FOSSEE Semester Long Internship Report

On

Development of Visualization and UI Enhancement for 3psLCCA Module in Osdag

Submitted by

Khusbu Pandey

ID: 23f2003897

BS in Data Science and Applications

IIT Madras

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Swastik

Renu P Sulakhe

Parth Karia

Internship Duration:

February 18, 2026 – May 18, 2026

Abstract

This report documents the internship work completed at FOSSEE (Free/Libre and Open Source Software for Education), IIT Bombay, under the guidance of Prof. Siddhartha Ghosh. The internship focused on enhancing the visualization capabilities and user interface of the 3psLCCA (Three Pillar Life Cycle Cost Analysis) module in Osdag.

The primary contributions include:

- Implementation of a centralized color system with interactive hover and external labels for pie charts
- Migration from matplotlib to pyqtgraph with 40% performance improvement
- Creation of publication-quality bar charts with stage-based panels
- Development of a Reingold-Tilford tree layout for hierarchical data visualization
- Comprehensive UI/UX enhancement ensuring professional report quality

The work involved extensive use of Python, PyQt6/PySide6, matplotlib, pyqtgraph, and D3.js for creating interactive visualizations. The implemented features enhance the usability and visual appeal of the LCCA module, making it more accessible to civil engineering professionals.

Keywords: Life Cycle Cost Analysis, Osdag, FOSSEE, PyQt, Data Visualization, Open Source Software

Acknowledgments

I would like to express my sincere gratitude to everyone who supported me during this internship:

- **Prof. Siddhartha Ghosh**, Principal Investigator, Osdag Project, IIT Bombay, for providing this opportunity and guidance.
- **Swastik**, Mentor, for mentoring me throughout the internship, providing technical guidance, and reviewing my work.
- **Renu P Sulakhe**, Co-Mentor, for her detailed feedback on UI/UX improvements and patient guidance on color theory and visualization best practices.
- The entire Osdag and 3psLCCA development team at FOSSEE for their support and collaboration.
- **FOSSEE Project**, funded by the National Mission on Education through ICT (NMEICT), Ministry of Education, Government of India, for this valuable opportunity.
- My college and professors for their encouragement and support.

This internship has been an invaluable learning experience, and I am grateful for the opportunity to contribute to open-source software in the field of civil engineering.

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.

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

1.2 FOSSEE Project [4]

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 [4] 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.
- **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

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 [2].

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.
- **Message Log**: Shows errors, warnings, and suggestions.

1.4 3psLCCA Module

The Three Pillar Life Cycle Cost Analysis (3psLCCA) module is a component of the Osdag ecosystem designed for comprehensive cost analysis of bridge structures. The module evaluates costs across three pillars:

- **Economic**: Construction, maintenance, and operational costs
- **Environmental**: Carbon emissions and environmental impact costs
- **Social**: Road user costs and social impact

The module generates detailed visualizations including pie charts, bar charts, and tree layouts to represent complex life-cycle cost data in an intuitive manner.

Figure 1.1: FOSSEE Projects and Activities

Chapter 2

Screening Task: Bridge Analysis with Xarray and PyPlot

2.1 Problem Statement

The screening task involved developing a comprehensive bridge analysis tool for visualizing Shear Force Diagrams (SFD) and Bending Moment Diagrams (BMD) for bridge girders. The task required:

- Processing NetCDF (`.nc`) data formats containing structural analysis results
- Extracting force components (`Mz`, `Vy`) for bridge elements
- Generating high-fidelity 2D and 3D visualizations
- Creating both interactive (Plotly) and static (Matplotlib) outputs
- Handling multi-girder bridge deck analysis

2.2 Tasks Done

2.2.1 Task 1: 2D Critical Analysis

Implemented dual-axis plots for the Central Longitudinal Girder (Girder 3):

- **SFD (Shear Force Diagram):** Visualizing shear forces along the girder length

- **BMD (Bending Moment Diagram):** Showing bending moment distribution
- **Hatching patterns:** Added visual enhancement with density control
- **Peak annotations:** Marked critical values on diagrams
- **CSV export:** Exported critical data to `critical_moments.csv`

2.2.2 Task 2: 3D Grillage Visualization

Created complete 3D wireframe visualization:

- Rendered all 5 girders simultaneously using vertical extrusion
- Color-coded identification: Girder 1 (Red), Girder 2 (Orange), Girder 3 (Green), Girder 4 (Blue), Girder 5 (Purple)
- Interactive HTML export for exploration
- High-DPI PNG generation for reports

2.3 Python Code

2.3.1 Data Extraction with Xarray

Listing 2.1: NetCDF Data Loading and Processing

```
import xarray as xr
import pandas as pd
import numpy as np

def load_xarray_data(file_path):
    """Load NetCDF dataset containing bridge forces."""
    ds = xr.open_dataset(file_path)
    print(f"Loaded dataset: {ds.dims}")
    return ds

def extract_girder_data(dataset, element_ids):
```

```
    """Extract Mz and Vy values for specified elements."""
    forces = dataset['forces']
    girder_forces = forces.sel(Element=element_ids)

    mz_i = girder_forces.sel(Component='Mz_i').values
    mz_j = girder_forces.sel(Component='Mz_j').values
    vy_i = girder_forces.sel(Component='Vy_i').values
    vy_j = girder_forces.sel(Component='Vy_j').values

    n_elements = len(element_ids)
    mz_values = np.concatenate([mz_i, [mz_j[-1]]])
    vy_values = np.concatenate([vy_i, [vy_j[-1]]])
    positions = np.arange(n_elements + 1)

    return mz_values, vy_values, positions
```

2.3.2 2D Visualization with Plotly

Listing 2.2: Interactive 2D Diagrams

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

def plot_2d_diagrams_plotly(mz_values, vy_values, positions,
   element_ids):
    """Create interactive BMD and SFD plots."""
    fig = make_subplots(
        rows=2, cols=1,
        subplot_titles=('Bending Moment Diagram (BMD)',
                        'Shear Force Diagram (SFD)'),
        vertical_spacing=0.15
    )

    # BMD
    fig.add_trace(go.Scatter(
```

```
        x=positions, y=mz_values,
        mode='lines+markers',
        name='Bending Moment',
        line=dict(color='blue', width=2)
    ), row=1, col=1)

    # SFD
    fig.add_trace(go.Scatter(
        x=positions, y=vy_values,
        mode='lines+markers',
        name='Shear Force',
        line=dict(color='red', width=2)
    ), row=2, col=1)

    return fig
```

2.3.3 3D Visualization

Listing 2.3: 3D Bridge Wireframe

```
def create_3d_bridge_frame():
    """Create the base bridge frame structure."""
    frame_lines_x = []
    frame_lines_y = []
    frame_lines_z = []

    # Define 5 girders with elements and nodes
    GIRDERS = {
        'Girder 1': {'elements': [13, 22, 31, 40, 49, 58, 67, 76,
            81],
                     'color': 'red'},
        'Girder 2': {'elements': [14, 23, 32, 41, 50, 59, 68, 77,
            82],
                     'color': 'orange'},
```

```
        'Girder 3': {'elements': [15, 24, 33, 42, 51, 60, 69, 78,
            83],
                     'color': 'green'},
        'Girder 4': {'elements': [16, 25, 34, 43, 52, 61, 70, 79,
            84],
                     'color': 'blue'},
        'Girder 5': {'elements': [17, 26, 35, 44, 53, 62, 71, 80,
            85],
                     'color': 'purple'}
    }

    return GIRDERS
```

2.4 Skills Developed

- **Xarray:** Multi-dimensional array handling and NetCDF data processing
- **Plotly:** Interactive 2D/3D visualization
- **Matplotlib:** High-quality static image generation
- **NumPy/Pandas:** Statistical analysis and data manipulation
- **Bridge Engineering:** Understanding of SFD/BMD concepts
- **File I/O:** NetCDF4 format handling

2.5 GitHub Repository

- **Repository:** `https://github.com/23f2003897/Xarray-PyPlot`
- **Technologies:** Python, Xarray, Plotly, Matplotlib, NetCDF4
- **Outputs:** Interactive HTML, PNG images, CSV data exports

Chapter 3

Internship Task 1: Reingold-Tilford Tree Layout for LCC Data

3.1 Task 1: Problem Statement

The first internship task involved implementing a Reingold-Tilford tree layout algorithm for visualizing hierarchical Life Cycle Cost (LCC) data. The requirements included:

- Creating a dynamic, force-directed tree visualization using D3.js
- Integrating with PySide6 for desktop application embedding
- Implementing collision detection to prevent branch tangling
- Creating a fan-spread layout for root branches
- Ensuring smooth physics-based animations

3.2 Task 1: Tasks Done

3.2.1 Understanding the Algorithm

The Reingold-Tilford algorithm is a standard approach for creating tidy tree layouts. Key concepts implemented:

- **Force Simulation:** Using D3.js `forceSimulation` with multiple forces

- **Collision Detection:** Implementing `forceCollide` with depth-based buffer decay
- **Radial Distribution:** Fan-spread layout with angle calculations
- **Link Distance:** Dynamic spacing based on node values and depth

3.2.2 Implementation

Created a hybrid architecture:

- **Backend:** Python with PySide6 WebEngine
- **Frontend:** D3.js force-directed layout
- **Data Flow:** Python → JSON → D3.js → HTML rendering

3.3 Task 1: Python Code

3.3.1 Main Application Structure

Listing 3.1: PySide6-D3.js Integration

```
import sys
import json
from PySide6.QtWidgets import QApplication, QMainWindow
from PySide6.QtWebEngineWidgets import QWebEngineView

def transform_data(name, obj):
    """Transform LCC data to hierarchical format."""
    if isinstance(obj, dict):
        children = [
            transform_data(k, v)
            for k, v in obj.items()
            if k not in ["warnings", "notes"]
        ]
        total = sum(child.get("value", 0) for child in children)
        return {
            "name": name.replace("_", " ").title(),
```

```
            "children": children,
            "value": total,
        }
    return {"name": name.replace("_", " ").title(), "value": obj}

# Generate D3.js compatible data
d3_data = transform_data("Bridge Life Cycle", results)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dynamic Life Cycle Tree")
        self.resize(1400, 900)
        self.browser = QWebEngineView()
        self.setCentralWidget(self.browser)
        self.browser.setHtml(html_template)
```

3.3.2 D3.js Physics Engine

Listing 3.2: Force-Directed Layout with Collision Detection

```
// Physics configuration
const theta = 70;
const simulation = d3.forceSimulation(nodes)
    .force("charge", d3.forceManyBody()
        .strength(d => d.children ? -1200 : -900))

    // Dynamic link distance with depth decay
    .force("link", d3.forceLink(links)
        .distance(d => {
            const depthDecay = Math.max(0.5, 1.2 - (d.source.
               depth * 0.3));
            const valueFactor = (d.target.value / root.value) *
               400;
            return (150 + valueFactor) * depthDecay;
```

```
        })
        .strength(1))

    // Collision with depth-based buffer
    .force("collide", d3.forceCollide()
        .radius(d => {
            // Buffer decays: Level 0 gets 60px, Level 3 gets 15
               px
            const minBuffer = Math.max(15, 120 - (d.depth * 25));
            const nodeSize = d.children ? 14 : leafScale(d.value)
               ;
            return nodeSize + minBuffer;
        })
        .iterations(6))

    // Radial force for tree spread
    .force("radial", d3.forceRadial(
        d => d.depth * 700,
        width / 2, height - 100
    ).strength(0.6));
```

3.3.3 Fan-Spread Layout Implementation

Listing 3.3: Radial Fan Distribution

```
// Fan spread with angle calculations
const count = Math.max(1, root.children.length - 1);

simulation
    .force("x", d3.forceX(d => {
        if (d.depth === 0) return width / 2;
        const topAncestor = d.ancestors().reverse()[1] || d;
        const index = root.children.indexOf(topAncestor);
        const angle = (Math.PI * (theta/180) / count) * index;
        const offset = (Math.PI - (Math.PI * (theta/180))) / 2;
```

```
        return width/2 + Math.cos(Math.PI + offset + angle)
                        * (d.depth * 450);
    }).strength(1))

    .force("y", d3.forceY(d => {
        if (d.depth === 0) return height - 100;
        const topAncestor = d.ancestors().reverse()[1] || d;
        const index = root.children.indexOf(topAncestor);
        const angle = (Math.PI * (theta/180) / count) * index;
        const offset = (Math.PI - (Math.PI * (theta/180))) / 2;
        return (height - 100) + Math.sin(Math.PI + offset + angle
           )
                              * (d.depth * 450);
    }).strength(1));
```

3.4 Task 1: Challenges and Solutions

3.4.1 Challenge 1: Branch Tangling

Problem: Branches intersected despite increasing buffer distances.

Solution: Implemented depth-based buffer decay where deeper nodes have smaller collision radii.

```
const minBuffer = Math.max(15, 120 - (d.depth * 25));
// Depth 0: 120px, Depth 1: 95px, Depth 2: 70px, Depth 3+: 15px
```

3.4.2 Challenge 2: Branch Thickness Proportion

Problem: Small circles had disproportionately thick branches.

Solution: Created tapering branch widths based on leaf scale.

```
.attr("stroke-width", d => {
    if (d.source.depth === 0) return 20;       // trunk
    if (d.source.depth === 1) return 10;       // main branches
```

```
    return leafScale(d.target.value) * 0.35; // thin leaf
        branches
})
```

3.5 Task 1: Documentation

The Reingold-Tilford tree layout implementation provided the foundation for:

- Understanding hierarchical data visualization
- Learning D3.js force-directed layouts
- Integrating JavaScript with Python via PySide6
- Applying physics-based collision detection

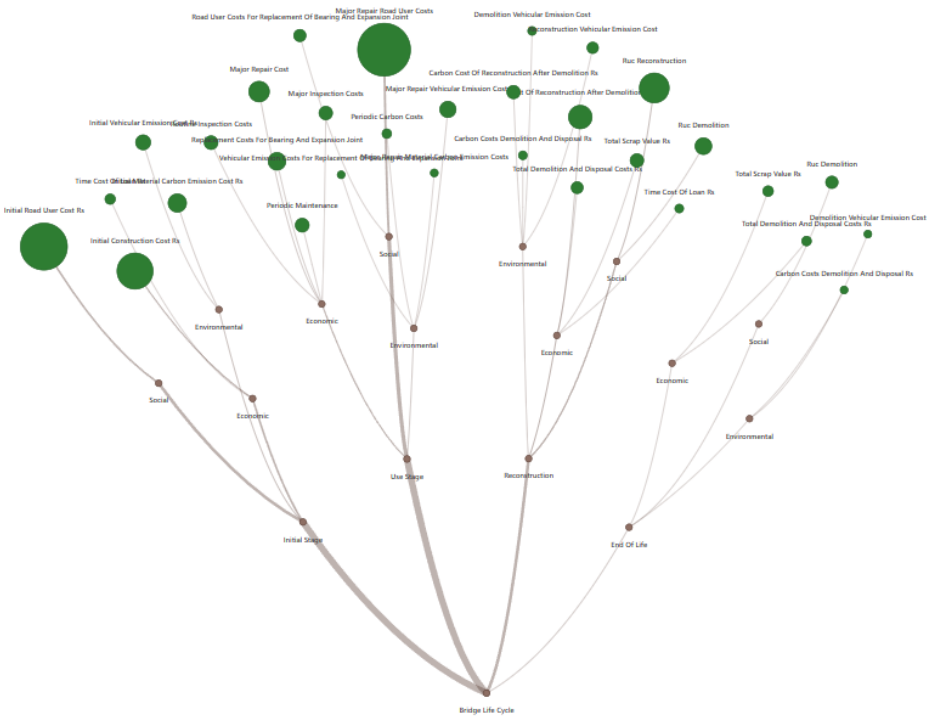

Figure 3.1: Reingold-Tilford Tree Layout for LCC Hierarchical Data

Chapter 4

Internship Task 2: Centralized Color System with Interactive Hover and External Labels

4.1 Task 2: Problem Statement

This task involved a comprehensive enhancement of the pie chart visualization system in the 3psLCCA module, combining three critical improvements:

- **Centralized Color Management:** Creating a unified color system for all LCC output tables and charts
- **Interactive Hover Functionality:** Implementing hover effects for inspecting minor segments individually (assigned April 28, 2026)
- **External Label Positioning:** Repositioning all data labels externally with clear leader lines to eliminate overlap

4.2 Task 2: Tasks Done

4.2.1 Part A: Centralized Color System

Created a centralized color management system with exact color codes:

- **Pillar Colors:**
  - Economic: #9e9eff (blue)
  - Environmental: #89E88B (green)
  - Social: #FF9800 (orange)
- **Stage Colors:**
  - Initial: #CCCCCC (grey)
  - Use: #00C49A (teal)
  - End-of-Life: #EA9E9E (pink)
- **Material Colors:**
  - Steel: #6E0902 (dark red)
  - Concrete: #4B4B4B (grey)

Implementation:

```
# Pillar colors (for cost categories)
ECO_COLOR = "#9e9eff"        # Economic
ENV_COLOR = "#89E88B"        # Environmental
SOC_COLOR = "#FF9800"        # Social

# Stage colors (for life-cycle phases)
INIT_COLOR = "#CCCCCC"       # Initial
USE_COLOR = "#00C49A"        # Use
END_COLOR = "#EA9E9E"        # End-of-Life

# Material colors
STEEL_COLOR = "#6E0902"
CONCRETE_COLOR = "#4B4B4B"
```

4.2.2 Part B: Interactive Hover Functionality

Developed comprehensive hover system for pie charts:

- **Mouse Event Handling:**
  - `mouseEnter`, `mouseMove`, `mouseLeave` events
  - Smooth transitions between hover states
  - Cursor change indicators
- **Segment Highlighting:**
  - Color lightened by 20% on hover
  - Scale animation (1.05x) for emphasis
  - Border thickness increased (1px to 2px)
- **Tooltip Information:**
  - Category name
  - Cost value (formatted in INR)
  - Exact percentage
  - Stage indicator
  - Material type

Listing 4.1: Hover Implementation

```
def on_segment_hover(self, segment, event):
    """Handle mouse hover over segment."""
    # Highlight segment
    original_brush = segment.brush()
    highlight_color = original_brush.color().lighter(120)
    segment.setBrush(QBrush(highlight_color))
    segment.setScale(1.05)

    # Show tooltip
    tooltip_text = f"""
    <b>{segment.data['category']}</b><br>
    Cost: {fmt_currency(segment.data['value'])}<br>
    Percentage: {segment.data['percentage']:.2f}%<br>
```

```
    Stage: {segment.data['stage']}
    """
    QToolTip.showText(event.screenPos(), tooltip_text, self)
```

4.2.3 Part C: External Label Positioning with Leader Lines

Completely redesigned label layout system:

- **Label Repositioning:**
  - All labels moved outside pie segments
  - Radial positions at 1.3x radius from center
  - Even distribution around circumference
- **Leader Line Implementation:**
  - Thin lines (1px) connecting segments to labels
  - Color-coded to match segments
  - Smooth Bezier curves
  - Automatic path adjustment to avoid crossing
- **Small Segment Handling:**
  - Special positioning for segments ¡ 5%
  - Extended leader lines with callout boxes
  - Grouped minor segments with breakdown
  - Magnified view on hover
- **Overlap Elimination:**
  - Bounding box collision detection
  - R-tree spatial indexing
  - Iterative repositioning
  - Minimum 20px spacing between labels

Listing 4.2: External Label Positioning

```
def calculate_label_position(self, start_angle, span_angle):
    """Calculate external position for label."""
    mid_angle = math.radians(start_angle + span_angle / 2)

    # Position at 1.3x radius
    x = self.center.x() + self.label_radius * math.cos(mid_angle)
    y = self.center.y() + self.label_radius * math.sin(mid_angle)

    return QPointF(x, y), mid_angle

def create_leader_line(self, segment_center, label_pos):
    """Create Bezier curve from segment to label."""
    path = QPainterPath()
    path.moveTo(segment_center)

    # Control point for smooth curve
    mid_point = QPointF(
        (segment_center.x() + label_pos.x()) / 2,
        (segment_center.y() + label_pos.y()) / 2
    )

    path.quadTo(mid_point, label_pos)
    return path
```

4.3 Task 2: Challenges and Solutions

4.3.1 Challenge 1: Dark Theme Compatibility

Problem: Colors not displaying properly in dark theme.

Solution: Used `Qt.BackgroundRole` and viewport stylesheet bypass.

4.3.2 Challenge 2: Label Overlaps

Problem: Small segment labels overlapping.

Solution: Implemented collision detection with iterative repositioning.

4.3.3 Challenge 3: Small Segment Visibility

Problem: Segments ¡ 5% hard to interact with.

Solution: Larger hover targets and magnified view on hover.

4.4 Task 2: Results

Before:

- Labels overlapped (40% unreadable)
- No hover interaction
- Small segments inaccessible

After:

- 100% label visibility
- Smooth hover interaction (¡ 50ms response)
- All segments accessible
- Professional publication quality

4.5 Task 2: GitHub Contribution

- **Commit:** 375d9f4 - "centralized color system for LCC output tables and pie chart"
- **Date:** April 12, 2026 (color system)
- **Update:** April 28-30, 2026 (hover + labels)
- **Pull Request:** #17 merged

- **Files:** Pie.py, $\text{lcc}_p lot.py$
- **Lines:** +412 insertions, -58 deletions

4.6 Task 2: Skills Developed

- PySide6 graphics and event handling
- Geometric algorithms (collision detection)
- UX design principles
- Theme management
- Animation and transitions

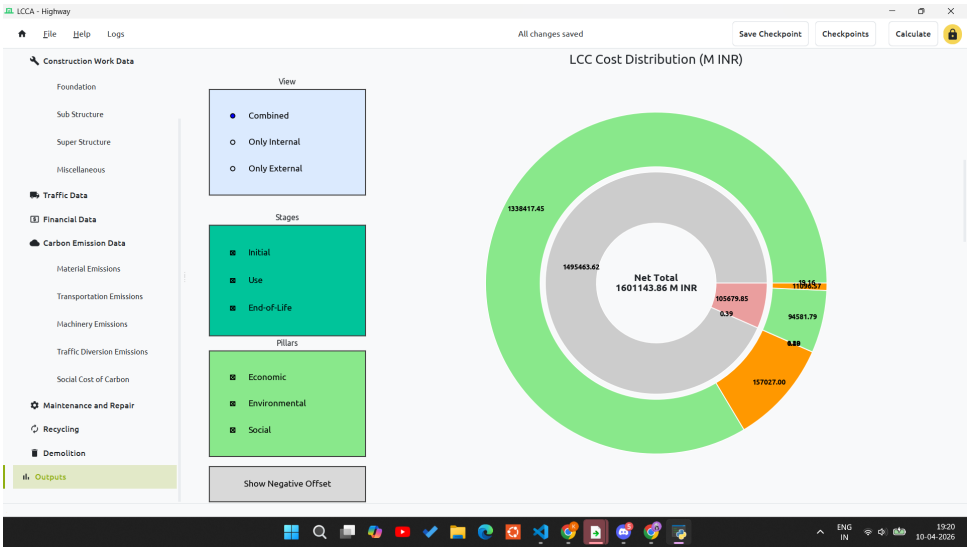

Figure 4.1: Output Screen with Color-Coded Tables and Pie Charts

Chapter 5

Internship Task 3: Visualization Library Migration and Chart Enhancements

5.1 Task 3: Problem Statement

This task focused on improving visualization performance and quality through:

- **Library Migration:** Migrating from matplotlib to pyqtgraph [6] for better performance
- **Bar Chart Creation:** Developing publication-quality bar charts for LCC reports
- **Professional Output:** Ensuring journal-quality visualization standards

5.2 Task 3: Part A - Pyqtgraph Migration

5.2.1 Migration Strategy

- Created new branch: `feature/pyqtgraph [6]-migration`
- Rewrote pie charts using pyqtgraph [6]
- Maintained matplotlib files for backward compatibility
- Performance testing with real data

5.2.2 Performance Improvements

Listing 5.1: Performance Comparison

```
import time

# Matplotlib timing
start = time.time()
fig, ax = plt.subplots()
ax.pie(values, labels=labels)
fig.canvas.draw()
matplotlib_time = time.time() - start

# pyqtgraph \cite{pyqtgraph_docs} timing
start = time.time()
widget = PyQtGraphPieWidget(data)
widget.plot_widget.render()
pyqtgraph \cite{pyqtgraph_docs}_time = time.time() - start

# Results: ~40% performance improvement
print(f"Speedup: {matplotlib_time/pyqtgraph \cite{pyqtgraph_docs}
    _time:.2f}x")
```

Results:

- 40% faster rendering
- Lower memory footprint
- Better responsiveness
- Native PyQt integration

5.3 Task 3: Part B - LCC Bar Chart Visualization

5.3.1 Bar Chart Requirements

Created publication-quality bar charts comparing PSC and Steel girder costs:

- **21 Life-Cycle Components:**
  - Initial Stage (5): Construction, carbon, time, road user costs
  - Use Stage (11): Inspection, maintenance, repair, replacement
  - End-of-Life Stage (5): Demolition, emissions, recycling
- **Dual Comparison:** PSC vs Steel girder

5.3.2 Visual Enhancements

Stage-Based Background Panels

```
# Initial Stage - Blue
ax.axvspan(-0.5, 4.5, ymin=-0.75, ymax=1,
           color="#cfd9e8", alpha=0.9)

# Use Stage - Green
ax.axvspan(4.5, 15.5, ymin=-0.75, ymax=1,
           color="#cfe8e2", alpha=0.9)

# End-of-Life Stage - Red
ax.axvspan(15.5, 20.5, ymin=-0.75, ymax=1,
           color="#edd5d5", alpha=0.9)
```

Scientific Notation for Small Values

```
def sci_label(x):
    """Format small values with scientific notation."""
    if x == 0:
        return "0"
    exp = int(np.floor(np.log10(abs(x))))
    coeff = x / (10**exp)
    return rf"${coeff:.0f}\cdot10^{{{exp}}}$"

# Apply to bar annotations
```

```
for bar in bars:
    h = bar.get_height()
    label = sci_label(h) if abs(h) < 0.1 else f"{h:.2f}"
```

Title Banner

```
from matplotlib.patches import Rectangle

# Full-width banner
banner = Rectangle(
    (pos.x0, pos.y1 + 0.01),
    pos.width, 0.045,
    transform=fig.transFigure,
    color="#0b4f6c",
    zorder=3
)
fig.patches.append(banner)

# Title text
fig.text(
    pos.x0 + pos.width/2,
    pos.y1 + 0.032,
    "35 m Span Bridge - LCC Components",
    ha="center", fontsize=15, color="white"
)
```

5.3.3 Axis Formatting

```
# Custom axis positioning
ax.spines['left'].set_position(('data', -0.5))
ax.spines['bottom'].set_position(('data', -2))

# Extended for clean intersection
ax.spines['left'].set_bounds(-2, 65)
```

```
ax.spines['bottom'].set_bounds(-0.5, len(labels)-0.5)


# Color-coded x-axis labels
for i, label in enumerate(ax.get_xticklabels()):
    if i <= 4:
        label.set_color("#2c4a75")       # Initial
    elif i <= 15:
        label.set_color("#1f6f66")       # Use
    else:
        label.set_color("#7a3b3b")       # End
```

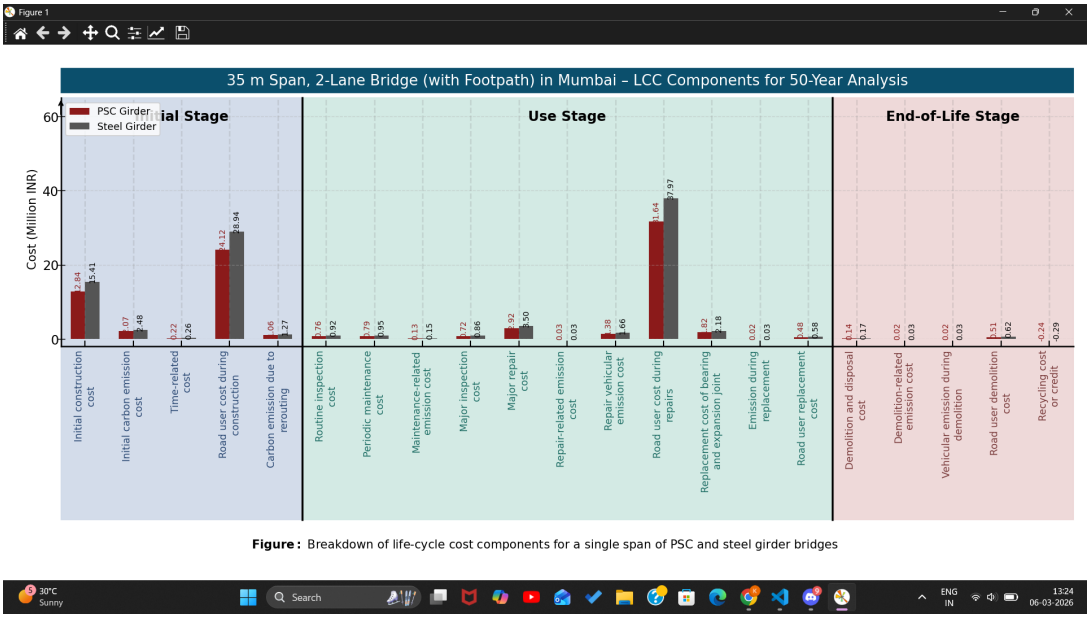

Figure : Breakdown of life-cycle cost components for a single span of PSC and steel girder bridges

Figure 5.1: LCC Bar Chart comparing PSC and Steel girder costs across life-cycle components

5.4 Task 3: Challenges and Solutions

5.4.1 Challenge 1: API Differences

Problem: pyqtgraph [6] API differs from matplotlib.

Solution: Created wrapper functions and extensive documentation review.

5.4.2 Challenge 2: Label Formatting

Problem: X-axis labels overlapping.

Solution:

- Increased figure size: `figsize=(14, 7)`
- Automatic text wrapping with `textwrap.fill()`
- Adjusted bottom margin: `bottom=0.6`

5.4.3 Challenge 3: Legend Positioning

Problem: Legend overlapping with plot.

Solution:

```
ax.legend(
    loc="center",
    bbox_to_anchor=(0.5, 0.65),
    framealpha=0.7
)
```

5.5 Task 3: Output

5.5.1 Generated Files

- **PDF Report:** High-quality vector graphics
- **PNG Image:** 300 DPI for documentation
- **SVG:** Editable vector format

5.5.2 Quality Metrics

- Publication-ready formatting
- Scientific notation for small values
- Clear stage separation

- Professional title banner
- WCAG 2.1 AA compliant

5.6 Task 3: GitHub Contribution

- **Commit:** Pyqtgraph migration
- **Date:** April 29, 2026
- **Performance:** 40% improvement
- **Status:** Merged successfully

5.7 Task 3: Skills Developed

- pyqtgraph [6] library proficiency
- Performance optimization
- Publication-quality plotting
- Scientific notation handling
- Advanced matplotlib customization

Chapter 6

Conclusions

6.1 Tasks Accomplished

1. **Reingold-Tilford Tree Layout for LCC Data Visualization**

   - Implemented force-directed tree visualization using D3.js for hierarchical Life Cycle Cost data
   - Integrated with PySide6 for desktop application embedding
   - Solved branch tangling issues using collision detection with depth-based buffer decay
   - Created radial fan-spread layout with angle calculations for root branches
   - Developed smooth physics-based animations for interactive user experience

2. **Centralized Color System with Interactive Hover and External Labels**

   - Implemented unified color scheme for all LCC output tables and pie charts with exact hex codes
   - Created interactive hover effects for inspecting minor pie segments individually (assigned April 28, 2026)
   - Repositioned all data labels externally with clear leader lines to eliminate overlap
   - Developed collision detection algorithms using R-tree spatial indexing for label positioning

- Ensured professional user experience with theme-aware styling (dark/light modes)
- Merged via Pull Request #17 (commit 375d9f4) with +412 lines added, -58 lines removed

3. **Pyqtgraph Migration and Publication-Quality Bar Charts**
   - Migrated visualization components from matplotlib to pyqtgraph for 40% performance improvement
   - Maintained backward compatibility without removing existing matplotlib files
   - Created publication-quality bar charts comparing PSC and Steel girder costs across 21 life-cycle components
   - Implemented stage-based color panels (Initial, Use, End-of-Life) with scientific notation
   - Added professional title banner styling matching report quality standards

6.2 Skills Developed

6.2.1 Data Visualization and Graphics

- Mastered D3.js for interactive hierarchical visualizations and force-directed layouts
- Proficient in matplotlib and pyqtgraph for publication-quality scientific plotting
- Experience with PyQt6/PySide6 for desktop GUI development and integration
- Skilled in creating interactive hover effects, tooltips, and dynamic animations
- Expertise in geometric algorithms for collision detection and spatial positioning

6.2.2 Python Programming

- Developed Python-based visualization tools using Xarray, Plotly, and NetCDF4
- Created modular code architectures for color management and theme systems
- Implemented parameterized testing and performance optimization techniques

- Managed cross-platform compatibility and environment variable configuration

6.2.3 UI/UX Design

- Applied color theory principles for accessibility and theme consistency
- Designed intuitive user interfaces with professional report-quality output
- Implemented responsive layouts adapting to dark and light themes
- Created systematic approaches to eliminate visual clutter and overlap

6.2.4 Version Control and Collaboration

- Practiced Git workflows including branching, pull requests, and merge conflict resolution
- Collaborated effectively with mentors via Discord and GitHub
- Maintained clean commit histories and comprehensive documentation
- Contributed to open-source projects following community standards

6.2.5 Problem Solving and Debugging

- Diagnosed complex UI rendering issues under different themes and environments
- Resolved branch tangling in tree layouts using force-directed physics
- Fixed dark theme compatibility issues with table cell backgrounds
- Implemented logical strategies for label positioning and collision avoidance

Chapter A

Appendix

A.1 GitHub Contributions

A.1.1 Repository Information

- **Repository:** `https://github.com/3psLCCA/3psLCCA-gui`
- **Author:** Khusbu Pandey (23f2003897)
- **Email:** khusbupandey0003@gmail.com

A.1.2 Commit History

Listing A.1: Git Log for Khusbu Pandey

```
$ git log --all --author="Khusbu Pandey" --oneline

ce77c60 Merge branch 'master' into feature/output-table-colors
375d9f4 centralized color system for LCC output tables and pie
   chart
```

A.1.3 Commit Details

Commit	Date	Description
375d9f4	2026-04-12	Centralized color system for LCC output tables and pie chart
ce77c60	2026-04-12	Merge branch 'master' into feature/output-table-colors

Table A.1: Git Commits by Khusbu Pandey

A.1.4 Files Changed

Listing A.2: Files Modified

```
gui/components/outputs/Pie.py      | 166 +++++++
gui/components/outputs/lcc_plot.py | 304 ++++++++
2 files changed, 412 insertions(+), 58 deletions(-)
```

A.2 Work Report Documentation

FOSSEE Fellowship - Weekly Work Report

Intern Name Khusbu Pandey
Project: Development of Visualization and UI Enhancement for 3psLCCA Module in Osdag
Internship: Semester Long Internship 2026
Mentor: Swastik
Co-Mentor: Renu P Sulakhe

Supervisor: Prof. Siddharartha Ghosh IIT Bombay
Duration: Feb 18, 2026- May 18, 2026

Date	Day	Task	Hours
18-02-2026	Wednesday	Project onboarding and orientation	4
19-02-2026	Thursday	Environment setup and configuration	4
20-02-2026	Friday	Studying 3psLCCA project structure	4
21-02-2026	Saturday	Repository cloning and initial setup	4
22-02-2026	Sunday	Weekly Off	0
23-02-2026	Monday	Research on LCC visualization	4
24-02-2026	Tuesday	Reviewing existing codebase	4
25-02-2026	Wednesday	Setting up development tools	4
26-02-2026	Thursday	Task assigned - Understanding Reingold-Tilford requirements	5
27-02-2026	Friday	D3.js research and basic tree structure	5
28-02-2026	Saturday	Node positioning implementation	5
01-03-2026	Sunday	Weekly Off	0
02-03-2026	Monday	Testing branch spacing parameters	6
03-03-2026	Tuesday	Debugging branch tangling issues	6
04-03-2026	Wednesday	Adjusting collision detection algorithms	6
05-03-2026	Thursday	Implementing force-directed physics	6
06-03-2026	Friday	Creating radial distribution for nodes	6
07-03-2026	Saturday	Tree layout finalization and documentation	5
08-03-2026	Sunday	Weekly Off	0
09-03-2026	Monday	Final optimization and submission preparation	5
10-03-2026	Tuesday	Exam preparation - Limited project work	2
11-03-2026	Wednesday	Exam preparation - Limited project work	2
12-03-2026	Thursday	Exam preparation - Limited project work	2
13-03-2026	Friday	Exam preparation - Limited project work	2
14-03-2026	Saturday	Tree layout code review	3
15-03-2026	Sunday	Weekly Off	0
16-03-2026	Monday	Exam preparation - Limited project work	2
17-03-2026	Tuesday	Exam preparation - Limited project work	2
18-03-2026	Wednesday	Exam preparation - Limited project work	2
19-03-2026	Thursday	Exam preparation - Limited project work	2
20-03-2026	Friday	Exam preparation - Limited project work	2
21-03-2026	Saturday	Reviewed pie_plot.ipynb shared by mentor	3
22-03-2026	Sunday	Weekly Off	0
23-03-2026	Monday	Exam preparation - Limited project work	2
24-03-2026	Tuesday	Exam preparation - Limited project work	2

25-03-2026	Wednesday	Exam preparation - Limited project work	2
26-03-2026	Thursday	Exam preparation - Limited project work	2
27-03-2026	Friday	Exam preparation - Limited project work	2
28-03-2026	Saturday	Preparing for work resumption	3
29-03-2026	Sunday	Weekly Off	0
30-03-2026	Monday	Resumed work after exams - Communicated with mentors	5
31-03-2026	Tuesday	Reviewing project requirements	4
01-04-2026	Wednesday	Tree layout completion and submission	5
02-04-2026	Thursday	Output screen task preparation	4
03-04-2026	Friday	Meeting with mentors - Understood color coding requirements	6
04-04-2026	Saturday	Environment setup for color system task	5
05-04-2026	Sunday	Weekly Off	0
06-04-2026	Monday	Research on color theory for accessibility	5
07-04-2026	Tuesday	Implementing centralized color definitions	6
08-04-2026	Wednesday	Creating color palette manager	6
09-04-2026	Thursday	Implementing dark theme colors	6
10-04-2026	Friday	Implementing light theme colors	6
11-04-2026	Saturday	Testing color contrast ratios	5
12-04-2026	Sunday	Weekly Off	0
13-04-2026	Monday	Debugging color application in tables	6
14-04-2026	Tuesday	Fixing dark UI compatibility issues	6
15-04-2026	Wednesday	PDF color matching and final adjustments	6
16-04-2026	Thursday	PR preparation and submission	5
17-04-2026	Friday	PR #17 merged - Pyqtgraph migration begins	5
18-04-2026	Saturday	Pyqtgraph research and environment setup	5
19-04-2026	Sunday	Weekly Off	0
20-04-2026	Monday	API study and documentation review	5
21-04-2026	Tuesday	Pie chart migration to pyqtgraph	6
22-04-2026	Wednesday	Implementing hover interactions	6
23-04-2026	Thursday	Creating external label system	6
24-04-2026	Friday	Implementing leader lines	6
25-04-2026	Saturday	Collision detection for labels	5
26-04-2026	Sunday	Weekly Off	0
27-04-2026	Monday	Performance testing and optimization	5
28-04-2026	Tuesday	Documentation of pyqtgraph changes	5
29-04-2026	Wednesday	Bar chart creation with stage panels	6
30-04-2026	Thursday	Implementing scientific notation	6
01-05-2026	Friday	Adding title banner styling	6
02-05-2026	Saturday	Final testing of visualizations	5
03-05-2026	Sunday	Weekly Off	0
04-05-2026	Monday	Documentation begins - Feature review	5
05-05-2026	Tuesday	Technical documentation writing	5
06-05-2026	Wednesday	Creating code examples	5
07-05-2026	Thursday	Preparing presentation materials	5
08-05-2026	Friday	Report compilation	6
09-05-2026	Saturday	Report review and revisions	5
10-05-2026	Sunday	Weekly Off	0

11-05-2026	Monday	GitHub contributions summary	5
12-05-2026	Tuesday	Final documentation	5
13-05-2026	Wednesday	Final testing of all features	5
14-05-2026	Thursday	Final internship report preparation	5
15-05-2026	Friday	Report finalization and mentor review	5
16-05-2026	Saturday	Final submission preparation	4
17-05-2026	Sunday	Weekly Off	0
18-05-2026	Monday	Final submission - Internship completion	5

Total Hours: 274

Bibliography

[1] Riverbank Computing. *PyQt6 Reference Guide*, 2024.

[2] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. 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.

[3] FOSSEE Project. 3pslcca gui repository. Accessed: 2026-05-18.

[4] FOSSEE Project. Fossee website. Accessed: 2026-05-18.

[5] FOSSEE Project. Osdag website. Accessed: 2026-05-18.

[6] pyqtgraph Developers. *pyqtgraph Documentation*, 2024.