Osdag IFC Wrapper - Development Report

1. Project Overview

The default Osdag application generates 3D CAD models using the OpenCASCADE Technology (OCCT) kernel via `pythonocc-core`. This project aimed to extend Osdag's capabilities by implementing an **IFC (Industry Foundation Classes) export wrapper**. This allows the structural steel connections designed in Osdag to be seamlessly transferred to BIM (Building Information Modeling) software like Revit, Tekla Structures, or open-source viewers like BIMvision and BlenderBIM.

The wrapper was designed to be modular and "plug-and-play," capable of sitting alongside existing Osdag CAD logic without requiring deep modifications to the core geometric engine.

2. Methodology

2.1 Architectural Approach

The wrapper is built around the `IfcOpenShell` library (specifically targeting the `IFC4` schema) to construct the BIM data model. The core class `IFCWrapper` handles the entire lifecycle of the export process:

1. **File Initialization**: Creates a new `.ifc` file with a standard header.
2. **Project Structure**: Defines the mandatory spatial hierarchy (`IfcProject` -> `IfcSite` -> `IfcBuilding` -> `IfcBuildingStorey`).
3. **Geometry Translation**: Converts Osdag's raw `TopoDS_Shape` objects into IFC geometric representations.
4. **Semantic Enrichment**: Assigns meaningful IFC entity types based on component names.

2.2 Core Geometry Extraction (Manual Triangulation)

A critical design decision was to implement a **Manual Triangulation Engine** rather than relying on high-level serialization functions (like `ifcopenshell.geom.serialize`). This ensures the wrapper works reliably across different Python environments, regardless of how `IfcOpenShell` was compiled.

The extraction process follows these steps:

1. **Meshing**: The input `TopoDS_Shape` is first tessellated using `BRepMesh_IncrementalMesh` with a deflection parameter (set to 1.0mm) to control smoothness.
2. **Face Exploration**: The wrapper iterates over every topological face in the shape using `TopExp_Explorer`.
3. **Triangulation Retrieval**: For each face, the underlying triangulation is extracted using `BRep_Tool.Triangulation`.
4. **Coordinate Transformation**: Vertex coordinates are transformed to the correct global location using `TopLoc_Location`.
5. **IFC Reconstruction**: The triangles are rebuilt as `IfcPolyLoop` entities, which form `IfcFace` objects. These faces are then assembled into an `IfcClosedShell`, resulting in a final `IfcFacetedBrep` representation.

This low-level approach guarantees 100% geometry fidelity and independence from external dependencies.

2.3 Semantic Mapping

To make the IFC files useful for downstream engineering, generic shapes are classified into specific structural entities. The wrapper implements an inference engine (`_infer_ifc_class`) that maps Osdag component names to IFC classes:

Osdag Component Keyword	Mapped IFC Entity	Description
"Beam"	`IfcBeam`	Horizontal structural members
"Column"	`IfcColumn`	Vertical structural members
"Plate", "Gusset"	`IfcPlate`	Connection plates and stiffeners
"Bolt"	`IfcMechanicalFastener`	Bolts, nuts, and washers
(Default)	`IfcBuildingElementProxy`	Generic/Other geometry

2.4 Spatial Hierarchy & Context

Simply exporting geometry is not enough for BIM. The wrapper explicitly constructs a valid spatial tree compliant with ISO 16739 (IFC standard):

- **IfcProject**: The root of the data exchange.
- **IfcSite**: Represents the logical site location.
- **IfcBuilding**: Containers for the structure.
- **IfcBuildingStorey**: Grouping mechanism (e.g., "Level 0").

Crucially, an `IfcGeometricRepresentationContext` is created and linked to the project. This defines the 3D coordinate system and precision (1e-05), ensuring that BIM viewers can correctly interpret the scale and orientation of the model.

3. Implementation Details

The wrapper resides in `src/osdag/cad/ifc_wrapper/ifc_exporter.py` . Key implementation highlights include:

- **Robust Error Handling**: The geometry extraction is wrapped in try-catch blocks per component. If a single complex shape fails to mesh, it logs an error but continues exporting the rest of the model.
- **Viewer Compatibility**: By using `IfCFacetedBrep` with `IfcPolyLoop` , the output files are compatible with strict viewers that might reject simpler wireframe representations.
- **Unit Management**: The project explicitly sets SI units (Millimeters for length), aligning with Osdag's internal unit system.

4. Challenges & Solutions

Challenge 1: Library Version Incompatibility

Issue: Standard pip-installable versions of `IfcOpenShell` often lack the PythonOCC bridge (`geom.serialize`).

Code relying on this function would crash in users' environments unless they compiled the library from source.

Solution: I developed the **Manual Triangulation Engine** (described in Section 2.2). This uses pure PythonOCC calls to extract raw vertex data, bypassing the need for the serialization function entirely, making the wrapper truly portable.

Challenge 2: BIM Viewer Visibility (The "Empty Model" Bug)

Issue: Initial exports contained valid geometric data but appeared "empty" or invisible in viewers like BIMvision.

Solution: Investigation revealed that many viewers require a strictly defined `IfcGeometricRepresentationContext` linked to the `IfcProject`. I added code to explicitly generate this context with a defined World Coordinate System.

Challenge 3: Mesh Density vs. Performance

Issue: Detailed components like bolts could result in millions of triangles, bloating the IFC file size.

Solution: I optimized the `BRepMesh_IncrementalMesh` deflection parameter. A value of **1.0mm** was found to provide the best balance—smooth enough for bolt heads but coarse enough to keep file sizes small.

5. Results

The wrapper was successfully tested against simulated Osdag output for four key modules:

1. Beam-to-Column End Plate Connection (`Osdag_Output_BCEndplate.ifc`)
2. Column-to-Column Splice (`Osdag_Output_CCSpliceCoverPlateCAD.ifc`)
3. Beam-to-Beam Splice (`Osdag_Output_BBCad.ifc`)
4. Tension Member (`Osdag_Output_Tension.ifc`)

All files were verified in **BIMvision**, confirming correct geometry, hierarchy, and semantic classification.

6. References

- Osdag Official Documentation
- IfcOpenShell-Python API Docs
- PythonOCC-Core Documentation
- BuildingSMART IFC4 Specification
- BIMvision Viewer