shape
unifier.Build();
TopoDS_Shape optimizedShape = unifier.Shape();
```

Ref: [11]

3.2 Handling Cylindrical Split Faces

A specific nuisance in sheet metal is the splitting of cylindrical bends. A 90-degree bend might be split into two 45-degree segments. The UnifySameDomain tool is effective here, but

it requires that the underlying surfaces be truly coaxial. If the upstream CAD system approximated the bend with a generic NURBS surface rather than a canonical Geom_CylindricalSurface, the unifier might fail to recognize them as identical.

In such cases, a pre-processing step using ShapeFix_Face to convert NURBS surfaces to analytical forms (Plane, Cylinder) where possible is recommended. This conversion is often referred to as "canonical recognition".[7]

3.3 Preserving Functional Topology

While the goal is to merge faces, certain edges must be preserved. For example, a "scribe line" (a line etched onto the part for alignment) might split a face but should not be removed if it carries manufacturing information. The ShapeUpgrade_UnifySameDomain::KeepShape() method allows the kernel to explicitly exempt specific shapes from unification.[12] However, for the specific purpose of generating the FAG for bending, we generally desire maximum unification. Scribe lines are better handled as auxiliary geometry overlaid on the flange, rather than topological boundaries that split the flange.

---

4. The Geometry of Bends: Virtual Feature Extraction

With a clean, unified topology, the kernel proceeds to the core of the Feature Recognition task: classifying the topological entities into manufacturing features. The most complex aspect of this is the detection of **Zero-Radius Bends** (also known as Virtual Bends or Sharp Bends).

In theoretical sheet metal design, bends always have a radius determined by the tooling. In practice, designers often model sharp corners ($R = 0$) for speed. The kernel must treat a sharp edge between two flanges exactly as if it were a cylindrical bend face. It must instantiate a "Bend Edge" in the FAG with a radius of 0.0.[14]

4.1 Dihedral Angle Analysis

The primary discriminator for a zero-radius bend is the **Dihedral Angle** between the adjacent faces.

- **Angle** $\approx 0^\circ$: The faces are coplanar. They should have been unified. If they remain, it implies a step or a seam that UnifySameDomain could not heal.
- **Angle** $\approx$: The faces are folded back on themselves (a Hem or Dutch Bend).[1]
- $0^\circ < \text{Angle} < 180^\circ$: The edge represents a bend.

4.1.1 Robust Angle Calculation Algorithm

Calculating the angle between two faces in a B-Rep is non-trivial due to the concept of **Face Orientation**. In OCCT, a TopoDS_Face has an orientation (TopAbs_FORWARD or TopAbs_REVERSED) relative to its underlying geometric surface.[16]

- If Orientation is FORWARD, the topological normal aligns with the geometric surface normal.
- If Orientation is REVERSED, the topological normal is opposite to the geometric surface normal.

To calculate the physical angle between the material of two flanges, one must use the *topological* normals, not just the raw surface normals.

Algorithm:

1. Identify an edge E shared by Face F_1 and Face F_2.
2. Compute the midpoint P of edge E.
3. Project P onto the surfaces of F_1 and F_2 to find parameters (u_1, v_1) and (u_2, v_2).
4. Calculate the surface normals $\mathbf{n}_{s1}$ and $\mathbf{n}_{s2}$ at these parameters using BRepGProp_Face or GeomLProp_SLProps.[18]
5. Apply orientation correction:

$$\mathbf{n}_{f1} = (\text{F1.Orientation}() == \text{REVERSED})? -\mathbf{n}_{s1} : \mathbf{n}_{s1}$$

$$\mathbf{n}_{f2} = (\text{F2.Orientation}() == \text{REVERSED})? -\mathbf{n}_{s2} : \mathbf{n}_{s2}$$

6. Compute the angle θ:

$$\theta = \pi - \arccos(\mathbf{n}_{f1} \cdot \mathbf{n}_{f2})$$

Note: The use of $\pi - \arccos$ calculates the internal bend angle. A dot product of 0 (perpendicular normals) yields $\pi - \pi/2 = \pi/2$ (90°)..[20]

4.2 Convexity Detection: The Cross Product Method

Determining the angle magnitude is insufficient; the kernel must also determine the **direction** of the bend (Up vs. Down). In computational geometry terms, this is the distinction between a **Convex** edge and a **Concave** edge.[21]

For a solid body representing sheet metal, "Convex" edges typically represent the outer radius of a bend, while "Concave" edges represent the inner radius. However, for zero-radius bends, there is no thickness. The "convexity" defines the folding direction relative to the observer.

The Cross-Product Algorithm:

1. Retrieve the tangent vector $\mathbf{t}$ of Edge E at the midpoint.
   - *Critical:* The tangent must be oriented consistent with the traversal direction of Face F_1. If E has REVERSED orientation in the wire of F_1, the tangent vector must be flipped.[21]
2. Calculate the cross product of the face normals:

$$\mathbf{c} = \mathbf{n}_{f1} \times \mathbf{n}_{f2}$$

3. Compute the dot product with the tangent:

$$\delta = \mathbf{c} \cdot \mathbf{t}$$

4. **Classification:**
   - If $\delta > 0$, the edge is **Convex** (Mountain Fold).
   - If $\delta < 0$, the edge is **Concave** (Valley Fold).

This classification is stored in the FAG edge attributes and is essential for generating the correct NC code (e.g., instructing the Panel Bender to bend Up or Down).[21]

4.3 Distinguishing Thickness Faces

A robust kernel must distinguish between "Flange Faces" (the major surfaces) and "Thickness Faces" (the thin strips along the edges). Thickness faces should not become nodes in the FAG; otherwise, the graph would look like a ladder rather than a tree, with connections traversing the thickness of the sheet.

Identification Heuristics:

1. **Area Threshold:** Thickness faces typically have a much smaller area than flanges.
2. **Connectivity:** Thickness faces connect the "Top" skin to the "Bottom" skin.
3. **Ray Casting:** A ray cast from a candidate face along its inverted normal should hit another face at a distance equal to the material thickness (T). If the distance is much larger, it is likely a flange. If it is T, and the hit face has an opposing normal, the pair constitutes a Top/Bottom flange pair, and the faces connecting them are thickness faces.[24]

Implementation Note: BRepExtrema_DistShapeShape can be used to accurately measure the distance between parallel faces to validate thickness.[25]

---

5. Semantic Graph Construction and Traversal

The synthesis of the geometry (healed and unified) and the analysis (angles and convexity) culminates in the construction of the Face-Adjacency Graph (FAG).

5.1 Graph Data Structures: Boost Graph Library (BGL)

While custom pointer-based structures can implement graphs, the **Boost Graph Library (BGL)** is the industry standard for C++ implementations due to its performance and rich algorithm set (Dijkstra, Connected Components, Topological Sort).[27]

The FAG is modeled as an undirected graph where properties are attached to both vertices and edges.

Graph Element	Semantic Equivalent	Attributes Stored
Vertex	Logical Flange	TopoDS_Face Handle, Surface Area, Normal Vector, IsThickness (bool)
Edge	Bend Connection	Connection Type (Physical/Virtual), Angle, Radius, K-Factor, Convexity (bool)

BGL Definition:

C++

```
// Defining the Graph using Boost
struct FaceProperty {
    int id;
    TopoDS_Face shape;
    gp_Dir normal;
    bool isThickness;
```

```
};

struct EdgeConnectionProperty {
    bool isVirtualBend; // True for Zero-Radius
    double angle;       // Radians
    double radius;      // 0.0 for Virtual
    bool isConvex;      // Up/Down direction
};

typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::undirectedS,
                              FaceProperty, EdgeConnectionProperty> FaceAdjacencyGraph;
```

5.2 Construction Algorithm

The construction proceeds by iterating over the map of shared edges generated by TopExp::MapShapesAndAncestors.[17]

1. **Map Edges to Faces:** Create a map where every edge points to the list of faces that share it.
2. **Filter Boundaries:** Ignore edges with only 1 face (outer boundaries, assuming valid sewing).
3. **Process Connections:** For every edge shared by Face A and Face B:
    - **Case 1: Face A and B are Planar.**
        - Calculate Dihedral Angle.
        - If Angle > Threshold: Create a **Virtual Bend Edge** in FAG between Node A and Node B.
    - **Case 2: Face A is Planar, Face B is Cylindrical.**
        - This represents a physical bend.
        - Traverse across Face B to find the *other* Planar Face (Face C).
        - Create a **Physical Bend Edge** in FAG between Node A and Node C.
        - The cylindrical Face B becomes an *attribute* of the edge, not a node itself (to maintain the dual-graph structure where nodes are flanges).
        - *Optimization:* If the bend is split into multiple cylindrical faces, the traversal must continue until a planar face is found, collapsing the chain of cylinders into a single semantic bend.[3]

5.3 Unfolding and Flattening

The FAG allows for the rapid generation of the flat pattern without complex geometric unrolling operations on the solid. Unfolding is treated as a spanning tree problem.

1. **Root Selection:** Select the largest planar face as the "Base."
2. **Tree Traversal:** Perform a Breadth-First Search (BFS) from the Root.
3. **Transformation:** For each edge traversed:

    - Retrieve the Bend Angle θ and Convexity.
    - Calculate the transformation matrix M that rotates the child flange by $-\theta$ around the bend axis.
    - Apply a translation compensation (Bend Deduction) based on the material's K-factor.[1]
    - Accumulate transformations down the tree.

This semantic unfolding is orders of magnitude faster than geometric solid unfolding because it operates on rigid transforms of face abstractions rather than deforming NURBS surfaces.[2]

---

6. Architectural Implementation in C++ with OpenCASCADE

The implementation of this kernel requires a modular C++ architecture that separates data ingestion, healing, analysis, and graph management.

6.1 Module Structure

1. **Ingestion Module:**
    - Wraps STEPControl_Reader.[29]
    - Handles unit conversion (OCCT defaults to Millimeters).
    - Performs initial validity checks (BRepCheck_Analyzer).
2. **Healing Module:**
    - Encapsulates BRepBuilderAPI_Sewing.
    - Implements the "Progressive Sewing" logic (try 10^{-4}, then 10^{-3}, etc.).
    - Wraps ShapeUpgrade_UnifySameDomain.
3. **Feature Recognition Module:**
    - Implements the math for Normal extraction and Dihedral Angle calculation.
    - Contains the heuristics for Thickness Face detection.
    - Classifies faces into PLANAR_FLANGE, CYLINDRICAL_BEND, THICKNESS_WALL.
4. **Graph Module:**
    - Manages the BGL data structures.
    - Provides API for high-level queries (GetNeighbors, GetBendSequence).

6.2 Performance Considerations

- **Spatial Indexing:** When performing operations like ray casting or distance checking (BRepExtrema_DistShapeShape), using Bounding Volume Hierarchies (BVH) is essential. OCCT's BRepBndLib allows rapid computation of bounding boxes to cull non-interacting faces before performing expensive geometric checks.[30]

- **Parallelism:** OCCT supports parallel processing in some algorithms. However, the graph construction is largely sequential. Threading can be introduced during the "Face Classification" phase, where the analysis of each face (normal, area, type) is independent and can be parallelized using OSD_Parallel or standard C++ threads.

6.3 Robustness against Dirty Geometry

The kernel must be defensive.

- **Try/Catch Blocks:** OCCT geometry algorithms can throw exceptions (Standard_Failure) on degenerate geometry (e.g., zero-length edges). All geometric calls should be guarded.
- **Null Checks:** Algorithms like BRep_Tool::Surface can return null handles if the topology is corrupt.
- **Fallback Logic:** If UnifySameDomain fails to merge a split face (e.g., due to tolerance issues), the Graph Builder must be able to handle "Split Nodes" (two nodes with a 0-degree, 0-radius connection) and treat them effectively as a single rigid body during unfolding.

---

7. Conclusion

The development of a Geometric Kernel for Panel Bender CAM is a sophisticated exercise in Computational Geometry. It requires moving beyond the passive reading of B-Rep data to an active reconstruction of manufacturing intent. By leveraging OpenCASCADE Technology, we utilize a robust foundation of geometric primitives. However, the success of the kernel lies in the layer built *above* OCCT: the **Sewing strategies** that close the gaps of imperfect exports, the **Unification algorithms** that restore the canonical simplicity of flanges, and the **Vector Analysis** that renders invisible zero-radius bends into actionable machine instructions.

The **Face-Adjacency Graph** constructed through this pipeline provides the semantic abstraction necessary for automation. It decouples the complexity of 3D topology from the logic of path planning, enabling the CAM software to reason about "Bends" and "Flanges" rather than "Cylindrical Surfaces" and "Trimmed Planes." This abstraction is the key to robust, efficient, and intelligent automated manufacturing.

Data Summary Tables

Table 1: Algorithm Selection for Geometric Tasks

Task	OCCT Class/Method	Critical Parameters	Recommended Value

Healing Gaps	BRepBuilderAPI_Sewing	Tolerance	10^{-3} mm (Start)
Merging Faces	ShapeUpgrade_UnifySameDomain	UnifyFaces, UnifyEdges	Standard_True
Surface Normal	BRepGProp_Face::Normal	(u, v) Parameters	Center of Face
Face Area	BRepGProp_Face::Mass	Epsilon	10^{-6}
Distance Check	BRepExtrema_DistShapeShape	Value()	N/A
Wire Fixing	ShapeFix_Wire	Precision	10^{-6}

Table 2: Feature Classification Logic

Feature Type	Geometric Signature	Topological Signature	FAG Representation
Flange	Planar Surface	Area $>$ Thickness Threshold	Node (Vertex)
Physical Bend	Cylindrical Surface	Connects 2 Planar Faces	Edge (Attribute)
Virtual Bend	Sharp Edge (C^0)	Connects 2 Planar Faces, Angle $\neq$	Edge (Virtual)
Hem (Crushed)	Sharp Edge (C^0)	Angle $\approx$	Edge Attribute
Thickness Face	Planar/Curved	Area $<$ Threshold, Perpendicular to	Ignored / Attribute

		Flange	

Citations: .[1]

Works cited

1. Sheet Metal Operations Component - Open Cascade, accessed February 2, 2026, https://www.opencascade.com/components/sheet-metal-operations/
2. Bending Simulation Framework for Rapid Feasibility Checks of Sheet Metal Parts - CAD Journal, accessed February 2, 2026, https://www.cad-journal.net/files/vol_23/CAD_23(1)_2026_85-100.pdf
3. 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
4. BRepBuilderAPI_Sewing Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_builder_a_p_i___sewing.html
5. Open CASCADE Technology: Modeling Algorithms, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_algos.html
6. STEP Translator - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__step.html
7. 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
8. Shape Healing - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__shape_healing.html
9. ShapeFix_Shape Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_shape_fix___shape.html
10. Volume differs between identical models - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/volume-differs-between-identical-models
11. 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
12. 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

13. Merging cylindrical faces using ShapeUpgrade_UnifySameDomain, accessed February 2, 2026, https://dev.opencascade.org/content/merging-cylindrical-faces-using-shapeupgradeunifysamedomain
14. Sheet Metal Bends with zero radius - BricsCAD Mechanical - YouTube, accessed February 2, 2026, https://www.youtube.com/watch?v=ulMb6yy3rOQ
15. Finding connected faces of a CAD model - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/finding-connected-faces-cad-model
16. TopoDS_Face with REVERSED orientation - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/topodsface-reversed-orientation
17. TopAbs_Orientation.hxx File Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/_top_abs___orientation_8hxx.html
18. BRepGProp_Face Class Reference - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_g_prop___face.html
19. Edge Concavity or Convexity - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/edge-concavity-or-convexity
20. How to get the dihedral angle of adjoin faces - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/content/how-get-dihedral-angle-adjoin-faces
21. Calculating the attribute of a edge - Forum Open Cascade Technology, accessed February 2, 2026, https://dev.opencascade.org/content/calculating-attribute-edge
22. Identify concave/convex Edge - SIEMENS Community, accessed February 2, 2026, https://community.sw.siemens.com/s/question/0D54O000061xNbRSAU/identify-concaveconvex-edge
23. Draw Test Harness - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__test_harness.html
24. 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
25. 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
26. Tutorial - Open CASCADE Technology, accessed February 2, 2026, https://old.opencascade.com/doc/occt-7.5.0/overview/html/occt__tutorial.html
27. The Boost Graph Library, accessed February 2, 2026, https://www.boost.org/libs/graph
28. CXXGraph vs Boost Graph Library: The Complete 2025 Comparison Guide, accessed February 2, 2026,

https://dev.to/zigrazor/cxxgraph-vs-boost-graph-library-the-complete-2025-comparison-guide-4n4
29. How to merge faces in open cascade so that the faces are meshed together, accessed February 2, 2026, https://dev.opencascade.org/content/how-merge-faces-open-cascade-so-faces-are-meshed-together
30. Modeling Data - Open CASCADE Technology, accessed February 2, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_data.html
31. Topology and Geometry in Open CASCADE. Part 4, accessed February 2, 2026, https://opencascade.blogspot.com/2009/02/continued.html