(MASS) TIMBER

STRUCTURALLY OPTIMIZED TIMBER BUILDINGS

KRISTEN STROBEL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF:

MASTER OF ARCHITECTURE

UNIVERSITY OF WASHINGTON

2016

COMMITTEE:

KATHRINA SIMONEN
ROBERT CORSER
JEFFREY BERMAN

PROGRAM AUTHORIZED TO OFFER DEGREE:

ARCHITECTURE

UNIVERSITY OF WASHINGTON

ABSTRACT

(MASS) TIMBER: STRUCTURALLY OPTIMIZED TIMBER BUILDINGS

KRISTEN STROBEL

CHAIR OF THE SUPERVISORY COMMITTEE:
KATHRINA SIMONEN
DEPARTMENT OF ARCHITECTURE

This thesis explores the challenges and potential of mass timber as a paradigm shifting technology for the building industry through the application of parametric modeling technology to the design of office buildings. By testing building configurations in three zoning envelopes—low-rise suburban, mid-rise urban, and high-rise urban—optimization strategies for mass timber office buildings were developed. Facades and floor slabs were identified as the primary contributors to building cost and environmental impacts and therefore the easiest targets for optimization. The primary method for optimizing facades is replacing curtain wall with solid cross laminated timber (CLT) walls, this method runs counter to developer driven standards of fully glazed facades making short term adoption of this strategy unlikely without major shifts in building developer and owner expectations. Slabs and floor systems can be optimized through the implementation of novel solutions that take advantage of composite action between glulam, CLT, and concrete elements. Additionally, zoning height incentives could be used to make timber construction competitive with steel and concrete systems despite increased floor-to-floor heights. Finally, future research opportunities and needs, both architectural and technical, are identified.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF TABLES

INTRODUCTION

PROBLEM STATEMENT

As of the year 2012, there were over 16 billion square feet of office space in the United States accounting for approximately 18% of all commercial space. Of the commercial building stock 20% had been constructed in the past decade.[1] Extrapolating from this data, there will be approximately four billion new square feet of office space constructed in the next decade. If built following the status quo, these buildings will contribute 2.3 billion metric tons of greenhouse gas emissions (CO_2e) to the environment due to construction and operations over the course of a 50-year lifespan.[2]

Figure 1. New office space constructed in the next decade in the US will be enough to replace all office space in downtown Seattle nearly 100 times.[3]

Greenhouse gas emissions are the primary contributor to climate change, with studies showing that "atmospheric concentrations are...unprecedented in at least the last 800,000 years".[4] Studies have shown that "substantial emissions reductions over the next few decades can reduce climate risks in the 21st century and beyond, increase prospects for effective adaptation, reduce the costs and challenges of mitigation in the longer term and contribute to climate-resilient pathways for sustainable development".[5] Seven percent of all energy globally is used by buildings in the United States; therefore, the construction of high efficiency buildings represents a significant opportunity for diverting carbon from the environment.[6]

One way to reduce the carbon emissions associated with buildings is to use materials with lower carbon intensity, such as wood. Wood sequesters carbon from the environment and is renewable when harvested from sustainably managed forests. Research has shown that transitioning from steel and concrete construction to timber construction could reduce global carbon emissions by 14 to 31 percent.[7] Among many scenarios, optimized timber construction is the most promising using only 38% of available sustainable forest resources to offset steel and concrete construction. Currently there are significant research and construction efforts to realize tall mass-timber buildings. Examples in North America include the newly constructed Wood Innovation and Design Center in Prince George, BC,[8] the Bullitt Center in Seattle, WA,[9] as well as research proposals by leading architects and engineers including: Michael Green,[10] SOM,[11] and Mahlum.[12]

Figure 2. Timber construction could reduce global carbon emissions by 14 to 31%

While these projects represent a significant investment in bringing to market many years of research, there still exist significant opportunities to explore wood structures to better understand how they are different from more traditional construction methods and what structural forms lead to the greatest efficiencies.

PROJECT RATIONALE

Studies project that nearly 90% of the US population will live in urban areas by the year 2050.[13] In response to this influx of population, cities will need to grow increasingly dense. Towers—traditionally one of the most materially intensive forms of construction—rely on robust steel and concrete structural systems to support their height.[14] Timber structures pose a promising alternative, supporting density at a lower carbon footprint.[15] The majority of existing timber structures are located in Northern Europe. These buildings are primarily multi-family housing and due to the design and construction methods feature a large amount of structural redundancy. As timber construction is adopted in more locations, it is crucial that it becomes more optimized in order to reduce its costs and enable it to more effectively compete with steel and concrete structures.

Historically, building optimization has occurred through trial and error with each new building representing a step forward in engineering and architectural knowledge. Emerging digital technologies allow designers

to quickly test a multitude of configurations during design to allow each building to consider a greater range of design alternatives. Thereby advancing the state of practice more quickly.

This project tests both opportunities for mass timber construction and iterative parametric design to develop optimization guidelines for mass timber office buildings in Seattle.

THESIS OVERVIEW

A parametric algorithm was developed in Grasshopper for Rhino that allows for the rapid comparison of many different structural configurations. Given a series of user inputs, the algorithm generates 3-D geometry, sizes all pertinent structural members, and calculates output material quantities. Based on the execution of the algorithm for a range of inputs, general optimization guidelines for timber buildings are developed.

Results from the analysis show that the most effective optimization strategies are those that target material usage in the facade and the floor slabs. In analyzing test cases, these two assemblies form approximately 75% of the total impact of the building.

For the building facade, the primary challenge is balancing daylighting, thermal performance, and the layout of CLT shear walls. Analysis shows that moving away from the existing model for office building construction, a completely transparent facade, to a model where CLT shear walls are located along the facade has myriad benefits. Buildings with increased opacity to the facade cost less, can be designed to meet daylighting demands, have improved thermal performance, and free up the structural layout of shear walls. The true challenge to this model is not technical challenges but instead the willingness of building owners and developers to accept buildings that are not fully glazed.

The optimization of floor systems relies on two primary factors, decreasing the quantity of material in the slabs and reducing the overall depth of the floor system. Comparison of timber systems to steel systems found that at best timber systems have equal weight to steel and are twice as deep. Composite floor systems have a moderate effect, reducing slab depth and beam depth by about 20%. To allow timber to compete effectively with steel and concrete structures, technologies to reduce floor depth will need to be developed, or zoning incentives in the form of increased height will need to be provided for timber construction.

While this thesis provides an overview of key issues, it is also limited and leaves significant room for further research. Key aspects for future implementation include in depth studies of floor layout to improve system efficiency, refinement of considered load cases, continued architectural application of key findings, integration of thermal and daylighting analyses, further research into cost impacts of designing timber members for charring, and development of novel structural systems to improve floor system efficiency.

REFERENCES

[1] US Energy Information Administration. (2014). *2012 Commercial Building Energy Consumption Survey* [Data file]. Retrieved from http://www.eia.gov/consumption/commercial/data/2012/

[2] US Department of Energy. (2011). *Buildings Energy Data Book* [Data file]. Retrieved from: http:/buildingsdatabook.eren.doe.gov/default.aspx

[3] Metropolitan Improvement District and Downtown Seattle Association. (2015). *Downtown Office Space.* Seattle, WA: Downtown Seattle Association.

[4] Intergovernmental Panel on Climate Change. (2014). *Climate Change 2014 Synthesis Report Summary for Policymakers.* (pp. 4) Switzerland; IPCC.

[5] IPCC, 2014, pp. 17

[6] US DOE, 2011, Chapter 1

[7] Oliver, CD., Nassar, NT., Lippke, BR., & McCarter, JB. (2014). Carbon, Fossil Fuel, and Biodiversity Mitigation with Wood and Forests. *Journal of Sustainable Forestry, 33*(3), 248-275.

[8] Michael Green Architects. (2015). *Wood Innovation and Design Center.* Retrieved from: http://mg-architecture.ca/work/wood-innovation-design-center/

[9] Bullitt Foundation. (2013). *Bullitt Center.* Retrieved from: http://www.bullittcenter.org/

[10] Michael Green. (2012). *The Case for Tall Wood Buildings.* British Columbia, Canada: MGB.

[11] Skidmore, Owings, & Merrill. (2013). *Timber Tower Research Project.* Chicago, IL: SOM.

[12] Mahlum. (2014). *CLT Feasibility Study: A Study of Alternative Construction Methods in the Pacific Northwest.* Seattle, WA: Mahlum.

[13] UN Department of Economic and Social Affairs. (2014). *World Urbanization Prospects, the 2014 Revision.* Retrieved from: http://esa.un.org/unpd/wup/Country-Profiles/

[14] SOM, 2013, pp. 2

[15] SOM, 2013, pp. 40

[16] US General Services Administration. (2013). *Federal Real Property Profile.* Retrieved from: http://www.gsa.gov/portal/content/102880

[17] US General Services Administration. (2015). *Sustainable Design.* Retrieved from: http://www.gsa.gov/portal/content/104462

BACKGROUND RESEARCH

02

This thesis lies at the intersection of a number of established and emerging fields in architecture and structural engineering. An overview of key theoretical issues follows; starting with building type and precedents, then exploring structural issues, assessing developments in environmental performance, and finally providing an analysis of code issues and obstacles to implementation.

TYPE BASED ISSUES AND DRIVERS

> *More than any other building type, the office building is conventionally dominated by the plan. Its efficiency, its marketability, as well as the corporate culture of the client, unfold primarily in plan. This is not to say that the section has no role in the design of the office building; but in the end, it is the plan multiplied vertically, that betrays its true workings. The corner office, the cellular offices, the open plan: these spatial divisions may evolve with lightweight partitions, but they are anchored by the permanent positioning of a building's infrastructures. In that sense, the ultimate limitations and opportunities of an office building manifest in the disposition of these services in plan.*[1]

The plan of a modern office building is carefully orchestrated in response to a series of rigorous requirements. Commercial building developers aim to maximize leasable floor space and future adaptability while minimizing construction costs.

The maximum allowable floor area for a given site is dictated by zoning requirements, usually through an allowable floor area ratio (FAR). FAR dictates the maximum allowable space that can be constructed on any site as a function of the total square footage of the site. This in addition to height limitations along with other zoning prescriptions dictate the maximum envelope a building can fill on any given site.

Once the overall envelope is determined, architects test layouts against interior space requirements. The efficiency of interior space for an office building is driven by lease depth (typically 40 to 50 feet in the US, 20 to 25 feet in the EU)[2] as well as the ability of the structure to accommodate the module of furnishings (typically five feet in the US).[2] Secondary criteria for the suitability of office space include access to daylight and views and,

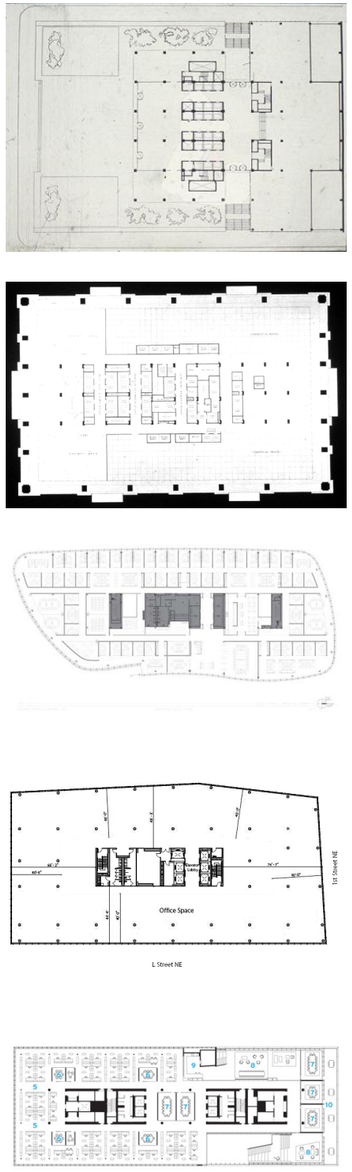


Figure 3. American and European Office Building Floor Plans

Left Column (from Top):

Seagram Building (1958) (from artstor.org)

John Hancock Center (1969) (from artstor.org)

Centennial Tower (2009) (from archrecord.construction.com)

1100 First Street (2009) (from archrecord.construction.com)

Building 335 (2012) (from archrecord.construction.com)

Right Column (from Top):

Petronas Towers (1996) (from pcparch.com)

30 St Mary Axe (2004) (from fosterandpartners.com)

Turninn (2010) (from PK Arkitektar)

Herostrasse 12 (2014) (from Max Dudler Architects)

DNB C-Building (2012) (from Dark Arkitekter)

The Carve (2014) (from A-Lab)

for buildings with operable windows, natural ventilation. The standard for Class A office space in the US is a fully glazed facade allowing views and daylight from all sides. Associated with all of these variables is the location of the core.

The location of the core must meet requirements for fire, egress, and lateral force resistance but also allow for flexibility in the subdivision of floors to accommodate multiple office layouts. Additionally, designs must accommodate changes to tenants and working environments over time in response to new technologies. These requirements in the United States often lead to rectangular office buildings with a central core as the market standard.[2]

Other key elements affecting the efficiency and economic viability of office buildings include: column spacing and floor-to-floor heights. Column spacing and floor-to-floor height are opposing criteria, with optimal solutions balancing the demands of each. Longer spans (wider column spacing) is valued as it allows a greater degree of flexibility for the layout of furnishings and spaces within the building. However, longer spans increase the depth of the building floor system thus increasing floor-to-floor heights. As floor-to-floor heights become increasingly large, fewer floors fit within the zoning envelope thus decreasing the overall economic viability of a project. The optimal relationship between these two factors for steel and concrete buildings is typically a column spacing of 30 to 40 feet.

TIMBER STRUCTURAL SYSTEMS

While timber buildings pre-date the development of modern structural types methods of building with wood have largely been forgotten in favor of steel and concrete construction. Due to concerns about the environmental impact of building out of steel and concrete coupled with advances in timber material and manufacturing technologies, there is a renewed interest in exploring how new wood technologies can fit into the modern building ecosystem. There have been a number of tall wood buildings constructed over the past decade as well as research efforts intended to move wood construction forward.

Two primary reports target the development of tall timber structures for North America. The first research report developed by Green[3] proposed a mass timber steel composite system for a hypothetical residential building in Vancouver BC. The proposed structure was analyzed for its ability to withstand gravity, wind, and seismic forces at a schematic level. Additionally, the proposal treated issues associated with codes, systems, and architectural viability. The proposed structure was tested for its application to 12, 20, and 30 story residential buildings. Key structural issues identified include the need for more intensive structural testing and modeling to ensure that the proposed system behaves as designed.

The second study, by SOM[4], outlined the conceptual design of a mass timber concrete hybrid structural system for a 42-story residential tower in

Chicago. The building has a centrally located core and a 9'-0" floor to floor dimension. The structural system for the proposed building consists of timber floors, columns, and shear walls. Connections are made between structural elements by steel reinforcing epoxy bonded into the wood and cast into concrete joints. Overall the building is 70% timber and 30% concrete by volume. Key issues identified for the proposed structural system include: uplift in the foundation caused by the lightness of the structure, as well as the need to test the proposed connection type to ensure it performs as designed.

Appendix A contains key information and photos of mid- to high-rise case study projects. These case studies demonstrate the current state of practice for timber design and construction. The case study projects are primarily located in Europe and Canada. Of note is the high degree of redundancy and material inefficient inherent in the panelized construction of many of these buildings. Also significant is that the majority of these examples are built in locations without significant seismic risks.

From these case studies, two primary approaches to gravity systems for timber buildings emerge. The first is a panel-based system where the building has load bearing CLT walls that support CLT floor panels. This construction method is suited to buildings with a large number of interior partitions, typically housing. The second approach is a frame approach where glulam beams and columns support CLT floor slabs. This thesis will implement the second approach, a sketch of primary elements is shown in Figure 4.

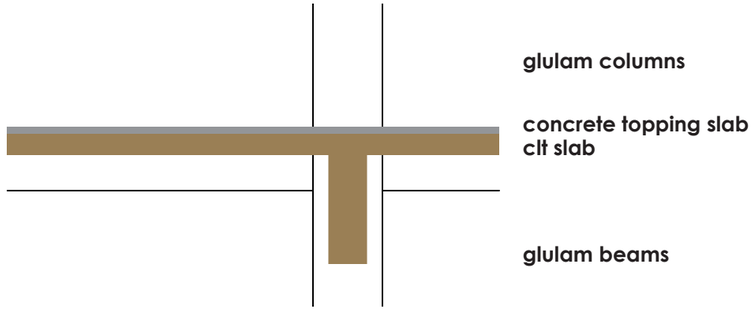


Figure 4. Gravity System Elements

A significant challenge to tall mass timber buildings in high seismic regions is the design of CLT shear walls. A number of researchers have developed systems to enhance the performance of CLT shear wall systems. Dolan et al[5] propose the introduction of viscous dampers at multiple levels in a 10-story CLT building to reduce story shears and displacements, Buchanan et al[6] propose post-tensioned timber connections for seismic resistance, and Ganey[7] studied the design of rocking CLT walls. While these emerging technologies represent a significant opportunity for higher performing, more materially efficient systems, they all require significant research and testing before they are incorporated as code accepted timber seismic systems. This thesis will implement CLT shear walls using standard connections and tie downs, which will be included in the next cycle of structural design codes[8], to understand the potential for wide spread near-term implementation of fully timber buildings in seismic zones.

PARAMETRIC OPTIMIZATION

Parametric optimization is a general category describing a series of methods for finding optimal solutions to problems with multiple input variables. At the most rudimentary level, a brute force method can be used to optimize a problem. In a brute force method, the problem is executed over all possible combinations of parameters within the solution space to provide a full set of possible solutions. By solving a problem over the entire solution space, optimum solutions and the effects of different input variables can be understood. While a number of more intelligent algorithms exist[9], that allow multi-variable problems to be optimized by selectively seeding solutions and iterating towards the optimum thus reducing computational time, this thesis will use a brute force method. The ability to understand when and why the models were under-performing was one reason for choosing a brute force method. Additionally, in this instance the time to iterate through all possible solutions of interest was not prohibitively computationally expensive.

LIFE CYCLE ASSESSMENT

Life cycle assessment (LCA) is a method for accounting the environmental impacts of a material or process.[10] To calculate the environmental impact of a building over its life span this typically requires accounting for raw material extraction, manufacturing, transportation, construction, operational energy, repairs and replacement, demolition, and disposal. There has been a recent push—led by the inclusion of LCA in LEED v4[11]—to perform LCA during building design in order to better understand the environmental impacts of construction.

For this thesis, equivalent global warming potential will be used as one measure of the efficiency of building alternatives. Using LCA to measure buildings' material efficiency helps to further test whole building LCA providing benchmark data for timber buildings. A brief overview of the LCA methodology for this project follows.[12]

GOAL

This LCA is a study of the impact of the shell and structure of mass timber buildings, developed to understand the differences in performance of parametrically iterated designs. The LCA is intended for structural engineers and architects, to serve as a baseline for the carbon intensity of mass timber buildings.

SCOPE

For each building iteration, the LCA includes impacts from glulam beams and columns (including steel connections), floor assemblies, roof assemblies, core wall assemblies, and facade wall assemblies. The analysis does not include foundations, finishes, mechanical, electrical, and plumbing equipment, material repair and replacement during the life cycle of the building, or operational energy. The impacts of materials due to production, construction, and disposal at end of life are included while credits from carbon sequestration and recycling are excluded. If credits from carbon sequestration and recycling are included, the impact of timber products can be negative (carbon sequestered). Since the goal of this analysis is to minimize material usage, including timber as a material that increases the impact of the building rather than decreases it aligns with this goal.

In order to simplify the analysis, only GWP was used to assess the environmental impact of the buildings. Analyzing other impact categories (e.g. acidification potential, eutrophication, etc.) might guide different conclusions about minimizing the environmental impacts of buildings.

The LCA work done for this thesis has not been critically reviewed and therefore should only be used for increased understanding of the impacts of timber construction not to make definitive conclusions.

INVENTORY

Data extracted from Athena Impact Estimator[13] are used to calculate the impacts of each of the assemblies. Structural material quantities are estimated using the structural sizing algorithm. Material quanitities for facades are estimated based on the area of the facades using Athena assembly data for the quantity of glass and aluminum per square foot of curtain wall.

For further detail about materials included in each assembly and their impacts see Chapter 03. For discussion of results and conclusions see Chapter 04.

CODE ISSUES

While the technology exists for wide scale implementation of high efficiency tall timber buildings, there are still significant obstacles from a regulatory and code standpoint. Chief among these are concerns related to the fire and seismic performance of timber.

Although flammable, mass timber is naturally fire resistant once a charred layer has formed, allowing structural members to continue carrying load in a fire. Some European codes allow the calculation of this effect as part of a strategy for designing mass timber buildings that meet life safety objectives. Through the use of formulas that calculate the charring rate of timber and the required fire rating, structural members are oversized to ensure sufficient load capacity to prevent collapse in the event of a fire.

American codes rely on a height limitation to govern the use of wood in buildings, Table 1 outlines fire ratings and height limits for the heights of buildings considered for this project.[14] This is largely due to concerns about the flammable nature of wood and a lack of incorporation of current scientific understanding of the performance of wood members during a fire. A performance based approach that demonstrates code equivalent performance carried out in consultation with the local jurisdiction is likely the best approach for overcoming current code restrictions.

Table 1. IBC Fire Rating Requirements

Construction Type	Structural Materials	Maximum Height	Required Primary Frame Fire Rating
Type 1A	Fire Resistive Non-Combustible	UL	3 hr
Type 1B	Fire Resistive Non-Combustible	160 ft	2 hr
Type 4	Heavy Timber	65 ft	1 hr

For this thesis, the effect of including a one hour fire rating for timber members is examined. Both the International Building Code (IBC)[14] and National Design Specification (NDS)[15] include equations for calculating charring depths and resulting member capacities. For buildings above 65-feet in height this is insufficient as the sole fire design approach. As such additional fire protection methods such as fireproofing structural assemblies with gypsum wallboard will be necessary for meeting the code-required minimum fire ratings.

The lateral system for tall wood buildings is the other primary obstacle to implementation in seismic zones. Only a few buildings have been designed in seismic regions (Cenni di Cambiamento (Italy) and Framework (Portland) see Appendix A) both required significant testing work to validate the design and performance of lateral force systems. While there is a significant amount of research around the development of novel systems for the use of CLT as a lateral force resisting system, American structural design provisions as defined by ASCE 7[16] require all wood systems to meet performance based design criteria with analytical calculations and test data to be submitted to the authority having jurisdiction for approval.

REFERENCES

[1] Picon, A. (2013). A-Typical Plan: Projects and Essays on Identity, Flexibility and Atmosphere in the Office Building. Kuo, J. (Ed.). (pp. 21) Zurich, Switzerland: Park Books.

[2] Kohn, A., & Katz, P. (2002). Building Type Basics for Office Buildings. Kliment, S. (Ed.). New York: J. Wiley & Sons.

[3] Green (2012)

[4] SOM (2013)

[5] Dolan, JD., Bordry, V., Pei, S., & Van de Lindt, J. (2014). Tall Cross Laminated Timber Building: Design and Performance. *Structures Congress 2014*.

[6] Buchanan, A., Palermo, A., Carradine, D., & Pampanin, S. (2011). Post-tensioned timber frame buildings. *The Structural Engineer.* 89(17). 24-30.

[7] Ganey, R. (2015). *Seismic Design and Testing of Rocking Cross Laminated Timber Walls.* Master's thesis, University of Washington, Seattle, Washington.

[8] American Society of Civil Engineers. (2016). Minimum Design Loads for Buildings and Other Structures. Reston, VA.

[9] Rutten, D. (2010). Evolutionary Principles Applied to Problem Solving. Retrieved from: http://www.grasshopper3d.com/profiles/blogs/evolutionary-principles

[10] International Organization for Standardization. (2006). Environmental Management - Life Cycle Assessment - Principles and Framework. Geneva, Switzerland.

[11] US Green Building Council. (2015). LEED. Retrieved from: http://www.usgbc.org/leed

[12] Simonen, K. (2014). Life Cycle Assessment. Smith, RE. (Ed.). New York: Routledge.

[13] Athena Sustainable Materials Institute. (2015). Athena Impact Estimator for Buildings 5.1.01 [Computer software]. Ottawa, Canada: Athena Institute.

[14] International Code Council. (2015). International Building Code. Country Club Hills, IL: International Code Council.

[15] American Wood Council. (2015). National Design Specification for Wood Construction. Leesburg, VA.

[16] American Society of Civil Engineers. (2010). Minimum Design Loads for Buildings and Other Structures. Reston, VA.

PARAMETRIC ALGORITHM

As the primary method for understanding the material implications of basic architectural decisions (floor-to-floor height, column spacing) a parametric algorithm was developed within Grasshopper for Rhinoceros 5. The algorithm generates building geometry based on input parameters and sizes all pertinent structural elements. The various iterations are evaluated with the goal of minimizing material usage, with material quantities calculated based on the structural design and used to generate cost and environmental impacts for each iteration.

The choice to build a customized algorithm on top of existing software tools rather than using more established timber design programs was driven by the desire to be able to iterate geometry and add members as the building structure changed as well as to have full control and understanding of design assumptions. Apart from the generation of geometry and checks of members, the algorithm uses existing software tools.

SOFTWARE TOOLS

A number of existing software tools and plug-ins for Grasshopper were combined with sections of custom code to generate and analyze each iteration of the structure. A brief overview of tools used and their role in the algorithm follows.

RHINOCEROS 5

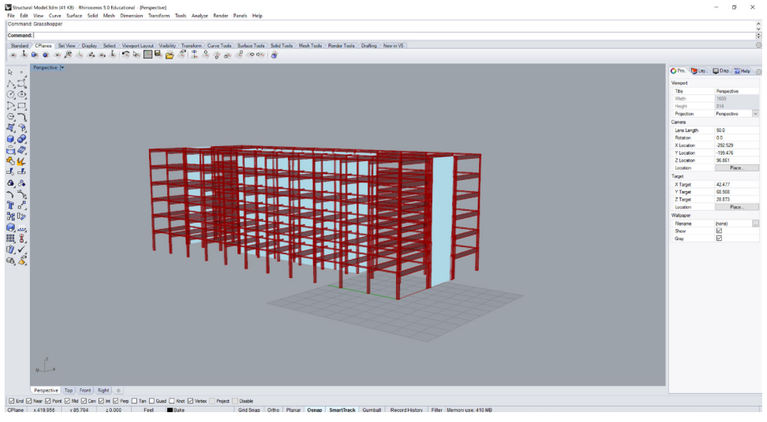


Figure 5. Structural Model Visualized in Rhino

Rhinoceros 5[1] (Rhino) is a 3D-CAD environment. For the purposes of this analysis, Rhino served primarily as a visualization environment where input data and code could be graphically checked to ensure accuracy of the algorithm. The iteration of the Grasshopper algorithm was automated by a Python script in Rhino allowing iteration of the Grasshopper algorithm without manual intervention.

GRASSHOPPER

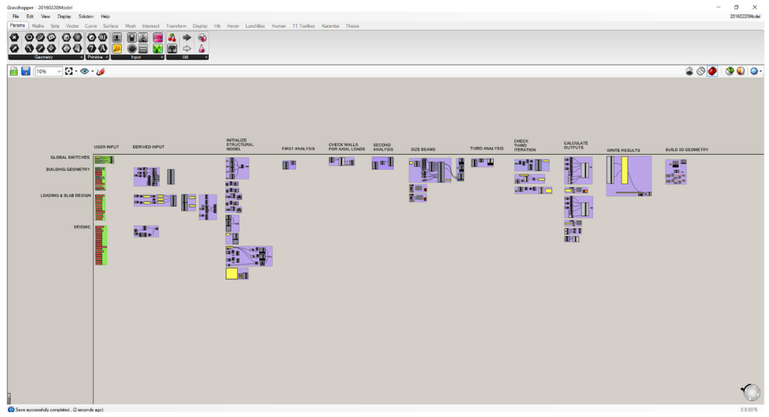


Figure 6. Grasshopper Algorithm

Grasshopper[2] is a graphical algorithm editor for Rhino. Grasshopper contains a suite of components that allows geometry to be created and manipulated in Rhino. Grasshopper also supports custom C# components that allow the user to program specialized functions into Grasshopper. User input, generation of geometry, and calculation of outputs were all handled in Grasshopper using a mixture of stock components and custom C# scripts. For a full overview of custom script components, see Appendix C. Data from the analysis is automatically written to a text file at the end of each iteration to record both the inputs and corresponding results.

KARAMBA

Karamba[3] is a finite element analysis (FEA) plug-in for Grasshopper. Based on the input of geometry and other parameters that define the structure (cross sections, materials, fixities, loads) Karamba calculates forces and deflections for the structure. Demands from Karamba were used to design structural members.

LUNCHBOX

LunchBox[4] is a software tool that enables the transfer of data between Grasshopper and Microsoft Excel. Static data tables with cross section sizes and properties for CLT slabs and walls and glulam beams and columns and cost and environmental impact data were imported into grasshopper and subsequently internalized into the grasshopper script. This allows data tables to be updated as required before the iteration is performed and subsequently kept static over the course of the analysis. Since the transfer of data between grasshopper and excel is time intensive, minimizing the transfer of data between the two programs during analysis was key to minimizing total analysis time.

PROCESS

The primary processes in the parametric algorithm are shown in Figure 7. The following sections provide a detailed description of each of the sections and their implementation. Worked structural design examples for each of the elements are shown in Appendix D. A combination of building design codes and guidelines were used as appropriate: IBC[5], ASCE 7[6], NDS[7], the CLT Handbook[8], and ACI 318[9].

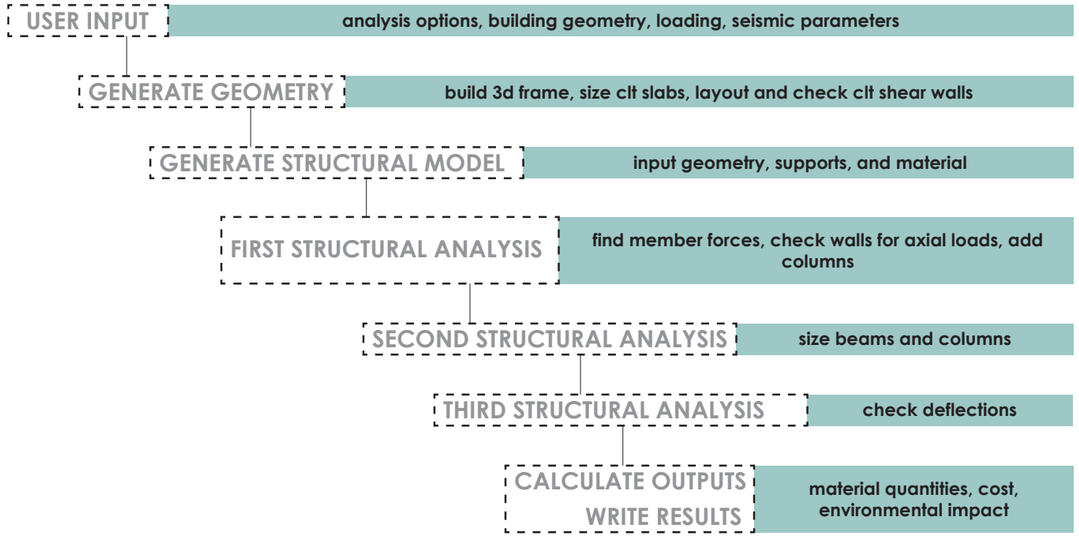


Figure 7. Parametric Algorithm Flow Chart

INPUTS

The algorithm generates analysis based on a number of input parameters that automatically initialize different geometric variations and structural design options. A list of all user-input values follows:

- Design Options
  - Include 60-minute fire rating for beams, columns, and slabs
  - Include Timber-Concrete Composite Slabs
  - Include Composite Beam-Slab
- Geometry Inputs
  - Bay Length (X and Y)
  - Maximum Building Dimensions (X and Y)
  - Maximum Building Height
  - First Floor Height
  - Typical Floor to Floor Height
  - Core Dimensions (X and Y)
  - Number of Interior Core Walls
  - Length of Facade Wall Segments
  - CLT Shear Wall Thickness
  - Concrete Topping Slab Thickness
- Loading Inputs
  - Dead Loads (Finishes, MEP)
  - Live Loads (Occupancy, Partitions)
  - Seismic (Importance Factor (I_e), Ground Accelerations (S_{DS}, S_{D1}), Response Modification Coefficient (R), Deflection Amplification Factor (C_d), Building Period Coefficient (C_t), Distribution Exponent (k))

Results are based on analysis for three primary scenarios: low-rise suburban, mid-rise urban, and high-rise urban.

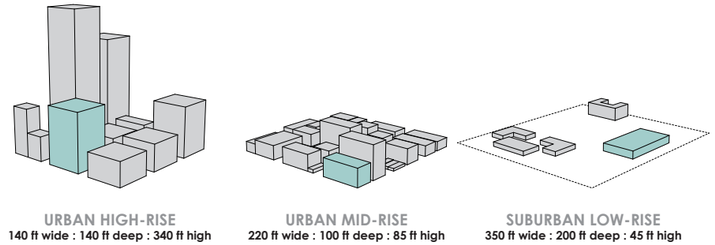


Figure 8. Zoning Envelopes

Based on the input zoning envelope and an input core layout varying geometry is generated based off the parameters shown in Figure 9. Core layouts were specified using rules of thumb for core area as a percent of floor area by building height. For each of the layouts, five interior core walls were included in the x-direction to account for walls between stairs, restrooms, and elevators in the core. The floor plan and section are layed out such that actual overall dimensions vary based on the number of grids and floors that divide evenly into the maximum dimensions.

Table 2. Core Layout

Zoning Envelope	% Floor Area	X-Dimension	Y-Dimension	# Interior Walls
Low-Rise	5%	20'	35'	5
Mid-Rise	10%	20'	110'	5
High-Rise	20%	40'	80'	5

At the element level, a number of parameters direct the structural design of each of the elements in the building. Figure 10 shows primary criteria for each of the structural elements.

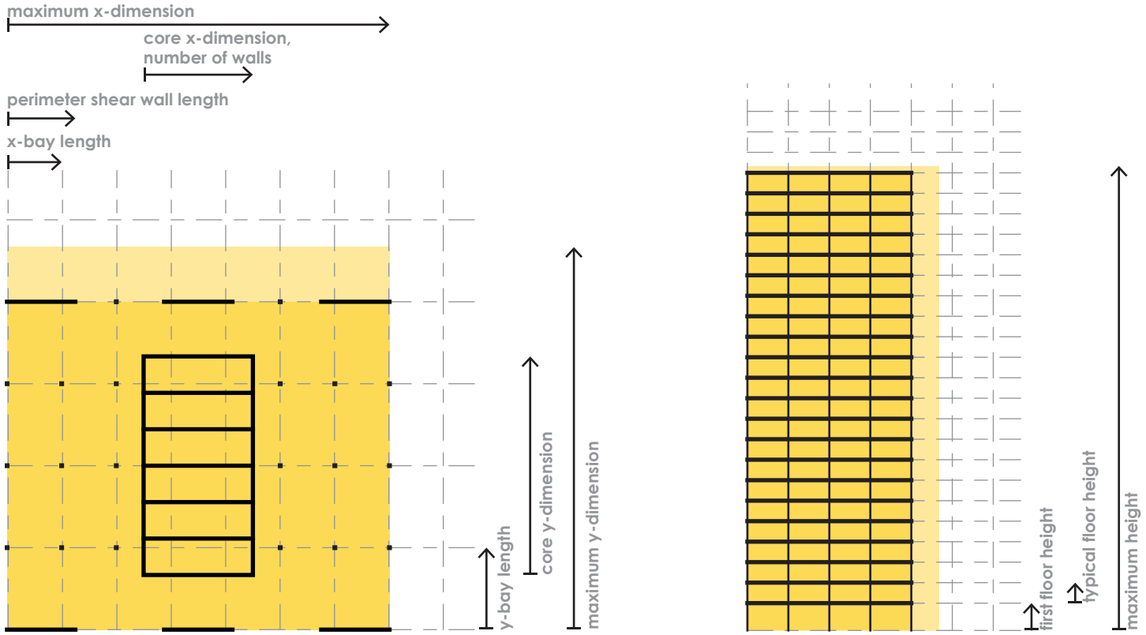


Figure 9. Geometric Parameters

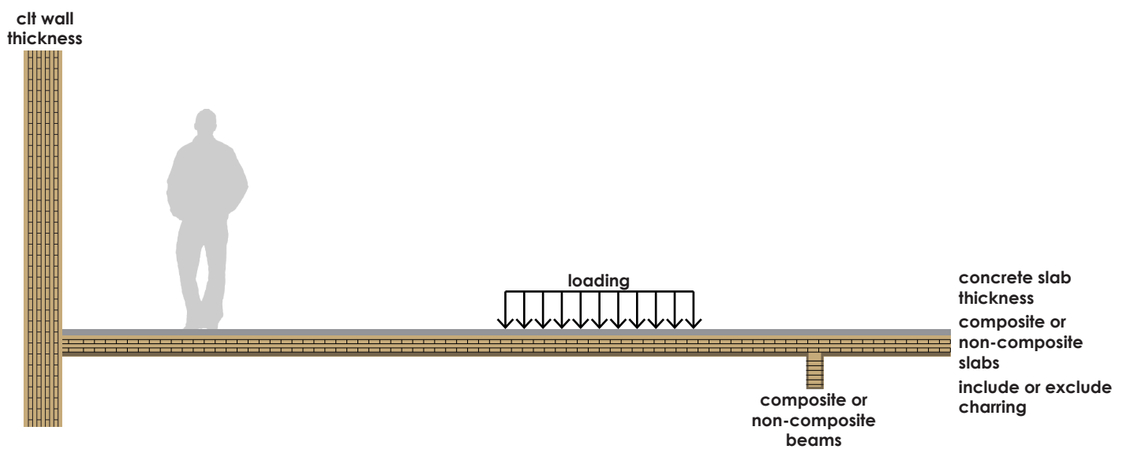


Figure 10. Loading and Design Parameters

SLABS

Based on loading, concrete topping slab thickness, and span length, the thickness of CLT slabs was calculated. Two distinct, but similar design procedures were used for the design of composite and non-composite slabs as shown in Figure 12. In both cases slabs were designed for strengths, deflections, vibrations, and a 60 minute fire rating.

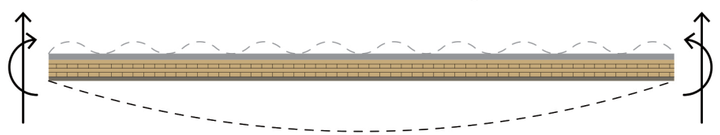


Figure 11. Slab Design Criteria Strengths, Deflections, Vibrations, and Charring

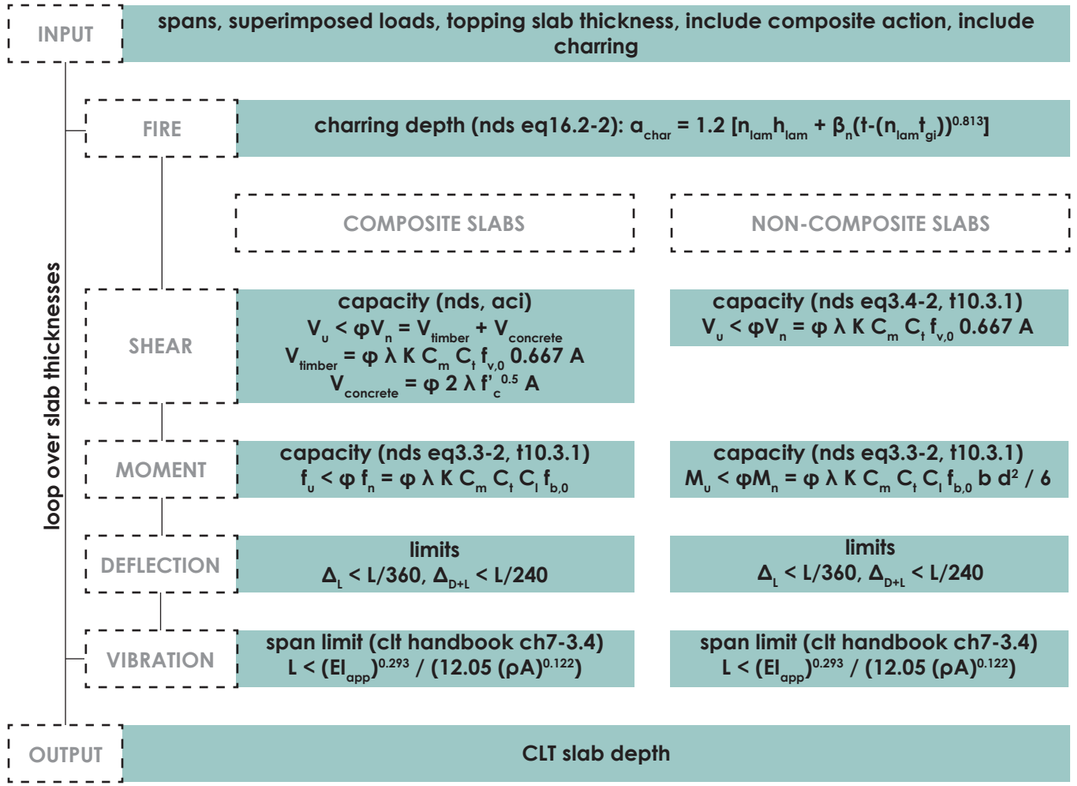


Figure 12. Slab Design Procedure

For composite slabs, transformed section properties and densities were used to increase the strength and stiffness for deflection and vibration design. This procedure assumes a connection between the CLT and concrete can develop the full demands.

Where fire is considered in the design of the timber slabs, the reduced section is used to check the slab for shear and moment capacity while the full section is used for deflection and vibration design. If slabs are not designed for charring, slabs are assumed to be fireproofed.

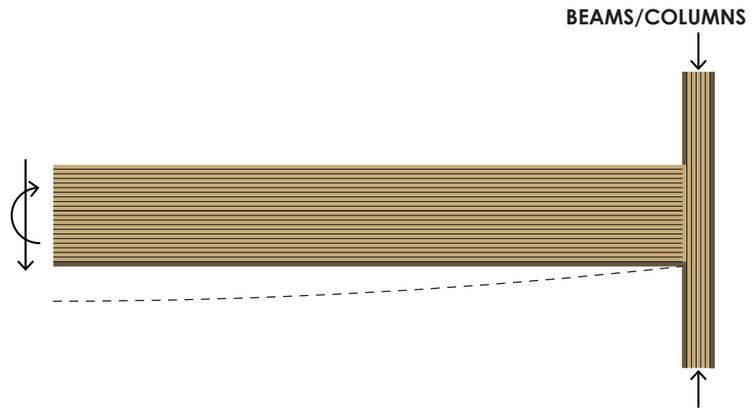


Figure 13. Column Design Criteria Strengths, Deflections, and Charring

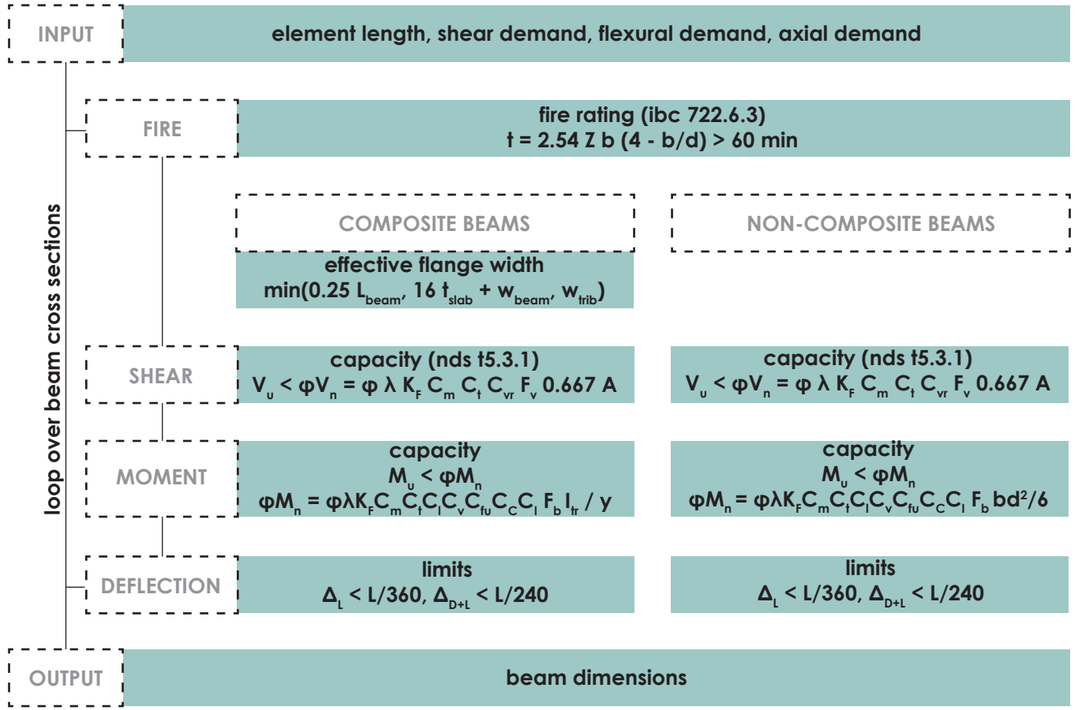
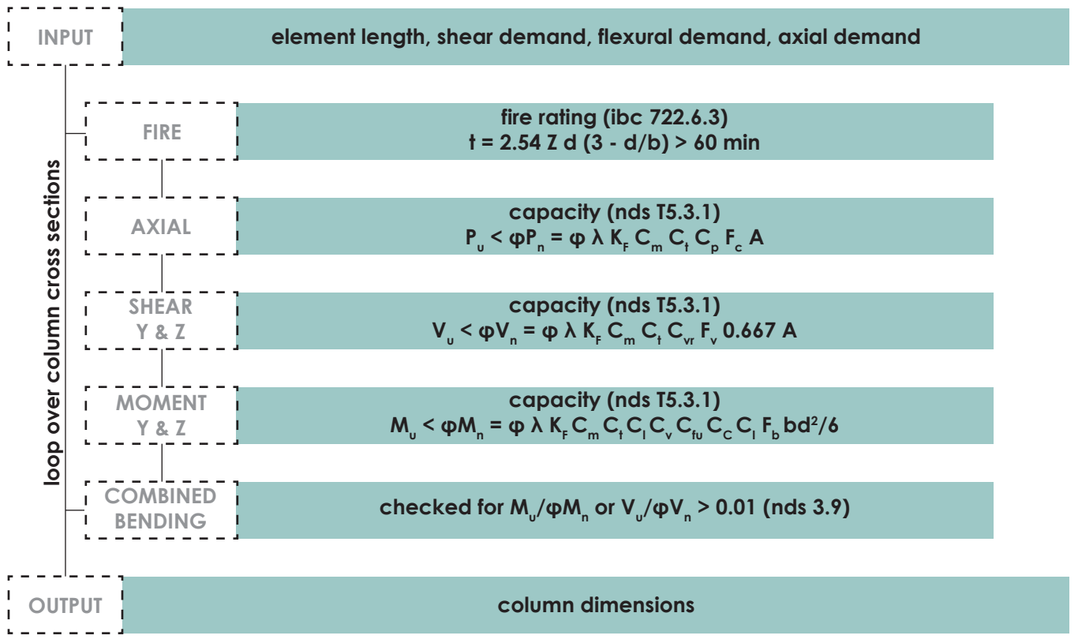


Figure 14. Beam Design Procedure

Figure 15. Column Design Procedure

From inputs of element demands and lengths, beam and column cross sections are selected for strength, deflections, and a 60 minute fire rating.



Beams are designed to either act compositely with CLT slabs or non-compositely. For composite beams, an effective flange with and corresponding effective section are used in checking the strength and deflections of the members.

For both beams and columns, fire design criteria from the IBC to determine the fire rating of members. Fire calculations use the demand-capacity ratio of the member, a measure of how close to it's ultimate load carrying capacity a member is, to calculate a corresponding time rating for each member. For this analysis, beam and column dimensions were increased until members meet a 60-minute fire rating in addition to passing checks for strengths and deflections.

SHEAR WALLS

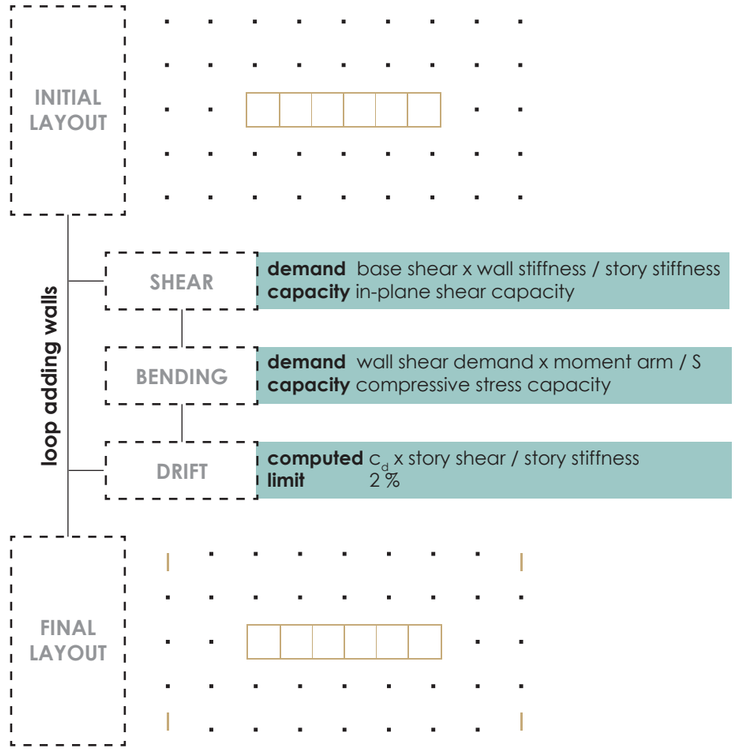


Figure 16. Shear Wall Design Procedure

Shear walls were designed to resist seismic and gravity loads. Fire was not considered for shear walls as they are assumed to be fireproofed with gypsum wallboard to achieve increased fire ratings at the core for fire and life safety. A simplified design procedure was developed that accounts for primary limit states; however, as limited design guidance exists, this procedure is likely limited and will benefit from further testing and development of consensus driven design guidelines. A number of assumptions were made to simplify the shear wall design procedure. Among these, rigid diaphragms were assumed (lateral forces distribute to walls by stiffness) and shear walls were assumed to act independently in each direction.

An overview of the shear wall design procedure is shown in Figure 16, for a more complete example see Appendix D. The algorithm starts by checking the core walls for shear, bending stress, and drifts. If the core walls are adequate for resisting lateral forces, the algorithm exits. If they do not pass all of the checks in either direction, a wall is added in the failing direction and the calculation is rerun, continuing to add walls until a passing configuration is found. Shear walls are checked for combined forces from bending and axial loads. If combined forces exceed wall capacity, columns are added to take the gravity loads.

CONNECTIONS

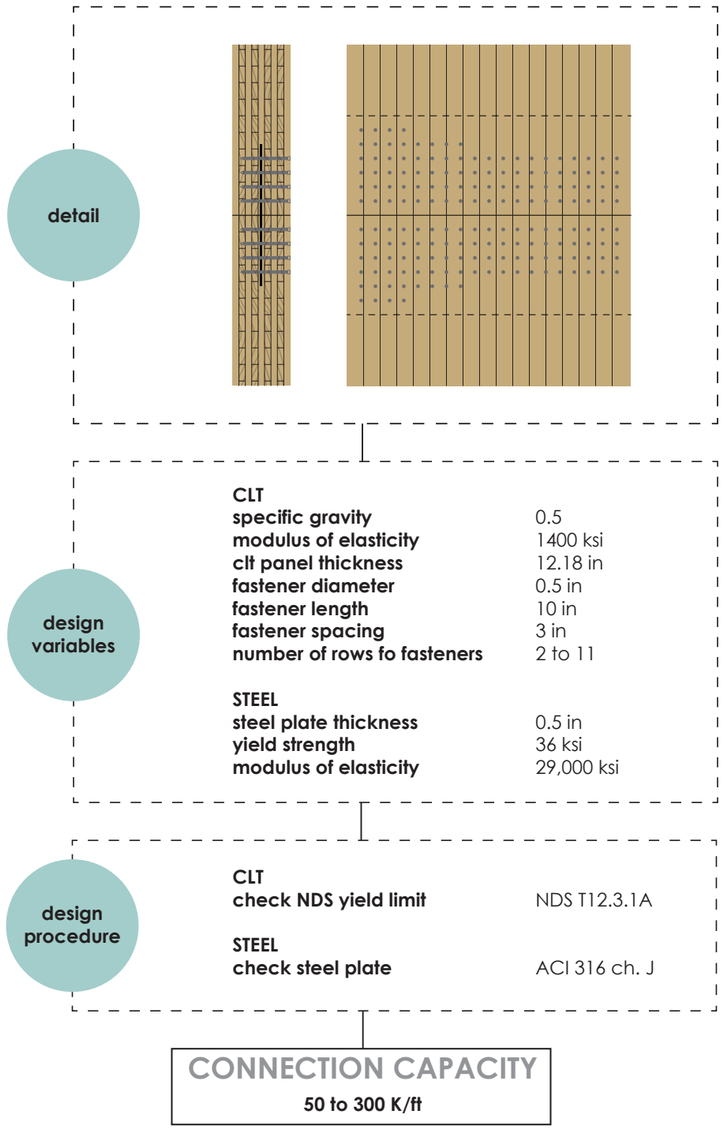


Figure 17. Connection Design Procedure

A single shear wall connection was designed and checked for its ability to resist combined shear and bending from seismic forces at the base of the wall. The connection was designed on a per foot basis looking at the last foot of wall. The connection was checked for an equivalent point force checking limit states for the CLT panel, bolts, and steel plate. Figure 17 has an overview of the connection design procedure and resulting capacity.

OUTPUTS

Based on the results of the structural design, the sizes and material quantities of all relevant structural and facade elements in the building are known. Figure 18 shows the building assemblies used for calculating costs and environmental impacts. Assemblies for floor slabs, beams and columns, roofs, facades, and core walls are used to estimate cost and environmental impacts for the shell and structure of the building.

Figure 18. Building Assemblies

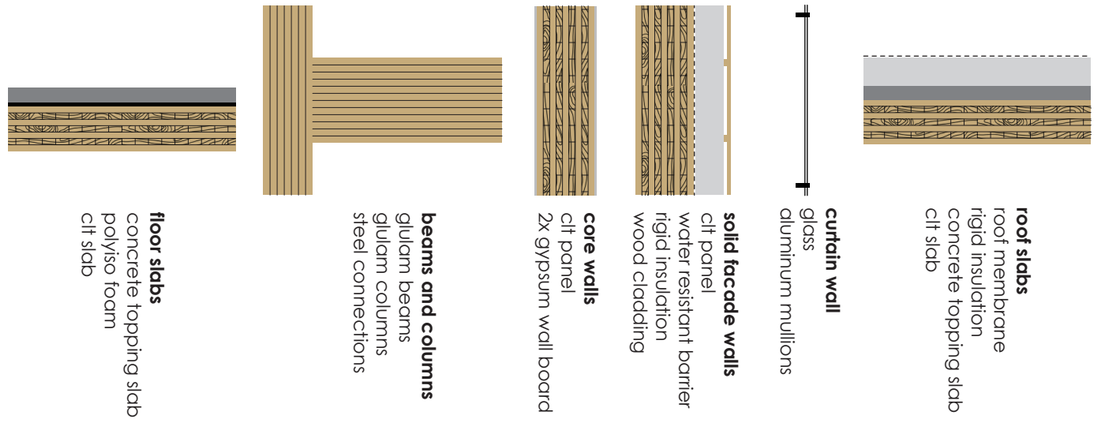


Figure 19 shows the cost and environmental impact per unit of the different assemblies. Cost data is derived from a mixture of RS Means[10] data and cost estimates generated by Hovhannisyan[11]. Environmental impact data is based on data from the Athena Impact Estimator database[12]. Environmental impacts are calculated excluding credits for carbon sequestration and end of life recycling. If carbon sequestration is included, the total life cycle carbon impact for wood products is negative (more carbon is absorbed during growth than is emitted during manufacturing).

Figure 19. Assembly Impacts

Assembly	Quantity	Material	Cost	GWP
Floor Slabs	ft^2	Concrete Topping Slab	$6.75/$ft^2$	0.85 kg CO_2e/in-ft^2
		Polyiso Foam	$2.71/$ft^2$	0.22 kg CO_2e/ft^2
		CLT Slab	$2.58/in-$ft^2$	0.36 kg CO_2e/in-ft^2
		Steel Connections	1% CLT Cost	0.06 kg CO_2e/in-ft^2 CLT
Beams	ft^3	Glulam Beams	$30.24/$ft^3$	5.17 kg CO_2e/ft^3
		Steel Connections	1% Glulam Cost	0.75 kg CO_2e/ft^3
Columns	ft^3	Glulam Columns	$35.27/$ft^3$	5.17 kg CO_2e/ft^3
		Steel Connections	1% Glulam Cost	0.75 kg CO_2e/ft^3
Core Walls	ft^2	CLT Panel	$2.58/in-$ft^2$	0.36 kg CO_2e/in-ft^2
		Gypsum Board	$11/$ft^2$	0.64 kg CO_2e/ft^2
		Steel Connections	1% CLT Cost	0.06 kg CO_2e/in-ft^2 CLT
Facade Wall	ft^2	CLT Panel	$2.58/in-$ft^2$	0.36 kg CO_2e/in-ft^2
		Water Resistant Barrier	$2.66/$ft^2$	0.04 kg CO_2e/ft^2
		6" Rigid Insulation	$4.80/$ft^2$	2.38 kg CO_2e/ft^2
		Cladding	$3.88/$ft^2$	0.384 kg CO_2e/ft^2
		Steel Connections	1% CLT Cost	0.06 kg CO_2e/in-ft^2 CLT
Curtain Wall	ft^2	Glass	$85.50/$ft^2$	6.5 kg CO_2e/ft^2
		Aluminum Mullions		9.04 kg CO_2e/ft^2
Roof Slab	ft^2	Roof Membrane	$2.66/$ft^2$	0.04 kg CO_2e/ft^2
		6" Rigid Insulation	$3.23/$ft^2$	2.38 kg CO_2e/ft^2
		Concrete Topping Slab	$6.75/$ft^2$	0.85 kg CO_2e/in-ft^2
		CLT Slab	$2.58/in-$ft^2$	4.32 kg CO_2e/ft^3
		Steel Connections	1% CLT Cost	0.06 kg CO_2e/in-ft^2 CLT

Table 3. Assembly Impacts

Thus if carbon sequestration is included, the structure with the least impacts would be the one that uses the most wood. Since the aim of this study is to minimize material usage, excluding credits so that the environmental impact of timber reflects the cost of harvesting, manufacturing, and transporting the products, makes more materially intensive buildings have larger impacts. The similarity in scale between cost and global warming potential means that the two measures of efficiency can be used interchangeably to understand the overall impact of a given alternative.

LIMITATIONS

A number of items fell outside the scope of the current exploration. A brief overview of limitations of the current algorithm follows. A number of simplified loading assumptions were made: wind loading was omitted, floor live loads were assumed to be uniform at 50 psf excluding corridor loading, and a number of simplifying assumptions were made in the distribution of seismic forces (rigid diaphragms, simplified accidental torsion).

Additionally, a significant limitation in the layout of beam and column elements is that beam and grid columns match (See Figure 20). This limits the range of floor layouts that can be explored and likely omits some floor layouts that may be more efficient.

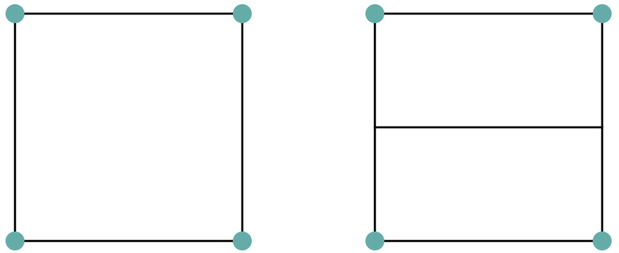


Figure 20. Existing/Future Beam Column Layout

Finally, the analysis of only a 60-minute fire rating for beam, column, and slab elements is insufficient to develop the full fire-resistance as required by codes for buildings greater than 65 feet in height.

LESSONS LEARNED

In the process of creating the custom algorithm, I ran into a number of challenges. One major challenge was that the built-in components in Grasshopper and its plug-ins often become highly inefficient and limiting to work with when implementing complicated procedures. To overcome this, I wrote custom C# code to simplify and streamline complicated procedures that would be difficult to implement using stock Grasshopper components. The custom C# scripts are included in Appendix C.

I also encountered difficulty transferring data into and out of Grasshopper. Although many plug-ins, including Lunchbox, provide components to read from and write to Excel, these components are time-intensive to execute slowing down the overall execution of the code. As a solution, I ended up internalizing data from excel into the grasshopper script and appending results to text files significantly speeding up the execution of the code.

One final lesson learned involved rethinking the order and method of problem solving to use available information to solve problems out of the traditional order and to find unknown solutions. While engineering judgment is typically used to determine the starting point for member sizes, when approaching problems with variable inputs code has to be flexible enough to adjust for all possible variations.

REFERENCES

[1] Robert McNeel & Associates (2015). Rhinoceros 5 3D Commercial Release 2015-8-10 [Computer software]. Seattle, WA: McNeel.

[2] Rutten, D. (2014). Grasshopper 3D v0.9.0076 [Computer software]. Seattle, WA: McNeel.

[3] Preisinger, C. (2013). Linking Structure and Parametric Geometry. *Archit Design, 83:* 110-113.

[4] Miller, N. (2015). LunchBox for Grasshopper v2015.7.20 [Computer software]. Omaha, NE: Proving Ground.

[5] International Code Council. (2015). International Building Code. Country Club Hills, IL: International Code Council.

[6] American Society of Civil Engineers. (2010). Minimum Design Loads for Buildings and Other Structures. Reston, VA.

[7] American Wood Council. (2015). National Design Specification for Wood Construction. Leesburg, VA.

[8] FP Innovations, & Binational Softwood Lumber Council. (2013). CLT Handbook - US Edition. Pointe-Claire, Quebec. Farmington Hills, MI: American Concrete Institute.

[9] American Concrete Institute. (2011). Building Code Requirements for Structural Concrete (ACI 318-11).

[10] RS Means Company. (2013). Building Construction Cost Data. Kingston, MA.

[11] Hovhannisyan, M. (2015). *Wood Cityscapes: Mass Timber Office Buildings.* Master's thesis, University of Washington, Seattle, Washington.

[12] Athena Sustainable Materials Institute. (2015). Athena Impact Estimator for Buildings 5.1.01 [Computer software]. Ottawa, Canada: Athena Institute.

RESULTS

04

Two primary implementations of the parametric algorithm were executed to understand the influence of different variables and different possible building configurations. The first was a general analysis looking at the broad scale interactions of primary variables. The second implementation tested the adaptability of the algorithm for custom geometry input.

GENERAL ANALYSIS

The first analysis was used to understand the behavior of the buildings as a whole and to guide further study.

EXECUTION

Analysis of the model for varying parametric inputs was performed using a brute force iteration method. Within each zoning envelope, parameters were varied in the ranges shown in Table 4.

Table 4. Parametric Inputs

Parameter	Range
Include Charring	True, False
Composite Slabs	True, False
Composite Beams	True, False
X-Bay Length	20' to 40' in 5' increments
Y-Bay Length	20' to 40' in 5' increments
First Floor Height	15' to 20' in 1' increments
Typical Floor Height	11' to 15' in 1' increments
Concrete Slab Thickness	2" to 3" in 0.5" increments
CLT Wall Thickness	12"
Length of Facade Walls	25'

In total 18,000 iterations were executed for each of the zoning envelopes. The run time for each model varied based on the number of elements with low-rise buildings having an iteration time of seven seconds, mid-rise buildings having an iteration time of three seconds, and high-rise buildings having an iteration time of ten seconds. About a third of the iterations resulted in successful buildings. About half of the buildings were not viable as floor depth were too deep to maintain an eight foot minimum clear height from top of floor to bottom of beam. About a quarter of buildings were not viable as manufactured depths of CLT were insufficient to span the required span lengths. As beams were only located along grid lines, at large column spaces slabs span up to 40 feet. The rest of the unsuccessful iterations were due to iteration times being too short for the model to completely execute.

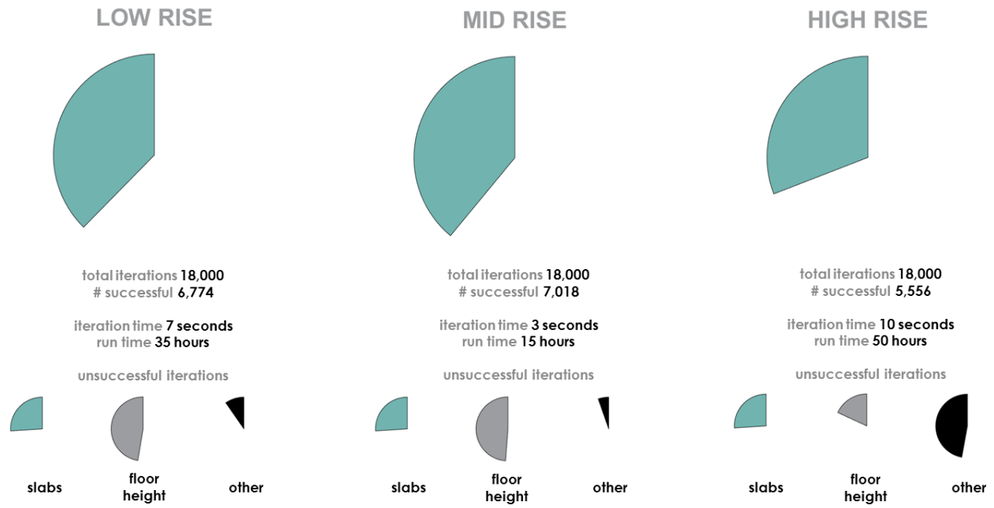
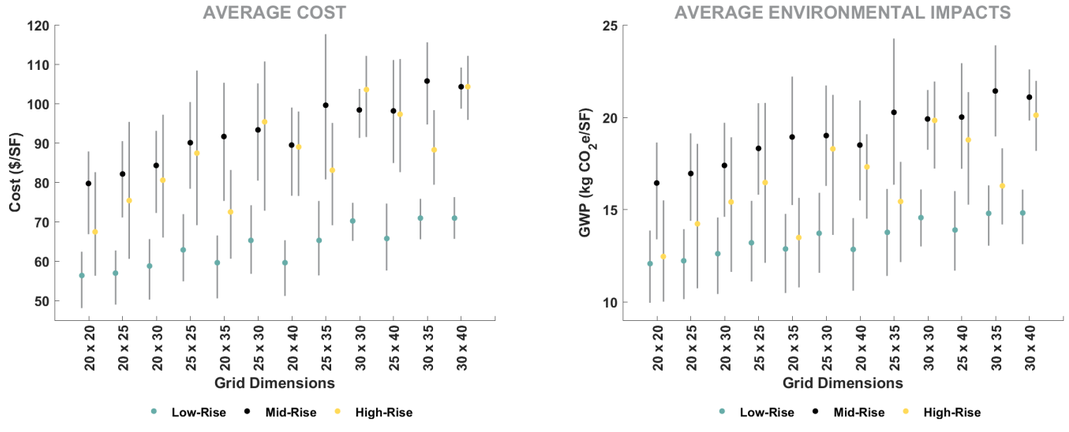


Figure 21. Execution

GENERAL FINDINGS

Based on the results from each successful iteration, average, maximum, and minimum costs and environmental impacts per square foot of the building were calculated as shown in Figure 22.

Figure 22. Average Cost and Environmental Impacts



These results show that mid- and high-rise buildings are more materially intensive than low-rise buildings. And buildings with larger grid areas, and therefore longer spans, are also more material intensive. Differences between the different zoning envelopes likely arise from the ratio of facade to floor area. While increasing impacts with spans are likely due to the increased amount of material in the floor slabs. The effect of these two variables can be seen by looking at the breakdown of cost and environmental impact per square foot for a single mid-rise iteration, as shown in Figure 23.

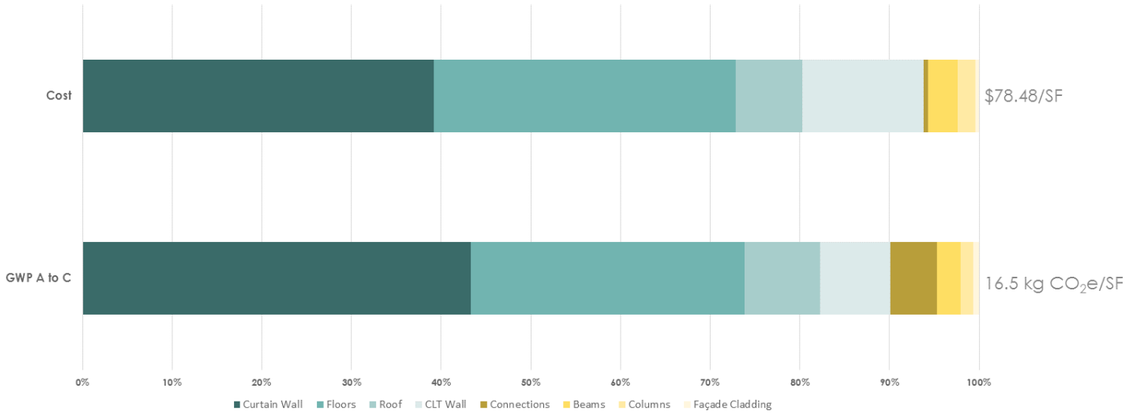


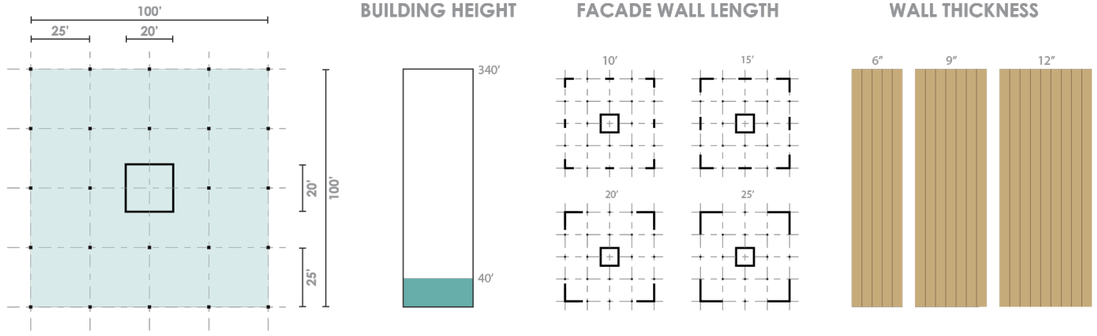
Figure 23. Cost and Environmental Impacts, Mid-Rise Iteration

The graph shows that the curtain wall and floor slabs contribute roughly 75 percent of the total cost and global warming potential of the building. Thus differences between iterations is primarily driven by differences between these two assemblies. Further analysis and strategies for optimizing timber buildings will focus on these two assemblies.

SHEAR WALLS

In order to isolate and understand factors affecting shear wall layout a simplified analysis was done using the floor plan and variables shown in Figure 24. As primary variables, the building height, floor-to-floor heights, length of shear walls along the facade, and wall thicknesses were varied. A total of 23,040 iterations were completed resulting in a thorough understanding of shear wall layout for the given simplified 100 foot by 100 foot floor plan with a 20 foot by 20 foot core.

Figure 24. Shear Wall Analysis



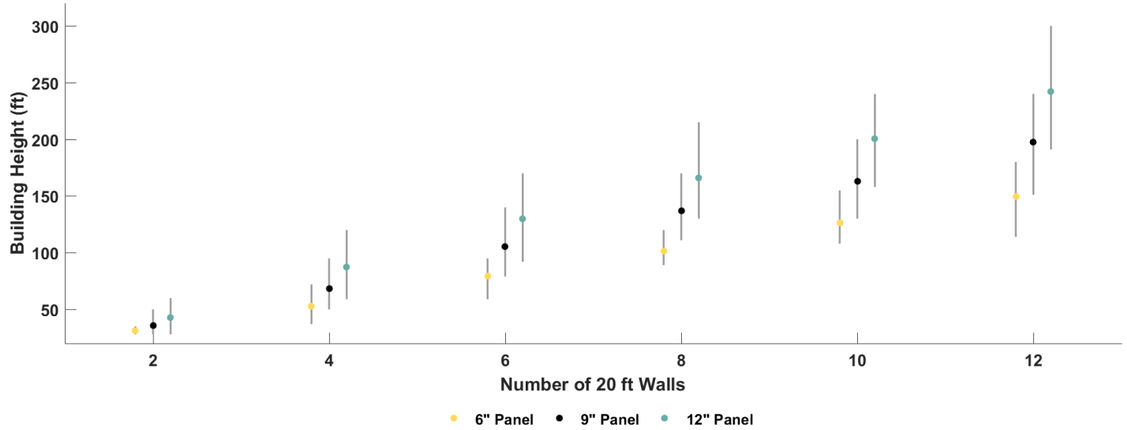


Figure 25 shows the number of 20 foot long shear walls required for varying building heights and CLT panel thicknesses. As could be expected, more shear walls are needed for taller buildings and for buildings of the same height with thinner shear walls. Figure 26 shows examples of shear wall layouts for a 100 foot tall building with varying CLT panel thicknesses.

Figure 25. Varying Shear Wall Thickness

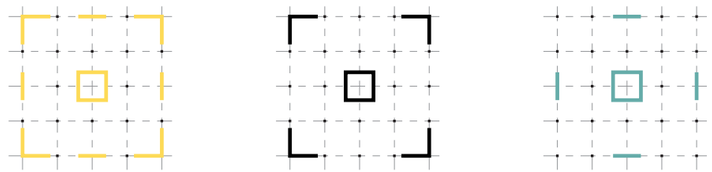


Figure 26. 6", 9", and 12" Wall Layouts, 100' Tall Building

While the configuration with 12 inch CLT panels is closest to the industry norm, fully glazed facades, the 6 inch panel configuration has significant benefits if owners and developers can be convinced that buildings with opaque walls along the facade will perform well and can be sold.

Key benefits include reduced facade costs (32% reduction, $1 million) and improved thermal performance. From a daylighting perspective, All three alternatives have glazing ratios at or above 40 percent. This is in line with prescriptive requirements in the Seattle Energy Code[1] that dictate a maximum glazed ratio of 40 percent unless the performance of the facade is tested using advanced energy modeling tools.

Figure 27. Varying Facade Shear Wall Length

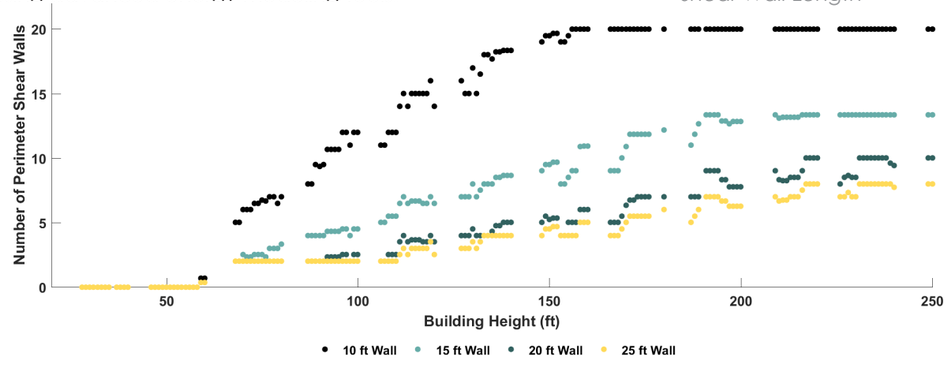


Figure 27 shows the effects of instead varying the length of walls along the perimeter of the building using 12 inch CLT panels. The results are illustrated for a 100 foot tall building in Figure 28, showing that as the shear walls become shorter more are needed to resist forces. This is due to tensile and compressive bending forces at the ends of the wall increasing as walls become shorter. Despite this effect, all four test layouts still maintain glazing ratios at or above 40 percent. With the building with 10 foot long walls having the lowest facade costs (20% reduction from building with 20' walls).

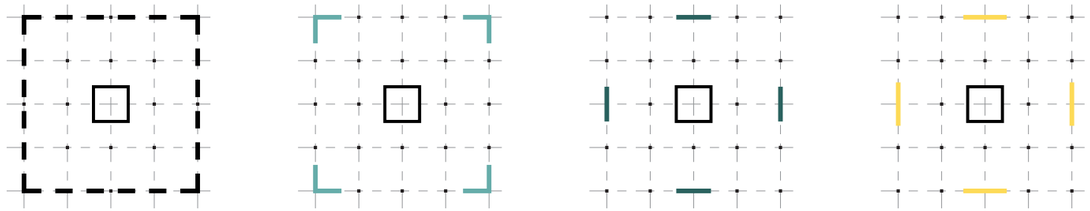


Figure 28. 10', 15', 20', and 25' Wall Layouts, 100' Tall Building

As a goal for this study was to look at the effect of placing shear walls along the facade of the building, a minimal amount of core walls were used. In practice, the developer driven standard of completely glazed facades would likely lead to more robust core layouts eliminating the need for shear walls along the perimeter of the building for low- and mid-rise buildings. For high-rise buildings, the increased demands on shear walls in the majority of cases necessitate shear walls along the facade. In this case, the design work to implement more advanced timber systems (e.g. post-tensioned rocking CLT walls) or a hybrid system with concrete shear walls and timber gravity system are more likely to fit within the demand for fully glazed facades.

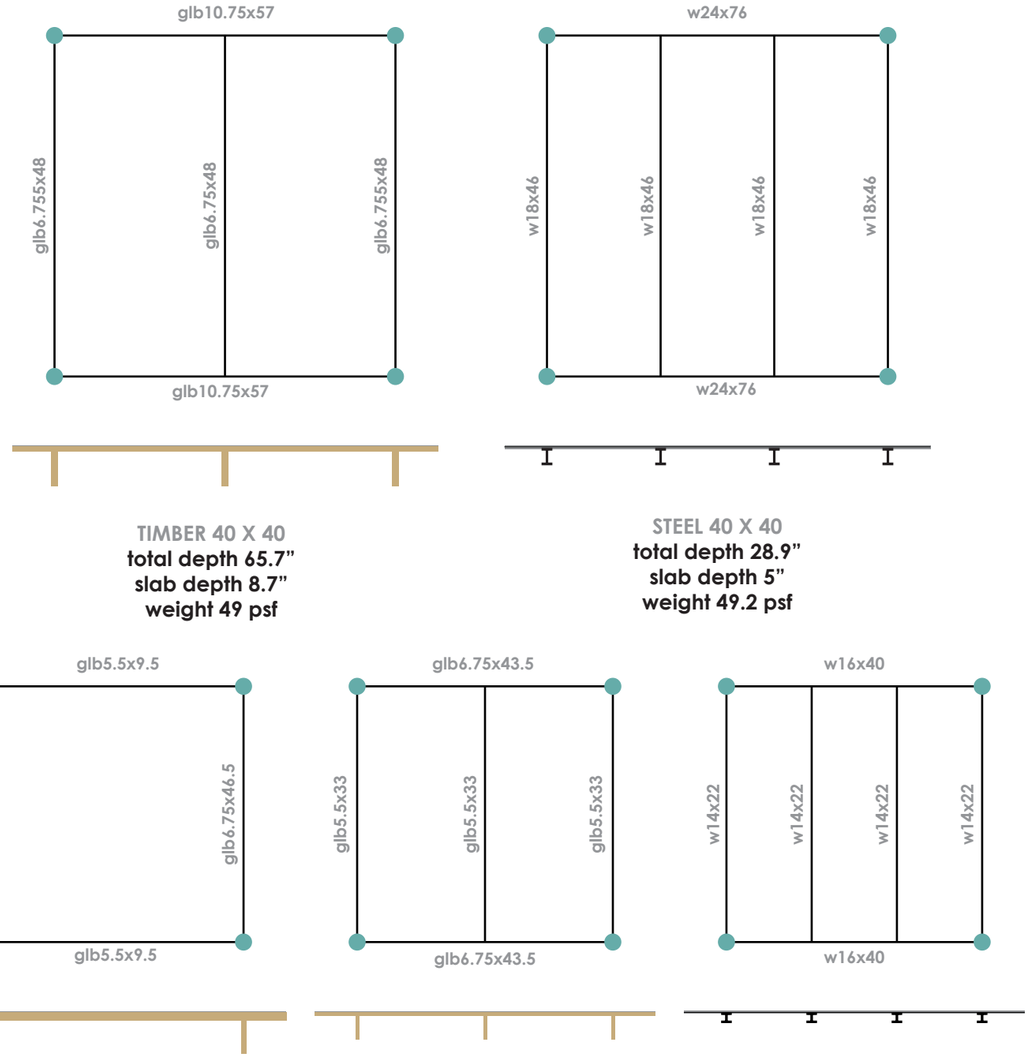
As the architecture, engineering, and construction industry moves into the future, finding ways to shift developer and building occupant expectations away from fully glazed facades would help to improve the performance and efficiency of buildings. While this thesis presents no clear path to this market transformation, some factors that may provide incentives are: building regulations further incentivizing opacity along the facade of buildings, economics tied to the increasing costs of energy and carbon, and test-cases showing that buildings do not have to be fully glazed to provide pleasant and marketable working environments.

FLOOR SYSTEMS

As the second most materially intensive assembly in timber buildings, optimized slabs and floor systems will have a significant effect on the overall performance of timber buildings. Two primary factors affect the efficiency and suitability of timber floor systems: the total quantity of material in the system (driven by slab depth) and the overall floor system depth (driven by beam depth).

To be considered as a viable alternative to steel and concrete systems, timber floor systems will need to show some competitive advantages to existing systems. To understand the strengths and weaknesses of timber floor systems, a comparison to typical steel floor systems was done. Figure 29 shows depths and weights for 30 foot square and 40 foot square grids for steel and timber systems.

Figure 29. Comparison of Steel and Timber Floor Systems



TIMBER 40 X 40
total depth 65.7”
slab depth 8.7”
weight 49 psf

STEEL 40 X 40
total depth 28.9”
slab depth 5”
weight 49.2 psf

TIMBER 30 X 30
total depth 60.7”
slab depth 14.2”
weight 59 psf

TIMBER 30 X 30
total depth 49.4”
slab depth 5.9”
weight 39 psf

STEEL 30 X 30
total depth 19.5”
slab depth 3.5”
weight 29 psf

At the outset of research, one hypothesis was that timber structural systems would prove competitive by reducing building weight thereby reducing demands on foundations and lateral systems. In preliminary testing, this

hypothesis was found to be false with timber and steel systems having similar weights in the best cases and timber systems weighing significantly more in the worst cases. Beyond this, timber systems also have total system depths roughly double those of steel systems. This puts timber at a competitive disadvantage as deeper floor systems equate to greater floor-to-floor heights reducing the number of floors that can be constructed within a given zoning envelope and increasing the amount of facade that needs to be constructed per square foot of built area.

Several strategies were tested to understand pathways for improving the performance of timber floor systems. The first strategy for reducing the material intensity and cost of floor slabs was having the concrete topping slab act compositely with the floor slab. Standard CLT slab construction requires a topping slab to minimize acoustic transmission through the floor and to help with vibrations. By using this layer structurally, minimal decreases in total slab thickness are possible. More important to minimize slab depth is shortening spans. While the full effects of adding beams to shorten spans were not studied, it is likely more efficient to add beams reducing slab thickness. Errors in the code, found after completion of results, were oversizing the thickness of composite CLT slabs by one to two inches. Corrected thicknesses are shown in Figure 29 and Figure 30; however all other results include this error. The design of non-composite slabs was not affected by this error.

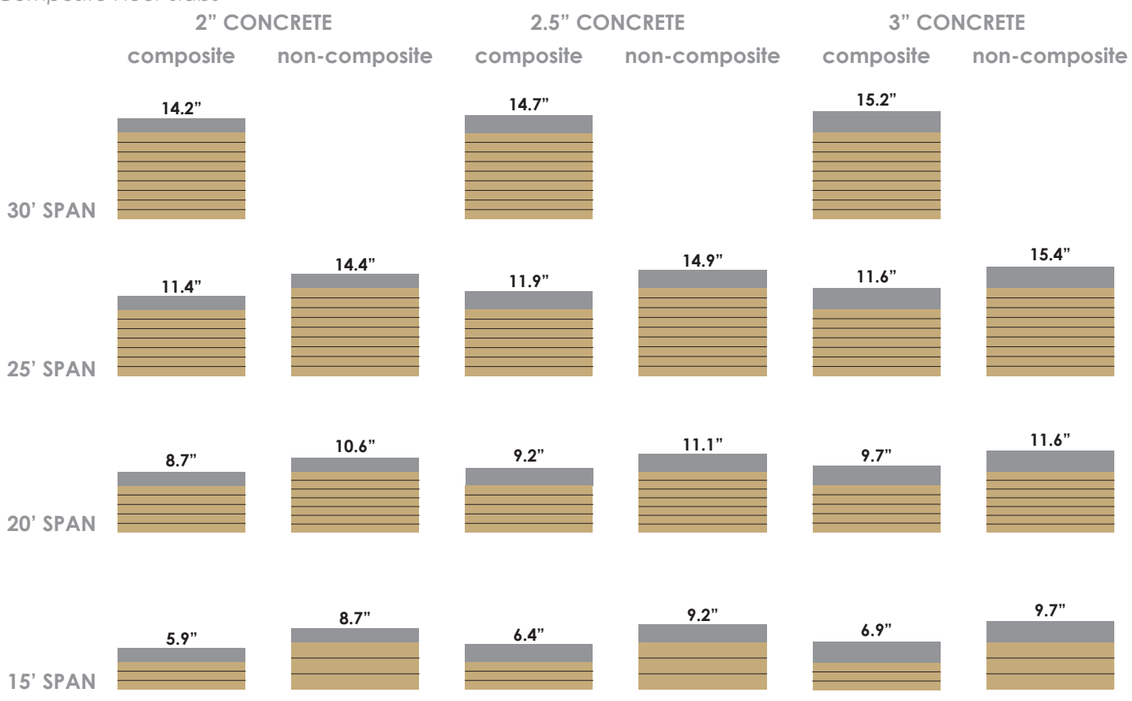


Figure 30. Composite and Non-Composite Floor Slabs

While slabs contribute the majority of the weight and material of the floor system, the beams contribute the majority of the depth of the system. Figure 31 shows the overall floor system depth for varying grid dimensions.

The total system depth varies between three and six feet with corresponding minimum floor-to-floor heights varying between 11 and 14 feet. Composite beams have a modest effect on the floor depth, reducing depths by approximately 10 inches. The overall depth is still significant as compared to steel and concrete systems.

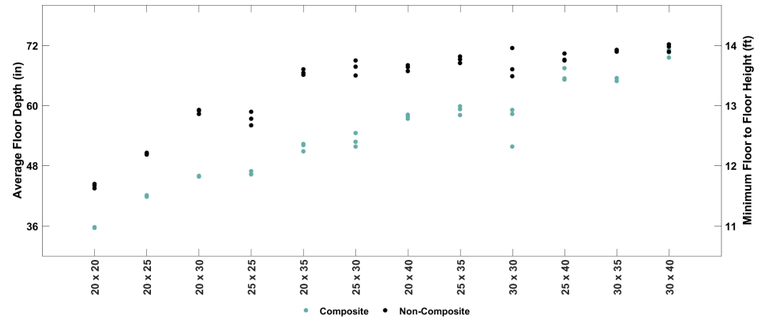


Figure 31. Composite and Non-Composite Beams

There are two possible pathways to overcoming the depth of timber systems. The first is the development of novel timber floor systems that are more efficient. This is likely a less accessible pathway in the short term as it will require investment in research and development. The second, is to provide zoning height incentives for timber buildings allowing developers to build the same number of floors. Zoning height incentives are a more viable short term solution to push timber construction. A survey of buildings in King County[2] serves as guidance for the required height increases for timber systems in each of the three zoning envelopes (See Figure 32). The required height increase varies from 15 to 50 percent depending on the chosen floor system.

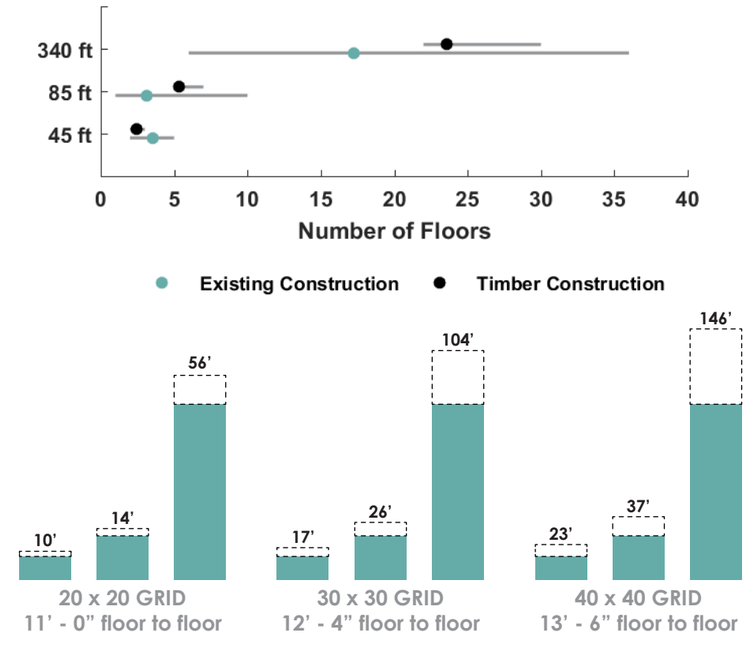


Figure 32. Zoning Height Increases

CHARRING

The effect of fire and strategies for mitigating the flammability of timber are an ongoing area of study and are one of the main deterrents to the widespread construction of tall mass timber buildings. To understand the impact of including a 60-minute fire rating for timber members, structure material quantities were calculated for structures including and excluding charring. Columns had the most significant material increase with an increase of about 3 percent as they are strength controlled. Beams had a smaller material increase of 1 percent as beam design is controlled by deflections and they are therefore initially oversized for strength. Charring has no effect on slabs as they are oversized for vibrations.

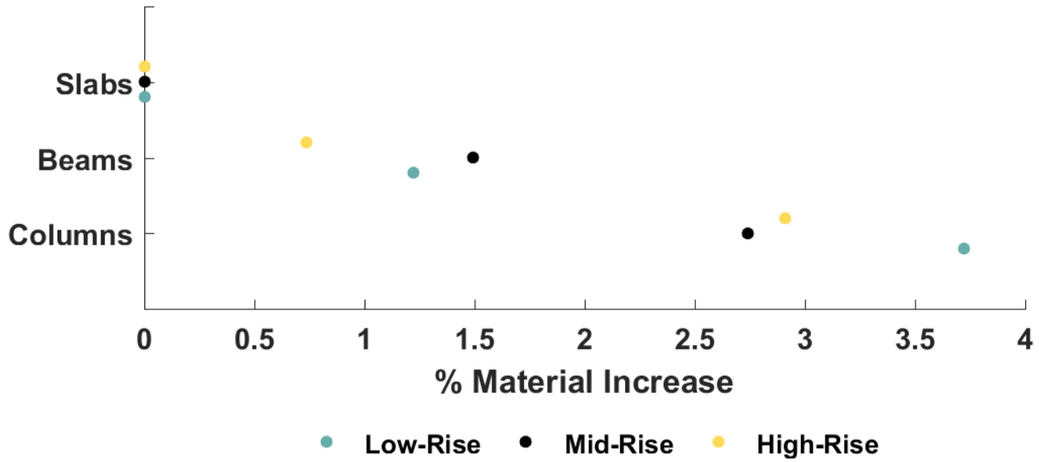


Figure 33. Charring

While this study shows that a 60-minute fire rating has a minimal effect on material quantities, it omits the full impact of alternate fireproofing methods and therefore does not provide a comprehensive understanding of the full costs. Additionally, for high-rise buildings a 60-minute fire rating is probably insufficient for the life-safety of building occupants.

ARCHITECTURAL TEST APPLICATIONS

To test the algorithm as an architectural design tool, the algorithm was adapted to take user input geometry. A basic plan and section for a low-rise (Figure 34), mid-rise (Figure 35), and high-rise (Figure 36) test buildings were developed. Based on these test cases a couple of key design issues for shear walls were uncovered. In order to get core layouts to work, longer walls leading to squarer core layouts are more effective. Additionally, matching the length of all walls in each direction is more efficient that having a mixture of longer and shorter walls.

The revised algorithm provides a quick way to test the efficiency of user specified architectural layouts against the benchmark generated through the iterative analysis.

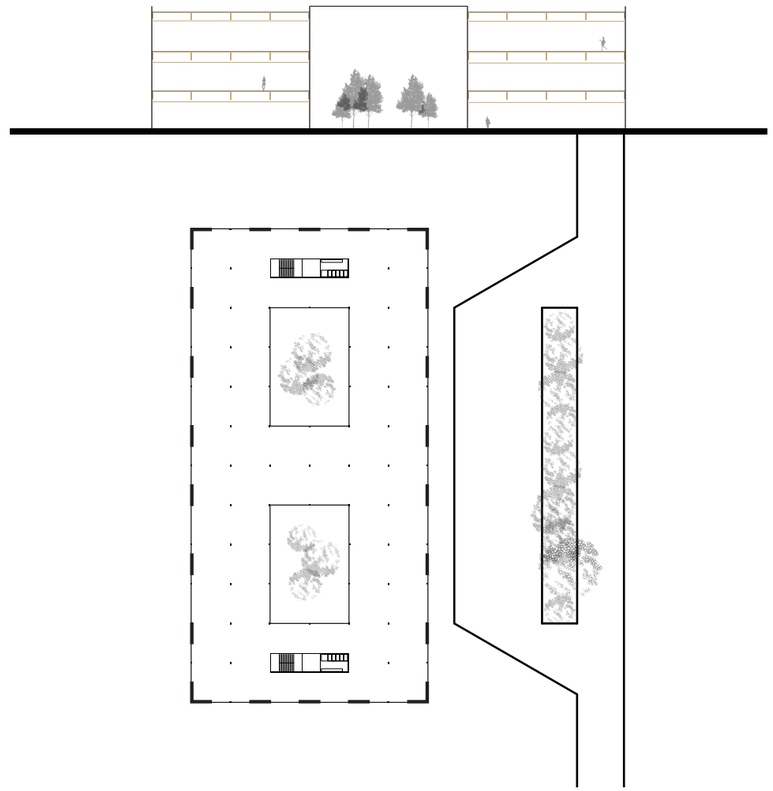
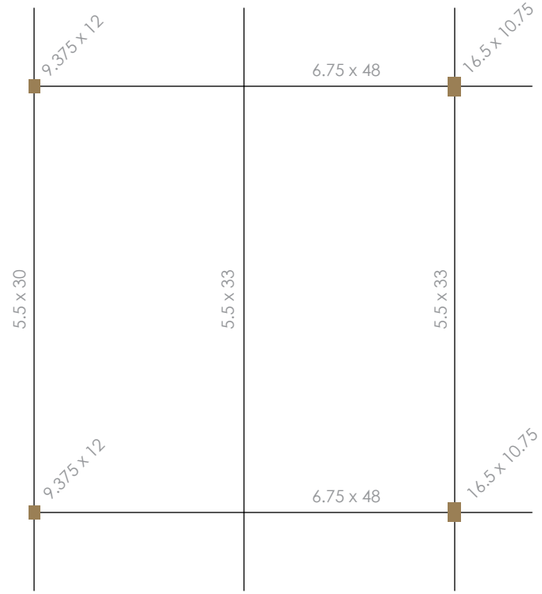
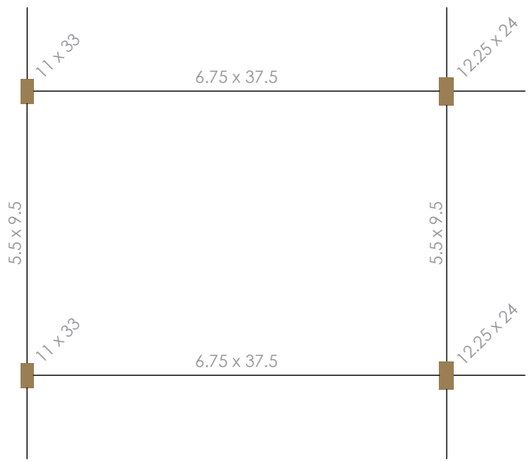


Figure 34. Low-Rise Test Building



grid 30' x 30'
slab 2" concrete over 5.125" CLT
cost $50.62/sf
gwp 10.79 kg CO_2e/sf

Figure 35. Mid-Rise Test Building



grid 30' x 20'
slab 2" concrete over 12.18" CLT
cost $93.51/sf
gwp 18.05 kg CO_2e/sf

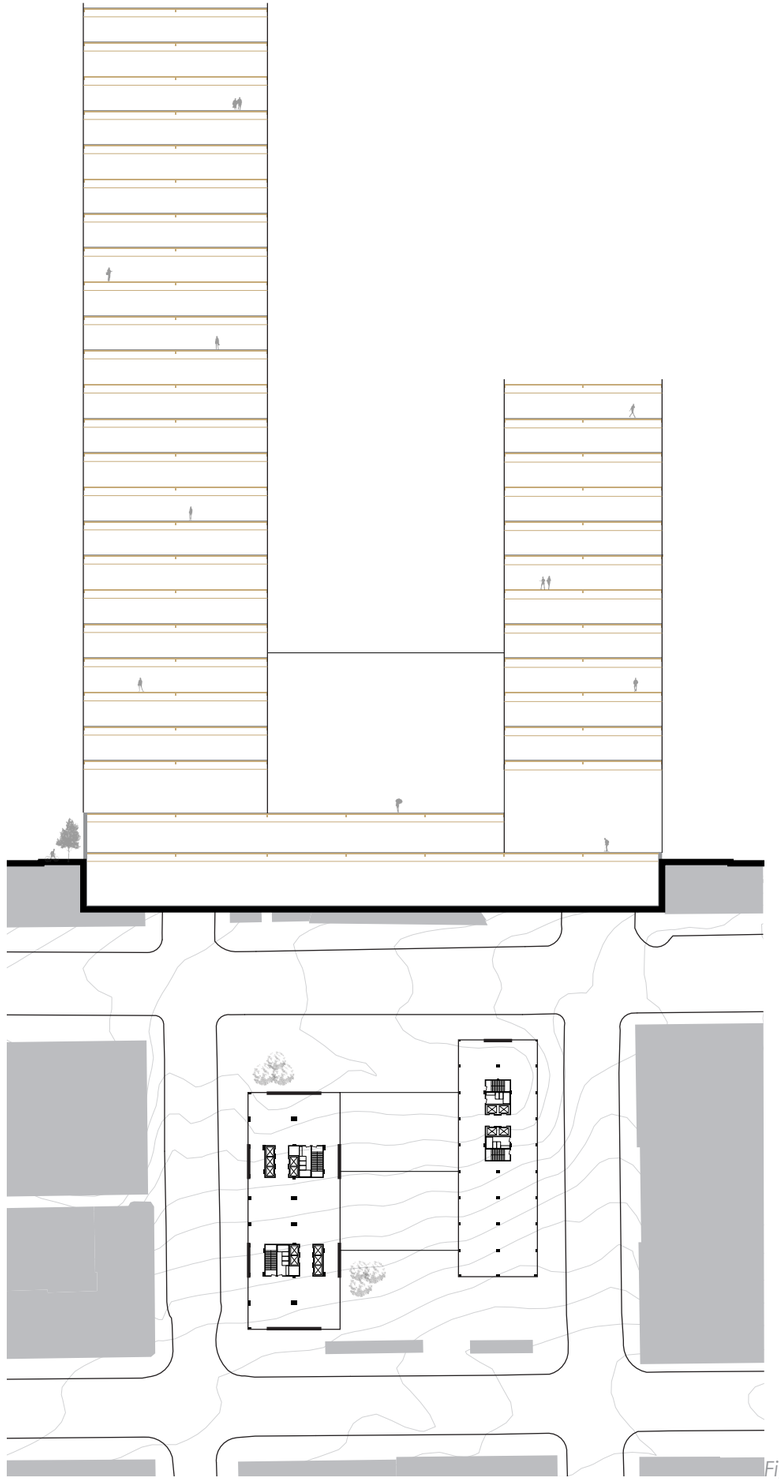
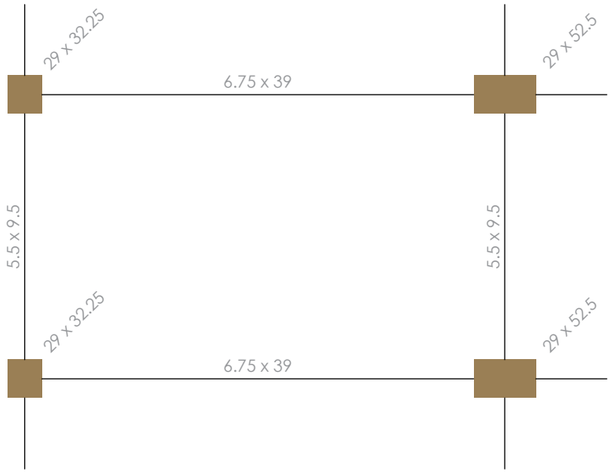


Figure 36. High-Rise Test Building



340' TOWER

grid 35' x 20'
slab 2" concrete over 6.875" CLT
cost $94.38/sf
gwp 17.83 kg CO_2e/sf

Figure 37. High-Rise Framing Plan

REFERENCES

[1] City of Seattle. (2014). *Seattle Energy Code*. Country Club Hills, IL.: International Code Council.

[2] King County Department of Assessment. (2013). Assessments Data [CSV file]. Retrieved from http://info.kingcounty.gov/assessor/DataDownload/default.aspx.

CONCLUSIONS

05

CONCLUSIONS

This thesis has explored opportunities for optimizing mass timber buildings through the development and execution of a custom parametric algorithm.

- Facades and floor systems contribute roughly 75 percent of the impacts of the building, optimization efforts focused on these two systems will have the greatest impact.
- Facades can be optimized by reducing the amount of glazing, one method for reducing the area of glazing is placing shear walls along the perimeter of the building.
    - While it is possible for this strategy to work with overall building performance goals, it is unlikely to meet developer standards for Class A office space.
- Optimized floor systems are the best near-term option for improving the efficiency of timber structures. There are two main optimization strategies:
    - Development of innovative composite floor systems, that reduce system depth and material usage.
    - Providing zoning height allowances to offset floor-to-floor height increases driven by the depth of timber floor systems.
- The algorithm provides a way to test the viability of timber structures and provides schematic level sizing of structural members.
    - This could allow architects to quickly test the implications of building with timber and understand whether or not it is a competitive alternative for a given project.
- Data from this study also serves to benchmark timber construction establishing the general range for the cost and environmental impact of timber construction.

SUMMARY OF REVIEW COMMENTS

As a part of this work a final review with guest reviewers was held, a brief summary of comments from this review follows. Reviewers were: Scott Crawford, Architect-LMN Architects, Brett Mozden, Engineer-Swenson Say Faget, Susan Jones, Architect-Atelier Jones, Joe Mayo, Architect-Mahlum, and John Stanton, Civil Engineering Professor-University of Washington.

Comments from the review focused on how the further development of the program could provide comparisons that would help understand the tipping points where timber becomes attractive as compared to steel and concrete systems. These might involve other variables (such as daylighting) that help to assess architectural criteria, or modifying the program to directly compare to other systems such as steel, concrete, or light frame timber construction. They also discussed moving away from how timber buildings are supposed to look and instead focusing on how timber technologies can work within existing norms for office construction. While the time frame for this thesis was limited, these suggestions provide guidance for how this work can be adapted to further serve practicing architects and engineers.

FUTURE WORK

While the current study provides a comprehensive overview of key structural issues affecting timber construction, there are a number of potential future additions that would serve to make the tool more robust and more useful in practice.

- A study of independent beam-column grids to find systems optimized for weight and depth.
- Refinement of loading included in the model—adding wind loading, corridor loading, and refining seismic loading/design
- Research and development of timber composite floor assemblies
- Further comparison of timber systems with steel and concrete systems
- Addition of thermal and daylighting analysis to the algorithm
- Continued architectural application and exploration of timber structures.
- Further study of the cost and material impacts of different fire performance strategies (charring vs fireproofing)

CASE STUDIES

A

The following sections provide an overview of existing and proposed mass timber buildings around the world. The buildings selected represent the state of practice for mass timber office buildings four stories or more in height. A number of tall (eight or more stories) residential buildings are also included.

TREET

Figure 38. Rendering of Treet (from ARTEC AS)

LOCATION:	BERGEN, NORWAY
COMPLETION:	2016
TYPE:	HOUSING
AREA:	3780 SQ. M
HEIGHT:	14 STORIES
DEVELOPER:	BOB EIENDOMSUTVIKLING
ARCHITECT:	ARTEC AS
STRUCTURAL ENGINEER:	SWECO
STRUCTURE OVERVIEW:	GLULAM FRAME SUPPORTING PREFABRICATED UNITS

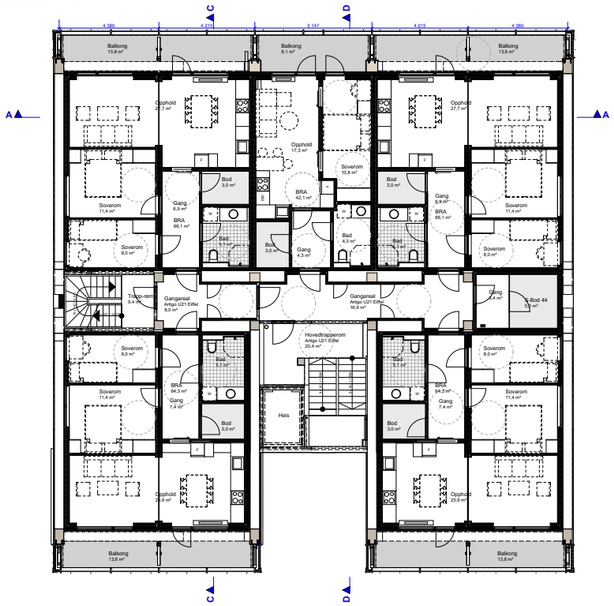


Figure 39. Stair Detail (Upper Left) (from author)

Figure 40. Exterior Facade (Upper Right) (from author)

Figure 41. Building Section (Lower Left) (from ARTEC AS)

Figure 42. Plan (Lower Right) (from ARTEC AS)

FRAMEWORK

Figure 43. Rendering of Framework (from LEVER Architecture)

LOCATION:	PORTLAND, OR
COMPLETION:	2016
TYPE:	MIXED USE
HEIGHT:	12 STORIES
DEVELOPER:	PROJECT
ARCHITECT:	LEVER ARCHITECTURE
STRUCTURAL ENGINEER:	KPFF
STRUCTURE OVERVIEW:	GLULAM BEAMS AND COLUMNS WITH CLT DECKING AND CORE WALLS

Figure 44. Rendering of Framework (from LEVER Architecture)

WOOD INNOVATION AND DESIGN CENTER

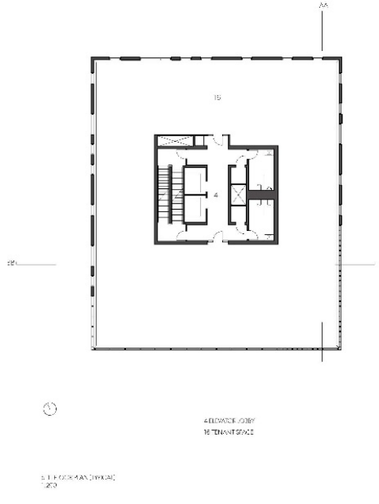
Figure 45. Wood Innovation and Design Center (from Michael Green Architecture)

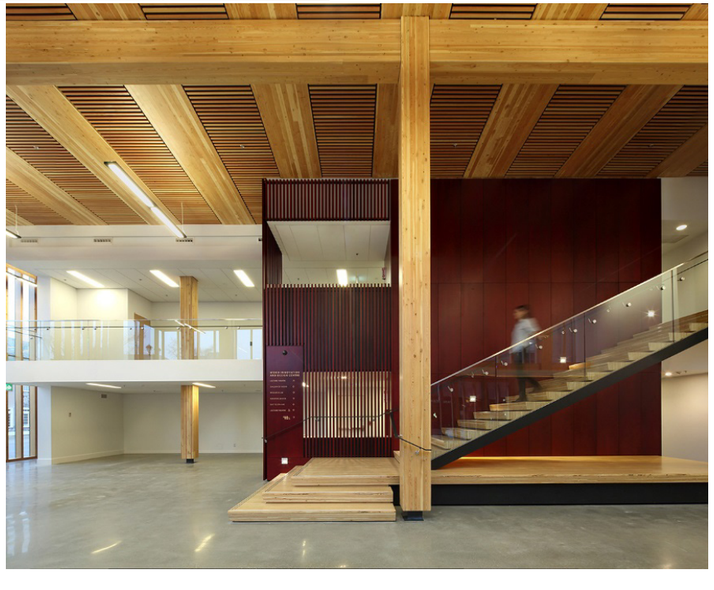
LOCATION:	PRINCE GEORGE, BC, CANADA
COMPLETION:	2014
TYPE:	OFFICE AND LABORATORY
AREA:	4820 SQ. M
HEIGHT:	8 STORIES
OWNER:	UNIVERSITY OF BRITISH COLUMBIA
ARCHITECT:	MICHAEL GREEN ARCHITECTURE
STRUCTURAL ENGINEER:	EQUILIBRIUM CONSULTING INC
STRUCTURE OVERVIEW:	CLT FLOOR AND WALL PANELS WITH GLULAM BEAMS AND COLUMNS, NO CONCRETE ABOVE FOUNDATION

Figure 46. Typical Floor Construction (Left) (from Michael Green Architecture)

Figure 47. Typical Floor Plan (Lower Left) (from Michael Green Architecture)

Figure 48. Lobby (Lower Right) (From Michael Green Architecture)





STADTWERKE LÜBECK HQ

Figure 49. Stadtwerke Lübeck Headquarters (from FpbrAG)

LOCATION:	LÜBECK, GERMANY
COMPLETION:	2014
TYPE:	OFFICE
AREA:	13,856 SQ. M
HEIGHT:	4 STORIES
OWNER:	STADTWERKE LÜBECK GMBH
ARCHITECT:	KLEIN ARCHITEKTEN
WOOD USE:	2,438 CU. M
STRUCTURE OVERVIEW:	GLULAM BEAMS AND COLUMNS WITH CLT FLOOR DECKS

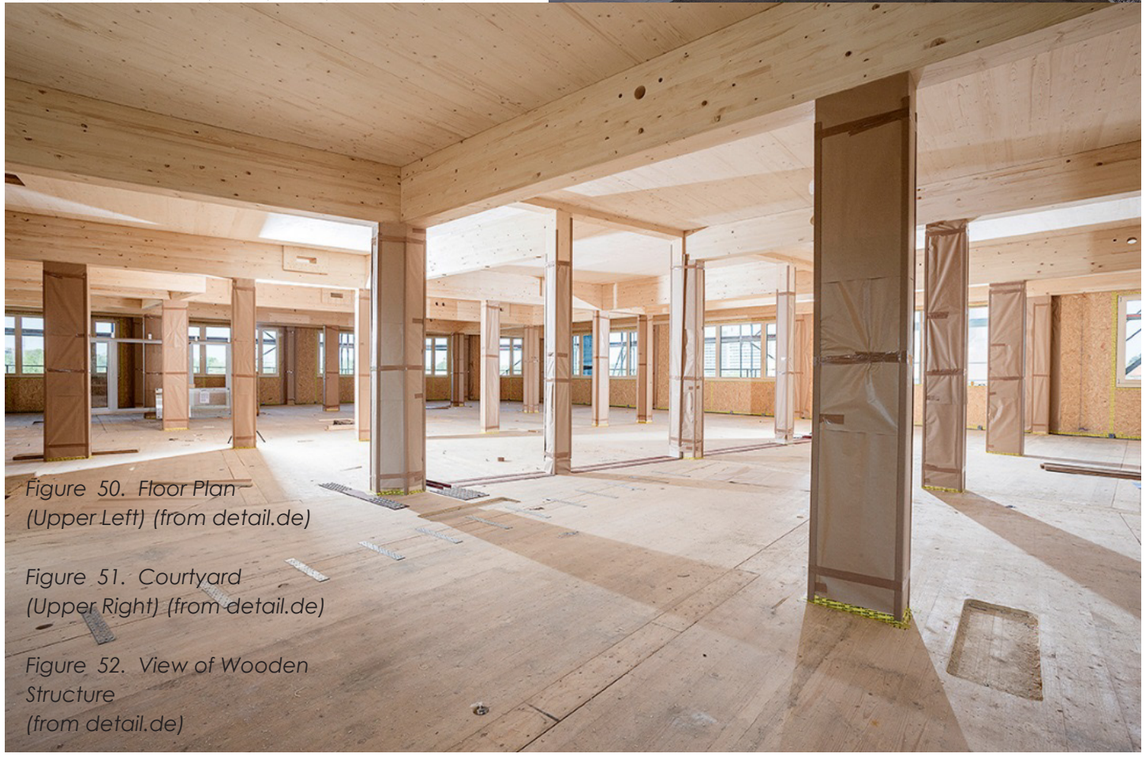


Figure 50. Floor Plan (Upper Left) (from detail.de)

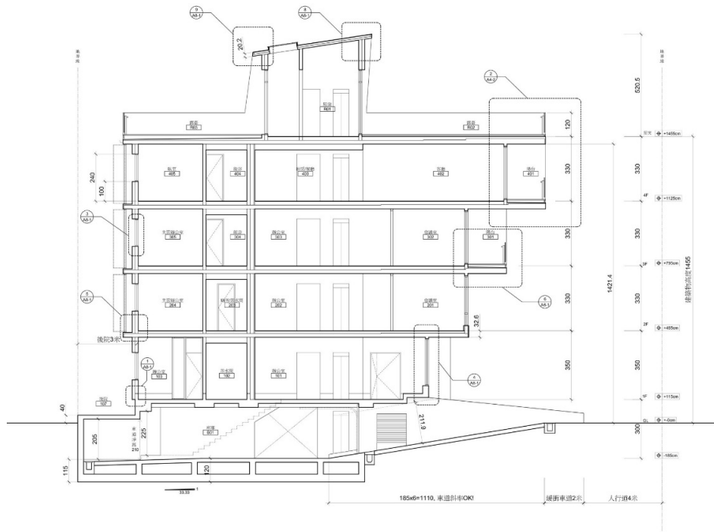
Figure 51. Courtyard (Upper Right) (from detail.de)

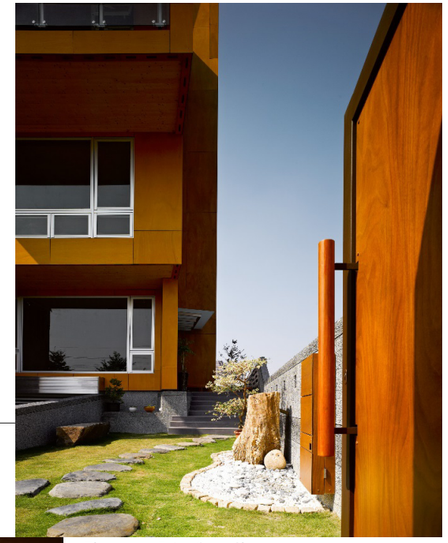
Figure 52. View of Wooden Structure (from detail.de)

WOODTEK HQ

Figure 53. Woodtek Headquarters (from Figure x Lee Kuo-Min Studio)

LOCATION:	TAIWAN
COMPLETION:	2014
TYPE:	OFFICE
AREA:	600 SQ. M
HEIGHT:	5 STORIES
OWNER:	WOODTEK
ARCHITECT:	ORIGIN
STRUCTURE OVERVIEW:	CLT STRUCTURE





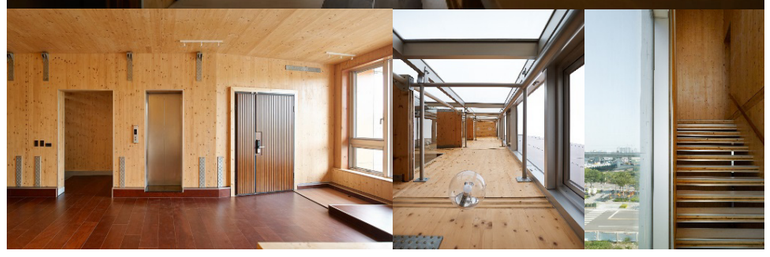


Figure 54. Section (Upper Left) (from Origin)

Figure 55. Courtyard (Upper Right) (from Figure x Lee Kuo-Min Studio)

Figure 56. Kitchen (Middle) (from Figure x Lee Kuo-Min Studio)

Figure 57. Entry (Lower Left) (from Figure x Lee Kuo-Min Studio)

Figure 58. Roof (Lower Center) (from Figure x Lee Kuo-Min Studio)

Figure 59. Stair (Lower Right) (from Figure x Lee Kuo-Min Studio)

TAMEDIA

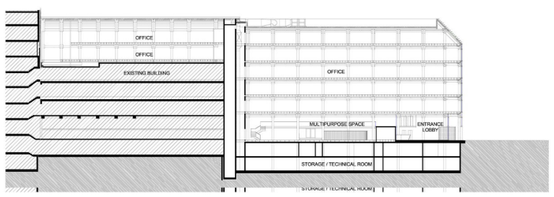
Figure 60. Tamedia Office Building
(from Shigeru Ban Architects)

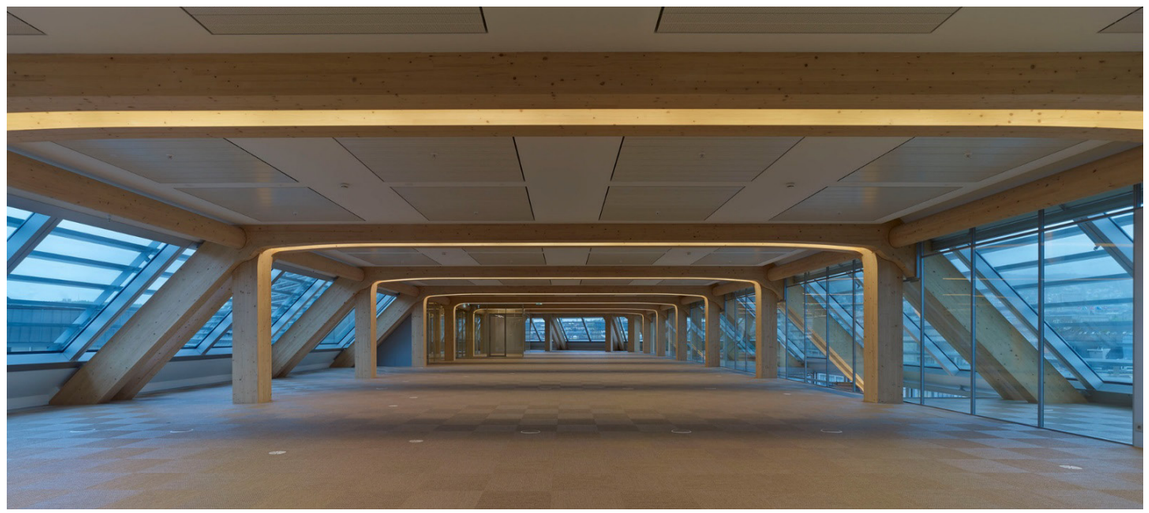
LOCATION:	ZURICH, SWITZERLAND
COMPLETION:	2013
TYPE:	OFFICE
AREA:	8,900 SQ. M
HEIGHT:	7 STORIES
OWNER:	TAMEDIA
ARCHITECT:	SHIGERU BAN ARCHITECTS
STRUCTURAL ENGINEER:	CREATION HOLZ GMBH
WOOD USE:	2,000 CU. M
STRUCTURE OVERVIEW:	GLULAM BEAMS AND COLUMNS WITH CLT FLOOR DECKS

Figure 61. View of Structure (Top) (from Shigeru Ban Architects)

Figure 62. Section and Plan (Middle) (from Shigeru Ban Architects)

Figure 63. View of Upper Floor (Bottom) (from Shigeru Ban Architects)





C13 BUILDING

Figure 64. C13 Building (Left) (from Kaden Klingbeil)

Figure 65. C13 Building Rear Stair Detail (Right) (from Kaden Klingbeil)

LOCATION:	BERLIN, GERMANY
COMPLETION:	2013
TYPE:	FAMILY, EDUCATIONAL, AND HEALTH CENTER
AREA:	2,350 SQ. M
HEIGHT:	7 STORIES
OWNER:	STIFTUNG FÜR CHRISTLICHE BILDUNG
ARCHITECT:	KADEN KLINGBEIL ARCHITEKTEN
STRUCTURAL ENGINEER:	PIRMIN JUNG
STRUCTURE OVERVIEW:	STEEL BEAMS WITH CLT FLOOR DECKS, CONCRETE AND GLULAM COLUMNS, CLT PARTITION WALLS

Figure 66. Exposed CLT in Finished Structure (Left) (from Kaden Klingbeil)

Figure 67. Exterior View (Lower Left) (from Kaden Klingbeil)

Figure 68. Structure During Construction (Lower Right) (from Kaden Klingbeil)

NINA

Figure 69. NINA Headquarters (from Pir II AS)

LOCATION:	TRONDHEIM, NORWAY
COMPLETION:	2013
TYPE:	OFFICE AND LABORATORY
AREA:	8,374 SQ. M
HEIGHT:	5 STORIES
OWNER:	NINA
ARCHITECT:	PIR II AS
STRUCTURE OVERVIEW:	STEEL STRUCTURE WITH CONCRETE CORE AND 170 MM CLT DECK

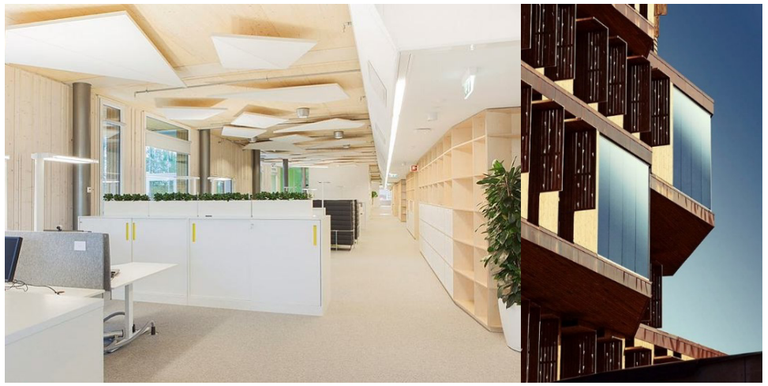


Figure 70. Office Interior (Top Left) (from Pir II AS)

Figure 71. Exterior Panelling Detail (Top Right) (from Pir II AS)

Figure 72. Plan (MIddle) (from Pir II AS)

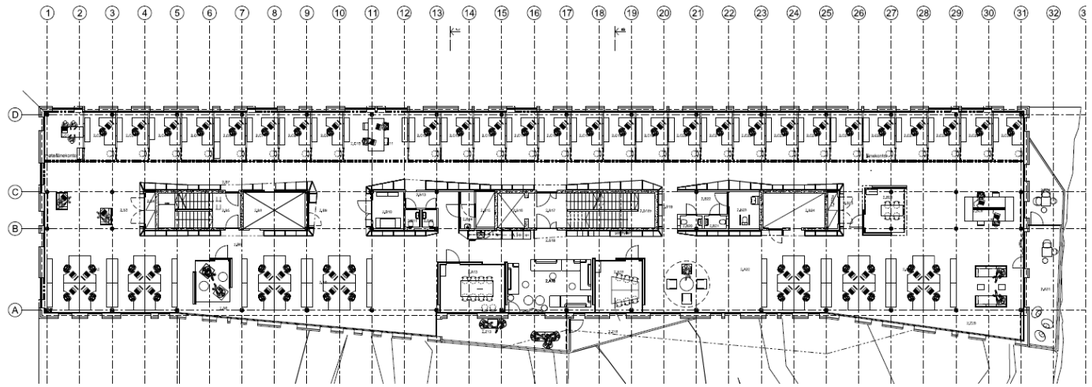


Figure 73. Cafeteria (Bottom) (from Pir II AS)

ILLWERKE ZENTRUM MONTAFON VANDANS

Figure 74. Illwerke Zentrum Montafon Vandans (from Hermann Kaufmann)

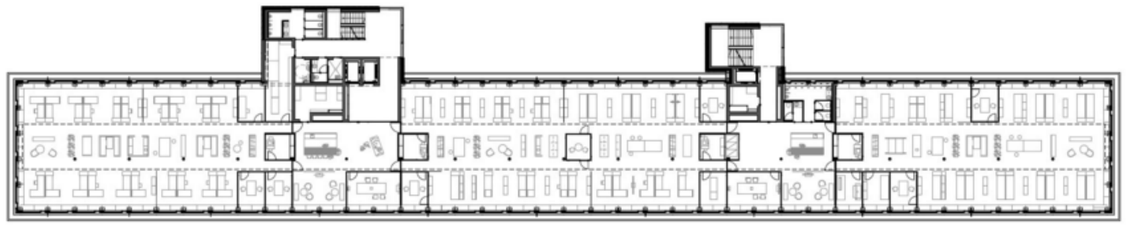
LOCATION:	VANDANS, AUSTRIA
COMPLETION:	2013
TYPE:	OFFICE
AREA:	10,000 SQ. M
HEIGHT:	5 STORIES
OWNER:	VORARLBERGER ILLWERKE AG
ARCHITECT:	HERMANN KAUFMANN
STRUCTURAL ENGINEER:	MERZ KLEY PARTNER
STRUCTURE OVERVIEW:	HYBRID CONCRETE-CLT SLAB SYSTEM WITH CONCRETE AND STEEL COLUMNS

Figure 75. View of Exterior (Left) (from Hermann Kaufmann)

Figure 76. Plan (Middle) (from Hermann Kaufmann)

Figure 77. Office (Lower Left) (from Hermann Kaufmann)

Figure 78. Lobby Atrium (Lower Right) (from Hermann Kaufmann)



SKUTEVIKSBODER 13

Figure 79. Skuteviksboder 13 (from Treteknisk)

LOCATION:	BERGEN, NORWAY
COMPLETION:	2013
TYPE:	OFFICE
AREA:	1,300 SQ. M
HEIGHT:	4 STORIES + LOFT
OWNER:	JAN HARALD HELMICH PEDERSEN
ARCHITECT:	BERTRAM BROCHMAN
WOOD USE:	334 CU. M CLT
STRUCTURE OVERVIEW:	CLT WALLS AND FLOORS, WALLS ARE 95 AND 140 MM THICK, FLOORS ARE 165 MM THICK

Figure 80. Skylight Detail (Upper Left) (from OBAS)

Figure 81. Construction (Upper Right) (from OBAS)

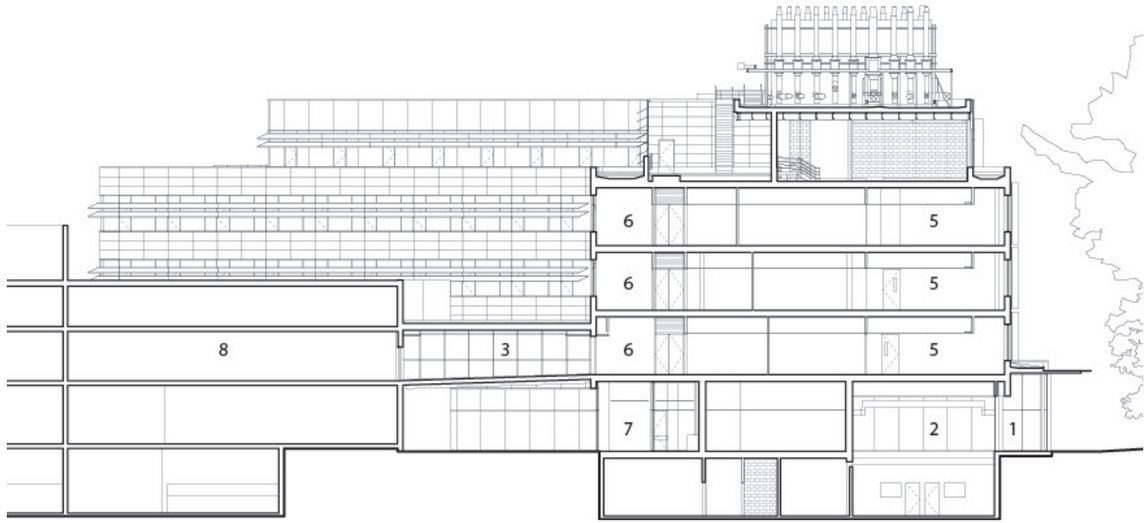
Figure 82. Construction (Lower Left) (from OBAS)

Figure 83. Panel Erection (Lower Right) (from OBAS)

EARTH SCIENCES BUILDING

Figure 84. Earth Sciences Building (from Martin Tessler)

LOCATION:	VANCOUVER, BC, CANADA
COMPLETION:	2012
TYPE:	ACADEMIC
AREA:	15,794 SQ. M
HEIGHT:	5 STORIES
OWNER:	UNIVERSITY OF BRITISH COLUMBIA
ARCHITECT:	PERKINS AND WILL
STRUCTURAL ENGINEER:	EQUILIBRIUM CONSULTING
STRUCTURE OVERVIEW:	GLULAM BEAMS AND COLUMNS WITH CLT FLOOR DECKS IN CLASSROOM WING, CONCRETE IN LABORATORY WING



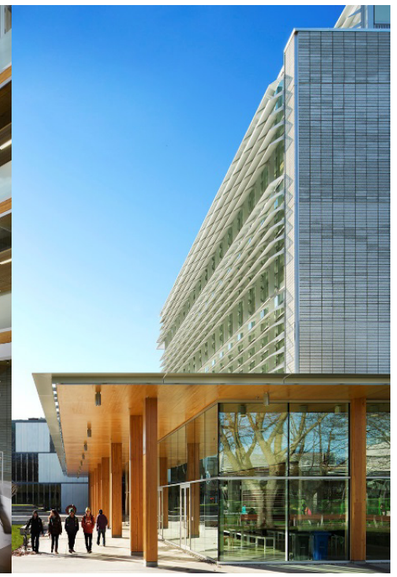


Figure 85. Section (Above) (from Perkins and Will)

Figure 86. Main Atrium (Lower Left) (from Martin Tessler)

Figure 87. View of Exterior (Lower RIght) (from Martin Tessler)

MÜHLEBACHSTRASSE

Figure 88. Mühlebachstrasse (from Kämpen Für Architektur)

LOCATION:	ZURICH, SWITZERLAND
COMPLETION:	2012
TYPE:	OFFICE
HEIGHT:	6 STORIES
ARCHITECT:	KÄMPEN FÜR ARCHITEKTURE
STRUCTURE OVERVIEW:	HYBRID CLT-CONCRETE FLOOR PANELS WITH CLT WALL PANELS, GLULAM COLUMNS, AND LVL BEAMS

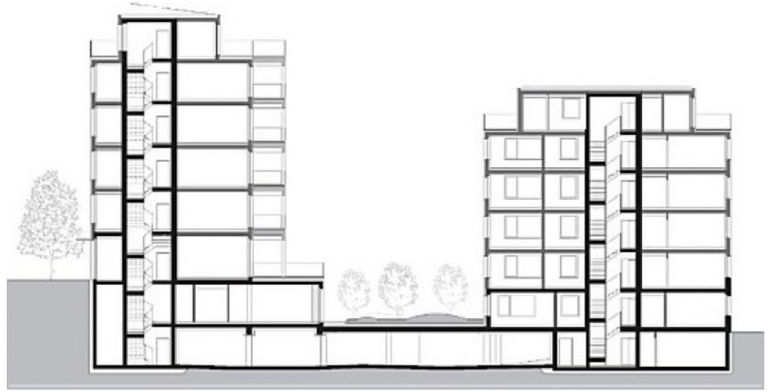


Figure 89. Street Facade (Top Left) (from KF Arch)

Figure 90. Siding Detail (Top Right) (from KF Arch)

Figure 91. Section (Left) (from KF Arch)

Figure 92. Interior (Lower Left) (from KF Arch)

Figure 93. Patio Detail (Lower Right) (from KF Arch)

LCT ONE

Figure 94. LCT One (from Hermann Kaufmann)

LOCATION:	DORNBIRN, AUSTRIA
COMPLETION:	2012
TYPE:	OFFICE
AREA:	2,319 SQ. M
HEIGHT:	8 STORIES
OWNER:	CREE GMBH
ARCHITECT:	HERMANN KAUFMANN ZT GMBH
STRUCTURAL ENGINEER:	ARUP
STRUCTURE OVERVIEW:	PRIMARY STRUCTURAL SYSTEM GLULAM TIMBER AND CONCRETE, HYBRID GLULAM-CONCRETE FLOOR DECK WITH CONCRETE CORE

Figure 95. Exterior Elevation (Upper Left) (from Hermann Kaufmann)

Figure 96. Office (Upper Right) (from Hermann Kaufmann)

Figure 97. Plan (Middle) (from Hermann Kaufmann)

Figure 98. Concrete-Timber Composite Floor Deck Erection (Lower Left) (from Hermann Kaufmann)

Figure 99. Interior (Lower Right) (from Hermann Kaufmann)

ÉDIFICE FONDACTION QUÉBEC

Figure 100. Édifice Fondaction Québec (from GHA)

LOCATION:	QUÉBEC, FRANCE
COMPLETION:	2010
TYPE:	OFFICE
AREA:	6,000 SQ. M
HEIGHT:	6 STORIES
OWNER:	FONDACTION CSN
ARCHITECT:	GILLES HUOT ARCHITECTE
STRUCTURAL ENGINEER:	BES
STRUCTURE OVERVIEW:	ALL TIMBER STRUCTURE WITH GLULAM BEAMS AND COLUMNS AND TIMBER DECKING

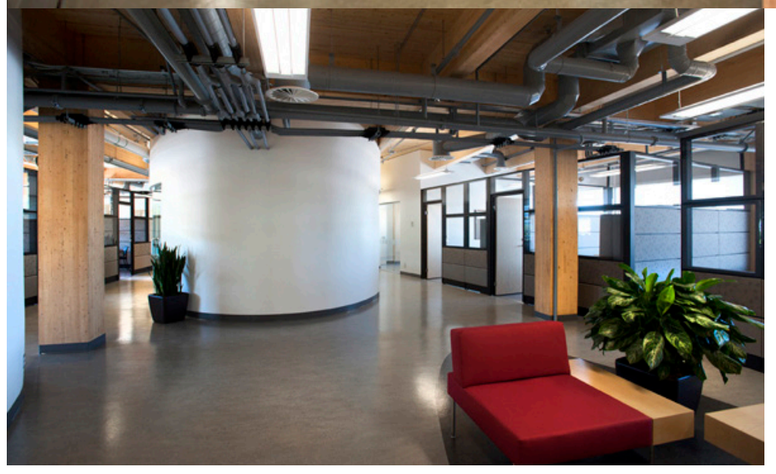


Figure 101. Conference Room (Above) (from GHA)

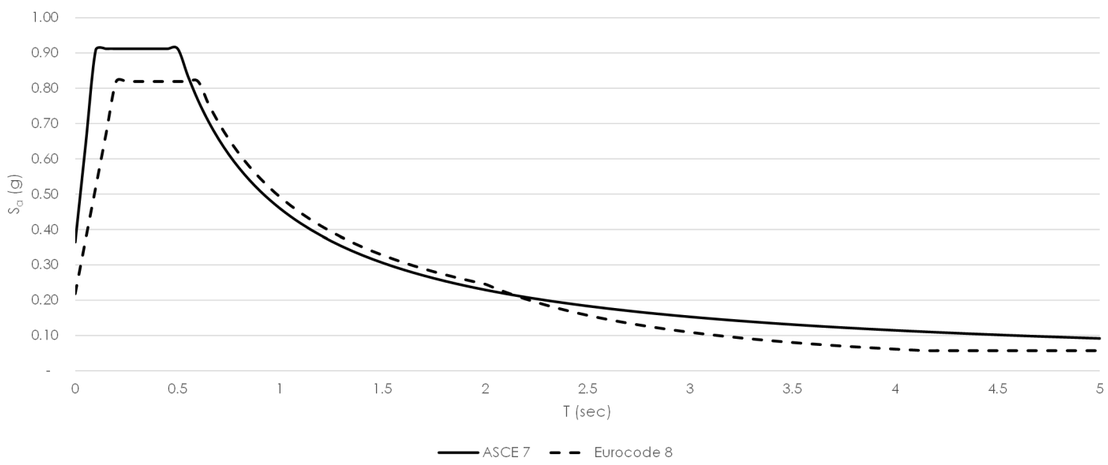
Figure 102. Office Corridor (Left) (from GHA)

SEISMIC CODE COMPARISON

This section contains a comparison of the calculation of base shears using the equivalent lateral force method for a simplified prototypical building in Seattle, Washington. Code provisions in the United States (ASCE 7-10[1]) and Europe (Eurocode 8[2]) are compared to determine if there are significant differences between the codes making the structural design of tall timber buildings more favorable on one continent as compared to the other. The key finding is that results for this case from the two building codes are fairly similar, the building designed to Eurocode 8 has an 11 percent reduction in base shear as compared to the building designed to ASCE 7. A description of key differences in the codes as well as a discussion of challenges follows.

The first challenge to comparing the codes was in defining a consistent seismic hazard. While ASCE 7 defines the response spectra by specifying short period and one second spectral accelerations which are then used to scale the response spectrum curve, Eurocode 8 uses the peak ground acceleration to define the response spectrum curve. For this case all values are taken from USGS data[3][4]. Figure 103 shows the elastic design response spectra for the seismic hazard in Seattle for both codes. Under Eurocode 8, short period structures see a ten percent reduction in accelerations as compared to ASCE 7. In this case, since estimated periods fall outside this range (ASCE 7 - 0.75 sec, Eurocode 8 - 0.77 sec) the spectral acceleration should not be a primary point of variation between the two codes.

Figure 103. Elastic Design Response Spectra



ASCE 7-10			Eurocode 1998		
General Parameters					
Parameter	**Value**	**Reference**	**Parameter**	**Value**	**Reference**
Risk Category	II	[T1.5-1]			
I_e	1	[T1.5-2]			
Number of Stories	11				
Ground Floor Height	4.6 m				
Typical Floor to Floor	3.4 m				
Building Height	38.1 m		H	38.1 m	
Floor Area	557 m^2				
Total Area	6132 m^2				
Seismic Parameters					
Parameter	**Value**	**Reference**	**Parameter**	**Value**	**Reference**
Site Class	C	[11.4.2]	Ground Type	C	[T3.1]
S_S	1.37 g		S	1.15	[T3.2]
S_1	0.53 g		γ_1	1	[3.2.2.1]
S_{DS}	0.91 g		a_{gR}	0.29 g	[3.2.2.1]
S_{D1}	0.46 g		a_g	2.80 g	[3.2.2.1]
T_0	0.10 s		T_B	0.2 s	[T3.2]
T_S	0.50 s		T_C	0.6 s	[T3.2]
T_L	6 s		T_D	2 s	[T3.2]
F_a	1	[T11.4-2]	ξ	5 %	[3.2.2.2]
F_v	1.3	[T11.4-2]	η	1	[EQ3.6]
Seismic Design Category	D	[T11.6-1]	β	0.2	[3.2.2.5]
F_{PGA}	1		S_d	2.10 g	[3.2.2.5]
Structural Parameters					
Parameter	**Value**	**Reference**	**Parameter**	**Value**	**Reference**
System Type	CLT	[11.4.2]			
R	3		q	3	[T8.1]
C_d	2		λ	0.85	[4.3.3.2.2]
Ω_0	1.4				
C_t	0.0488	[T12.8-2]	C_t	0.05	[4.3.3.2.2]
x	0.75				
T_a	0.75 s	[12.8.2.1]	T_1	0.77 s	[EQ4.6]
Equivalent Lateral Force Procedure					
Parameter	**Value**	**Reference**	**Parameter**	**Value**	**Reference**
C_s	0.204	[12.8.1.1]			
W	18,195 kN		m	1,854,700 kg	
V	3,720 kN		F_b	3,305 kN	[EQ4.5]

Table 5. Seismic Design Calculations (yellow values indicate input parameters, white values indicate calculated parameters)

Table 5 shows the calculations for both codes, values that are related between the two codes are shown on the same line. Both codes use an adjustment factor to implicitly account for energy dissipation through inelastic deformation. While CLT shear walls are not currently a code accepted system and therefore R, C_d, and Ω_0 values are not provided in the code, Eurocode 8 gives a q-value of three for timber structures with a high capacity to dissipate energy. This value has also been suggested as an appropriate R-value based on preliminary research. Therefore to maintain consistency, R and q are taken equal to three for purposes of this comparison. This value is not a given and has a linear effect on the total base shear of the building, significantly impacting the structural design.

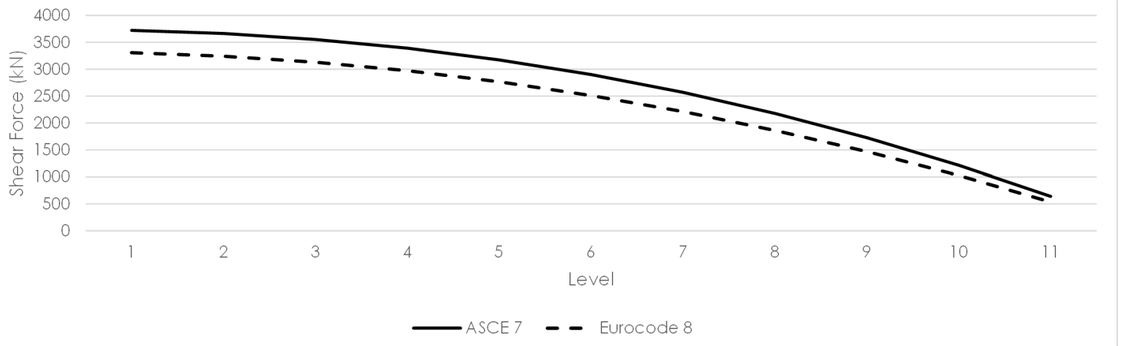


Figure 104. Shear force by level

Based on the analysis, the total base shear for the building designed to ASCE 7 is 3,720 kN and to Eurocode 8 is 3,305 kN. The total shear force by level is shown in Figure 104, both codes provide a similar procedure for distributing shear force by seismic mass at a level therefore producing similar distributions of force over the building.

While based on this analysis both codes provide similar results suggesting that neither code provides significantly more favorable conditions for the design of timber buildings in seismic regions, several key assumptions were made could significantly alter this conclusion.

The first is code acceptance of CLT shear wall systems. While Eurocode explicitly allows the use of CLT as a lateral force resisting system, ASC 7 does not include CLT shear walls as a code approved system. This means that any buildings using CLT shear walls must prove code equivalent performance and are subject to increased testing, more extensive design procedures, and peer review. This significantly increases the cost of design and discourages the use of CLT as a lateral force resisting system. Future inclusion of CLT shear wall systems will considerably mitigate many of the current challenges to design.

The second assumption is related to R- and q-values. While a q-value of three is currently permitted based on Eurocode provisions, no R-value is currently defined in ASCE 7. While this is an area of ongoing research, comparison of R- and q-values for other systems shows that the two codes do not always adopt similar values and this can have a significant effect on base shear calculated using the two codes. Since preliminary research has shown good inelastic behavior for CLT systems[5], an R-value of three does not seem unreasonable; however, establishment of these values will be a key step in the development of CLT lateral systems.

REFERENCES

[1] American Society of Civil Engineers. (2010). ASCE 7-10: Minimum design loads for buildings and other structures. Reston, VA: American Society of Civil Engineers.

[2] European Committee for Standardization. (2004). Eurocode 8: Design of structures for earthquake resistance. Brussels, Belgium: European Committee for Standardization.

[3] US Department of the Interior, & US Geological Survey. (2013). 2008 interactive deaggregations. Retrieved from http://geohazards.usgs.gov/deaggint/2008/

[4] US Department of the Interior, & US Geological Survey. (2014). US seismic design maps. Retrieved from http://earthquake.usgs.gov/designmaps/us/application.php?

[5] Van de Lindt, John W., Rammer, D., Popvski, M., Line, P., Pei, S., & Pryor, S. E. (2013). Lateral design of cross-laminated timber buildings. In R. O. Hamburger, J. D. Dolan, E. Karacabeyli, B. Douglas, T. D. Skaggs & A. Ceccotti (Eds.), CLT handbook (US ed., pp. 119). Pointe-Claire, QC: FP Innovations.

CUSTOM SCRIPT COMPONENTS

C

Appendix C contains a copy of custom C# scripted components written for grasshopper. A series of components were used to build the geometry in grasshopper and to perform the structural design of timber elements. For access to full grasshopper algorithm, contact author.

```
using Rhino.Geometry;
using Grasshopper.Kernel;
using System;
using System.Linq;
using System.Collections.Generic;

namespace Geometry3D
{
    public class Geometry3DComponent : GH_Component
    {
        /// LOCATES COMPONENT IN GRASSHOPPER PANEL
        public Geometry3DComponent()
          : base("Geometry3D", "G3D",
              "Build Frame and Lay Out Shear Walls",
              "Thesis", "Geometry")
        {
        }

        /// REGISTER INPUT PARAMETERS
        protected override void
        RegisterInputParams(GH_Component.GH_InputParamManager pManager)
        {
            pManager.AddNumberParameter("nx", "nx", "Number of X
            Gridlines", GH_ParamAccess.item, 2);
            pManager.AddNumberParameter("ny", "ny", "Number of Y
            Gridlines", GH_ParamAccess.item, 2);
            pManager.AddNumberParameter("nz", "nz", "Number of Levels",
            GH_ParamAccess.item, 1);
            pManager.AddNumberParameter("sx", "sx", "X-Bay Length",
            GH_ParamAccess.item, 20.0);
            pManager.AddNumberParameter("sy", "sy", "Y-Bay Length",
            GH_ParamAccess.item, 20.0);
            pManager.AddNumberParameter("sz0", "sz0", "Ground Level Floor
            to Floor Height", GH_ParamAccess.item, 15.0);
            pManager.AddNumberParameter("sz1", "sz1", "Typical Floor to
            Floor Height", GH_ParamAccess.item, 11.0);
            pManager.AddNumberParameter("cx", "cx", "Core X-Dimension",
            GH_ParamAccess.item, 20.0);
            pManager.AddNumberParameter("cy", "cy", "Core Y-Dimension",
            GH_ParamAccess.item, 20.0);
            pManager.AddNumberParameter("cox", "cox", "Core Offset X",
            GH_ParamAccess.item, 10.0);
            pManager.AddNumberParameter("coy", "coy", "Core Offset Y",
            GH_ParamAccess.item, 10.0);
            pManager.AddNumberParameter("nCoreInt", "nCoreInt", "Number of
            Interior Core Walls", GH_ParamAccess.item, 4);
            pManager.AddNumberParameter("tWall", "tWall", "CLT Shear Wall
            Thickness (in)", GH_ParamAccess.item, 12);
            pManager.AddNumberParameter("Veq", "Veq", "Seismic Base Shear",
            GH_ParamAccess.item, 1000);
            pManager.AddNumberParameter("Cd", "Cd", "ASCE 7 Cd Parameter",
            GH_ParamAccess.item, 3.2);
            pManager.AddNumberParameter("k", "k", "ASCE 7 k Parameter",
            GH_ParamAccess.item, 1.2);
            pManager.AddNumberParameter("EqMassTyp", "EqMassTyp",
            "Earthquake Mass on Typical Floor", GH_ParamAccess.item, 75);
            pManager.AddNumberParameter("EqMassRoof", "EqMassRoof",
            "Earthquake Mass on Roof", GH_ParamAccess.item, 50);
            pManager.AddNumberParameter("LSeg", "LSeg", "Length of CLT
            Shear Walls at Facade", GH_ParamAccess.item, 10);
        }

        /// REGISTER OUTPUT PARAMETERS
        protected override void
        RegisterOutputParams(GH_Component.GH_OutputParamManager pManager)
        {
            pManager.AddCurveParameter("X Beams", "BX", "Beams in
            X-Direction", GH_ParamAccess.list);
            pManager.AddCurveParameter("Y Beams", "BY", "Beams in
            Y-Direction", GH_ParamAccess.list);
            pManager.AddCurveParameter("columns", "C", "Columns",
            GH_ParamAccess.list);
            pManager.AddMeshParameter("FloorSlabs", "FloorSlabs", "Floor
            Slab Meshes", GH_ParamAccess.list);
            pManager.AddMeshParameter("RoofSlab", "RoofSlab", "Roof Slab
            Meshes", GH_ParamAccess.item);
            pManager.AddMeshParameter("CLTWallCore", "CLTWallCore", "CLT
            Core Walls", GH_ParamAccess.list);
            pManager.AddMeshParameter("CLTWallFac", "CLTWallFac", "CLT
            Facade Walls", GH_ParamAccess.list);
            pManager.AddCurveParameter("CLTWallLoc", "CLTWallLoc", "Lines
            with Location of All CLT Walls", GH_ParamAccess.list);
            pManager.AddCurveParameter("WallCol", "WallCol", "Columns at
            CLT Walls", GH_ParamAccess.list);
            pManager.AddNumberParameter("VWall", "VWall", "Base Shear in
            Wall (K)", GH_ParamAccess.list);
            pManager.AddNumberParameter("MWall", "MWall", "Moment at base
            of Wall (K-ft)", GH_ParamAccess.list);
        }

        protected override void SolveInstance(IGH_DataAccess DA)
        {

            ///-----------------------------------------------------------
            ----------------

            ///-----------------------------------------------------------
            ----------------
            /// RETRIEVE INPUT VARIABLES
            ///
            -----------------------------------------------------------
            ------------
            ///
            -----------------------------------------------------------
            ------------
            double nx = 0;
            double ny = 0;
            double nz = 0;
            double sx = 0.0;
            double sy = 0.0;
            double sz0 = 0.0;
            double sz1 = 0.0;
            double cx = 0.0;
            double cy = 0.0;
            double cox = 0.0;
            double coy = 0.0;
```

```
            double nIntCore = 4;
            double tWall = 12;
            double baseshear = 1000;
            double Cd = 3.2;
            double ASCEkEq = 1.2;
            double EMassTyp = 75;
            double EMassRoof = 30;
            double LSeg = 10;

            if (!DA.GetData(0, ref nx)) return;
            if (!DA.GetData(1, ref ny)) return;
            if (!DA.GetData(2, ref nz)) return;
            if (!DA.GetData(3, ref sx)) return;
            if (!DA.GetData(4, ref sy)) return;
            if (!DA.GetData(5, ref sz0)) return;
            if (!DA.GetData(6, ref sz1)) return;
            if (!DA.GetData(7, ref cx)) return;
            if (!DA.GetData(8, ref cy)) return;
            if (!DA.GetData(9, ref cox)) return;
            if (!DA.GetData(10, ref coy)) return;
            if (!DA.GetData(11, ref nIntCore)) return;
            if (!DA.GetData(12, ref tWall)) return;
            if (!DA.GetData(13, ref baseshear)) return;
            if (!DA.GetData(14, ref Cd)) return;
            if (!DA.GetData(15, ref ASCEkEq)) return;
            if (!DA.GetData(16, ref EMassTyp)) return;
            if (!DA.GetData(17, ref EMassRoof)) return;
            if (!DA.GetData(18, ref LSeg)) return;

            ///------------------------------------------------------------
            ---------------

            ///------------------------------------------------------------
            ---------------
            ///INITIALIZE VARIABLES

            ///------------------------------------------------------------
            ---------------

            ///------------------------------------------------------------
            ---------------
            List<Mesh> floorMesh = new List<Mesh>(); //OUTPUT LIST WITH
            FLOOR MESHES
            Mesh roofMesh = new Mesh(); //OUTPUT LIST WITH ROOF MESH
            List<Line> xCoreWalls = new List<Line>();
            List<Line> yCoreWalls = new List<Line>();
            List<Line> xFacWalls = new List<Line>();
            List<Line> yFacWalls = new List<Line>();


            List<double> height = GetHeights(nz, sz0, sz1); //LIST OF FLOOR
            LOCATIONS

            double XDim = (nx - 1) * sx; //BUILDING X-DIMENSION (FT)
            double YDim = (ny - 1) * sy; //BUILDING Y-DIMENSION (FT)

            double kappa = 0.833; //TIMOSHENKO SHEAR COEFFICIENT
```



```
            double G = 50000; //SHEAR MODULUS (psi)
            double E = 1800000; //MODULUS OF ELASTICITY (psi)
            double Cm = 1; //WET SERVICE FACTOR
            double Ct = 1; //TEMPERATURE FACTOR
            double phiComp = 0.9;
            double KComp = 2.4; //FORMAT CONVERSION FACTOR
            double phiShear = 0.75;
            double KShear = 2.88; //FORMAT CONVERSION FACTOR
            double lambda = 1; //TIME EFFECT FACTOR
            double Ie = 1; //EARTHQUAKE IMPORTANCE FACTOR
            double DriftLimit = 0.02; //2% Max Story Drift
            double Fc, Vr;

            double LWallX, LWallY;
            double XCOG, YCOG;
            double XOff, YOff;
            double Stiff1X, Stiff1Y;
            double StiffTypX, StiffTypY;
            double kXoffX = 0;
            double kYoffY = 0;
            double Inertia;
            string WallDir;

            double XCOR, YCOR; //CENTER OF ROTATION
            double AccTorX, AccTorY; //Accidental Torsion

            List<double> ListWallShear = new List<double>();
            List<double> ListWallMoment = new List<double>();
            List<Line> AllWalls = new List<Line>();

            //WALL PROPERTIES BY THICKNESS FROM STRUCTURLAM
            if (tWall == 3.9)
            {
                Fc = 1150; //psi
                Vr = 2906; //lb/ft
            }
            else if (tWall == 6.66)
            {
                Fc = 1150; //psi
                Vr = 5812; //lb/ft
            }
            else if (tWall == 9.42)
            {
                Fc = 1150; //psi
                Vr = 8718; //lb/ft
            }
            else
            {
                Fc = 1150; //psi
                Vr = 11624; //lb/ft
            }

            //TEMPORARY VARIABLES
            Mesh tempMesh;
            Point3d tempPt1, tempPt2, tempPt3, tempPt4;
            double temp1, temp2;
            List<double> temp3 = new List<double>();
            double Fail = 0;
```

```
///------------------------------------------------------------
----------------

///------------------------------------------------------------
----------------
///BUILD FLOOR SLABS

///------------------------------------------------------------
----------------

///------------------------------------------------------------
----------------
for (int i = 0; i < nz; i++)
{
    tempMesh = BuildMesh(new Point3d(0, 0, height[i + 1]), new
    Point3d(0, YDim, height[i + 1]), new Point3d(XDim, YDim,
    height[i + 1]), new Point3d(XDim, 0, height[i + 1]));
    if (i != (nz - 1))
    {
        floorMesh.Add(tempMesh);
    }
    else
    {
        roofMesh = tempMesh;
    }
}
AreaMassProperties amp = AreaMassProperties.Compute(roofMesh);
Point3d Centroid = amp.Centroid;

///------------------------------------------------------------
----------------

///------------------------------------------------------------
----------------
///DESIGN SHEAR WALLS

///------------------------------------------------------------
----------------

///------------------------------------------------------------
----------------

//DRAW PERIMETER CORE WALLS
tempPt1 = new Point3d(cox, coy, 0);
tempPt2 = new Point3d(cox + cx, coy, 0);
tempPt3 = new Point3d(cox, coy + cy, 0);
tempPt4 = new Point3d(cox + cx, coy + cy, 0);

xCoreWalls.Add(new Line(tempPt1, tempPt2));
xCoreWalls.Add(new Line(tempPt3, tempPt4));
yCoreWalls.Add(new Line(tempPt1, tempPt3));
yCoreWalls.Add(new Line(tempPt2, tempPt4));

//DRAW INTERIOR CORE WALLS
double gap = cy / (nIntCore + 1);
```



```
for (int i = 0; i < nIntCore; i++)
{
    tempPt1 = new Point3d(cox, coy + (1 + i) * gap, 0);
    tempPt2 = new Point3d(cox + cx, coy + (1 + i) * gap, 0);
    xCoreWalls.Add(new Line(tempPt1, tempPt2));
}

//CALCULATE STORY SHEARS TO FIND MOMENT ARM FOR SEISMIC FORCE
SeismicForces EqForce = new SeismicForces();
EqForce.CalculateSeismicForces(height, EMassTyp, EMassRoof,
amp.Area, baseshear, ASCEkEq);

//CHECK WALLS AND ADD PERIMETER WALLS UNTIL WALLS PASS
double nXfacWalls = 0;
double nYfacWalls = 0;
int loopCount = 0;

while (Fail < 3)
{
    AllWalls = new List<Line>();
    ListWallShear = new List<double>();
    ListWallMoment = new List<double>();

    List<double> ListLWallX = new List<double>();
    List<double> ListLWallY = new List<double>();
    List<double> ListXCOG = new List<double>();
    List<double> ListYCOG = new List<double>();
    List<double> ListXOff = new List<double>();
    List<double> ListYOff = new List<double>();
    List<double> ListStiff1X = new List<double>();
    List<double> ListStiff1Y = new List<double>();
    List<double> ListStiffTypX = new List<double>();
    List<double> ListStiffTypY = new List<double>();
    List<string> ListWallDir = new List<string>();


    foreach (Line v in xCoreWalls)
    {
        AllWalls.Add(v);
    }
    foreach (Line v in yCoreWalls)
    {
        AllWalls.Add(v);
    }
    foreach (Line v in xFacWalls)
    {
        AllWalls.Add(v);
    }
    foreach (Line v in yFacWalls)
    {
        AllWalls.Add(v);
    }

    for (int i = 0; i < AllWalls.Count; i++)
    {
        //FIND WALL LENGTH FROM ENDPOINTS
        LWallX = Math.Abs(AllWalls[i].ToX - AllWalls[i].FromX);
```

```
                //ft
280             ListLWallX.Add(LWallX); //ft
281             LWallY = Math.Abs(AllWalls[i].ToY - AllWalls[i].FromY);
                //ft
282             ListLWallY.Add(LWallY); //ft
283
284             //FIND WALL CENTER OF GRAVITY
285             XCOG = LWallX / 2 + AllWalls[i].FromX; //ft
286             ListXCOG.Add(XCOG); //ft
287             YCOG = LWallY / 2 + AllWalls[i].FromY; //ft
288             ListYCOG.Add(YCOG); //ft
289
290             //FIND WALL OFFSET FROM BUILDING CENTER OF MASS
291             XOff = XCOG - Centroid.X; //ft
292             ListXOff.Add(XOff); //ft
293             YOff = YCOG - Centroid.Y; //ft
294             ListYOff.Add(YOff); //ft
295
296             //FIND DIRECTION OF WALL
297             if (LWallX < 0.1)
298             {
299                 WallDir = "y";
300             }
301             else
302             {
303                 WallDir = "x";
304             }
305             ListWallDir.Add(WallDir);
306
307             //CALCULATE WALL STIFFNESS
308             if (WallDir == "x")
309             {
310                 Stiff1Y = 0;
311                 StiffTypY = 0;
312                 Inertia = tWall * Math.Pow(LWallX * 12, 3) / 12;
                    //in4
313                 Stiff1X = 1 / ((Math.Pow(sz0 * 12, 3) / (3 * E *
                    Inertia)) + (sz0 * 12 / (kappa * tWall * LWallX *
                    12 * G))); //lb/in
314                 StiffTypX = 1 / ((Math.Pow(sz1 * 12, 3) / (3 * E *
                    Inertia)) + (sz1 * 12 / (kappa * tWall * LWallX *
                    12 * G))); //lb/in
315
316             }
317             else
318             {
319                 Stiff1X = 0;
320                 StiffTypX = 0;
321                 Inertia = tWall * Math.Pow(LWallY * 12, 3) / 12;
                    //in4
322                 temp1 = Math.Pow(sz0 * 12, 3) / (3 * E * Inertia);
323                 temp2 = sz0 * 12 / (kappa * tWall * LWallY * 12 * G);
324                 Stiff1Y = 1 / (temp1 + temp2); //lb/in
325                 temp1 = Math.Pow(sz1 * 12, 3) / (3 * E * Inertia);
326                 temp2 = sz1 * 12 / (kappa * tWall * LWallY * 12 * G);
327                 StiffTypY = 1 / (temp1 + temp2); //lb/in
328             }
329             ListStiff1Y.Add(Stiff1Y);
```



```
330             ListStiffTypY.Add(StiffTypY);
331             ListStiff1X.Add(Stiff1X);
332             ListStiffTypX.Add(StiffTypX);
333
334             kXoffX = kXoffX + Stiff1X * XOff * 12; //lb
335             kYoffY = kYoffY + Stiff1Y * YOff * 12; //lb
336         }//END LOOP OVER EACH WALL
337
338         XCOR = kXoffX / (12 * ListStiff1X.Sum()); //ft
339         YCOR = kYoffY / (12 * ListStiff1Y.Sum()); //ft
340         AccTorX = Math.Max(0.05, Math.Abs(XCOR) / XDim);
341         AccTorY = Math.Max(0.05, Math.Abs(YCOR) / YDim);
342
343         //CHECK WALLS FOR FORCES
344         double WallShear, WallMoment;
345         double ShearPerLengthX, ShearPerLengthY, BendingStressX,
            BendingStressY;
346         double SWall; //Section Modulus
347         List<double> ShearDCX = new List<double>();
348         List<double> ShearDCY = new List<double>();
349         List<double> BendDCX = new List<double>();
350         List<double> BendDCY = new List<double>();
351         List<int> WallPassX = new List<int>();
352         List<int> WallPassY = new List<int>();
353
354         for (int i = 0; i < AllWalls.Count; i++)
355         {
356             if (ListWallDir[i] == "x")
357             {
358                 WallShear = baseshear * 1000 * ListStiff1X[i] * (1
                    + AccTorX * 2) / ListStiff1X.Sum(); //lb
359                 ListWallShear.Add(WallShear / 1000);
360                 WallMoment = WallShear * EqForce.MomentArm * 12;
                    //lb-in
361                 ListWallMoment.Add(WallMoment / (1000 * 12));
362                 ShearPerLengthX = WallShear / ListLWallX[i]; //lb/ft
363                 SWall = tWall * Math.Pow((ListLWallX[i] * 12), 2) /
                    6; //in^3
364                 BendingStressX = WallMoment / SWall; //psi
365
366                 ShearDCX.Add(ShearPerLengthX / (phiShear * KShear *
                    Vr * Cm * Ct * lambda));
367                 BendDCX.Add(BendingStressX / (phiComp * KComp * Fc
                    * Cm * Ct * lambda));
368                 if (ShearDCX.Max() > 1)
369                 {
370                     WallPassX.Add(0); //WALL FAILS
371                 }
372                 else
373                 {
374                     if (BendDCX.Max() > 1)
375                     {
376                         WallPassX.Add(0); //WALL FAILS
377                     }
378                     else
379                     {
380                         WallPassX.Add(1); //WALL PASSES
381                     }
```

```
            }
        }
        else
        {
            WallShear = baseshear * 1000 * ListStiff1Y[i] * (1
            + AccTorY * 2) / ListStiff1Y.Sum(); //lb
            ListWallShear.Add(WallShear / 1000);
            WallMoment = WallShear * EqForce.MomentArm * 12;
            //lb-in
            ListWallMoment.Add(WallMoment / (1000 * 12));
            ShearPerLengthY = WallShear / ListLWallY[i]; //lb/ft
            SWall = tWall * Math.Pow(ListLWallY[i] * 12, 2) /
            6; //in^3
            BendingStressY = WallMoment / SWall; //psi

            ShearDCY.Add(ShearPerLengthY / (phiShear * KShear *
            Vr * Cm * Ct * lambda));
            BendDCY.Add(BendingStressY / (phiComp * KComp * Fc
            * Cm * Ct * lambda));
            temp3.Add(ShearDCX.Max());
            temp3.Add(ShearDCY.Max());
            temp3.Add(BendDCX.Max());
            temp3.Add(BendDCY.Max());
            if (ShearDCY.Max() > 1)
            {
                WallPassY.Add(0); //WALL FAILS
            }
            else
            {
                if (BendDCY.Max() > 1)
                {
                    WallPassY.Add(0); //WALL FAILS
                }
                else
                {
                    WallPassY.Add(1); //WALL PASSES
                }
            }
        }
    }//END LOOP OVER EACH WALL

    //LOOP OVER FLOORS TO CHECK DRIFTS
    List<double> StoryDispX = new List<double>();
    List<double> StoryDispY = new List<double>();
    List<double> StoryDriftX = new List<double>();
    List<double> StoryDriftY = new List<double>();
    List<double> DispX = new List<double>();
    List<double> DispY = new List<double>();
    List<double> DriftX = new List<double>();
    List<double> DriftY = new List<double>();

    for (int i = 0; i < nz; i++)
    {
        if (i == 0)
        {
            //FIRST FLOOR
            StoryDispX.Add(EqForce.StoryShear[i] * 1000 * (1 +
            2 * AccTorX) * Cd / (ListStiff1X.Sum() * Ie));
```



```
            StoryDispY.Add(EqForce.StoryShear[i] * 1000 * (1 +
            2 * AccTorY) * Cd / ListStiff1Y.Sum() * Ie);
            StoryDriftX.Add(StoryDispX[i] / (sz0 * 12));
            StoryDriftY.Add(StoryDispY[i] / (sz0 * 12));
            DispX.Add(StoryDispX[i]);
            DispY.Add(StoryDispY[i]);
            DriftX.Add(StoryDriftX[i]);
            DriftY.Add(StoryDriftY[i]);
        }
        else
        {
            StoryDispX.Add(EqForce.StoryShear[i] * 1000 * (1 +
            2 * AccTorX) * Cd / ListStiffTypX.Sum() * Ie);
            StoryDispY.Add(EqForce.StoryShear[i] * 1000 * (1 +
            2 * AccTorY) * Cd / ListStiffTypY.Sum() * Ie);
            StoryDriftX.Add(StoryDispX[i] / (sz1 * 12));
            StoryDriftY.Add(StoryDispY[i] / (sz1 * 12));
            DispX.Add(StoryDispX[i] + DispX[i - 1]);
            DispY.Add(StoryDispY[i] + DispY[i - 1]);
            DriftX.Add(DispX[i] / (height[i + 1] * 12));
            DriftY.Add(DispY[i] / (height[i + 1] * 12));
        }
    }

    double p, q, r, s;
    p = DriftX.Max();
    q = DriftY.Max();
    r = StoryDriftX.Max();
    s = StoryDriftY.Max();
    List<double> DriftMaxX = new List<double>();
    List<double> DriftMaxY = new List<double>();
    DriftMaxX.Add(p);
    DriftMaxY.Add(q);
    DriftMaxX.Add(r);
    DriftMaxY.Add(s);

    double DriftMax = Math.Max(DriftMaxX.Max(), DriftMaxY.Max());
    double WallPass = Math.Min(WallPassX.Min(), WallPassY.Min());

    if (DriftMax > DriftLimit)
    {
        Fail = 0;
    }
    else
    {
        if (WallPass < 1)
        {
            Fail = 0;
        }
        else
        {
            Fail = 3;
        }
    }

    loopCount = loopCount + 1;
    if (loopCount > 15)
    {
```

```
            break;
        }

        //ADD ONE X-WALL IF X WALLS FAIL
        temp3.Add(DriftMaxX.Max());
        temp3.Add(DriftMaxY.Max());
        if (DriftMaxX.Max() > DriftLimit)
        {
            nXfacWalls = nXfacWalls + 1;
        }
        else
        {
            if (WallPassX.Min() < 1)
            {
                nXfacWalls = nXfacWalls + 1;
            }
            else { }
        }

        //ADD ONE Y-WALL IF Y WALLS FAIL
        if (DriftMaxY.Max() > DriftLimit)
        {
            nYfacWalls = nYfacWalls + 1;
        }
        else
        {
            if (WallPassY.Min() < 1)
            {
                nYfacWalls = nYfacWalls + 1;
            }
            else { }
        }

        //BUILD FACADE WALL (LINE)
        double Xwall = nXfacWalls * LSeg;
        double Ywall = nYfacWalls * LSeg;

        double NXsp = nXfacWalls - 1;
        double NYsp = nYfacWalls - 1;

        double LXsp = (XDim - nXfacWalls * LSeg) / NXsp;
        double LYsp = (YDim - nYfacWalls * LSeg) / NYsp;

        //X-Facade Walls
        if (Xwall > (XDim))
        {
            xFacWalls = new List<Line>();
            //Full wall is Solid
            tempPt1 = new Point3d(0, 0, 0);
            tempPt2 = new Point3d(XDim, 0, 0);
            tempPt3 = new Point3d(0, YDim, 0);
            tempPt4 = new Point3d(XDim, YDim, 0);
            xFacWalls.Add(new Line(tempPt1, tempPt2));
            xFacWalls.Add(new Line(tempPt3, tempPt4));
        }
        else if (Xwall != 0)
        {
            //Distribute wall segments of length Seg along facade
```



```
            xFacWalls = new List<Line>();
            for (int i = 0; i < nXfacWalls; i++)
            {
                if (nXfacWalls != 1)
                {
                    tempPt1 = new Point3d(i * LSeg + i * LXsp, 0, 0);
                    tempPt2 = new Point3d((i + 1) * LSeg + i *
                    LXsp, 0, 0);
                    tempPt3 = new Point3d(i * LSeg + i * LXsp,
                    YDim, 0);
                    tempPt4 = new Point3d((i + 1) * LSeg + i *
                    LXsp, YDim, 0);
                    xFacWalls.Add(new Line(tempPt1, tempPt2));
                    xFacWalls.Add(new Line(tempPt3, tempPt4));
                }
                else
                {
                    tempPt1 = new Point3d(0.5 * XDim - LSeg / 2, 0,
                    0);
                    tempPt2 = new Point3d(0.5 * XDim + LSeg / 2, 0,
                    0);
                    tempPt3 = new Point3d(0.5 * XDim - LSeg / 2,
                    YDim, 0);
                    tempPt4 = new Point3d(0.5 * XDim + LSeg / 2,
                    YDim, 0);
                    xFacWalls.Add(new Line(tempPt1, tempPt2));
                    xFacWalls.Add(new Line(tempPt3, tempPt4));
                }
            }
        }

        //Y-Facade Walls
        if (Ywall > (YDim))
        {
            yFacWalls = new List<Line>();
            //Full wall is Solid
            tempPt1 = new Point3d(0, 0, 0);
            tempPt2 = new Point3d(0, YDim, 0);
            tempPt3 = new Point3d(XDim, 0, 0);
            tempPt4 = new Point3d(XDim, YDim, 0);
            yFacWalls.Add(new Line(tempPt1, tempPt2));
            yFacWalls.Add(new Line(tempPt3, tempPt4));
        }
        else if (Ywall != 0)
        {
            yFacWalls = new List<Line>();
            //Distribute wall segments of length Seg along facade
            for (int i = 0; i < nYfacWalls; i++)
            {
                if (nYfacWalls != 1)
                {
                    tempPt1 = new Point3d(0, i * LSeg + i * LYsp, 0);
                    tempPt2 = new Point3d(0, (i + 1) * LSeg + i *
                    LYsp, 0);
                    tempPt3 = new Point3d(XDim, i * LSeg + i *
                    LYsp, 0);
                    tempPt4 = new Point3d(XDim, (i + 1) * LSeg + i
                    * LYsp, 0);
```

```
                    yFacWalls.Add(new Line(tempPt1, tempPt2));
                    yFacWalls.Add(new Line(tempPt3, tempPt4));
                }
                else
                {
                    tempPt1 = new Point3d(0, 0.5 * YDim - LSeg / 2,
                    0);
                    tempPt2 = new Point3d(0, 0.5 * YDim + LSeg / 2,
                    0);
                    tempPt3 = new Point3d(XDim, 0.5 * YDim - LSeg /
                    2, 0);
                    tempPt4 = new Point3d(XDim, 0.5 * YDim + LSeg /
                    2, 0);
                    yFacWalls.Add(new Line(tempPt1, tempPt2));
                    yFacWalls.Add(new Line(tempPt3, tempPt4));
                }
            }
        }
    }//END WHILE LOOP TO DESIGN SHEAR WALLS

    ///-----------------------------------------------------------------
    ---------------

    ///-----------------------------------------------------------------
    ---------------
    ///BUILD FRAME OF BEAM AND COLUMN ELEMENTS

    ///-----------------------------------------------------------------
    ---------------

    ///-----------------------------------------------------------------
    ---------------

    ///Initialize Variables
    List<Point3d> nodes = new List<Point3d>();
    List<LineCurve> beamsX = new List<LineCurve>();
    List<LineCurve> beamsY = new List<LineCurve>();
    List<LineCurve> columnsZ = new List<LineCurve>();
    List<LineCurve> WallColumn = new List<LineCurve>();

    double xx, yy; ///Point Coordinates
    double xx1, yy1; ///Adjacent Point Coordinates
    double cxmin, cxmax, cymin, cymax; ///Core Boundaries
    double xx0, yy0; ///Coordinate Overwrites
    double xx2, yy2; //Adjacent Coordinate Overwrites
    double xx3, yy3, xx4, yy4; //Interior Coordinate Overwrites
    bool ptin, pt1xin, pt1yin, bmXin, bmYin;
    Point3d node, node1x, node1y, node1z, node0z;

    ///Define Core Wall Boundaries
    cxmin = cox;
    cxmax = cox + cx;
    cymin = coy;
    cymax = coy + cy;

    ///Loop X Coordinates
    for (double i = 1.0; i <= nx; i += 1.0)
```



```
    {
        ///Loop Y Coordinates
        for (double j = 1.0; j <= ny; j += 1.0)
        {
            ///Loop Z Coordinates
            for (int k = 0; k <= nz; k++)
            {
                ///Node Coordinate X & Y
                xx = (i - 1.0) * sx;
                yy = (j - 1.0) * sy;

                ///Adjacent Coordinate X & Y
                xx1 = i * sx;
                yy1 = j * sy;

                ///Check if coordinates are in or out of core
                ptin = false;
                pt1xin = false;
                pt1yin = false;
                bmXin = false;
                bmYin = false;

                ///Point
                if (xx <= cxmax && xx >= cxmin && yy <= cymax && yy
                >= cymin)
                {
                    ptin = true;
                }
                ///Next Point X
                if (xx1 <= cxmax && xx1 >= cxmin && yy <= cymax &&
                yy >= cymin)
                {
                    pt1xin = true;
                }
                ///Next Point Y
                if (yy1 <= cymax && yy1 >= cymin && xx <= cxmax &&
                xx >= cxmin)
                {
                    pt1yin = true;
                }
                ///Beam Crosses Core
                if (xx < cxmin && xx1 > cxmax && yy <= cymax && yy
                >= cymin)
                {
                    bmXin = true;
                }
                if (yy < cymin && yy1 > cymax && xx <= cxmax && xx
                >= cxmin)
                {
                    bmYin = true;
                }


                //CHECK IF POINT IS ON WALL
                tempPt1 = new Point3d(xx, yy, 0);
                tempPt2 = new Point3d(xx1, yy, 0);
                tempPt3 = new Point3d(xx, yy1, 0);
                bool PtOnLine = false;
```

```
List<Line> OnWall = new List<Line>();
List<Line> X1OnWall = new List<Line>();
List<Line> Y1OnWall = new List<Line>();

foreach (Line v in AllWalls)
{
    if (v.MinimumDistanceTo(tempPt1) < 0.1)
    {
        PtOnLine = true;
        OnWall.Add(v);
    }
    if (v.MinimumDistanceTo(tempPt2) < 0.1)
    {
        X1OnWall.Add(v);
    }
    if (v.MinimumDistanceTo(tempPt3) < 0.1)
    {
        Y1OnWall.Add(v);
    }
}

///Draw X Beams
if (i == nx) { }
else if (j == 1)
{
    if (k != 0)
    {
        //FIRST PERIMETER
        if (nXfacWalls == 0)
        {
            node = new Point3d(xx, yy, height[k]);
            node1x = new Point3d(xx1, yy, height[k]);
            beamsX.Add(new LineCurve(node, node1x));
        }
        else
        {
            xx0 = xx;
            foreach (Line v in OnWall)
            {
                if (v.FromX > xx0)
                {
                    xx0 = v.FromX;
                }
                if (v.ToX > xx0)
                {
                    xx0 = v.ToX;
                }
            }
            if (xx0 >= xx1)
            {
                //WALL ALONG ENTIRE BAY, NO BEAM
            }
            else
            {
                xx2 = xx1;
                foreach (Line v in X1OnWall)
                {
                    if (v.FromX < xx2)
```



```
                    {
                        xx2 = v.FromX;
                    }
                    if (v.ToX < xx2)
                    {
                        xx2 = v.ToX;
                    }
                }
                bool wallBtwn = false;
                xx3 = 0;
                xx4 = 0;
                foreach (Line v in AllWalls)
                {
                    if (v.FromY == v.ToY && v.FromY
                    == yy)
                    {
                        if (v.FromX > xx && v.ToX <
                        xx1)
                        {
                            xx3 = v.FromX;
                            xx4 = v.ToX;
                            wallBtwn = true;
                        }
                    }
                }
                if (wallBtwn == true)
                {
                    //WALL BETWEEN POINTS
                    node = new Point3d(xx0, yy,
                    height[k]);
                    node1x = new Point3d(xx3, yy,
                    height[k]);
                    beamsX.Add(new LineCurve(node,
                    node1x));
                    node = new Point3d(xx4, yy,
                    height[k]);
                    node1x = new Point3d(xx2, yy,
                    height[k]);
                    beamsX.Add(new LineCurve(node,
                    node1x));
                }
                else
                {
                    node = new Point3d(xx0, yy,
                    height[k]);
                    node1x = new Point3d(xx2, yy,
                    height[k]);
                    beamsX.Add(new LineCurve(node,
                    node1x));
                }
            }
        }
    }
    else if (j == ny)
    {
        if (k != 0)
        {
```

```
//SECOND PERIMETER
if (nXfacWalls == 0)
{
    node = new Point3d(xx, yy, height[k]);
    node1x = new Point3d(xx1, yy, height[k]);
    beamsX.Add(new LineCurve(node, node1x));
}
else
{
    xx0 = xx;
    foreach (Line v in OnWall)
    {
        if (v.FromX > xx0)
        {
            xx0 = v.FromX;
        }
        if (v.ToX > xx0)
        {
            xx0 = v.ToX;
        }
    }
    if (xx0 >= xx1)
    {
        //WALL ALONG ENTIRE BAY, NO BEAM
    }
    else
    {
        xx2 = xx1;
        foreach (Line v in X1OnWall)
        {
            if (v.FromX < xx2)
            {
                xx2 = v.FromX;
            }
            if (v.ToX < xx2)
            {
                xx2 = v.ToX;
            }
        }
        bool wallBtwn = false;
        xx3 = 0;
        xx4 = 0;
        foreach (Line v in AllWalls)
        {
            if (v.FromY == v.ToY && v.FromY
            == yy)
            {
                if (v.FromX > xx && v.ToX <
                xx1)
                {
                    xx3 = v.FromX;
                    xx4 = v.ToX;
                    wallBtwn = true;
                }
            }
        }
        if (wallBtwn == true)
        {
```



```
                //WALL BETWEEN POINTS
                node = new Point3d(xx0, yy,
                height[k]);
                node1x = new Point3d(xx3, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
                node = new Point3d(xx4, yy,
                height[k]);
                node1x = new Point3d(xx2, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
            }
            else
            {
                node = new Point3d(xx0, yy,
                height[k]);
                node1x = new Point3d(xx2, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
            }
        }

    }
}
else
{
    if (k != 0)
    {
        if (ptin == true && pt1xin == true)
        {
            ///No X Beam
        }
        else if (ptin == true && pt1xin == false)
        {
            xx0 = cxmax;
            node = new Point3d(xx0, yy, height[k]);
            node1x = new Point3d(xx1, yy, height[k]);
            beamsX.Add(new LineCurve(node, node1x));

            if (k != nz)
            {
                if (k == 1)
                {
                    node0z = new Point3d(xx0, yy,
                    height[k - 1]);
                    node1z = new Point3d(xx0, yy,
                    height[k]);
                    WallColumn.Add(new
                    LineCurve(node0z, node1z));
                    nodes.Add(node0z);
                }
                node0z = new Point3d(xx0, yy,
                height[k]);
                node1z = new Point3d(xx0, yy,
```

```
                height[k + 1]);
                WallColumn.Add(new
                LineCurve(node0z, node1z));
                nodes.Add(node0z);
            }
        }
        else if (ptin == false && pt1xin == true)
        {
            xx1 = cxmin;
            node = new Point3d(xx, yy, height[k]);
            node1x = new Point3d(xx1, yy, height[k]);
            beamsX.Add(new LineCurve(node, node1x));

            if (k != nz)
            {
                if (k == 1)
                {
                    node0z = new Point3d(xx1, yy,
                    height[k - 1]);
                    node1z = new Point3d(xx1, yy,
                    height[k]);
                    WallColumn.Add(new
                    LineCurve(node0z, node1z));
                    nodes.Add(node0z);
                }
                node0z = new Point3d(xx1, yy,
                height[k]);
                node1z = new Point3d(xx1, yy,
                height[k + 1]);
                WallColumn.Add(new
                LineCurve(node0z, node1z));
                nodes.Add(node0z);
            }
        }
        else if (ptin == false && pt1xin == false)
        {
            if (bmXin == true)
            {
                node = new Point3d(xx, yy,
                height[k]);
                node1x = new Point3d(cxmin, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
                nodes.Add(new Point3d(cxmin, yy, 0));
                node = new Point3d(cxmax, yy,
                height[k]);
                node1x = new Point3d(xx1, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
                nodes.Add(new Point3d(cxmax, yy, 0));
            }
            else
            {
                node = new Point3d(xx, yy,
                height[k]);
                node1x = new Point3d(xx1, yy,
                height[k]);
                beamsX.Add(new LineCurve(node,
                node1x));
            }
        }
    }

    ///Draw Y Beams
    if (j == ny) { }
    else if (i == 1)
    {
        if (k != 0)
        {
            //FIRST PERIMETER
            if (nYfacWalls == 0)
            {
                node = new Point3d(xx, yy, height[k]);
                node1y = new Point3d(xx, yy1, height[k]);
                beamsY.Add(new LineCurve(node, node1y));
            }
            else
            {
                yy0 = yy;
                foreach (Line v in OnWall)
                {
                    if (v.FromY > yy0)
                    {
                        yy0 = v.FromY;
                    }
                    if (v.ToY > yy0)
                    {
                        yy0 = v.ToY;
                    }
                }
                if (yy0 >= yy1)
                {
                    //WALL ALONG ENTIRE BAY, NO BEAM
                }
                else
                {
                    yy2 = yy1;
                    foreach (Line v in Y1OnWall)
                    {
                        if (v.FromY < yy2)
                        {
                            yy2 = v.FromY;
                        }
                        if (v.ToY < yy2)
                        {
                            yy2 = v.ToY;
                        }
                    }
                    bool wallBtwn = false;
                    yy3 = 0;
                    yy4 = 0;
                    foreach (Line v in AllWalls)
                    {
```

```
                    if (v.FromX == v.ToX && v.FromY
                    == xx)
                    {
                        if (v.FromY > yy && v.ToY <
                        yy1)
                        {
                            yy3 = v.FromY;
                            yy4 = v.ToY;
                            wallBtwn = true;
                        }
                    }
                }
                if (wallBtwn == true)
                {
                    //WALL BETWEEN POINTS
                    node = new Point3d(xx, yy0,
                    height[k]);
                    node1y = new Point3d(xx, yy3,
                    height[k]);
                    beamsY.Add(new LineCurve(node,
                    node1y));
                    node = new Point3d(xx, yy4,
                    height[k]);
                    node1y = new Point3d(xx, yy2,
                    height[k]);
                    beamsY.Add(new LineCurve(node,
                    node1y));
                }
                else
                {
                    node = new Point3d(xx, yy0,
                    height[k]);
                    node1y = new Point3d(xx, yy2,
                    height[k]);
                    beamsY.Add(new LineCurve(node,
                    node1y));
                }
            }
        }
    }
    else if (i == nx)
    {
        if (k != 0)
        {
            //SECOND PERIMETER
            if (nYfacWalls == 0)
            {
                node = new Point3d(xx, yy, height[k]);
                node1y = new Point3d(xx, yy1, height[k]);
                beamsY.Add(new LineCurve(node, node1y));
            }
            else
            {
                yy0 = yy;
                foreach (Line v in OnWall)
                {
                    if (v.FromY > yy0)
```



```
                    {
                        yy0 = v.FromY;
                    }
                    if (v.ToY > yy0)
                    {
                        yy0 = v.ToY;
                    }
                }
                if (yy0 >= yy1)
                {
                    //WALL ALONG ENTIRE BAY, NO BEAM
                }
                else
                {
                    yy2 = yy1;
                    foreach (Line v in Y1OnWall)
                    {
                        if (v.FromY < yy2)
                        {
                            yy2 = v.FromY;
                        }
                        if (v.ToY < yy2)
                        {
                            yy2 = v.ToY;
                        }
                    }
                    bool wallBtwn = false;
                    yy3 = 0;
                    yy4 = 0;
                    foreach (Line v in AllWalls)
                    {
                        if (v.FromX == v.ToX && v.FromY
                        == xx)
                        {
                            if (v.FromY > yy && v.ToY <
                            yy1)
                            {
                                yy3 = v.FromY;
                                yy4 = v.ToY;
                                wallBtwn = true;
                            }
                        }
                    }
                    if (wallBtwn == true)
                    {
                        //WALL BETWEEN POINTS
                        node = new Point3d(xx, yy0,
                        height[k]);
                        node1y = new Point3d(xx, yy3,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
                        node = new Point3d(xx, yy4,
                        height[k]);
                        node1y = new Point3d(xx, yy2,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
```

```
                    }
                    else
                    {
                        node = new Point3d(xx, yy0,
                        height[k]);
                        node1y = new Point3d(xx, yy2,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
                    }
                }
            }
        }
        else
        {
            if (k != 0)
            {
                if (ptin == true && pt1yin == true)
                {
                    ///No Y Beam
                }
                else if (ptin == true && pt1yin == false)
                {
                    yy0 = cymax;
                    node = new Point3d(xx, yy0, height[k]);
                    node1y = new Point3d(xx, yy1, height[k]);
                    beamsY.Add(new LineCurve(node, node1y));

                    if (k != nz)
                    {
                        if (k == 1)
                        {
                            node0z = new Point3d(xx, yy0,
                            height[k - 1]);
                            node1z = new Point3d(xx, yy0,
                            height[k]);
                            WallColumn.Add(new
                            LineCurve(node0z, node1z));
                            nodes.Add(node0z);
                        }
                        node0z = new Point3d(xx, yy0,
                        height[k]);
                        node1z = new Point3d(xx, yy0,
                        height[k + 1]);
                        WallColumn.Add(new
                        LineCurve(node0z, node1z));
                        nodes.Add(node0z);
                    }
                }
                else if (ptin == false && pt1yin == true)
                {
                    yy1 = cymin;
                    node = new Point3d(xx, yy, height[k]);
                    node1y = new Point3d(xx, yy1, height[k]);
                    beamsY.Add(new LineCurve(node, node1y));

                    if (k != nz)
```



```
                    {
                        if (k == 1)
                        {
                            node0z = new Point3d(xx, yy1,
                            height[k - 1]);
                            node1z = new Point3d(xx, yy1,
                            height[k]);
                            WallColumn.Add(new
                            LineCurve(node0z, node1z));
                            nodes.Add(node0z);
                        }
                        node0z = new Point3d(xx, yy1,
                        height[k]);
                        node1z = new Point3d(xx, yy1,
                        height[k + 1]);
                        WallColumn.Add(new
                        LineCurve(node0z, node1z));
                        nodes.Add(node0z);
                    }
                }
                else if (ptin == false && pt1yin == false)
                {
                    if (bmYin == true)
                    {
                        node = new Point3d(xx, yy,
                        height[k]);
                        node1y = new Point3d(xx, cymin,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
                        nodes.Add(new Point3d(xx, cymin, 0));
                        node = new Point3d(xx, cymax,
                        height[k]);
                        node1y = new Point3d(xx, yy1,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
                        nodes.Add(new Point3d(xx, yy1, 0));
                    }
                    else
                    {
                        node = new Point3d(xx, yy,
                        height[k]);
                        node1y = new Point3d(xx, yy1,
                        height[k]);
                        beamsY.Add(new LineCurve(node,
                        node1y));
                    }
                }
            }
        }

        ///Draw Columns
        if (k == nz) { }
        else
        {
            if (ptin == true) { }
            else
```

```
                        {
                            node = new Point3d(xx, yy, height[k]);
                            node1z = new Point3d(xx, yy, height[k + 1]);
                            if (PtOnLine == true)
                            {
                                WallColumn.Add(new LineCurve(node,
                                node1z));
                            }
                            else
                            {
                                columnsZ.Add(new LineCurve(node,
                                node1z));
                            }
                            nodes.Add(node);
                        }
                    }
                }
            }
        }

        ///------------------------------------------------------------
        ----------------

        ///------------------------------------------------------------
        ----------------
        ///BUILD WALL MESHES

        ///------------------------------------------------------------
        ----------------
        ///------------------------------------------------------------
        ----------------
        List<Mesh> WallMeshCore = new List<Mesh>();
        List<Mesh> WallMeshPerim = new List<Mesh>();
        List<Point3d> supports = new List<Point3d>();

        //COLUMN/WALL BASE POINTS
        foreach (Point3d v in nodes)
        {
            if (v.Z < 0.1)
            {
                supports.Add(v);
            }
        }
        foreach (Line v in AllWalls)
        {
            supports.Add(v.From);
            supports.Add(v.To);
        }

        foreach (Line u in AllWalls)
        {
            for (int j = 0; j < nz; j++)
            {
                //FIND POINTS ON LINE
                List<Point3d> WallPoints = new List<Point3d>();
                foreach (Point3d v in supports)
```



```
                {
                    if (u.MinimumDistanceTo(v) < 0.1)
                    {
                        WallPoints.Add(v);
                    }
                }

                //BUILD WALL MESHES
                if (u.FromX == u.ToX)
                {
                    //Y-Direction Wall
                    WallPoints = WallPoints.OrderByDescending(p =>
                    p.Y).ToList();
                    for (int i = 1; i < (WallPoints.Count); i++)
                    {
                        Mesh WallCurrent = BuildMesh(new
                        Point3d(u.FromX, WallPoints[i - 1].Y,
                        height[j]), new Point3d(u.FromX, WallPoints[i -
                        1].Y, height[j + 1]), new Point3d(u.ToX,
                        WallPoints[i].Y, height[j + 1]), new
                        Point3d(u.ToX, WallPoints[i].Y, height[j]));
                        if (u.FromX == 0 || u.FromX == XDim)
                        {
                            WallMeshPerim.Add(WallCurrent);
                        }
                        else
                        {
                            WallMeshCore.Add(WallCurrent);
                        }
                    }
                    else
                    {
                        //X-Direction Wall
                        WallPoints = WallPoints.OrderByDescending(p =>
                        p.X).ToList();
                        for (int i = 1; i < WallPoints.Count; i++)
                        {
                            Mesh WallCurrent = BuildMesh(new
                            Point3d(WallPoints[i - 1].X, WallPoints[i -
                            1].Y, height[j]), new Point3d(WallPoints[i -
                            1].X, WallPoints[i - 1].Y, height[j + 1]), new
                            Point3d(WallPoints[i].X, WallPoints[i].Y,
                            height[j + 1]), new Point3d(WallPoints[i].X,
                            WallPoints[i].Y, height[j]));
                            if (u.FromY == 0 || u.FromY == YDim)
                            {
                                WallMeshPerim.Add(WallCurrent);
                            }
                            else
                            {
                                WallMeshCore.Add(WallCurrent);
                            }
                        }
                    }
                }

            }
```

```
        }


        ///------------------------------------------------------------
        ----------------

        ///------------------------------------------------------------
        ----------------
        ///OUTPUTS

        ///------------------------------------------------------------
        ----------------

        ///------------------------------------------------------------
        ----------------
        DA.SetDataList(0, beamsX);
        DA.SetDataList(1, beamsY);
        DA.SetDataList(2, columnsZ);
        DA.SetDataList(3, floorMesh);
        DA.SetData(4, roofMesh);
        DA.SetDataList(5, WallMeshCore);
        DA.SetDataList(6, WallMeshPerim);
        DA.SetDataList(7, AllWalls);
        DA.SetDataList(8, WallColumn);
        DA.SetDataList(9, ListWallShear);
        DA.SetDataList(10, ListWallMoment);
    }

    private List<double> GetHeights(double nStories, double Height1,
    double HeightTyp)
    {
        List<double> height = new List<double>();
        height.Add(0);
        for (int i = 0; i < nStories; i++)
        {
            height.Add(Height1 + i * HeightTyp);
        }
        return height;
    }

    private Mesh BuildMesh(Point3d A, Point3d B, Point3d C, Point3d D)
    {
        Mesh NewMesh = new Mesh();

        NewMesh.Vertices.Add(A);
        NewMesh.Vertices.Add(B);
        NewMesh.Vertices.Add(C);
        NewMesh.Vertices.Add(D);
        NewMesh.Faces.AddFace(0, 1, 2, 3);

        return NewMesh;
    }

    public class SeismicForces
    {
        public List<double> wxhxk;
        public List<double> StoryForce;
        public List<double> StoryShear;
```



```
        public List<double> StoryMoment;
        public double MomentArm;

        public SeismicForces()
        {
            wxhxk = new List<double>();
            StoryForce = new List<double>();
            StoryShear = new List<double>();
            StoryMoment = new List<double>();
            MomentArm = 0;
        }

        public SeismicForces(List<double> wxhxkIn, List<double>
        StoryForceIn, List<double> StoryShearIn, List<double>
        StoryMomentIn, double MomentArmIn)
        {
            wxhxk = wxhxkIn;
            StoryForce = StoryForceIn;
            StoryShear = StoryShearIn;
            StoryMoment = StoryMomentIn;
            MomentArm = MomentArmIn;
        }

        public void CalculateSeismicForces(List<double> Heights, double
        EqMassTyp, double EqMassRoof, double FloorArea, double Veq,
        double kEq)
        {
            List<double> wxhxkCalc = new List<double>();
            for (int i = 1; i <= (Heights.Count() - 1); i++)
            {
                if (i < (Heights.Count() - 1))
                {
                    //FLOOR
                    wxhxkCalc.Add(EqMassTyp * FloorArea *
                    Math.Pow(Heights[i], kEq)); //lb and ft
                }
                else
                {
                    //ROOF
                    wxhxkCalc.Add(EqMassRoof * FloorArea *
                    Math.Pow(Heights[i], kEq)); //lb and ft
                }
            }
            wxhxk = wxhxkCalc;

            List<double> storyforce = new List<double>();
            List<double> storyshear = new List<double>();
            List<double> storymoment = new List<double>();
            for (int i = 1; i <= Heights.Count() - 1; i++)
            {
                if (i == 1)
                {
                    storyforce.Add(wxhxk[i - 1] * Veq / wxhxk.Sum()); //K
                    storyshear.Add(Veq); //K
                    storymoment.Add(storyforce[i - 1] * Heights[i]);
                    //K-ft
                }
                else
```

```
                {
                    storyforce.Add(wxhxk[i - 1] * Veq / wxhxk.Sum()); //K
                    storyshear.Add(storyshear[i - 2] - storyforce[i -
                    2]);//K
                    storymoment.Add(storyforce[i - 1] * Heights[i]);
                    //K-ft
                }
            }
            double momentarm = storymoment.Sum() / Veq; //ft

            StoryForce = storyforce;
            StoryShear = storyshear;
            StoryMoment = storymoment;
            MomentArm = momentarm;
        }
    }

    public override GH_Exposure Exposure
    {
        get { return GH_Exposure.primary; }
    }

    protected override System.Drawing.Bitmap Icon
    {
        get
        {
            return null;
        }
    }

        public override Guid ComponentGuid
        {
            get { return new
            Guid("{138e8ac5-1b07-4010-b2fa-ae6fa531e513}"); }
        }
    }
}

//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------
// SLAB DESIGN
//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------

using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
```



```
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;

public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
    private void Print(string text) { /* Implementation hidden. */ }
    private void Print(string format, params object[] args) { /*
    Implementation hidden. */ }
    private void Reflect(object obj) { /* Implementation hidden. */ }
    private void Reflect(object obj, string method_name) { /*
    Implementation hidden. */ }
    #endregion

    #region Members
    private readonly RhinoDoc RhinoDocument;
    private readonly GH_Document GrasshopperDocument;
    private readonly IGH_Component Component;
    private readonly int Iteration;
    #endregion

    private void RunScript(double TypSIL, double TypSDL, double TypLL,
    double SpanX, double SpanY, double DLFact, double lambda, double tConc,
    bool cSlab, bool fire, ref object tSlab, ref object SlabSW, ref object
    SlabDir, ref object pass)
    {

        //------------------------------------------------------------------
        ------------

        //------------------------------------------------------------------
        ------------
        //INITIALIZE VARIABLES

        //------------------------------------------------------------------
        ------------

        //------------------------------------------------------------------
        ------------
        double Itr, n;
        double Vc, Vtot;
        double TypFactLoad, TypDL;
        double Vdem, Mdem, Vcap, Mcap, Ddl, Dl;
        double yC, yCLT, yNA;
        double sigmaConc, sigmaCLT;
        double vibSpan, EIapp;
```

```
        int A;
        double chardepth = 1.2 * (1.5 * Math.Pow(1, 0.813));
        double depth;
        double Mmax, Vmax;
        double yCchar, yCLTchar, yNAchar, Itrchar;

        //NDS ADJUSTMENT FACTORS
        double Cm = 1;
        double Ct = 1;
        double Cl = 1;
        double KBend = 2.54;
        double PhiBend = 0.85;
        double KShear = 2.88;
        double PhiShear = 0.75;

        //MATERIAL PROPERTIES
        double ECLT = 1400000;
        double rho = 1.0625 * 0.5; //SPECIFIC GRAVITY
        double rhoh20 = 62.43; //DENSITY WATER PCF
        double rhoconc = 150; //PCF
        double rhoCLT = rho * rhoh20;
        double sg;

        //CLT PANEL PROPERTIES FROM MANUFACTURERS
        List<double> ListPanelLength = new List<double>(); //FT
        ListPanelLength.Add(64);
        ListPanelLength.Add(40);
        ListPanelLength.Add(64);
        ListPanelLength.Add(64);
        ListPanelLength.Add(40);
        ListPanelLength.Add(64);
        ListPanelLength.Add(64);
        ListPanelLength.Add(40);
        ListPanelLength.Add(64);
        ListPanelLength.Add(40);
        ListPanelLength.Add(64);

        List<double> ListDepth = new List<double>(); //IN
        ListDepth.Add(3.125);
        ListDepth.Add(3.9);
        ListDepth.Add(4.125);
        ListDepth.Add(5.125);
        ListDepth.Add(6.66);
        ListDepth.Add(6.875);
        ListDepth.Add(8.625);
        ListDepth.Add(9.42);
        ListDepth.Add(9.625);
        ListDepth.Add(12.18);
        ListDepth.Add(12.375);

        List<double> ListSW = new List<double>(); //PSF
        ListSW.Add(8.33);
        ListSW.Add(10.5);
        ListSW.Add(11.0);
        ListSW.Add(13.7);
        ListSW.Add(17.0);
        ListSW.Add(18.3);
        ListSW.Add(23.0);
```



```
        ListSW.Add(25.0);
        ListSW.Add(25.7);
        ListSW.Add(32.0);
        ListSW.Add(33.0);

        List<double> ListEI = new List<double>(); //LB-IN2
        ListEI.Add(48000000);
        ListEI.Add(79000000);
        ListEI.Add(115000000);
        ListEI.Add(184000000);
        ListEI.Add(321000000);
        ListEI.Add(440000000);
        ListEI.Add(853000000);
        ListEI.Add(818000000);
        ListEI.Add(1404000000);
        ListEI.Add(1662000000);
        ListEI.Add(2794000000);

        List<double> ListGA = new List<double>(); //LB
        ListGA.Add(340000);
        ListGA.Add(490000);
        ListGA.Add(460000);
        ListGA.Add(690000);
        ListGA.Add(1000000);
        ListGA.Add(920000);
        ListGA.Add(1400000);
        ListGA.Add(1500000);
        ListGA.Add(2000000);
        ListGA.Add(2100000);
        ListGA.Add(2400000);

        List<double> ListVmax = new List<double>(); //LB
        ListVmax.Add(1070);
        ListVmax.Add(1340);
        ListVmax.Add(1430);
        ListVmax.Add(1470);
        ListVmax.Add(1860);
        ListVmax.Add(1970);
        ListVmax.Add(2400);
        ListVmax.Add(2370);
        ListVmax.Add(3200);
        ListVmax.Add(2875);
        ListVmax.Add(3875);

        List<double> ListMmax = new List<double>(); //LB-IN
        ListMmax.Add(2525);
        ListMmax.Add(1800);
        ListMmax.Add(4525);
        ListMmax.Add(5800);
        ListMmax.Add(4275);
        ListMmax.Add(10400);
        ListMmax.Add(15975);
        ListMmax.Add(7700);
        ListMmax.Add(23700);
        ListMmax.Add(12075);
        ListMmax.Add(36700);

        List<double> ListFb0 = new List<double>();
```

```
        ListFb0.Add(1950);
        ListFb0.Add(875);
        ListFb0.Add(1950);
        ListFb0.Add(1950);
        ListFb0.Add(875);
        ListFb0.Add(1950);
        ListFb0.Add(1950);
        ListFb0.Add(875);
        ListFb0.Add(1950);
        ListFb0.Add(875);
        ListFb0.Add(1950);

        List<double> ListFv0 = new List<double>();
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);
        ListFv0.Add(135);

        //CONCRETE PROPERTIES
        double fc = 3000; //psi
        double Ec = 57000 * Math.Sqrt(fc / 1000); //psi
        double Ic = 1 / 12 * 12 * Math.Pow(tConc, 3);

        //GEOMETRY PARAMETERS
        double nSpan;
        double span = Math.Min(SpanX, SpanY);

        //SLAB SPAN DIRECTION
        int dir;
        if (SpanX <= SpanY)
        {
            dir = 2;
        }
        else
        {
            dir = 1;
        }
        SlabDir = dir;

        //LOOP THROUGH POSSIBLE CLT SLAB DEPTHS INCREASING DEPTH EACH LOOP
        for (int e = 0; e < 11; e++)
        {
            if (fire == false)
            {
                depth = ListDepth[e];
                Mmax = ListMmax[e];
                Vmax = ListVmax[e];
            }
            else
            {
                depth = ListDepth[e] - chardepth;
```



```
                Mmax = ListFb0[e] * 12 * Math.Pow(depth, 2) / 6;
                Vmax = ListFv0[e] * 2 * 12 * depth / 3;
            }

            if (cSlab == false)
            {
                //NON-COMPOSITE SLABS
                A = 0;
                Vcap = Vmax * Cm * Ct * KShear * PhiShear * lambda;
                Vc = 0.75 * 2 * 1 * Math.Sqrt(fc) * 12 * tConc;
                Vtot = Vcap + Vc;
                Mcap = Mmax * Cm * Ct * Cl * KBend * PhiBend * lambda;
                nSpan = Math.Floor(ListPanelLength[e] / span);

                TypFactLoad = TypSIL + DLFact * ListSW[e];
                TypDL = TypSDL + ListSW[e];

                //SINGLE SPAN
                if (nSpan == 1)
                {
                    Mdem = TypFactLoad * Math.Pow(span, 2) / 8;
                    Vdem = TypFactLoad * span / 2;
                    Ddl = 5 * (TypDL + TypLL) * Math.Pow(span, 4) * 12 /
                    (384 * ListEI[e]);
                    Dl = 5 * (TypLL) * Math.Pow(span, 4) * 12 / (384 *
                    ListEI[e]);
                }
                //DOUBLE SPAN
                else if (nSpan == 2)
                {
                    Mdem = 0.125 * TypFactLoad * Math.Pow(span, 2);
                    Vdem = 0.625 * TypFactLoad * span;
                    Ddl = (TypDL + TypLL) * Math.Pow(span, 4) * 12 / (185 *
                    ListEI[e]);
                    Dl = TypLL * Math.Pow(span, 4) * 12 / (185 * ListEI[e]);
                }
                //TRIPLE SPAN
                else
                {
                    Mdem = 0.1 * TypFactLoad * Math.Pow(span, 2);
                    Vdem = 0.6 * TypFactLoad * span;
                    Ddl = 0.0069 * (TypDL + TypLL) * Math.Pow(span, 4) * 12
                    / ListEI[e];
                    Dl = 0.0069 * TypLL * Math.Pow(span, 4) * 12 / ListEI[e];
                }

                //CHECK VIBRATIONS
                EIapp = 1 / ((1 / ListEI[e]) + (11.52 / (ListGA[e] *
                Math.Pow((12 * span), 2))));
                vibSpan = Math.Pow(EIapp, 0.293) / (12.05 * Math.Pow((rho *
                ListDepth[e] * 12), 0.122)); //CLT HANDBOOK CH7-3.4

                if (Mdem < Mcap & Vdem < Vcap & Ddl < (span / 240) & Dl <
                (span / 360) & span < vibSpan)
                {
                    A = 1;
                }
                else
```

```
            {
                A = 0;
            }

            //SLAB PASSES WRITE OUTPUT
            if (A == 1)
            {
                tSlab = ListDepth[e];
                SlabSW = ListSW[e];
                pass = true;
                break;
            }
            else
            {
                if (e == 10)
                {
                    tSlab = ListDepth[e];
                    SlabSW = ListSW[e];
                    pass = false;
                }
            }
            //SLAB FAILS LOOP
        }
        else
        {
            //DESIGN COMPOSITE SLABS
            A = 0;
            Vcap = ListFv0[e] * Cm * Ct * KShear * PhiShear * lambda;
            //ADJUSTED SHEAR STRESS
            Mcap = ListFb0[e] * Cm * Ct * Cl * KBend * PhiBend *
lambda; //ADJUSTED BENDING STRESS
            nSpan = Math.Floor(ListPanelLength[e] / span);

            TypFactLoad = TypSIL + DLFact * ListSW[e];
            TypDL = TypSDL + ListSW[e];

            //NO CHARRING
            yC = tConc / 2 + ListDepth[e];
            yCLT = ListDepth[e] / 2;
            yNA = (tConc * 12 * Ec * yC + ListDepth[e] * 12 * ECLT) /
(tConc * 12 * Ec + ListDepth[e] * 12 * ECLT);

            n = Ec / ECLT;
            Itr = 1 * 12 * n * Math.Pow(tConc, 3) / (12) + tConc * 12 *
Math.Pow((yC - yNA), 2) * n + 1 * 12 *
Math.Pow(ListDepth[e], 3) / 12 + ListDepth[e] * 12 *
Math.Pow((yCLT - yNA), 2);

            //CHARRING
            yCchar = tConc / 2 + depth;
            yCLTchar = depth / 2;
            yNAchar = (tConc * 12 * Ec * yC + depth * 12 * ECLT) /
(tConc * 12 * Ec + depth * 12 * ECLT);

            Itrchar = 1 * 12 * Math.Pow(tConc, 3) * n / (12) + tConc *
12 * Math.Pow((yCchar - yNAchar), 2) * n + 1 * 12 *
Math.Pow(depth, 3) / 12 + depth * 12 * Math.Pow((yCLTchar -
yNAchar), 2);
```



```

            if (fire == false)
            {
                yCchar = yC;
                yCLTchar = yCLT;
                yNAchar = yNA;
                Itrchar = Itr;
            }

            if (nSpan == 1)
            {
                Mdem = TypFactLoad * Math.Pow(span, 2) / 8;
                Vdem = 3 * TypFactLoad * span / (2 * 2 * (depth +
tConc) * 12); //fv = 3wL/4bd
                Ddl = 5 * (TypDL + TypLL) * Math.Pow(span, 4) * 12 /
(384 * ECLT * Itr);
                Dl = 5 * (TypLL) * Math.Pow(span, 4) * 12 / (384 * ECLT
* Itr);
            }
            else if (nSpan == 2)
            {
                Mdem = 0.125 * TypFactLoad * Math.Pow(span, 2);
                Vdem = 3 * 0.625 * TypFactLoad * span / (2 * (depth +
tConc) * 12);
                Ddl = (TypDL + TypLL) * Math.Pow(span, 4) * 12 / (185 *
ECLT * Itr);
                Dl = TypLL * Math.Pow(span, 4) * 12 / (185 * ECLT * Itr);
            }
            else
            {
                Mdem = 0.1 * TypFactLoad * Math.Pow(span, 2);
                Vdem = 3 * 0.6 * TypFactLoad * span / (2 * (depth +
tConc) * 12);
                Ddl = 0.0069 * (TypDL + TypLL) * Math.Pow(span, 4) * 12
/ (ECLT * Itr);
                Dl = 0.0069 * TypLL * Math.Pow(span, 4) * 12 / (ECLT *
Itr);
            }

            sigmaConc = -Mdem * (yCchar - yNAchar) * Ec / (Ec * Ic +
(ECLT * 12 * Math.Pow(depth, 3) / 12));
            sigmaCLT = -Mdem * (yCLTchar - yNAchar) * ECLT / (Ec * Ic +
(ECLT * 12 * Math.Pow(depth, 3) / 12));

            //CHECK VIBRATIONS
            sg = ((rhoconc * tConc * 144 + rhoCLT * ListDepth[e] * 144)
/ (tConc * 144 + ListDepth[e] * 144)) / rhoh20;
            EIapp = 1 / ((1 / (ECLT * Itr)) + (11.52 / (ListGA[e] *
Math.Pow((12 * span), 2))));
            vibSpan = Math.Pow(EIapp, 0.293) / (12.05 * Math.Pow((sg *
(tConc + ListDepth[e]) * 12), 0.122)); //CLT HANDBOOK CH7-3.4

            if (sigmaCLT < Mcap & sigmaConc < fc & Vdem < Vcap & Ddl <
(span / 240) & Dl < (span / 360) & span < vibSpan)
            {
                A = 1;
            }
            else
```

```
                {
                    A = 0;
                }

                if (A == 1)
                {
                    tSlab = ListDepth[e];
                    SlabSW = ListSW[e];
                    pass = true;
                    break;
                }
                else
                {
                    if (e == 10)
                    {
                        tSlab = ListDepth[e];
                        SlabSW = ListSW[e];
                        pass = false;
                    }
                }
            }
        }//END LOOP THROUGH SLAB DEPTHS
    }
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// CHECK WALLS FOR AXIAL LOADS & ADD COLUMNS AS REQUIRED
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

    using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
```



```
using System.Runtime.InteropServices;

public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
    private void Print(string text) { /* Implementation hidden. */ }
    private void Print(string format, params object[] args) { /*
    Implementation hidden. */ }
    private void Reflect(object obj) { /* Implementation hidden. */ }
    private void Reflect(object obj, string method_name) { /*
    Implementation hidden. */ }
    #endregion

    #region Members
    private readonly RhinoDoc RhinoDocument;
    private readonly GH_Document GrasshopperDocument;
    private readonly IGH_Component Component;
    private readonly int Iteration;
    #endregion

    private void RunScript(List<Line> WallLoc, List<Vector3d> RF,
    List<Point3d> Pos, List<double> VWall, List<double> MWall, double
    tWall, double Ht1, double HtTyp, double nFloors, double lambda, ref
    object WallColOut, ref object WallDC)
    {
        // INTIALIZE VARIABLES
        double LWall, PWall, PCap, PEq;
        double z0, z1;
        double phiComp = 0.9;
        double KComp = 2.4;
        List<double> ListPWall = new List<double>();
        List<double> DC = new List<double>();
        List<Line> WallColInt = new List<Line>();

        // PANEL ALLOWABLE AXIAL LOAD FROM STRUCTURLAM
        double Pr = 0; // LB/FT
        if (Ht1 <= 16)
        {
            if (tWall == 6.66)
            {
                Pr = 34344;
            }
            else if (tWall == 9.42)
            {
                Pr = 61864;
            }
            else if (tWall == 12.18)
            {
                Pr = 84689;
            }
        }
        else if (Ht1 <= 18)
        {
            if (tWall == 6.66)
            {
                Pr = 29124;
            }
            else if (tWall == 9.42)
```

```
            {
                Pr = 57440;
            }
            else if (tWall == 12.18)
            {
                Pr = 82206;
            }
        }
        else if (Ht1 <= 20)
        {
            if (tWall == 6.66)
            {
                Pr = 24667;
            }
            else if (tWall == 9.42)
            {
                Pr = 52187;
            }
            else if (tWall == 12.18)
            {
                Pr = 78947;
            }
        }

        // LOOP OVER EACH WALL, CHECKING FOR COMBINED AXIAL LOADING SEISMIC
        LOADING
        for (int i = 0; i < WallLoc.Count(); i++)
        {
            LWall = WallLoc[i].Length; // FT
            PWall = 0;
            PEq = MWall[i] * 1000 * 12 * 0.5 * LWall * 12 / (tWall *
            Math.Pow(LWall * 12, 3) / 12); // PSI

            for (int j = 0; j < Pos.Count(); j++)
            {
                if (WallLoc[i].DistanceTo(Pos[j], true) < 0.1)
                {
                    PWall = PWall + RF[j].Z; // K
                }
            }

            ListPWall.Add(PWall);
            PWall = PWall * 1000 / (LWall * 12 * tWall) + PEq; // PSI
            PCap = phiComp * KComp * Pr / (12 * tWall); // PSI
            DC.Add(PWall / PCap);

            if (DC[i] > 1)
            {
                // ADD COLUMNS TO TAKE GRAVITY LOADS
                foreach (Point3d v in Pos)
                {
                    if (WallLoc[i].DistanceTo(v, true) < 0.1)
                    {
                        for (double k = 0; k < nFloors; k += 1)
                        {
                            if (k == 0)
                            {
                                z0 = 0;
                                z1 = Ht1;
                            }
                            else
                            {
                                z0 = Ht1 + (k - 1) * HtTyp;
                                z1 = Ht1 + k * HtTyp;
                            }
                            WallColInt.Add(new Line(new Point3d(v.X, v.Y,
                            z0), new Point3d(v.X, v.Y, z1)));
                        }
                    }
                }
            }
        }

        WallColOut = WallColInt;
        WallDC = DC;
    }
}

//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------
// SIZE BEAMS AND COLUMNS
//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------

    using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;

public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
```

```
        private void Print(string text) { /* Implementation hidden. */ }
        private void Print(string format, params object[] args) { /*
        Implementation hidden. */ }
        private void Reflect(object obj) { /* Implementation hidden. */ }
        private void Reflect(object obj, string method_name) { /*
        Implementation hidden. */ }
        #endregion

        #region Members
        private readonly RhinoDoc RhinoDocument;
        private readonly GH_Document GrasshopperDocument;
        private readonly IGH_Component Component;
        private readonly int Iteration;
        #endregion

        private void RunScript(List<double> Nmax, List<double> Nmin,
        List<double> Vzmax, List<double> Vzmin, List<double> Vymax,
        List<double> Vymin, List<double> Mtmax, List<double> Mtmin,
        List<double> Mymax, List<double> Mymin, List<double> Mzmax,
        List<double> Mzmin, double FracDLFloor, double FracLLFloor, double
        lambda, List<double> Length, List<Point3d> StartPt, List<Point3d>
        EndPt, DataTree<double> GLBeams, DataTree<double> GLColumns, double
        nElem, bool fire, bool cBeam, double tCLTslab, double tCLTroof, double
        htActual, ref object width, ref object depth, ref object MaxBmD, ref
        object ifBeam, ref object Error)
        {

            double Cm = 1;
            double Ct = 1;
            double Cp = 1;
            double CComp = 0.9;
            double KComp = 2.4;
            double PhiComp = 0.9;
            double Cl = 1;
            double Cv = 1;
            double Cfu = 1;
            double Cc = 1;
            double Ci = 1;
            double KBend = 2.54;
            double PhiBend = 0.85;
            double Cvr = 1;
            double KShear = 2.88;
            double PhiShear = 0.75;

            bool Flag = false;
            double DC, DCNComp, DCNTens, DCMyPos, DCMyNeg, DCMz, DCVy, DCVz;
            List<double> DCList;
            int Fail, A, B, C, Section;
            double NCompDem, NTensDem, MyPosDem, MyNegDem, MyDem, MzDem, VzDem,
            VyDem, MtDem;
            double NCompAdj, NTensAdj, MyPosAdj, MyNegAdj, MzAdj, VzAdj, VyAdj;
            double NCompAll, NTensAll, MyPosAll, MyNegAll, MzAll, VzAll, VyAll;
            List<double> wOut = new List<double>();
            List<double> dOut = new List<double>();
            double wIt, dIt;
            double DeflFact;
            double FcE, FcStar, KELD, Z, time, fc, fb1, fb2, FcPrime, Fb1Prime,
            Emin1Prime, Le1, FcE1, Fb2Prime, Emin2Prime, Le2, FcE2, RB1, RB2,
```



```
            FbE, EQ393, EQ394, DeflL, DeflDL;
            bool Beam, roof;
            Point3d pts, pte;
            List<bool> BeamOut = new List<bool>();

            double fbGLB = 2400; //psi
            double ECLT = 1400000; //psi
            double EGLB = 1800000; //psi
            double yNA, yCLT, yGLB, ICLT, IGLB, Itr, n, bEff;
            double tCurr;

            List<double> debug = new List<double>();

            //ITERATE THROUGH ELEMENTS
            for (int i = 0; i < nElem; i++)
            {
                if (Nmin[i] < 0)
                {
                    NCompDem = Math.Abs(Nmin[i]);
                }
                else
                {
                    NCompDem = 0;
                }
                if (Nmax[i] > 0)
                {
                    NTensDem = Math.Abs(Nmax[i]);
                }
                else
                {
                    NTensDem = 0;
                }
                if (Mymin[i] < 0)
                {
                    MyPosDem = Math.Abs(Mymin[i]);
                }
                else
                {
                    MyPosDem = 0;
                }
                if (Mymax[i] > 0)
                {
                    MyNegDem = Math.Abs(Mymax[i]);
                }
                else
                {
                    MyNegDem = 0;
                }
                MyDem = Math.Max(MyPosDem, MyNegDem);
                MzDem = Math.Max(Math.Abs(Mzmax[i]), Math.Abs(Mzmin[i]));
                VzDem = Math.Max(Math.Abs(Vzmax[i]), Math.Abs(Vzmin[i]));
                VyDem = Math.Max(Math.Abs(Vymax[i]), Math.Abs(Vymin[i]));
                MtDem = Math.Max(Math.Abs(Mtmax[i]), Math.Abs(Mtmin[i]));

                pts = StartPt[i];
                pte = EndPt[i];
                if (pts.Z == pte.Z)
                {
```

```
            Beam = true;
        }
        else
        {
            Beam = false;
        }
        BeamOut.Add(Beam);

        if (Beam == true)
        {
            Fail = 0;
            A = 0;
            B = 0;
            C = 0;
            Section = 0;

            if (cBeam == false)
            {
                while (Fail < 3)
                {
                    //BEAM DESIGN

                    foreach (double v in GLBeams.Branch(5))
                    {
                        if (MyDem <= v)
                        {
                            break;
                        }
                        Section = Section + 1;
                    }

                    //SECTION PROPERTIES
                    wIt = GLBeams.Branch(0)[Section];
                    dIt = GLBeams.Branch(1)[Section];
                    NCompAll = GLBeams.Branch(2)[Section];
                    NTensAll = GLBeams.Branch(3)[Section];
                    MyPosAll = GLBeams.Branch(4)[Section];
                    MyNegAll = GLBeams.Branch(5)[Section];
                    MzAll = GLBeams.Branch(6)[Section];
                    VzAll = GLBeams.Branch(7)[Section];
                    VyAll = GLBeams.Branch(8)[Section];

                    //GENERATE PROPERTY SPECIFIC FACTORS
                    FcE = 0.822 * 0.85 * Math.Pow(10, 6) /
                    Math.Pow((Length[i] * 12 / dIt), 2);
                    FcStar = 1650 * Cm * Ct * KComp * PhiComp * lambda;
                    if (NCompDem == 0)
                    {
                        Cp = 1;
                    }
                    else
                    {
                        Cp = ((1 + (FcE / FcStar)) / (2 * CComp)) -
                        Math.Sqrt(Math.Pow(((1 + (FcE / FcStar)) / (2 *
                        CComp)), 2) - (FcE / FcStar) / CComp);
                    }
                    Cv = Math.Min(1, Math.Pow((21 / Length[i]), 0.1) *
                    Math.Pow((12 / dIt), 0.1) * Math.Pow((5.125 /
                    Math.Min(wIt, 10.75)), 0.1));
                    if (MzDem > 1 && wIt < 12)
                    {
                        Cfu = Math.Pow((12 / wIt), (1 / 9));
                    }
                    else
                    {
                        Cfu = 1;
                    }

                    //ADJUST DEMANDS
                    NCompAdj = NCompDem / (Cm * Ct * Cp * KComp *
                    PhiComp * lambda);
                    NTensAdj = NTensDem / (Cm * Ct * KComp * PhiComp *
                    lambda);
                    MyPosAdj = MyPosDem / (Cm * Ct * Cl * Cv * Cfu * Cc
                    * Ci * KBend * PhiBend * lambda);
                    MyNegAdj = MyNegDem / (Cm * Ct * Cl * Cv * Cfu * Cc
                    * Ci * KBend * PhiBend * lambda);
                    MzAdj = MzDem / (Cm * Ct * Cl * Cv * Cfu * Cc * Ci
                    * KBend * PhiBend * lambda);
                    VzAdj = VzDem / (Cm * Ct * Cvr * KShear * PhiShear
                    * lambda);
                    VyAdj = VyDem / (Cm * Ct * Cvr * KShear * PhiShear
                    * lambda);

                    //CHECK D/C RATIOS
                    DCList = new List<double>();
                    DCNComp = NCompAdj / NCompAll;
                    DCList.Add(DCNComp);
                    DCNTens = NTensAdj / NTensAll;
                    DCList.Add(DCNTens);
                    DCMyPos = MyPosAdj / MyPosAll;
                    DCList.Add(DCMyPos);
                    DCMyNeg = MyNegAdj / MyNegAll;
                    DCList.Add(DCMyNeg);
                    DCMz = MzAdj / MzAll;
                    DCList.Add(DCMz);
                    DCVy = VyAdj / VyAll;
                    DCList.Add(DCVy);
                    DCVz = VzAdj / VzAll;
                    DCList.Add(DCVz);
                    DC = DCList.Max();

                    if (DC < 1)
                    {
                        A = 1;
                    }
                    else
                    {
                        A = 0;
                    }
                    //FIRE DESIGN
                    KELD = 1 * Length[i] * 12 / dIt;
                    if (DC <= 0.5)
                    {
                        if (KELD <= 11)
                        {
```

```
                Z = 1.5;
            }
            else
            {
                Z = 1.3;
            }
        }
        else
        {
            if (KELD <= 11)
            {
                Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC +
                2.1333;
            }
            else
            {
                Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC +
                1.9333;
            }
        }
        time = 2.54 * Z * wIt * (4 - wIt / dIt);
        if (fire == false)
        {
            B = 1;
        }
        else if (time < 60)
        {
            B = 0;
        }
        else
        {
            B = 1;
        }
        //NO COMBINED BENDING FOR BEAMS

        //CHECK BEAM DEFLECTIONS
        DeflFact = MyPosDem * 12 * 40 * Math.Pow((Length[i]
        * 12), 2) / (384 * 1800 * wIt * Math.Pow(dIt, 3) /
        12);
        DeflDL = (FracDLFloor + FracLLFloor) * DeflFact;
        DeflL = FracLLFloor * DeflFact;
        if (DeflDL < ((Length[i] * 12) / 240) && DeflL <
        ((Length[i] * 12) / 360))
        {
            C = 1;
        }
        else
        {
            C = 0;
        }
        if (A + B + C == 3)
        {
            wOut.Add(wIt);
            dOut.Add(dIt);
        }
        Fail = A + B + C;
        Section = Section + 1;
        if (Section > 56)
```



```
            {
                Flag = true;
                wOut.Add(0);
                dOut.Add(0);
                break;
            }
            else { }
        } //END BEAM SECTION WHILE LOOP
    }
    else
    {
        //COMPOSITE BEAM DESIGN
        while (Fail < 3)
        {
            //BEAM DESIGN

            //SECTION PROPERTIES
            wIt = GLBeams.Branch(0)[Section];
            dIt = GLBeams.Branch(1)[Section];
            NCompAll = GLBeams.Branch(2)[Section];
            NTensAll = GLBeams.Branch(3)[Section];
            MyNegAll = GLBeams.Branch(5)[Section];
            MzAll = GLBeams.Branch(6)[Section];
            VyAll = GLBeams.Branch(8)[Section];

            yGLB = dIt / 2;
            IGLB = 1 * wIt * Math.Pow(dIt, 3) / 12;
            n = ECLT / EGLB;
            if (pts.Z == htActual)
            {
                roof = true;
            }
            else
            {
                roof = false;
            }

            if (roof == true)
            {
                tCurr = tCLTroof;
            }
            else
            {
                tCurr = tCLTslab;
            }

            yCLT = dIt + tCurr / 2;
            bEff = Math.Min(0.25 * Length[i], 2 * 8 * tCurr +
            wIt);
            ICLT = 1 * bEff * Math.Pow(tCurr, 3) / 12;
            yNA = (tCurr * bEff * ECLT * yCLT + dIt * wIt *
            EGLB * yGLB) / (tCurr * bEff * ECLT + wIt * dIt *
            EGLB);
            Itr = 1 * wIt * Math.Pow(dIt, 3) / (12 * n) + wIt *
            dIt * Math.Pow((yGLB - yNA), 2) / n + 1 * bEff *
            Math.Pow(tCurr, 3) / 12 + bEff * tCurr *
            Math.Pow((yGLB - yNA), 2);
```

```
MyPosAll = fbGLB * (ECLT * ICLT + EGLB * IGLB) /
((yCLT - yNA) * EGLB); //lb-in
MyPosAll = MyPosAll / (1000 * 12); //K-ft
VzAll = GLBeams.Branch(7)[Section];

//GENERATE PROPERTY SPECIFIC FACTORS
FcE = 0.822 * 0.85 * Math.Pow(10, 6) /
Math.Pow((Length[i] * 12 / dIt), 2);
FcStar = 1650 * Cm * Ct * KComp * PhiComp * lambda;
if (NCompDem == 0)
{
    Cp = 1;
}
else
{
    Cp = ((1 + (FcE / FcStar)) / (2 * CComp)) -
    Math.Sqrt(Math.Pow(((1 + (FcE / FcStar)) / (2 *
    CComp)), 2) - (FcE / FcStar) / CComp);
}
Cv = Math.Min(1, Math.Pow((21 / Length[i]), 0.1) *
Math.Pow((12 / dIt), 0.1) * Math.Pow((5.125 /
Math.Min(wIt, 10.75)), 0.1));
if (MzDem > 1 && wIt < 12)
{
    Cfu = Math.Pow((12 / wIt), (1 / 9));
}
else
{
    Cfu = 1;
}

//ADJUST DEMANDS
NCompAdj = NCompDem / (Cm * Ct * Cp * KComp *
PhiComp * lambda);
NTensAdj = NTensDem / (Cm * Ct * KComp * PhiComp *
lambda);
MyPosAdj = MyPosDem / (Cm * Ct * Cl * Cv * Cfu * Cc
* Ci * KBend * PhiBend * lambda);
MyNegAdj = MyNegDem / (Cm * Ct * Cl * Cv * Cfu * Cc
* Ci * KBend * PhiBend * lambda);
MzAdj = MzDem / (Cm * Ct * Cl * Cv * Cfu * Cc * Ci
* KBend * PhiBend * lambda);
VzAdj = VzDem / (Cm * Ct * Cvr * KShear * PhiShear
* lambda);
VyAdj = VyDem / (Cm * Ct * Cvr * KShear * PhiShear
* lambda);

//CHECK D/C RATIOS
DCList = new List<double>();
DCNComp = NCompAdj / NCompAll;
DCList.Add(DCNComp);
DCNTens = NTensAdj / NTensAll;
DCList.Add(DCNTens);
DCMyPos = MyPosAdj / MyPosAll;
DCList.Add(DCMyPos);
DCMyNeg = MyNegAdj / MyNegAll;
DCList.Add(DCMyNeg);
DCMz = MzAdj / MzAll;
```



```
DCList.Add(DCMz);
DCVy = VyAdj / VyAll;
DCList.Add(DCVy);
DCVz = VzAdj / VzAll;
DCList.Add(DCVz);
DC = DCList.Max();
if (DC < 1)
{
    A = 1;
}
else
{
    A = 0;
}
//FIRE DESIGN
KELD = 1 * Length[i] * 12 / dIt;
if (DC <= 0.5)
{
    if (KELD <= 11)
    {
        Z = 1.5;
    }
    else
    {
        Z = 1.3;
    }
}
else
{
    if (KELD <= 11)
    {
        Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC +
        2.1333;
    }
    else
    {
        Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC +
        1.9333;
    }
}
time = 2.54 * Z * wIt * (4 - wIt / dIt);
if (fire == false)
{
    B = 1;
}
else if (time < 60)
{
    B = 0;
}
else
{
    B = 1;
}
//NO COMBINED BENDING FOR BEAMS

//CHECK BEAM DEFLECTIONS
DeflFact = MyPosDem * 12 * 40 * Math.Pow((Length[i]
* 12), 2) / (384 * ECLT * Itr);
```

```
                    DeflDL = (FracDLFloor + FracLLFloor) * DeflFact;
                    DeflL = FracLLFloor * DeflFact;
                    if (DeflDL < ((Length[i] * 12) / 240) && DeflL <
                    ((Length[i] * 12) / 360))
                    {
                        C = 1;
                    }
                    else
                    {
                        C = 0;
                    }
                    if (A + B + C == 3)
                    {
                        wOut.Add(wIt);
                        dOut.Add(dIt);
                    }
                    Fail = A + B + C;
                    Section = Section + 1;
                    if (Section == 56)
                    {
                        Flag = true;
                        wOut.Add(0);
                        dOut.Add(0);
                        break;
                    }
                    else { }
                } //END COMPOSITE BEAM SECTION WHILE LOOP
            }
        } //End Beam If
        else
        {
            Fail = 0;
            A = 0;
            B = 0;
            C = 0;
            Section = 0;
            foreach (double v in GLColumns.Branch(2))
            {
                if (NCompDem <= v)
                {
                    break;
                }
                Section = Section + 1;
            }

            while (Fail < 3)
            {
                //SECTION PROPERTIES
                wIt = GLColumns.Branch(0)[Section];
                dIt = GLColumns.Branch(1)[Section];
                NCompAll = GLColumns.Branch(2)[Section];
                NTensAll = GLColumns.Branch(3)[Section];
                MyPosAll = GLColumns.Branch(4)[Section];
                MyNegAll = GLColumns.Branch(5)[Section];
                MzAll = GLColumns.Branch(6)[Section];
                VzAll = GLColumns.Branch(7)[Section];
                VyAll = GLColumns.Branch(8)[Section];
```



```
                //GENERATE PROPERTY SPECIFIC FACTORS
                FcE = 0.822 * 0.85 * Math.Pow(10, 6) /
                Math.Pow((Length[i] * 12 / dIt), 2);
                FcStar = 1650 * Cm * Ct * KComp * PhiComp * lambda;
                if (NCompDem == 0)
                {
                    Cp = 1;
                }
                else
                {
                    Cp = ((1 + (FcE / FcStar)) / (2 * CComp)) -
                    Math.Sqrt(Math.Pow(((1 + (FcE / FcStar)) / (2 *
                    CComp)), 2) - (FcE / FcStar) / CComp);
                }
                Cv = Math.Min(1, Math.Pow((21 / Length[i]), 0.1) *
                Math.Pow((12 / dIt), 0.1) * Math.Pow((5.125 /
                Math.Min(wIt, 10.75)), 0.1));
                if (MzDem > 1 && wIt < 12)
                {
                    Cfu = Math.Pow((12 / wIt), (1 / 9));
                }
                else
                {
                    Cfu = 1;
                }

                //ADJUST DEMANDS
                NCompAdj = NCompDem / (Cm * Ct * Cp * KComp * PhiComp *
                lambda);
                NTensAdj = NTensDem / (Cm * Ct * KComp * PhiComp *
                lambda);
                MyPosAdj = MyPosDem / (Cm * Ct * Cl * Cv * Cfu * Cc *
                Ci * KBend * PhiBend * lambda);
                MyNegAdj = MyNegDem / (Cm * Ct * Cl * Cv * Cfu * Cc *
                Ci * KBend * PhiBend * lambda);
                MzAdj = MzDem / (Cm * Ct * Cl * Cv * Cfu * Cc * Ci *
                KBend * PhiBend * lambda);
                VzAdj = VzDem / (Cm * Ct * Cvr * KShear * PhiShear *
                lambda);
                VyAdj = VyDem / (Cm * Ct * Cvr * KShear * PhiShear *
                lambda);

                //CHECK D/C RATIOS
                DCList = new List<double>();
                DCNComp = NCompAdj / NCompAll;
                DCList.Add(DCNComp);
                DCNTens = NTensAdj / NTensAll;
                DCList.Add(DCNTens);
                DCMyPos = MyPosAdj / MyPosAll;
                DCList.Add(DCMyPos);
                DCMyNeg = MyNegAdj / MyNegAll;
                DCList.Add(DCMyNeg);
                DCMz = MzAdj / MzAll;
                DCList.Add(DCMz);
                DCVy = VyAdj / VyAll;
                DCList.Add(DCVy);
                DCVz = VzAdj / VzAll;
                DCList.Add(DCVz);
```

```
                DC = DCList.Max();
                if (DC < 1)
                {
                    A = 1;
                }
                else
                {
                    A = 0;
                }
                //FIRE DESIGN
                KELD = 1 * Length[i] * 12 / dIt;
                if (DC <= 0.5)
                {
                    if (KELD <= 11)
                    {
                        Z = 1.5;
                    }
                    else
                    {
                        Z = 1.3;
                    }
                }
                else
                {
                    if (KELD <= 11)
                    {
                        Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC + 2.1333;
                    }
                    else
                    {
                        Z = 2 * Math.Pow(DC, 2) / 3 - 1.6 * DC + 1.9333;
                    }
                }
                time = 2.54 * Z * wIt * (3 - wIt / dIt);
                if (fire == false)
                {
                    B = 1;
                }
                else if (time < 60)
                {
                    B = 0;
                }
                else
                {
                    B = 1;
                }
                //COMBINED BENDING
                if ((MyPosAdj / MyPosAll) < 0.01 && (MyNegAdj /
                MyNegAll) < 0.01 && (MzAdj / MzAll) < 0.01)
                {
                    //NO COMBINED BENDING
                    C = 1;
                }
                else
                {
                    fc = NCompDem * 1000 / (wIt * dIt);
                    fb1 = Math.Max(MyPosDem, MyNegDem) * 1000 * 12 /
                    (wIt * Math.Pow(dIt, 2) / 6);
```



```
                    fb2 = MzDem * 1000 * 12 / (wIt * Math.Pow(dIt, 2) /
                    6);
                    FcPrime = NCompAll * 1000 * (Cm * Ct * Cp * KComp *
                    PhiComp * lambda) / (wIt * dIt);
                    Fb1Prime = MyPosAll * 1000 * (Cm * Ct * Cl * Cv *
                    Cfu * Cc * Ci * KBend * PhiBend * lambda) / (wIt *
                    Math.Pow(dIt, 2) / 6);
                    Emin1Prime = 0.95 * Cm * Ct * 1.76 * 0.85 *
                    Math.Pow(10, 6);
                    if ((Length[i] * 12 / dIt) < 7)
                    {
                        Le1 = 1.87 * Length[i] * 12;
                    }
                    else
                    {
                        Le1 = 1.44 * Length[i] * 12 + 3 * dIt;
                    }
                    FcE1 = 0.822 * Emin1Prime / Math.Pow((Le1 / dIt), 2);
                    Fb2Prime = MzDem * 1000 * 12 * (Cm * Ct * Cl * Cv *
                    Cfu * Cc * Ci * KBend * PhiBend * lambda) / (dIt *
                    Math.Pow(wIt, 2) / 6);
                    Emin2Prime = 0.85 * Cm * Ct * 1.76 * 0.85 *
                    Math.Pow(10, 6);
                    if ((Length[i] * 12 / wIt) < 7)
                    {
                        Le2 = 1.87 * Length[i] * 12;
                    }
                    else
                    {
                        Le2 = 1.44 * Length[i] * 12 + 3 * wIt;
                    }
                    FcE2 = 0.822 * Emin2Prime / Math.Pow((Le2 / wIt), 2);
                    RB1 = Math.Sqrt(Le1 * dIt / Math.Pow(wIt, 2));
                    RB2 = Math.Sqrt(Le2 * wIt / Math.Pow(dIt, 2));
                    FbE = Math.Min(1.2 * Emin1Prime / Math.Pow(RB1, 2),
                    1.2 * Emin2Prime / Math.Pow(RB2, 2));
                    EQ393 = Math.Pow(fc / FcPrime, 2) + fb1 / (Fb1Prime
                    * (1 - (fc / FcE1))) + fb2 / (Fb2Prime * (1 - (fc /
                    FcE2) - Math.Pow((fb1 / FbE), 2)));
                    EQ394 = fc / FcE2 + Math.Pow((fb1 / FbE), 2);

                    if (EQ393 < 1 && EQ394 < 1)
                    {
                        C = 1;
                    }
                    else
                    {
                        C = 0;
                    }
                }
                if (A + B + C == 3)
                {
                    wOut.Add(wIt);
                    dOut.Add(dIt);
                }
                Fail = A + B + C;
                Section = Section + 1;
                if (Section == 289)
```

```
                    {
                        Flag = true;
                        wOut.Add(0);
                        dOut.Add(0);
                        break;
                    }
                    else { }
                }//END COLUMN SECTION LOOP
            } //End Column If
        } //End Element Loop

        double MBD = 0;
        for (int i = 0; i < BeamOut.Count(); i++)
        {
            if (BeamOut[i] == true)
            {
                if (MBD < dOut[i])
                {
                    MBD = dOut[i];
                }
            }
        }

        width = wOut;
        depth = dOut;
        MaxBmD = MBD;
        ifBeam = BeamOut;
        Error = Flag;
    }
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
// CALCULATE LCA IMPACTS
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
```



```
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;

public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
    private void Print(string text) { /* Implementation hidden. */ }
    private void Print(string format, params object[] args) { /*
    Implementation hidden. */ }
    private void Reflect(object obj) { /* Implementation hidden. */ }
    private void Reflect(object obj, string method_name) { /*
    Implementation hidden. */ }
    #endregion

    #region Members
    private readonly RhinoDoc RhinoDocument;
    private readonly GH_Document GrasshopperDocument;
    private readonly IGH_Component Component;
    private readonly int Iteration;
    #endregion

    private void RunScript(double AreaCLTCoreWall, double AreaCLTFacWall,
    double tWall, double AreaCLTFloor, double tCLTFloor, double
    tConcreteFloor, double AreaCLTRoof, double tCLTRoof, double
    tConcreteRoof, List<double> LElem, List<double> tElem, List<double>
    dElem, List<bool> ifBeam, double XDimAct, double YDimAct, double HtAct,
    List<double> ACin, List<double> ADin, List<double> waste, ref object
    TotAtoC, ref object TotAtoD, ref object AtoCperSF, ref object
    AtoDperSF, ref object OutputTot, ref object OutputSF, ref object CVol,
    ref object BVol, ref object AFac)
    {
        //INITIALIZE VARIABLES
        double beamvol = 0;
        double colvol = 0;
        double slabCLTvol = 0;
        double roofCLTvol = 0;
        double slabConcVol = 0;
        double roofConcVol = 0;
        double wallCLTvol = 0;
        double AreaGlass = 0;
        double steelVol = 0;
        double ac = 0;
        double ad = 0;
        int matNo = 0;
        List<double> acIn = ACin;
        List<double> adIn = ADin;
        List<double> acOut = new List<double>();
        List<double> adOut = new List<double>();
        List<string> names = new List<string>();

        //CALCULATE VOLUME OF GLULAM BEAMS AND COLUMNS
        for (int i = 0; i < LElem.Count; i++)
        {
            if (ifBeam[i] == true)
```

```
            {
                //ELEMENT IS A BEAM
                beamvol = beamvol + tElem[i] * dElem[i] * LElem[i] / 144;
                //FT3
            }
            else
            {
                //ELEMENT IS A COLUMN
                colvol = colvol + tElem[i] * dElem[i] * LElem[i] / 144; //FT3
            }
        }

        //CALCULATE VOLUME OF CLT FLOOR SLABS
        slabCLTvol = AreaCLTFloor * tCLTFloor / 12; //FT3
        slabConcVol = AreaCLTFloor * tConcreteFloor / (12 * 27); //CY
        roofCLTvol = AreaCLTRoof * tCLTRoof / 12; //FT3
        roofConcVol = AreaCLTRoof * tConcreteRoof / (12 * 27); //CY

        //CALCULATE VOLUME OF CLT SHEAR WALLS
        wallCLTvol = (AreaCLTCoreWall + AreaCLTFacWall) * tWall / 12;
        AreaGlass = 2 * (XDimAct + YDimAct) * HtAct - AreaCLTFacWall;

        //IMPACTS BY MATERIAL/ASSEMBLY
        //GLULAM BEAMS
        matNo = 102; //GluLam Sections
        ac = ac + acIn[matNo] * beamvol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * beamvol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * beamvol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * beamvol * waste[matNo] / 100);
        names.Add("GluLam Beams");

        //GLULAM COLUMNS
        matNo = 102; //GluLam Sections
        ac = ac + acIn[matNo] * colvol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * colvol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * colvol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * colvol * waste[matNo] / 100);
        names.Add("GluLam Columns");

        //CLT FLOOR SLABS
        matNo = 54; //Cross Laminated Timber
        ac = ac + acIn[matNo] * slabCLTvol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * slabCLTvol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * slabCLTvol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * slabCLTvol * waste[matNo] / 100);
        names.Add("CLT Floors");

        //FLOOR SLAB UNDERLAYMENT
        matNo = 145; //Polyiso Foam Board
        ac = ac + acIn[matNo] * AreaCLTFloor * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaCLTFloor * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaCLTFloor * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * AreaCLTFloor * waste[matNo] / 100);
        names.Add("Floor Underlayment");

        //FLOOR SLAB CONCRETE TOPPING
        matNo = 47; //Concrete Benchmark 3000 psi
        ac = ac + acIn[matNo] * slabConcVol * waste[matNo] / 100;
```



```
        ad = ad + adIn[matNo] * slabConcVol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * slabConcVol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * slabConcVol * waste[matNo] / 100);
        names.Add("Floor Concrete Topping");

        //CLT ROOF SLAB
        matNo = 54; //Cross Laminated Timber
        ac = ac + acIn[matNo] * roofCLTvol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * roofCLTvol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * roofCLTvol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * roofCLTvol * waste[matNo] / 100);
        names.Add("CLT Roof");

        //ROOF SLAB CONCRETE TOPPING
        matNo = 47; //Concrete Benchmark 3000 psi
        ac = ac + acIn[matNo] * roofConcVol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * roofConcVol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * roofConcVol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * roofConcVol * waste[matNo] / 100);
        names.Add("Roof Concrete Topping");

        //ROOF SLAB MEMBRANE
        matNo = 9; //3 mil Polyethylene
        ac = ac + acIn[matNo] * AreaCLTRoof * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaCLTRoof * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaCLTRoof * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * AreaCLTRoof * waste[matNo] / 100);
        names.Add("Roof Membrance");

        //ROOF SLAB INSULATION
        matNo = 69; //Extruded Polystyrene
        ac = ac + acIn[matNo] * AreaCLTRoof * 6 * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaCLTRoof * 6 * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaCLTRoof * 6 * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * AreaCLTRoof * 6 * waste[matNo] / 100);
        names.Add("Roof Insulation");

        //CLT WALL
        matNo = 54; //Cross Laminated Timber
        ac = ac + acIn[matNo] * wallCLTvol * waste[matNo] / 100;
        ad = ad + adIn[matNo] * wallCLTvol * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * wallCLTvol * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * wallCLTvol * waste[matNo] / 100);
        names.Add("CLT Wall");

        //CORE WALL GWB
        matNo = 5; //1/2" Gypsum Board
        ac = ac + acIn[matNo] * 2 * AreaCLTCoreWall * waste[matNo] / 100;
        ad = ad + adIn[matNo] * 2 * AreaCLTCoreWall * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * 2 * AreaCLTCoreWall * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * 2 * AreaCLTCoreWall * waste[matNo] / 100);
        names.Add("Core Wall GWB");

        //FACADE WALL WATERPROOFING
        matNo = 9; //3 mil Polyethylene
        ac = ac + acIn[matNo] * AreaCLTFacWall * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaCLTFacWall * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaCLTFacWall * waste[matNo] / 100);
```

```
        adOut.Add(adIn[matNo] * AreaCLTFacWall * waste[matNo] / 100);
        names.Add("Facade Waterproofing");

        //FACADE WALL INSULATION
        matNo = 69; //Extruded Polystyrene
        ac = ac + acIn[matNo] * 6 * AreaCLTFacWall * waste[matNo] / 100;
        ad = ad + adIn[matNo] * 6 * AreaCLTFacWall * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * 6 * AreaCLTFacWall * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * 6 * AreaCLTFacWall * waste[matNo] / 100);
        names.Add("Facade Insulation");

        //FACADE WALL CLADDING
        matNo = 142; //Pine T&G Siding
        ac = ac + acIn[matNo] * AreaCLTFacWall * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaCLTFacWall * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaCLTFacWall * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * AreaCLTFacWall * waste[matNo] / 100);
        names.Add("Facade Cladding");

        //CURTAIN WALL
        matNo = 56; //Double Glazed Windows
        ac = ac + acIn[matNo] * AreaGlass * waste[matNo] / 100;
        ad = ad + adIn[matNo] * AreaGlass * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * AreaGlass * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * AreaGlass * waste[matNo] / 100);
        names.Add("Curtain Wall");

        //CURTAIN WALL ALUMINUM
        matNo = 24; //Aluminum Window Frame
                    //From Athena 0.1262 Tons/100 SF of Standard Curtain
                    Wall (includes waste)
        ac = ac + acIn[matNo] * 0.1262 * AreaGlass / (100 * 100);
        ad = ad + adIn[matNo] * 0.1262 * AreaGlass / (100 * 100);
        acOut.Add(acIn[matNo] * 0.1262 * AreaGlass / (100 * 100));
        adOut.Add(adIn[matNo] * 0.1262 * AreaGlass / (100 * 100));
        names.Add("Curtain Wall Mullions");

        //TIMBER STEEL CONNECTIONS
        matNo = 171; //Steel Plate
        steelVol = 0.0025 * (beamvol + colvol + slabCLTvol + roofCLTvol +
        wallCLTvol);
        //steel density 490 lb/ft^3
        ac = ac + acIn[matNo] * steelVol * 0.245 * waste[matNo] / 100;
        ad = ad + adIn[matNo] * steelVol * 0.245 * waste[matNo] / 100;
        acOut.Add(acIn[matNo] * steelVol * 0.245 * waste[matNo] / 100);
        adOut.Add(adIn[matNo] * steelVol * 0.245 * waste[matNo] / 100);
        names.Add("Steel Connections");

        //Itemized Output
        DataTree<System.Object> LCAItems = new DataTree<System.Object>();
        DataTree<System.Object> LCAItemsSF = new DataTree<System.Object>();
        GH_Path A = new GH_Path(0);
        GH_Path B = new GH_Path(1);
        GH_Path C = new GH_Path(2);

        for (int i = 0; i < names.Count; i++)
        {
            LCAItems.Add(names[i], A);
```



```
            LCAItemsSF.Add(names[i], A);
            LCAItems.Add(acOut[i], B);
            LCAItemsSF.Add(acOut[i] / (AreaCLTFloor + AreaCLTRoof), B);
            LCAItems.Add(adOut[i], C);
            LCAItemsSF.Add(adOut[i] / (AreaCLTFloor + AreaCLTRoof), C);
        }

        //Output per SF
        double acSFOut = ac / (AreaCLTFloor + AreaCLTRoof);
        double adSFOut = ad / (AreaCLTFloor + AreaCLTRoof);

        //OUTPUTS
        TotAtoC = ac;
        TotAtoD = ad;
        AtoCperSF = acSFOut;
        AtoDperSF = adSFOut;
        OutputTot = LCAItems;
        OutputSF = LCAItemsSF;
        CVol = colvol;
        BVol = beamvol;
        AFac = AreaGlass;
    }
}

//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------
//  CALCULATE COST IMPACTS
//------------------------------------------------------------------------
----------------
//------------------------------------------------------------------------
----------------

    using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;
```

```
public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
    private void Print(string text) { /* Implementation hidden. */ }
    private void Print(string format, params object[] args) { /*
    Implementation hidden. */ }
    private void Reflect(object obj) { /* Implementation hidden. */ }
    private void Reflect(object obj, string method_name) { /*
    Implementation hidden. */ }
    #endregion

    #region Members
    private readonly RhinoDoc RhinoDocument;
    private readonly GH_Document GrasshopperDocument;
    private readonly IGH_Component Component;
    private readonly int Iteration;
    #endregion

    private void RunScript(double AreaCLTCoreWall, double AreaCLTFacWall,
    double tWall, double AreaCLTFloor, double tCLTFloor, double
    tConcreteFloor, double AreaCLTRoof, double tCLTRoof, double
    tConcreteRoof, List<double> LElem, List<double> tElem, List<double>
    dElem, List<bool> ifBeam, double XDimAct, double YDimAct, double HtAct,
    List<double> CostIn, ref object TotCost, ref object CostperSF, ref
    object OutputTot, ref object OutputSF)
    {
        //INITIALIZE VARIABLES
        double beamvol = 0;
        double colvol = 0;
        double slabCLTvol = 0;
        double roofCLTvol = 0;
        double slabConcVol = 0;
        double roofConcVol = 0;
        double wallCLTvol = 0;
        double AreaGlass = 0;
        int matNo = 0;
        double tCost = 0;
        List<double> CostOut = new List<double>();
        List<string> names = new List<string>();

        //CALCULATE VOLUME OF GLULAM BEAMS AND COLUMNS
        for (int i = 0; i < LElem.Count; i++)
        {
            if (ifBeam[i] == true)
            {
                //ELEMENT IS A BEAM
                beamvol = beamvol + tElem[i] * dElem[i] * LElem[i] / 144;
                //FT3
            }
            else
            {
                //ELEMENT IS A COLUMN
                colvol = colvol + tElem[i] * dElem[i] * LElem[i] / 144; //FT3
            }
        }

        //CALCULATE VOLUME OF CLT FLOOR SLABS
        slabCLTvol = AreaCLTFloor * tCLTFloor / 12; //FT3
        slabConcVol = AreaCLTFloor * tConcreteFloor / (12 * 27); //CY
        roofCLTvol = AreaCLTRoof * tCLTRoof / 12; //FT3
        roofConcVol = AreaCLTRoof * tConcreteRoof / (12 * 27); //CY

        //CALCULATE VOLUME OF CLT SHEAR WALLS
        wallCLTvol = (AreaCLTCoreWall + AreaCLTFacWall) * tWall / 12;
        AreaGlass = 2 * (XDimAct + YDimAct) * HtAct - AreaCLTFacWall;

        //IMPACTS BY MATERIAL/ASSEMBLY
        //GLULAM BEAMS
        matNo = 0; //GluLam Sections
        tCost = tCost + CostIn[matNo] * beamvol;
        CostOut.Add(CostIn[matNo] * beamvol);
        names.Add("GluLam Beams");

        //GLULAM COLUMNS
        matNo = 1; //GluLam Sections
        tCost = tCost + CostIn[matNo] * colvol;
        CostOut.Add(CostIn[matNo] * colvol);
        names.Add("GluLam Columns");

        //CLT FLOOR SLABS
        matNo = 2; //Cross Laminated Timber
        tCost = tCost + CostIn[matNo] * slabCLTvol;
        CostOut.Add(CostIn[matNo] * slabCLTvol);
        names.Add("CLT Floors");

        //FLOOR SLAB UNDERLAYMENT
        matNo = 3; //Polyiso Foam Board
        tCost = tCost + CostIn[matNo] * AreaCLTFloor;
        CostOut.Add(CostIn[matNo] * AreaCLTFloor);
        names.Add("Floor Underlayment");

        //FLOOR SLAB CONCRETE TOPPING
        matNo = 4; //Concrete Benchmark 3000 psi
        tCost = tCost + CostIn[matNo] * AreaCLTFloor;
        CostOut.Add(CostIn[matNo] * AreaCLTFloor);
        names.Add("Floor Concrete Topping");

        //CLT ROOF SLAB
        matNo = 5; //Cross Laminated Timber
        tCost = tCost + CostIn[matNo] * roofCLTvol;
        CostOut.Add(CostIn[matNo] * roofCLTvol);
        names.Add("CLT Roof");

        //ROOF SLAB CONCRETE TOPPING
        matNo = 6; //Concrete Benchmark 3000 psi
        tCost = tCost + CostIn[matNo] * AreaCLTRoof;
        CostOut.Add(CostIn[matNo] * AreaCLTRoof);
        names.Add("Roof Concrete Topping");

        //ROOF SLAB MEMBRANE
        matNo = 7; //3 mil Polyethylene
        tCost = tCost + CostIn[matNo] * AreaCLTRoof;
        CostOut.Add(CostIn[matNo] * AreaCLTRoof);
        names.Add("Roof Membrane");
```

```
        //ROOF SLAB INSULATION
        matNo = 8; //Extruded Polystyrene
        tCost = tCost + CostIn[matNo] * AreaCLTRoof;
        CostOut.Add(CostIn[matNo] * AreaCLTRoof);
        names.Add("Roof Insulation");

        //CLT WALL
        matNo = 9; //Cross Laminated Timber
        tCost = tCost + CostIn[matNo] * wallCLTvol;
        CostOut.Add(CostIn[matNo] * wallCLTvol);
        names.Add("CLT Wall");

        //CORE WALL GWB
        matNo = 10; //1/2" Gypsum Board
        tCost = tCost + CostIn[matNo] * 2 * AreaCLTCoreWall;
        CostOut.Add(CostIn[matNo] * 2 * AreaCLTCoreWall);
        names.Add("Core Wall GWB");

        //FACADE WALL WATERPROOFING
        matNo = 11; //3 mil Polyethylene
        tCost = tCost + CostIn[matNo] * AreaCLTFacWall;
        CostOut.Add(CostIn[matNo] * AreaCLTFacWall);
        names.Add("Facade Waterproofing");

        //FACADE WALL INSULATION
        matNo = 12; //Extruded Polystyrene
        tCost = tCost + CostIn[matNo] * AreaCLTFacWall;
        CostOut.Add(CostIn[matNo] * AreaCLTFacWall);
        names.Add("Facade Insulation");

        //FACADE WALL CLADDING
        matNo = 13; //Pine T&G Siding
        tCost = tCost + CostIn[matNo] * AreaCLTFacWall;
        CostOut.Add(CostIn[matNo] * AreaCLTFacWall);
        names.Add("Facade Cladding");

        //CURTAIN WALL
        matNo = 14; //Double Glazed Windows & aluminum mullions
        tCost = tCost + CostIn[matNo] * AreaGlass;
        CostOut.Add(CostIn[matNo] * AreaGlass);
        names.Add("Curtain Wall");

        //TIMBER STEEL CONNECTIONS
        //1% of timber cost
        tCost = tCost + 0.01 * (CostOut[0] + CostOut[1] + CostOut[2] +
        CostOut[5] + CostOut[9]);
        CostOut.Add(0.01 * (CostOut[0] + CostOut[1] + CostOut[2] +
        CostOut[5] + CostOut[9]));
        names.Add("Steel Connections");

        //Itemized Output
        DataTree<System.Object> CostItems = new DataTree<System.Object>();
        DataTree<System.Object> CostItemsSF = new DataTree<System.Object>();
        GH_Path A = new GH_Path(0);
        GH_Path B = new GH_Path(1);

        for (int i = 0; i < names.Count; i++)
        {
            CostItems.Add(names[i], A);
            CostItemsSF.Add(names[i], A);
            CostItems.Add(CostOut[i], B);
            CostItemsSF.Add(CostOut[i] / (AreaCLTFloor + AreaCLTRoof), B);
        }

        //Output per SF
        double CostSFOut = tCost / (AreaCLTFloor + AreaCLTRoof);

        //OUTPUTS
        TotCost = tCost;
        CostperSF = CostSFOut;
        OutputTot = CostItems;
        OutputSF = CostItemsSF;
    }
}
```

DESIGN EXAMPLES

D

The following sections contain design examples showing the structural design of slabs, beams, column, and shear walls in the buildings.

SLAB DESIGN:

INPUTS:

TOTAL SUPERIMPOSED LOAD: 146 PSF (FACTORED, (1.2D+1.6L))

SUPERIMPOSED DEAD LOAD: 35 PSF (UNFACTORED)

LIVE LOAD: 65 PSF (UNFACTORED)

SPAN LENGTH: 15'

CONCRETE THICKNESS: 2"

$\lambda = 0.8$ NDS TIME EFFECT FACTOR, SPECIFIED BY LOAD CASE

CHECK 5 1/8" THICK PANEL, MAXIMUM LENGTH 64'

* ALGORITHM STARTS WITH THINNEST PANEL, CHECKS AND INCREMENTS TO NEXT THICKEST IF IT FAILS.

FROM MANUFACTURER:

$M_{allowable} = 5{,}800$ lb-ft/ft $\quad f_{b,0} = 1950$ psi

$V_{allowable} = 1{,}470$ lb/ft $\quad f_{v,0} = 135$ psi

CHARRING DEPTH:

$d_{char} = 1.2 * (1.5 * (1)^{0.813}) = 1.8$ inches

DEMANDS:

$M_u = 0.125\ w L^2 = 0.125 \cdot 146 \cdot 15^2 = 4106$ lb-ft/ft (3-SPAN CONTINUOUS)

$V_u = 0.6\ wL = 0.6 \cdot 146 \cdot 15 = 1314$ lb/ft

CAPACITY:

EXCLUDING CHARRING, NON-COMPOSITE:

$\phi M_n = \phi K_F \lambda C_m C_t C_L \cdot f_b \cdot b \cdot d^2/6$

$= 0.85 \cdot 2.54 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1 \cdot 1950 \cdot 12 \cdot 5.125^2/6 = 14{,}744$ lb-ft/ft

$\phi V_n = \phi K_F \lambda C_m C_t \cdot 2/3\ bd \cdot f_v$

$= 0.75 \cdot 2.88 \cdot 0.8 \cdot 1 \cdot 1 \cdot 2/3 \cdot 12 \cdot 5.125 \cdot 135 = 9564$ lb/ft

INCLUDING CHARRING, NON-COMPOSITE:

$\phi M_n = \phi K_F \lambda C_m C_t C_L f_b \cdot b d^2/6$

$= 0.85 \cdot 2.54 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1 \cdot 1950 \cdot 12 \cdot (5.125-1.8)^2/6$

$= 6206$ lb-ft/ft

$\phi V_n = \phi K_f \lambda C_m C_t \cdot f_v \cdot 2/3\ b \cdot d$

$= 0.75 \cdot 2.88 \cdot 0.8 \cdot 1 \cdot 1 \cdot 135 \cdot 2/3 \cdot 12 \cdot (5.125-1.8)$

$= 6205$ lb/ft

EXCLUDING CHARRING, COMPOSITE:

CALCULATE POSITION OF NEUTRAL AXIS:

$$y_{NA} = \frac{t_{conc} \cdot w \cdot E_{conc} \cdot y_{conc} + t_{CLT} \cdot w \cdot E_{CLT} \cdot y_{CLT}}{t_{conc} \cdot w \cdot E_{conc} + t_{CLT} \cdot w \cdot E_{CLT}}$$

$E_{conc} = 57000\sqrt{f'_c} = 3{,}122{,}018.1$ psi

$f'_c = 3000$ psi

$$y_{NA} = \frac{2 \cdot 12 \cdot 3122018 \cdot 6.125 + 5.125 \cdot 12 \cdot 1400000 \cdot 2.5625}{2 \cdot 12 \cdot 3122018 + 5.125 \cdot 12 \cdot 1400000}$$

$y_{NA} = 4.22$ in

$E_{CLT} = 1{,}400{,}000$ psi

CALCULATE TRANSFORMED SECTION:

$n = E_{conc}/E_{CLT} = 3122018/1400000 = 2.23$

$$I_{tr} = \left(\frac{1}{12} bh^3\right)_{CONC} + \left(Ad^2\right)_{CONC} + \left(\frac{1}{12} bh^3\right)_{CLT} + \left(Ad^2\right)_{CLT}$$

$$= \frac{1}{12} \cdot 12 \cdot 2.23 \cdot 2^3 + 12 \cdot 2.23 \cdot 2 \cdot (6.125-4.22)^2$$

$$+ \frac{1}{12} \cdot 12 \cdot 5.125^3 + 12 \cdot 5.125 \cdot (2.5625-4.22)^2$$

$= 516$ in^4

DEMANDS:

$$f_{b,u} = M_u \cdot y / I$$

FOR CLT:

$$f_{b,u} = 4106 \text{ lb-ft/ft} \cdot 4.22 \text{ in} / 516 \text{ in}^4 = 403 \text{ psi}$$

FOR CONCRETE

$$f_{b,u} = 4106 \cdot (7.125 - 4.22) / 516 = 23 \text{ psi}$$

CAPACITY

$$\phi f_{b,n} = \phi K_F \lambda \cdot C_m \cdot C_t \cdot C_i \cdot f_{b,o} \quad \text{(CLT)}$$

$$= 0.85 \cdot 2.54 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1950 = 3368 \text{ psi}$$

$$\phi V_n = \phi K_F \lambda C_m C_t \, 2/3 \, bd f_v + \phi 2 \lambda \sqrt{f'_c} \, b \cdot t_{conc}$$

$$= 0.75 \cdot 2.88 \cdot 0.8 \cdot 1 \cdot 1 \cdot 2/3 \cdot 12 \cdot 5.125 \cdot 135$$

$$+ 0.75 \cdot 2 \cdot 1 \cdot \sqrt{3000} \cdot 12 \cdot 2$$

$$= 11536 \text{ lb/ft}$$

$$\phi f_c = 0.9 \cdot 3000 = 2700 \text{ psi}$$

INCLUDING CHARRING, COMPOSITE:

NEUTRAL AXIS:

$$y_{NA} = \frac{2 \cdot 12 \cdot 3122018 \cdot (1 + 5.125 - 1.8) + (5.125 - 1.8) \cdot 12 \cdot 1400000 \cdot 1.6625}{2 \cdot 12 \cdot 3122018 + (5.125 - 1.8) \cdot 12 \cdot 1400000}$$

$$y_{NA} = 3.19 \text{ in}$$

TRANSFORMED SECTION

$$I_{tr} = \frac{1}{12} \cdot 12 \cdot 2.23 \cdot 2^3 + 12 \cdot 2.23 \cdot 2 \cdot (4.325 - 3.19)^2$$

$$+ \frac{1}{12} \cdot 12 \cdot 3.325^3 + 12 \cdot 3.325 \cdot (1.6625 - 3.19)^2$$

$$= 216.6 \text{ in}^4$$

DEMANDS:

FOR CLT:

$$f_{b,u} = 4106 \cdot 3.19 / 216.6 = 726 \text{ psi} \quad \text{(CLT)}$$

FOR CONCRETE:

$$f_{b,u} = 4106 \cdot (5.325 - 3.19) / 216.6 = 40.5 \text{ psi} \quad \text{(CONCRETE)}$$

CAPACITY:

$\phi f_{b,n} = 3368$ psi (CLT)

$\phi f'_c = 2700$ psi (CONCRETE)

$$\phi V_n = \phi K_F \lambda C_m C_t \, 2/3 \, bd f_v + \phi 2 \lambda \sqrt{f'_c} \, b \, t_{conc}$$

$$= 0.75 \cdot 2.88 \cdot 0.8 \cdot 1 \cdot 1 \cdot 2/3 \cdot 12 \cdot 3.325 \cdot 135$$

$$+ 0.75 \cdot 2 \cdot 1 \cdot \sqrt{3000} \cdot 12 \cdot 2$$

$$= 8177 \text{ lb/ft}$$

DEMAND CAPACITY RATIOS.

	NON-COMPOSITE NO CHARRING	NON-COMPOSITE CHARRING	COMPOSITE NO CHARRING	COMPOSITE CHARRING
BENDING	0.28	0.666	0.12	0.22
SHEAR	0.14	0.21	0.11	0.16

DEFLECTIONS AND VIBRATIONS.

NON-COMPOSITE

$$\Delta_{D+L} = 0.0069 \, (w) \, L^4 / EI$$

$$= 0.0069 \left(35 + 13.7 + (65) \frac{lb}{ft}\right) (15)^4 \text{ ft}^4 / 184000000 \text{ lb-in}^2 \cdot \frac{144 \text{ in}^2}{\text{ft}^2}$$

$$= 0.03 \text{ ft}$$

$$\Delta_L = 0.0069 (65) \cdot (15)^4 \cdot 144 / 184,000,000 = 0.02 \text{ ft}$$

$EI = 184,000,000$ lb-in^2/ft (FROM MANUFACTURER)

$$EI_{APP} = \frac{1}{\left(\frac{1}{EI}\right) + \left(\frac{11.52}{GA \cdot L^2}\right)}$$

$$= \frac{1}{\left(\frac{1}{184{,}000{,}000}\right) + \left(\frac{11.52}{690000\,(15.12)^2}\right)} = 94{,}448{,}669$$

GA = 690,000 lb/ft (FROM MANUFACTURER)

$$\text{VIBRATION SPAN} = EI_{APP}^{0.293} \Big/ 12.05\,(\rho A)^{0.122}$$

$$= 94{,}448{,}669^{0.293} \Big/ 12.05 \cdot (0.5 \cdot 5.125 \cdot 12)^{0.122} = 11.9 \text{ ft}$$

$\rho = 0.5$ (DOUGLAS FIR SPECIFIC GRAVITY)

COMPOSITE SLABS:

$$\Delta_{D+L} = 0.0069 \cdot (35 + 13.7 + 65) \cdot (15)^4 \cdot 144 \Big/ 1400000 \cdot 516$$

$$= 0.008 \text{ ft}$$

$$\Delta_L = 0.0069\,(65)\,(15)^4 \cdot 144 \Big/ 1400000 \cdot 516$$

$$= 0.005 \text{ ft}$$

$$EI_{APP} = \frac{1}{\left(\frac{1}{1400000 \cdot 516}\right) + \left(\frac{11.52}{690000\,(15 \cdot 12)^2}\right)} = 526{,}434{,}224$$

$$\rho = \frac{\rho_{conc}\, t_{conc} \cdot 144\,in^2 + \rho_{LT} \cdot t_{LT} \cdot 144\,in^2}{(t_{conc} + t_{LT}) \cdot 144\,in^2} \cdot \frac{1}{\rho_{H_2O}}$$

$$= \frac{150\,lb/ft^3 \cdot 2in \cdot 144in^2 + 33\,lb/ft^3 \cdot 5.125in \cdot 144in^2}{7.125\,in \cdot 144\,in^2} \cdot \frac{1}{62.43\,lb/ft^3}$$

$$= 1.05$$

$$\text{VIBRATION SPAN} = \frac{526434224^{0.293}}{(1.05 \cdot 12 \cdot 7.125)^{0.122} \cdot 12.05} = 17.2 \text{ ft}$$

DEFLECTION AND VIBRATION DEMAND/CAPACITY RATIOS.

	NON-COMPOSITE	COMPOSITE
Δ_{D+L}	0.48	0.13
Δ_L	0.48	0.12
VIBRATION	1.26	0.87

BEAM DESIGN

INPUTS: $V_u = 19.9\ K$
$M_u = -74.5\ K\text{-}ft$
$Length = 15\ ft$

CHECK 5.5x16.5 BEAM
*ALGORITHM ITERATES INCREASING BEAM AREA UNTIL SECTION PASSES.

NON-COMPOSITE:
CAPACITY:
$\phi V_n = \phi K_F \lambda \cdot C_m \cdot C_t \cdot C_{VR} \cdot 2/3\ b \cdot d \cdot f_{v,o}$

$= 0.75 \cdot 2.88 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1 \cdot 2/3 \cdot 5.5 \cdot 16.5 \cdot 265$

$= 27{,}704\ lb$

$\phi M_n = \phi K_F \lambda \cdot C_m \cdot C_t \cdot C_L \cdot C_V \cdot C_{fu} \cdot C_i \cdot bd^2/6 \cdot f_{b,o}$

$= 0.85 \cdot 2.54 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 5.5 \cdot 16.5^2/6 \cdot 2400$

$= 86{,}208\ lb\text{-}ft$

DEMAND CAPACITY RATIOS:

SHEAR $19.9/27.7 = 0.72$

BENDING $74.5/86.2 = 0.86$

FIRE RATING (ONLY IF CHARRING IS INCLUDED)

$K_e \cdot L/d = 1 \cdot (15 \cdot 12)/16.5 = 10.9$

$Z = \frac{2}{3}(D/c)^2 - 1.6(D/c) + 2.1333$

$= \frac{2}{3}(0.86)^2 - 1.6(0.86) + 2.1333 = 1.25$

$t = 2.54 \cdot Z \cdot w\ (4 - w/d)$

$= 2.54 \cdot 1.25 \cdot 5.5\ (4 - 5.5/16.5)$

$= 64\ min > 60\ min?$

DEFLECTIONS

$\Delta_{FACTORED} = \frac{M_u \cdot \ldots \cdot 40 \cdot L^2}{384\ E\ I}$

$= \frac{(74.5 \cdot 12)\ K\text{-}in \cdot 40 \cdot 15^2\ ft^2 \cdot 144\ in^2/ft^2}{384 \cdot 1800\ ksi \cdot 1/12 \cdot 5.5 \cdot 16.5^3\ in^4}$

$= 0.81\ in$

$\Delta_{D+L} = 0.703 \cdot 0.81 = 0.57\ in \quad L/240 = 0.75\ in?$

$\frac{\text{UNFACTORED D+L}}{\text{FACTORED LOADS}} = 0.703$

$\Delta_L = 0.39 \cdot 0.81 = 0.32\ in \quad L/360 = 0.5\ in?$

$\frac{\text{UNFACTORED L}}{\text{FACTORED LOADS}} = 0.39$

COMPOSITE BEAM- CLT SLAB.

EFFECTIVE FLANGE WIDTH:

$b_{eff} = min \begin{cases} 0.25\ L_{beam} = 0.25 \cdot 15ft = 45\ in \\ 16 \cdot t_{slab} + W_{beam} = 16 \cdot 5.125in + 5.5\ in = 87.5\ in \\ slab\ span = 15\ ft = 180in \end{cases}$

$b_{eff} = 45\ in$

CALCULATE LOCATION OF NEUTRAL AXIS:

$$y_{NA} = \frac{t_{CLT} \cdot b_{eff} \cdot E_{CLT} \cdot y_{CLT} + d_{beam} \cdot w_{beam} \cdot E_{beam} \cdot y_{beam}}{t_{CLT} \cdot b_{eff} \cdot E_{CLT} + d_{beam} \cdot w_{beam} \cdot E_{beam}}$$

$$= \frac{5.125 \cdot 45 \cdot 1400 \cdot 19.0625 + 16.5 \cdot 5.5 \cdot 1800 \cdot 8.25}{5.125 \cdot 45 \cdot 1400 + 16.5 \cdot 5.5 \cdot 1800}$$

$$= 15.4 \text{ in}$$

CALCULATE TRANSFORMED SECTION

$$n = E_{CLT}/E_{GLB} = 1400/1800 = 0.78$$

$$I_{tr} = \left(\frac{1}{12} bh^3\right)_{CLT} + \left(Ad^2\right)_{CLT} + \left(\frac{1}{12} bh^3\right)_{GLB} + \left(Ad^2\right)_{GLB}$$

$$= \left(\frac{1}{12} \cdot 45 \cdot 5.125^3\right) + \left(45 \cdot 5.125 \cdot (19.0625 - 15.4)^2\right)$$

$$+ \left(\frac{1}{12} \cdot 5.5 \cdot 16.5^3\right) + \left(5.5 \cdot 16.5 \cdot (8.25 - 15.4)^2\right)$$

$$= 10296 \text{ in}^4$$

CAPACITY:

$$\phi V_n = \phi K_F \lambda C_m C_t C_{vr} \cdot 2/3 \, b d f_{tr}$$

$$= 27{,}704 \text{ lb}$$

$$\phi M_n = \phi K_F \lambda C_m C_t C_L C_V C_{fu} C_i f_{b,0} \cdot \frac{I_{tr}}{y_{NA}}$$

$$= 0.85 \cdot 2.54 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 2400 \cdot \frac{10296}{15.4}$$

$$= 230951 \text{ lb-ft}$$

DEMAND-CAPACITY RATIOS

SHEAR $19.9/27.7 = 0.72$

BENDING $74.5/231 = 0.32$

FIRE RATING (ONLY IF CHARRING IS INCLUDED)

$$K_e L/d = 1 \cdot 15.12/16.5 = 10.9$$

$$Z = \frac{2}{3} \cdot 0.72^2 - 1.6 \cdot 0.72^2 + 2.1333$$

$$= 1.165$$

$$t = 2.54 \cdot 1.65 \cdot 5.5 \cdot (4 - 5.5/16.5)$$

$$= 84.5 \text{ min}$$

DEFLECTIONS

$$\Delta_{factored} = \frac{(74.5 \cdot 12) \cdot 40 \cdot 15^2 \, 144}{384 \cdot 1400 \cdot 10296}$$

$$= 0.21 \text{ in}$$

$\Delta_{D+L} = 0.15$ in $< 4/240 = 0.75$ in?

$\Delta_L = 0.08$ in $< 4/360 = 0.5$ in?

COLUMN DESIGN

INPUTS: $P_u = 169.2\ K$

$V_u = M_u \simeq 0$

LENGTH = 12 ft

CHECK: 8.75×12 COLUMN

*ALGORITHM ITERATES INCREASING COLUMN AREA UNTIL SECTION PASSES.

ADJUSTMENT FACTORS:

$$F_{cE} = 0.822 \cdot E_{min} \Big/ \left(\frac{l_e}{d}\right)^2$$

$$= \frac{0.822 \cdot 850{,}000}{\left(\frac{12 \cdot 12}{12}\right)^2} = 4852\ PSI$$

$$F_c^* = \phi K_F \lambda \cdot C_m \cdot C_t \cdot F_c$$

$$= 0.9 \cdot 2.4 \cdot 0.8 \cdot 1 \cdot 1 \cdot 1650$$

$$= 2851\ PSI$$

$$C_p = \frac{(1 + F_{cE}/F_c^*)}{2c} - \sqrt{\left(\frac{1 + F_{cE}/F_c^*}{2c}\right)^2 - \frac{F_{cE}/F_c^*}{c}}$$

$$= \frac{\left(1 + \frac{4852}{2851}\right)}{2 \cdot 0.9} - \sqrt{\left(\frac{1 + \frac{4852}{2851}}{2 \cdot 0.9}\right)^2 - \frac{4852/2851}{0.9}}$$

$$= 0.9$$

CAPACITY

$$\phi P_n = \phi K_F \lambda \cdot C_m C_t C_p f_c' \cdot A$$

$$= 0.9 \cdot 2.4 \cdot 0.8 \cdot 1 \cdot 1 \cdot 0.9 \cdot 1650 \cdot 8.75 \cdot 12$$

$$= 269438\ lb$$

DEMAND CAPACITY RATIO:

AXIAL $169.2/269.4 = 0.63$

FIRE RATING (ONLY IF CHARRING IS INCLUDED)

$$K_e \cdot L/d = 1 \cdot 12 \cdot 12/12 = 12$$

$$Z = \frac{2}{3} \cdot 0.63^2 - 1.6 \cdot 0.63 + 1.933$$

$$= 1.19$$

$$t = 2.54 \cdot 1.19 \cdot 8.75 \cdot (3 - 8.75/12)$$

$$= 60.1\ min$$

$V_u \simeq M_u \simeq 0$ NO COMBINED BENDING/AXIAL LOADING

SHEAR WALL DESIGN

INPUTS: BASE SHEAR = 2148 K (SEE SPREADSHEET APPENDIX B).

USE 12.18 in THICK WALL, 20' LONG

V_R = 11624 lb/ft (FROM MANUFACTURER)

BUILDING CENTER OF MASS (INPUT)

$I = 1/12 \cdot bh^3 = 1/12 \cdot 12.18 \cdot (20 \cdot 12)^3 = 14{,}031{,}360\ in^4$

$$k_{WALL} = 1 \Big/ \left(\frac{L^3}{3EI} + \frac{L}{KGA} \right)$$

$$= 1 \Big/ \left(\frac{(20 \cdot 12)^3}{3 \cdot 1800 \cdot 14031360} + \frac{(20 \cdot 12)}{0.833 \cdot 50{,}000 \cdot 12.18 \cdot 20 \cdot 12} \right)$$

$= 5422.4$ lb/in

BUILDING CENTER OF RIGIDITY

$$COR = \sum_{i=1}^{n_{walls}} k_{wall,i} \cdot x_{wall\ offset\ from\ cog,i} \Big/ \sum_{i=1}^{n_{walls}} k_{wall,i}$$

ACCIDENTAL TORSION

$$T = \max \begin{Bmatrix} 0.05 \\ COR/BLDG\ DIM \end{Bmatrix} = 0.05\ \%$$

FORCES IN WALL (RIGID DIAPHRAGM ASSUMED)

$$V_{wall} = V_{base\ shear} \cdot k_{wall} \cdot (1 + 2 \cdot T) \Big/ \sum_{i=1}^{n_{wall}} k_{wall,i}$$

$$= 2148 \cdot 5422.4 \, (1 + 2 \cdot 0.05) \Big/ 21689.6$$

$= 590.7$ K

$M_{wall} = V_{wall} \cdot h = 590.7 \cdot 40 = 23{,}628$ K-ft

$$h = \sum_{i=1}^{n_{floors}} F_{story,i} \cdot h_i \Big/ \sum F_{story,i} = 40\ ft$$

$$S_{wall} = \frac{t \cdot L^2}{6} = 12.18 \, (20 \cdot 12)^2 / 6 = 116928\ in^3$$

DEMAND/CAPACITY

$V_u \leq \phi V_n$

$590.7\ K \leq \phi K_F \lambda C_m C_t \cdot V_R \cdot L_{wall}$

$590.7\ K \leq 0.75 \cdot 2.88 \cdot 1 \cdot 1 \cdot 1 \cdot 11.624 \cdot 20$

$590.7\ K \leq 502.2\ K$ → MORE WALL NEEDED.

$f_{b,u} \leq \phi f_{b,n}$

$\frac{M}{S} \leq \phi K_F \lambda \cdot C_m C_t f_c$

$\frac{236028 \cdot 12}{116928} \leq 0.9 \cdot 2.4 \cdot 1 \cdot 1 \cdot 1 \cdot 1150$

$2424\ psi \leq 2484\ psi$

ADD STEEL REINFORCING AT END OF WALL FOR TENSION

1" ϕ ROD $A = \pi/4 \cdot 1^2 = 0.79\ in^2$

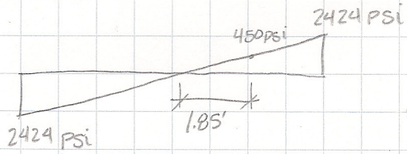
Strength = $0.9 \cdot 0.79\ in^2 \cdot 60\ ksi = 42\ K$

$f_{t,CLT} = 450\ psi$

T = tensile force · $L_{tension}$ · t_{wall}

$T = (2424 - 450) \cdot \frac{1}{2} \cdot (10 - 1.85) \cdot 12 \cdot 12.18$

$= 1175.7\ K$



(28) 1" RODS AT 3.5" O.C. ALONG BOTH ENDS OF WALL.

CHECK DRIFTS.

STORY DISPLACEMENT = $F_{story} \cdot (1 + 2T) \cdot C_d / k_{story} \cdot I_e$

$= 200 \cdot (1.1) \cdot 2 / 21689.6 \cdot 1$

$= 20.3\ in$

% DRIFT = $\frac{20.3}{12 \cdot 12} = 14\% \leq 2\%$ (SINGLE STORY)

BUILDING DRIFT

% DRIFT = $\sum_{i=1}^{n_{stories}} \Delta_{Story,i} / h_{story} \leq 2\%$

Wall-Foundation Connection Design (per Linear Foot)		
CLT Panel		
G	0.5	NDS T12.3.3A Douglas Fir-Larch
E	1400000	psi
Panel Thickness	12.18	in
Dowel Type Fasteners		
Fastener Diameter	0.5	in
Hole Diameter	0.5625	in
Fastener Length	10	
Fastener Spacing (Along Wall)	3	in
s_{min}	1.5	in
Number of Rows	11	
Row Spacing	3	in
s_{min}	1.5	in
Edge Distance	3	in
# of Screws per Foot	44	
Steel Plate		
$f_{y,pl}$	36000	psi
$f_{u,pl}$	50000	psi
E	29000000	psi
Plate Thickness	0.5	in
Plate Edge Distance	2	in
Plate Height	35	in
Concrete Reinforcing		
Check Timber		
C_M	1	
C_t	1	
R_{EA}	0.85	
γ	95459	
u_{vert}	1	
u_{horiz}	1	
m_{vert}	0.97	
m_{horiz}	0.95	
n_{vert}	11.00	
n_{horiz}	4.00	
$C_{g,vert}$	0.99	
$C_{g,horiz}$	1.00	
C_Δ	0.86	
C_{eg}	1	
C_{di}	1	
C_{tn}	1	
K_F	3.32	
ϕ	0.65	
λ	1	

Double Shear T12.3.1A		
l_m	0.5	in
l_s	5.8	in
F_{em}	36000	psi
F_{es}	3150	psi
F_{yb}	45000	psi
R_e	11.43	
k_3	0.50	
K_θ	1.25	
Z_{Im}	*Bolt Hole Bearing	lb
Z_{Is}	3679	lb
Z_{IIIs}	7865	lb
Z_{IV}	4661	lb
Z	3679	lb
Z'	6737	lb/fastener
Connection Capacity/ft	296.4	Kips/ft
Check Steel Plate		
Bolt Hole Bearing	16.2	K/bolt
Plate Tension	194.4	K/ft
Plate Shear	110	K/ft