Algorithmic Reconstruction of Sheet Metal Semantics: A Computational Geometry Framework for Automated CAM

1. Executive Summary

The modern sheet metal fabrication industry stands at a critical technological intersection where the precision of Computer-Aided Design (CAD) meets the automation of Computer-Aided Manufacturing (CAM). A persistent bottleneck in this workflow is the "semantic gap" introduced by interoperability standards. While native CAD formats retain the procedural history of a part—defining features such as flanges, bends, and cutouts—neutral exchange formats like STEP (Standard for the Exchange of Product model data) reduce these intelligent models to "dumb" Boundary Representations (B-Rep). For advanced panel bending systems, such as the Salvagnini P4 or P2 lines utilizing STREAMBEND software, the ability to programmatically resurrect this lost intelligence is a prerequisite for automation. The machine cannot bend a "cylindrical surface"; it bends a "flange" relative to a "base face" by a specific "angle."

This report provides an exhaustive technical analysis of the algorithmic methodologies required to reverse-engineer feature recognition logic using OpenCASCADE Technology (OCCT). We explore the graph-theoretical underpinnings of B-Rep analysis, specifically the construction of Attributed Adjacency Graphs (AAG) and Face Transition Graphs (FTG) to map topological relationships. The core of the report details the heuristic and deterministic algorithms necessary to distinguish "Bend Features" from standard cylindrical surfaces and to identify the optimal "Base Face" for clamping operations, adhering to the kinematic constraints of panel benders. By synthesizing computational geometry principles with practical CAM constraints, we derive a robust logic for feature recognition. Furthermore, we provide a concrete implementation strategy using pythonocc, validating the theoretical concepts with executable logic. This document serves as a comprehensive blueprint for computational geometry engineers tasked with bridging the gap between static geometry and dynamic fabrication instructions.

2. Introduction: The Semantic Gap in CAM Automation

2.1 The Interoperability Paradox

In the lifecycle of a sheet metal product, data is the currency of production. Design engineers utilize parametric modeling software (e.g., SolidWorks, Inventor, CATIA) to create sophisticated assemblies. These tools store data as a "Construction History"—a sequential recipe of operations (e.g., "Sketch Rectangle," "Extrude Base," "Edge Flange," "Cut Extrude"). This history inherently contains the semantic intent of the designer; the software "knows" that a specific cylindrical face is a bend because it was created using a bend feature.

However, the global supply chain necessitates a common language. The STEP format (ISO 10303) serves this purpose by standardizing the geometric description of the product. In doing so, it strips away the construction history, leaving only the explicit boundary representation (B-Rep): a collection of faces, edges, and vertices.[1] When this STEP file reaches the manufacturing floor, the CAM software (e.g., Salvagnini STREAMBEND) sees only a topology of connected surfaces. It does not explicitly see "bends" or "flanges."

This creates the "Interoperability Paradox": to achieve the highest level of automation (Industry 4.0), we must rely on data formats that strip away the very intelligence required to drive that automation. Consequently, the burden of intelligence shifts from the file format to the intake algorithm of the CAM system. The CAM software must function as an expert system, visually "parsing" the geometry to reconstruct the designer's intent, a process known as Feature Recognition (FR).

2.2 The Role of Salvagnini STREAMBEND

Salvagnini's STREAMBEND represents the state-of-the-art in this domain. Unlike traditional press brakes where an operator might manually sequence bends, Salvagnini's panel benders (e.g., P4, P2) are fully automated work cells.[3] The machine manipulates the sheet using a central clamp and applies bends using universal tools. This automation imposes strict algorithmic requirements:

1. **Automatic Unfolding:** The system must flatten the 3D B-Rep into a 2D blank, calculating precise K-factors and bend deductions.[5]
2. **Base Face Identification:** The system must select a reference face for the manipulator clamp. This choice dictates the entire bending sequence and the kinematics of the part.[6]
3. **Collision Detection:** The system must simulate the folding process in reverse to ensure the swinging flanges do not collide with the tools or the manipulator.[1]

Reverse-engineering the logic of STREAMBEND requires us to think not just as geometers, but as kinematic planners. The recognition of a "bend" is not merely identifying a cylinder; it is identifying a kinematic hinge that transforms the state of the part.

3. Theoretical Framework: B-Rep Topology in Sheet Metal

To reverse-engineer the feature recognition logic, one must first master the data structure it consumes. The Boundary Representation (B-Rep) model is the industry standard for solid modeling, yet its application to sheet metal introduces specific topological constraints and invariants that recognition algorithms must exploit.

3.1 The Hierarchy of Topological Entities

In OpenCASCADE Technology (OCCT), the geometry kernel acts as the bedrock for our analysis. OCCT uses a hierarchical data structure TopoDS to represent shapes. Understanding this hierarchy is non-negotiable for programmatic analysis.[7]

Topological Entity	**Description**	**Sheet Metal Context**
Compound/Solid	The root entity representing the physical object.	Typically a single, manifold solid representing the finished folded part.
Shell	A collection of connected faces.	A valid sheet metal part usually consists of a single closed shell defining the "skin" of the solid.
Face	The fundamental unit of surface area, bounded by wires.	Semantically categorized into "Flanges" (planar), "Bends" (cylindrical/conical), and "Thickness" (narrow planar/cylindrical strips).
Wire	A collection of edges that form a loop.	The outer wire defines the boundary of a face; inner wires define holes or cutouts.
Edge	The curve segment connecting two vertices.	Edges act as the "hinges" in the topological graph, defining adjacency between faces.
Vertex	A zero-dimensional point in 3D space.	Defines the corners of the sheet and endpoints of bend lines.

3.2 The "Sheet Metal Manifold" Assumption

Successful feature recognition relies on the premise that the input geometry represents a valid sheet metal part. This imposes specific constraints that serve as axiomatic truths for our algorithms:

1. **Uniform Thickness (T):** The perpendicular distance between opposing faces (offset surfaces) is constant throughout the part, with localized exceptions for deformation features like countersinks or embosses.
2. **Developability:** The mid-surface of the part must be a developable surface. Mathematically, this implies that the Gaussian curvature K is zero everywhere on the mid-surface ($K = \kappa_1 \kappa_2 = 0$). This allows the part to be unrolled into a plane without stretching or tearing the material.
3. **Topological Duality:** For every "Top" face (Positive side), there exists a corresponding "Bottom" face (Negative side). For every "Inner" bend (concave), there is an "Outer" bend (convex).[2]
4. **Adjacency Logic:** Planar faces (flanges) are connected by Cylindrical faces (bends). A Planar-Planar connection usually implies a sharp edge (welded or non-bent), while a Cylindrical-Cylindrical connection implies a complex rolling offset or a multi-step bend.

3.3 Graph Theory in Computational Geometry

The raw B-Rep data structure can be traversed, but for feature recognition, it is often converted into a graph representation.

- **Face Adjacency Graph (FAG):** A graph $G = (V, E)$ where vertices V represent the faces of the solid, and edges E represent the topological adjacency (shared edges).
- **Attributed Adjacency Graph (AAG):** An extension of the FAG where arcs are weighted or labeled with geometric properties. For example, an arc might carry the attribute "Convex," "Concave," or "Tangent" depending on the dihedral angle between the faces.[8]
- **Face Transition Graph (FTG):** A derived graph used specifically in unfolding.[8] Here, "Bend" nodes are collapsed into arcs connecting "Flange" nodes. This abstraction simplifies the sheet metal part into a kinematic linkage, where nodes are rigid bodies (flanges) and arcs are revolute joints (bends).

4. Algorithmic Architecture: Recognizing the Bend Feature

The user query asks specifically: *How do we distinguish a 'Bend Feature' from a normal face?* This is the fundamental classification task. A "normal face" in this context typically refers to a planar flange, a thickness face, or a machining feature (like a hole or a fillet).

4.1 The Geometric Filter: Surface Typing

The first pass of the algorithm is purely local and geometric. We iterate through every face F_i in the model and classify its underlying geometry. OpenCASCADE’s BRepAdaptor_Surface provides the interface to query the mathematical definition of the face.[9]

- **Planar Faces (S_{plane}):** Defined by a point P and a normal vector n. These are candidates for Flanges, Webs, or Thickness faces.
- **Cylindrical Faces (S_{cyl}):** Defined by an axis A, a radius R, and a coordinate system. These are the primary candidates for Bends.
- **Conical/Toroidal Faces:** Less common but possible in complex transition parts.
- **BSpline/NURBS Faces:** Complex freeform surfaces. In "dumb" STEP files, simple cylinders are sometimes encoded as NURBS. The algorithm must attempt to convert these to analytic surfaces (canonical recognition) to verify if they are effectively cylindrical.[5]

Step 1: Filter faces where SurfaceType == GeomAbs_Cylinder.

This provides a set of *Candidate Bends*. However, this set also includes cylindrical holes, round cutouts, and machined fillets.

4.2 The Topological Filter: Connectivity Analysis

The distinction between a Bend and a Hole lies in topology.

- **Hole:** A cylindrical surface that typically constitutes an *inner* loop of a planar face. It passes *through* the material thickness. Topologically, a hole face often connects a "Top" face to a "Bottom" face.
- **Bend:** A cylindrical surface that connects two *distinct* planar faces (flanges) that define the macroscopic shape of the part.

Algorithm for Connectivity:

For each Candidate Bend Face F_{cyl} :

1. **Extract Edges:** Retrieve the list of edges E_{list} belonging to F_{cyl} using TopExp_Explorer.
2. **Find Ancestors:** Use TopExp::MapShapesAndAncestors to find all faces connected to each edge in E_{list} .[10]
3. **Classify Neighbors:** For each edge, identify the adjacent face F_{adj} .

    - If F_{adj} is Planar, increment PlanarNeighborCount.
    - If F_{adj} is Cylindrical, the bend might be segmented (a chain of faces).
4. **Valid Bend Criterion:** A standard bend must act as a bridge.

$$\text{IsBend}(F_{cyl}) \iff \exists\{F_A, F_B\} \subset \text{Neighbors}(F_{cyl}) \text{ s.t. } \text{Type}(F_A) = \text{ Plane} \wedge \text{Type}(F_B) = \text{Plane}$$

4.3 The Semantic Filter: Geometric Constraints

Having identified a cylinder connecting two planes, we must further verify it is a *Sheet Metal Bend* and not a *Machined Fillet* or *Welded Tube*.

4.3.1 Tangency (G1 Continuity)

Sheet metal bending implies a smooth transition of material. The connection between the flange and the bend must be tangent.

- **Check:** Calculate the angle between the normal of the planar face (n_{plane}) and the normal of the cylindrical face (n_{cyl}) at the shared edge.
- **Condition:** $n_{plane} \parallel n_{cyl}$ (Angle ≈ 0).
- If the connection is sharp (G0 continuity), it is likely a weld or a different assembly component.[5]

4.3.2 Inner vs. Outer Radius Duality

A valid bend in a sheet of thickness T typically consists of a pair of coaxial cylindrical faces:

- **Inner Bend (F_{in}):** Radius R_{in}. Concave curvature relative to the material.
- **Outer Bend (F_{out}):** Radius $R_{out} = R_{in} + T$. Convex curvature.

Algorithm for Duality:

1. Calculate the radius R of the candidate face.
2. Cast a ray along the surface normal (inverted) to find the opposing face.
3. Verify the opposing face is also cylindrical and coaxial.
4. **Thickness Validation:** Check if $|R_{neighbor} - R| \approx T_{global}$.
    - If valid, the pair constitutes a single logic "Bend Feature."
    - If the opposing face is missing or planar, it is an **"Abnormal Bend"** (e.g., a crushed hem where $R_{in} \approx 0$).[5]

4.3.3 Fillet vs. Bend Distinction

Snippet [12] highlights the confusion between fillets and bends.

- **Fillet:** Often applied to edges of a solid block. The "thickness" is not uniform around the feature.
- **Bend:** Maintains uniform wall thickness throughout the arc.
- **Heuristic:** If the "Inner" cylinder does not have a corresponding "Outer" cylinder (or if the distance between them varies), it is likely a machined fillet or a chamfered edge, not a bend feature suitable for unfolding.

4.4 Advanced Bend Types

The algorithm must also account for non-standard bends identified in [5]:

- **Hems (Crushed Bends):** 180° bends where the inner radius is near zero. The recognition logic looks for a cylindrical face connecting a flange back to itself or to a parallel flange with distance $\approx 2T$.
- **Jogs:** A pair of bends in a zig-zag configuration (Up-Down) with a very short web between them.
- **Lofted Bends:** Represented by Conical or BSpline surfaces. Recognition requires verifying that the surface is developable (ruling lines exist).

5. Algorithmic Architecture: Identification of the Base Face

The second part of the user query addresses the "Base Face." In the context of Salvagnini STREAMBEND, the Base Face (or Fixed Face) is the geometric anchor for the bending process. It is the face that remains stationary (clamped by the manipulator) while the rest of the part is folded up or down.[6]

Selecting the correct base face is critical because it determines the *bending sequence*. A poor choice can lead to a sequence that is physically impossible to execute due to collisions or reachability issues.

5.1 Heuristic 1: Geometric Dominance (Maximum Area)

The most fundamental heuristic is physical stability. The manipulator needs surface area to grip.

- **Logic:** Iterate through all planar faces identified in the recognition phase. Calculate the surface area using BRepGProp::SurfaceProperties.
- **Selection:** The face with the global maximum area is the primary candidate.

$$F_{base} = \arg \max_{F \in \{Planes\}} (\text{Area}(F))$$

- **Rationale:** A larger area provides a more secure vacuum or mechanical clamp and minimizes vibration during the rapid bending strokes (up to 17 bends per minute on a P4 [13]).

5.2 Heuristic 2: Topological Centrality (Graph Theory)

In complex brackets, the largest face might be a long, narrow flange at the extremity of the part. Unfolding from an extremity accumulates tolerance errors and makes the part prone to "whipping" effects.

- **Logic:** Construct the Face Transition Graph (FTG) where nodes are flanges and edges are bends. Calculate the *closeness centrality* of each node.
- **Selection:** Choose the face that minimizes the maximum distance (in number of bends) to any leaf node (unbent flange).
- **Rationale:** Central faces minimize the lever arm of the moving masses during bending.

5.3 Heuristic 3: Kinematic Reachability (StreamBend Specifics)

Salvagnini's P4/P2 machines use a specific manipulator kinematics. The Base Face must be accessible.

- **Bounding Box Analysis:** The base face is often co-planar with the "bottom" or "back" of the bounding box of the folded part.
  - Compute the Axis-Aligned Bounding Box (AABB) of the solid.
  - Identify faces that lie on the Z_{min} or Z_{max} planes (depending on machine coordinate system).
- **Feature Avoidance:** A face full of holes (perforated) or deformations (louvers) is a poor candidate for suction cups.
  - **Net Area Calculation:** $A_{net} = A_{gross} - \sum A_{holes}$.
  - **Constraint:** If $A_{net} < A_{min\_clamp\_size}$, reject the candidate.[3]

5.4 Heuristic 4: Orientation Alignment

CAM software often imports parts in arbitrary orientations.

- **Logic:** Prefer faces whose normal vector aligns with the principal axes (X, Y, Z).
- **Tie-Breaker:** If two faces have equal area, prefer the one aligned with the -Z axis (gravity vector), assuming the part is placed flat on the bed.

5.5 The "Manipulator" Constraint

Snippet [6] explicitly states: "The Base face is the only part of the panel not subject to roto-translation following the bend."

- **Algorithm:** This implies a simulation-based check. If we select Face A as base, and simulate the unfolding of all other faces, do we encounter collisions?
- **Symmetry Check:** For symmetric parts, the base face is usually the axis of symmetry.

6. Technical Implementation: A PythonOCC Framework

The following section translates the theoretical logic into a practical Python script using pythonocc-core. This script demonstrates the specific API calls required to iterate topology, classify geometry, and identify the "Bend" and "Base" features.

6.1 Prerequisites and Setup

The script utilizes:

- TopoDS: For the underlying shape data structure.
- TopExp: For topological exploration (traversing faces, edges).
- BRepAdaptor: For geometric interrogation (is this face a cylinder?).
- BRepGProp: For calculating surface area.
- TopAbs: For topological type definitions.

6.2 The Python Code Snippet

Python

```python
from __future__ import print_function
import sys
import math

from OCC.Core.STEPControl import STEPControl_Reader
from OCC.Core.TopExp import TopExp_Explorer, topexp_MapShapesAndAncestors
from OCC.Core.TopAbs import TopAbs_FACE, TopAbs_EDGE, TopAbs_WIRE
from OCC.Core.TopoDS import topods, TopoDS_Face
from OCC.Core.BRepAdaptor import BRepAdaptor_Surface
from OCC.Core.GeomAbs import GeomAbs_Plane, GeomAbs_Cylinder, GeomAbs_Cone
from OCC.Core.BRep import BRep_Tool
from OCC.Core.TopTools import TopTools_IndexedDataMapOfShapeListOfShape
from OCC.Core.GProp import GProp_GProps
from OCC.Core.BRepGProp import brepgprop_SurfaceProperties

class SheetMetalAnalyzer:
    def __init__(self, step_filename):
        """
       Initialize the analyzer with a STEP file path.
       Loads the shape and builds the topological map.
       """
        self.shape = self._load_step(step_filename)
        self.faces =
        self.edges_to_faces = TopTools_IndexedDataMapOfShapeListOfShape()
        self._map_topology()

    def _load_step(self, filename):
        reader = STEPControl_Reader()
        status = reader.ReadFile(filename)
        if status == 1:
            reader.TransferRoots()
            return reader.OneShape()
        else:
            raise ValueError("Error reading STEP file")

    def _map_topology(self):
        """
       Builds a map linking every Edge to its ancestor Faces.
       Critical for finding adjacency in O(1) time.
       """
        # Map Edges to Faces: This allows us to query "Which faces share this edge?"
        topexp_MapShapesAndAncestors(self.shape, TopAbs_EDGE, TopAbs_FACE,
self.edges_to_faces)

        # Also collect all faces for easy iteration
        exp = TopExp_Explorer(self.shape, TopAbs_FACE)
        while exp.More():
            self.faces.append(topods.Face(exp.Current()))
            exp.Next()

    def get_face_type(self, face):
        """
       Returns the geometric type of the face (Plane, Cylinder, etc.)
       Using BRepAdaptor to abstract away the underlying curve math.
       """
        surf = BRepAdaptor_Surface(face, True)
        return surf.GetType()
    def get_adjacent_faces(self, face):
        """
        Returns a list of faces adjacent to the input face.
        """
        neighbors =
        # Iterate over edges of the face
        exp = TopExp_Explorer(face, TopAbs_EDGE)
        while exp.More():
            edge = exp.Current()
            # Retrieve ancestors from the map
            ancestors = self.edges_to_faces.FindFromKey(edge)

            # Iterate through list of ancestors (usually 2 for manifold edges)
            it = ancestors.Iterator()
            while it.More():
                ancestor_face = topods.Face(it.Value())
                # Don't add the face itself, only neighbors
                if not ancestor_face.IsSame(face):
                    neighbors.append(ancestor_face)
                it.Next()
            exp.Next()
        return neighbors

    def identify_base_face(self):
        """
        Heuristic: Identify Base Face as the Planar Face with Maximum Area.
        This satisfies the 'clamping stability' requirement of StreamBend.
        """
        max_area = 0.0
        base_face = None

        for face in self.faces:
            if self.get_face_type(face) == GeomAbs_Plane:
                props = GProp_GProps()
                brepgprop_SurfaceProperties(face, props)
                area = props.Mass()

                # Update if we found a larger planar face
                if area > max_area:
                    max_area = area
                    base_face = face
        return base_face, max_area

    def detect_bends(self):
        """
        Detects bends by finding Cylindrical faces connected to two distinct Planar faces.
        This implements the 'Connecting Planar Faces' logic.
        """
        bends =

        for face in self.faces:
            # Criteria 1: Must be Cylindrical (Geometry Filter)
            if self.get_face_type(face) == GeomAbs_Cylinder:

                neighbors = self.get_adjacent_faces(face)
                planar_neighbors = 0

                # Criteria 2: Check neighbors (Topology Filter)
                # We use a unique list to handle cases where multiple edges connect to the same face
                unique_neighbor_faces =
                # Simple deduplication based on memory address/hash for this snippet
                # In production, use IsSame() comparison

                temp_neighbors =
                for n in neighbors:
                    is_present = False
                    for tn in temp_neighbors:
                        if tn.IsSame(n):
                            is_present = True
                            break
                    if not is_present:
                        temp_neighbors.append(n)

                # Count how many neighbors are Planes
                planar_neighbor_list =
                for n_face in temp_neighbors:
                    if self.get_face_type(n_face) == GeomAbs_Plane:
                        planar_neighbors += 1
                        planar_neighbor_list.append(n_face)

                # Criteria 3: Must connect at least two planar faces (The Flanges)
                if planar_neighbors >= 2:
                    surf = BRepAdaptor_Surface(face, True)
                    cyl = surf.Cylinder()
                    radius = cyl.Radius()

                    # Store the detected feature with metadata
                    bends.append({
                        "face": face,
                        "radius": radius,
                        "connected_flanges": planar_neighbors,
                        # In a full app, we would calculate the bend angle here
                        # using the normals of the planar_neighbor_list
                    })

        return bends

# --------------------------------------------------------------------------
# Main Execution Block
# --------------------------------------------------------------------------
if __name__ == "__main__":
    # Replace 'sheet_metal_part.step' with your actual STEP file path
    step_file = "sheet_metal_part.step"

    try:
        print(f"Loading and Analyzing {step_file}...")
        analyzer = SheetMetalAnalyzer(step_file)

        # 1. Identify Base Face
        base_face, area = analyzer.identify_base_face()
        if base_face:
            print(f" Base Face Detected.")
            print(f"      Surface Area: {area:.4f} sq units")
            # Further logic: Visualize this face in the viewer
        else:
            print(" No suitable planar Base Face found.")

        # 2. Detect Bends
        detected_bends = analyzer.detect_bends()
        print(f" Feature Recognition Complete.")
        print(f"      Number of Bends Detected: {len(detected_bends)}")

        for i, bend in enumerate(detected_bends):
            print(f"      Bend #{i+1}: Radius = {bend['radius']:.4f}, "
                  f"Connects {bend['connected_flanges']} flanges.")

    except Exception as e:
```

```
    print(f"Error processing file: {e}")
    # In production, specific exceptions for file not found or invalid topology
```

6.3 Detailed Code Analysis

6.3.1 Topology Mapping (topexp_MapShapesAndAncestors)

The efficiency of the algorithm hinges on _map_topology. A naive approach to finding neighbors would involve iterating over all faces for every edge, resulting in $O(N^2)$ complexity. By using topexp_MapShapesAndAncestors, we invert the data structure, creating a lookup table where Edge ->. This allows $O(1)$ adjacency queries, which is critical for performance when processing complex assemblies with thousands of faces.[10]

6.3.2 Geometric Abstraction (BRepAdaptor)

We do not query the Geom_Surface directly. Instead, we use BRepAdaptor_Surface. This is a crucial design pattern in OCCT. A face in a STEP file might be a "trimmed surface"—a small rectangular patch of an infinite cylinder. The Adaptor provides a uniform interface to query the properties (Radius, Axis) of the underlying primitive, regardless of how the face boundaries are defined.[9]

6.3.3 The "Unique Neighbors" Problem

In the detect_bends method, we implement a deduplication loop. A cylindrical bend face and a planar flange face often share *multiple* edges. For example, if a relief cut (a small notch) intersects the bend line, the single logical connection is split into two topological edges. If we simply counted edges, we might count the same flange twice. The code specifically checks IsSame() to ensure we are counting distinct topological **Faces**, not just edges.

7. Implications for CAM Workflows and Unfolding

7.1 From Recognition to Unfolding

Once the bends and base face are identified, the system must generate the "Flat Pattern." This is not a simple geometric projection. It requires the application of the **K-factor** (or Y-factor), which accounts for the shifting of the Neutral Axis during plastic deformation.

- **The Algorithm:**

$$L_{flat} = L_{straight_1} + L_{bend} + L_{straight_2}$$

$$L_{bend} = \frac{\pi \cdot \alpha}{180}(R_{inside} + K \cdot T)$$

- **Significance:** The feature recognition algorithm *must* accurately extract R_{inside} and the bend angle α. If the recognition logic mistakes a fillet ($R \neq T$) for a bend, it will apply the wrong unfolding formula, resulting in a flat pattern that is too long or too short. This leads to parts that do not fit in the assembly.[5]

7.2 Handling "Dirty" Geometry

Real-world STEP files are rarely perfect. They may contain:

- **Slivers:** Micro-faces caused by bad boolean operations.
- **Gaps:** Disconnected edges (tolerance issues).
- **Split Faces:** A single logical flange represented as two coplanar faces split by a sketch line.
  The recognition algorithm must include a "Healing" or "Sewing" phase (using BRepOffsetAPI_Sewing) before building the adjacency graph to ensure that logical connections are topologically valid.

7.3 Integration with Salvagnini P4 Logic

For STREAMBEND specifically, the recognition of **Simultaneous Bends** is vital.

- **Scenario:** A box panel with bends on all four sides.
- **Logic:** If two bends share a common corner vertex (or are separated by a corner relief) and their axes are collinear, they can be bent simultaneously by the long blade of the P4 machine.
- **Detection:** The algorithm checks the Axis vector of all recognized bends.

$$\text{IsSimultaneous}(B_1, B_2) \iff \text{Axis}(B_1) \parallel \text{Axis}(B_2) \wedge \text{Collinear}(B_1, B_2)$$

  This grouping allows the CAM software to optimize the cycle time by reducing the number of manipulator rotations.[6]

8. Conclusion

Reverse-engineering the feature recognition logic of software like Salvagnini STREAMBEND is a rigorous exercise in computational geometry. It moves beyond simple shape display to semantic understanding. We have demonstrated that the distinction between a "Bend" and a "Face" is not inherent in the STEP file but is a derived property of topological relationships (Connectivity) and geometric constraints (Tangency, Radius-Thickness Duality).

The identification of the "Base Face" is equally heuristic-driven, balancing geometric certainty (Area) with kinematic necessity (Centrality, Clamping Area). The provided PythonOCC implementation establishes a robust foundation for this analysis, leveraging the MapShapesAndAncestors and BRepAdaptor classes to bridge the gap between raw B-Rep data and intelligent CAM automation.

Future developments in this field are moving toward **Deep Learning** approaches (e.g., UV-Net, BRepNet) to replace brittle heuristics with probabilistic classifiers.[15] However, for the foreseeable future, the deterministic, graph-based algorithms detailed in this report remain the gold standard for precision manufacturing, ensuring that the "dumb" geometry of today can drive the smart factories of tomorrow.

Works cited

1. Bending Simulation Framework for Rapid Feasibility Checks of Sheet Metal Parts - CAD Journal, accessed February 1, 2026, https://www.cad-journal.net/files/vol_23/CAD_23(1)_2026_85-100.pdf
2. On sheet metal recognition and unfolding (Part 1), accessed February 1, 2026, https://quaoar.su/blog/page/on-sheet-metal-unfolding-part-1
3. Salvagnini P-Robot | Custom Panel Bending Automation, accessed February 1, 2026, https://www.salvagninigroup.com/en-US/products/panel-benders/P-Robot
4. Salvagnini panel bending: P2 features easy and quick production changeover - YouTube, accessed February 1, 2026, https://www.youtube.com/watch?v=UYWfEazLE5s
5. Sheet Metal Operations Component - Open Cascade, accessed February 1, 2026, https://www.opencascade.com/components/sheet-metal-operations/
6. Streambend en | PDF | Rotation | Art - Scribd, accessed February 1, 2026, https://www.scribd.com/document/761106869/Streambend-En
7. Modeling Data - PythonOCC Tutorial - Read the Docs, accessed February 1, 2026, https://pythonocc-doc.readthedocs.io/en/latest/modeling_data/
8. sheet metal features - Analysis Situs, accessed February 1, 2026, https://analysissitus.org/features/features_sheet-metal.html
9. pythonocc-demos/examples/core_geometry_recognize_feature.py at master - GitHub, accessed February 1, 2026, https://github.com/tpaviot/pythonocc-demos/blob/master/examples/core_geometry_recognize_feature.py
10. Give a TopoDS_Face,find connected face with the given one - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/content/give-topodsfacefind-connected-face-given-one
11. pythonocc-utils/OCCUtils/Topology.py at master - GitHub, accessed February 1, 2026, https://github.com/tpaviot/pythonocc-utils/blob/master/OCCUtils/Topology.py
12. Fillets vs Chamfers: How to Handle Edges and Corners in Machining - Protolabs, accessed February 1, 2026, https://www.protolabs.com/resources/blog/fillets-and-chamfers/
13. Bending? Easy, with Salvagnini!, accessed February 1, 2026, https://www.salvagninigroup.com/en-INT/campaigns/panel-benders-range
14. One way to override bending radii in sheet metal unfolding algorithm | by Analysis

Situs, accessed February 1, 2026, https://analysis-situs.medium.com/one-way-to-override-bending-radii-in-sheet-metal-unfolding-algorithm-75e6f111989d

15. Interfacing rule-based feature recognition with machine learning (AI) methods, accessed February 1, 2026, https://quaoar.su/blog/page/interfacing-with-machine-learning-feature-recognition