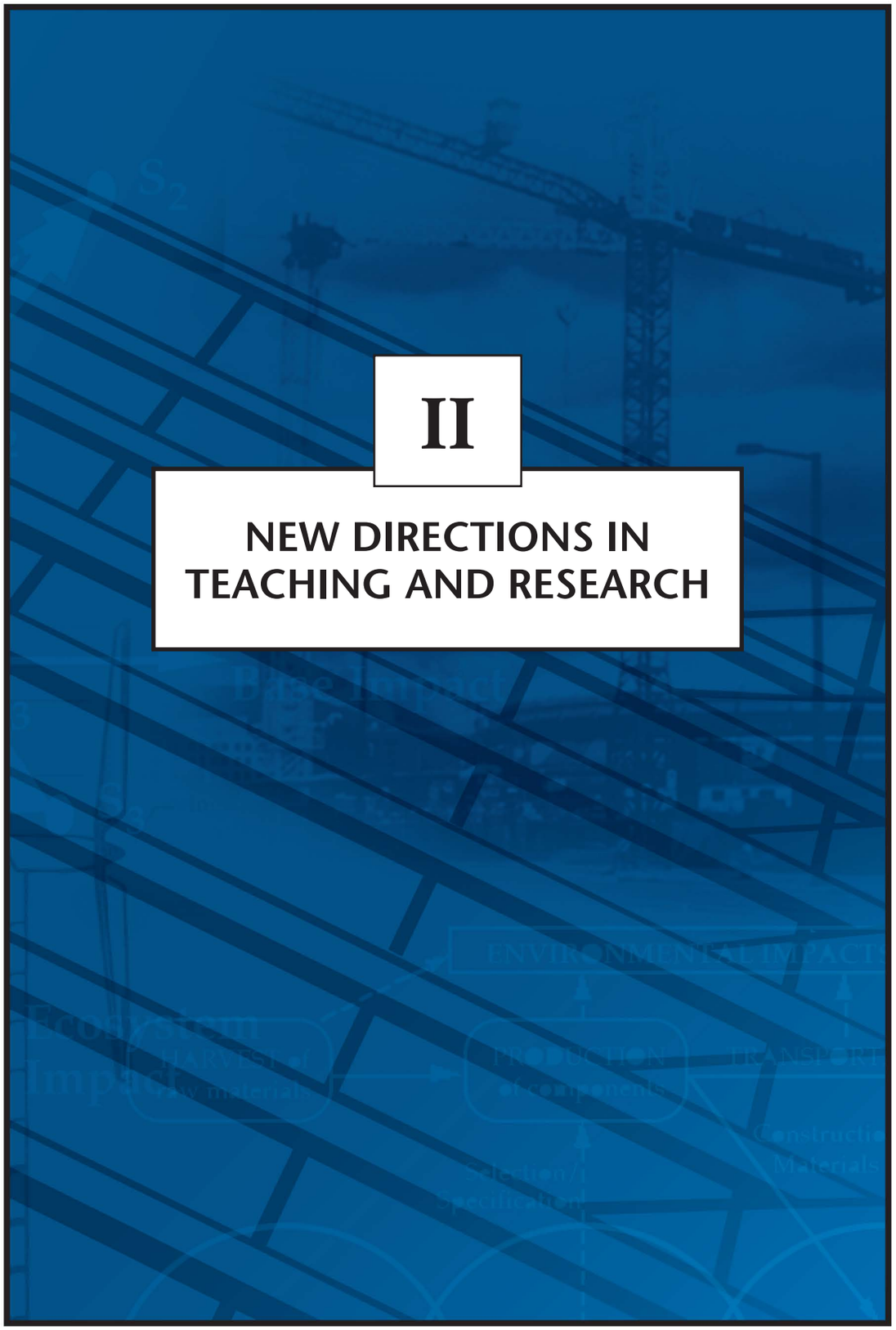


II

NEW DIRECTIONS IN TEACHING AND RESEARCH

TEACHING VISUAL SCRIPTING IN BIM: A CASE STUDY USING A PANEL CONTROLLED BY SOLAR ANGLES

Karen M. Kensek,[1] LEED AP BD+C

ABSTRACT

Programming and scripting can be used to activate a 3D parametric model to create a more intelligent and flexible building information model. There has been a trend in the building industry towards the use of visual scripting that allow users to create customized, flexible, and powerful programs without having to first learn how to write traditional code. Using visual scripting, users graphically interact with program elements instead of typing lines of text-based code. Nodes are created and virtually wired together; they can be numbers, sliders for adjusting values, operators and functions, list manipulation tools, graphic creators, and other types. Text based coding programs such as Python can also be used for the creation of custom nodes when greater flexibility is desired.

Examples from professional firms include scripts that help automate work in the office to increase efficiency and accuracy (e.g. create escape routes, renumber rooms by levels, create documentation), assist in form generation (e.g. parametric design of metal panels, rebar generation, coordination between Revit and Rhino), analyze BIM files (e.g. terminal airflow, visual loads and capacity), and provide analysis results (e.g. daylighting, thermal comfort, window optimization).

One can learn the basic steps of learning a visual programming language through the use of Dynamo within Autodesk Revit. The example used is for a façade component that changes based on the sun's altitude.

KEYWORDS

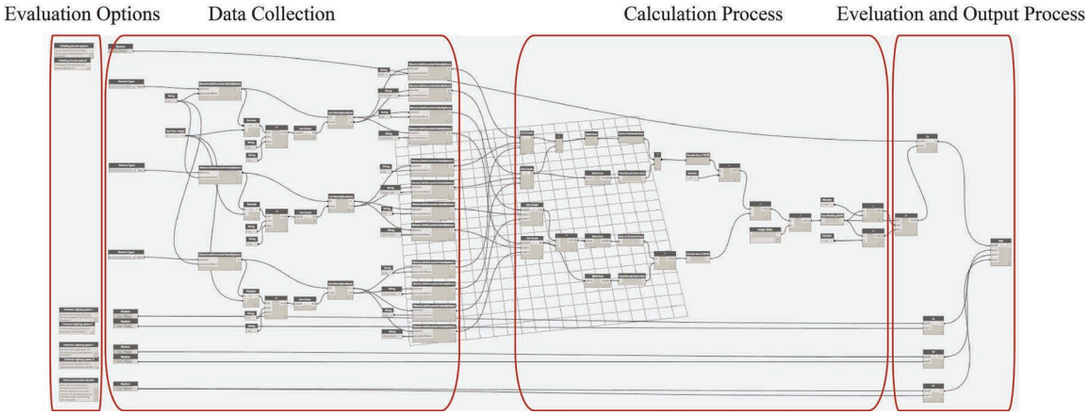
BIM, building information modeling, visual programming, scripting, Revit, Dynamo, teaching

1. INTRODUCTION

Visual programming is an alternative to text based programming languages such as Python and C#. Using nodes that can be as simple as a dialogue box with a number in it, programs are wired together. Grasshopper (developed by David Rutten at Robert McNeel & Associates—http://www.grasshopper3d.com/) for Rhino 3D has made tremendous progress and inroads in the last ten years as a platform of choice for creating customizable scripts for the building industry. It has sprouted numerous (hundreds) of components (http://www.food4rhino.com/) that work

1. University of Southern California School of Architecture, Watt Hall #204, Los Angeles, CA 90089-029, kensek@usc.edu

FIGURE 1. Dynamo script for determining if a building complies with a LEED pilot point for avoiding bird collisions; note that the script is composed of nodes and wires (courtesy of Ding).



with it, including simulation tools such as Ladybug for analyzing weather data; Honeybee for connecting to EnergyPlus, Radiance, and Daysim; DIVA-for-Rhino for daylighting and energy modeling; and Octopus for evolutionary based optimization.

Dynamo for Autodesk Revit (and Dynamo Studio) is another visual scripting environment (dynamobim.org) that is heavily supported by users who create custom packages (https://dynamopackages.com/) such as Ladybug and Honeybee mentioned previously; TASmanianDevil to perform thermal and daylight simulation; Energy Analysis for Dynamo to connect Dynamo

FIGURE 2. Explanation of a Dynamo script for a kinetic facade (courtesy of Yan and Lin).

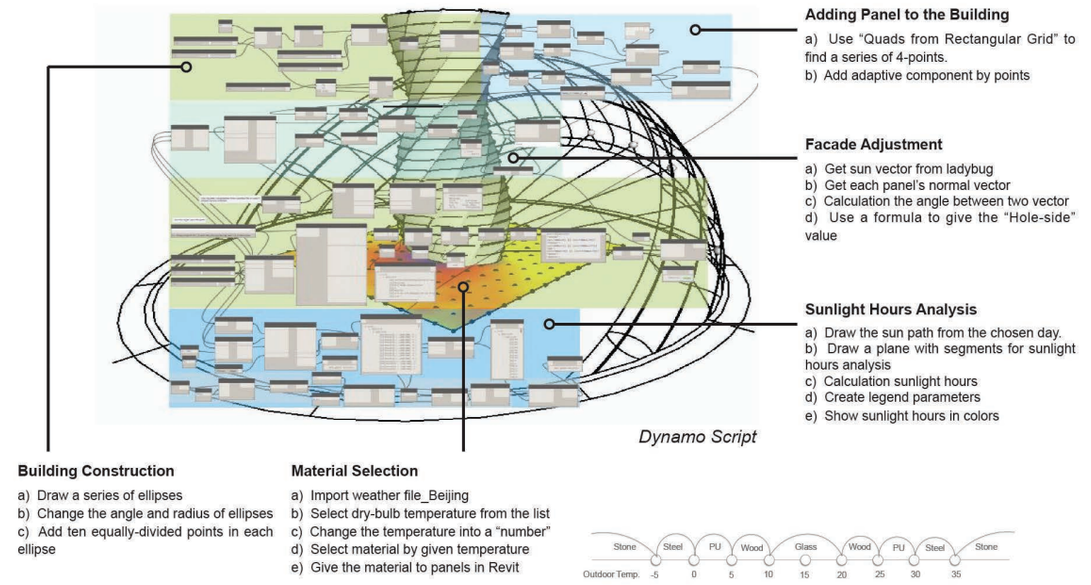


FIGURE 3. Dynamo used with Project Fractal to study glare off a facade (courtesy of Chen, Luo, and Yuan).

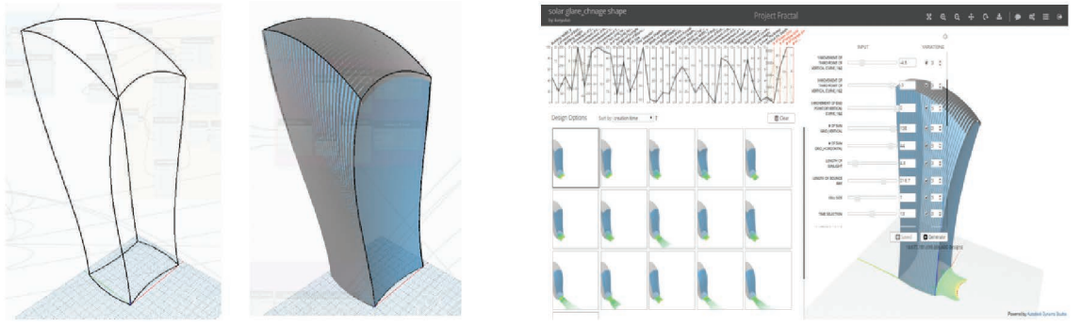
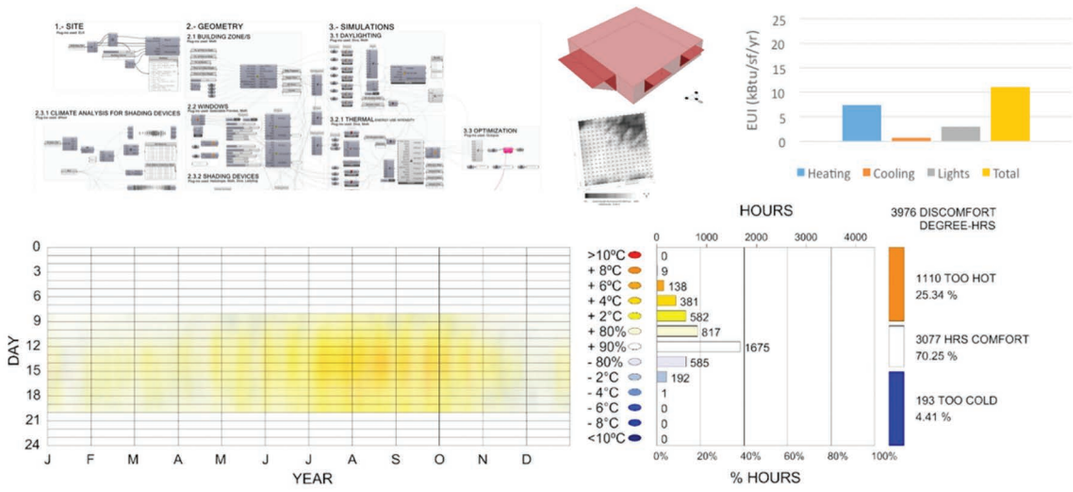


FIGURE 4. Grasshopper script to optimize EUI and thermal autonomy using Rhino 3D, Grasshopper, DIVA, Dhour, Heliotrope, Ladybug, ArchSim, and others (courtesy of Gamas).



to Green Building Studio; and Optimo, a multi-objective optimization tool. Both Grasshopper and Dynamo have many, many more customizations beyond the scope of sustainability.

2. SOLAR ACTIVATED FACADE COMPONENT

There are many websites that have tutorials for learning these tools. Presented here is a brief introduction of Dynamo explaining the key concepts in the context of a solar activated kinetic façade component. The reader should know some Revit, including family parameters, conceptual mass, adaptive pattern based components, and a minimal amount of Dynamo (how to start the program, navigate the software, and select nodes). Other introductory tutorials can be accessed within the Dynamo opening screen. The intent of this exercise is to explain how a

TABLE 1. Example of a cross product.

first list	second list	cross product		
0	3	0–3	1–3	2–3
1	4	0–4	1–4	2–4
2	5	0–5	1–5	2–5

visual scripting program works with a building information modeling software program (Revit). Note that different versions of the software might need different inputs.

The process also shows how to develop a program from a less complex routine to a more sophisticated solution. This is a common teaching technique where not only does the student have to understand the relationship between a simple example and a more complex project, but that the complex project can be broken into a series of small steps. The goal of this exercise is to have a panel in Revit change based on solar angles.

There are four major steps for this exercise: creating a grid of circles with an attractor point; making a grid of boxes with a height parameter; having the grid of boxes respond to the sun's position; and finally using a solar angle to change the size of an opening in a façade component.

2.1 Grid of circles with an attractor point

Conceptually, a grid of circles changing their radii based on the location of an attractor point is similar to having a façade component respond to the position of the sun. It is also a good exercise for learning the basics of Dynamo.

2.1.1 Make a grid of points

Eight nodes are needed (some duplicates): Number (3), Number Slider (2), Range (1), Point.ByCoordinates (1), and Circle.ByCenterPointRadius (1). See the diagram of how to wire them together (Figure 5). Change the values and watch how the number and direction of points change. Note that it will not yet create a grid.

Change the lacing to cross product. As there is no grid command in the basic version of Dynamo (one can download a user-developed "package" with additional commands that will do this), a grid is created by using two lists of numbers that are "cross-producted" to create a grid. A cross product of two lists uses all the combinations in both lists (Table 1).

To change the "lacing" to "cross product," right click on the lower right corner of the Point.ByCoordinates node.

2.1.2 Make a grid of circles

For each grid point, a circle is placed by its center point. Later you will want to change the circle's radii; for now, the radius is made smaller by dividing the number by two. There are other nodes for "normalizing" values, but for these exercises, dividing a list of numbers that are too large is an easy way to scale the graphics.

2.1.3 Create an attractor point

A 2D point is created ("attractor point") that can move by changing the numbers on the sliders. In the next step, the distance from this point to each of the circles' center points will change the radius of each of the circles.

FIGURE 5. Right click on the xxx on the bottom corner of the Point.ByCoordinates node for many features include removing nodes and changing the lacing option.

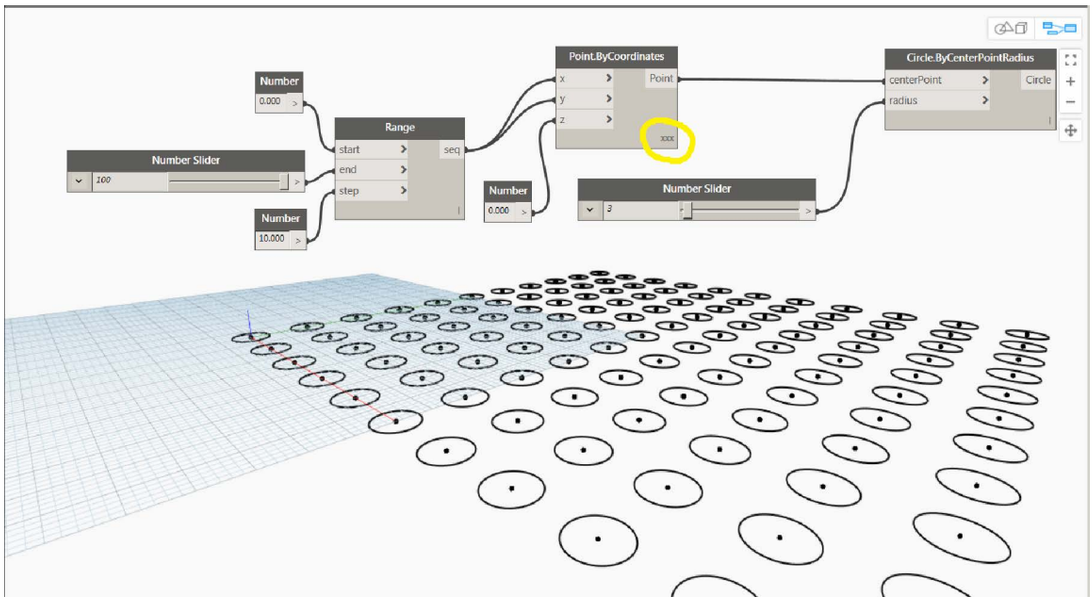


FIGURE 6. Try dividing the radius by different values.

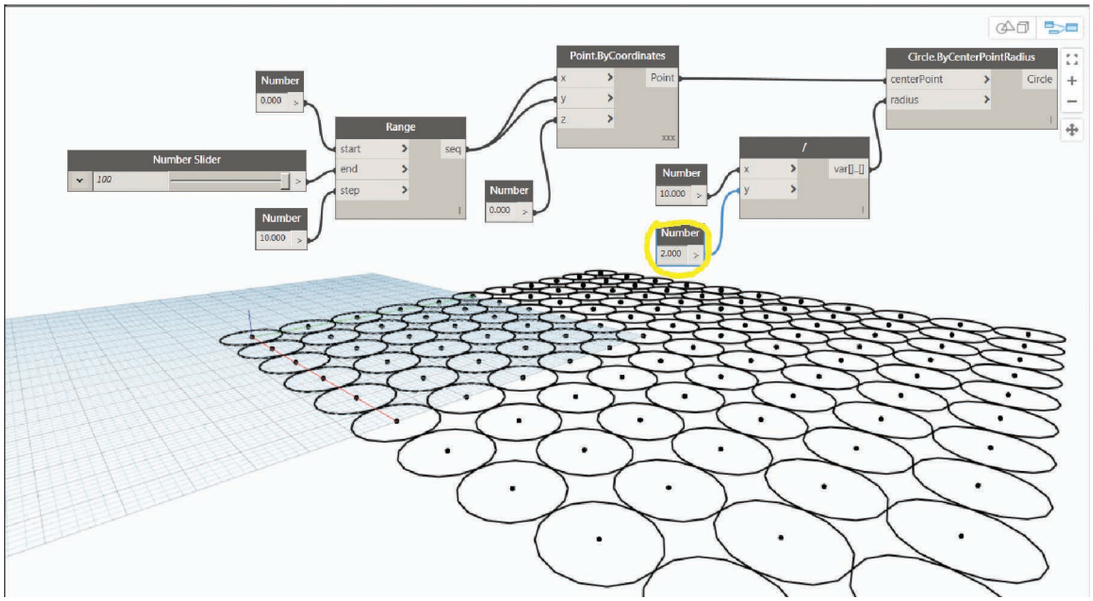


FIGURE 7. View of completed script in Dynamo. Make sure that the lower left corner is set to automatic re-calculation. When the script is more complex, it should be set to manual so that changes are only updated when the user hits Run.

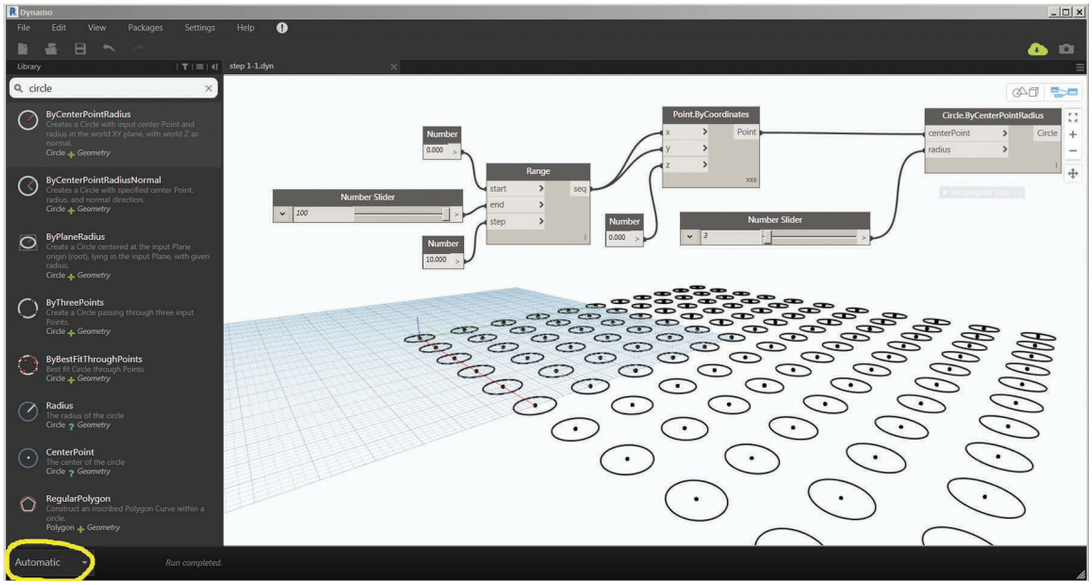


FIGURE 8. This is called the "attractor point."

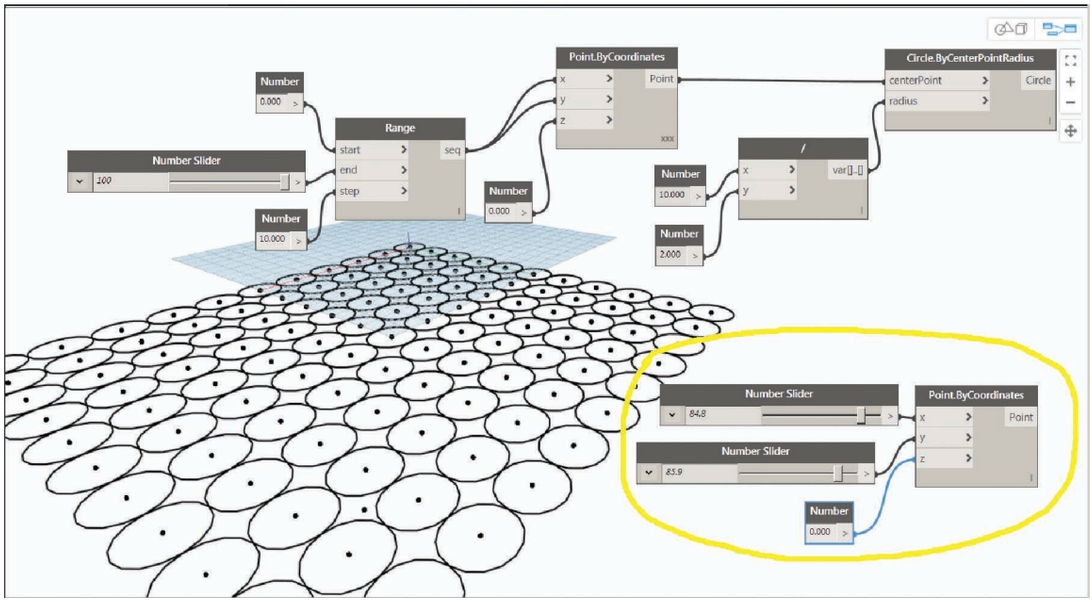
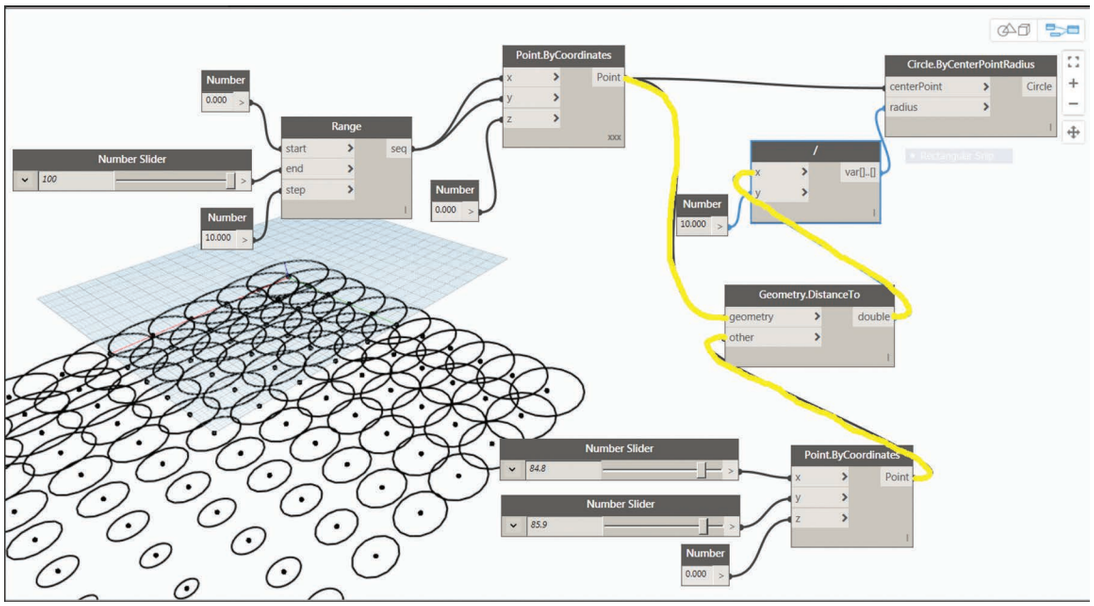


FIGURE 9. Originally, the distance between the center point of the circle and attractor point became the radius of the circle. But because the circles turned out too big, the radius was divided by 10.



Through the use of two number sliders, the user will be easily able to move the point in the XY directions, but not Z.

2.1.4 Have the attractor point changes the circles' radii

A relationship is created between the distance from each circle's center point to the circle's radius. As the attractor point moves, the radius of each of the circles changes.

2.2 Grid of boxes with height instance parameter

A similar exercise to step 2.1.4 (change the radii of circles based on the distance to the attractor point) is done with a grid of boxes. The major difference with this step is that the user will create a Revit box with an instance height parameter first. This will be used instead of a circle. The new important concepts are how to select Revit objects in Dynamo and change their parameter values. These commands will be useful later when the façade component is created.

2.2.1 Create a grid and attractor point

This section is the same as 2.1.1 for creating a grid of points and an attractor point.

2.2.2 Select the Revit family type

The Revit family is selected from inside of Dynamo. As with the circles, a box is placed on each of the points. Note that the boxes are currently now in Revit, not Dynamo.

2.2.3 Have the attractor point change the height parameter

The height parameter of the box is given a value of 12 to check that it is working properly.

FIGURE 10. The box was made with the generic model family command. It was given an instance parameter called "height." The box was loaded into a Revit project file and Dynamo launched; in this way, the Dynamo script is linked to a specific Revit file.

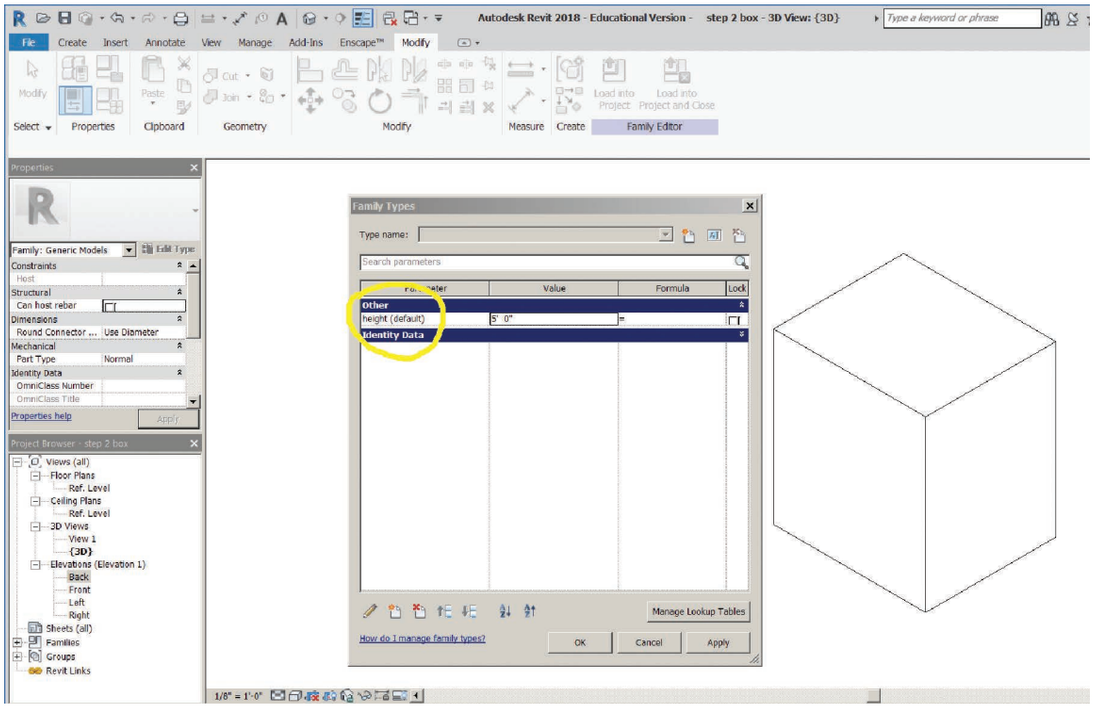


FIGURE 11. The image on the left is the Dynamo script; the window on the right is the Revit project file.

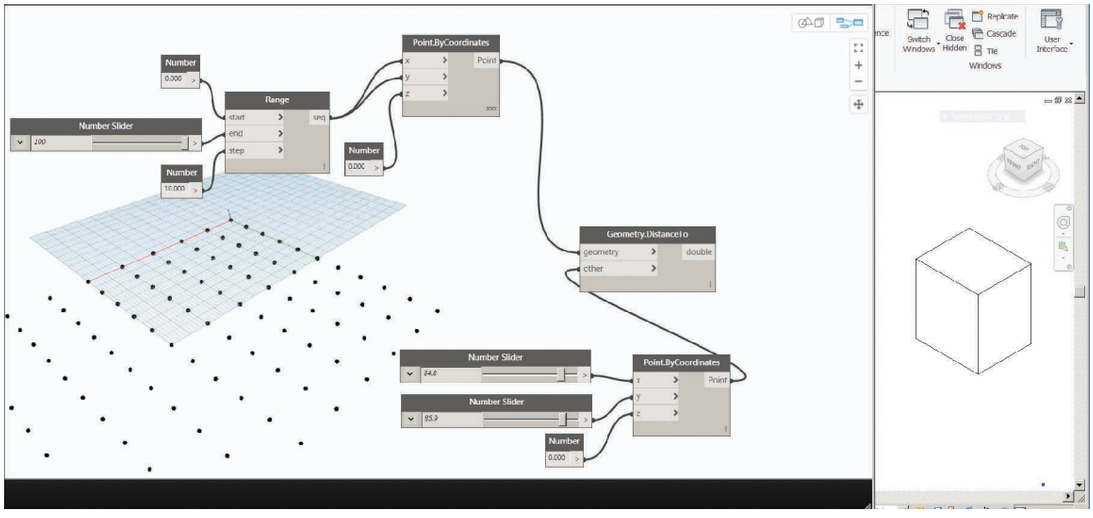


FIGURE 12. The Family Types node lists all the families in the active Revit file. FamilyInstance.ByPoint places an instance of the chosen family centered at each point.

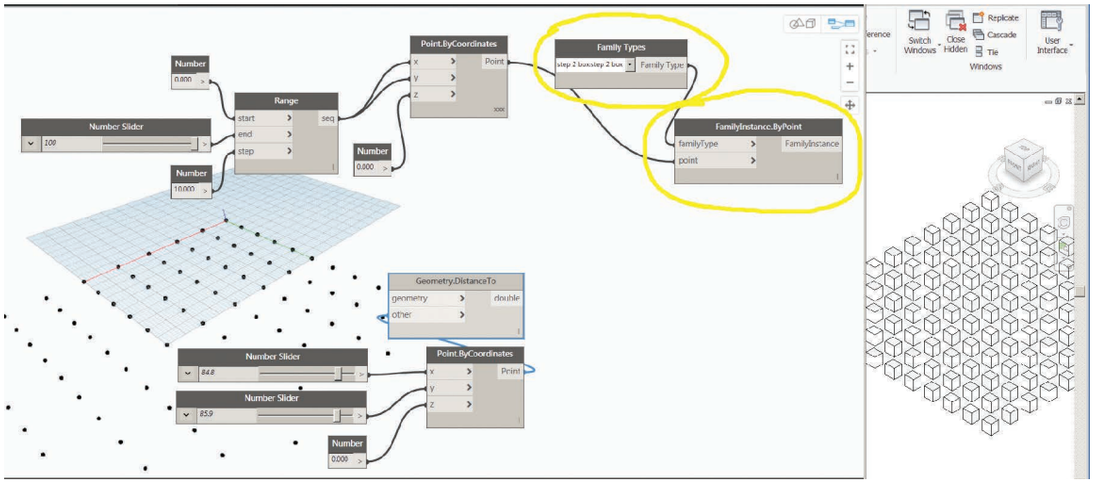
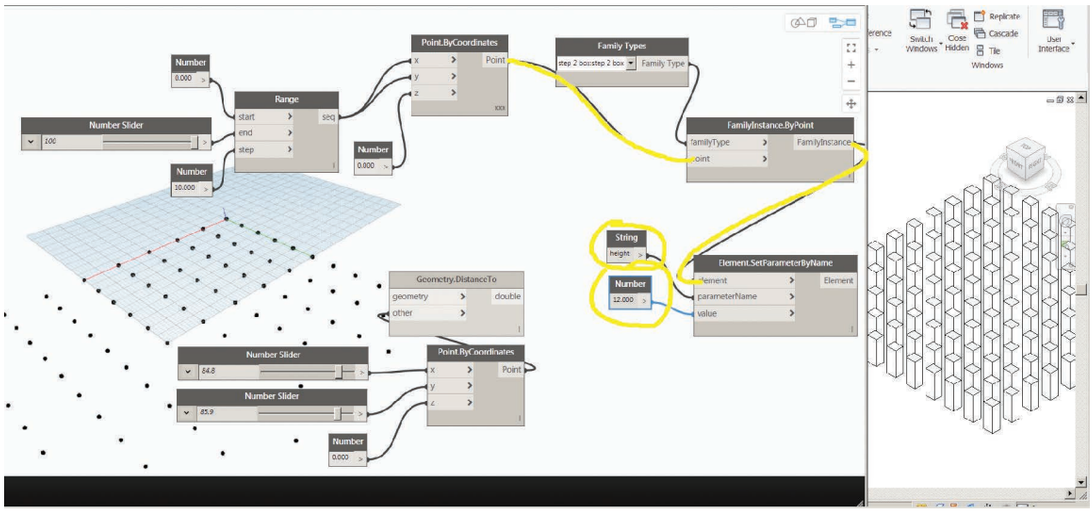


FIGURE 13. The parameter name must be spelled exactly the same way it is in Revit (height). Do not hit a carriage return after height.



2.2.4 Activate the attractor point

Connect the attractor point and distance calculation into the script. As the attractor point moves, it changes the height of the box using the Element.SetParameterByName node. Note that the distance values were divided by four to make the boxes shorter.

2.3 Grid of boxes responding to the sun's position

Instead of using an attractor point, the boxes will change their height based on the sun's altitude. This exercise is setting the stage for the last one where an adaptive component on the façade of a conceptual mass will change based on solar angles.

FIGURE 14. Similar to the circle attractor point, this attractor's points distance to the grid of points will set the height parameter of the boxes.

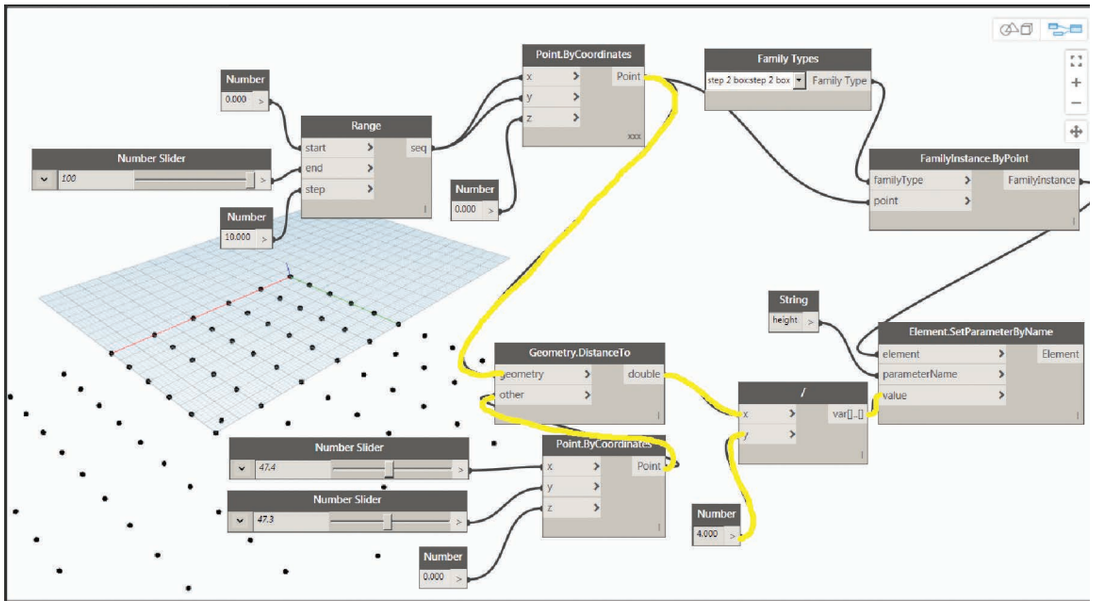


FIGURE 15. The grid of boxes with different heights will be visible in the Revit project file window.

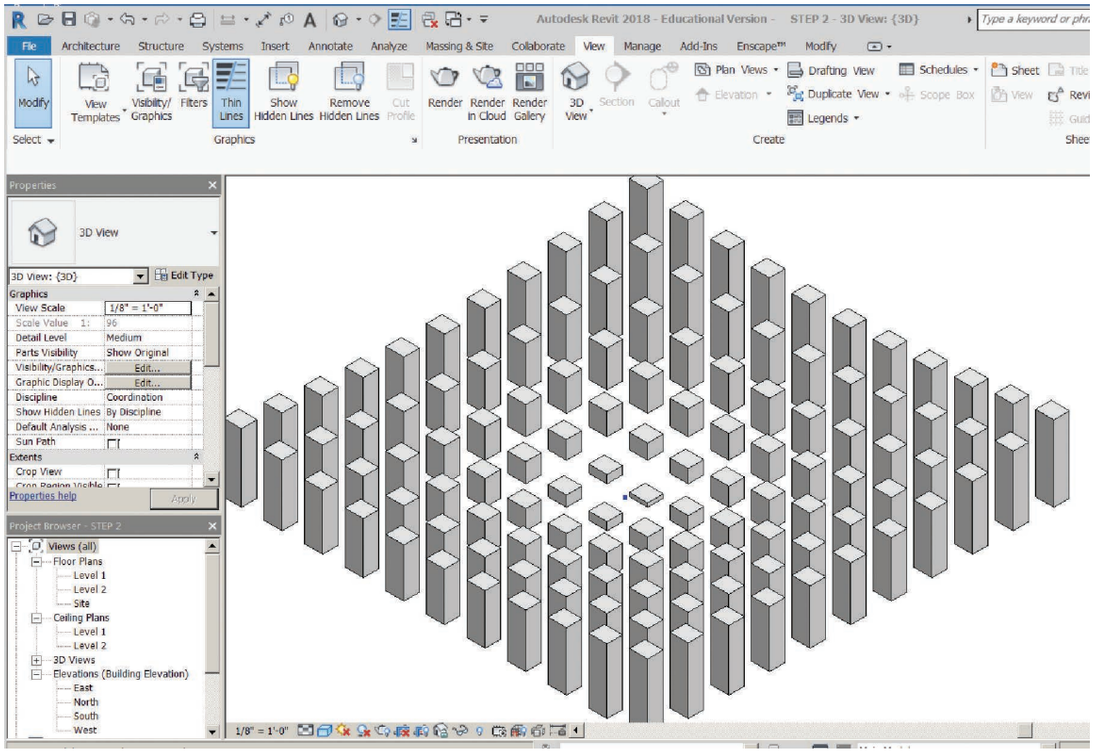


FIGURE 16. As the attractor point changes position in Dynamo, the heights of the boxes change.

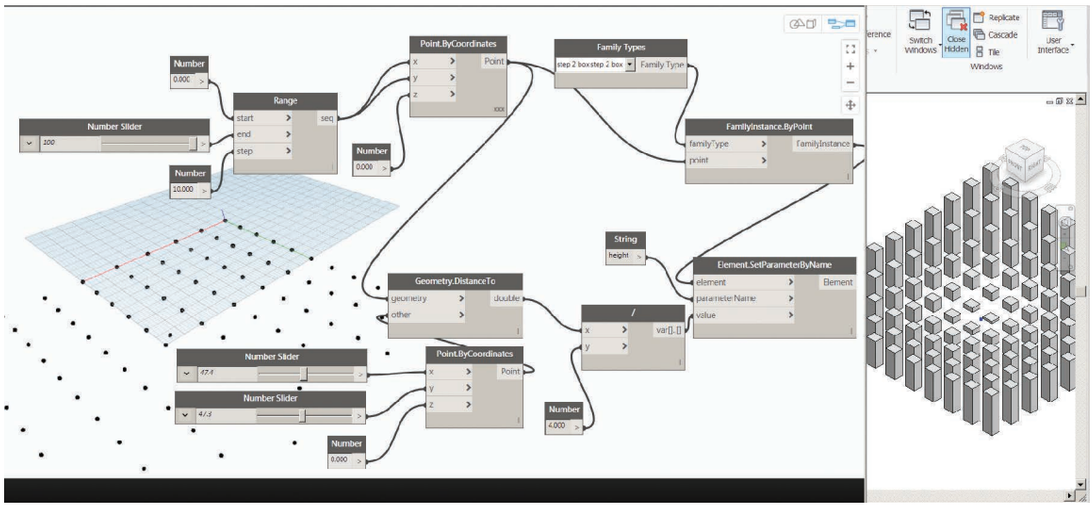
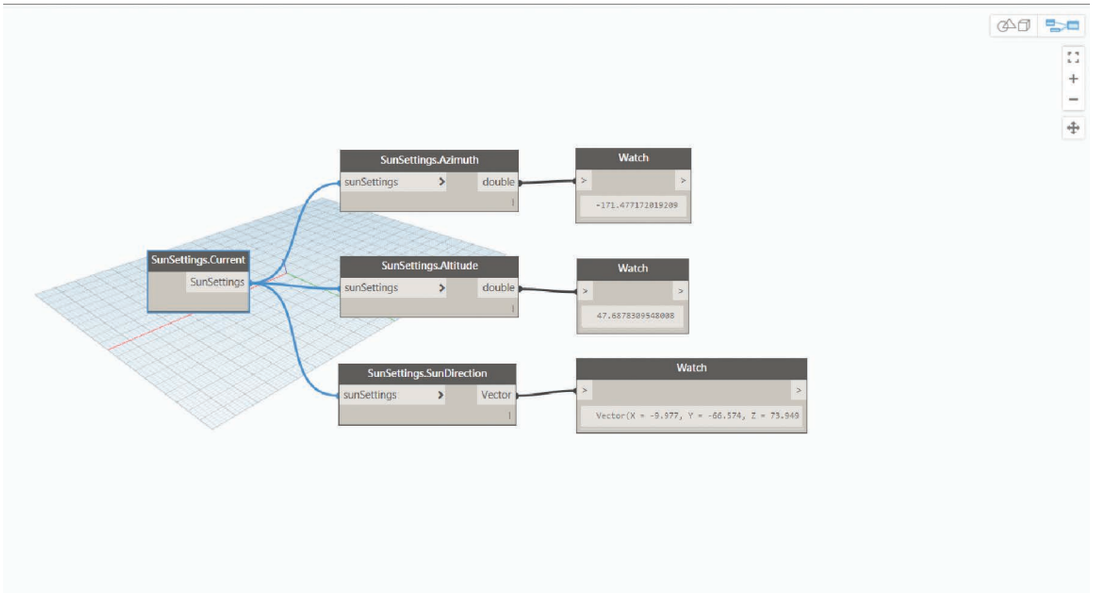


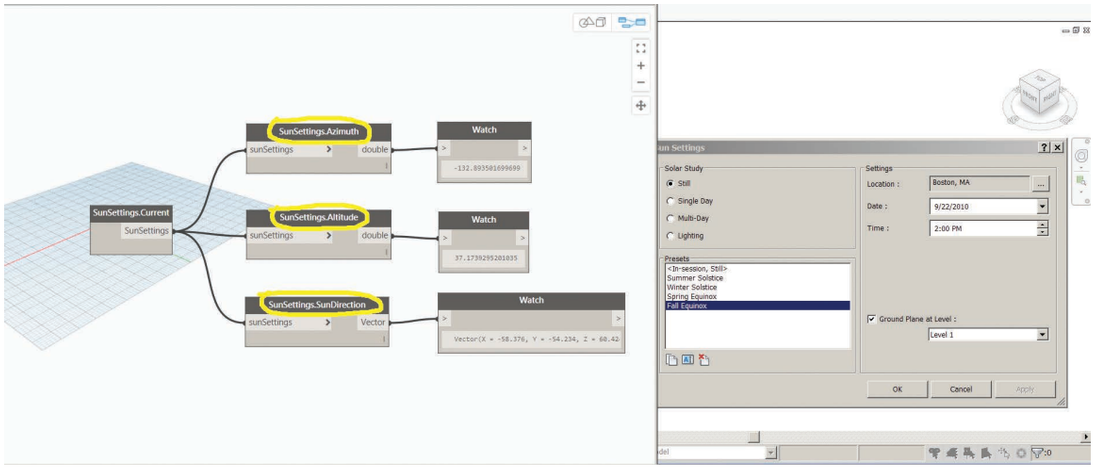
FIGURE 17. Azimuth, altitude, and sun direction are there on the notes. SunSettings.Current reads the values from the Revit file. SunDirection is a short vector (defined by 0,0,0 and the location of the sun) that points towards the current direction of the sun.



2.3.1 Review solar nodes input and outputs

There are several solar nodes in Dynamo, four of which are current sun settings, sun direction (a vector), altitude, and azimuth. They receive their values from the sun's location in the Revit file. First set the location, date, and time in Revit. Then Dynamo can access those values.

FIGURE 18. The nodes in Dynamo are responding to the sun's position in Revit.



2.3.2 Have solar altitude set the height of the box

This step is essentially the same as 2.2.4 except that the sun's altitude is setting the height of the box rather than the distance to an attractor point. Because the altitude is one value, all the boxes will be the same height.

2.4 Adaptive façade component responding to solar angles

The final exercise replaces the box and its height parameter with an adaptive component and an opening size parameter. To simplify the exercise, a "generic model pattern based family" is used on a conceptual mass that has been divided into "panels" in Revit. In addition, instead of solar altitude setting the opening size, the difference of the surface normal and the altitude will be used so that façade components facing different directions will react differently.

FIGURE 19. Note that the boxes are all the same height in Revit because the angle (90 – altitude)/4 is the same for all of them.

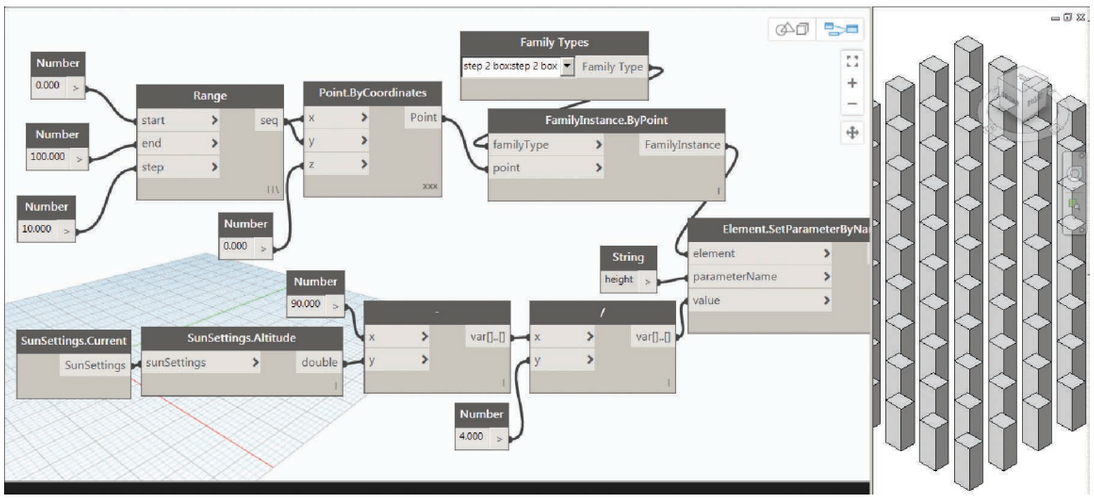
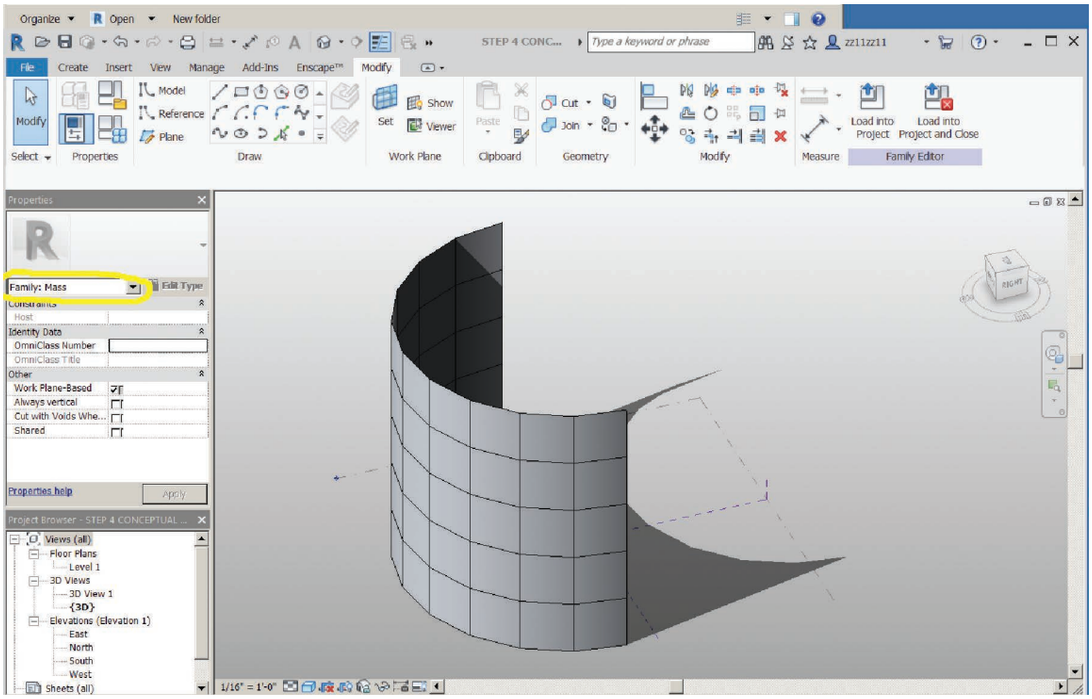


FIGURE 20. The massing component faces south.



The first step is to create the conceptual mass and the generic model pattern based families in Revit. A conceptual mass family created in Revit that has been divided into 10 by 5 panels. It was purposefully created to have panels face different directions.

Then a "generic model pattern based family" is created. It is loaded into the conceptual mass family. An adaptive component could be used, but it is more complex to place it correctly on a façade, so it was decided to use a pattern instead. A height instance parameter has been added to the hole in the panel.

2.4.1 The sun direction

Using SunSettings.SunDirection, a vector is drawn from point (0,0,0) towards the current sun position. The location, date, or time in Revit will change the coordinates that define the vector.

2.4.2 Find the location of grid points on the conceptual mass facade

With the three nodes on the bottom of the screen (Family Types, All Elements of Family Types, and AdaptiveComponent.Locations), the panel family is chosen, all elements of that family are selected, and the points associated with each of the points of the conceptual mass are determined and placed in Dynamo as reference points.

2.4.3 Convert list (array) of points to one point per panel

In order to reduce redundancy, the four coordinates for each panel are reduced to just one coordinate per panel. In Dynamo, this involves two steps: flattening the list of point locations and slicing the point coordinate list so that only every fourth point is kept.

FIGURE 21. The panel was created as a generic model pattern based family.

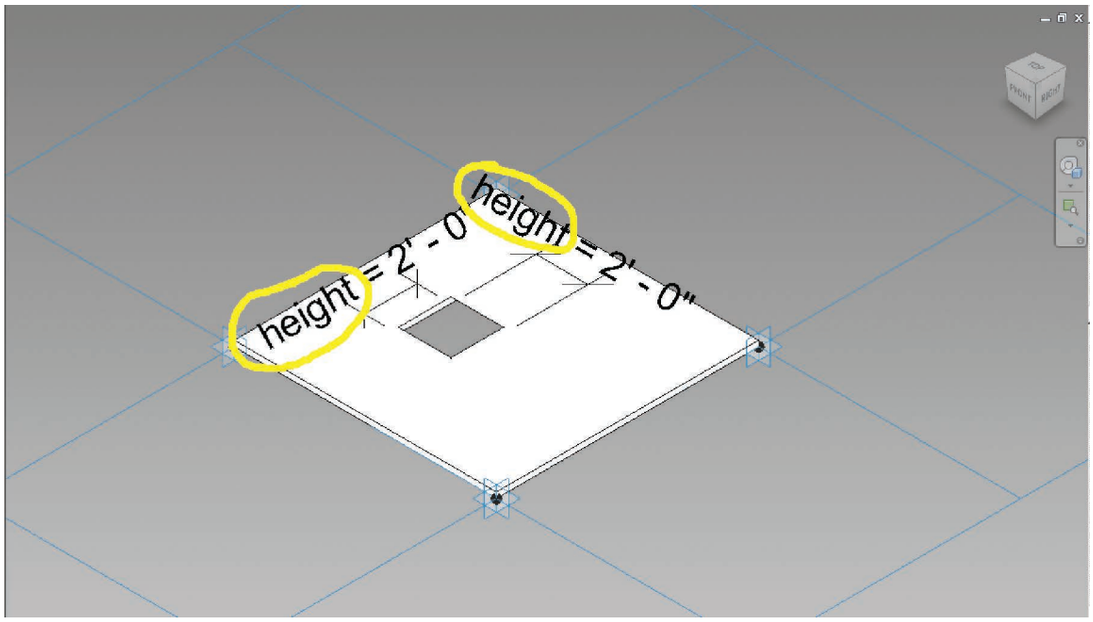


FIGURE 22. In the conceptual mass file, the panel is placed on the façade by picking the component in the pattern list.

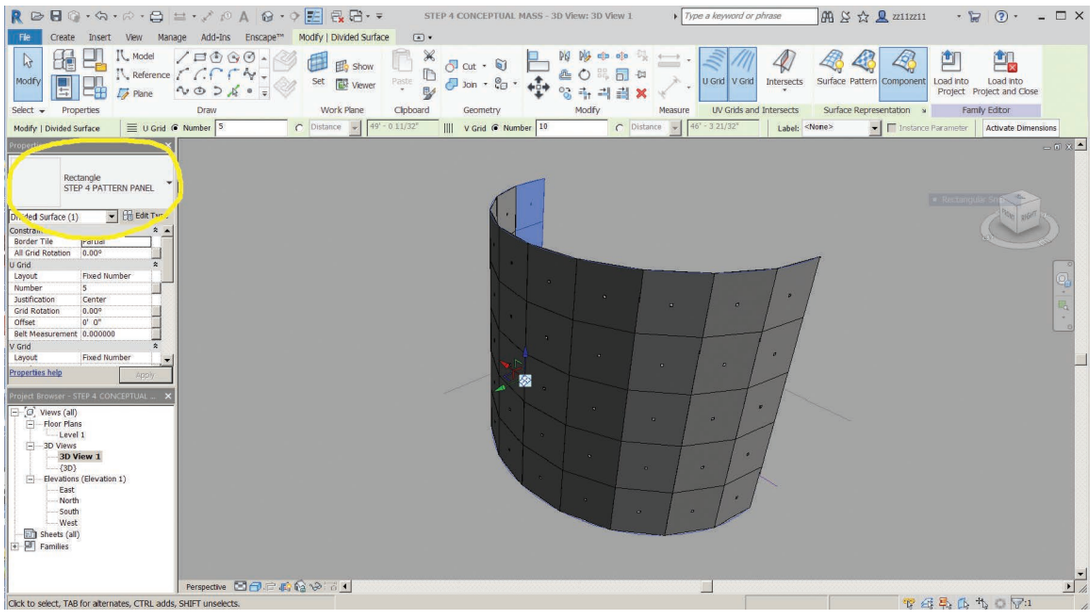


FIGURE 23. Note that the automatic running of the script has been changed to manual so that several steps can be done without triggering the script. Remember to hit Run when needed.

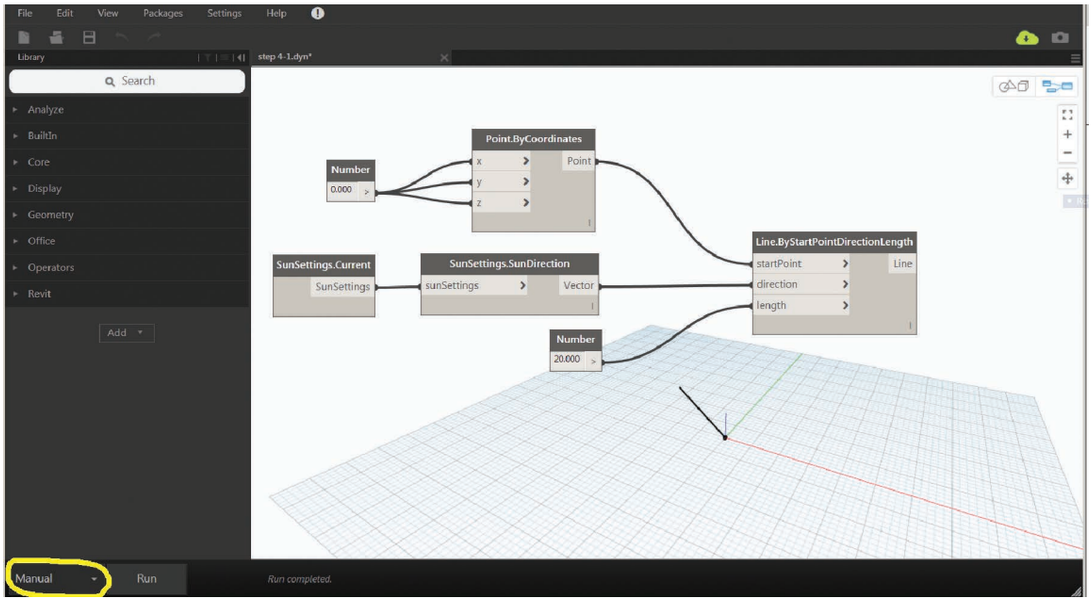


FIGURE 24. Note that each panel has four coordinates, one for each corner.

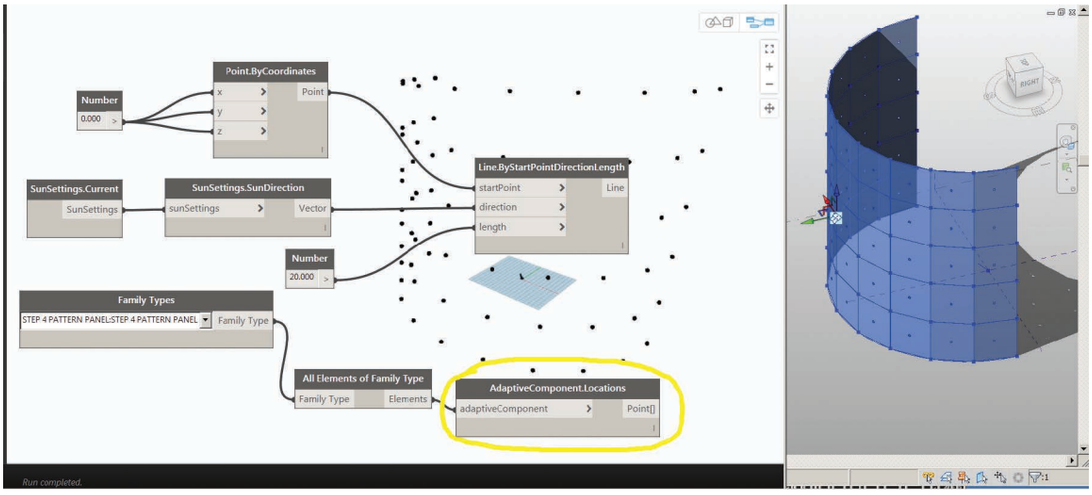


FIGURE 25. List.Flatten uses the AdaptiveComponent.Locations component to flatten this list by 1 degree.

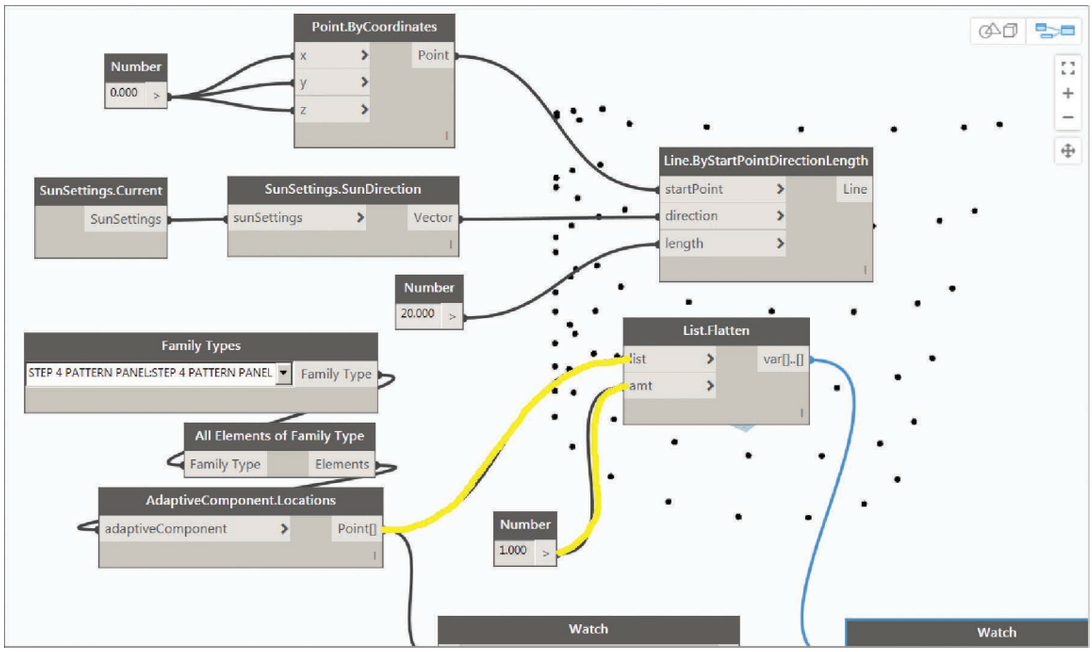


FIGURE 26. The Watch node shows the difference between the original point list and the flattened point list.

FIGURE 27. List.Slice takes the flattened list, starts at the zero member (lists begin with zero, not one), ends at the last member of the point list (determined by List.Count), and steps by 4 so that only every fourth point is left. This is to reduce the redundancy in the list; the original list had four points per panel and some of them overlapped.

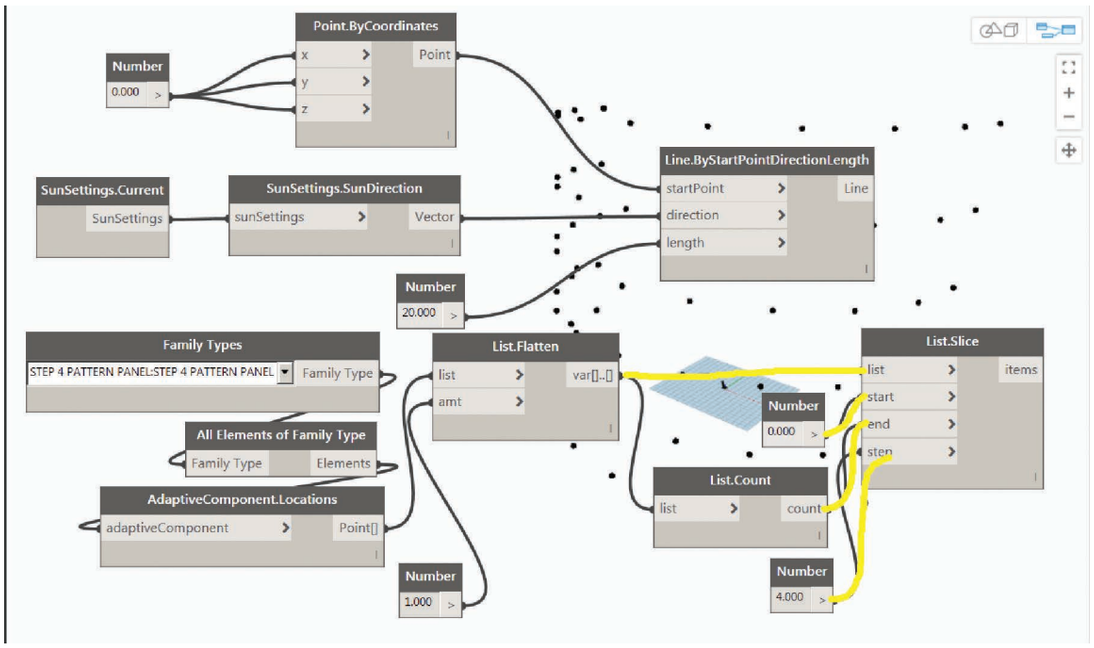
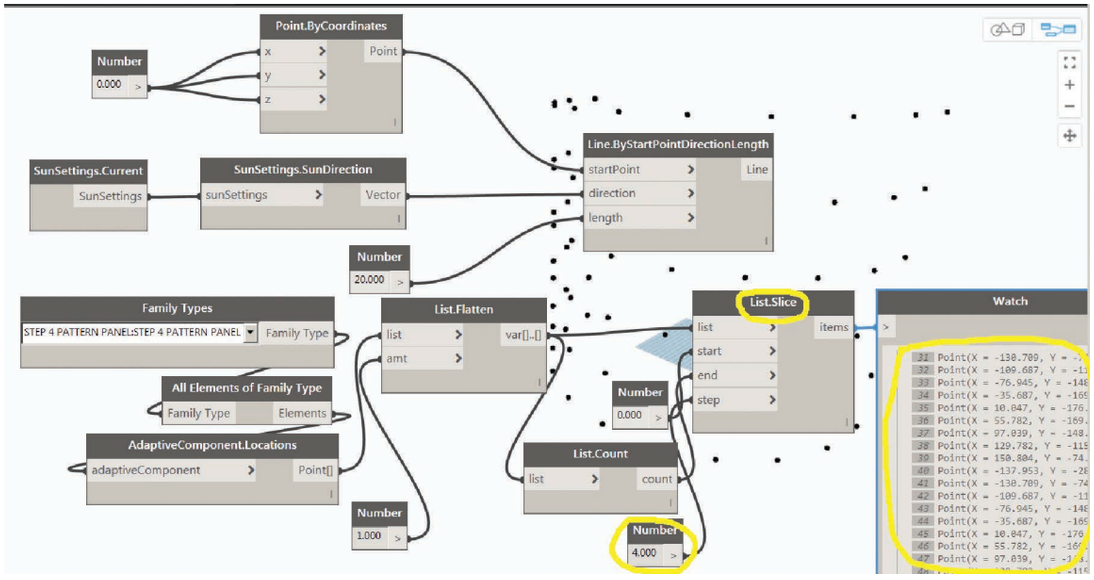


FIGURE 28. The Watch node shows that only every fourth point is left in the list.



2.4.4 Visualize the normal vector

For each of the points, a sun direction vector is added. The difference between this angle (the current altitude of the sun) and the normal vector of the planes will eventually determine the size of the opening in the façade panel.

Polygons are created in Dynamo from the reference points. Then for each polygon, a normal (perpendicular) vector can be determined. The difference between this vector and the sun direction vector will be used to size the opening in each of the panels. Other solar angles could have been used. This is just one method.

FIGURE 29. Sun direction vectors shown in both Dynamo and Revit.

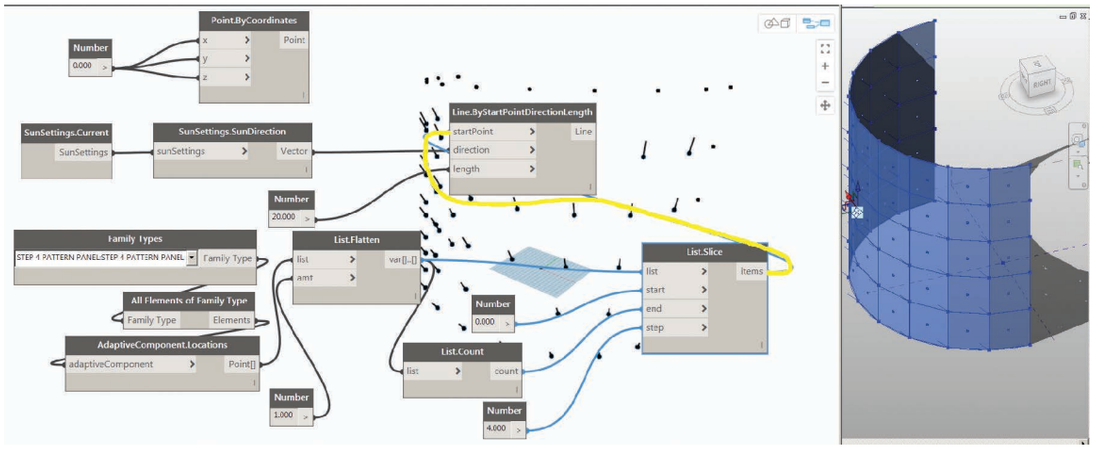
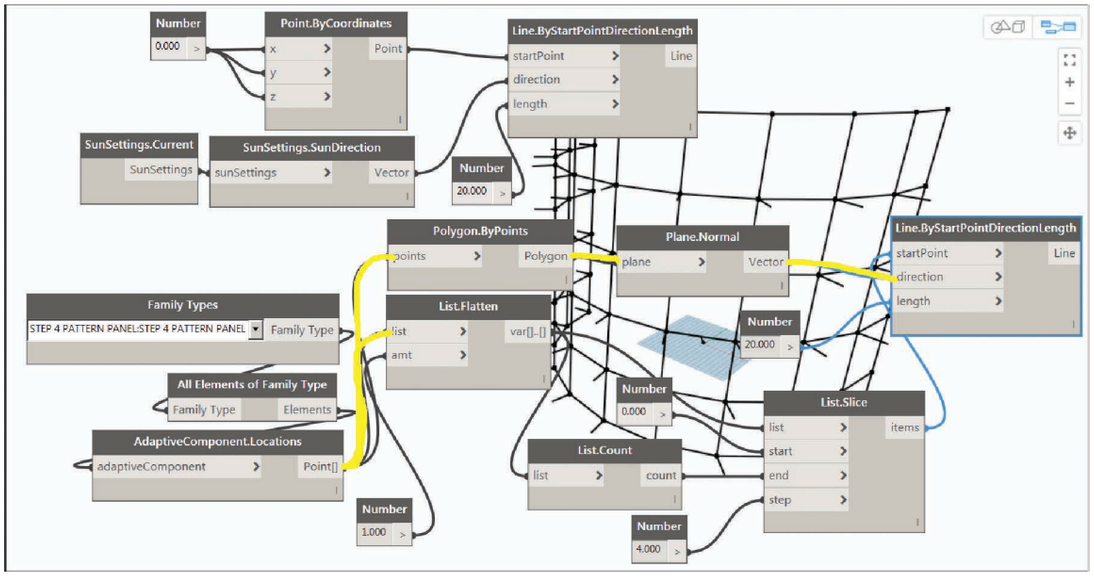


FIGURE 30. The points are connected as polygons in Dynamo so that the normal (perpendicular) vector can be determined.



2.4.5 Calculate angle between two angles and visualize results

For each panel, the difference between its normal vector and the sun direction vector is calculated. This number will be used to change each panel opening's size. This is analogous to the previous circle exercise where the distance from the attractor point to the center of the circle set the radius of the circle.

FIGURE 31. Vector.AngleWithVector is used to determine the difference in the angles.

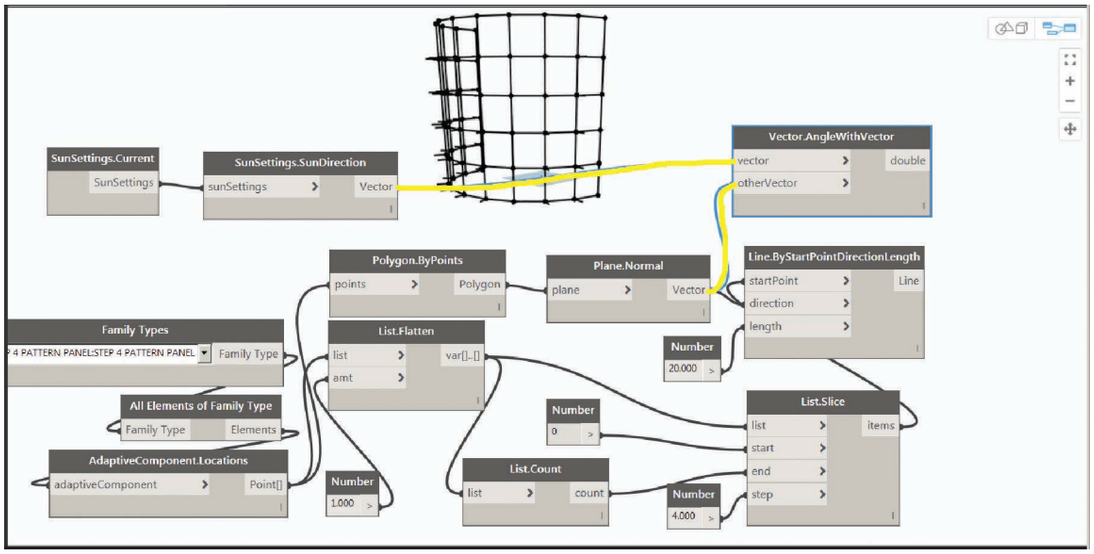


FIGURE 32. After trying several numbers, 5 was chosen as the factor to divide by so that the results made sense for putting into the height instance parameter of the panel.

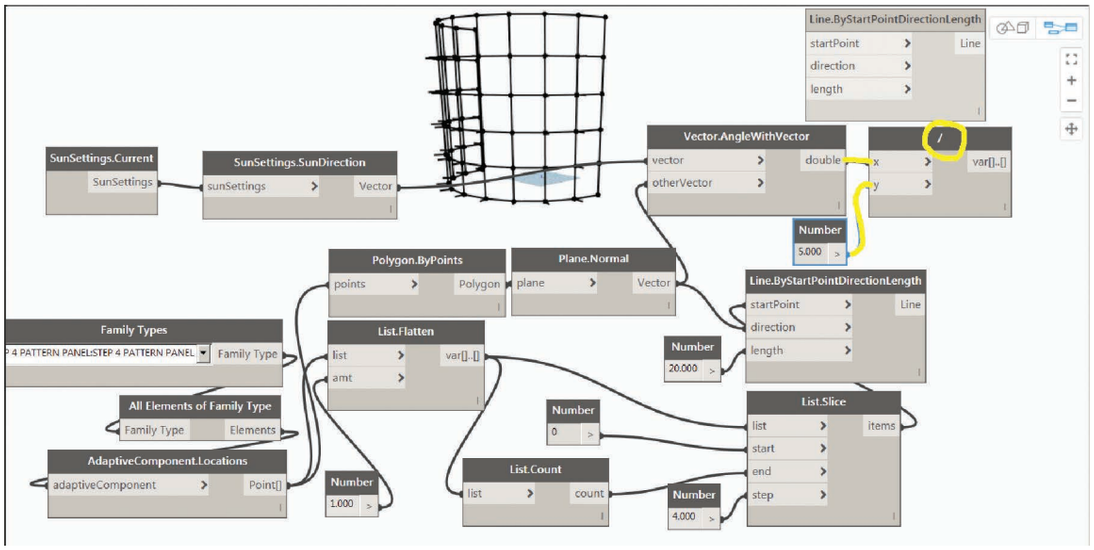
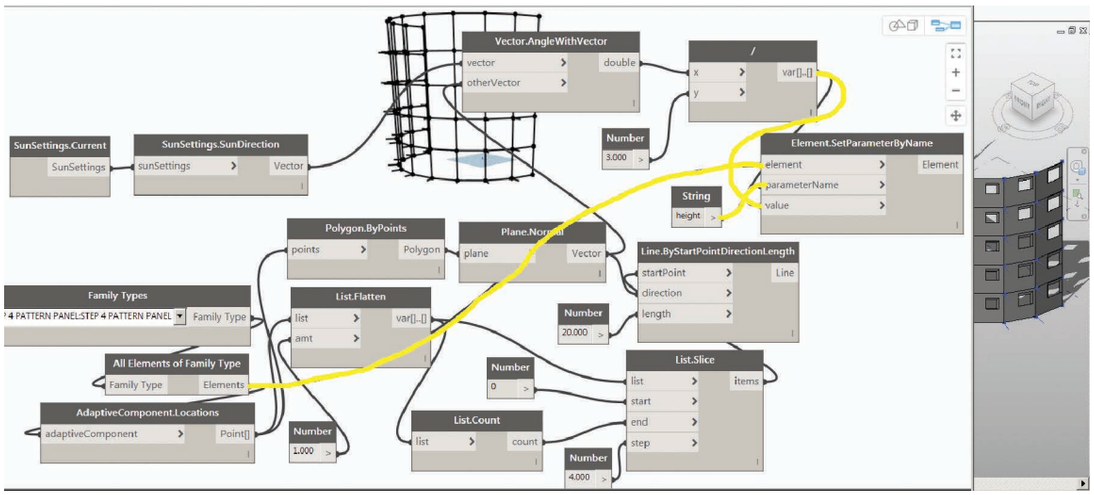


FIGURE 33. Element.SetParameterByName takes as its input all of the panels, the name of the instance parameter (height), and the difference vector angle (divided by 3) to make the hole in the panel smaller. As the solar angle is changed in Revit (by editing the location, date, or time), the opening in the façade should change size when the script is run in Dynamo.



2.4.6 Re-size vector angle

The vector angle might be too large or too small in comparison with the size of the panel. It is divided by another value to make it reasonable. This is similar to how the radii of the circles were re-sized also.

2.4.7 Set size of opening in façade based on difference of the two angles (vector angle)

The difference vector angle is attached to the height parameter of the Revit adaptive component.

Congratulations! You should now experiment by making more complex adaptive components and re-running the Dynamo script.

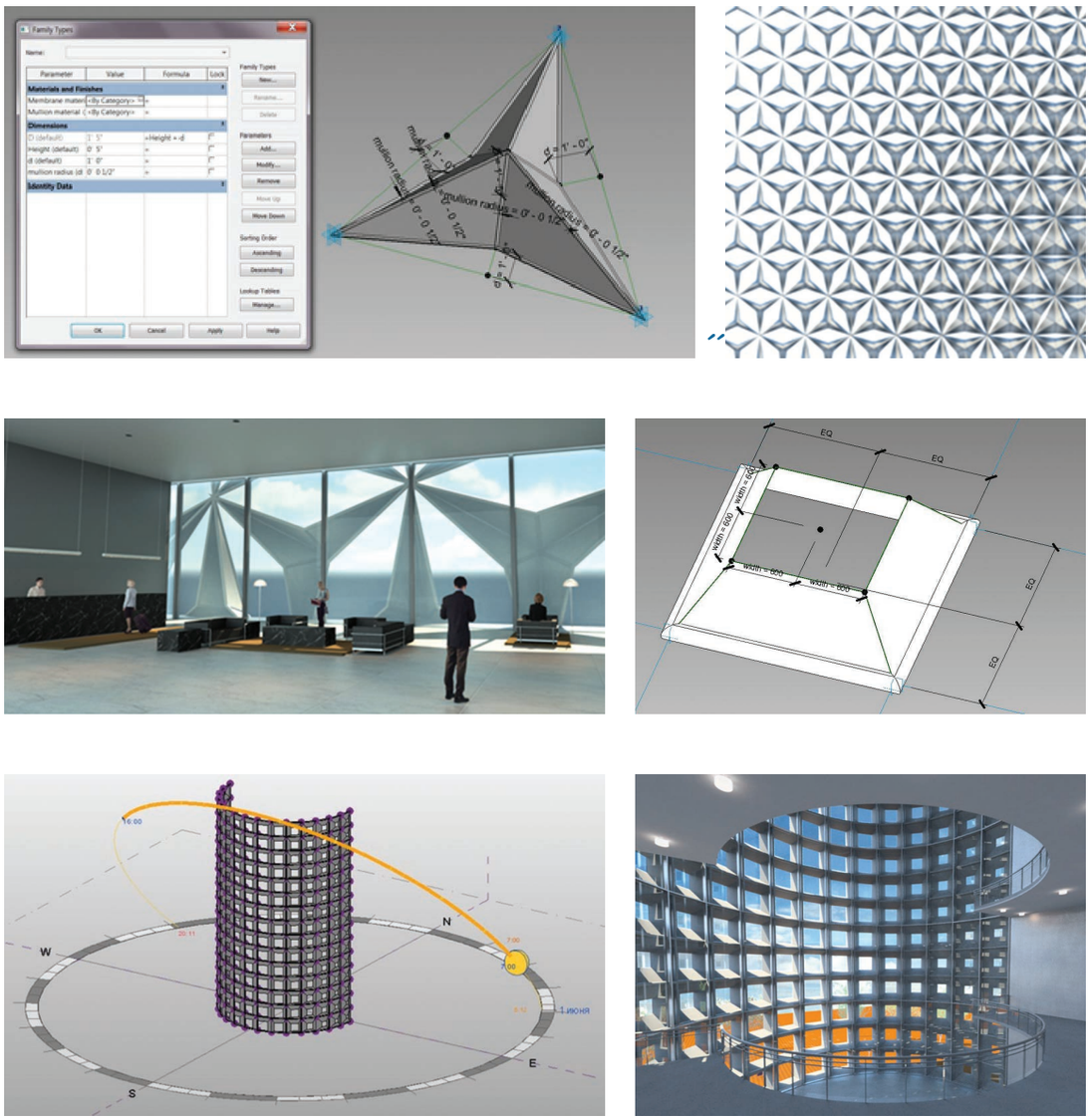
2.4.8 Other examples

Here are a few examples of other adaptive components.

CONCLUSION

Recently, there has been a trend towards visual user interfaces for programming that allow users to create customized, flexible, and powerful form-generating algorithms without having to first learn how to write code. Visual programming is a type of computer programming where users graphically interact with program elements instead of typing lines of text code. Nodes are created; they can be numbers, sliders for adjusting values, operators and functions, list manipulation tools, graphic creators, scripts, notes, "watch" nodes, customizable nodes, and other types depending on the tool. They are virtually wired together, and the program is resolved from left to right.

FIGURE 34. Revit adaptable components, parameters, and façade. The size of the opening corresponds to difference vector angles (based on the normal/perpendicular angle and sun direction vector) (courtesy of Toldo and Chow—top and Ilyassov and Tazhibayev—bottom).



Both software companies and enthusiastic programmers have been developing a suite of components that enhance these innovative graphic tools, including packages for complex form generation; thermal, daylight, solar radiation, and glare simulations; optimization; structural calculations and live physics; analyzing and designing urban configurations, etc. Users from beginners to advanced can find utility in these tools. The first step is learning the basics of the software and understanding its applicability to the problems and opportunities that arise in building design, construction, and operation.

ACKNOWLEDGMENTS

This paper contains a portion of an already published material and has been altered to the extent needed to illustrate certain features. It is based on a lecture given in a graduate architecture class and was inspired by work done by Reid Johnson from Autodesk and other tutorials that Autodesk has developed for Dynamo.

RESOURCES

The two major visual programming environments used in the building industry are Grasshopper and Dynamo. Recently, Marionette has also been released. There are incredible amounts of information available on websites and YouTube for learning these. The following are good places to get started:

Rhino with Grasshopper

http://www.grasshopper3d.com/

http://www.food4rhino.com/

Dynamo with Revit and Dynamo Studio

In the opening menu of Dynamo are a list of tutorials.

dynamobim.org

http://dynamobim.org/learn/

http://dynamoprimer.com/en/

https://dynamopackages.com/

Vectorworks with Marionette

http://www.vectorworks.net/training/marionette

A good place to look for more tutorials and information about visual programming is Google under "Dynamo architecture" and "Grasshopper architecture." Then select the "image" filter.

A few papers that the author has worked on about visual programming are listed:

Kensek, Kensek; Zhong, Chen; and Yan, Jiayi presented "Creating an intelligent BIM model," at the second Nanhai Ave International BIM Summit, May 2017. Co-organized by the National Higher Education Advisory Committee on Architecture, Xi'an Jiaotong-Liverpool University (Suzhou, China), Shenzhen University, and the China Smart Construction Group, Co., Ltd. Included professional and student work demonstrating the variety of visual programming solutions in the building industry. Zahn Yang and Ceran Li also helped create the presentation.

Konis, Kyle; Gamas, Alejandro; and Kensek, Karen (2015), "Passive Performance and Building Form: An Optimization Framework for Early-Stage Design Support," *Solar Energy 125* (2016): 161–179.

Kensek, Karen; Ding, Ye; Longcore, Travis. "Green Building and Biodiversity: Facilitating Bird Friendly Design with Building Information Models," *Journal of Green Building*, V11 N2 issue, Fall 2016, pp. 116–130.

Kensek, Karen, "Visual Programming for Building Information Modeling: Energy and Shading Analysis Case Studies," *Journal of Green Building*, the Industry Corner, V10 N4 issue, December 2015

Kensek, Karen. "Integration of Environmental Sensors with BIM: case studies using Arduino, Dynamo, and the Revit API," "Integración de sensores medioambientales con BIM: casos de estudio usando Arduino, Dynamo, y Revit API." *Informes de la Construcción*. Vol. 66, 536, e044 octubre-diciembre 2014 ISSN-L: 0020-0883, pp. 31–39. doi: http://dx.doi.org/10.3989/ic.13.151.

Tucker, Tyler, Karen Kensek, Kyle Konis, and Douglas Noble. "Performative Shading Design: Parametric Based Study of Shading System Configuration Effectiveness and Trends." ASES/Solar 14 Annual Conference, San Francisco, CA, July 2014.

Gamas, Alejandro, Kyle Konis, and Karen Kensek. "A Parametric Fenestration Design Approach for Optimizing Thermal and Daylighting Performance in Complex Urban Settings." ASES/Solar 14 Annual Conference, San Francisco, CA, July 2014.

BIOGRAPHY

Karen Kensek is an Associate Professor of the Practice of Architecture at the University of Southern California, School of Architecture, where she has taught for over 25 years. Her research work includes BIM + Sustainability, BIM + digital simulation, building science, virtual reconstruction of ancient places, the role of ambiguity in reconstructions, solar envelopes, and digital design.

Professor Kensek has hosted twelve annual building information modeling symposia at USC, is the co-editor with Douglas Noble on *Building Information Modeling: BIM in Current and Future Practice* (Wiley 2014), and is the author of *Building Information Modeling* (Routledge), which has been released in English (2014), French (2015), and Chinese (2017).