A Web-Based Tool for Cross Dock Trailer Scheduling

Final Report

by

William G. Ferrell, Jr.
262 Freeman Hall
864 656 2724
fwillia@clemson.edu
Clemson University

Nathan Huynh, University of Nebraska-Lincoln
Alison Gaddy, Clemson University

December 2024

Center for Connected Multimodal Mobility (C^2M^2)

CLEMSON UNIVERSITY

SC State University · The Citadel, The Military College of South Carolina · University of South Carolina · Benedict College

200 Lowry Hall
Clemson, SC 29634

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by the Center for Connected Multimodal Mobility (C^2M^2) (Tier 1 University Transportation Center) Grant, which is headquartered at Clemson University, Clemson, South Carolina, USA, from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Non-exclusive rights are retained by the U.S. DOT.

ACKNOWLEDGMENT

This study is based on a study supported by the Center for Connected Multimodal Mobility (C2M2) (USDOT Tier 1 University Transportation Center) Grant headquartered at Clemson University, Clemson, South Carolina, USA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Center for Connected Multimodal Mobility (C2M2), and the U.S. Government assumes no liability for the contents or use thereof.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle	5. Report Date
A Web-Based Tool for Cross Dock Trailer Scheduling	September 30, 2024
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
William G. Ferrell, Jr., ORCID: 0000-0002-6578-522X Nathan Huynh, ORCID: 0000-0002-4605-5651 Alison Gaddy	

9. Performing Organization Name and Address	10. Work Unit No.
Department of Industrial Engineering, Clemson University, Box 340920, Freeman Hall, Clemson, SC 29634-0920	11. Contract or Grant No.

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Center for Connected Multimodal Mobility (C2M2) Clemson University 200 Lowry Hall, Clemson Clemson, SC 29634	Final Report (September 2024)
	14. Sponsoring Agency Code DOT

15. Supplementary Notes

16. Abstract

Cross docks are transshipment facilities designed to consolidate freight between shippers and customers. Freight arrives at cross-docks from geographically dispersed suppliers and is consolidated to destinations (e.g., customers) thereby increasing trailer utilization by avoiding having each supplier send orders to each customer individually, often are partially filled trailers. A well-run cross-dock has high throughput, so scheduling arrival and departure times of trucks is critical to effective and efficient operations.

A previous C2M2 project focused on this problem using a mixed integer programming model that is well-known to be non-deterministic polynomial-time hard. A new solution approach based on simulation annealing was proposed that found near-optimal solutions. This project animate the results of this research so that anyone to view the it even if they do not have the skills to understand the underlying research.

This was accomplished with a web-based animation tool that is available on the C2M2 website. There are a number of user-specified inputs that define the scenario which is animated. Times were scaled so that the animation runs in a few minutes rather than the multiple hours that would occur at a real cross dock; however, since parameters of the model and animation are consistent with practice, the sequence reflects the solution determined by the near-optimal solution of the mixed integer programming model and all timing of trucks (arrival and departures) are also the same as the near-optimal solution in a relative sense (i.e., they are just scaled to be shorter).

17. Keywords	18. Distribution Statement
Cross docks, scheduling	No restrictions.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified		NA

Table of Contents

List of Figures

EXECUTIVE SUMMARY

During this project, an animation was developed that visually displays the results of previous C2M2 research on cross docks. Users specify inputs (e.g., number of inbound and outbound trucks, number of dock doors) and watch the scenario evolve from the first inbound truck arrival to the last outbound truck departure. This visual tool allows users to assess important performance measures that are hard measure like congestion simply by watching the animation. Quantitative measures like makespan are also provided. By changing the inputs, the user can perform a visual sensitivity analysis to better understand the impact of design parameters on system performance. A link to the animation will be available on the C2M2 website.

This project was motivated by research that the investigators and their students have been exploring over the past several years regarding horizontal collaboration in a logistics network. One type of facility in current logistics networks that will also be integral in a next generation network based on horizontal collaboration is the cross dock. These transshipment facilities are designed to consolidate freight between shippers and customers. Freight arrives at cross-docks from geographically dispersed suppliers and is consolidated to destinations (e.g., customers) thereby increasing trailer utilization by avoiding having each supplier send orders to customers individually, often are partially filled trailers. Cross docking accomplishes this using four basic functions: 1) unload freight from inbound trailers, 2) sort freight by destination, 3) transfer the minimum amount of freight necessary to temporary storage, and 4) load freight to destinations on outbound trailers. A well-run cross-dock has high throughput, so any inventory is only held for a short time and utilization of all assets is high.

A previous C2M2 research focused on developing mathematical programming models to schedule the arrival times of inbound trucks and the departure times of outbound trucks to a cross dock. The objective was twofold: maximize the efficiency of cross dock operations and schedule outbound truck departures to minimize late deliveries to customers. The basic structure of this model prevents it from being solved to optimality for larger, more realistic sized problems so heuristics are justified. One was developed for this problem that used a novel construction heuristic to produce good-quality starting solutions for a population-based simulated annealing metaheuristic and this combination found good solutions with reasonable computational times.

This project animates the results obtained from running the model for an I-shaped cross dock with multiple dock doors is the basis for the model and animation. Products are interchangeable which means that demand for a product on an outbound trailer can be satisfied from an inbound trailer with that product type from any supplier. Hence, there is no one-to-one, *a priori* assignment of products from inbound trailers to outbound trailers.

The output of the math programming model solved by the metaheuristic provides the input that direct the animation. Users select key input parameters and can then watch inbound and outbound trucks arrive to and depart from the cross dock in the sequence and at the times determined to be near-optimal by the model. This allows users to gain intuition into the sensitivity of these inputs on cross dock performance.

The parameters used in the model reflect a real cross dock, so the time durations are too long for an animation (e.g., several hours). Hence, the times are scaled so that the animation will run in a few minutes, yet the arrivals and departures are sequenced correctly and their times at the cross dock are preserved in a relative sense. A counter is displayed that tracks the simulated time and the equivalent real time is shown as well.

CHAPTER 1

Introduction

1.1 Problem Overview

For the past several years, the investigators and their students have been exploring various aspects of horizontal collaboration in a logistics network. One research theme of a previous C2M2 projects was scheduling inbound and outbound trucks to a cross dock when the trucks and cross dock operator could communicate and chose to collaborate. This problem is quite important because the volume of freight flow is increasing rather dramatically in the United States and increased use of horizontal collaboration appears to be one of the promising strategies to effectively cope with stresses this increase is inducing. Cross docks are currently found in many logistics networks and their function is consistent with facilities that support horizontal collaboration. So, it seems logical that the number of cross docks will increase in these future systems because they are familiar, and they are effective. This, in turn, places an even greater importance on ensuring their operation is efficient and effective.

For context, cross-docks are transshipment facilities that are designed to consolidate freight between shippers and customers. Freight arrives at cross-docks from geographically dispersed suppliers and is consolidated to destinations (e.g., customers) thereby increasing trailer utilization by avoiding having each supplier send orders to each customer individually, often in partially filled trailers. This is accomplished using four basic functions: 1) unload freight from suppliers arriving on inbound trailers, 2) sort freight by destination, 3) hold only the minimum amount of freight necessary in temporary storage, and 4) load freight to destination/customers on outbound trailers. Freight moves through a well-run cross-dock at a high velocity so there is little inventory and any that is held only resides for a short time. Regardless, the advantage afforded by freight consolidation has a price because there is additional material handling associated with unloading freight, moving it across the cross-dock, and loading it on outbound trailers when compared with direct deliveries from origins to destination. On the other hand, cross-docks have much less storage and retrieval compared with traditional warehouses (W. Yu & Egbelu, 2008). Hence, Bartholdi and Gue (2004) concluded that cross-docking is economically viable if the inventory and transportation savings exceed the additional material handling costs. Make-span for cross-docks are directly related to the storage time of products. Longer product storage time results in higher logistic costs and delivery delays (Nogueira, Coutinho, Ribeiro, & Ravetti, 2020). Therefore, it is important to minimize the cross-dock makespan. There are a number of factors that influence the performance of a cross-dock, and all must be addressed effectively to achieve the necessary efficiency. These include the shape of the cross-dock, the schedule for inbound and outbound trailers, the resource management plan (e.g. labor, equipment), the plan for best use of the area near the cross-dock (i.e., yard optimization), and the schedule for transferring freight from inbound to outbound trailers, to name a few. Many of these factors have interactions. For example, the cross-dock shape determines docking capacity as well as transshipment paths within the facility. Effective synchronization between the docking and leaving times of inbound and outbound trailers have a tremendous impact on the speed of transshipments and the amount of inventory that must be held (Van Belle, Valckenaers, & Cattrysse, 2012). This lack of synchronization can lead to delays in outbound trailer departures, to lack of dock doors availability for arriving trailers, and to the need for excessive storage, all of which are unacceptable.

Project focus

This project animates a cross dock to present the results from the previous research so that it can be understood by a wide range of users. The inbound and outbound trucks schedules from the models in the previous research are used as inputs to the animation. The goal is for users to understand the results of the theoretical research without needing any background in technical aspects of the mathematical model or the heuristic that solves it. The web-based animation provides users with a simple dashboard from which they can adjust key parameters and watch in bound and outbound trucks execute the near-optimal solution. That is, the animation shows inbound and outbound trucks arriving, being docked, and departing in the sequence and at the times provided by the near-optimal solution that minimizes outbound delays while ensuring dock doors are available and storage is kept as small as possible. Users can build intuition into how a few inputs to a cross dock like the number of inbound and outbound trucks or the number of strip and sack doors impact key performance characteristics without any knowledge of the underlying models and algorithms. For example, a user can, for example, vary the number of strip doors for a scenario and note the impact on a traditional quantitative measure like makespan because the animation will report the duration between when the first inbound truck arrives until the last outbound truck departs. By watching the animation, they will also be able to gain intuition into qualitative measures like “congestion” by observing the truck arrivals and departures.

1.2 Underlying Research

The research that generates data driving the animation focused on synchronizing inbound and outbound trailer schedules in an I-shaped cross dock with multiple dock doors using a mixed integer programming model. The model includes inbound trailer arrival times that are non-uniform, characterizes the tardiness of outbound trailers using a nonlinear p/enalty function, and permits interchangeable products. The latter means that demand for a product on an outbound trailer can be satisfied from an inbound trailer with that product type from any supplier. That is, there is no one-to-one, a priori assignment of products from inbound trailers to outbound trailers.

This mixed integer programming model is well-known to be non-deterministic polynomial-time hard, so heuristics are justified for addressing larger, more realistic problems. In this research, new solution approach was developed that uses a construction heuristic to produce good-quality starting solutions for a population-based simulated annealing metaheuristic. The proposed hybrid metaheuristic leads to near-optimal solutions with a very good Relative Percentage Deviation which is one measure of a metaheuristic’s solution quality. Hence, the model was solved for a number of scenarios and the results are used as input to the animation.

Details of the research can be found in Badyal (2021); however, the MIP model, Mixed-doors Cross-dock Heuristic and the population-based simulated annealing are provided in Appendices A through C for the reader’s convenience.

1.3 Project Objectives

The objective of this project was to provide a visual representation of the research results previously described that can be viewed by users without the technical background to understand the underlying theoretical research. The vision was a web-based animation that would allow a user to select a few key input parameters and watch inbound and outbound truck arrive to and depart from the cross dock in the sequence and at the time determined by the near optimal solution found by the metaheuristic.

It was also important that the animation faithfully represent the near-optimal solution but run from start to finish in a few minutes. The was a concern because the parameters used in the model reflect a real cross dock, so the time durations from arrival of the first inbound truck to the departure of the last outbound truck is several hours. Hence, animation times were carefully scaled so that the arrivals and departures are sequenced consistent with the near-optimal solution and that the arrival and departure times are correct are consistent in a relative sense.

CHAPTER 2

Animation

2.1 Strategy

The primary goal of this project was to develop an interactive web-based animation tool for scheduling cross-dock trailer operations. This was aimed at demonstrating optimal scheduling results derived from a mixed-integer programming model, which aimed to minimize delays and ensure effective dock door management. The approach was to create a 3D animated representation of the optimal scheduling results, allowing the user to observe inbound and outbound truck movements in a visual format.

The animation needed to be user-friendly and informative, providing options for customizing parameters such as the number of inbound and outbound trucks, the number of stack and strip doors, and the ability to view the animation from different camera angles. The visualization needed to provide key features: trucks docking and leaving based on scheduled times, different viewpoints for visualization, and a clock counter to track the progress of the simulation. These tailor each animation to different cross-dock scenarios.

Initial steps for the project involved assessing the necessary items to build a website and determining the assets that could be used in the development. To build and deploy a website with 3D graphics, some of the crucial elements are a framework, a development environment and a 3D graphics library. More details are provided in this report.

2.2 Implementation

The animation of the cross-dock was developed in a structured manner, starting with a basic foundation and incrementally adding complexity.

The initial phase involved creating a *Django website framework* within *Docker containers*. This setup ensured a fast, portable, and reliable development environment. Django served as the backend framework, managing data flows and rendering the web page templates, while Docker provided consistency across deployments by isolating dependencies.

Django, running inside Docker containers, was used as the backend to serve HTML templates and manage the configuration settings for the 3D animation tool. Docker facilitated seamless collaboration and deployment, providing isolated containers for all necessary dependencies.

A 2D animated proof of concept was implemented in the development environment using basic HTML, CSS, and JavaScript. This helped to establish the visual representation and interactions before moving to a 3D environment. The initial proof of concept validated core features such as truck movement and user controls.

Building on the 2D concept, a 3D proof of concept was developed using the Three.js 3D graphics library. Three.js was chosen for its ability to create 3D graphics in a web browser without requiring extensive software installations. This library provided an effective way to load 3D models (using *OBJLoader* and *MTLLoader*), control camera views, and apply materials and textures to the models.

The core of the animation functionality relied on *Three.js*, which allowed for an effective transition from 2D to 3D, enhancing the user experience by adding depth and interactivity to the cross-dock simulation.

To bring the 3D environment to life, the inbound/outbound trucks, the distribution center, the labels, and the bay doors were modeled and textured in *Autodesk Maya*. This ensured that the 3D assets were visually accurate and optimized for use in a web-based animation.

A proof of concept for the procedural generation of the distribution center and its associated bay doors and labels was implemented using *Houdini FX*. Procedural generation allowed for flexibility in adjusting the configuration of the cross-dock, making it easy to change the number of doors or their positions as per the user's input. This feature added an element of scalability to the project, which was crucial for testing different configurations.

With the 3D models in place, the next step was implementing animation and truck scheduling logic on the website using data provided by the research team's programming model. The inbound and outbound trucks were animated based on a clock counter implemented in *JavaScript*. The counter was used to control the arrival and departure times of the trucks at their respective doors. This ensured a smooth and synchronized movement pattern for the trucks that aligned with the optimal schedule provided by the mathematical model.

Additionally, users could interact with the scene using *OrbitControls*, allowing for zooming and panning of the scene. Buttons were provided for switching camera views (bird's eye view and free roam) and controlling the animation (play and reset buttons). A form was provided to allow users to configure the number of inbound and outbound trucks, as well as the number of stack and strip doors. This form submission triggered updates in the 3D scene, reflecting the new configurations. The truck and door elements were dynamically added or removed based on the user input.

Several optimization techniques were implemented to enhance the performance of the website, particularly for running smoothly on the *Google Chrome browser*. Given the high number of graphical elements involved, optimization was crucial to maintaining a responsive and efficient user experience.

- **Polygon Reduction**: Simplified 3D models to reduce the number of polygons, thereby reducing the rendering load.
- **Clamping Interpolation Calculations**: Restricted truck animation interpolation calculations to avoid unnecessary computations and kept the trucks smoothly in transit between scheduled points.
- **Throttling Animation Frames**: Adjusted frame update rates to ensure animations ran efficiently without straining the browser.
- **Reducing Shadow Quality**: Lowered the resolution of shadows to improve performance while retaining sufficient visual quality for the scene.
- **Disabling Anti-Aliasing**: Anti-aliasing was disabled in certain views to reduce computational overhead, which was particularly useful when a large number of objects were in view.

One of the key challenges was synchronizing the animations of the trucks with the real-time clock. Initially, delta time was used, which varied based on rendering frame rates, causing inconsistencies. This was later replaced with direct reliance on the clock counter, which significantly improved the consistency of animations.

Another major challenge was optimizing the scene to maintain performance, given the high number of graphical elements involved. The optimization tactics mentioned above were critical in achieving a smooth user experience without compromising the visual quality of the simulations.

The final animation uses two files that are provided in Appendix D (Data for the Animations) and Appendix E (Animation Program). The GUI (GUI Program) is provided in Appendix F.

2.3 Results

The animation can be accessed via a link on the C2M2 website. Below are screenshots to illustrate some of the key features.

The Cross-Dock Environment: Figures 1 and 2 display the free roam view of two scenarios – one with two inbound/strip and two outbound/stack doors (2275) and one with four strip and stack doors (4475). Trucks and dock door labels are visually differentiated with blue for strip doors and green for stack doors. Figures 3 and 4 display the bird's eye view of these same two scanarios.

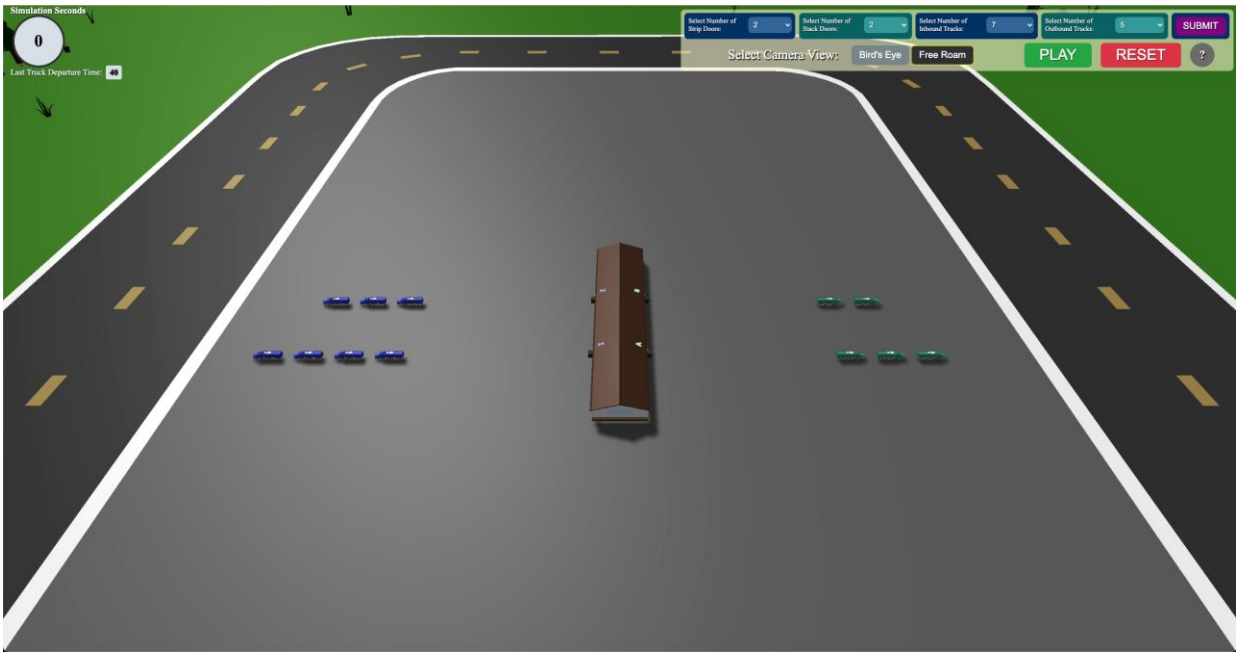

Figure 1: Free Roam View of Scenario_2275

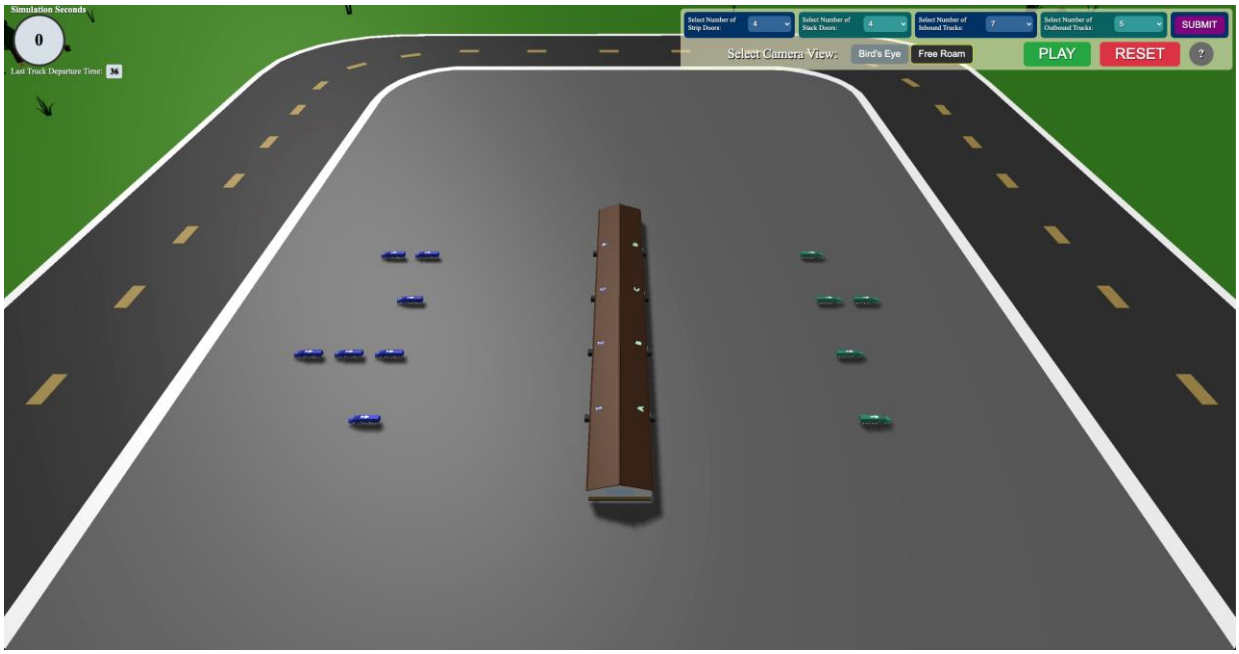

Figure 2: Free Roam View of Scenario_4475

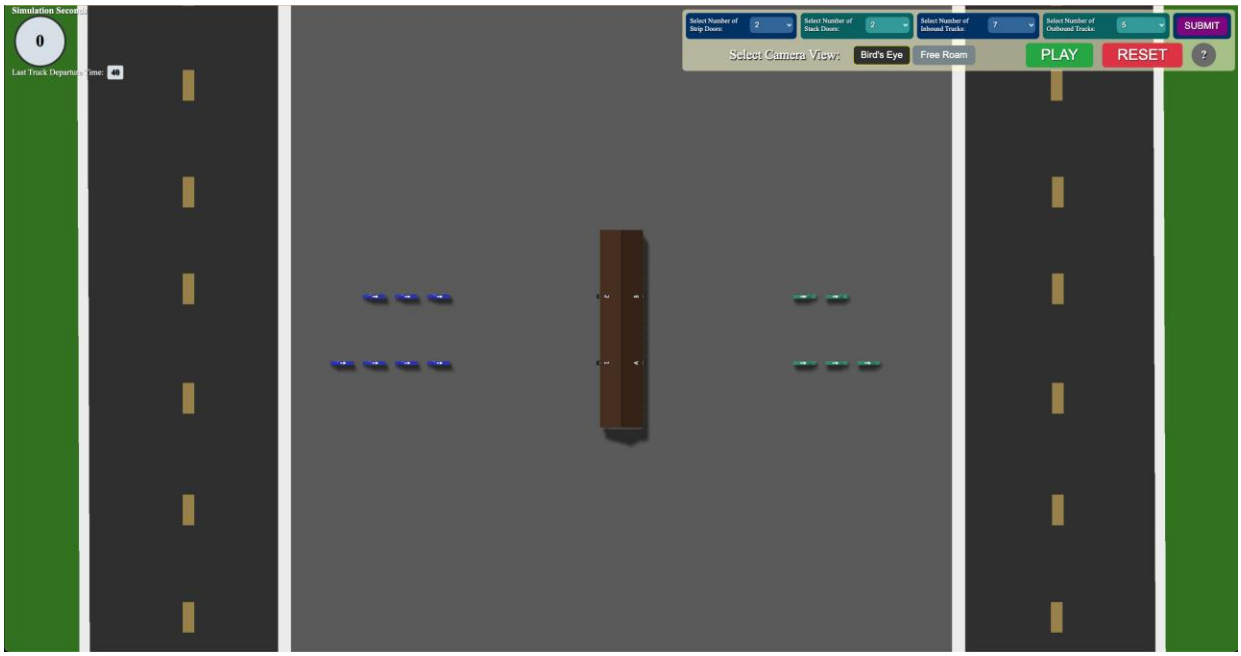

Figure 3 Bird's Eye View of Scenario_2275

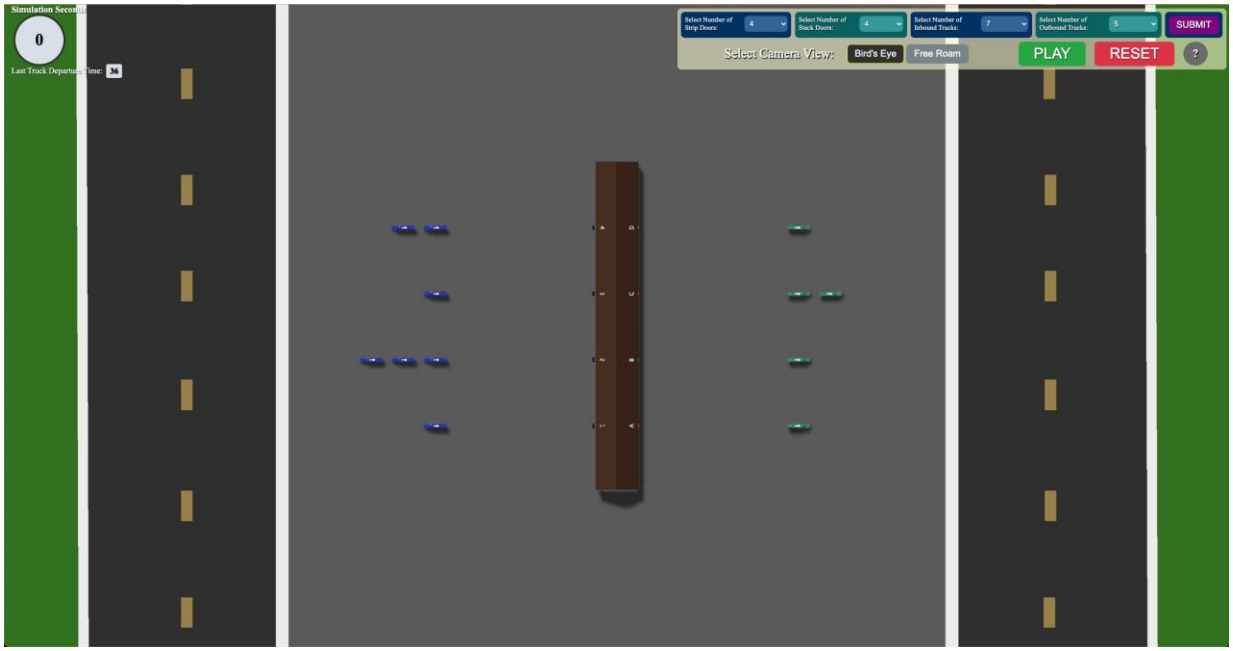

Figure 4: Bird's Eye View of Scenario_4475

Simulation-End Screens: Once an animation finishes, the user is presented with a calculation of the makespan in seconds and in real time minutes. Makespan is defined as the time between the arrival of the first inbound truck and the departure of the last outbound truck. Hence, the last truck departure time is equal to the makespan. Figure 5 shows the popup window indicating the animation is complete.

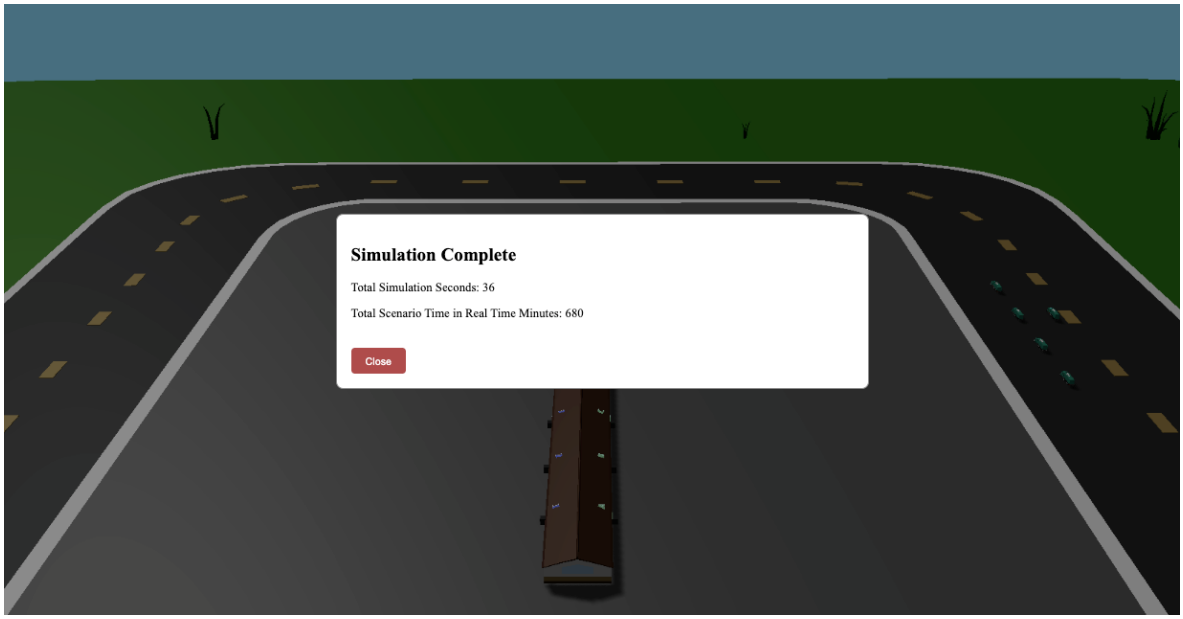

Figure 5: Animation-end pop-up

In Figure 6, one can see how the trucks are positioned after an animation is finished with the inbound trucks on the left and the outbound trucks on the right.

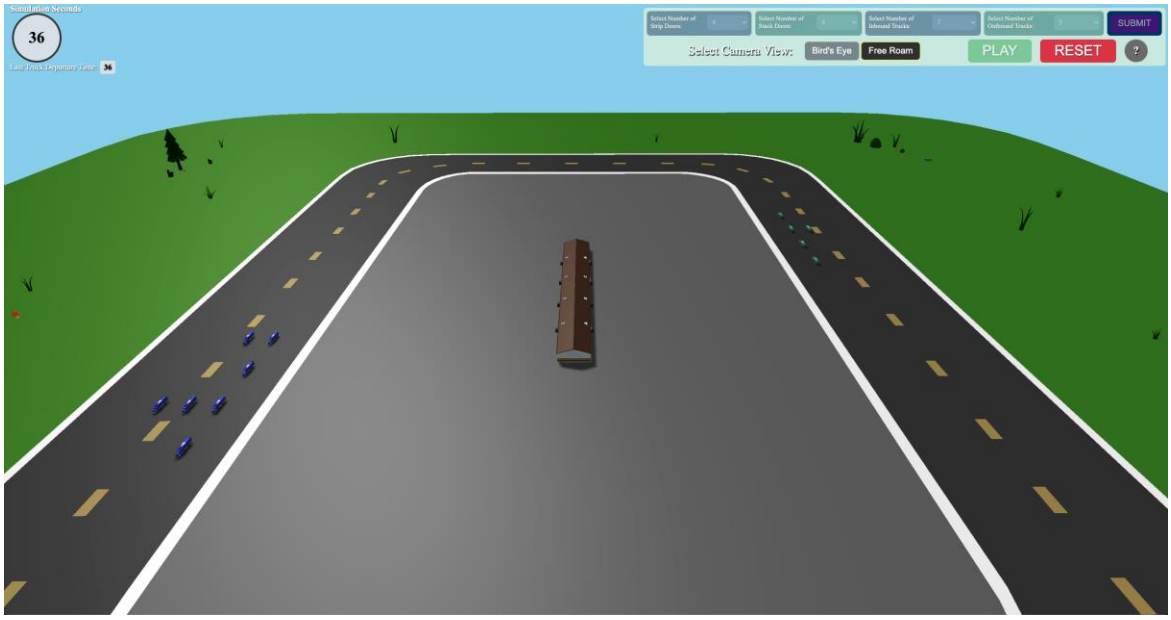

Figure 6: Trucks in parked positions after an animation scenario ends

Full Scene Mid-Animation: As the trucks progress through an animation, they will animate from their spawned-in positions to their docked positions and flash yellow as they're being loaded or unloaded. The clock will count up to provide a sense of time, then the controls will be greyed out and disabled as the animation plays.

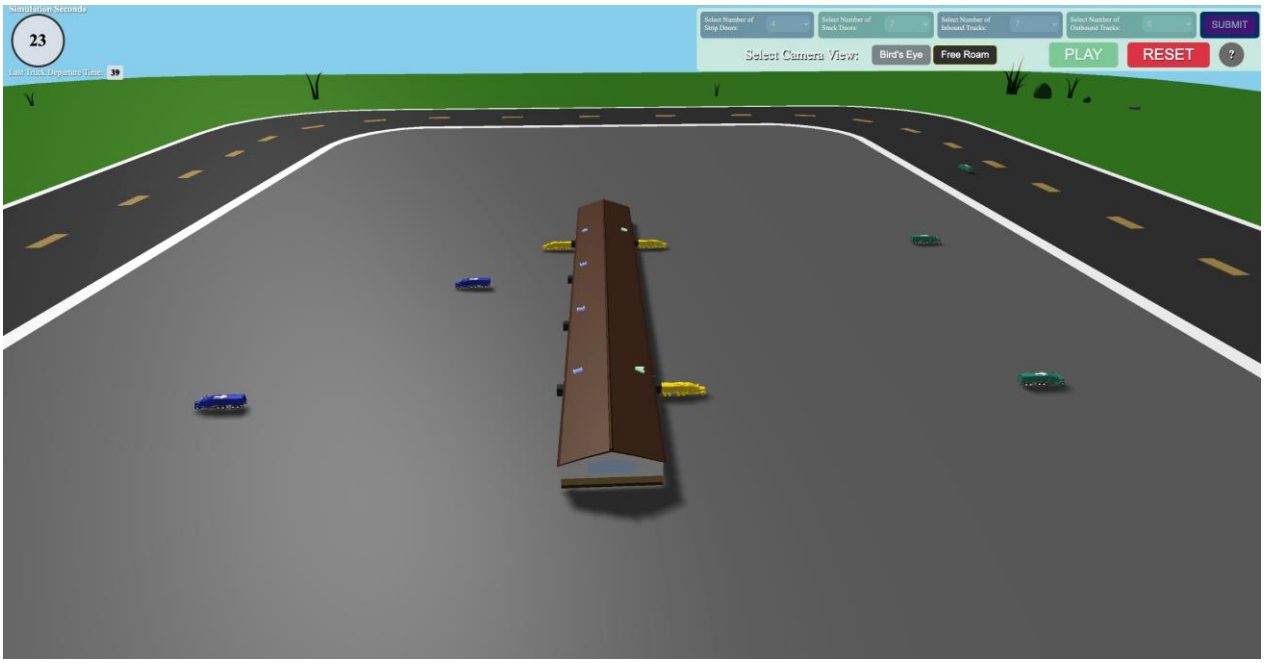

Figure 7: Trucks mid-animation capture

Clock, Controls, and Instructions GUIs: The website provides an interface for the user to select the number of dock doors and trucks in the scenario. The geometry will procedurally update based on user input. A clock is provided that will allow the user to track the number of simulations seconds as the scenario animates and observe how the makespan changes based on the scenario. Figure 8 displays this GUI.

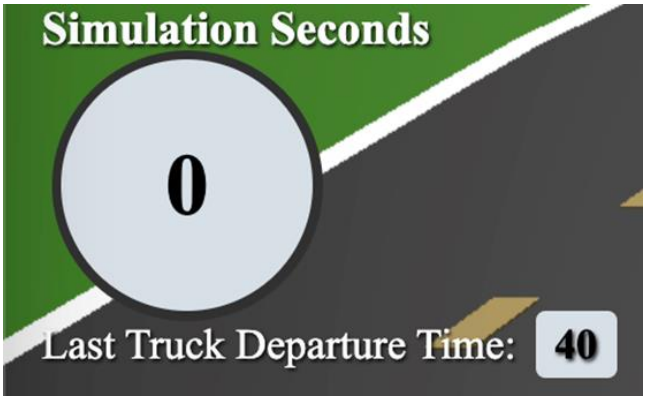

Figure 8: Clock counting simulation seconds and last truck departure (i.e., makespan)

The user interacts with the animation through the Controls GUI as shown in Figure 8. As can be seen, this allows the user to update the scenario, change camera views, play and reset the animation, and open a Help menu. The Help menu is opened when a user clicks the question mark on the right side of the Controls GUI. This includes information about the animation, instructions for use, and an about section. Figure 10 is the Help menu.

Figure 9: Controls GUI

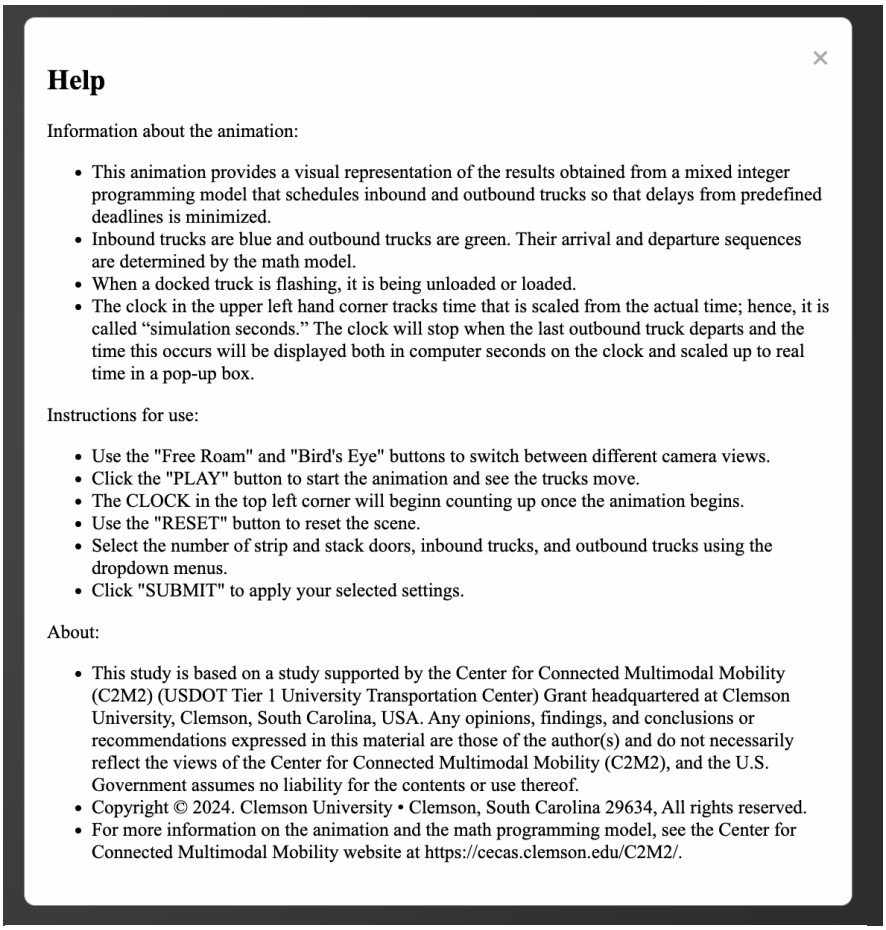

Figure 10: Help Menu

CHAPTER 3
Conclusions

This project used animation to present the results of previous research that addressed scheduling inbound and outbound trucks to and from a cross dock. This underlying research requires advanced knowledge of optimization to understand; however, the results are quite relevant for practical freight movement and could be used by practitioners who might well not have this knowledge. The fact that cross docking is a commonly used approach to load consolidation in today's logistics networks and will surely be more prominently features in future networks based on horizontal collaboration provided the motivation for this research which is to provide a mechanism for anyone to use the research results regardless of technical background.

This was accomplished with a web-based animation tool that will be available on the C2M2 website. There are a number of user-specified inputs that define the scenario which is animated. Times were scaled so that the animation runs in a few minutes rather than the multiple hours that would occur at a real cross dock; however, since parameters of the model and animation are consistent with practice, the sequence reflects the solution determined by the near-optimal solution of the mixed integer programming model and all timing of trucks (arrival and departures) are also the same as the near-optimal solution in a relative sense (i.e., they are just scaled to be shorter).

REFERENCES

Badyal, V. (2021), Design and Analysis of Efficient Freight Transportation Networks in a Collaborative Logistics Environment, Clemson University.

Bartholdi, J. J., & Gue, K. R. (2004). The best shape for a crossdock. *Transportation Science*, 38(2), 235-244.

Nogueira, T. H., Coutinho, F. P., Ribeiro, R. P., & Ravetti, M. G. (2020). Parallel-machine scheduling methodology for a multi-dock truck sequencing problem in a cross-docking center. *Computers & Industrial Engineering*, 143, 106391.

Van Belle, J., Valckenaers, P., & Cattrysse, D. (2012). Cross-docking: State of the art. *Omega*, 40(6), 827–846.

Yu, W. (2002). Operational strategies for cross docking systems, Iowa State University.

Yu, W. & P. Egbelu (2008). Scheduling of inbound and outbound trucks in cross docking systems with temporary storage, European Journal of Operational Research 184 (1) 377–396.

APPENDICES

Appendix A: MIP Model

This problem is first modeled as a mixed integer non-linear programming model and then linearized.

Notation

Sets

I	strip doors
O	stack doors
T_I	inbound trailers
T_O	outbound trailers
P	products

Parameters

H	length of the planning horizon (min)
A_i	arrival time of inbound trailer $i \in T_I$ (min)
T_{kg}	transshipment time between strip door $k \in I$ and stack door $g \in O$ (min)
W_p	time to unload or load product $p \in P$ (min)
Q_j	departure deadline for outbound trailer $j \in T_O$ (min)
F_j	penalty for each time unit outbound trailer $j \in T_O$ is late
C	trailer changeover time ($/min)
S_{ip}	number of units of product $p \in P$ available from inbound trailer $i \in T_I$
D_{jp}	number of units of product $p \in P$ required by outbound trailer $j \in T_O$
M	big number (greater than or equal to $\sum_{p \in P} D^i_{jp}$)
ϕ_1, ϕ_2	break points for the convex piece-wise linear function (min)
$\lambda_1, \lambda_2, \lambda_3$	penalty multipliers for the convex piece-wise linear function

Decision Variables

e^I_{ik}	arrival time of inbound trailer $i \in T_I$ at strip door $k \in I$ (≥ 0)
l^I_{ik}	leave time of inbound trailer $i \in T_I$ from strip door $k \in I$ (≥ 0)
e^O_{jg}	entry time of outbound trailer $j \in T_O$ at stack door $g \in O$ (≥ 0)
l^O_{jg}	leave time of outbound trailer $j \in T_O$ from stack door $g \in O$ (≥ 0)
$l_{\max}$	makespan or the time that last outbound trailer departs (≥ 0)
γ_j	tardiness of outbound trailer $j \in T_O$ (≥ 0)
r^p_{ij}	number of units of product $p \in P$ transferred from inbound trailer $i \in T_I$ to outbound trailer $j \in T_O$ (≥ 0 and Int)
α^k_{iq}	$= \begin{cases} 1, & \text{if inbound trailer } i \in T_I \text{ precedes inbound trailer } q \in T_I \text{ at strip door } k \in I \\ 0, & \text{otherwise} \end{cases}$
β^g_{jr}	$= \begin{cases} 1, & \text{if outbound trailer } j \in T_O \text{ precedes inbound trailer } r \in T_O \text{ at stack door } g \in O \\ 0, & \text{otherwise} \end{cases}$
x_{ik}	$= \begin{cases} 1, & \text{if inbound trailer } i \in T_I \text{ is assigned to strip door } k \in I \\ 0, & \text{otherwise} \end{cases}$

$$y_{jg} = \begin{cases} 1, & \text{if outbound trailer } j \in T_O \text{ is assigned to stack door } g \in O \\ 0, & \text{otherwise} \end{cases}$$

$$v_{ij}^{kg} = \begin{cases} 1, & \text{if freight is transferred from inbound trailer } i \text{ at door } k \text{ to outbound trailer } j \text{ at door } g \\ 0, & \text{otherwise} \end{cases}$$

Convex Piece-wise Linear Penalty Function $f(\gamma_j)$

$$f(\gamma_j) = \begin{cases} (\lambda_1 F_j)\gamma_j, & 0 \le \gamma_j < \phi_1 \\ (\lambda_2 F_j)\gamma_j + \phi_1 F_j(\lambda_1 - \lambda_2), & \phi_1 \le \gamma_j < \phi_2 \\ (\lambda_3 F_j)\gamma_j + \phi_1 F_j(\lambda_1 - \lambda_2) + \phi_2 F_j(\lambda_2 - \lambda_3), & \phi_2 \le \gamma_j \end{cases} \quad (1)$$

$$\text{Minimize} \sum_{j \in T_O} f(\gamma_j) + \max_{\substack{j \in T_O, \\ g \in O}} \{l_{jg}^O\} \quad (2)$$

Subject to,

$$\sum_{k \in I} x_{ik} = 1 \qquad \forall i \in T_I \quad (3)$$

$$\sum_{g \in O} y_{jg} = 1 \qquad \forall j \in T_O \quad (4)$$

$$e_{ik}^I \ge x_{ik} A_i \qquad \forall i \in T_I, k \in I \quad (5)$$

$$l_{ik}^I - e_{ik}^I \ge x_{ik} \left(\sum_{p \in P} S_{ip} W_p \right) \qquad \forall i \in T_I, k \in I \quad (6)$$

$$e_{qk}^I \ge C + l_{ik}^I - H\left(1 - \alpha_{iq}^k\right) \qquad \forall i, q \in T_I : i \ne q, k \in I \quad (7)$$

$$\alpha_{iq}^k + \alpha_{qi}^k \ge x_{ik} + x_{qk} - 1 \qquad \forall i, q \in T_I : i \ne q, k \in I \quad (8)$$

$$\alpha_{iq}^k + \alpha_{qi}^k \le 1 \qquad \forall i, q \in T_I : i \ne q, k \in I \quad (9)$$

$$\alpha_{ii}^{k} = 0 \qquad \forall i \in T_I, k \in I \tag{10}$$

$$e_{rg}^{O} \geq C + l_{jg}^{O} - H\left(1 - \beta_{jr}^{g}\right) \qquad \forall j,r \in T_O : j \neq r, g \in O \tag{11}$$

$$\beta_{jr}^{g} + \beta_{rj}^{g} \geq y_{jg} + y_{rg} - 1 \qquad \forall j,r \in To : j \neq r, g \in O \tag{12}$$

$$\beta_{jr}^{g} + \beta_{rj}^{g} \leq 1 \qquad \forall j,r \in T_O : j \neq r, g \in O \tag{13}$$

$$\beta_{jj}^{g} = 0 \qquad \forall j \in T_O, g \in O \tag{14}$$

$$\sum_{j \in T_O} \sum_{k \in I} \sum_{g \in O} r_{ij}^{p} \leq S_{ip} \qquad \forall i \in T_I, p \in P \tag{15}$$

$$\sum_{i \in T_I} \sum_{k \in I} \sum_{g \in O} r_{ij}^{p} \geq D_{jp} \qquad \forall j \in T_O, p \in P \tag{16}$$

$$\sum_{p \in P} r_{ij}^{p} \leq M \left(\sum_{k \in I} \sum_{g \in O} v_{ij}^{kg} \right) \qquad \forall i \in T_I, j \in T_O \tag{17}$$

$$v_{ij}^{kg} \leq x_{ik} \qquad \forall i \in T_I, j \in T_O, k \in I, g \in O \tag{18}$$

$$v_{ij}^{kg} \leq y_{jg} \qquad \forall i \in T_I, j \in T_O, k \in I, g \in O \tag{19}$$

$$l_{jg}^{O} \geq e_{ik}^{I} + \frac{\sum_{p \in P} S_{ip} W_p}{2} + T_{kg} + \sum_{p \in P} r_{ij}^{p} W_p - H\left(1 - v_{ij}^{kg}\right) \qquad \forall i \in T_I, j \in T_O, k \in I, g \in O \tag{20}$$

$$l_{jg}^{O} \geq e_{jg}^{O} + y_{jg} \left(\sum_{p \in P} D_{jp} W_p \right) \qquad \forall j \in T_O, g \in O \tag{21}$$

$$l_{jg}^{O} \leq y_{jg} Q_j + \gamma_j \qquad \forall j \in T_O, g \in O \tag{22}$$

$$e_{ik}^{I} \leq H x_{ik} \qquad \forall i \in T_I, k \in I \tag{23}$$

$$e_{ik}^{I}, e_{jg}^{O}, l_{ik}^{I}, l_{jg}^{O}, l_{\max}, \gamma_j, r_{ij}^{p} \geq 0 \qquad \forall i, j, k, g \tag{24}$$

$$\alpha_{iq}^{k}, \beta_{iq}^{k}, x_{ik}, y_{jg}, v_{ij}^{kg} \in [0,1] \qquad \forall i, j, k, g \tag{25}$$

The nonlinear objective function (2) minimizes the makespan and outbound trailer tardiness penalty. Inbound and outbound trailers are restricted to one strip and stack door, respectively, by Constraints (3) and Constraints (4). Constraints (5) enforce the physical restriction that inbound trailers must arrive before they can be docked. Constraints (6) address the service time of inbound trailers (e.g., leave time - entry time) at their strip doors by ensuring that it is should be greater than or equal to the unloading time for all the loaded products.

Constraints (7)-(10) enforce precedence relationships for arriving trailers at strip doors. Specifically, if trailer 'q' is scheduled to follow trailer 'i' at a strip door, its docking time must be greater than the sum of trailer 'i' departure time and the changeover time. Outbound trailers have analogous precedence constraints, (11)-(14), that function like the inbound trailers' precedence constraints.

The next set of constraints are standards for the transportation problem. Constraints (15) restrict the number of units shipped from an inbound trailer to all outbound trailers to be less than the capacity. The demand of each outbound trailer is met is ensured by constraints (16). Freight transfer path tracking is performed by constraints (17)-(19).

Outbound trailer service time constraints are modeled by constraints (20) and constraints (21). Here, an outbound trailer cannot leave its stack door before all demanded products are loaded from their inbound trailers. Constraints (22) tracks any tardiness that occurs when an outbound trailer's actual departure exceeds the deadline and applies a penalty. Finally, Constraints (23) ensures that all inbound trailers are scheduled before the end of the planning horizon.

Linearizing the Model

The previous model has a nonlinear objective function that can be linearized for computational efficiency. This is achieved with the addition of a few new variables and constraints and described now.

$$\text{Minimize} \sum_{j \in T_O} z_j + l_{\max} \tag{26}$$

$$z_j \geq (\lambda_1 F_j)\, \gamma_j \qquad \forall j \in T_O \tag{27}$$

$$z_j \geq (\lambda_2 F_j)\, \gamma_j + \phi_1 F_j (\lambda_1 - \lambda_2) \qquad \forall j \in T_O \tag{28}$$

$$z_j \geq (\lambda_3 F_j)\gamma_j + \phi_1 F_j(\lambda_1 - \lambda_2) + \phi_2 F_j(\lambda_2 - \lambda_3) \qquad \forall j \in T_O \tag{29}$$

$$l_{\max} \geq \sum_{g \in O} l_{jg}^{O} \qquad \forall j \in T_O \tag{30}$$

The new model is linear with objective function (26) and constraints (3)-(23) and constraints (27)-(30).

The scheduling problem that is focus of this research extends the work on single-door, single-product cross-docks in important ways that include addressing multi-products and multi-doors, allowing products from inbound trailers to be used interchangeably to satisfy demand in outbound trailers, and including soft-deadlines for outbound trailers. This class of scheduling problems have been proven to be NP-hard (Chen & Lee, 2009; Tootkaleh, Ghomi, & Sajadieh, 2016; ?) which means that using any known algorithm that finds an optimal solution will require exponentially increasing computational time as the number of trailers increase. Hence, using heuristic methods that find approximate solutions is justified for larger, more realistic-sized scenarios. If this model is used in real-time, the cross-dock operator needs to make docking decisions soon after trailers arrive. To accomplish this, a two stage approach was developed. The first stage is a constructive heuristic that produces a good starting solution quickly. Then, a PBSA metaheuristic is used to improve solution quality. The construction heuristic output also assists in creating product assignments.

Appendix B: Mixed-Door Cross-Dock Heuristic (MDCDH)

Yu (2002); Yu and Egbelu (2008) proposed a heuristic for scheduling trailers in a cross-dock with one strip and one stack door. They assumed freight could be moved across the cross-dock by either direct transfer (e.g., move directly from inbound to outbound trailers) or by using temporary storage. Their objective was to maximize the former or, alternatively, minimizing the latter. With this objective, the iterative two-stage procedure was proposed. Stage 1 determines a sequence of inbound trailers with products that can meet the demand for each outbound trailer. This is called the "associated inbound trailers/trailers" or AIT. Clearly, there can be multiple AITs for each outbound trailer; however, a strategy for selecting the inbound trailers based on minimizing the storage/unloading time is used to create one AIT for each outbound trailer. Stage 2 considers all outbound trailers not scheduled for docking and selects the one whose AIT requires shortest storage and/or unloading time. The available inbound freight is updated and the process repeats until all of the outbound trailers are scheduled. Finally, these papers concluded that two-stage heuristics performed the best.

Because of their success, a two-stage approach was selected as the basis for the Multi-Door Cross-Dock Heuristic (MDCDH) that is proposed for this research problem of scheduling trailers when the cross-dock has multiple doors and the inbound trailer arrivals are asynchronous. The MDCDH is explained in the subsequent sections.

2.1. MDCDH: First Stage

The MDCDH is predicated on the following basic dynamics of a cross-dock. Inbound trailers arrive from suppliers with products that can either be transferred directly to outbound trailers or placed in temporary storage. Outbound trailers can have their demand satisfied either from products that are transferred directly from inbound trailers docked at a strip door or from temporary storage.

The first stage iteratively creates an AIT for each outbound trailer that defines the product transfers required to satisfy each trailer's demand. Only inbound trailers that have not been scheduled can be used to create an AIT. The AIT uses an iterative approach in which inbound trailers are selected one-by-one using the inbound trailer selection strategy described below. The iterations stop when the trailer's demand is satisfied. That is, the products available in temporary storage plus those on the trailers in the AIT exceed the each product type.

Inbound Trailer Selection Strategy

Inbound trailers are selected to create the AIT using an iterative process. The selection is based on the time required for unloading products in excess of what is required by scheduled outbound trailers and the earliest time that the product is available.

The first metric, minimizing unloading excess supply, is intended to reduce the time that a trailer is docked at a strip door with no activity. Its calculation is based on a foresight method Yu (2002); Yu and Egbelu (2008). However, using the metric alone to select inbound trailers can have an unintended consequence because the product that is excess after satisfying demand for an outbound trailer at a specific stack door in this iteration might be able to satisfy the demand of outbound trailer that will be assigned to the stack door in the next iteration, but this would not be considered. Hence, inbound trailers that place all items in temporary storage for short duration would be selected instead.

Let ES_{ij} be the time required to unload the excess products in inbound trailer $i \in T_I$ that is part of the AIT for outbound trailer $j \in T_O$. Define T_{UO} as the set of unscheduled outbound trailers, where $T_{UO} \subseteq T_O$ and D_{jp}^{iter} as the number of product $p \in P$ that will be used to satisfy the demand on outbound trailer $j \in T_O$ after iteration 'iter'. $\alpha = min(|O|, |T_{UO} \setminus j|)$ denotes how many outbound trailers will be scheduled immediately after outbound trailer $j \in T_O$. Finally, $T_{UO}^{\alpha} \subseteq T_{UO} \setminus j$ is the set of unscheduled outbound trailers ($|T_{UO}^{\alpha}| = \alpha$). With these definitions, ES_{ij} is given by equation 31. Calculations involving a hypothetical example is shown

in Appendix 6.6.

$$ES_{ij} = min_{l \in T^{\alpha}_{UO}} \left(\sum_{p \in P} max(0, S_{ip} - D^{iter}_{jp} - \sum_{l \in T^{\alpha}_{UO}} D_{lp}) W_p \right) \tag{31}$$

Using the second metric seeks to assign products from inbound trailers so that the outbound trailer resides at the stack door for the shortest time possible. Hence, products that are available for loading as soon as as the outbound trailer is docked are preferable. Equation 32 is used to calculate the product availability time (AT_i) for inbound trailer $i \in T_I$.

$$AT_i = min_{k \in I}(e^I_{ik}) + \frac{\sum_{p \in P} S_{ip} W_p}{2} \tag{32}$$

Both measures are calculated for all available inbound trailers and combined using an equally weighted sum of the relative percentage deviations (RPD).

$$\text{Relative Percentage Deviation (RPD)} = \left(\frac{\text{measure value-best measure value}}{\text{best measure value}} \right) * 100 \tag{33}$$

The inbound trailer with the smallest value of the weighted sum is selected and ties are randomly broken. This multi-criteria decision-making algorithm is presented in 6.3.

2.2. MDCDH: Second Stage

The second stage uses information regarding outbound trailers and their associated inbound trailer(s) from the first stage in an outbound trailer selection strategy to determine the outbound trailer and its AIT that is scheduled next.

Outbound Trailer Selection Strategy: The two measures as previously to identify the outbound trailer (and its AIT) are the time that the outbound trailer departs and the time to unload the excess supply for all inbound trailers in the outbound trailer's AIT. An outbound trailer and trailer leave the cross-dock when all products that are demanded have been loaded. This measure is used to evaluate the assumed goal to maximize stack door availability which is accomplished by having outbound trailers leave immediately after loading.

As explained before a similar goal exists for the strip doors of maximum availability. This, too, is facilitated by having the docked trailers leave as soon as it is completely unloaded. This is facilitated by having products not useful in the short term unloaded as quickly as possible; hence, the second measure.

If one or more outbound trailers has a computed departure time that exceeds its deadline, the most tardy outbound trailer is scheduled next. Otherwise, multi-criteria decision-making algorithm presented in 6.3 is used for outbound trailer selection.

Inbound and outbound trailers are scheduled to dock as quickly as early as possible. For inbound trailers, this is the earliest of when it arrives and when a strip door is free. For outbound trailers, it is when a stack door is free. In case of ties, the door with the lowest idle time after the last scheduled trailer leaves is selected.

To illustrate the relationship between schedule idle time, consider the hypothetical situation depicted in Figures 1a and 1b. Consider a cross-dock with two strip doors and the task is to schedule IT-2 that will arrive at 'A'. IT-2 can be scheduled at strip door 1 because IT-4 leaves at 'B' and the changeover time is 'T.C.' so $B + T.C. < A$. So, IT-2 can use either strip door at the same entry time; however, the outcome depicted in

Figure 1a has more idle time and less utilization than the outcome in Figure 1b.

(a) Strip door-2 idle time = A

(b) Strip door-1 idle time = (A - B - T.C.)

Figure 1.: Selection of a docking door using idle time when candidate doors have the same earliest possible docking time

Appendix C: Population-based Simulated Annealing

Simulated Annealing (SA) is a probabilistic technique inspired by the process of annealing materials that has been found effective in finding good solutions to combinatorial optimization problems. SA was developed by Kirkpatrick, Gelatt, and Vecchi (1983) and has been successfully applied to applied to large-scale problems in many application domains containing continuous variables, discrete variables, or both. One noteworthy advantage that SA has over greedy algorithms is that is is more effective in escaping local optima. SA accomplishes this through diversification that includes inferior solutions in the neighborhood of the better solutions using the Boltzmann distribution. Relative to this research, Liao, Egbelu, and Chang (2013) and Keshtzari, Naderi, and Mehdizadeh (2016) have successfully used SA to address post-distribution cross-dock scheduling problems.

Population Based Simulated Annealing (PBSA) is an extension of SA that relies on a *population* of solutions to determine the search direction rather than one solution from each iteration. PBSA was successfully applied to scheduling a cross-dock with multiple -doors by Assadi and Bagheri (2016) where the population of solutions is generated by pairwise exchanges and transshipment flows across the cross-dock are determined by a "first-come first-serve" approach.

The approach taken in this research uses the solution from the previously-described MDCDH as the starting solution for a PBSA refinement phase. The population of solutions in each iteration are neighborhood solutions of the best solution from the previous iteration. That is, the best solution in one iteration is subjected to pairwise exchange and insert operations to produce a population of solutions for the next generation. For each solution in the generated neighborhood, the inbound-to-outbound paths to transfer products as well as the departure times for the outbound trailer departure times are computed using the inbound trailer selection strategy. The solution with the best value of the objective function is retained and used to generate the population of solutions for the next iteration in a process called intensification. Now, if this best solution is better than the current best global solution, it is declared the current best global solution. If it is less than the best solution, it is identified as a worst solution and is accepted for the next generation's population based on a probability from the Boltzmann distribution. The algorithm terminates based on a predetermined number of iterations.

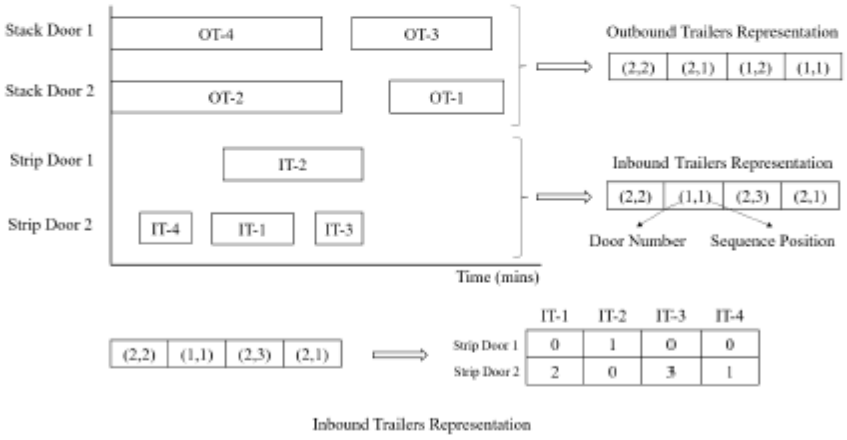

Figure 2.: Trailer-to-door assignment and sequence representation

A tuple is used for the solution representation in this research as illustrated in Figure 2. There is a list of tuples for inbound and outbound trailers that identify the docking door assigned to each trailer and the order that trailers will dock a their assigned doors. Specifically, the list's index is the trailer number, the first tuple number is docking door to which this trailer is assigned, and the second tuple number is the position in the docking sequence for this trailer at its assigned door.

Using the PBSA to improve the MDCDH's initial solutions poses one significant issue. Most if not all of the generated neighborhood solutions will have different trailer docking sequences so it would be pure luck if the product transfers from inbound to outbound trailers that were produced by the MDCDH were effective. Hence, these must be updated to move towards better solutions.

To create a more efficient set of product transfers, the MDCDH is used with one important modification modification, namely, that an AIT is not created for the outbound trailers that are unscheduled at an iteration. Instead, *subsets* of outbound trailers are scheduled for each stack door. Trailers are assigned to subsets based on their docking position that is defined by the solution. The subsets are scheduled based on their docking time priority. To illustrate this process, consider the outbound trailer schedule in Figure 2. The first step is to define the subsets which, in this example, are {OT-4, OT-2} and {OT-3, OT-1}. Then, MDCDH is used to schedule the trailers in the first subset, {OT-4, OT-2}. Next, trailers are scheduled in {OT-3, OT-1}. (If there were other subsets, these would be scheduled sequentially.) One advantage of this approach is that it avoids an infeasibility problem that can arise if all trailers are scheduled at one time. In this example, this would happen if OT-4 is scheduled after OT-3 because OT-4 must be loaded before OT-3.

The pseudo-codes describing the pairwise exchange, insertion method, and the PBSA algorithm are provided in the appendix, Sections 6.1, 6.2, and 6.5, respectively. Section 4 provides a comparison of the computational time and solution quality between the PBSA and an optimization approach using a commercial solver.

Appendix D: Data for the Animations

This file contains the arrival and departure times for all inbound and outbound trucks for the scenarios that are provided in the animation.

```
const scenario_2275 = [
   { door: '1', arrive: 12, depart: 16 },
   { door: '1', arrive: 18, depart: 21 },
   { door: '1', arrive: 24, depart: 27 },
   { door: '1', arrive: 29, depart: 31 },

   { door: '2', arrive: 2, depart: 5 },
   { door: '2', arrive: 16, depart: 20 },
   { door: '2', arrive: 27, depart: 31 },

   { door: 'A', arrive: 13, depart: 20 },
   { door: 'A', arrive: 21, depart: 25 },
   { door: 'A', arrive: 26, depart: 40 },

   { door: 'B', arrive: 13, depart: 26 },
   { door: 'B', arrive: 28, depart: 38 }
];

const scenario_2475 = [
   { door: '1', arrive: 2, depart: 5 },
   { door: '1', arrive: 12, depart: 16 },
   { door: '1', arrive: 18, depart: 21 },
   { door: '1', arrive: 23, depart: 25 },
   { door: '1', arrive: 27, depart: 31 },

   { door: '2', arrive: 16, depart: 20 },
   { door: '2', arrive: 24, depart: 27 },

   { door: 'A', arrive: 18, depart: 22 },

   { door: 'B', arrive: 29, depart: 39 },

   { door: 'C', arrive: 13, depart: 20 },
   { door: 'C', arrive: 22, depart: 36 },

   { door: 'D', arrive: 13, depart: 26 }
];

const scenario_4275 = [
   { door: '1', arrive: 27, depart: 31 },

   { door: '2', arrive: 12, depart: 16 },

   { door: '3', arrive: 2, depart: 5 },
   { door: '3', arrive: 17, depart: 20 },
   { door: '3', arrive: 24, depart: 27 },
```

```
    { door: '4', arrive: 16, depart: 20 },
    { door: '4', arrive: 23, depart: 25 },

    { door: 'A', arrive: 13, depart: 26 },
    { door: 'A', arrive: 28, depart: 38 },

    { door: 'B', arrive: 13, depart: 19 },
    { door: 'B', arrive: 21, depart: 24 },
    { door: 'B', arrive: 26, depart: 39 }
];

const scenario_4475 = [
    { door: '1', arrive: 17, depart: 20 },

    { door: '2', arrive: 16, depart: 20 },
    { door: '2', arrive: 23, depart: 25 },
    { door: '2', arrive: 27, depart: 31 },

    { door: '3', arrive: 12, depart: 16 },

    { door: '4', arrive: 2, depart: 5 },
    { door: '4', arrive: 24, depart: 27 },

    { door: 'A', arrive: 21, depart: 31 },

    { door: 'B', arrive: 13, depart: 26 },

    { door: 'C', arrive: 13, depart: 19 },
    { door: 'C', arrive: 22, depart: 36 },

    { door: 'D', arrive: 16, depart: 20 }
];
```

Appendix E: Animation Program

This appendix is the code for the animation.

```
// Import the scenarios.js dynamically
const scenarios = {
    scenario_2275,
    scenario_2475,
    scenario_4275,
    scenario_4475
};

function main() {
    const canvas = document.querySelector('#c');
    const renderer = new THREE.WebGLRenderer({ antialias: false, canvas });
    renderer.shadowMap.enabled = true;
    renderer.shadowMap.type = THREE.PCFSoftShadowMap; // Use soft shadows
    renderer.shadowMap.autoUpdate = true; // Update the shadow map every frame
    renderer.shadowMap.needsUpdate = true;
    renderer.setSize(window.innerWidth, window.innerHeight);
    renderer.setClearColor(0x87CEEB, 1); // Set the background color to sky blue
    document.body.appendChild(renderer.domElement);

    const fov = 54.43;
    const aspect = window.innerWidth / window.innerHeight;
    const near = 1;
    const far = 1000000;

    const cameraPerspective = new THREE.PerspectiveCamera(fov, aspect, near, far);
    cameraPerspective.position.set(0, 110, -120);
    cameraPerspective.lookAt(0, 0, 0);

    const cameraRightPerspective = new THREE.PerspectiveCamera(fov, aspect, near, far);
    cameraRightPerspective.position.set(-44, 10, 48);
    cameraRightPerspective.rotation.y = -0.872665;
    cameraRightPerspective.rotation.x = -0.0698132;

    const frustumSize = 200;
    const aspectOrtho = window.innerWidth / window.innerHeight;
    const cameraTopView = new THREE.OrthographicCamera(
        frustumSize * aspectOrtho / -2,
        frustumSize * aspectOrtho / 2,
        frustumSize / 2,
        frustumSize / -2,
        0.1,
        2000
    );

    // Set the position for a top-down view
    cameraTopView.position.set(0, 200, 0); // Elevated to get a top-down view
    cameraTopView.lookAt(0, 0, 0); // Look at the center of the scene
```

```
    // Explicitly set the 'up' vector to correct the orientation
    cameraTopView.up.set(0, 0, 1); // Ensure the "up" direction is set to match your scene orientation

    // Apply a 180-degree rotation around the Y-axis to align the view
    cameraTopView.rotation.set(-Math.PI / 2, 0, 0); // Rotate to look directly downwards

    const SIMULATION_SPEED_FACTOR = 2;

    // Define the yellow Lambert material
    const yellowLambertMaterial = new THREE.MeshStandardMaterial({ color: 0xbfa600 });

    const scene = new THREE.Scene();

    // Directional Light with Shadows
    {
        const directionalLight = new THREE.DirectionalLight(0xffffff, 1.0);
        directionalLight.position.set(100, 200, 200); // Adjust position as needed
        directionalLight.castShadow = true;

        // Increase the shadow map size for higher resolution shadows
        directionalLight.shadow.mapSize.width = 1024;
        directionalLight.shadow.mapSize.height = 1024;

        // Adjust the shadow camera frustum to focus on the area of interest
        directionalLight.shadow.camera.near = 1;
        directionalLight.shadow.camera.far = 1000;
        directionalLight.shadow.camera.left = -300;
        directionalLight.shadow.camera.right = 300;
        directionalLight.shadow.camera.top = 300;
        directionalLight.shadow.camera.bottom = -300;

        // Adjust the shadow bias to reduce shadow artifacts
        directionalLight.shadow.bias = -0.001; // Adjust this value as needed

        scene.add(directionalLight);
    }

    // Ambient Light
    {
        const ambColor = 0xFFFFFF;
        const ambIntensity = 0.6;
        const ambLight = new THREE.AmbientLight(ambColor, ambIntensity);
        scene.add(ambLight);
    }

    const objLoader = new THREE.OBJLoader();
    const mtlLoader = new THREE.MTLLoader();

    // Define materials for stack and strip door labels
```

```
    const stackLabelMaterial = new THREE.MeshStandardMaterial({
        color: new THREE.Color(0.75, 0.82, 1),
        emissive: 0x0208c2,
        side: THREE.DoubleSide
    });

    const stripLabelMaterial = new THREE.MeshStandardMaterial({
        color: new THREE.Color(0.4, 0.8, 0.5),
        emissive: 0x555555,
        side: THREE.DoubleSide
    });

    let distributionCenter = null;
    let ground = null;
    let bayDoors = [];
    let trucks = [];
    let stripLabels = [];
    let stackLabels = [];
    let duplicateLabels = [];

    const controlsPerspective = new THREE.OrbitControls(cameraPerspective,
renderer.domElement);
    controlsPerspective.target.set(0, 0, 0);
    controlsPerspective.update();

    const controlsTopView = new THREE.OrbitControls(cameraTopView, renderer.domElement);
    controlsTopView.enableRotate = false; // Disable rotation
    controlsTopView.enablePan = false; // Disable panning
    controlsTopView.enableZoom = true; // Enable zoom
    controlsTopView.update();

    // Initial active camera setup
    let activeCamera = cameraPerspective; // Default camera is Free Roam
    let activeControl = controlsPerspective; // Default controls for the perspective camera

    // Camera button elements
    const birdEyeButton = document.getElementById('birdEyeView');
    const freeRoamButton = document.getElementById('freeRoamView');

    // Function to switch to Bird's Eye view
    function switchToBirdEye() {
        activeCamera = cameraTopView;
        activeControl = controlsTopView;
        birdEyeButton.style.backgroundColor = '#313131';
        freeRoamButton.style.backgroundColor = '#78868a';
        birdEyeButton.style.border = '2px solid #fbff00';
        freeRoamButton.style.border = '2px solid #78868a';
        activeControl.update();
    }

    // Function to switch to Free Roam view
```

```
function switchToFreeRoam() {
    activeCamera = cameraPerspective;
    activeControl = controlsPerspective;
    freeRoamButton.style.backgroundColor = '#313131';
    birdEyeButton.style.backgroundColor = '#78868a';
    freeRoamButton.style.border = '2px solid #fbff00';
    birdEyeButton.style.border = '2px solid #78868a';
    activeControl.update();
}

// Attach event listeners to the camera buttons
birdEyeButton.addEventListener('click', switchToBirdEye);
freeRoamButton.addEventListener('click', switchToFreeRoam);

// Set the initial active button
switchToFreeRoam(); // Ensure Free Roam is the default view

const clock = new THREE.Clock();
let counterInterval; // Global variable to store the counter interval

let truckSchedule = scenarios.scenario_2275; // Load default scenario on page load

// Find the last truck departure time from the truck schedule
const lastTruckDepartureTime = Math.max(...truckSchedule.map(truck => truck.depart));

// Update the last truck departure time display
document.getElementById('lastTruckDepartureTime').textContent = lastTruckDepartureTime;

function loadDistributionCenter() {
    if (distributionCenter) {
        scene.remove(distributionCenter);
        bayDoors.forEach(door => scene.remove(door));
        trucks.forEach(truck => scene.remove(truck));
        stripLabels.forEach(label => scene.remove(label));
        stackLabels.forEach(label => scene.remove(label));
        duplicateLabels.forEach(label => scene.remove(label));
        bayDoors = [];
        trucks = [];
        stripLabels = [];
        stackLabels = [];
        duplicateLabels = [];
    }

    mtlLoader.load('/static/distribution_center.mtl', function (materials) {
        materials.preload();
        objLoader.setMaterials(materials);
        objLoader.load(
            '/static/distribution_center.obj',
            function (object) {
                adjustDistributionCenterWidth(object);
                object.traverse(function (child) {
```

```
                if (child instanceof THREE.Mesh) {
                    child.castShadow = true;
                    child.receiveShadow = false;
                }
            });
            scene.add(object);
            distributionCenter = object;
            loadAndPlaceBayDoorsAndTrucks(object);
            if (!ground) {
                loadGround();
            }
            renderer.render(scene, activeCamera);
        },
        function (xhr) {
            console.log((xhr.loaded / xhr.total * 100) + '% loaded');
        },
        function (error) {
            console.error('Error loading distribution center', error);
        }
    );
}, function (error) {
    console.error('Error loading distribution center materials', error);
});
}

async function adjustDistributionCenterWidth(object) {
    const numStackDoors = parseInt(document.getElementById('numStackDoors').value);
    const numStripDoors = parseInt(document.getElementById('numStripDoors').value);
    const doorWidth = 1;
    const doorOffset = 30;
    const n = Math.max(numStackDoors, numStripDoors);
    const width = doorWidth * n + doorOffset * (n + 1);

    const boundingBox = new THREE.Box3().setFromObject(object);
    const size = boundingBox.getSize(new THREE.Vector3());

    object.scale.set(1, 1, width / size.z);
}

async function loadAndPlaceBayDoorsAndTrucks(distributionCenter) {

    // Load bay doors and truck materials
    mtlLoader.load('/static/bay_door.mtl', function (materials) {
        materials.preload();
        objLoader.setMaterials(materials);
        objLoader.load(
            '/static/bay_door.obj',
            function (bayDoor) {
                // Load trucks
                mtlLoader.load('/static/outbound_truck.mtl', function (outboundTruckMaterials) {
                    outboundTruckMaterials.preload();
```

```
                mtlLoader.load('/static/inbound_truck.mtl', function (inboundTruckMaterials) {
                    inboundTruckMaterials.preload();
                    objLoader.setMaterials(outboundTruckMaterials);
                    objLoader.load(
                        '/static/outbound_truck.obj',
                        function (outboundTruck) {
                            outboundTruck.traverse(function (child) {
                                if (child instanceof THREE.Mesh) {
                                    child.castShadow = true;
                                    child.receiveShadow = false;
                                    child.originalMaterial = child.material; // Store the original material
                                }
                            });
                            objLoader.setMaterials(inboundTruckMaterials);
                            objLoader.load(
                                '/static/inbound_truck.obj',
                                function (inboundTruck) {
                                    inboundTruck.traverse(function (child) {
                                        if (child instanceof THREE.Mesh) {
                                            child.castShadow = true;
                                            child.receiveShadow = false;
                                            child.originalMaterial = child.material; // Store the original material
                                        }
                                    });
                                    placeBayDoorsAndTrucks(bayDoor, outboundTruck, inboundTruck, distributionCenter);
                                    loadAndPlaceLabels();
                                    renderer.render(scene, activeCamera);
                                },
                                function (xhr) { },
                                function (error) { }
                            );
                        },
                        function (xhr) { },
                        function (error) { }
                    );
                }, function (error) { });
            }, function (error) { });
        },
        function (xhr) { },
        function (error) { }
    );
}, function (error) { });
}

async function placeBayDoorsAndTrucks(bayDoor, outboundTruck, inboundTruck, distributionCenter) {
    const numStackDoors = parseInt(document.getElementById('numStackDoors').value);
    const numStripDoors = parseInt(document.getElementById('numStripDoors').value);
    const doorWidth = 1;
```

```
const doorOffset = 30;

const doorVerticalOffset = 2;
const truckVerticalOffset = 0.1;

const boundingBox = new THREE.Box3().setFromObject(distributionCenter);
const size = boundingBox.getSize(new THREE.Vector3());

// Clear previous bay doors, trucks, and labels from the scene
bayDoors.forEach(door => scene.remove(door));
trucks.forEach(truck => scene.remove(truck));
stripLabels.forEach(label => scene.remove(label));
stackLabels.forEach(label => scene.remove(label));
bayDoors = [];
trucks = [];
stripLabels = [];
stackLabels = [];
duplicateLabels = [];

function calculateDoorPositions(numDoors, totalDoors, side) {
    const totalWidth = doorWidth * totalDoors + doorOffset * (totalDoors + 1);

    const xPosition = side === 'right' ? size.x / 2 + doorWidth / 2 : -size.x / 2 - doorWidth / 2;
    const positions = [];

    if (numDoors === 0) return positions;

    const spacing = (size.z - doorWidth * numDoors - doorOffset * 2) / (numDoors - 1);

    for (let i = 0; i < numDoors; i++) {
        const zPosition = -size.z / 2 + doorOffset + i * (doorWidth + spacing);
        positions.push({
            x: xPosition,
            z: zPosition,
        });
    }

    return positions;
}

const totalWidthStack = numStackDoors * doorWidth + (numStackDoors - 1) * doorOffset;
const totalWidthStrip = numStripDoors * doorWidth + (numStripDoors - 1) * doorOffset;

const stripDoorPositions = calculateDoorPositions(numStackDoors, totalWidthStack, 'right');
const stackDoorPositions = calculateDoorPositions(numStripDoors, totalWidthStrip, 'left');

const initialOffset = 115;
const queueSpacing = 5;

// Place trucks for strip doors
stripDoorPositions.forEach((position, index) => {
```

```
        const doorClone = bayDoor.clone();
        doorClone.rotation.y = Math.PI;
        doorClone.position.set(position.x, doorVerticalOffset, position.z);
        doorClone.castShadow = false;
        doorClone.receiveShadow = false;
        bayDoors.push(doorClone);
        scene.add(doorClone);

        const trucksForDoor = truckSchedule.filter(item => item.door === (index + 1).toString());
        trucksForDoor.forEach((scheduleItem, queueIndex) => {
            let truckClone = inboundTruck.clone();
            truckClone.rotation.y = Math.PI;
            truckClone.position.set(position.x + initialOffset + queueIndex * queueSpacing,
truckVerticalOffset, position.z);

            // Store initial position and rotation
            truckClone.userData.initialPosition = truckClone.position.clone();
            truckClone.userData.initialRotation = truckClone.rotation.clone();

            prepareTruckAnimation(truckClone, scheduleItem, position.x, position.z, 'inbound',
queueIndex);
            trucks.push(truckClone);
            scene.add(truckClone);
        });
    });

    // Place trucks for stack doors
    stackDoorPositions.forEach((position, index) => {
        const doorClone = bayDoor.clone();
        doorClone.position.set(position.x, doorVerticalOffset, position.z);
        doorClone.castShadow = false;
        doorClone.receiveShadow = false;
        bayDoors.push(doorClone);
        scene.add(doorClone);

        const trucksForDoor = truckSchedule.filter(item => item.door ===
String.fromCharCode(65 + index));
        trucksForDoor.forEach((scheduleItem, queueIndex) => {
            let truckClone = outboundTruck.clone();
            truckClone.position.set(position.x - initialOffset - queueIndex * queueSpacing,
truckVerticalOffset, position.z);

            // Store initial position and rotation
            truckClone.userData.initialPosition = truckClone.position.clone();
            truckClone.userData.initialRotation = truckClone.rotation.clone();

            prepareTruckAnimation(truckClone, scheduleItem, position.x, position.z, 'outbound',
queueIndex);
            trucks.push(truckClone);
            scene.add(truckClone);
        });
```

```
    });
}

function prepareTruckAnimation(truck, scheduleItem, originalX, originalZ, type, queueIndex) {
    const offset = 75;
    const direction = type === 'inbound' ? 1 : -1;
    const queueOffset = queueIndex * 15;
    const startX = originalX + offset * direction + queueOffset * direction;

    truck.position.x = startX;

    // Calculate times in simulation seconds
    const arrivalTime = scheduleItem.arrive * SIMULATION_SPEED_FACTOR;
    const departureTime = scheduleItem.depart * SIMULATION_SPEED_FACTOR;
    const dockDuration = (departureTime - arrivalTime); // Dock duration in simulation seconds

    const dockX = originalX + 6 * direction; // Dock position
    const dockZ = originalZ;
    const moveDuration = 4; // Move duration in simulation seconds

    truck.animation = {
        startX: startX,
        dockX: dockX,
        dockZ: dockZ,
        moveDuration: moveDuration,
        arrivalTime: arrivalTime,
        dockDuration: dockDuration,
        direction: direction,
        state: 'waiting',
        moveStartTime: arrivalTime - moveDuration, // Adjusted for simulation seconds
        returnStartTime: departureTime,
        currentTime: 0, // Initialize current time
        stateStartTime: 0 // Initialize state start time
    };
}

let popupShown = false; // Flag to prevent multiple popups

let frameCount = 0; // Counter for frames
const frameThrottle = 10; // Update every 2 frames

function animateTrucks() {
    frameCount++; // Increment the frame counter

    // Only update every few frames
    if (frameCount % frameThrottle !== 0) {
        animationFrameId = requestAnimationFrame(animateTrucks); // Request next frame
        return; // Exit early
    }
    let allTrucksFinished = true;
    const elapsedTime = clock.getElapsedTime(); // Get the elapsed time in simulation seconds
```

```
        trucks.forEach(truck => {
            const anim = truck.animation;

            switch (anim.state) {
                case 'waiting':
                    if (elapsedTime >= anim.moveStartTime) {
                        anim.state = 'movingToDock';
                        anim.stateStartTime = elapsedTime; // Set start time for the new state
                    }
                    allTrucksFinished = false;
                    break;

                case 'movingToDock':
                    const timeSinceMoveStart = elapsedTime - anim.moveStartTime;
                    const moveProgressToDock = timeSinceMoveStart / anim.moveDuration;

                    truck.position.x = THREE.MathUtils.lerp(anim.startX, anim.dockX,
THREE.MathUtils.clamp(moveProgressToDock, 0, 1));

                    if (moveProgressToDock < 1) {
                        allTrucksFinished = false;
                    } else {
                        truck.position.x = anim.dockX; // Snap to the dock position
                        anim.state = 'docked';
                        anim.stateStartTime = elapsedTime; // Set start time for the new state
                        startFlashing(truck); // Start the flashing effect
                    }
                    break;

                case 'docked':
                    if (elapsedTime >= anim.returnStartTime) {
                        anim.state = 'returning';
                        anim.stateStartTime = elapsedTime; // Set start time for the new state
                        stopFlashing(truck); // Stop the flashing effect
                    }
                    allTrucksFinished = false;
                    break;

                case 'returning':
                    const timeSinceReturnStart = elapsedTime - anim.returnStartTime;
                    const moveProgressReturn = timeSinceReturnStart / anim.moveDuration;

                    truck.position.x = THREE.MathUtils.lerp(anim.dockX, anim.startX + anim.direction *
96, THREE.MathUtils.clamp(moveProgressReturn, 0, 1));
                    truck.position.z = THREE.MathUtils.lerp(anim.dockZ, anim.dockZ + anim.direction *
-100, THREE.MathUtils.clamp(moveProgressReturn, 0, 1));

                    let rotation1 = 0;
                    let rotation2 = 0;
```

```
                if (anim.direction == 1) {
                    rotation1 = Math.PI;
                    rotation2 = (3 * Math.PI) / 2;
                } else {
                    rotation1 = 0;
                    rotation2 = Math.PI / 2;
                }

                truck.rotation.y = THREE.MathUtils.lerp(rotation1, rotation2,
THREE.MathUtils.clamp(moveProgressReturn, 0, 1));

                if (moveProgressReturn < 1) {
                    allTrucksFinished = false;
                } else {
                    anim.state = 'finished';
                    trucksDone++;

                }
                break;

            case 'finished':
                break;
        }

        if (anim.state !== 'finished') {
            allTrucksFinished = false;
        }
    }
    );

    if (allTrucksFinished && trucks.length !== 0) {
        cancelAnimationFrame(animationFrameId); // Stop the animation
        if (!popupShown) {
            showSimulationEndPopup(); // Show popup
            popupShown = true; // Set flag to true
        }
    } else {
        animationFrameId = requestAnimationFrame(animateTrucks);
    }
}

// Function to show the simulation end popup
async function showSimulationEndPopup() {
    const maxDepartureTime = Math.max(...truckSchedule.map(truck => truck.depart));
    const totalRealTimeMinutes = 20 * (maxDepartureTime - 2);

    // Update the popup content
    document.getElementById('totalSimSeconds').textContent = maxDepartureTime;
    document.getElementById('totalRealTimeMinutes').textContent = totalRealTimeMinutes;

    // Display the popup
```

```
        const simulationEndPopup = document.getElementById('simulationEndPopup');
        simulationEndPopup.style.display = 'block';

        // Attach close button functionality
        const closePopupButton = document.getElementById('closeSimPopup');
        closePopupButton.onclick = function () {
            simulationEndPopup.style.display = 'none';
        };

        // Close the popup if the user clicks outside the modal content
        window.onclick = function (event) {
            if (event.target === simulationEndPopup) {
                simulationEndPopup.style.display = 'none';
            }
        };
    }

    function startFlashing(truck) {
        // Ensure the truck object exists and has children
        if (!truck) return;

        truck.flashingInterval = setInterval(() => {
            truck.traverse(function (child) {
                // Check if the child is a Mesh and has a valid material
                if (child instanceof THREE.Mesh && child.material && typeof child.material.clone ===
'function') {
                    // Ensure the material and originalMaterial are properly initialized
                    if (!child.originalMaterial) {
                        child.originalMaterial = child.material.clone(); // Clone the original material to
ensure it is properly initialized
                    }

                    // Toggle between the yellow material and the original material
                    child.material = (child.material === yellowLambertMaterial) ? child.originalMaterial :
yellowLambertMaterial;
                }
            });
        }, 500); // Toggle every 500ms (0.5 seconds)
    }

    function stopFlashing(truck) {
        // Ensure the truck object exists and is valid
        if (!truck || !truck.flashingInterval) return;

        clearInterval(truck.flashingInterval); // Stop the interval
        truck.traverse(function (child) {
            // Check if the child is a Mesh and has a valid material
            if (child instanceof THREE.Mesh && child.material && child.originalMaterial) {
                // Reset to the original material
                child.material = child.originalMaterial;
            }
```

```
    });
}

function loadGround() {

    let groundFilePath = '/static/ground2.obj';
    let groundMtlPath = '/static/ground2.mtl';

    mtlLoader.load(groundMtlPath, function (materials) {
        materials.preload();
        objLoader.setMaterials(materials);
        objLoader.load(
            groundFilePath,
            function (object) {
                object.traverse(function (child) {
                    if (child instanceof THREE.Mesh) {
                        child.castShadow = false;
                        child.receiveShadow = true;
                        if (child.material.map) {
                            child.material.map.needsUpdate = true; // Force texture update
                        }
                    }
                });

                // Add the new ground object to the scene
                scene.add(object);
                ground = object;
            },
            function (xhr) {
                console.log((xhr.loaded / xhr.total * 100) + '% loaded');
            },
            function (error) {
                console.error('Error loading ground', error);
            }
        );
    }, function (error) {
        console.error('Error loading ground materials', error);
    });
}

function loadAndPlaceLabels() {

    const numStackDoorsInput = document.getElementById('numStackDoors');
    const numStripDoorsInput = document.getElementById('numStripDoors');

    if (!numStackDoorsInput || !numStripDoorsInput) {
        return;
    }

    const numStackDoors = parseInt(numStackDoorsInput.value, 10);
    const numStripDoors = parseInt(numStripDoorsInput.value, 10);
```

```
function loadLabel(filePath, position, labelsArray, rotationY, isStackDoor) {
    objLoader.load(
        filePath,
        function (label) {
            label.traverse(function (child) {
                if (child instanceof THREE.Mesh) {
                    child.material = isStackDoor ? stackLabelMaterial : stripLabelMaterial;
                    child.castShadow = false;
                    child.receiveShadow = false;
                }
            });

            label.rotation.y = rotationY; // Rotate label
            label.scale.set(1.5, 1.5, 1.5); // Increase the size of the label by 50%
            label.position.copy(position);
            scene.add(label);
            labelsArray.push(label);

            // Duplicate the label and place it on top of the distribution center
            const duplicateLabel = label.clone();
            duplicateLabel.position.copy(position); // Copy original position
            duplicateLabel.position.y += 3; // Shift up

            // Apply the additional rotation for the duplicate label
            if (isStackDoor) {
                duplicateLabel.position.x += -3; // Shift closer to distribution center
                duplicateLabel.rotation.order = 'ZYX'; // Adjust rotation order
                duplicateLabel.rotation.z = Math.PI / 2.5;
            } else {
                duplicateLabel.position.x += 3; // Shift closer to distribution center
                duplicateLabel.rotation.order = 'ZYX'; // Adjust rotation order
                duplicateLabel.rotation.x = -Math.PI / 2.5;
            }
            scene.add(duplicateLabel);
            duplicateLabels.push(duplicateLabel); // Store the duplicate label
        },
        function (xhr) {
            console.log((xhr.loaded / xhr.total * 100) + '% loaded');
        },
        function (error) {
            console.error('Error loading label', error);
        }
    );
}

// Clear previous labels and duplicate labels
stackLabels.forEach(label => scene.remove(label));
stackLabels = [];
stripLabels.forEach(label => scene.remove(label));
stripLabels = [];
```

```
duplicateLabels.forEach(label => scene.remove(label));
duplicateLabels = [];

// Load and place labels for stack doors (numerically labeled)
for (let i = 0; i < numStackDoors; i++) {
    const stackDoorIndex = i;

    if (stackDoorIndex < bayDoors.length && stackDoorIndex >= 0) {
        const stackDoor = bayDoors[stackDoorIndex];
        if (stackDoor && stackDoor.position) {
            const stackDoorPosition = stackDoor.position.clone();
            stackDoorPosition.y += 2; // Adjust Y position if needed
            stackDoorPosition.x -= 0.7; // Adjust X position to move closer to distribution center

            const number = i + 1;
            let labelFiles = [];

            if (number > 99) {
                const hundredsDigit = Math.floor(number / 100);
                const tensDigit = Math.floor((number % 100) / 10);
                const onesDigit = number % 10;

                labelFiles = [
                    { filePath: `/static/${hundredsDigit}.obj`, offset: 0.8 },
                    { filePath: `/static/${tensDigit}.obj`, offset: 0 },
                    { filePath: `/static/${onesDigit}.obj`, offset: -1 }
                ];
            } else if (number > 9) {
                const tensDigit = Math.floor(number / 10);
                const onesDigit = number % 10;

                labelFiles = [
                    { filePath: `/static/${tensDigit}.obj`, offset: 0.5 },
                    { filePath: `/static/${onesDigit}.obj`, offset: -0.5 }
                ];
            } else {
                labelFiles = [{ filePath: `/static/${number}.obj`, offset: 0 }]; // Single digit label
            }

            labelFiles.forEach(({ filePath, offset }) => {
                const labelPosition = stackDoorPosition.clone();
                labelPosition.z += offset;
                loadLabel(filePath, labelPosition, stackLabels, Math.PI / 2, true);
            });
        } else {
            console.error(`Invalid stack door position for index ${stackDoorIndex}`);
        }
    } else {
        console.error(`Stack door index ${stackDoorIndex} is out of bounds`);
    }
}
```

```
        // Load and place labels for strip doors (alphabetically labeled with 180 degree rotation)
        const alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

        function getExcelLabel(index) {
            let label = '';
            let n = index + 1;

            while (n > 0) {
                const remainder = (n - 1) % 26;
                label = alphabet.charAt(remainder) + label;
                n = Math.floor((n - 1) / 26);
            }

            return label;
        }

        for (let i = 0; i < numStripDoors; i++) {
            const stripDoorIndex = i;

            if (stripDoorIndex < bayDoors.length && stripDoorIndex >= 0) {
                const stripDoor = bayDoors[numStackDoors + stripDoorIndex]; // Adjust index for strip
doors
                if (stripDoor && stripDoor.position) {
                    const stripDoorPosition = stripDoor.position.clone();
                    stripDoorPosition.y += 2; // Adjust Y position if needed
                    stripDoorPosition.x += 0.7; // Adjust X position to move closer to distribution center

                    const label = getExcelLabel(i);
                    const labelFiles = label.split('').map((char, idx) => ({
                        filePath: `/static/${char}.obj`,
                        offset: (idx - (label.length - 1) / 2) * 1.4  // Increase the offset to spread out the
labels more
                    }));

                    labelFiles.forEach(({ filePath, offset }) => {
                        const labelPosition = stripDoorPosition.clone();
                        labelPosition.z += offset;
                        loadLabel(filePath, labelPosition, stripLabels, -Math.PI / 2, false); // Rotate label
180 degrees
                    });
                } else {
                    console.error(`Invalid strip door position for index ${stripDoorIndex}`);
                }
            } else {
                console.error(`Strip door index ${stripDoorIndex} is out of bounds`);
            }
        }
    }

    document.getElementById('userInputForm').addEventListener('submit', function (event) {
```

```
    event.preventDefault();

    const numStripDoors = parseInt(document.getElementById('numStripDoors').value);
    const numStackDoors = parseInt(document.getElementById('numStackDoors').value);
    const numInboundTrucks = parseInt(document.getElementById('numInboundTrucks').value);
    const numOutboundTrucks = parseInt(document.getElementById('numOutboundTrucks').value);

    // Construct the scenario key based on user input
    const scenarioKey = `scenario_${numStackDoors}${numStripDoors}${numInboundTrucks}${numOutboundTrucks}`;

    // Check if the scenario exists in the loaded scenarios
    if (scenarios[scenarioKey]) {
        truckSchedule = scenarios[scenarioKey];
        console.log(`Loaded ${scenarioKey}`);
    } else {
        console.error(`Scenario ${scenarioKey} does not exist.`);
        return;
    }

    // Find the last truck departure time from the truck schedule
    const lastTruckDepartureTime = Math.max(...truckSchedule.map(truck => truck.depart));

    // Update the last truck departure time display
    document.getElementById('lastTruckDepartureTime').textContent = lastTruckDepartureTime;

    // Reset popupShown flag before a new animation starts, but ensure it doesn't trigger prematurely
    popupShown = false;

    // Clear the previous popup if visible
    const popup = document.getElementById('simulationEndPopup');
    popup.style.display = 'none';

    if (distributionCenter) {
        // Reset scene
        scene.remove(distributionCenter);
        bayDoors.forEach(door => scene.remove(door));
        trucks.forEach(truck => scene.remove(truck));
        stripLabels.forEach(label => scene.remove(label));
        stackLabels.forEach(label => scene.remove(label));
        duplicateLabels.forEach(label => scene.remove(label));
        bayDoors = [];
        trucks = [];
        stripLabels = [];
        stackLabels = [];
        duplicateLabels = [];
    }
```

```
    loadDistributionCenter();

});

let animationStarted = false;
let animationPaused = false;

function render() {
    controlsPerspective.update();
    controlsTopView.update();
    renderer.render(scene, activeCamera);

    if (animationStarted && !animationPaused) {
        animateTrucks(); // Directly call the animation loop
    }

    requestAnimationFrame(render);
}

function startCounter() {
    // Find the maximum departure time from the truck schedule
    const maxDepartureTime = Math.max(...truckSchedule.map(truck => truck.depart));

    // Get the circle element
    const counterCircle = document.getElementById('counterCircle');
    let counterValue = 0;

    // Define the interval to update the counter
    const intervalTime = 1000 * SIMULATION_SPEED_FACTOR;

    // Start the counter and store the interval ID
    counterInterval = setInterval(() => {
        counterValue++;
        counterCircle.textContent = counterValue;

        // Stop the counter when it reaches the maximum departure time
        if (counterValue >= maxDepartureTime) {
            clearInterval(counterInterval);
        }
    }, intervalTime);
}

document.getElementById('beginAnimation').addEventListener('click', function () {
    clock.start();
    animationStarted = true;
    animationPaused = false;

    popupShown = false; // Reset popup shown flag on new animation

    // Disable input elements and hover effects when animation starts
```

```
        setInputElementsDisabled(true);
        setHoverEffectsDisabled(true);

        // Reset all trucks to start the animation from the beginning
        trucks.forEach(truck => {
            truck.animation.state = 'waiting'; // Start all trucks in the 'waiting' state
            truck.animation.currentTime = 0; // Reset current animation time
            truck.animation.stateStartTime = 0; // Reset state start time
        });

        startCounter(); // Start the counter animation
        render(); // Start the render loop

    });

    document.getElementById('resetScene').addEventListener('click', function () {
        // Pause and reset animation states
        animationPaused = true;
        animationStarted = false;

        // Stop and reset the clock
        clock.elapsedTime = 0;
        clock.stop(); // Stop the clock during reset

        // Reset the popup shown flag to allow the popup to appear again
        popupShown = false;

        // Re-enable input elements and hover effects when reset is clicked
        setInputElementsDisabled(false);
        setHoverEffectsDisabled(false);

        // Reset counter
        const counterCircle = document.getElementById('counterCircle');
        counterCircle.textContent = '0'; // Reset the counter display
        clearInterval(counterInterval); // Clear the interval to stop the counter

        // Hide any visible popup
        const popup = document.getElementById('simulationEndPopup');
        popup.style.display = 'none';

        // // Reset truck positions and states
        trucks.forEach(truck => {
            truck.animation.state = 'waiting';  // Reset state to waiting
            truck.animation.currentTime = 0; // Reset the animation time
            truck.animation.stateStartTime = 0; // Reset the start time for the state
            truck.position.set(truck.animation.startX,   truck.position.y,   truck.animation.dockZ);      //
Reset position
            truck.rotation.y = truck.animation.direction === 1 ? Math.PI : 0; // Reset rotation
            stopFlashing(truck); // Stop any flashing effect
        });
```

```
    // Re-render the scene to reflect changes
    renderer.render(scene, activeCamera);
});

// Get the modal and the button that opens it
const instructionsModal = document.getElementById('instructionsModal');
const helpButton = document.getElementById('helpButton');
const closeModal = document.getElementById('closeModal');

// Open the modal when the user clicks the question mark
helpButton.addEventListener('click', function () {
    instructionsModal.style.display = 'block';
});

// Close the modal when the user clicks the close button
closeModal.addEventListener('click', function () {
    instructionsModal.style.display = 'none';
});

// Prevent closing when clicking inside the modal content
document.querySelector('.modal-content').addEventListener('click', function (event) {
    event.stopPropagation();
});

// Close the modal when the user clicks outside the modal content
window.addEventListener('click', function (event) {
    if (event.target === instructionsModal) {
        instructionsModal.style.display = 'none';
    }
});

function setInputElementsDisabled(disabled) {
    // Get all input elements
    const numStripDoors = document.getElementById('numStripDoors');
    const numStackDoors = document.getElementById('numStackDoors');
    const numInboundTrucks = document.getElementById('numInboundTrucks');
    const numOutboundTrucks = document.getElementById('numOutboundTrucks');
    const submitButton = document.getElementById('updateButton');
    const playButton = document.getElementById('beginAnimation');

    // Set the disabled property for each element
    numStripDoors.disabled = disabled;
    numStackDoors.disabled = disabled;
    numInboundTrucks.disabled = disabled;
    numOutboundTrucks.disabled = disabled;
    submitButton.disabled = disabled;
    playButton.disabled = disabled;

    // Apply a visual cue (grey out) by changing the opacity
    const inputElements = [numStripDoors, numStackDoors, numInboundTrucks,
numOutboundTrucks, submitButton, playButton];
```

```
        inputElements.forEach(element => {
            element.style.opacity = disabled ? '0.5' : '1';
            element.style.cursor = disabled ? 'not-allowed' : 'pointer';
        });
    }

    function setHoverEffectsDisabled(disabled) {
        // Include both dropdowns and their container elements
        const elementsToDisableHover = [
            document.getElementById('numStripDoors'),
            document.getElementById('numStackDoors'),
            document.getElementById('numInboundTrucks'),
            document.getElementById('numOutboundTrucks'),
            document.getElementById('updateButton'),
            document.getElementById('beginAnimation'),
            document.querySelector('.input-rectangle.dark-blue'),  // Container for Strip Doors dropdown
            document.querySelector('.input-rectangle.dark-teal'),  // Container for Stack Doors dropdown
            document.querySelector('.input-rectangle.dark-blue:nth-child(3)'),  // Container for Inbound Trucks dropdown
            document.querySelector('.input-rectangle.dark-teal:nth-child(4)')  // Container for Outbound Trucks dropdown
        ];

        elementsToDisableHover.forEach(element => {
            if (disabled) {
                element.classList.add('no-hover');
            } else {
                element.classList.remove('no-hover');
            }
        });
    }

    loadDistributionCenter();
    render();
}

main();
```

Appendix F: GUI Program

```
{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>3D Distribution Center</title>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
    <script src="{% static 'three.js-r100/examples/js/loaders/OBJLoader.js' %}"></script>
    <script src="{% static 'three.js-r100/examples/js/loaders/MTLLoader.js' %}"></script>
    <script src="{% static 'three.js-r100/examples/js/controls/OrbitControls.js' %}"></script>

    <style>
        body {
            margin: 0;
            overflow: hidden;
        }

        canvas {
            display: block;
        }

        /* New background circle style */
        .counter-circle-background {
            position: absolute;
            top: 25px;
            /* Slightly adjusted to position behind the counter-circle */
            left: 25px;
            /* Slightly adjusted to position behind the counter-circle */
            width: 140px;
            /* Slightly larger width */
            height: 140px;
            /* Slightly larger height */
            border-radius: 50%;
            background-color: #323232;
            /* Dark grey color */
            z-index: 0;
            /* Ensure it is behind the counter-circle */
        }

        .counter-circle {
            position: absolute;
            top: 30px;
            left: 30px;
            width: 130px;
            height: 130px;
            border-radius: 50%;
```

```
    background-color: #d8dfe7;
    color: rgb(0, 0, 0);
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 45px;
    font-weight: bold;
    z-index: 1;
    /* Ensure it is above the background circle */
}

.user-input-container {
    position: absolute;
    top: 10px;
    right: 10px;
    background-color: rgba(255, 254, 211, 0.536);
    /* Semi-transparent background */
    padding: 10px;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.row {
    display: flex;
    justify-content: flex-start;
    gap: 10px;
}

.input-rectangle {
    display: flex;
    flex-direction: row;
    align-items: flex-start;
    justify-content: center;
    padding: 10px;
    border-radius: 10px;
    color: white;
    text-align: left;
}

.input-rectangle label {
    color: white;
    margin-bottom: 5px;
    font-size: 18px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
    /* Adds shadow to make text more visible */
}

select {
    font-size: 20px;
```

```
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
    /* Adds shadow to make text more visible */
    padding: 10px;
    width: 55%;
    border-radius: 10px;
    color: white;
    background-color: #336699;
    border: none;
    cursor: pointer;

}

select:hover {
    border: 1px solid #fff;
}

.dark-blue {
    background-color: #003366;
}

.light-blue {
    background-color: #336699;
    border: none;
}

.dark-teal {
    background-color: #0a6262;
}

.light-teal {
    background-color: #339999;
    border: none;
}

.button-container {
    display: flex;
    align-items: center;
    justify-content: flex-end;
    gap: 25px;
    margin-right: 40px;
}

.play-button,
.reset-button {
    padding: 10px 40px;
    font-size: 40px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
    /* Adds shadow to make text more visible */
    border: none;
    border-radius: 10px;
    color: white;
```

```
    cursor: pointer;
}

.play-button {
    background-color: #28a745;
}

.reset-button {
    background-color: #dc3545;
}

.play-button:hover {
    background-color: #135122;
}

.reset-button:hover {
    background-color: #841f29;
}

.question-circle {
    width: 65px;
    height: 65px;
    border-radius: 50%;
    background-color: #6b6b6b;
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 30px;
    font-weight: bold;
    text-shadow: 2px 2px 2px rgb(0, 0, 0);
    /* Adds shadow to make text more visible */
    color: white;
    cursor: pointer;
}

.question-circle:hover {
    background-color: #404040;
}

/* Update button styling */
#updateButton {
    background-color: purple;
    color: white;
    padding: 10px 20px;
    font-size: 25px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
    /* Adds shadow to make text more visible */

    border: none;
    border-radius: 10px;
    cursor: pointer;
```

```
}

#updateButton:hover {
    background-color: #4B0082;
    /* Dark purple hover color */
}

.camera-label {
    font-size: 35px;
    color: white;
    margin-right: 10px;
    align-self: center;
    text-shadow: 2px 2px 2px rgba(0, 0, 0);
    /* Adds shadow to make text more visible */
}

.camera-button {
    padding: 10px 20px;
    font-size: 25px;
    border: none;
    border-radius: 10px;
    color: white;
    cursor: pointer;
    margin-right: 110px;
    background-color: #313131;
}

#birdEyeView {
    margin-right: -20px;
    /* Adjusts space between "Bird's Eye" button and the next button */
}

.camera-button:hover {
    background-color: #4e4e4e;
    /* Darker color on hover */
}

/* Modal Styles */
.modal {
    display: none;
    /* Hidden by default */
    position: fixed;
    /* Stay in place */
    z-index: 1000;
    /* Sit on top */
    left: 0;
    top: 0;
    width: 100%;
    /* Full width */
    height: 100%;
    /* Full height */
```

```
    background-color: rgba(0, 0, 0, 0.5);
    /* Black background with opacity */
}

.modal-content {
    background-color: #fefefe;
    margin: 15% auto;
    /* 15% from the top and centered */
    padding: 20px;
    border: 1px solid #888;
    width: 100%;
    /* Could be more or less, depending on screen size */
    max-width: 700px;
    /* Maximum width */
    border-radius: 10px;
    /* Rounded corners */
    z-index: 1001; /* Higher than the overlay */

}

#closeSimPopup {
    margin-top: 20px;
    padding: 10px 20px;
    background-color: #af4c4c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    z-index: 1002; /* Ensure button is clickable */
    text-align: center;

}

#closeSimPopup:hover {
    background-color: #471313;
}

.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}

.close:hover,
.close:focus {
    color: #000;
    text-decoration: none;
    cursor: pointer;
}
```

```
.no-hover {
    pointer-events: none;
    /* Disable all mouse events */
    opacity: 0.5;
    /* Optional: grey out the element */
}

/* Specific styles to disable hover effects */
.input-rectangle.no-hover:hover,
select.no-hover:hover,
button.no-hover:hover {
    background-color: inherit;
    /* Prevent background color change on hover */
    cursor: not-allowed;
    /* Change cursor to indicate that interaction is disabled */
}

/* Styles for the new elements */
.simulation-label {
    position: absolute;
    top: 0px;
    left: 20px;
    font-size: 24px;
    font-weight: bold;
    color: white;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
}

.last-truck-departure-container {
    position: absolute;
    top: 160px;
    left: 20px;
    display: flex;
    flex-direction: row;
    align-items: center;
    font-size: 22px;
    color: white;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
}

.last-truck-departure-label {
    margin-right: 10px;
}

.last-truck-departure-value {
    background-color: #d8dfe7;
    padding: 5px 10px;
    border-radius: 5px;
    color: black;
    font-weight: bold;
}
```

```
    </style>

</head>

<body>
    <!-- Add a new div for the background circle -->
    <div class="counter-circle-background"></div>
    <div id="counterCircle" class="counter-circle">0</div>

    <!-- Add the label above the clock for Simulation Seconds -->
    <div class="simulation-label">Simulation Seconds</div>

    <!-- Add label and value box for Last Truck Departure Time -->
    <div class="last-truck-departure-container">
        <span class="last-truck-departure-label">Last Truck Departure Time:</span>
        <span id="lastTruckDepartureTime" class="last-truck-departure-value">N/A</span>
    </div>

    <div class="user-input-container">
        <form id="userInputForm">
            <!-- First Row: Input Controls -->
            <div class="row">
                <div class="input-rectangle dark-blue">
                    <label for="numStackDoors">Select Number of Strip Doors:</label>
                    <select id="numStackDoors" class="light-blue">
                        <option value="2">2</option>
                        <option value="4">4</option>
                    </select>
                </div>

                <div class="input-rectangle dark-teal">
                    <label for="numStripDoors">Select Number of Stack Doors:</label>
                    <select id="numStripDoors" class="light-teal">
                        <option value="2">2</option>
                        <option value="4">4</option>
                    </select>
                </div>

                <div class="input-rectangle dark-blue">
                    <label for="numInboundTrucks">Select Number of Inbound Trucks:</label>
                    <select id="numInboundTrucks" class="light-blue">
                        <option value="7">7</option>
                    </select>
                </div>

                <div class="input-rectangle dark-teal">
                    <label for="numOutboundTrucks">Select Number of Outbound Trucks:</label>
                    <select id="numOutboundTrucks" class="light-teal">
                        <option value="5">5</option>
                    </select>
                </div>
```

```
            <div class="input-rectangle dark-blue">
                <button type="submit" id="updateButton">SUBMIT</button>
            </div>
        </div>
    </form>

    <!-- Second Row: Control Buttons -->
    <div class="row button-container">
        <div class="camera-label">Select Camera View:</div>
        <button class="camera-button" id="birdEyeView">Bird's Eye</button>
        <button class="camera-button" id="freeRoamView">Free Roam</button>
        <button class="play-button" id="beginAnimation">PLAY</button>
        <button class="reset-button" id="resetScene">RESET</button>
        <div class="question-circle" id="helpButton">?</div>

        <!-- Instructions Modal -->
        <div id="instructionsModal" class="modal">
            <div class="modal-content">
                <span class="close" id="closeModal">&times;</span>
                <h2>Help</h2>
                <p>Information about the animation:</p>
                <ul>
                    <li>This animation provides a visual representation of the results obtained from a
mixed integer
                        programming model that schedules inbound and outbound trucks so that delays
from
                        predefined deadlines is minimized.</li>
                    <li>Inbound trucks are blue and outbound trucks are green. Their arrival and
departure
                        sequences are determined by the math model.</li>
                    <li>When a docked truck is flashing, it is being unloaded or loaded.</li>
                    <li>The clock in the upper left hand corner tracks time that is scaled from the actual
time;
                        hence,
                        it is called “simulation seconds.” The clock will stop when the last outbound truck
departs
                        and the time this occurs will be displayed both in computer seconds on the clock
and scaled
                        up to real time in a pop-up box.</li>
                </ul>
                <p>Instructions for use:</p>
                <ul>
                    <li>Use the "Free Roam" and "Bird's Eye" buttons to switch between different
camera views.</li>
                    <li>Click the "PLAY" button to start the animation and see the trucks move.</li>
                    <li>The CLOCK in the top left corner will beginn counting up once the animation
begins.</li>
                    <li>Use the "RESET" button to reset the scene.</li>
                    <li>Select the number of strip and stack doors, inbound trucks, and outbound
trucks using the
```

```
                dropdown menus.</li>
            <li>Click "SUBMIT" to apply your selected settings.</li>
        </ul>
        <p>About:</p>
        <ul>
            <li>This study is based on a study supported by the Center for Connected Multimodal Mobility
                (C2M2) (USDOT Tier 1 University Transportation Center) Grant headquartered at Clemson
                University, Clemson, South Carolina, USA. Any opinions, findings, and conclusions or
                recommendations expressed in this material are those of the author(s) and do not
                necessarily reflect the views of the Center for Connected Multimodal Mobility (C2M2), and
                the U.S. Government assumes no liability for the contents or use thereof.</li>
            <li>Copyright © 2024. Clemson University • Clemson, South Carolina 29634, All rights reserved.
            </li>
            <li>For more information on the animation and the math programming model, see the Center for
                Connected Multimodal Mobility website at https://cecas.clemson.edu/C2M2/.</li>
        </ul>
    </div>
</div>

<!-- Simulation End Popup -->
<div id="simulationEndPopup" class="modal">
    <div class="modal-content">
        <h2>Simulation Complete</h2>
        <p>Total Simulation Seconds: <span id="totalSimSeconds">0</span></p>
        <p>Total Scenario Time in Real Time Minutes: <span id="totalRealTimeMinutes">0</span></p>
        <button id="closeSimPopup">Close</button>
    </div>
</div>

<canvas id="c" width="1920" height="1080"></canvas>

<!-- Load scenarios and 3d_cube.js scripts directly -->
<script src="{% static 'js/scenarios.js' %}"></script>
<script src="{% static 'js/3d_cube.js' %}"></script>
</body>


</html>
```