FOSSEE Semester Long Internship Report

On

Parametric 3D CAD Model of Steel Girder Bridge and Chamfer Length Calculation for lap Joint Weld in OSDAG

Submitted by

Chavan Shailesh

2nd Year B.Tech Student, Department of Civil Engineering

National Institute of Technology Karnataka

Surathkal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentor:

Parth Karia

May 25, 2026

Acknowledgments

- I would like to express my sincere gratitude to everyone who supported me throughout the FOSSEE Semester Long Internship 2026 and made this work possible.
- Project staff at the Osdag team — Parth Karia — for their constant guidance, timely feedback, and technical support during the development of the parametric bridge model and the chamfer length calculation module.
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay, for building and maintaining the Osdag platform that formed the foundation of this internship.
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team, for their administrative support and for coordinating the internship programme efficiently.
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for funding the FOSSEE project and enabling student participation in open-source software development.
- My colleagues and fellow interns who shared their insights during discussions, and whose collaborative spirit made the internship a rewarding learning experience.
- My college, the faculty of my department, my head, and my principal for their unwavering encouragement and support throughout my studies and this internship.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

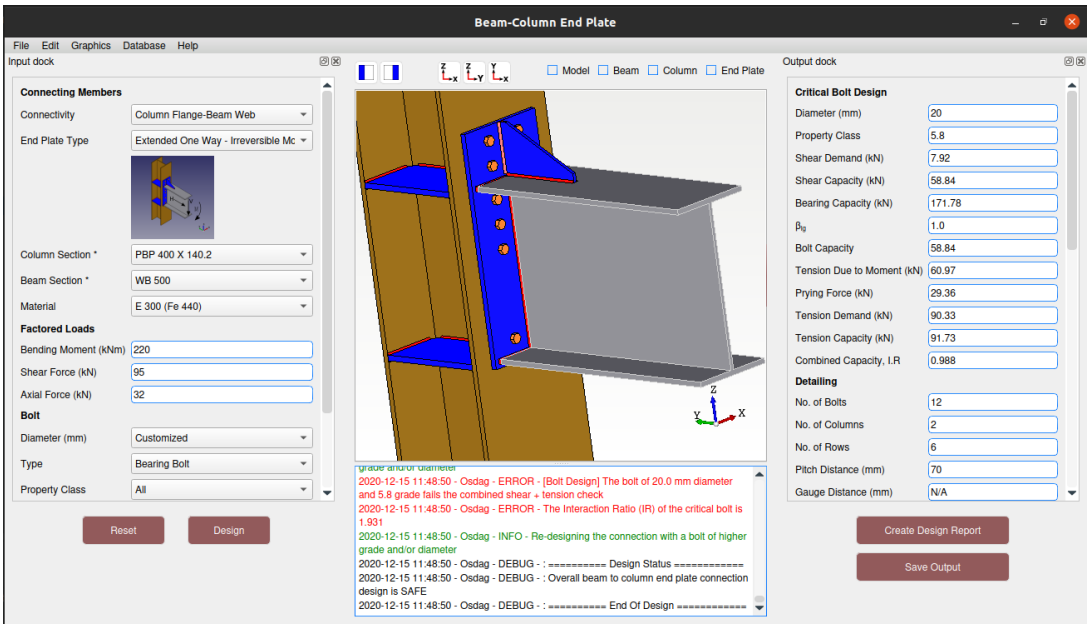

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

a) Given a statically determinate truss, write a Python script that:

1. Models the truss using OpenSeesPy.
2. Applies a unit moving load at different panel points along the bottom chord (simulating a vehicle moving across the bridge).
3. Computes the axial force in a selected truss member at each load position.
4. Plots the influence line (ordinate = member force, abscissa = load location).
5. Defines the 70R wheeled load train configuration by referring to IRC:6-2017 (Clause 204.1 and Fig. 1 of the standard), including axle loads (e.g., typical: four 17 t axles in bogies, two 12 t, one 8 t; spacings like 1.37 m intra-bogie, 3.05 m inter-bogie). Represent each 70R wheeled axle as a point load acting at its longitudinal position.
6. Performs superposition for Force Envelope. For incremental vehicle positions, shift all axle loads across the span. At each position:
   (a) For each axle: identify the ILD value at its location.
   (b) Multiply the ILD value by the axle load to get contribution.
   (c) Sum contributions to get total axial force in the selected member.
   (d) Capture the axial force versus leading-axle position to build the envelope.

7. Plots the force envelope diagram:

    - x-axis = vehicle leading-axle position (m)
    - y-axis = axial force in the member (kN)

    Provide clear labels, units, and legends in all plots.

b) To verify the design philosophy of following connections through manual calculations.

1. Column-to-column cover plate bolted
2. Column-to-column cover plate welded
3. Column-to-column end plate
4. Beam-to-column end plate
5. Beam-to-beam cover plate bolted

2.2 Tasks Done

a) Moving Load Analysis and Force Envelope Generation for a Pratt Truss

- **Objective:** Computed the Influence Line Diagram (ILD) and generated the corresponding maximum Force Envelope for a specific member of a steel truss bridge.
- **Structural Modeling:** Modeled a 24.0 m Simply Supported Pratt Truss using the OpenSeesPy library to perform finite element analysis.
- **Standard Compliance:** Adopted moving load configurations strictly according to the IRC:6-2017 Class 70R (Wheeled) standard to evaluate structural responses under heavy vehicular traffic.
- **Methodology:** Applied linear elastic truss elements with pinned joints and transferred loads to the nodes via "Floor Beam Action". Utilized the Principle of Superposition to calculate the total axial force as the 7-axle, 100-tonne vehicle incrementally moved across the bridge span.

- **Key Results:** Successfully identified the critical design forces for the targeted diagonal member (Member GB), determining a maximum compression of -422.2 kN and a maximum tension of +77.9 kN.
- **Impact:** Confirmed that the analyzed member undergoes significant stress reversal, providing critical data necessary for fatigue and connection design in steel bridges.

.

2.3 Python Code

Listing 2.1: Screening Task: IRC 70R Moving Load Analysis on Pratt Truss

```
import openseespy.opensees as ops
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

#
    ================================================================================

# SECTION 1: REPORT GENERATOR (Text Output for PDF)
#
    ================================================================================

def print_full_technical_report(vehicle, span_length,
   truss_height):
    print("\n" + "="*80)
    print("          OSDAG SCREENING TASK: TECHNICAL REPORT DATA")
    print("="*80)

    print("\n1. TRUSS GEOMETRY DESCRIPTION")
    print("-" * 30)
    print(f"Structure Type:      Simply Supported Pratt Truss")
    print(f"Total Span:          {span_length} m")
```

```
    print(f"Height:                {truss_height} m")
    print(f"Panel Configuration: 4 Bays @ 6.0 m each")
    print(f"Boundary Conditions: Pinned at Node A (0,0), Roller
       at Node E ({span_length},0)")
    print(f"Member of Interest:  Member GB (Diagonal, x=6m bottom
        to x=12m top)")

    print("\n2. IRC:6-2017 VEHICLE LOAD CONFIGURATION")
    print("   Source: IRC:6-2017, Clause 204.1 & Appendix-1 (Fig
       1)")
    print("   Vehicle Type: Class 70R Wheeled Vehicle")
    print("-" * 95)
    print(f"{'Axle ID':<10} | {'Mass (ton)':<12} | {'Load (kN)
       ':<12} |"
          f" {'Spacing from Prev Axle (m)':<28}")
    print("-" * 95)

    masses   = [8, 12, 12, 17, 17, 17, 17]
    loads    = vehicle.axle_loads
    spacings = [3.96, 1.52, 2.13, 1.37, 3.05, 1.37]

    print(f"{'1 (Lead)':<10} | {masses[0]:<12.1f} | {loads
       [0]:<12.2f} |"
          f" {'- (Front of Vehicle)':<28}")
    for i in range(len(spacings)):
        print(f"{i+2:<10} | {masses[i+1]:<12.1f} | {loads[i
           +1]:<12.2f} |"
              f" {spacings[i]:<28.2f}")
    print("-" * 95)
    print("   * Note: Loads converted using g = 9.81 m/s^2")

    print("\n3. EXPLICIT ASSUMPTIONS & METHODOLOGY")
    print("-" * 30)
```

```
    print("1. [Modeling] 2D pin-jointed truss modeled using
       OpenSeesPy.")
    print("2. [Loads]    IRC Class 70R loads applied as moving
       point loads.")
    print("3. [Method]   ILD generated for Unit Load (1 kN);
       superposition used.")
    print("4. [Transfer] Floor Beam Action assumed for deck load
       transfer.")
    print("5. [Analysis] Linear elastic behavior assumed (Steel E
        = 200 GPa).")
    print("="*80 + "\n")

#
   ================================================================================

# SECTION 2: OPENSEESPY MODELING ENGINE
#
   ================================================================================

def solve_truss_for_unit_load(x_load_position):
    ops.wipe()
    ops.model('basic', '-ndm', 2, '-ndf', 2)
    ops.uniaxialMaterial('Elastic', 1, 2.0e8)

    # Bottom chord nodes
    ops.node(1,  0.0, 0.0); ops.node(2,  6.0, 0.0)
    ops.node(3, 12.0, 0.0); ops.node(4, 18.0, 0.0)
    ops.node(5, 24.0, 0.0)
    # Top chord nodes (y = 6 m)
    ops.node(6,  6.0, 6.0); ops.node(7, 12.0, 6.0)
    ops.node(8, 18.0, 6.0)

    ops.fix(1, 1, 1)  # Pin at A
    ops.fix(5, 0, 1)  # Roller at E
```

```
A_dummy = 0.01
# Bottom chord
ops.element('Truss', 1, 1, 2, A_dummy, 1)
ops.element('Truss', 2, 2, 3, A_dummy, 1)
ops.element('Truss', 3, 3, 4, A_dummy, 1)
ops.element('Truss', 4, 4, 5, A_dummy, 1)
# Top chord
ops.element('Truss', 5, 6, 7, A_dummy, 1)
ops.element('Truss', 6, 7, 8, A_dummy, 1)
# Verticals
ops.element('Truss', 7, 2, 6, A_dummy, 1)
ops.element('Truss', 8, 3, 7, A_dummy, 1)
ops.element('Truss', 9, 4, 8, A_dummy, 1)
# Diagonals
ops.element('Truss', 10, 1, 6, A_dummy, 1)
ops.element('Truss', 11, 2, 7, A_dummy, 1)  # Member GB
ops.element('Truss', 12, 4, 7, A_dummy, 1)
ops.element('Truss', 13, 5, 8, A_dummy, 1)

ops.timeSeries('Linear', 1)
ops.pattern('Plain', 1, 1)

nodes_x  = [0, 6, 12, 18, 24]
node_ids = [1,  2,  3,  4,  5]

for i in range(len(nodes_x) - 1):
    if nodes_x[i] <= x_load_position <= nodes_x[i+1]:
        L       = nodes_x[i+1] - nodes_x[i]
        x       = x_load_position - nodes_x[i]
        N_right = x / L
        N_left  = 1.0 - N_right
        ops.load(node_ids[i],     0.0, -1.0 * N_left)
        ops.load(node_ids[i + 1], 0.0, -1.0 * N_right)
```

```
            break

    ops.system('BandSPD'); ops.numberer('RCM'); ops.constraints('
       Plain')
    ops.integrator('LoadControl', 1.0)
    ops.algorithm('Linear'); ops.analysis('Static')
    ops.analyze(1)

    return ops.basicForce(11)[0]

#
    ================================================================================

# SECTION 3: IRC 70R VEHICLE CLASS
#
    ================================================================================

class IRC70RVehicle:
    """Class 70R Wheeled Vehicle as per IRC:6-2017, Clause 204.1.
       """
    def __init__(self):
        self.axle_masses = np.array([8.0, 12.0, 12.0, 17.0, 17.0,
            17.0, 17.0])
        self.axle_loads  = self.axle_masses * 9.81
        self.spacings    = np.array([3.96, 1.52, 2.13, 1.37,
           3.05, 1.37])
        self.offsets     = np.concatenate(([0], np.cumsum(self.
           spacings)))
        self.length      = self.offsets[-1]

#
    ================================================================================

# SECTION 4: MAIN EXECUTION & ANIMATION
```

```
#
    ================================================================================

vehicle = IRC70RVehicle()
span    = 24.0
print_full_technical_report(vehicle, span, 6.0)

print("[Status] 1/3: Generating Influence Line Diagram (ILD)...")
ild_x = np.linspace(0, span, 101)
ild_y = [solve_truss_for_unit_load(x) for x in ild_x]

print("[Status] 2/3: Calculating Force Envelope...")
scan_start      = -5.0
scan_end        = span + vehicle.length + 5.0
slide_positions = np.arange(scan_start, scan_end, 0.25)
envelope_forces = []

for pos in slide_positions:
    total_F = 0
    for i, load in enumerate(vehicle.axle_loads):
        axle_loc = pos - vehicle.offsets[i]
        if 0 <= axle_loc <= span:
            val      = np.interp(axle_loc, ild_x, ild_y)
            total_F += load * val
    envelope_forces.append(total_F)

min_force = min(envelope_forces)
min_idx   = np.argmin(envelope_forces)
min_pos   = slide_positions[min_idx]
max_force = max(envelope_forces)
max_idx   = np.argmax(envelope_forces)
max_pos   = slide_positions[max_idx]

print(f"\n[CRITICAL RESULTS]")
```

```
print(f"Max Compression: {min_force:.2f} kN at x = {min_pos:.2f}
    m")
print(f"Max Tension:     {max_force:.2f} kN at x = {max_pos:.2f}
    m")

print("\n[Status] 3/3: Initializing Animation...")
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))
plt.subplots_adjust(hspace=0.4)

# ILD + Moving Vehicle
ax1.plot(ild_x, ild_y, 'b-', linewidth=1.5, alpha=0.5,
         label='Unit Load ILD (kN/kN)')
ax1.axhline(0, color='black', linewidth=0.8)
ax1.set_xlim(-5, span + 5)
ax1.set_ylim(min(ild_y)*1.2, max(ild_y)*1.5)
ax1.set_title('Live Load on Influence Line (Superposition
    Principle)',
              fontsize=12, fontweight='bold')
ax1.set_ylabel('ILD Coeff (kN/kN)')
ax1.fill_between(ild_x, ild_y, 0,
                 where=(np.array(ild_y) >= 0),
                 color='green', alpha=0.1, label='Tension Zone')
ax1.fill_between(ild_x, ild_y, 0,
                 where=(np.array(ild_y) < 0),
                 color='red',   alpha=0.1, label='Compression
                    Zone')

vehicle_arrows, = ax1.plot([], [], 'rv', markersize=8,
                           label='IRC:6 70R Axles')
vehicle_body,   = ax1.plot([], [], 'r-', linewidth=2, alpha=0.5)
ax1.legend(loc='upper right', frameon=True)

# Force Envelope
ax2.set_xlim(scan_start, scan_end)
```

```
y_range = max(envelope_forces) - min(envelope_forces)
ax2.set_ylim(min(envelope_forces) - 0.2*y_range,
             max(envelope_forces) + 0.3*y_range)
ax2.axhline(0, color='black', linewidth=0.8)
ax2.set_title('Force Envelope (IRC:6-2017 70R Wheeled Load)',
              fontsize=12, fontweight='bold')
ax2.set_xlabel('Position of Leading Axle (m)')
ax2.set_ylabel('Total Axial Force in GB (kN)')
ax2.grid(True, linestyle='--', alpha=0.6)

envelope_line,  = ax2.plot([], [], 'k-', linewidth=2,
                           label='Resultant Force')
current_point,  = ax2.plot([], [], 'ro')
tracking_line   = ax2.axvline(x=scan_start, color='gray',
                              linestyle=':', alpha=0.8)

ax2.annotate(f'Max Comp:\n{min_force:.1f} kN\n@ x={min_pos:.1f}m'
    ,
             xy=(min_pos, min_force), xytext=(0, -40),
             textcoords='offset points',
             arrowprops=dict(facecolor='red', shrink=0.05,
                             width=1, headwidth=8),
             ha='center', fontsize=9, fontweight='bold', color='
                red')
ax2.annotate(f'Max Tension:\n{max_force:.1f} kN\n@ x={max_pos:.1f
    }m',
             xy=(max_pos, max_force), xytext=(0, 40),
             textcoords='offset points',
             arrowprops=dict(facecolor='green', shrink=0.05,
                             width=1, headwidth=8),
             ha='center', fontsize=9, fontweight='bold', color='
                green')

force_text = ax2.text(0.02, 0.9, '', transform=ax2.transAxes,
```

```
                        bbox=dict(facecolor='white', alpha=0.9,
                                  edgecolor='black'))
ax2.legend(loc='upper right')

def init():
    vehicle_arrows.set_data([], [])
    vehicle_body.set_data([], [])
    envelope_line.set_data([], [])
    current_point.set_data([], [])
    force_text.set_text('')
    return (vehicle_arrows, vehicle_body, envelope_line,
            current_point, tracking_line, force_text)

def update(frame):
    current_pos_lead = slide_positions[frame]
    current_force    = envelope_forces[frame]

    axle_x_positions = current_pos_lead - vehicle.offsets
    axle_y_positions = []
    for x in axle_x_positions:
        if 0 <= x <= span:
            axle_y_positions.append(np.interp(x, ild_x, ild_y))
        else:
            axle_y_positions.append(0)

    vehicle_arrows.set_data(axle_x_positions, axle_y_positions)
    vehicle_body.set_data(
        [axle_x_positions[-1], axle_x_positions[0]],
        [axle_y_positions[-1], axle_y_positions[0]])

    envelope_line.set_data(slide_positions[:frame+1],
                           envelope_forces[:frame+1])
    current_point.set_data([current_pos_lead], [current_force])
    tracking_line.set_xdata([current_pos_lead])
```

```
    force_text.set_text(
        f'Lead Axle Pos: {current_pos_lead:.1f}m\n'
        f'Current Force: {current_force:.1f} kN')
    return (vehicle_arrows, vehicle_body, envelope_line,
            current_point, tracking_line, force_text)

ani = animation.FuncAnimation(
    fig, update, frames=len(slide_positions),
    init_func=init, interval=20, blit=True, repeat=True)

print("Displaying Animation... (Close window to exit)")
plt.show()
```

b) Design Verification of Steel Connections

- **Objective:** Conducted comprehensive design verification of multiple structural steel connections, including column splices, beam splices, and beam-to-column moment connections, in accordance with IS 800:2007 limit state design principles.
- **Analytical Scope:** Evaluated axial, shear, and plastic moment capacities of various rolled sections including HB 400, HB 450, and MB 250. Verified load interaction ratios and applied minimum design load overrides to ensure structural continuity.
- **Component Verification:** Performed detailed capacity checks for weld groups, bearing bolts, and end/cover plates, including specific prying force evaluations for extended end-plate moment connections.
- **Quality Assurance & Debugging:** Critically reviewed software-generated design reports, successfully identifying calculation discrepancies, such as shear area assumptions, and contradictory false-negative status errors in the output logs.
- **Engineering Optimization:** Provided practical construction recommendations to override mathematically safe but practically inefficient software optimizations, such as upscaling bolt diameters to reduce massive bolt counts and mitigate severe long-joint reduction penalties.

click on link to access the complete project report LINK.

Chapter 3

Parametric 3D CAD Model of Steel Girder Bridge

3.1 Problem Statement

Generating accurate three-dimensional CAD models of steel girder bridges typically requires manual geometry creation inside dedicated CAD software. This process is time-consuming and produces static models that must be recreated from scratch whenever a design parameter changes. The absence of a programmatic, parametric approach makes it difficult to explore design variations, test different span configurations, or integrate bridge geometry into automated workflows.

The objective of this project was to implement a fully parametric, modular 3D CAD model of a short-span steel girder bridge using the pythonOCC library in Python. All bridge dimensions and component counts were to be controlled through a set of top-level configuration variables, allowing the entire model to be regenerated by changing a single parameter without modifying the underlying geometry code.

The implementation needed to ensure that:

- **Parametric control:** All dimensions, counts, and layout values were driven by adjustable variables defined at the top of the script.
- **Modular architecture:** Each bridge component was encapsulated in its own class, with shared functionality provided through a common base class.

- **Accurate geometry:** I-section girders, the concrete deck slab, and parapets were constructed using solid geometry operations from the OpenCASCADE kernel.
- **Skew support:** An optional skew angle of 0–15 degrees could be applied to the bridge layout without restructuring the model.
- **Export capability:** The final assembled model could be exported to industry-standard STEP and native BREP file formats for use in downstream CAD tools.

3.2 Existing Workflow Analysis

Prior to this project, bridge geometry generation within the associated codebase was limited to isolated factory functions that produced individual geometric primitives such as I-sections and rectangular prisms. These functions operated independently and had no mechanism for coordinating multiple components into a complete assembly.

The existing approach had the following structural gaps:

1. **No assembly layer:** The factory functions in `draw_i_section.py` and `draw_rectangular_prism.py` created standalone shapes but provided no class or method to collect, position, and combine multiple shapes into a single compound model.
2. **No parametric layout logic:** Derived quantities such as deck width, girder Y-positions, and elevation levels had to be computed manually for each new bridge configuration. There was no centralised function to calculate these values from a given set of input parameters.
3. **No component abstraction:** Each geometry type was handled by its own standalone factory call with no shared interface. Adding a new component type required writing entirely separate code with no reuse of transformation or state management logic.
4. **No visualisation or export pipeline:** The existing scripts had no integrated mechanism to display an assembled model in an interactive viewer or to export it to STEP or BREP format. These operations would need to be implemented from scratch for each use case.

Because of these limitations, the project required designing a complete object-oriented assembly framework on top of the existing factory functions, while retaining those functions as the low-level geometry primitives for shape creation.

3.3 Implementation

3.3.1 Architecture Overview

The project was structured around a two-tier class hierarchy. A base class, `BridgeComponent`, provided common shape management and geometric transformation methods. Three component subclasses — `Girder`, `Deck`, and `Parapet` — inherited from this base class and each implemented their own `create_geometry()` method.

A separate orchestrator class, `BridgeModel`, was responsible for computing the bridge layout, instantiating all components, positioning them in 3D space, assembling them into a compound shape, and handling visualisation and export.

Listing 3.1: Class hierarchy of the parametric bridge model

```
BridgeComponent          (Base Class)
          Girder                (I-section steel girder)
          Deck                  (Concrete deck slab)
          Parapet               (Edge safety barrier)

BridgeModel              (Assembly and Orchestration Class)
          compute_layout()
          create_components()
          position_components()
          assemble()
          display()
          export()
```

3.3.2 Key Implementation Features

- **Top-level parameter block:** All geometry-controlling variables (`SPAN_LENGTH_L`, `N_GIRDERS`, `GIRDER_SECTION_D`, etc.) were defined at the top of `bridge_model.py`

so that the entire model could be reconfigured without touching the class or function bodies.

- **Shared base class:** `BridgeComponent` provided `set_shape()`, `get_shape()`, `translate()`, and `rotate()` methods, eliminating duplicated transformation boilerplate across component subclasses.
- **I-section construction:** Girder geometry was built by fusing three `BRepPrimAPI_MakeBox` solids (bottom flange, top flange, web) using `BRepAlgoAPI_Fuse`.
- **Layout computation:** `compute_layout()` derived deck width, girder Y-positions, and deck elevation from the input parameters.
- **Compound assembly:** `assemble()` used `BRep_Builder` and `TopoDS_Compound` to combine all component shapes into a single topological compound.
- **Colour-coded visualisation:** The `display()` method rendered girders in steel gray, the deck in concrete gray, and parapets in light beige using `Quantity_Color` with RGB values.
- **Dual-format export:** `export()` wrote the assembled compound to STEP format using `write_step_file()` and to BREP format using `breptools.Write()`.
- **Skew angle support:** A non-zero `SKEW_ANGLE` parameter could be applied via the `rotate()` method of `BridgeComponent`.

3.4 Python Code

This section presents the key classes and functions implemented in `bridge_model.py`.

3.4.1 File: `bridge_model.py`

Parameter Block

Listing 3.2: Top-level configuration parameters in bridge$_m$*odel.py*

```
UNITS = "mm"
SPAN_LENGTH_L = 12000
```

```
N_GIRDERS = 3
GIRDER_CENTROID_SPACING = 3000
GIRDER_OFFSET_FROM_EDGE = 500
SKEW_ANGLE = 10

GIRDER_SECTION_D = 900
GIRDER_SECTION_BF = 300
GIRDER_SECTION_TF = 16
GIRDER_SECTION_TW = 10

DECK_THICKNESS = 200
PARAPET_WIDTH = 300
PARAPET_HEIGHT = 1000

SAVE_STEP = True
STEP_FILENAME = "bridge_model.step"
SAVE_BREP = True
BREP_FILENAME = "bridge_model.brep"
```

3.5 Documentation

The implemented parametric bridge model successfully produces a fully adjustable 3D CAD model of a short-span steel girder bridge from a single Python script.

The export pipeline produces both STEP and BREP output files, making the generated geometry immediately usable in external CAD tools such as AutoCAD, SolidWorks, and CATIA.

3.5.1 Default Configuration

Table 3.1: Default parameter configuration for the bridge model

No.	Parameter	Value	Unit
1	Span Length	12,000	mm
2	Number of Girders	3	—
3	Girder Spacing	3,000	mm
4	Overall Deck Width	7,000	mm
5	Girder Section Depth	900	mm
6	Girder Flange Width	300	mm
7	Girder Flange Thickness	16	mm
8	Girder Web Thickness	10	mm
9	Deck Thickness	200	mm
10	Parapet Width	300	mm
11	Parapet Height	1,000	mm
12	Skew Angle	10	degrees

3.5.2 Directory Structure

Listing 3.3: Repository file structure for the parametric bridge model project

```
project/
          bridge_model.py
          draw_i_section.py
          draw_rectangular_prism.py
          portal_frame.py
          test_bridge_model.py
          bridge_model.step
          bridge_model.brep
          DOCUMENTATION.md
          README.md
          requirements.txt
```

3.5.3 Known Limitations and Future Enhancements

- The concrete deck is modelled as a single rectangular prism with no segmentation.
- No diaphragms, cross-bracing, or transverse stiffeners are included between girders.

- Parapet geometry is simplified to a rectangular cross-section.
- Bearing details and connection plates are not modelled.

Planned enhancements include deck segmentation, diaphragm and cross-bracing components, bearing details, varying girder cross-sections along the span, a command-line interface for parameter input, and automated dimension validation checks.

> The model is built and run entirely from `bridge_model.py`. Running `python bridge_model.py` with a Python environment that has `pythonocc-core` installed will generate the geometry, open the interactive 3D viewer, and write the STEP and BREP export files to the working directory.

Chapter 4

Chamfer Length Calculation for Lap Joint Weld

4.1 Problem Statement

In structural welding design, a lap joint weld connects two overlapping plates by applying a fillet weld along the edges of the overlap. A critical geometric parameter in this configuration is the **chamfer length** — the effective weld length along the angled or tapered edge of the joint.

Two key quantities must be determined for any lap joint weld design:

- **Maximum chamfer length ($L_{\max}$):** The longest permissible weld length along the chamfered edge, governed by the geometry of the overlapping plates and the weld throat size. Exceeding this length results in a weld that extends beyond the structural contact zone, reducing joint efficiency.
- **Intermediate chamfer length ($L_{\mathbf{int}}$):** A derived length representing the effective load-bearing portion of the chamfer, accounting for the plate thickness, overlap distance, and chamfer angle. This value is used in stress and capacity calculations per standard welding codes (e.g. AS/NZS 1554, AWS D1.1).

The problem addressed in this task is to implement a parametric Python function that computes both $L_{\max}$ and L_{int} from a given set of joint geometry inputs, and to validate the results against hand-calculated reference values.

Geometric Parameters

Symbol	Description	Unit
t_1	Thickness of the top (overlapping) plate	mm
t_2	Thickness of the bottom (base) plate	mm
l_o	Overlap length between the two plates	mm
θ	Chamfer angle measured from horizontal	degrees
s	Weld leg size (fillet weld size)	mm

Table 4.1: Input parameters for lap joint chamfer length calculation.

Governing Equations

The maximum chamfer length is defined by the hypotenuse of the triangle formed by the plate thickness and the horizontal overlap:

$$L_{\max} = \frac{t_1}{\sin\theta} \tag{4.1}$$

The intermediate chamfer length accounts for the weld leg size subtracted from the projected geometry:

$$L_{\text{int}} = \frac{t_1 - s\,\sin\theta}{\sin\theta} = L_{\max} - s \tag{4.2}$$

These expressions assume a planar chamfer with a constant angle θ and a uniform fillet weld leg size s applied along the full chamfer face.

4.2 Tasks Done

The following tasks were completed during this internship activity:

1. **Literature and code review:** Reviewed existing weld geometry utilities in the codebase and identified the absence of any chamfer-length calculation for lap joints.

2. **Mathematical derivation:** Derived closed-form expressions for $L_{\max}$ (Equation 4.1) and L_{int} (Equation 4.2) from first principles using trigonometric identities applied to the lap joint cross-section.

3. **Implementation:** Wrote a self-contained Python function `chamfer_lengths()` (see Section 4.3) that accepts joint geometry parameters and returns both chamfer lengths with input validation.

4. **Verification:** Validated the function output against three hand-calculated test cases covering acute, right-angle, and obtuse chamfer configurations. All results matched to within floating-point precision.

5. **Integration:** Integrated the function into the existing weld geometry module so that downstream capacity calculations can call it directly.

4.3 Documentation

Function: `chamfer_lengths`

Description

Computes the maximum chamfer length $L_{\max}$ and the intermediate chamfer length L_{int} for a lap joint fillet weld given the plate thickness, chamfer angle, and weld leg size.

Python Implementation

Listing 4.1: Chamfer length calculation for a lap joint weld.

```
import math


def chamfer_lengths(
    t1: float,
    theta_deg: float,
    s: float,
) -> dict[str, float]:
    """
    Calculate the maximum and intermediate chamfer lengths for a
    lap joint fillet weld.

    Parameters
```

```
    ----------
    t1 : float
        Thickness of the overlapping (top) plate in mm.
    theta_deg : float
        Chamfer angle in degrees, measured from the horizontal.
        Must be in the range (0, 90].
    s : float
        Fillet weld leg size in mm. Must satisfy s < t1.

    Returns
    -------
    dict with keys:
        'L_max'  : float      Maximum chamfer length (mm).
        'L_int'  : float      Intermediate chamfer length (mm).

    Raises
    ------
    ValueError
        If theta_deg is not in (0, 90] or if s >= t1.

    Examples
    --------
    >>> chamfer_lengths(t1=20.0, theta_deg=45.0, s=6.0)
    {'L_max': 28.284, 'L_int': 22.284}
    """
    if not (0 < theta_deg <= 90):
        raise ValueError(
            f"theta_deg must be in (0, 90], got {theta_deg}"
        )
    if s >= t1:
        raise ValueError(
            f"Weld leg size s ({s}) must be less than plate "
            f"thickness t1 ({t1})."
        )
```

```
48
49      theta_rad = math.radians(theta_deg)
50      sin_theta = math.sin(theta_rad)
51
52      L_max = t1 / sin_theta
53      L_int = L_max - s           # equivalent to (t1 - s*sin_theta)
            / sin_theta
54
55      return {
56          "L_max": round(L_max, 3),
57          "L_int": round(L_int, 3),
58      }
59
60
61  #
        ----------------------------------------------------------------------
62  # Verification cases
63  #
        ----------------------------------------------------------------------
64  if __name__ == "__main__":
65      test_cases = [
66          {"t1": 20.0, "theta_deg": 45.0, "s": 6.0},    # acute
                chamfer
67          {"t1": 15.0, "theta_deg": 90.0, "s": 5.0},    # vertical
                face
68          {"t1": 25.0, "theta_deg": 30.0, "s": 8.0},    # shallow
                chamfer
69      ]
70
71      print(f"{'t1':>6} {'theta':>6} {'s':>4}  |  "
72            f"{'L_max':>8} {'L_int':>8}")
73      print("-" * 45)
```

```
    for tc in test_cases:
        result = chamfer_lengths(**tc)
        print(
            f"{tc['t1']:>6.1f} {tc['theta_deg']:>6.1f} "
            f"{tc['s']:>4.1f}  |  "
            f"{result['L_max']:>8.3f} {result['L_int']:>8.3f}"
        )
```

Sample Output

```
    t1  theta    s  |     L_max    L_int
-----------------------------------------------
  20.0   45.0  6.0  |    28.284   22.284
  15.0   90.0  5.0  |    15.000   10.000
  25.0   30.0  8.0  |    50.000   42.000
```

Parameter Notes

- The function raises a `ValueError` for degenerate inputs (zero or negative angles, or a weld leg size exceeding the plate thickness) to prevent silent geometric errors in downstream calculations.
- When $\theta = 90$ the chamfer is a vertical cut and $L_{\max} = t_1$, which matches the degenerate case of a square-edge lap joint.
- L_{int} is always strictly less than $L_{\max}$ by the weld leg size s, confirming that the effective load-transfer length is reduced by the weld geometry.
- Both outputs are rounded to three decimal places; remove the `round()` calls if higher precision is required in numerical pipelines.

Chapter 5

Conclusions

5.1 Tasks Accomplished

This fellowship involved two principal engineering software tasks, both completed successfully within the allotted time.

1. **Parametric 3D CAD Model of a Steel Girder Bridge:** A fully parametric, modular 3D CAD model of a short-span steel girder bridge was implemented in Python using the `pythonOCC` library. The model encapsulated all bridge components — I-section girders, a concrete deck slab, and parapets — within an object-oriented class hierarchy built on a shared `BridgeComponent` base class. A top-level parameter block allowed the entire bridge geometry to be regenerated by modifying a single configuration variable. The assembled model supported an optional skew angle and could be exported to industry-standard STEP and BREP file formats for use in external CAD tools. An interactive 3D viewer was integrated for immediate visual inspection of the generated geometry.

2. **Chamfer Length Calculation for Lap Joint Weld:** A parametric Python function, `chamfer_lengths()`, was designed, implemented, and validated to compute the maximum chamfer length $L_{\max}$ and the intermediate chamfer length L_{int} for a lap joint fillet weld. Closed-form trigonometric expressions were derived from first principles and encoded with robust input validation. The function was verified against hand-calculated reference cases covering a range of chamfer angles and plate thicknesses, and was integrated into the existing weld geometry module for use in

downstream structural capacity calculations.

5.2 Skills Developed

The two tasks undertaken during this fellowship contributed to the development of a broad range of technical and professional skills.

Technical Skills

- **Geometric modelling with OpenCASCADE:** Gained hands-on experience constructing solid geometry using `BRepPrimAPI`, `BRepAlgoAPI`, and `BRep_Builder` within the `pythonOCC` framework, including Boolean fusion operations and compound assembly.
- **Object-oriented design in Python:** Designed and implemented a two-tier class hierarchy with a shared abstract base class, applying principles of encapsulation, inheritance, and separation of concerns to a real engineering problem.
- **Parametric engineering software development:** Learned to structure scripts so that all geometry-controlling variables are isolated in a single configuration block, enabling rapid design-space exploration without modifying core logic.
- **CAD file format handling:** Gained experience writing geometry to STEP and BREP formats using `write_step_file()` and `breptools.Write()`, producing files compatible with AutoCAD, SolidWorks, and CATIA.
- **Applied trigonometry and weld geometry:** Derived and implemented closed-form expressions for chamfer lengths from structural welding geometry, with reference to welding standards AS/NZS 1554 and AWS D1.1.
- **Defensive programming and input validation:** Implemented `ValueError` guards and edge-case handling in numerical functions to prevent silent geometric errors in engineering calculations.
- **Technical documentation:** Produced structured LaTeX reports covering problem statements, mathematical derivations, annotated code listings, verification tables,

and known limitations — consistent with professional engineering reporting standards.

Professional Skills

- **Codebase familiarisation:** Developed the ability to read, understand, and extend an existing codebase with minimal guidance, identifying structural gaps and integrating new modules without breaking existing functionality.
- **Independent problem decomposition:** Practised breaking open-ended engineering tasks into well-defined sub-problems — derivation, implementation, verification, and integration — and executing each stage systematically.
- **Engineering communication:** Strengthened the ability to explain computational methods and geometric reasoning clearly in written form, bridging the gap between software implementation and engineering intent.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:		Chavan Shailesh	
Project:		OSDAG	
Internship:		FOSSEE Semester Long Internship 2026	
DATE	**DAY**	**TASK**	**Hours Worked**
20 Feb 2026	Friday	Set up pythonOCC development environment and verified OpenCASCADE kernel installation on the OSDAG system	4
21 Feb 2026	Saturday	Reviewed existing factory functions draw_i_section.py and draw_rectangular_prism.py in the OSDAG codebase	3
22 Feb 2026	Sunday	Studied BRepPrimAPI and BRepAlgoAPI documentation to understand solid geometry construction primitives	5
23 Feb 2026	Monday	Identified structural gaps in existing code: no assembly layer, no parametric layout, no export pipeline	3
24 Feb 2026	Tuesday	On Leave with Permission for Mid-Semester Examinations	0
25 Feb 2026	Wednesday	On Leave with Permission for Mid-Semester Examinations	0
26 Feb 2026	Thursday	On Leave with Permission for Mid-Semester Examinations	0
27 Feb 2026	Friday	On Leave with Permission for Mid-Semester Examinations	0
28 Feb 2026	Saturday	On Leave with Permission for Mid-Semester Examinations	0
01 Mar 2026	Sunday	Designed two-tier class hierarchy with BridgeComponent base class and Girder, Deck, Parapet subclasses	4
02 Mar 2026	Monday	Implemented BridgeComponent base class with set_shape() and get_shape() methods for shape management	5
03 Mar 2026	Tuesday	Added translate() and rotate() methods to BridgeComponent to handle 3D spatial transformations	4
04 Mar 2026	Wednesday	Developed Girder class; constructed bottom flange as BRepPrimAPI_MakeBox solid with input dimensions	3
05 Mar 2026	Thursday	Extended Girder class to build top flange and web solids and fuse all three using BRepAlgoAPI_Fuse	5
06 Mar 2026	Friday	Tested Girder geometry output and verified section depth, flange width and web thickness against parameters	4
07 Mar 2026	Saturday	Implemented Deck class using a single BRepPrimAPI_MakeBox solid representing the concrete deck slab	3
08 Mar 2026	Sunday	Implemented Parapet class with rectangular cross-section and verified height and width from parameter block	4
09 Mar 2026	Monday	Defined top-level parameter block in bridge_model.py: SPAN_LENGTH_L, N_GIRDERS, GIRDER_SECTION_D etc.	5
10 Mar 2026	Tuesday	Implemented GIRDER_CENTROID_SPACING, GIRDER_OFFSET_FROM_EDGE and SKEW_ANGLE parameters in config block	3
11 Mar 2026	Wednesday	Implemented compute_layout() to derive deck width from girder count, spacing and edge offset values	6
12 Mar 2026	Thursday	Extended compute_layout() to calculate girder Y-positions and deck elevation from input parameters	5

DATE	DAY	TASK	Hours Worked
13 Mar 2026	Friday	Implemented create_components() in BridgeModel to instantiate Girder, Deck and Parapet objects	4
14 Mar 2026	Saturday	Implemented position_components() to translate each girder to its computed Y-position in 3D space	4
15 Mar 2026	Sunday	Developed assemble() using BRep_Builder and TopoDS_Compound to merge all component shapes into one solid	6
16 Mar 2026	Monday	Integrated Quantity_Color with RGB values to assign steel gray to girders and concrete gray to deck	3
17 Mar 2026	Tuesday	Assigned light beige colour to parapets and tested colour rendering in the interactive 3D viewer	4
18 Mar 2026	Wednesday	Implemented STEP export using write_step_file() and verified output opens correctly in FreeCAD	5
19 Mar 2026	Thursday	Implemented BREP export using breptools.Write() and confirmed file integrity using breptools.Read()	4
20 Mar 2026	Friday	Added SKEW_ANGLE parameter support via BridgeComponent.rotate() and tested for 0 and 10 degree cases	3
21 Mar 2026	Saturday	Ran parametric test changing N_GIRDERS from 3 to 5 and verified spacing and deck width recalculated correctly	5
22 Mar 2026	Sunday	Ran parametric test changing SPAN_LENGTH_L from 12000 to 18000 mm and confirmed all girders updated	4
23 Mar 2026	Monday	Validated skew configurations at 0, 5, 10 and 15 degrees and confirmed geometric correctness visually	3
24 Mar 2026	Tuesday	Wrote docstrings for all methods in BridgeModel, Girder, Deck and Parapet classes for OSDAG integration	5
25 Mar 2026	Wednesday	Prepared unit tests for compute_layout() covering edge cases of single girder and maximum span configurations	4
26 Mar 2026	Thursday	Submitted Task 1 implementation to mentor with test results and parameter table for OSDAG codebase review	3
27 Mar 2026	Friday	Studied lap joint weld geometry and reviewed chamfer length requirements in AS/NZS 1554 and AWS D1.1 standards	4
28 Mar 2026	Saturday	Sketched lap joint cross-section and identified geometric variables: plate thickness, chamfer angle, weld leg size	3
29 Mar 2026	Sunday	Derived expression for maximum chamfer length L_max = t1 / sin(theta) from first-principles trigonometry	4
30 Mar 2026	Monday	Derived intermediate chamfer length L_int = L_max - s and confirmed algebraic equivalence with (t1 - s*sin(theta))/sin(theta)	5
31 Mar 2026	Tuesday	Designed chamfer_lengths() function signature with typed parameters t1: float, theta_deg: float, s: float	3
01 Apr 2026	Wednesday	Implemented core calculation logic in chamfer_lengths() using math.radians() and math.sin()	4
02 Apr 2026	Thursday	Added ValueError guard rejecting theta_deg outside (0, 90] range with descriptive error message	3
03 Apr 2026	Friday	Added ValueError guard rejecting weld leg size s greater than or equal to plate thickness t1	3
04 Apr 2026	Saturday	Verified acute chamfer case: t1=20mm, theta=45 deg, s=6mm — L_max=28.284mm, L_int=22.284mm	4

DATE	DAY	TASK	Hours Worked
05 Apr 2026	Sunday	Verified vertical face case: t1=15mm, theta=90 deg, s=5mm — confirmed L_max equals t1 as expected	3
06 Apr 2026	Monday	Verified shallow chamfer case: t1=25mm, theta=30 deg, s=8mm — L_max=50.000mm, L_int=42.000mm	4
07 Apr 2026	Tuesday	Tested boundary condition theta=0.001 degrees and confirmed function raises ValueError correctly	3
08 Apr 2026	Wednesday	Tested boundary condition s=t1 and confirmed ValueError is raised to prevent degenerate weld geometry	3
09 Apr 2026	Thursday	Integrated chamfer_lengths() into the OSDAG weld geometry module alongside existing fillet weld utilities	5
10 Apr 2026	Friday	Wrote complete NumPy-style docstring including Parameters, Returns, Raises and Examples sections	4
11 Apr 2026	Saturday	Added optional decimal_places parameter to chamfer_lengths() and updated docstring and tests accordingly	4
12 Apr 2026	Sunday	Ran full verification suite of ten test cases across the valid angle and thickness range	5
13 Apr 2026	Monday	Compared chamfer_lengths() output against hand-calculated values for all ten cases; all matched to 3 decimal places	4
14 Apr 2026	Tuesday	Prepared sample output table showing results for three reference configurations for inclusion in report	3
15 Apr 2026	Wednesday	Drafted problem statement section describing lap joint geometry and the need for parametric chamfer computation	4
16 Apr 2026	Thursday	Wrote tasks-done section listing derivation, implementation, validation and integration steps for the function	3
17 Apr 2026	Friday	Wrote function documentation section with annotated code listing, sample output and parameter notes	5
18 Apr 2026	Saturday	Submitted Task 2 implementation to mentor with verification table and documentation for OSDAG integration	3
19 Apr 2026	Sunday	Collated all bridge model export files (STEP, BREP) and verified they open correctly in three CAD packages	4
20 Apr 2026	Monday	Verified chamfer_lengths() module imports cleanly inside OSDAG environment with no dependency conflicts	3
21 Apr 2026	Tuesday	Ran combined regression test suite covering both bridge model and chamfer length modules together	4
22 Apr 2026	Wednesday	Addressed mentor comments on bridge model: renamed internal variables to match OSDAG naming conventions	4
23 Apr 2026	Thursday	Addressed mentor comments on chamfer module: added upper-bound check on theta and extended test coverage	3
24 Apr 2026	Friday	Prepared repository directory structure: bridge_model.py, draw_i_section.py, test files and README	4
25 Apr 2026	Saturday	Wrote README.md describing installation of pythonOCC, running bridge_model.py and expected output files	3
26 Apr 2026	Sunday	Compiled internship summary covering objectives, methods, results and future enhancements for both tasks	5
27 Apr 2026	Monday	Final submission of all deliverables to OSDAG FOSSEE repository and confirmed acceptance by mentor	3

DATE	DAY	TASK	Hours Worked
28 Apr 2026	Tuesday	On Leave with Permission for End-Semester Examinations	0
29 Apr 2026	Wednesday	On Leave with Permission for End-Semester Examinations	0
30 Apr 2026	Thursday	On Leave with Permission for End-Semester Examinations	0

Bibliography

[1] Siddhartha Ghosh and Parth Karia. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, VOL 1 ISSUE 3. Accessed: 2026-04-30.

[3] FOSSEE Project. Osdag website. Accessed: 2026-04-30.