Software for Facilitating the Creation of Parametric Urban Resource Models

by

Sean M. Cockey

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

at the Massachusetts Institute of Technology

June 2014

© Sean M. Cockey. All rights reserved

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature redacted

Department of Mechanical Engineering
January 17, 2014

Certified by: __________

Signature redacted

Kent Larson
Director, Changing Places
Co-Director, City Science Initiative
MIT Media Lab

Signature redacted

Accepted by: __________

Anette (Peko) Hosoi
Associate Professor of Mechanical Engineering
Undergraduate Officer
Software for Facilitating the Creation of Parametric Urban Resource Models

by

Sean M. Cockey

Submitted to the Department of Mechanical Engineering On January 17, 2014 in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Mechanical Engineering

Abstract

This thesis describes a new software tool to facilitate Parametric Urban Resource Modeling, a method for quantitatively studying and improving the distribution of resources in a city. The software is intended to help users with no CAD or programming experience make digital Parametric Urban Resource Models of cities and optimize them algorithmically using a pre-determined set of rules. These models may help urban planners understand how to raise the population density of an area while maintaining or improving its livability, a critical challenge in our rapidly-urbanizing world. Preliminary feedback regarding this limited early prototype of the software has been promising. Parametric Urban Resource Modeling software such as this may eventually become an important early step in the urban design process, potentially saving time and improving the quality of the final city designs.

Thesis Supervisor: Kent Larson
Title: Director, Changing Places and Co-Director, City Science Initiative, MIT Media Lab
Acknowledgements

The author is grateful to the Changing Places group at the MIT Media Lab for supporting this thesis and for supporting the author’s prior work with the group as part of the Undergraduate Research Opportunities Program (UROP). The author would like to acknowledge Prof. Kent Larson for advising this thesis and for providing guidance regarding the research and the paper. He would also like to recognize the advice and feedback of J. Ira Winder and Dr. Ryan Chin. Additionally, the author would like to thank Cynthia Wilkes for her dedication to coordinating group logistics.
# Table of Contents

Abstract .......................................................................................................................................................... 3  
Acknowledgements ....................................................................................................................................... 4  
Table of Contents ....................................................................................................................................... 5  
Introduction .................................................................................................................................................. 7  
Background .................................................................................................................................................. 11  
  Compact Urban Cells .................................................................................................................................. 11  
  Parametric Urban Resource Modeling .......................................................................................................... 12  
  Digitization of Parametric Urban Resource Models ..................................................................................... 14  
Problem ......................................................................................................................................................... 16  
Approach ....................................................................................................................................................... 18  
User Experience ............................................................................................................................................ 20  
  Setting Up .................................................................................................................................................. 20  
  Building the Initial Parametric Urban Resource Model Using the Drawing Features ............................. 26  
Statistical Analysis ......................................................................................................................................... 32  
Optimization ................................................................................................................................................. 34  
Wrapping Up ................................................................................................................................................ 40  
Evaluation ...................................................................................................................................................... 41  
  Author Self-Assessment ............................................................................................................................... 41  
  User Assessment ....................................................................................................................................... 42  
  Inclusion in the Design Process .................................................................................................................. 43  
Future Work .................................................................................................................................................. 44  
Conclusion .................................................................................................................................................... 46  
Appendix A: Technical Description of the User Interface ........................................................................ 47  
  Registries .................................................................................................................................................. 47  
  Set-Up Functions ..................................................................................................................................... 49  
  Drawing Functions .................................................................................................................................. 54  
  Exporting Buildings to Rhino ....................................................................................................................... 59  
  Statistics .................................................................................................................................................... 60
Introduction

The world is swiftly urbanizing: for the first time, more than half of the world’s population lives in cities, and this proportion is expected to grow rapidly over the next few decades. Rapid urbanization comes with a host of problems that need to be addressed in order for cities to be livable. Foremost among these is the issue of overcrowding. If cities grow rapidly and without direction they will quickly become unlivable; the population will demand more resources than the city can provide. Traffic will clog the roads, energy supplies will run out, and people will get frustrated with the difficulty of everyday life. Density does not, however, need to be a curse. High concentrations of population can allow for economies of scale, giving people access to a greater variety of resources. The more people living in an area, the more types of businesses can be sustained: there will be more restaurants, more retail stores, and more entertainment venues. The proximity between home, work, and other resources will create conveniences that may allow people to get rid of their private cars. The challenge that now faces urban planners is to design cities that facilitate the positive aspects and eliminate the negative aspects of density. With the right design tools, urban planners can design the denser cities of tomorrow to be more livable than the cities of today.2

Poor urban design is a huge problem that will only get worse as urban densities increase if city design methods are not improved. Many cities are not planned at all; other than generic zoning laws, these cities are allowed to grow in a haphazard manner without respect for livability. This form of unplanned expansion can have serious detrimental effects on the urban community. In the United States, most cities are unworkable: not having a car can be a huge inconvenience since the combination of walking and using public transportation cannot get a person to every place he or she needs to go. Unfortunately, many of these cities suffer from traffic congestion and expensive, scarce parking that can
make owning a car a heavy burden. These issues make it hard for residents to conduct their business and make the city an unpleasant place to live.

Kendall Square in Cambridge, Massachusetts, is a prime example of a location that suffers from severe livability issues due to a lack of quality urban planning. The square is dominated by corporate offices, and there is a shortage of many of the resources needed to make the area livable. There is insufficient housing, and the housing that exists is very expensive, out of the reach of most young people. For those who do live in the square, there is a surprising lack of resources. Dining options are minimal, and the closest grocery store is over a kilometer from the subway stop. Retail is almost non-existent, with the closest significant shopping located in a mall that is again over a kilometer away from the subway. Despite its growing population, Kendall Square is not able to sustain a high-quality of urban living at an affordable price. Accordingly, many MIT students who choose to live off-campus would prefer to live significantly farther away from campus than to live in Kendall Square. With its proximity to MIT, Kendall Square has the potential to be a great innovation hub, but until its livability issues are fixed, it will not be able to attract the smart young professionals and students that it needs to help it achieve this goal.

Clearly, there is a need to re-examine the ways in which cities are planned, especially as urban densities continue to rise. New planning methods are needed to prevent urban sprawl; specifically, the distribution of city resources must be taken into account when designing cities. It is no longer sufficient to assume that just because a resource exists somewhere in the city that it will be accessible to all residents. Each residential area should have all the resources people need within a reasonable walking distance. The question then becomes: what is the best way to analyze and optimize the distribution of resources in the city? With current techniques, it can be extremely difficult for urban designers to understand the complexities of resource distribution throughout the city and even more difficult to plan
the responsible expansion of the city’s population density, since many interconnected resources are required to be well-distributed in order to make the city livable.

Parametric Urban Resource Modeling is a new urban design method that attempts to provide a quantitative way of studying and improving the distribution of city resources. The main concept is that all city resources can be represented approximately by discrete units. For instance, one unit of “apartments” could be the average size of a two-person apartment and one unit of “fire stations” could be the average size of a fire station needed to serve one thousand people. From here on, the word “resources” will be used to describe anything man-made in the city with which residents interact directly or indirectly. This includes houses, offices, energy sources, retail stores, restaurants, roads, entertainment venues, parking lots, etc. This word is not being used to describe natural resources such as oil, coal, etc., since residents do not need the sources of these natural resources to be located in a city (but the word would describe a gas station or coal-fired power plant, since residents may need these located in or near the city).

In Parametric Urban Resource Modeling, units of all resource types have consistent volumes, but each resource has its own “usage coefficient” that translates the volume back into the terms of how many people it can serve. For instance, a usage coefficient of 10 means that 10 people can use one unit of the resource. These “usage coefficients” are still the subject of ongoing research: it will take years of study to come up with meaningful coefficients that adequately describe the usage of resources. It may even be the case that the coefficients vary with location. However, reasonable approximations of the coefficients will be sufficient for generating first-order analyses and optimizations that will still provide valuable information about resource distribution. The Changing Places group at the MIT Media Lab has tried to estimate these coefficients for its City Science workshops, but has not gone into the detail
necessary to assign a coefficient to every resource included in the software about which this thesis is
based. Thus, the coefficients in the current program are arbitrary and are included simply for the sake
of demonstrating the capabilities of the program.

This modeling technique creates a quantitative spatial model of the city’s resources that represents both
the quantity and distribution of resources. Mathematical relations can then be established between the
resources, allowing their distribution to be quantified and algorithmically optimized. More information
on Parametric Urban Resource Modeling can be found in the Background section.

This thesis studies Parametric Urban Resource Modeling and attempts to create a design tool to assist
urban planners in conducting this design process. The software described in the following sections is
intended to provide designers with an intuitive user interface that helps them create Parametric Urban
Resource Models of cities in Rhinoceros 3D CAD software. It also intends to provide statistical output
and optimization functions to help users increase the density of the city in a responsible way. The
optimization algorithms accept a desired target population and then determine the quantity and
distribution of resources that should be added to the city to support the target population. This tool is
far from complete, and significant additional work is required to turn this proof-of-concept into a design
tool. Eventually, it is hoped that this tool will be used by urban planners as a first step in the design
process when they are planning to increase the density of a city. It could also be used as a way to
generate templates for designing efficient urban neighborhoods from scratch. It is not expected that
this tool will be used to create the final design of a city; rather, it will help designers understand what
resources are needed in order to make their city function before they start the detailed, more
traditional design processes. It is hoped that with this tool, quantitative modeling will help eliminate
some of the uncertainty in the urban design process.
Background

Rapid population growth and urbanization is a serious problem that is straining the resources of the world’s cities. Although this problem is prevalent in the United States, it is far more of an issue in developing nations. Overcrowding and accelerating industrialization is creating pollution, congestion, and terrible urban conditions. Something must be done to manage urban growth and ensure that cities are places in which people want to live. Urban planning is becoming more critical than ever, but traditional design methods and assumptions are no longer adequate to deal with the unprecedented urbanization being witnessed today.

Compact Urban Cells

As urban densities increase it is imperative to create livable urban areas. A design concept called the Compact Urban Cell is one attempt to implement this goal. A Compact Urban Cell is a small high-density area that provides everything the residents need and is optimized for walking and livability. In theory, a person should be able to live, work, shop, dine and seek entertainment within this area. Of course, the person will still have to leave the area for some activities, but the overwhelming majority of his or her needs should be met in this area. Research has shown that a reasonable (and geometrically convenient) size for a Compact Urban Cell is a square of one kilometer on each side. At a moderate walking pace, it takes less than twenty minutes to walk the diagonal of this area, which means that a person could walk between any two points in the area in substantially less time. If designed properly, most residents in a Compact Urban Cell should have almost everything they need located within a convenient ten minute walk of their homes. In an area like this, there is no need for personal cars, since everything can be reached by foot. Public transportation and car sharing programs will provide flexibility when necessary, but they should not be an everyday mode of transportation. The Compact
Urban Cell should support a significantly higher population density than most modern American cities while maintaining or even improving livability, making them an attractive option for future urban development.

**Parametric Urban Resource Modeling**

Special design tools are needed to create Compact Urban Cells. These areas must be highly efficient; each resource must be optimized so nothing is wasted and nothing is left out. Parametric Urban Resource Modeling is one design tool that may play a critical role in designing these neighborhoods. In this type of modeling, all resources are discretized into units of equal volume but different usage value. By assigning numerical values to the resource distribution in a city, the resources can be analyzed mathematically and in theory can be optimized algorithmically.

A simple way to make a Parametric Urban Resource Model is to use LEGO bricks. LEGO bricks come in standard sizes and each size is an increment of a fundamental unit. By assigning a color code to the bricks and usage coefficients to the fundamental unit for each color, one can use the bricks to construct a Parametric Urban Resource Model (Figure 1). The model can then be analyzed; counting bricks gives the designers a rough estimate of how much of each resource exists in the city and where the resources are located. The designers can then perform statistical analysis on the resources, or simply use the model as a visualization tool that will influence design thinking.
Figure 1: A LEGO Parametric Urban Resource Model of Kendall Square made by the author and colleagues during a modeling workshop. The model depicts the area as it exists today and includes a part of MIT's campus (top of picture). Note the abundance of Offices (yellow) and Academic buildings (beige) and the lack of Residential buildings (black), Green Space (green) and Retail stores (white). The fundamental unit is a flat 1 x 1 LEGO brick (circled in red). This unit represents a volume 10 meters on each side and one story tall. Each large grey LEGO base plate represents a one square kilometer area, the size of a standard Compact Urban Cell. This model consists of six such cells arranged in a 2 x 3 grid. The goal of the workshop was for each team to transform its one-square-kilometer area into an efficient Compact Urban Cell (the results of the exercise are shown in Figure 2).

Using these Parametric Urban Resource Models, designers can study how an increase in population will affect resource distribution in the city. Once the new population target is set, blocks can be added to the model to simulate the growth in population. Using simple correlations that they can set, the designers can figure out how many units of each resource must be added and where they should be placed. The finished model will be a good starting point for the design of the expansion of the city (Figure 2). Such a schematic model will not be adequate for a final design, but it will give urban planners a starting concept for roughly how the city should look.
Figure 2: A detailed shot of a revised version of the same Kendall Square Parametric Urban Resource Model showing increased population density and improved resource distribution in an attempt to create efficient Compact Urban Cells. Note the increases in Residential (black), Green Space (green), and Retail (white). Other improvements have been made, such as the addition of CityHome smart housing (blue), skybridges to connect buildings and facilitate walking, and mixed-use buildings that provide multiple resources in one place. Resource distribution has been improved, such as by adding convenient ground-level retail to housing complexes and by placing rooftop gardens on offices and apartment buildings.

Digitization of Parametric Urban Resource Models

While LEGO models can be useful visualization tools, they face many limitations. First and foremost, they are bulky and require substantial time to build. Counting bricks for statistical analysis can be extremely tedious, and every change must be logged by hand. In order to perform any mathematical optimization, the data would have to be entered into some type of spreadsheet, which is inefficient. Thus, the Changing Places group is looking at ways to improve this design process. Rhinoceros 3D ("Rhino") CAD software allows users to make parametric models digitally, so this software was selected as the platform for attempting to digitize Parametric Urban Resource Modeling (note that this thesis uses Rhino Version 5 on Windows 7). Once a standard digital brick is made, the user can copy and paste them to build a digital urban model that mimics the form of a physical LEGO model. The color code is applied by putting the bricks into different colored layers. Unfortunately, this method of digitization of
the LEGO models only solves the size issues; placing and counting bricks is still a tedious and inefficient process and automatic optimization methods are limited.

The CityScope project is attempting to combine physical and digital modeling processes. Using this system, designers would build a physical model out of white LEGOs and then project the appropriate color code onto the model. Since the color code is not dependent on the physical blocks, it is much easier to iterate through different designs. The projection system can also be used to simulate other aspects of city life, such as pedestrian movement or the progression of sunlight during the day.

Digitization of Parametric Urban Resource Models would allow for algorithmic study and optimization of the models. Work has been done on designing cities algorithmically\textsuperscript{7,8,9}, but this work has focused mainly on designing geometry and improving aesthetics. These algorithmic models are also parametric, but they do not consider city resources in nearly as much detail as the Parametric Urban Resource Models created by the Changing Places group. They also have a very different input structure and serve different purposes in the design process. As such, this work was not consulted in the creation of the software described in this thesis.
Problem

With current technology, digital Parametric Urban Resource Models are difficult to create and optimize. There is no dedicated software for making these models, so the designs must be created in commercial CAD software. Rhino is a good choice for making parametric models because it is based on a grid and the global coordinate system is fundamental to the model. Its use of a command line and a scripting language is also valuable because it facilitates the creation of optimization algorithms. Unfortunately, CAD is difficult to learn, especially for designers with minimal engineering background. Rhino in particular can have a steep learning curve since its command-line interface may not be familiar to users of more traditional CAD programs. Not only does Rhino require knowledge of CAD, it also requires at least a basic understanding of programming.

Even if the user is a CAD and programming expert, making digital Parametric Urban Resource Models in Rhino is a tedious process. Blocks must be placed manually, either by creating new shapes or by copying and pasting. Both processes are tedious and require the user to count blocks by eye to make sure that everything is in the right place. It is also difficult to extract statistics from these models; the computer does not adequately track the number and placement of the blocks. Even performing a simple task such as counting the number of blocks in a layer requires the user to write a script of some kind, which is an inefficient way to complete a task that every user will have to do repeatedly. Unless the user is keeping careful track of the blocks in a spreadsheet, he or she will have to recount every time the model is changed.

If the user wants to write scripts to optimize this model, the best option is Grasshopper, a plug-in for Rhino. Once again, however, this software was not designed for Parametric Urban Resource Modeling. It was designed for applying mathematical transformations to shapes to make geometric models. Its
graphical interface makes writing programs slow and tedious; this software lacks significant power and can be very frustrating to use. Every script must start with the extraction of the brick locations, which is cumbersome and inefficient, since the program was not designed for tracking bricks. The process involves extracting points from the geometry, deleting extraneous points, and creating a mesh of points that represent bricks. Having a mesh of points is not as good as having control over individual bricks. Doing this process for many layers is tedious and confusing, and there is no good way to optimize the meshes. Adding new bricks is very difficult; writing optimization algorithms for them in Grasshopper would be a monumental task. Instead of using Grasshopper, the user might be able to create a spreadsheet model that could perform the optimization, but he or she would still have to create that spreadsheet by hand and export the model back into CAD, which would either require a sophisticated knowledge of programming or the patience to build the CAD model by hand.
Approach

Clearly, there is a need for a dedicated software tool for Parametric Urban Resource Modeling. The remainder of this thesis outlines a new software tool attempts to facilitate the creation of digital Parametric Urban Resource Models of cities. This tool could be especially helpful for CAD/programming novices who do not wish to learn a complicated new language or user interface before they begin modeling. The goals of this tool will be two-fold: to simplify the process of placing and counting blocks in a digital Parametric Urban Resource Model, and to provide optimization algorithms to help designers study how to increase the population density of a given urban area. Although this tool is far from finished, the software described herein is a good start and proof-of-concept for this new type of modeling technique. It shows that it is indeed possible to simplify the modeling process and automate the optimization of Parametric Urban Resource Models.

This software has a Microsoft Excel-based user interface. Unlike Rhino and other CAD programs, Excel is a common application. The target audience of professional urban planners and urban design students should have Excel experience due to the ubiquity of the program. Using a combination of simple spreadsheets, buttons, and forms, the Excel interface controls the Rhino model and prevents the user from having to learn the complexities of CAD and programming. The user can draw a city model in 2D simply by highlighting squares in the Excel drawing sheet. Resource types, colors, building heights, and usage coefficients are defined in a simple list, and are easily editable when the user's needs change. When the drawing is finished, the user only has to click one button to export it to Rhino in 3D. A statistical output sheet generates land usage graphs and population statistics with the touch of a button, completely eliminating the need to count blocks.
The software tool also attempts to facilitate the optimization of the digital Parametric Urban Resource Models. By pressing the “Optimize” button, the user will be presented with a form via which he or she can set rules for the optimization. Most importantly, he or she can select a new population density for the area. This simple form eliminates the need for the user to have any knowledge of programming to perform the optimization. When the user clicks “Run,” the program attempts to fill in the new population and resources automatically based on the city’s calculated need for each resource. When the optimization is over, the Rhino model will update without any input from the user.

As of now, the program will optimize a one-square-kilometer area, the size of a standard Compact Urban Cell. Thus, the optimization process can be used as the basis for designing one of these high-efficiency dense areas. To create a whole city, many of these cells can be spliced together. The “User Experience” section will outline a sample use case for this program in which one Compact Urban Cell is designed and the Appendices outline the technical details of the code.
User Experience

The following is a description of a sample use case for this software. In this example, the user will create a Parametric Urban Resource Model for a simple city section and run the optimization program to raise its population density substantially. This city section is not meant to be based on any particular real area. Each brick represents an area of 10 meters square and is one story tall. The grid in Rhino on which the model will be placed is laid out such that each grid square is the size of the footprint of one brick. The total size of the grid is 1 square kilometer, or 100 units by 100 units. The program can accommodate buildings up to 100 stories tall. This forms an imaginary cubic array of slots into which bricks can be placed that is 100 bricks by 100 bricks by 100 bricks. The coordinate system used to define the array is a standard Cartesian xyz system that starts at (0, 0, 0) and goes up to (99, 99, 99). For more on the technical aspects of the program, including a sample of the algorithms that power it, please see the Appendices.

Setting Up

The first step of the design process is to define the resources that will be present in the city and assign them color codes, building sizes, and usage coefficients. This is done by making a simple table in the “layers” tab of the user interface (Figure 3).
Figure 3: The “layers” tab of the user interface. From here, the user can define the types of resources used in the city, assign them color codes, building sizes, and usage coefficients. Over one hundred different resources are included in the program by default. To add another one, the user must simply add another line and enter the resource information in the same format. To delete one, the user must simply delete the entry (being careful not to delete the notes to the side of the resource list). The name of the type of resource goes in Column 2. Its color goes in Column 3 and RGB color code goes in Columns 4-6. Since color names can be ambiguous, the program defines colors using the RGB code; the name is only for user convenience. Each resource is color-coded with a shade of the color used for its category (but for simplicity in this example, the shades are only defined for the “Professional” category). In Column 7, the user enters the usage coefficient, which is the number of people who can use each unit of the resource. In this example, most coefficients are set arbitrarily since the values are irrelevant to the functionality of the software. The order-of-magnitude differences in the coefficients shown will demonstrate that the program does indeed take the coefficients into account when optimizing. Future versions will have more accurate coefficients after more research is done in this area. Column 8 contains a definition of the maximum building size that can be created for this resource during the optimization. The program interprets the verbal size definitions according to the notes in Column 9. For convenient statistical analysis, resources are grouped into categories that facilitate the grouping of data into graphs. The categories are listed in Column 1. Any resource without a defined category is part of the category listed immediately above the resource; the empty space in this column helps users visualize the groupings.
Next, the user can define any bodies of water that exist in the city. The optimization algorithms will not place buildings in water, but it is possible to build on top of water manually. To add a body of water, the user simply defines it in the list on the "water" tab (Figure 4), giving dimensions and starting coordinates. In this tab and throughout the program, "starting coordinates" always define the lower left corner of the rectangular footprint of an object (i.e. the lowest value of the object's $x$- and $y$-coordinates).

**Figure 4:** The "water" tab allows the user to define any bodies of water in the city by entering their size and starting coordinates, which are defined as the Cartesian coordinates of the lower left corner of the body of water’s footprint. The optimization program will not allow buildings to be built over the water.

The next tab facilitates the definition of a road grid (Figure 5). Roads are entered rectilinearly, and are given a number of lanes (each lane corresponds to one brick), a length in bricks, a direction ("North-South" means constant $x$-coordinate and "East-West" means constant $y$-coordinate), and starting coordinates. Unfortunately, due to the limitations of this program, the user cannot enter curved roads in this manner. If curved roads are desired, they must be drawn manually later in the design process;
however, it is important to have at least some rectilinear roads in order for the program to function properly. In this early program, there may be glitches if a full grid of roads is not defined. Note that in this early version, there are also glitches if a road is placed within one brick of a body of water. A feature in which the program detects the interference and adds bridges automatically is under development.

![Road Grid Tab](image)

**Figure 5:** The “road grid” tab. From here, the user can define the roads that run through the city by entering for each road a number of lanes, a direction, starting coordinates, and a length in bricks.

The “Main” tab is where the user controls the creation of the city (Figure 6). After filling in the above tables to define layers (resource names), water, and roads, the user can click “Set Up Terrain and Parcels” on the “Main” tab to export the layers, water, and road grid to Rhino.
After the user presses the “Setup Terrain and Parcels” button, a road grid will be displayed both in Rhino (Figure 7) and in Excel (Figure 8). The roads are displayed in Excel in order to define the building drawing system that will be described in the next subsection. Note that objects in Rhino render as solids, not collections of bricks. This simplification is to save processing power, as the model will crash if thousands of bricks are inserted. The Excel program still tracks the bricks and treats the buildings as if they were indeed made of bricks, so the model is still a true Parametric Urban Resource Model. Also note that the program tries to quit Rhino after it stops running. The user simply has to press “Cancel” on the Rhino dialog box to prevent the program from exiting. If the user selects either “Yes” or “No,” Rhino will quit and progress will be lost. There is currently no way to give this software control of an existing Rhino file, so saving and then trying to load the saved file will not work (see Appendix A for technical details on the interaction of Excel and Rhino in this software).
Figure 7: The road grid exported into Rhino. Note the presence of the sidewalks, the water, and the list of color-coded layers to the side.

Figure 8: The road model displayed on the "Main" tab in Excel. Note the addition of automatically-defined addresses and the traditional x-y Cartesian numbering system for the squares. This road grid forms the basis of the drawing program that allows the user to add buildings to the model quickly and easily.
Note that the program has automatically recognized the roads and inserted sidewalks next to them, as sidewalks are a key feature in walkable Compact Urban Cells (this automated feature can malfunction for some road grid configurations, however). In the Excel display, each region of empty space between roads is labeled with an address that is generated automatically based on the names chosen for the roads. These addresses are for the convenience of the designer. Future iterations might do more with the parcels (city blocks) defined by these addresses, i.e. optimizing within a parcel. The Excel display also has a numbering system for the squares in the road grid that corresponds to a traditional x-y Cartesian coordinate system starting at (0, 0). This system effectively reverses the row-column system used in Excel and turns it into the system most designers are accustomed to using in Rhino, preventing confusion.

Building the Initial Parametric Urban Resource Model Using the Drawing Features

Now that the basic layout of the area is defined, it is time to add buildings. The software provides a convenient drawing tool for Excel that allows the user to draw buildings in 2D and then export them to 3D in Rhino. If the user clicks the “Build” button on the “Main” tab (Figure 8), a pop-up dialog (Figure 9) will allow a building to be created. For convenience, the user could also press the keyboard shortcut Ctrl+g to bring up the same dialog. When the user selects a category with the first drop-down menu, the second drop-down menu updates to show only the resources in that category. The user then has the option of entering a name for this building (for reference only). Finally, the user must enter the number of stories, from 1 to 100 inclusive.
When the user clicks "OK," a new dialog box will appear prompting the user to select a footprint for the building (Figure 10). All the user must do to set the footprint is to highlight the squares in the "Main" tab corresponding to the desired footprint. The footprint could also be entered as text, but this method is not as efficient.

Figure 9: The "Build" dialog. Here the user can select what type of resource a new building will be, give the building a name, and enter the number of stories.

Figure 10: The dialog box for selecting the building footprint. The area in the dashed box has been selected for the new building. For visibility, this building is a bit larger than a typical building would be in this type of model.
When the user selects “OK,” the highlighted area will turn the color of the selected resource type to represent the footprint of the new building (Figure 11). A number will appear in the lower left corner, signifying the number of stories. The user should not hit “Cancel” here, as there is no error handling routine yet.

**Figure 11**: A sample building drawn in Excel. This building is a shade of yellow (Professional) and is 20 stories tall, as indicated by the number in the corner of the square.

By clicking the “Export” button on the “Main” tab (Figure 11), the user can export this building to Rhino (Figure 12). Note that the building has rendered as a solid to save processing power. The Excel program still treats it as a collection of bricks, preserving the integrity of the Parametric Urban Resource Model.
Figure 12: The first building exported to Rhino. The building is positioned to correspond exactly with its position in the Excel program. This building was placed in the correct layer to correspond to the resource type (note the check mark by the “Engineering” layer that signifies that “Engineering” is the active layer, as desired). The building renders as one large cube instead of as a series of bricks to save processing power.

Other buildings can be added to the model by a similar process. Buildings can be stacked simply by drawing one on top of another (the first building will be covered in the Excel display but it will still be exported). Note that the user does not have to export after each building is created; all buildings can be exported at once to save time. These steps will be omitted for brevity. There must only be at least one initial residential building to make the statistics and optimization work, but for this example, the base city will be a little more complex (Figure 13 and Figure 14) although it is still not based on a real city.

The existing buildings are exaggerated in size and are placed in an arbitrary distribution for the purpose of demonstration. Only city features added in Excel can be counted by the software. If buildings are added directly in Rhino (i.e. by copying and pasting) the software will not recognize them as part of the model. For user convenience (and for later use during algorithmic optimization), a list of all buildings is generated automatically in the “Buildings” tab (Figure 15).
Figure 13: The base city model in Excel. The buildings shown here are distributed arbitrarily and are exaggerated in size for demonstration purposes.
Figure 14: The base city model exported to Rhino.

Figure 15: The automatically-generated list of buildings in the “Buildings” tab. All the relevant parametric information for each building is shown here for user convenience. The Rhino ID is the tracking number used by Rhino to identify objects in its display; it is automatically generated by Rhino. The “At Max or Min Height?” column tells the user if there is room for the building to be raised or lowered by the optimization algorithm based on the limitations of its resource type.
Statistical Analysis

The Excel program has the ability to count the blocks in the model automatically and display data about the resources used. From the “Main” tab (Figure 13), the user can select the “Statistics” button. This action causes a statistical readout to be created on the “Statistics” tab (Figure 16). The graphs and tables display information on the types and amounts of each resource used in the model, which is crucial for optimization. Once the button has been pressed, it is not necessary to press it again; the graphs and tables will update automatically when changes are made to the model by the optimization algorithm. However, if more buildings are added to the model by hand using the drawing tool, their information will not be added to the statistics. Pressing the “Statistics” button again will cause the statistics function to crash. Future versions of the program will address this issue. For more information on the user interface, statistical display, and drawing tools, please see Appendix A.
Figure 16: The “Statistics” tab. This page contains graphs and tables that give the user data about the Parametric Urban Resource Model. Once created, these visual aids update when changes are made to the model by the optimization algorithm. The table on the left lists all the resources by category and the number of bricks/units of that resource that have been created. The graphs in the middle show all the resources in the map. The top graph includes the color codes for all resources (including shades for each resource within a category) and the middle graph shows the same information broken down by category (color coded only by the first shade in that category). The other graphs on the page (starting with the one in the bottom of the picture and extending downward) show the distribution of resources within the categories, with the resources color coded by shade. The box for total population is important to the optimization process; it estimates the total population based on the number of units of each type of housing resource and the number of people who can live in each unit according to each type’s usage coefficient. The table of “Parents” lists the counts of the bricks made in each category (also known as “parents” or “parent categories”). The buttons pertaining to targets relate to features still under development.
Optimization

This program comes with an algorithmic modeling method that helps the user study population densities in the city. The goal of the algorithm is to create a possible design for an improved city that has a target population density and all the resources necessary to support that density. This algorithm is far from perfect and needs a lot of improvement, but it proves that it is indeed possible to generate improvements to the city algorithmically based on rules (please see the Evaluation section for more details on this algorithm’s strengths and weaknesses).

To start the optimization process, the user must click on the “Optimize” button on the “Statistics” tab (Figure 16). This action brings up a dialog box in which the parameters of the optimization can be set (Figure 17). The most important parameter is the desired population density; the algorithm will prioritize meeting this target and will fill in resources as best it can in order to support this target. The optimization is still not very robust, and will crash if certain use cases are entered, such as a model in which the initial population is zero.
Op~tinvzatKon: Pepatato Dety Mwi~aaa Height Tokyo a Bud Urst Fir pepeIk2 C
Create More Bt&dAhigs Fa Green Spame 6 Prefer Rooftop Gardens Prefer Groaed-Level Green Space
Run Cancel
Figure 17: The “Optimization” dialog box. The population target is set in the upper left corner. Since a common design study is to imagine what an area would look like if it had the population density of another well-known urban area, a drop-down menu is provided with the names of famous cities, listed in order from most dense to least dense. When one of these cities is selected, the population density box below fills in with the appropriate density. This box can be manually overridden by typing in the desired number. The first set of selection circles allows the user to determine how population is added to the city. “Build Up First” causes each building to be increased to its maximum height before any more buildings are made. “Create More Buildings First” fills in all usable land area with new buildings before increasing the height of existing buildings. The “Maximum Height” dialog allows the input of the maximum height to which the algorithm will raise high-rise buildings, up to 100 stories. The second set of selection circles allows the user to have control over how green space is added. “Prefer Rooftop Gardens” adds green space on top of buildings and in empty space on the ground. “Prefer Ground-Level Green Space” prevents rooftop gardens and only fills in green space on the ground, although this feature has glitches. The empty space in the dialog box will be used to add more functionality in the future.

When the user clicks “Run,” the optimization algorithm will start. This process can take a while since it starts with counting all the existing buildings and has to generate a significant number of new buildings to fill in all the available space. The program starts by increasing the population to the target level. For the sake of computation time, the program stops when it gets close to the target, because hitting the target exactly can be extremely difficult. The resources are then filled in based on how much the city needs each resource, which is determined based on the quantity already present and the usage coefficient for that resource. The resources the city needs most are added first. Buildings are inserted as far away from other buildings of that type as possible in order to get an even distribution of resources
throughout the city. The statistics are updated after each building is added to account for the change in resource need and distribution. For more information on this process, please see Appendix B.

The new buildings will appear in both Excel (Figure 18) and in Rhino (Figure 19). This optimization currently does not include a feature for adjusting the heights of added resource buildings, which is why they are all significantly smaller than the residential buildings, which can be raised up to the maximum height specified in the "Optimization" dialog (Figure 17).

Figure 18: The optimized model viewed in Excel.
Note that since the “Prefer Rooftop Gardens” option was selected that most buildings in the Excel display have been covered with rooftop gardens. The program knows automatically to stack these buildings when exporting to Rhino. Rooftop gardens are an important feature for maximizing livability in dense areas, since they provide public access to the outdoors without taking up any additional land space. Only certain types of buildings can support rooftop gardens (this has been hardcoded into the algorithm). Also note that reasonable spacing between buildings is enforced. Buildings can touch but cannot completely surround each other. Buildings are not placed in water or on roads or sidewalks (although future iterations of the software might allow this, i.e. by having the building straddle the obstacle). All of the available land area has been filled; empty space is unusable because of the rules mentioned above.

The list of buildings has also been updated to account for the new buildings (Figure 20). Note that the original buildings are still in the table, but some Rhino IDs may have changed since the program re-
rendered these buildings. From the table, it can be seen that the population was adjusted first and then amenities were added. It is possible that the program ran out of space before some of the amenities reached their optimal levels. This problem will be mitigated when the function to vary the heights of the non-Residential buildings is added.

The statistical output has also been updated (Figure 21). Notice that the “Total Population” is now close to the target. The program does not hit the target exactly because there is some leeway built in to speed up the optimization. In this example, the usage coefficients for Residential buildings have been made an order of magnitude smaller than the coefficients for Professional buildings, which in turn have

![Figure 20: The updated list of buildings in the “Buildings” tab.](image-url)
been made an order of magnitude smaller than the coefficients for all other buildings. It is clear from the statistics and from the models that the buildings have indeed been added to fit these proportions, demonstrating that the optimization does indeed take the coefficients into account.

**Figure 21:** The updated “Statistics” tab. Note that the Total Population of 14308 is close to the target of 14400. In order to save significant computation time, the program does not have to get the target population perfectly. It is clear from the graphs that Residential dominates the city, since this category had the lowest usage coefficients. Professional comes next, since its usage coefficients were an order of magnitude higher (but this resource has an artificially large share because of the enormous Engineering building that was built at the beginning of this demonstration). The other resources (excluding Water, Roads, and Sidewalks, since they were defined at the beginning) have the smallest shares since their usage coefficients were another order of magnitude higher. Note that there are still shortages of some resources. This is because the city has run out of land. In future iterations of the program, the algorithm will be able to adjust the heights of the non-Residential buildings (as it currently does with the Residential buildings) to get a more optimal distribution. Note that it is now easy to see the shades on the category distribution graph at the bottom of the page.
Wrapping Up

Finally, on the "Main" tab (Figure 18) there is a "Clear" button. Pressing this button erases everything except the resource list, the water list, and the roads list. This allows the user to start fresh with the city design. Future iterations will allow the user to perform additional optimizations on the same model; however, the features for reducing population and deleting buildings to make room for new resources are still in the works. The program will crash if all the usable space is filled and it is commanded to add more population that cannot be added by raising building heights. Please see Appendix B for more information on the algorithm used to perform the optimization.
Evaluation

Author Self-Assessment

Compared to designing digital Parametric Urban Resource Models with traditional CAD methods, using this program has several advantages and some disadvantages. Initial user feedback on the program has been positive.

This program makes the modeling process easier in numerous ways. For CAD novices, the interface is simpler and has less of a learning curve. The built-in analytics save time and effort when studying the model. The optimization algorithm mechanizes the tedious exercise of adding resources to the city and does it in a mathematically optimal way given the rules of the program.

Unfortunately, there are still some major issues with this early version of the software. This software is not as flexible as a CAD program; one key restriction is the limitation to rectilinear roads and another is the restriction of building shapes to rectangular prisms. The lack of a “delete” feature is frustrating, and is an important omission. It is still difficult for users to make mixed-use buildings, and the optimization algorithm does not tackle this challenge at all. As mentioned earlier, the optimization algorithm is still missing a lot of key features, such as the ability to delete buildings or raise the heights of added resources; even if those features were added, it would still need many more rules in order to become a useful urban resource optimization algorithm. Complementary resources should be grouped and conflicting resources should be separated. For instance, it makes sense to put restaurants near retail but it does not make sense to put a bar next to a school.
User Assessment

Multiple people have given feedback on this project, and two users have gone through a detailed demonstration/test of the program. One user has significant experience in computational urban modeling and the other does not.

J. Ira Winder, a graduate student working on the CityScope project, reviewed the software and gave positive feedback. He said it would be a useful way to generate ideas about how to design a city. He liked the fact that the buildings are placed in a quasi-deterministic manner, meaning that although there are rules for placing buildings, there is still an element of randomness in the city design. He thought that the incremental placement of buildings during the export and optimization process would provide an interesting visualization of the urban development process. He also believes that it could be useful as a tool to send city data to urban simulation tools, such tools for conducting walkability studies. These studies could either be stand-alone simulations or part of the CityScope platform. He does not believe that this tool (or an improved version of it) could be used as the sole means of designing a city, because he believes in the importance of having some non-deterministic aspects of city development, a view which the author of this thesis shares. Overall, he thought that this was a very useful tool that has a lot of potential as a part of the urban design and simulation processes.

A mechanical engineering graduate student with CAD experience but no experience with Parametric Urban Resource Modeling or Rhino tested the program and was pleased overall. She though the interface was simple and easy to use, but could use aesthetic improvements such as reducing the number of worksheets and cleaning up the buttons. She said that it is much easier to draw blocks in this program then to draw them in a traditional CAD program, but she was disappointed by the lack of
flexibility. She strongly prefers the built-in statistics system over counting blocks by hand. Overall, she was impressed with the program and believes it has great potential given more refinement.

**Inclusion in the Design Process**

Parametric Urban Resource Modeling can become an important step in the urban design process, but it should not become the entire design process. This program can be used as a tool to study cities and get a first-pass idea of how a city will look and what resources need to be added when the population density rises. It can also be used for design studies regarding hypothetical population increases for a city. Of course, it is unwise to think that any program at this stage could generate a final model of a neighborhood. Accordingly, designers should instead take the information gleaned from this tool and apply it to their traditional design processes. Perhaps in the future, Parametric Urban Resource Modeling programs will be able to play a larger role in urban design, but for now, this design tool can only be one step in a multi-faceted urban design process.
Future Work

The next step in the design of this software is to demonstrate it to more users and acquire more feedback. It would be useful to speak with urban planners to ascertain exactly how they could see themselves using Parametric Urban Resource Modeling software in their design processes. From this feedback, the program could be tailored to their needs, and the direction of future development could be set.

Regardless of the user feedback, there are some critical features that must be added. The road grid must be improved; curved roads should be accommodated, and the sidewalk-generation algorithm needs to be smarter and more flexible. The bridge-building algorithm also needs refining. A delete button for the drawing page must be added, and a better way to represent stacked buildings needs to be developed. A tool to create mixed-use buildings should be added, and there should be more flexibility in designing buildings of different shapes.

The optimization algorithm must be improved to deal with complementary and conflicting resources better. Work can also be done to try to optimize the distance between resources in smarter ways, such as by considering walking paths or the relationships between each resident’s work, home, and commonly used resources. The algorithm can also be improved to make better use of space; it could suspend buildings over water, roads, and sidewalks, and it could space buildings in more logical ways. The algorithm should also have the ability to remove or change buildings, change single use buildings to mixed-use buildings, and stack buildings on top of each other. The program could also include ways to insert optimized public transportation or car-sharing stations. The algorithm should be able to reduce the population density as well as raise it, and there should be a way for the user to switch between different density scenarios easily.
To facilitate critical thinking about the design of a city, the algorithm could take the idea of exploring how a city would look if it had the population density of another well-known city one step further and impose the same design rules (or lack thereof) found in that city. For instance, the algorithm could impose the regular building patterns found in Barcelona or the regular street grid and large Central Park of Manhattan. By iterating through various existing design strategies, planners could get a better idea of the feasibility of implementing strategies that have been successful in other cities and could better understand how to avoid implementing strategies that have already failed.

The optimization algorithm must also be made faster. The program is currently very slow and there are many inefficiencies in the code that can be improved. It would also help the user if the base city could be imported from an external file or at least saved so it does not have to render every time. The user interface could be cleaned up and improved, and extraneous buttons, Excel sheets, and lines of code should be removed.

This program or a similar Parametric Urban Resource Modeling program could be integrated with the CityScope platform. In theory, a more developed version could be the back-end of the projection system. The vision is that the user could 3D scan a physical Parametric Urban Resource Model, import the digital model to the program, add buildings using the drawing interface, run optimization, and project the finished model back onto the CityScope system. This is a lofty goal, and is a long time away, if it is even possible.
Conclusion

This thesis has outlined a new program for conducting Parametric Urban Resource Modeling. It attempts to simplify the process and remove a great deal of the learning curve for CAD and programming novices. It aims to be faster and more efficient than traditional CAD tools when making Parametric Urban Resource Models. Its ability to optimize these Parametric Urban Resource Models algorithmically could be hugely beneficial to urban designers. It could save them time in the city design process and produce a higher-quality final design by helping them understand early in the design process what the area requires to be livable. There is much work that can be done to continue developing this program, but the initial results from this early prototype have been promising. It is hoped that this program will soon be powerful enough to be applied to a design study of Kendall Square or another neighborhood.
Appendix A: Technical Description of the User Interface

This program takes advantage of the fact that both Excel and Rhino can be programmed using a version of Visual Basic. Since Rhino already uses a command line, it is fairly straightforward to use the built-in RhinoScript language to write programs for creating 3D models. However, RhinoScript was not directly used in this project; instead, all of the code was written in Visual Basic for Applications (VBA) within Excel. This method of scripting allows Excel to take control of Rhino and issue commands via the Rhino command line and also by calling RhinoScript functions.

The remainder of this appendix will present a high-level overview of the main functions that make the user interface of the Parametric Urban Resource Modeling program work. There is far too much code to show it all here, so the appendix will include sample code that captures important ideas.

Note that some features are incomplete. Code related to incomplete features or features not discussed in the body of the thesis will not be discussed or shown. There is also some information in the worksheets that is deliberately ignored (like the “Targets” buttons and tables) since it relates to incomplete features or functions used in testing the program. These extraneous things do not affect the functionality of the features described in the body of the thesis and should be ignored until the features are completed.

Registries

The registries are worksheets that are used to store data about space usage in the city. The registries tell Excel the location and type of each resource unit, data which is the foundation of Parametric Urban Resource Modeling. Since the Rhino model is laid out in a 3D grid system, the registries track data by
mapping worksheet cells to Rhino grid cells, which in turn correspond to actual locations in the city. The value stored in the worksheet cell corresponds to the information needed to describe the resource available at the corresponding city location. The data must be stored in worksheets and not in a hidden array within the program because the contents of arrays are discarded each time a sub program ends; in order to use the data and update it over the course of multiple sub programs, it must be stored in worksheets. This program has three registries, each contained in a separate worksheet:

*Land Registry:* This worksheet is designed to tell the program what land is usable by tracking terrain features and indicating when a building has been built in a particular space. It maps a 100 x 100 grid of worksheet cells to the 100 x 100 base of the Rhino model. It is also used to track available land in the optimization algorithm. See Figure 24.

*Usage Registry:* This worksheet is a 2D representation of the 3D Rhino grid and is the main place in which parametric data about the city is stored. There are 1,000,000 cells in the Rhino model arranged in a 100 x 100 x 100 grid. This worksheet represents the same 1,000,000 cells with 100 tables that are 100 cells x 100 cells. Each table represents one story of the 3D model. These tables are patterned down the worksheet starting in row 101. For example, the first story of the city is tracked in the table located in rows 101-200 and the second story is tracked in the table in rows 201-300. Rows 1-100 contain another 100 x 100 table that tracks roads and sidewalks. See Figure 28 and Figure 29.

*Names Registry:* This worksheet tracks the names of streets and the addresses of parcels, using the same 100 x 100 grid as the Land Registry. This worksheet is not very important in the current version of the program but may become important if more functionality is added to the address feature in later iterations of the program.
Set-Up Functions

The “Setup Terrain and Parcels” button starts the modeling process by running the associated subprocedure when clicked. First, Excel creates a Rhino object that it can control using ActiveX Automation (Figure 22). It then calls the Rhino command to shade displayed surfaces for easy viewing of the buildings.

![Figure 22: A screenshot of the VBA developer window in Excel showing the code for ActiveX Automation of Rhino in Excel.](image)

The program then reads the resource type information that has been entered into the “layers” tab (Figure 3). Using loops and arrays, Excel gathers the data and then sends it to Rhino by calling the layer definition functions (Figure 23). This creates the layers seen in Figure 7 along with their color codes. The program also takes advantage of Rhino’s ability to group layers under parents to mimic in Rhino the same grouping of resources found on the “layers” tab. The program locks the layers so they cannot be edited by accident.
Next, the program loops through the layers again to create named ranges in Excel that facilitate the "Build" and "Optimize" functions. Named ranges make it easy for Excel to find lists of data programmatically. The program finds the group of layers that correspond to each parent layer (category), and the name of the parent is given to each group. A new list of the parent layers is also created and named. The ranges allow the creation of the lists in the "Build" form dropdown menus. The range names for the population density lists for the "Optimization" form are also defined here (These lists are stored on the "Code" tab that the user does not touch.).
The next step is to define the water. The program uses a similar looping technique as it did for the layers to read the list of water information contained in the "water" tab (Figure 4) and it translates that information to Rhino by using a command to create surfaces. Excel then saves the location of the water in the "Land Registry" tab (Figure 24).

![Figure 24: The "Land Registry" tab showing the entries for the water shown in Figure 4. The rows and columns have been interchanged from the display in Rhino, which makes coding simpler (Excel uses row-column notation while a traditional Cartesian ordered pair is expressed as column-row). The footprints appear distorted because the cells are not square.](image)

After defining the water, the roads are defined. The same ideas apply to the roads as to the water: Excel loops through the data on the "road grid" tab (Figure 5) and translates it to Rhino. The locations of the roads are stored in the Usage Registry and their names are stored in the Names Registry. The code then
makes use of the road information stored in the Usage Registry to add sidewalks that border each road automatically. It does this by looping through all the cells in the area of the Usage Registry where the program knows there could be roads and looking for the value designating roads. When it finds a road, it then finds adjacent empty space by looping though adjacent cells looking for any that have no value and puts sidewalks in these cells. To save rendering power, it tries to add each sidewalk as one surface that spans multiple empty squares. The sidewalk-creation feature currently works well for a regular grid of two-lane roads but may have trouble with other configurations. The code also tries to detect interference between roads/sidewalks and water in order to add bridges automatically, but this feature is still under development and is not demonstrated in this thesis. Some of the code for building things in Rhino has been encapsulated into functions that can be reused easily (described in more detail in the "Exporting Buildings to Rhino" section). This code is too long to show here, but the function of reading data from the registries will be shown later.

The program then translates the data from the registries into the 2D Excel map used for drawing (Figure 25). The idea of looping through cells in the registry, checking their values, and taking action depending on the value is consistent throughout all the functions. There is then code to set up some of the data storage worksheets and to define the parcel names, but this shall be omitted for brevity. Note that the parcel numbers are added to the Usage Registry's first 100 x 100 grid to fill the empty space in the roads/sidewalks table. This is merely for the convenience of not having confusing empty space in the Usage Registry.
For i = 0 To 99
    For j = 0 To 99
        If Worksheets("Land Registry").Cells(i + 1, j + 1).text = "Water" Then
            With Selection.Interior
                .Pattern = xlSolid
                .PatternColorIndex = xlAutomatic
                .ThemeColor = xlThemeColorAccent4
                .TintAndShade = 0.399975585192419
                .PatternTintAndShade = 0
            End With
        End If
        If Worksheets("Usage Registry").Cells(i + 1, j + 1).text = "Roads" Then
            With Selection.Interior
                .Pattern = xlSolid
                .PatternColorIndex = xlAutomatic
                .ThemeColor = xlThemeColorAccent4
                .TintAndShade = 0.349986266670736
                .PatternTintAndShade = 0
            End With
        End If
        If Worksheets("Usage Registry").Cells(i + 1, j + 1).text = "Sidewalks" Then
            With Selection.Interior
                .Pattern = xlSolid
                .PatternColorIndex = xlAutomatic
                .ThemeColor = xlThemeColorAccent4
                .TintAndShade = 0.599993896298105
                .PatternTintAndShade = 0
            End With
        End If
    Next
Next
Cells.Select
Selection.ColumnWidth = 2
Selection.RowHeight = 14

Figure 25: The most important code used to generate the 2D map in Excel. The code to generate the labels has been omitted. The code for setting the colors was created based on code generated using the Excel “Record Macro” function, which generates code from user actions. Unlike the way in which colors are defined in this function, the colors for buildings are assigned later using the corresponding resource type’s (layer’s) RGB color code and the appropriate VBA functions. The hardcoded color values seen here may be replaced with references to the layer’s colors in future iterations of the program.
Once the addresses are defined, the “Set Up Terrain and Parcels” sub program is complete. When a sub program ends, Excel gives up control of Rhino and Rhino attempts to quit. The user must manually cancel this command before modeling can proceed. Each time a new button is pressed, this same code will run, but instead of creating a new copy of Rhino, it will re-establish control over the open copy. This program can only control a version of Rhino that was originally launched by Excel. In future versions, attempts will be made to eliminate the automatic quitting and to allow Excel to take control of an existing copy of Rhino, which will help facilitate the importation of base models.

**Drawing Functions**

Once the 2D map is drawn, buildings can be added by pressing the “Build” button or pressing Ctrl+g on the “Main” tab (Figure 6). The “Build” dialog box (Figure 9) draws the list of resources from the named ranges mentioned in the last section. When a category is selected, the code updates the range used in the second dropdown menu to show only the resources in that category, simplifying the selection process.

When the “OK” button is pressed, the user is prompted to select a footprint on which to build the building. The information about the footprint and the information entered into the “Build” dialog are entered into the “Buildings” tab (Figure 15). The code then loops through all the existing buildings to see if the footprint of the new building overlaps that of an existing building; if so, the elevation is incremented to lift the new building on top of the existing one (Figure 26). Note that elevation is defined as the height in stories of the building footprint off the ground. The elevation information is stored in the “Buildings” tab.
Figure 26: Screenshot of the code for stacking buildings. If a new building footprint is drawn over an existing building footprint, the new building is lifted over all interfering existing footprints by adjusting its elevation. The code finds the tallest building that shares the new building’s footprint and makes the new building’s elevation equal to that building’s height (which is equal to that building’s elevation plus that building’s number of stories).

Note that since the buildings in the “Buildings” tab list are ordered with the last building drawn listed at the bottom, the program will elevate the new building on top of all buildings in a stack by making it sit on top of the highest stacked building that has an overlapping footprint since it will detect the interference of that building’s footprint last. The program also ensures that buildings that straddle multiple other buildings clear the tallest existing building by only updating the elevation if the new building detected in the footprint is taller than the ones already detected (Figure 27).
Figure 27: Examples of stacked buildings. Note that when multiple buildings are stacked, the newest building is elevated to sit on top of the whole stack. Note also that when a new building shares a footprint with multiple existing buildings that that new building is elevated to sit on top of the highest existing building in the footprint.

The code then assigns the layer color and story number to the 2D Excel map. It also reads the building type from the “layers” tab (Figure 3) and determines whether the building built is at its maximum or minimum height. This information is important for the optimization algorithm (Appendix B) and is recorded in the “Buildings” list (Figure 15). Note that it is possible to draw a building that has a height out of the range of its type, but this may cause the optimization to fail. Finally, the code updates the Usage Registry with the information about the newly built building (Figure 28).
Figure 28: Code for updating the Usage Registry with the new building information. Parametric information is stored in the worksheet in a series of 100 x 100 tables arrayed down the worksheet. Each table contains cells corresponding to the grid squares in one story, with the first table containing the road/sidewalk data.

The cells corresponding to the 3D Rhino grid cells filled by this building are filled in with the name of the resource (Figure 29). This creates a parametric record of the new building, even though the buildings are exported to Rhino in a non-parametric block form to save processing power. The code also records in the “Buildings” list the number of blocks that the building contains, which makes statistical analysis simpler, since it eliminates the need to count all the entries in the Usage Registry each time the number of blocks must be counted.
Figure 29: A screenshot of the Usage Registry showing the record of the stacked buildings shown in Figure 27. Note that the first 100 rows contain the road grid data, the next 100 rows contain the data for the first story, the next 100 rows contain the data for the second story, etc. As in the Land Registry, the rows and columns have been interchanged from the display in Rhino, and the footprints appear distorted because the cells are not square.
Exporting Buildings to Rhino

When the “Export” button is pressed, Excel takes control of Rhino using the same code as it did in the “Set Up Terrain and Parcels” sub program (Figure 22). It then loops through all the buildings in the “Buildings” tab (Figure 15) and creates models of the buildings in Rhino. This code will not be shown here because it contains code related to sorting the buildings by parcel, which is a function that is not yet complete. The code takes advantage of two functions that encapsulate the building code, the code for one of which is partially shown below (Figure 30).

![Figure 30: Screenshot of most of the code for one of the building functions. This function is also used when setting up the terrain. Note that this function also updates the Usage Registry with the information for the new building. This function returns an identifying string for the object created in Rhino. This string lets Excel control that object in the future.](image)

Note that in order to avoid confusion if buildings are changed (which for now is only relevant when the optimization algorithm is run since the manual edit functionality is not yet included in this program), the code deletes all buildings in Rhino and clears their entries from the Usage Registry. It then re-exports these buildings with the new buildings.

59
When an object is created in Rhino, it is given an identifying string by Rhino (which will be a called Rhino ID hereafter). The Rhino ID allows Rhino to control each object programmatically. When the buildings are exported, Excel records the Rhino ID of each building that is created in the appropriate cell in the “Buildings” tab (Figure 15). When Excel needs to control that building’s model again (such as to delete it), it can issue the appropriate command to Rhino and reference the stored Rhino ID to identify the Rhino object to be manipulated.

**Statistics**

When the “Statistics” button is pressed after the base city model is completed (Figure 13), the program automatically counts the existing resource units and generates the output tables and graphs on the “Statistics” tab (Figure 16). After performing some basic setup of the worksheet, the program loops through all the layers (Figure 3) and copies their names into the “Statistics” tab. For the purpose of quickly counting blocks, it is much faster and simpler to count the blocks recorded on the “Buildings” worksheet (Figure 15) than by counting all the entries in the Usage Registry. Instead of just counting the number of blocks used in each building of that category type and placing the sum in the corresponding cell on the “Statistics” tab, the code creates a formulaic sum in the corresponding cell that references the cell containing the number of blocks for each building of that category. This makes the statistics update automatically when a building size is changed during the optimization process. The code then adjusts the formulas to take into account the roads and other resources made during the initial setup. Since these setup resources do not change during the optimization process (at least in this early version of the program), their counts do not need to update dynamically. The program then performs similar operations to display the counts by resource category.
The program then creates the charts and color codes them with the colors assigned to each layer. The color for a parent layer is the same as the color assigned to the first resource in the category. Since creating charts is a common VBA exercise, the code will not be discussed further here. There is also code for setting targets for the optimization, but since this feature is not complete, the code will not be discussed here.

Finally, the code creates a dynamically-updating total population value, which is very important in the optimization algorithm. The program inserts a formula into the appropriate cell that sums the products of each Residential resource count times the usage coefficient for that resource (which is the number of people who can live in each unit of that residence type). Since the resource counts are included in the sum via cell reference (and the counts themselves are created via cell reference to the "Buildings" list), the total population will update automatically when buildings sizes are changed. This facilitates easy population accounting when the optimization algorithm is running. Without the dynamic updating, the program would constantly have to count buildings and update the statistics while adjusting the heights of buildings, which would take much more computational power.
Appendix B: Technical Description of the Optimization Algorithms

This thesis has presented a basic version of an optimization algorithm for improving the resource distribution of a city. The following appendix gives an overview of the rules used to optimize the resource distribution and the code written to implement those rules. Since many of the optimization functions are long and complex, code will not be included here.

Optimization Form

The “Optimization” dialog (Figure 17) gives the user four options (more will be developed in future versions of the program). The first option allows the user to select the desired population density by picking from a list of sample cities that have known densities. The lists of cities and densities are hardcoded into the “Code” worksheet and the ranges for these lists are given names as described in Appendix A. Please see Figure 17 for descriptions of the other options.

Population Adjustment

Since this program was intended for studies on raising urban density, the algorithm prioritizes building enough residences to support the target population density over building any other resource.

First, the program adjusts the heights of existing buildings (if needed) to get as close to the target population as possible. A function written to encapsulate some of the optimization steps called “changeHeightsByCategory” performs this operation. This function takes as inputs a resource category (which in this case is Residential), the current total population, the desired target population and the user-set maximum height. The function compares the current population density to the desired
population density. If the two do not match, the function increments or decrements building heights as needed. The function stops when the target has been reached or all the buildings can no longer be changed. Note that this function is written in such a manner that it could be generalized to other resource categories by replacing the total population and target population parameters by the current amount and desired amount of any resource.

The function does not try to hit the target population density exactly; instead, it tries to set the population density within plus or minus 100 people per square kilometer of the target. The flexibility is critical in making the computation end in a reasonable time, since the adjustments do not increment the population density by single units. For instance, raising a building by one story may add 10 people to the city. Thus, incrementing and decrementing stories might just cause the total population to oscillate around the target indefinitely if there is not flexibility in deciding when to terminate the program.

The function keeps track of which buildings can still be changed. Each resource has a user-defined building type that is input on the “layers” tab (Figure 3). The height ranges for these buildings are hardcoded into the height changing functions (but in future iterations the ranges could be made user-defined parameters). The only user-defined height in these ranges is the maximum height for high rises (skyscrapers), which is drawn from the “Max Height” input on the “Optimization” dialog (Figure 17). The program notes on the “Buildings” list (Figure 15) if each building is at its maximum or minimum height and these notations tell the program if the building height can be adjusted in a particular direction. Thus, if the target population has not yet been met but the buildings can be adjusted no further based on these height ranges then the height-changing function will stop.
Once the “changeHeightsByCategory” function has finished running, the program checks the population and starts creating new buildings. The assumption is that for any reasonable population density there will be no reason to remove buildings to meet the target, but building removal functionality might be added in future versions. The program randomly selects a Residential building type, size (within hardcoded limits), and location and attempts to build the building. The program checks the Usage Registry to see if the selected space already contains resources. If there is nothing in the space, then the building is built, and if there is something there, then the program tries again with another randomly-generated building and location.

The code uses a helper function to encapsulate the building code. This function uses similar logic as the drawing functions to create and document the new building. Another helper function updates the formulas in the “Statistics” tab (Figure 16) to account for the new building. The logic of this function is similar to that of the code that created the formulas in the setup of the “Statistics” tab.

If the “Create More Buildings First” option in the “Optimization” dialog (Figure 17) had been selected, then the program continues to create new residential buildings until the target population density has been met or all the available land is used. If the “Build Up First” option had been selected, then the program will re-run the “changeHeightsByCategory” function for the Residential category after each building is created, ensuring that all buildings are at their maximum height before any new buildings are added. When all the land has been filled or the target population density has been reached, the “changeHeightsByCategory” function is run once more for the Residential category to make sure that all Residential buildings are at their optimal heights. The program will then continue regardless of whether the target population density has been met. If the final model does not meet the target population density, then the user must change some of the parameters of the optimization and start over.
Note that all of the above building creation and manipulation is done in Excel. The Rhino model has not been changed at this point. All buildings will be exported at once after the optimization is complete.

**Addition of Amenities**

Once the population density target has been reached, the program fills in remaining space with other resources (which will be called “amenities” to distinguish them from Residential resources). This functionality is encapsulated in a sub program called “FillInAmenities.”

The sub program uses the Land Registry to keep track of which land is open for new buildings. As in the Usage Registry, any cell that does not contain an entry is considered open; however, the Land Registry is not concerned about the type of resources that occupy the land, so used space (except for water) is simply denoted as “blocked.” Water has already been listed in this registry, but roads and sidewalks have not, so the program finds the roads and sidewalks in the Usage Registry and denotes the corresponding cells in the Land Registry as “blocked.” Next, the program blocks out the footprints of existing buildings and the adjacent spaces to prevent new buildings from interfering with or being too close to existing buildings (however, this code still allows buildings to touch each other in some cases, which is a glitch that actually enhances the design process by making the building distribution more realistic). Note that for computational speed the program actually uses an array that tracks the blocked squares instead of actually using the Land Registry; the Land Registry is for user convenience and ease of explaining the concept. The Land Registry is not updated when new buildings are added by the algorithm (only the array is updated), but future versions of this program might continue to update it.

The program inserts amenities one building at a time. It prioritizes the resources that are needed the most; to determine this, the program uses a score for each resource called “peopleUnits.” This score is
calculated by multiplying the number of units that exist for the resource by the usage coefficient for that resource. This number effectively tells how many people the existing units of the resource can serve. This score is then subtracted from the total population to determine how many people are without this resource. The resource that the most people lack is considered the most needed resource and is built first. In the event of a tie, the resource listed first on the "layers" tab (Figure 3) is built first. If a resource has a usage coefficient of "n/a" then it is ignored; it must be added to the city manually. Residential resources are ignored here as well because the population density has already been optimized.

A similar scoring system is used for picking the location of each building. Each unblocked ground-level square is assigned a score for each resource based on the average distance of that square from all existing units of that resource. Distances are measured rectilinearly along the ground only (elevation is not considered). The square with the highest score for the chosen resource is picked for the building. In the event of a tie, the location is picked at random. Note that if there are no existing buildings of a particular resource then all open squares have a score of zero for that resource so the building is placed at random. The building size is then picked at random (within hardcoded limits), and the program sees if the building will fit in the selected area. If it does not, the selection of location and size continues until the building fits. The building is built and the statistics are updated with the same helper functions that have been used previously.

Once a building is built, the scores are updated so the optimization can continue. Counting the resources is very time consuming. Establishing the initial scores takes a long time, so only the scores for the resources affected by the new construction are updated. The program repeats until the city runs
out of land. In this early program, amenity heights cannot be adjusted, but this feature will be added in future iterations.

This sub program still needs improvement. In future iterations, this sub program will be turned into a function that can take the maximum height as an input and it will adjust the heights of the amenities. The program will also crash if all resource targets have been met before all the land is used, because the program only stops when all the land has been filled. Additionally, some resources are not handled properly at the moment. For instance, nothing prevents the program from placing a Rooftop Garden on the ground, which does not make sense. There is also no way to optimize the distribution of the various types of Residential resources or their locations.

**Green Space and Stacking**

Once all the residences and amenities have been added, the program fills in green space. Although green space is included as an amenity, it is also given its own function because it is considered very important to urban quality of life and because it can be placed on top of other resources easily. Regardless of the usage coefficients assigned to the different types of green space (and regardless of whether the target minimum amounts of each type of green space was met by the “FillInAmenities” function), the program fills in all remaining large areas of open land with green space (but in this version of the program, the “FillInAmenities” function cannot leave open space, so this feature will only be useful in future iterations). It also puts Rooftop Gardens on top of all buildings of appropriate resource types (the list of which has been hardcoded). The usage coefficients for the types of green space are ignored because there can never be too much green space in a city, and it is better to fill available space with nice public areas than to leave it empty.
The second-to-last function fixes any interference issues between buildings by stacking things that would otherwise interfere. The same logic is used as in the code for stacking buildings in the drawing function. The last function called is the “Export” function, which is the same function that is called when the “Export” button (Figure 11) is pressed. This causes all the new and changed buildings to appear in the Rhino model. The new buildings are exported in the order in which they were added, so watching the export process could provide useful information about the city’s need for resources.

**Observations on the Optimization Algorithm**

It is interesting to observe that this urban resource optimization algorithm is quasi-deterministic. Although there are rules that stipulate how resources should be placed, there is an element of randomness involved in the optimization that will produce different results each time the program is run with the same input (Figure 31 and Figure 32). Randomness is included because there is often no single correct answer to an urban design challenge. Using this tool, the designer could run the same study multiple times and compare a diverse array of possible design options in order to gain more information about how to proceed with the planning process.
Figure 31: An arbitrary input to the optimization algorithm that will be run three times to demonstrate the quasi-deterministic nature of this optimization algorithm.
Figure 32: The models generated by the optimization algorithm for the inputs given in Figure 31. Note that while all three models are different, they all follow similar trends.
Works Cited


