Comprehensive Analysis of Tube Geometry Reconstruction and Data Exchange using OpenCASCADE Technology

1. Introduction to Computational Tube Fabrication

The digital transformation of tube and pipe fabrication has necessitated a robust bridge between manufacturing parameters and geometric modeling. In the domain of Computer-Aided Manufacturing (CAM), the ability to algorithmically reconstruct precise 3D solid models from abstract machine codes is paramount for collision detection, process simulation, and automated quality assurance. This report provides an exhaustive technical analysis of generating standard Exchange of Product (STEP) files from two distinct data sources: parametric YBC (Feed-Rotate-Bend) coordinates and raw Centerline (XYZ) point clouds, utilizing the OpenCASCADE Technology (OCCT) kernel (version 7.8+).

Tube bending is a deformation process where a straight tube is manipulated into a complex 3D shape. The mathematical description of this shape exists in two primary domains: the **Cartesian space** (absolute x, y, z coordinates of the centerline) and the **Configuration space** (relative machine movements Y, B, C). While Cartesian coordinates are ubiquitous in design software, bending machines operate on configuration parameters. Consequently, the conversion between these formats—specifically the reconstruction of volumetric geometry from scalar parameters—is a fundamental problem in industrial software engineering.[1]

This report addresses the specific algorithmic challenges of this reconstruction, including the kinematic transformation of YBC data, the differential geometry of sweeping operations along B-Splines, and the topological validity of the resulting Boundary Representation (B-Rep) models. It further details the implementation of these algorithms using the C++ API of OpenCASCADE, focusing on the BRepPrimAPI and BRepOffsetAPI packages, and concludes with the standardization of the output using ISO 10303 (STEP) protocols.

2. Mathematical Foundations of Tube Geometry

The accurate generation of 3D geometry requires a rigorous definition of the coordinate systems employed in tube fabrication. The distinction between the relative frame of the bending machine and the absolute frame of the CAD model is the source of significant

complexity in geometric reconstruction.

2.1 The YBC (LRA) Coordinate System

The YBC coordinate system, frequently referred to as LRA (Length, Rotation, Angle), defines the tube geometry through a sequence of discrete machine motions. This system is intrinsic to the kinematics of Rotary Draw Bending machines and defines the tube shape relative to the machine's carriage and bend die.[1]

2.1.1 Parameter Definitions

- **Y (Length/Feed):** This parameter represents the linear translation of the tube through the collet. Geometrically, it corresponds to the length of the straight cylindrical segment between two tangent points (the end of one bend and the start of the next). In the machine frame, this is a translation along the longitudinal axis of the tube.[2]
- **B (Rotation):** This parameter denotes the rotation of the tube about its longitudinal axis. It establishes the orientation of the plane of the *next* bend relative to the plane of the *current* bend. This is a polar coordinate acting on the tube's cross-section.[1]
- **C (Bend Angle):** This parameter defines the radial deformation of the tube. It is the angle subtended by the toroidal arc formed around the bend die. The radius of this arc, known as the Centerline Radius (CLR), is typically a function of the tooling and is distinct from the YBC parameters themselves.[2]

2.1.2 Kinematic Chain and Order of Operations

The reconstruction of a tube from YBC data is analogous to the forward kinematics of a serial manipulator in robotics. Each row of YBC data (Y_i, B_i, C_i) can be viewed as a link in a kinematic chain. The position of the tube segment i is dependent on the cumulative transformations of all preceding segments 0 to $i-1$. This dependency creates a susceptibility to "stack-up errors," where small deviations in the calculation of early segments propagate and magnify in the final geometry.[2]

The standard operational sequence for transforming the local coordinate frame (the "turtle" or tool center point) for a single YBC row is:

1. **Feed (Y):** Translate the local frame along its Z-axis (or X-axis, depending on convention) by distance Y_i.
2. **Rotate (B):** Rotate the local frame about its longitudinal axis by angle B_i.
3. **Bend (C):** Rotate the local frame about an axis perpendicular to the feed axis by angle C_i, while simultaneously traversing the arc length $L_{arc} = CLR \cdot C_i$ (where C_i is in

radians).

2.2 Coordinate Transformation Matrices

To implement this kinematic chain in software, we utilize Homogeneous Transformation Matrices (4×4). These matrices allow for the combination of rotation and translation into a single linear algebraic operation, facilitating efficient computation of the global coordinates for each tube segment.[5]

Let T_i represent the transformation matrix of the tube frame after processing row i. Let M_i be the local transformation defined by the parameters (Y_i, B_i, C_i, CLR_i). The global state is updated recursively:

$$T_i = T_{i-1} \cdot M_i$$

The construction of M_i requires breaking down the Y, B, and C movements. Assuming a standard "Forward-Up-Right" convention where the tube feeds along the local Z-axis:

1. **Translation Matrix (D_Y):**

$$D_Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Y_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. **Rotation Matrix (R_B):**
   Rotation around the Z-axis (Feed Axis).

$$R_B = \begin{bmatrix} \cos(B_i) & -\sin(B_i) & 0 & 0 \\ \sin(B_i) & \cos(B_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. **Bend Transformation (T_C):**
   The bend is complex because it involves both rotation and translation along a circular arc. The frame moves from the start of the arc (Tangent Point 1) to the end of the arc (Tangent Point 2).
   If the bend occurs in the local X-Z plane (rotating around Y), the transformation involves rotating by C_i around Y, and translating by the vector chord of the arc.
   Alternatively, using the concept of a virtual pivot (the center of the bend radius), we can

express the bend as:

- Translate to Bend Center: $T_{in} = T$
- Rotate around Center: R_C (around Y axis)
- Translate back from Center: $T_{out} = T$

$$T_C = \text{Trans}(T_{out}) \cdot R_Y(C_i) \cdot \text{Trans}(T_{in})$$

Combining these, the local step M_i is:

$$M_i = D_Y \cdot R_B \cdot T_C$$

This matrix logic serves as the mathematical backbone for the reconstructTubeFromYBC algorithm implemented later in this report.[6]

2.3 The Cartesian Centerline Representation

In contrast to YBC, the Cartesian representation defines the tube via a series of 3D points (x, y, z). These points typically represent the **Intersection Points (IP)**—the theoretical vertices where the centerlines of adjacent straight sections intersect.

While simpler to store, this format lacks explicit definition of the bend geometry. The arc must be inferred by fitting curves between the straight segments. This introduces the challenge of continuity. A tube centerline must be at least G^1 continuous (tangent continuous) to be physically realizable. A sequence of lines connected by sharp corners (C^0 continuity) represents a mitered joint, not a bent tube. The conversion from Centerline Points to a swept surface therefore requires the generation of transition curves (fillets or BSplines) that maintain tangency with the linear segments.[8]

3. OpenCASCADE Technology (OCCT) Architecture

To implement the mathematical models described above, we leverage OpenCASCADE Technology. OCCT is a C++ class library designed for geometric modeling. Understanding its internal architecture—specifically the separation of Geometry and Topology—is critical for generating valid STEP files.

3.1 Geometry vs. Topology

OCCT enforces a strict distinction between the mathematical definition of a shape (Geometry) and its bounded occurrence in a model (Topology).

- **Geometry (Geom package):** Classes such as Geom_CylindricalSurface,

Geom_ToroidalSurface, and Geom_BSplineCurve define infinite or semi-infinite geometric entities. They contain the parametric equations but no concept of "start" or "end".[10]
- **Topology (TopoDS package):** Classes such as TopoDS_Face, TopoDS_Edge, and TopoDS_Vertex define the boundaries. A TopoDS_Edge is a topological object that references a Geom_Curve and delimits it with two TopoDS_Vertex objects.

Implication for Tube Reconstruction: We cannot simply "draw" a cylinder. We must:

1. Define a Geom_CylindricalSurface (Geometry).
2. Define TopoDS_Vertex points for the start and end (Topology).
3. Create a TopoDS_Edge along the surface (Topology).
4. Create TopoDS_Wire loops at the ends (Topology).
5. Create a TopoDS_Face bounded by these wires (Topology).
6. Join faces into a TopoDS_Shell and eventually a TopoDS_Solid.[10]

Fortunately, the BRepPrimAPI package automates this complex construction for standard primitives like cylinders and tori.

3.2 Precision and Tolerance

OCCT operates with a fundamental fuzzy tolerance, typically defined by Precision::Confusion() (default 1×10^{-7}). When constructing a tube segment-by-segment, the endpoint of the cylinder and the start point of the torus must be coincident within this tolerance. If the mathematical transformation of the YBC data results in a gap larger than Precision::Confusion(), topological algorithms like BRepBuilderAPI_Sewing will fail to create a closed shell. This results in a "leaky" model that cannot be capped into a solid or volume.[13]

3.3 Memory Management

OCCT uses a handle-based memory management system (Handle<T>), which acts as a smart pointer. Geometric objects (Geom_Curve) are manipulated via handles, while topological objects (TopoDS_Shape) are manipulated by value but contain internal handles to the underlying data structures. Correct usage of Handle is essential to prevent memory leaks during the iterative generation of complex tube assemblies.

4. Algorithmic Approach I: Parametric Reconstruction from YBC

The most robust method for converting YBC data to STEP is the "Constructive Solid Geometry" (CSG) approach. Rather than sweeping a profile along a path, we construct the tube by assembling precise analytic primitives (cylinders for straight sections, tori for bends) and stitching them together. This ensures that the resulting STEP file contains exact geometric definitions (Analytic Surfaces) rather than approximated NURBS surfaces, resulting

in smaller file sizes and higher precision for downstream CAM software.

4.1 Primitive Generation Strategy

The reconstruction algorithm iterates through the YBC data vector. For each row, it generates two potential topological objects: a TopoDS_Solid (or Shell) representing the straight feed Y, and a TopoDS_Solid representing the bend C.

4.1.1 Creating Straight Segments (BRepPrimAPI_MakeCylinder)

The BRepPrimAPI_MakeCylinder class creates a cylindrical primitive. By default, it builds a cylinder along the Z-axis of the local coordinate system.[15]

- **Input:** Radius ($OD/2$), Length (Y).
- **Alignment:** The local coordinate frame must be rotated so that the Z-axis aligns with the current tangent vector of the tube.
- **Result:** A solid cylinder with planar caps (top and bottom).

4.1.2 Creating Partial Bends (BRepPrimAPI_MakeTorus)

The creation of the bend is the most complex step. A torus in OCCT is generated by revolving a circle around a major axis. The BRepPrimAPI_MakeTorus class offers several constructors, but for tube bending, we specifically need the partial sweep constructor.[13]

Research Insight on Torus Angles: Snippet [16] identifies the constructor: BRepPrimAPI_MakeTorus(Axes, R1, R2, angle).

- **R1 (Major Radius):** Corresponds to the Centerline Radius (CLR) of the bend.
- **R2 (Minor Radius):** Corresponds to the Tube Radius ($OD/2$).
- **angle:** The sweep angle of the major radius (the Bend Angle C).
- **Axes (gp_Ax2):** This defines the center and orientation of the torus.

Crucial Alignment Logic:

The standard OCCT torus is centered at the origin of the Axes. The tube centerline, however, lies at a distance $R1$ from this origin. Furthermore, the torus starts at angle 0 relative to the X-axis of the Axes.

To align the torus with the end of the previous straight segment:

1. **Center Calculation:** The center of the torus is *not* the end of the straight tube. It is offset by the Bend Radius (CLR) in the direction perpendicular to the tube tangent, specifically in the direction of the *Center of Bend*.

$$P_{center} = P_{tangent} + V_{normal} \cdot CLR$$

Where V_{normal} is the vector pointing from the tube centerline towards the center of the bend arc.

2. **Orientation:** The gp_Ax2 for the torus must have:
    - **Location:** P_{center}
    - **Direction (Z-axis):** The axis of bend rotation (perpendicular to the Plane of Bend).
    - **X-Direction:** Points from P_{center} back to $P_{tangent}$. This ensures the torus arc starts exactly at the tangent point.

This rigorous alignment ensures that the circular face at the end of the cylinder is geometrically identical to the circular face at the start of the torus, allowing for seamless topological stitching.

4.2 Segment Alignment and Sewing

Once the primitives are created, they exist as separate solids in space. To create a single valid tube, they must be fused.

Comparison of Boolean Fuse vs. Sewing:

- **BRepAlgoAPI_Fuse:** Performs a boolean union. This is computationally expensive and can be unstable if faces are coplanar or tangent.[17] It calculates intersections, which is unnecessary here because we know the faces are coincident.
- **BRepBuilderAPI_Sewing:** "Stitches" faces together that share geometry within a tolerance. This is the preferred method for tube reconstruction.[13]

The Stitching Workflow:

1. Generate all primitives as TopoDS_Shell (hollow surfaces) rather than Solids.
2. Add all shells to a BRepBuilderAPI_Sewing object.
3. Set tolerance to 1×10^{-6} (slightly looser than default to account for float errors in matrix math).
4. Execute Perform().
5. The result is a single TopoDS_Shell.

4.3 End Caps and Solidification

The sewed shell is open at both ends (a pipe). To create a solid volume (necessary for mass property calculation or interference checking):

1. **Identify Free Edges:** Use ShapeAnalysis_FreeBounds or iterate through edges to find those shared by only one face.[12] Ideally, there are exactly two circular loops.

2. **Create Caps:** For each loop, create a TopoDS_Wire, then use BRepBuilderAPI_MakeFace to generate a planar disk.[12]
3. **Final Solid:** Create a new BRepBuilderAPI_Sewing or use BRepBuilderAPI_MakeSolid to combine the tube shell and the two end caps into a TopoDS_Solid.[13]

5. Algorithmic Approach II: Freeform Reconstruction from Centerlines

When the input data is a series of Cartesian coordinates (XYZ) rather than YBC parameters, the analytic primitive approach is insufficient. The path may contain complex splines or variable radii that do not map to simple cylinders and tori. In this scenario, the "Sweep" or "Pipe" generation method is required.

5.1 B-Spline Curve fitting

The first step is to convert the discrete points into a continuous mathematical curve.

GeomAPI_PointsToBSpline Analysis: This class converts a point array into a Geom_BSplineCurve.[24]

- **Interpolation vs. Approximation:**
  - *Interpolation* forces the curve through every point. This is appropriate if the points are "hard" intersection points from a CAD drawing.
  - *Approximation* fits a curve minimizing error. This is better for scanned data (reverse engineering) to smooth out noise.
- **Continuity:** The curve must be at least $G1$ (tangent continuous) for the pipe algorithm to succeed. GeomAbs_C2 (curvature continuous) is recommended for high-quality surfaces.[26]
- **Parameters:**
  - Degree: Typically 3 (Cubic).
  - Tolerance: The maximum allowed deviation (e.g., 0.001 mm).

5.2 Sweeping Algorithm (BRepOffsetAPI_MakePipeShell)

Sweeping involves moving a profile (a circle) along the spine (the BSpline curve). While BRepOffsetAPI_MakePipe is simple, it lacks control over the profile's orientation. The robust solution is BRepOffsetAPI_MakePipeShell.[8]

5.2.1 The Frame Transport Problem (Twist)

As the profile moves along the 3D curve, its local coordinate system must be defined at every point. The standard mathematical solution is the **Frenet Frame** (Tangent, Normal, Binormal).

However, the Frenet frame is undefined at inflection points (zero curvature) and flips 180°

when curvature direction changes. This causes the tube to pinch or twist violently.

Corrective Strategies [26]: The BRepOffsetAPI_MakePipeShell class provides modes to handle this:

- **SetMode(IsFrenet = Standard_False):** Uses a "Corrected Frenet" frame. This minimizes rotation relative to the previous frame, reducing twist. This is the default recommended setting for general 3D tubes.
- **SetMode(gp_Dir BiNormal):** Fixes the binormal to a specific vector (e.g., vertical Z-axis). This is useful if the tube lies mostly in a horizontal plane but fails if the tube goes vertical.
- **SetDiscreteMode():** Uses a discrete trihedron algorithm which is robust for complex paths with sharp corners.

5.3 Topology Generation

The output of MakePipeShell is a shell. Similar to the YBC method, if a solid is required, the start and end profiles must be capped.

- **Profile Definition:** The profile should be defined as a TopoDS_Wire containing a TopoDS_Edge (Circle).
- **Sweep:** The sweep operation generates the pipe wall.
- **Solidification:** MakePipeShell has a MakeSolid() method, but it only works if the sweep is closed (a torus) or if caps are handled internally. Often, manual capping (as described in 4.3) is more reliable.

6. Data Exchange and Standardization (STEP)

The final objective is to export the TopoDS_Shape to a STEP file (ISO 10303). This ensures the model is readable by solid modeling software (SolidWorks, Catia, Siemens NX).

6.1 STEP Protocol Configuration

The STEPControl_Writer class manages the translation.[29]

- **Application Protocol (AP):**
  - **AP203 (Config Control Design):** The classic standard. Good for geometry, but often lacks color and layer support.
  - **AP214 (Automotive Design):** The industry standard for manufacturing. Supports colors, layers, and assemblies. This is the recommended schema for tube fabrication.
  - **AP242 (Managed Model Based 3D Engineering):** The newest standard, merging 203 and 214 with Geometric Dimensioning and Tolerancing (GD&T).

Configuration Snippet [31]: To set the schema, use the Interface_Static class:
Interface_Static::SetCVal("write.step.schema", "AP214");

6.2 Unit Handling

OCCT defaults to Millimeters. The STEP writer generally infers units from the session. It is crucial to ensure the header of the STEP file reflects the correct units to prevent scaling errors (e.g., a 100mm tube importing as 100 meters).

7. Implementation Architecture

This section provides the comprehensive C++ code required to implement the research findings. The code is structured into a cohesive TubeBuilder class.

7.1 Header and Data Structures

The following code defines the data structures for YBC and the necessary includes.

C++

```
#include <vector>
#include <cmath>
#include <iostream>
#include <memory>

// OpenCASCADE Includes
#include <gp_Pnt.hxx>
#include <gp_Ax2.hxx>
#include <gp_Dir.hxx>
#include <gp_Circ.hxx>
#include <gp_Trsf.hxx>
#include <gp_Quaternion.hxx>

#include <TopoDS.hxx>
#include <TopoDS_Shape.hxx>
#include <TopoDS_Face.hxx>
#include <TopoDS_Wire.hxx>
#include <TopoDS_Edge.hxx>
#include <TopoDS_Shell.hxx>
#include <TopoDS_Solid.hxx>

#include <BRep_Builder.hxx>
#include <BRepBuilderAPI_MakeEdge.hxx>
#include <BRepBuilderAPI_MakeWire.hxx>
#include <BRepBuilderAPI_MakeFace.hxx>
#include <BRepBuilderAPI_Sewing.hxx>
```

```cpp
#include <BRepBuilderAPI_MakeSolid.hxx>
#include <BRepPrimAPI_MakeCylinder.hxx>
#include <BRepPrimAPI_MakeTorus.hxx>
#include <BRepOffsetAPI_MakePipeShell.hxx>

#include <Geom_BSplineCurve.hxx>
#include <GeomAPI_PointsToBSpline.hxx>
#include <Geom_Line.hxx>
#include <Geom_Circle.hxx>

#include <STEPControl_Writer.hxx>
#include <Interface_Static.hxx>
#include <ShapeAnalysis_FreeBounds.hxx>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// --- Data Structures ---

// YBC Data Row: Defines one step of the bending process
struct YBCRow {
    double Y;   // Feed Length (mm)
    double B;   // Rotation (Degrees)
    double C;   // Bend Angle (Degrees)
    double CLR; // Centerline Radius (mm) - Property of the tool
};

// Tube Parameters: Global properties of the raw material
struct TubeParams {
    double OD; // Outside Diameter
    double WT; // Wall Thickness (0 for solid rod)

    double Radius() const { return OD / 2.0; }
};

// Helper: Convert Degrees to Radians
inline double DegToRad(double deg) {
    return deg * (M_PI / 180.0);
}
```

7.2 Core Logic: YBC Reconstruction

The ReconstructTubeFromYBC function implements the CSG approach with rigorous frame

tracking.

Table 1: Matrix Operation vs OCCT Transform

YBC Operation	Math Operation	OCCT Class
Feed (Y)	Translate along Z	currentPose.Translate(vec)
Rotate (B)	Rotate around Z	gp_Trsf::SetRotation(axis, ang)
Bend (C)	Re-orient Axes + Offset	gp_Ax2 construction

Code Implementation:

C++

```
class TubeBuilder {
public:
    TubeBuilder(const TubeParams& params) : m_params(params) {}

    // ---------------------------------------------------------------------
    // Method 1: YBC to STEP Shape
    // ---------------------------------------------------------------------
    TopoDS_Shape ReconstructTubeFromYBC(const std::vector<YBCRow>& ybcData) {
        if (ybcData.empty()) return TopoDS_Shape();

        BRepBuilderAPI_Sewing sew(1.0e-6); // High precision sewing

        // Initial Coordinate System: Tube starts at Origin, pointing +X
        // Note: YBC machines vary. Here we assume:
        // Feed Axis: +X
        // Bend Axis: +Z (Bends go towards -Y or +Y depending on chirality)
        gp_Ax2 currentPose(gp_Pnt(0,0,0), gp_Dir(1,0,0), gp_Dir(0,0,1));

        for (size_t i = 0; i < ybcData.size(); ++i) {
            const YBCRow& row = ybcData[i];
```

```
        // --- 1. ROTATION (B) ---
        // Rotate the carriage/tube before feeding.
        // Rotation is around the Feed Axis (Main Direction of currentPose)
        if (std::abs(row.B) > 1.0e-5) {
            gp_Ax1 rotAxis(currentPose.Location(), currentPose.Direction());
            gp_Trsf rotTrsf;
            rotTrsf.SetRotation(rotAxis, DegToRad(row.B));

            // Apply rotation to the coordinate system axes
            // Note: We don't transform the shape, we transform the 'cursor'
            gp_Dir xDir = currentPose.Direction();
            gp_Dir yDir = currentPose.YDirection();
            xDir.Transform(rotTrsf);
            yDir.Transform(rotTrsf);
            currentPose.SetDirection(xDir);
            currentPose.SetYDirection(yDir);
        }

        // --- 2. FEED (Y) ---
        if (row.Y > 1.0e-5) {
            // Construct Cylinder
            // MakeCylinder builds along Z of the given Ax2.
            // Our feed axis is currentPose.Direction() (let's say X).
            // We need a temp frame for the primitive where Z' aligns with Feed Axis.
            gp_Ax2 primitiveAx = currentPose;
            // OCCT MakeCylinder(Ax2, R, H) builds along the Z direction of Ax2.
            // So we pass currentPose directly if we ensure currentPose.Direction is the cylinder axis.
            // However, MakeCylinder Z is "Height".
            // Let's force alignment:
            gp_Ax2 cylFrame(currentPose.Location(), currentPose.Direction());

            BRepPrimAPI_MakeCylinder cylMaker(cylFrame, m_params.Radius(), row.Y);
            sew.Add(cylMaker.Shape());

            // Update Position: Move 'cursor' forward by Y
            gp_Vec feedVec(currentPose.Direction());
            feedVec.Multiply(row.Y);
            currentPose.Translate(feedVec);
        }

        // --- 3. BEND (C) ---
        if (std::abs(row.C) > 1.0e-5 && row.CLR > 1.0e-5) {
            double bendAngleRad = DegToRad(row.C);
```

```
            // Calculate Torus Frame
            // The torus must meet the cylinder tangentially.
            // Center of Torus: Offset from current location by CLR
            // Direction: Perpendicular to Feed Axis, defined by B rotation.
            // We assume the bend happens "around" the YDirection of the pose,
            // meaning the Center is along the ZDirection x Direction (Cross product).

            // Let's define the Bend Plane Normal as currentPose.YDirection().
            // The direction to the Center is Cross(Normal, Tangent).
            gp_Dir tangent = currentPose.Direction(); // Feed axis
            gp_Dir bendAxis = currentPose.YDirection(); // Axis of rotation
            gp_Dir radialDir = tangent.Crossed(bendAxis); // Direction towards center

            gp_Vec offsetVec(radialDir);
            offsetVec.Multiply(row.CLR);
            gp_Pnt centerPnt = currentPose.Location().Translated(offsetVec);

            // The Torus primitive Ax2:
            // Location: CenterPnt
            // Direction (Z): The axis of revolution (bendAxis)
            // XDirection: Must point to the start of the arc (back to currentPose.Location)
            gp_Dir torusXDir(gp_Vec(centerPnt, currentPose.Location()));
            gp_Ax2 torusAx(centerPnt, bendAxis, torusXDir);

            // Create Partial Torus
            // R1 = CLR, R2 = Tube Radius, Angle = Bend Angle
            BRepPrimAPI_MakeTorus torusMaker(torusAx, row.CLR, m_params.Radius(), bendAngleRad);
            sew.Add(torusMaker.Shape());

            // Update Position: Move 'cursor' to end of arc
            // 1. Rotate the Tangent vector around Bend Axis by Angle C
            gp_Trsf bendRot;
            bendRot.SetRotation(gp_Ax1(centerPnt, bendAxis), bendAngleRad);

            // Transform the entire coordinate system to the end of the bend
            // We transform the LOCATION
            gp_Pnt endPnt = currentPose.Location().Transformed(bendRot);

            // Transform the AXES
            gp_Dir newTangent = tangent.Transformed(bendRot);
            gp_Dir newYDir = bendAxis.Transformed(bendRot); // Should stay same if axis is fixed
```

```
            currentPose.SetLocation(endPnt);
            currentPose.SetDirection(newTangent);
            currentPose.SetYDirection(newYDir);
        }
    }

    sew.Perform();
    TopoDS_Shape sewedShell = sew.SewedShape();

    // Cap the ends to make a solid
    return MakeSolidFromOpenShell(sewedShell);
  }

private:
  const TubeParams& m_params;

  // Helper: Cap Ends
  TopoDS_Shape MakeSolidFromOpenShell(const TopoDS_Shape& shell) {
    // 1. Find free edges
    ShapeAnalysis_FreeBounds freeBounds(shell);
    TopoDS_Compound closedWires = freeBounds.GetClosedWires();

    BRepBuilderAPI_Sewing capper;
    capper.Add(shell);

    TopoDS_Iterator it(closedWires);
    for (; it.More(); it.Next()) {
        TopoDS_Wire wire = TopoDS::Wire(it.Value());
        // Create planar face
        BRepBuilderAPI_MakeFace faceMaker(wire);
        if (faceMaker.IsDone()) {
            capper.Add(faceMaker.Face());
        }
    }

    capper.Perform();
    TopoDS_Shape closedShell = capper.SewedShape();

    // Verify if closed
    // BRepBuilderAPI_MakeSolid can create a solid from a closed shell
    BRepBuilderAPI_MakeSolid solidMaker;
    solidMaker.Add(TopoDS::Shell(closedShell));
```

```
        if (solidMaker.IsDone()) {
            return solidMaker.Solid();
        } else {
            // Fallback: return shell
            return closedShell;
        }
    }
};
```

7.3 Core Logic: Centerline Reconstruction

The ReconstructTubeFromCenterline function implements the BSpline sweep strategy.

C++

```
// ---------------------------------------------------------------------------
// Method 2: Centerline to STEP Shape
// ---------------------------------------------------------------------------
TopoDS_Shape ReconstructTubeFromCenterline(const std::vector<gp_Pnt>& points) {
    if (points.size() < 2) return TopoDS_Shape();

    // 1. Prepare Points Array
    TColgp_Array1OfPnt occtPoints(1, points.size());
    for (size_t i = 0; i < points.size(); ++i) {
        occtPoints.SetValue(i + 1, points[i]);
    }

    // 2. Generate Spine Curve (BSpline)
    // Use Approximation with C2 continuity for smooth pipes
    // Tolerance 0.1mm to smooth out noise
    GeomAPI_PointsToBSpline spineBuilder(occtPoints, GeomAbs_C2, 0.1, 3);
    Handle(Geom_BSplineCurve) spineCurve = spineBuilder.Curve();

    BRepBuilderAPI_MakeEdge edgeMaker(spineCurve);
    BRepBuilderAPI_MakeWire wireMaker(edgeMaker.Edge());
    TopoDS_Wire spineWire = wireMaker.Wire();

    // 3. Create Profile
    // A circle in the XY plane. The PipeShell algorithm transforms it automatically.
    gp_Ax2 profileAx(gp_Pnt(0,0,0), gp_Dir(0,0,1));
```

```
    gp_Circ profileCirc(profileAx, m_params.Radius());
    BRepBuilderAPI_MakeEdge profileEdge(profileCirc);
    BRepBuilderAPI_MakeWire profileWire(profileEdge);

    // 4. Sweep (MakePipeShell)
    BRepOffsetAPI_MakePipeShell pipeBuilder(spineWire);
    pipeBuilder.Add(profileWire);

    // Handle Twist: Use Corrected Frenet
    pipeBuilder.SetMode(Standard_False);

    if (pipeBuilder.Build()) {
        TopoDS_Shape pipeShell = pipeBuilder.Shape();
        return MakeSolidFromOpenShell(pipeShell); // Reuse capper
    }

    return TopoDS_Shape();
}
```

7.4 STEP Export

C++

```
// ---------------------------------------------------------------------------
// Export Function
// ---------------------------------------------------------------------------
bool ExportToSTEP(const TopoDS_Shape& shape, const std::string& filename) {
    if (shape.IsNull()) return false;

    STEPControl_Writer writer;

    // Configuration
    // AP214 is best for general CAD interoperability
    Interface_Static::SetCVal("write.step.schema", "AP214");

    // Transfer
    IFSelect_ReturnStatus status = writer.Transfer(shape, STEPControl_AsIs);

    if (status!= IFSelect_RetDone) {
        std::cerr << "STEP Transfer failed." << std::endl;
```

```
        return false;
    }

    // Write
    status = writer.Write(filename.c_str());
    return (status == IFSelect_RetDone);
}
```

8. Post-Processing and Validity Analysis

Ensuring the generated geometry is "watertight" is essential. The MakeSolidFromOpenShell helper function (included in Section 7.2) performs this critical step.

8.1 The Capping Algorithm

The algorithm uses ShapeAnalysis_FreeBounds. This tool analyzes the topology of a shell and returns compound wires representing the free boundaries (holes).

1. **Input:** TopoDS_Shell.
2. **Analysis:** Identify edges referenced by only one face. Connect these edges into wires.
3. **Face Construction:** BRepBuilderAPI_MakeFace assumes the wire lies on a plane. For cut tubes, the end profile is planar (a circle). If the tube was cut at an angle (miter), the profile is an ellipse, which is still planar.
4. **Integration:** The new faces are sewed to the original shell.

8.2 STEP Writer Diagnostics

The STEPControl_Writer provides diagnostic output. Snippets [33] suggest managing verbosity. While useful for debugging, in a production environment, redirecting std::cout stream buffers (as shown in [34]) prevents console spam during batch processing.

9. Conclusion

This research report has detailed the complete pipeline for generating valid STEP 3D models from tube fabrication data.

1. **YBC Reconstruction:** By treating the tube as a kinematic chain and utilizing BRepPrimAPI primitives with rigorous vector alignment, we can produce exact, analytic solid models. This method is preferred for manufacturing verification as it precisely represents the machine's theoretical output.
2. **Centerline Reconstruction:** For arbitrary paths, GeomAPI_PointsToBSpline combined with BRepOffsetAPI_MakePipeShell allows for flexible shape generation. The use of Corrected Frenet frames is crucial to avoid twisting artifacts.
3. **Data Exchange:** The STEPControl_Writer configured for AP214 ensures that these solids

are universally compatible with modern PLM and CAD systems.

The provided C++ code architecture encapsulates these findings into a modular TubeBuilder class, ready for integration into industrial CAM software solutions.

References & Data Sources

- **OCCT Primitives:** [13]
- **Coordinate Systems:** [1]
- **Sweeping & PipeShell:** [8]
- **STEP Export:** [29]
- **Topology & Sewing:** [13]
- **Math & Transformation:** [5]

Works cited

1. XYZ, YBC, and LRA: understanding coordinate systems for CNC tube bending, accessed February 1, 2026, https://www.thefabricator.com/tubepipejournal/article/tubepipefabrication/xyz-ybc-and-lra-understanding-coordinate-systems-for-cnc-tube-bending
2. The difference between YBC data and XYZ data of pipe - Nanjing BLMA Machinery Co.,Ltd., accessed February 1, 2026, https://www.chinablma.com/blog/the-difference-between-ybc-data-and-xyz-data-of-pipe_b110
3. Schematic diagram of YBC data description of tube - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/figure/Schematic-diagram-of-YBC-data-description-of-tube_fig1_366404836
4. Tube Bending Formulas & Common Terms | Tubular Components Design - McHone Industries, accessed February 1, 2026, https://www.mchoneind.com/blog/tube-bending-formulas-tubular-components-design
5. Transformation matrix - Wikipedia, accessed February 1, 2026, https://en.wikipedia.org/wiki/Transformation_matrix
6. Algorithm for coordinate transformation from XYZ to Forward/Up/Right based on user-specified directions - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/79220732/algorithm-for-coordinate-transformation-from-xyz-to-forward-up-right-based-on-us
7. A Fast Conversion Method of Tube Coordinates - Semantic Scholar, accessed February 1, 2026, https://pdfs.semanticscholar.org/5e5b/9100cf819a63aa7afeff47d16465d4a2d91c.pdf
8. BRepOffsetAPI_MakePipe Class Reference - Open CASCADE Technology, accessed February 1, 2026,

https://dev.opencascade.org/doc/refman/html/class_b_rep_offset_a_p_i___make_pipe.html
9. Modeling Algorithms - Open CASCADE Technology, accessed February 1, 2026, https://old.opencascade.com/doc/occt-6.7.1/overview/html/occt_user_guides__modeling_algos.html
10. Modeling Data - Open CASCADE Technology, accessed February 1, 2026, https://old.opencascade.com/doc/occt-6.7.0/overview/html/user_guides__modeling_data.html
11. Modeling Data - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_data.html
12. Create Simple Cylinder from basic geom_ primitives - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/create-simple-cylinder-basic-geom-primitives
13. Modeling Algorithms - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__modeling_algos.html
14. Precision Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_precision.html
15. BRepPrimAPI_MakeCylinder Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_prim_a_p_i___make_cylinder.html
16. BRepPrimAPI_MakeTorus Class ... - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_prim_a_p_i___make_torus.html
17. Class Hierarchy - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/occt-6.9.1/refman/html/hierarchy.html
18. BRepAlgoAPI_Fuse Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_algo_a_p_i___fuse.html
19. Fusing and coplanar feces - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/fusing-and-coplanar-feces
20. Fuse cylindrical faces - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/fuse-cylindrical-faces
21. Create a solid from face intersection and cap holes - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/create-solid-face-intersection-and-cap-holes
22. BRepBuilderAPI_MakeFace Class Reference - Open CASCADE Technology, accessed February 1, 2026,

https://dev.opencascade.org/doc/refman/html/class_b_rep_builder_a_p_i___make_face.html
23. BRepBuilderAPI_MakeSolid Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_builder_a_p_i___make_solid.html
24. Data Structures - Open CASCADE Technology, accessed February 1, 2026, https://old.opencascade.com/doc/occt-7.1.0/refman/html/annotated.html
25. GeomAPI_PointsToBSpline Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_geom_a_p_i___points_to_b_spline.html
26. Disturbances (torsion ?) when building a pipe with BRepOffsetAPI_MakePipe, accessed February 1, 2026, https://dev.opencascade.org/content/disturbances-torsion-when-building-pipe-brepoffsetapimakepipe
27. BRepOffsetAPI_MakePipeShell Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_offset_a_p_i___make_pipe_shell.html
28. sweep (extrude) with a twist angle - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/sweep-extrude-twist-angle
29. STEPControl_Writer Class Reference - Open CASCADE Technology Documentation, accessed February 1, 2026, https://dev.opencascade.org/doc/occt-7.5.0/refman/html/class_s_t_e_p_control__writer.html
30. STEP Translator - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/overview/html/occt_user_guides__step.html
31. STEPControl_Writer Class Reference - Open CASCADE Technology, accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_s_t_e_p_control___writer.html
32. STEP processor - Open CASCADE Technology Documentation, accessed February 1, 2026, https://dev.opencascade.org/doc/occt-6.7.0/overview/html/user_guides__step.html
33. How to use STEPControl_Writer in openCascade.js? - Stack Overflow, accessed February 1, 2026, https://stackoverflow.com/questions/75603196/how-to-use-stepcontrol-writer-in-opencascade-js
34. STEPControl_Writer verbosity - Forum Open Cascade Technology, accessed February 1, 2026, https://dev.opencascade.org/content/stepcontrolwriter-verbosity
35. BRepPrimAPI_MakeCone Class Reference - Open CASCADE Technology,

accessed February 1, 2026, https://dev.opencascade.org/doc/refman/html/class_b_rep_prim_a_p_i___make_cone.html
36. inc/BRepPrimAPI_MakeTorus.hxx · master · libs / opencascade - GitLab, accessed February 1, 2026, https://gitlab.iag.uni-stuttgart.de/libs/opencascade/-/blob/master/inc/BRepPrimAPI_MakeTorus.hxx