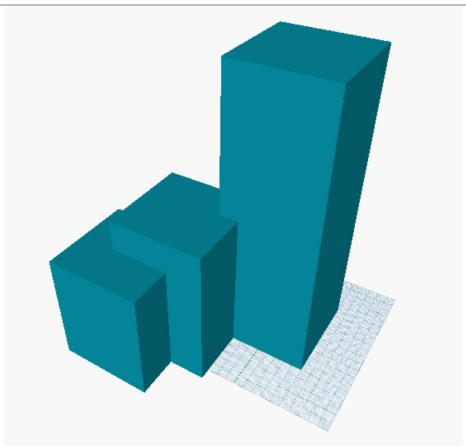
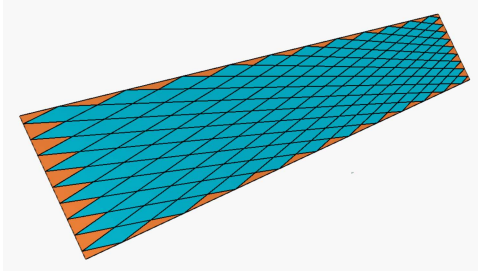
Documentation

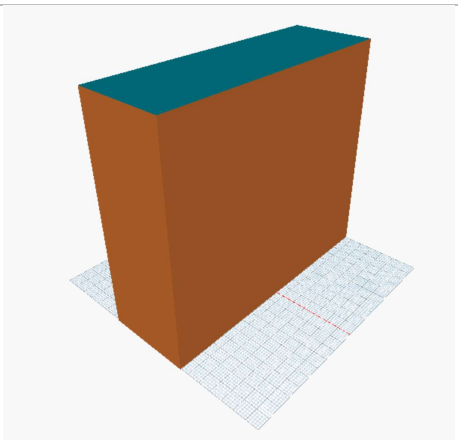
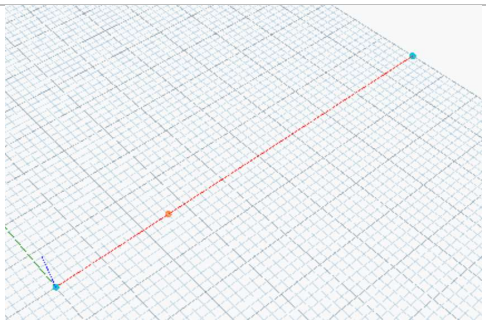
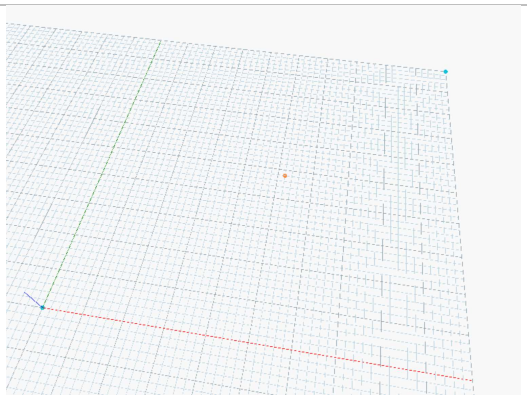
Refinery Sample Files

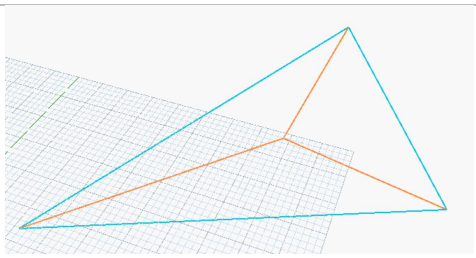
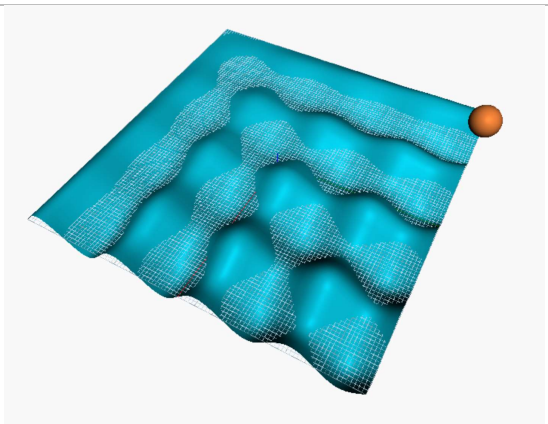
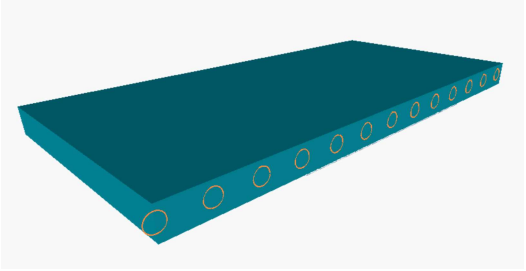
Dynamo sample files to optimize in Refinery

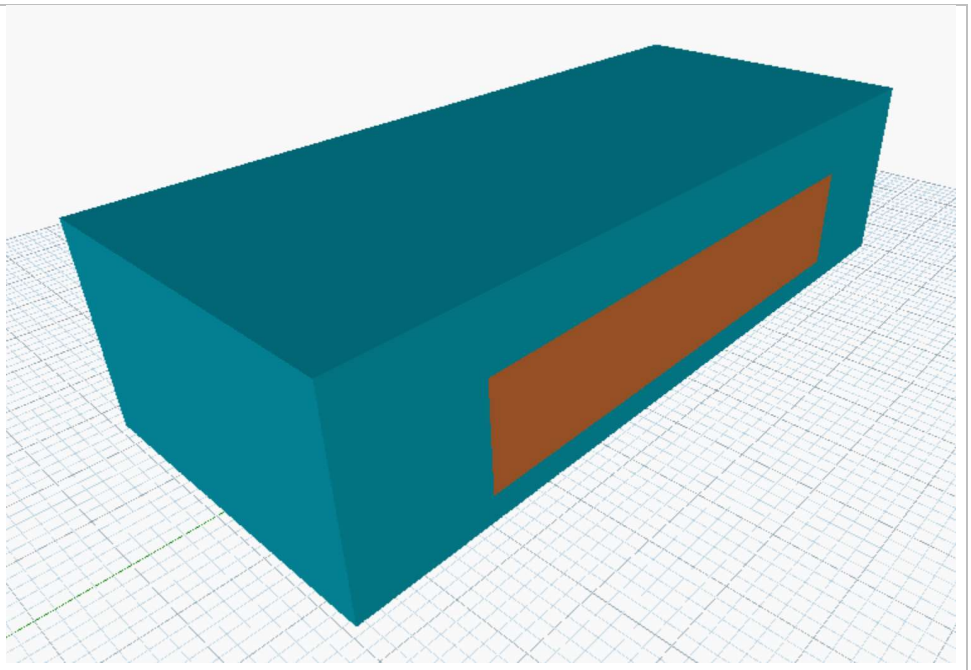
A collection of Dynamo sample files created by The Proving Ground and the Autodesk Project Refinery Team to demonstrate how optimization works in Project Refinery. Note Samples v4 have been updated for use with Refinery 0.35.0.

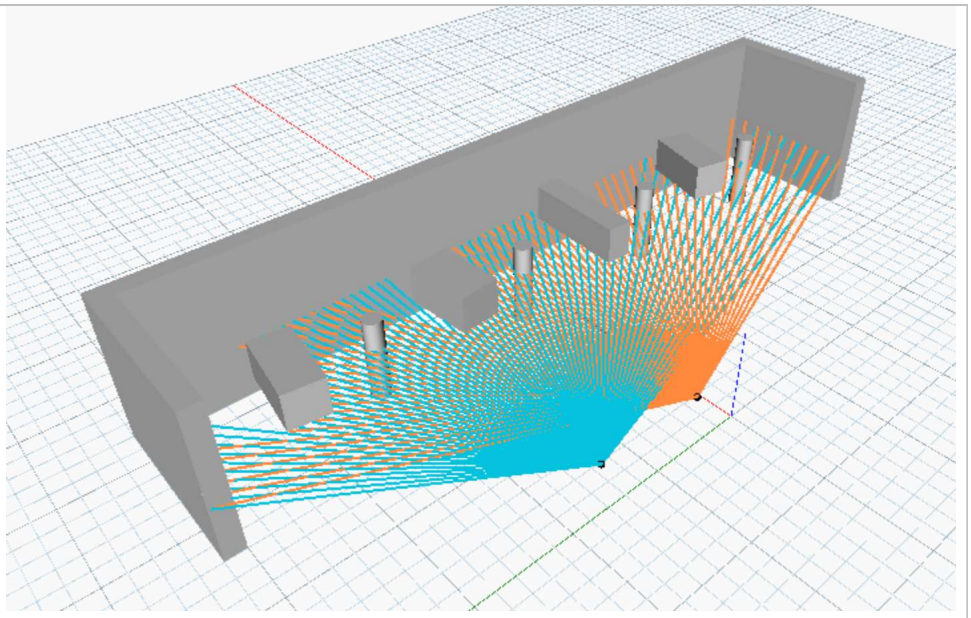
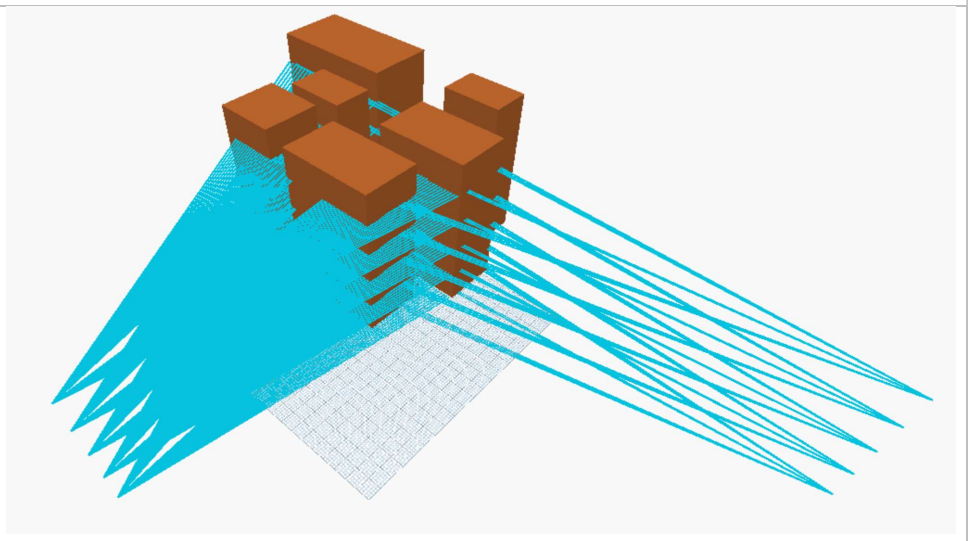
Download samples here

raph Name	Description
-bnh.dyn	This is a test function that is used to evaluate multi-objective optimization algorithms. It is basically a math problem with multiple inputs and multiple outputs that can be used for simple (and quick!) tests. No direct application to architectural workflows. Read more here if you are interested in learning cool math stuff: Test Functions for Optimization
-3BoxVolumeSurfaceArea.dyn	 This graph contains three Cuboids representing buildings. Cuboid One (C1) has a fixed location but varies in Height. Cuboid Two (C2) and Cuboid Three (C3) can both vary in Location (Both X and Y directions) as well as Height. The graph uses Refinery to find the minimum Volume that correlates with the maximum Surface Area.
-DiamondPanels.dyn	

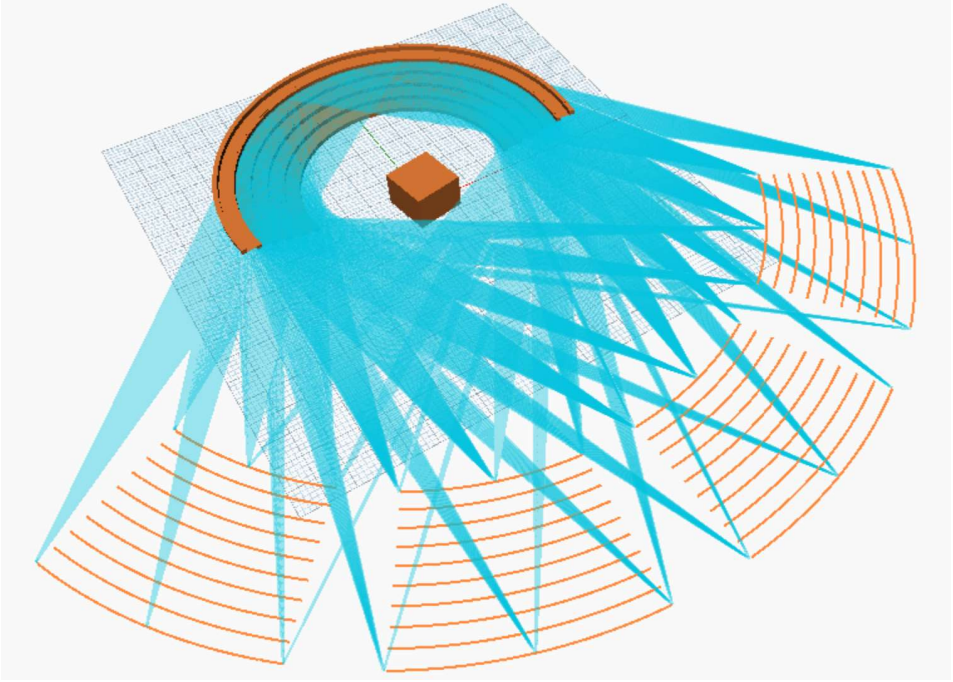
	This graph will take a given surface and apply a 'Diamond' panel across it based on chosen values for the 'U' and 'V' inputs. It will output a series of metrics to run Refinery against and the user to choose the most optimal result. **Note:** This graph needs to be initially run in Dynamo for Revit so that it can obtain a surface from Revit to operate on. It works with the *rac_basic_sample_project.rvt* or you can also select surfaces from other Revit models. Revit nodes will only function in Dynamo for Revit and not in Dynamo Sandbox.
1-MaterialCosts.dyn	 A building with a fixed volume is being built with different materials, and each material cost per square foot is different. In this example, the material for the sides of the building cost $6/sqft while the material for the top and bottom cost $10/sqft. If the length of the building is three times the width, what dimensions are needed to minimize the material cost for the entire building?
5-FindMidpoint1D.dyn	 This graph will find the distance between two fixed points (Start and End) in relation to a changeable point in the X-dimension only. It will get the absolute values (No positive/negative association) and query their distance relationships. If the queried value is "Zero" then the input has found the Midpoint. Refinery is used to optimize this process and find the midpoint computationally.
5-FindMidpoint2D.dyn	

	This graph will find the distance between two fixed points (Start and End) in 2-dimensional space in relation to a changeable point in the X and Y dimensions. It will get the absolute values (No positive/negative association) and query their distance relationships. If the queried value is "Zero" then the input has found the Midpoint. Refinery is used to optimize this process and find the midpoint computationally.
7-PointDistanceArea.dyn	 This graph will take three point inputs dictated by their X and Y values and generate a Triangle. Each point is a certain distance from the center of the shape, and distance to center must be minimized while the area of the shape is maximized. This could be a good starting point for object placement that defines a spatial area.
8-EvaluateSurface.dyn	 This graph will take a given surface and populate a point on this surfaced based on a U and V value. It will then report back the Z value (Height) of the point and can be used to find the highest peak through Refinery optimization. This can be a starter point for those who want to analyze topography of a site.
9-WindowsFloorArea.dyn	 This example is based on natural ventilation codes that say that in order for a room to be naturally ventilated, window spacing must be 2xCeiling Height for windows on opposite sides of a room. For every 1 window the floor length increases by 2xHeight due to window spacing code. 2 windows are added for every additional

	ceiling height of 1' (after 9') which increases the floor ratio (width=1/2Length). Minimize floor area and maximize number of windows. This could be used for those who want to optimize natural ventilation, as it is based on natural ventilation code. More windows may be necessary to add as ceiling height increases as it creates a larger volume to ventilate, but minimization of floor area may help reduce energy costs and create less space to ventilate.
!-*WindowHeightFloorDepth.dyn*	 This example is based on the architectural daylighting rule of thumb that says to maximize the amount of light in a room, the depth of the room should be 2.5x the height of the window. In this scenario, the length of the window increases as the length of the room increases, and the depth of the room increases as the window height increases. This solver aims to maximize window area but minimize depth of the room. This can be a quick tool to gauge how large a room and window can be while maintaining the most natural light.

?-IsovistTest.dyn	 Maximize visibility of objects from one or more points in a room through a collision test between Isovists and barrier objects. This is a great tool to optimize views to patients from nurse work stations. This can be especially important in cases like dialysis clinics where nurses need to be able to have visibility of all patients when obstacles such as RO water cabinets are located between patients. Additional visibility optimization scenarios are possible.
?-Views2Directions.dyn	 This can be a great tool for architects who are developing a complex of buildings either on a single site or on multiple sites in close vicinity and wish to maximize views in multiple directions for all buildings. A city block is being built with multiple buildings, each with their own lot size in which each building can move around. There are great view points both to the north and to the east of the block. What is the optimal arrangement of these buildings on their respective plots so that each building can have maximized views in both directions?

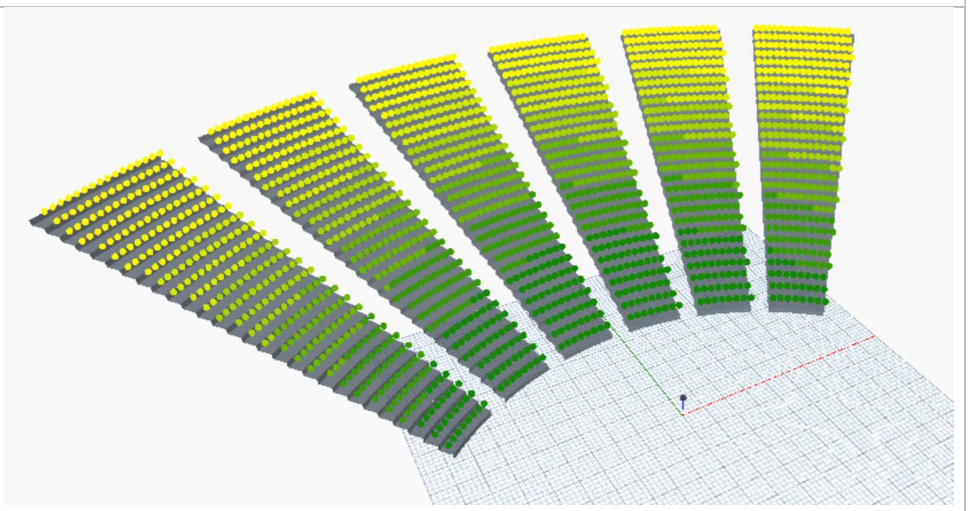
4-ChoirViews.dyn



This example is based on typical church design recommendations for the choir. A set of choir risers is positioned at the front of a room behind the band (Preferred location in front of choir and at center with 20-25 square feet per instrument). The number of riser rows is based on a 12% congregation capacity (capacity is set by the user). Views of the choir from the church pews need to be optimized based on band area rotation, pew aisle width, choir riser incline, choir riser curvature, and pew curvature.

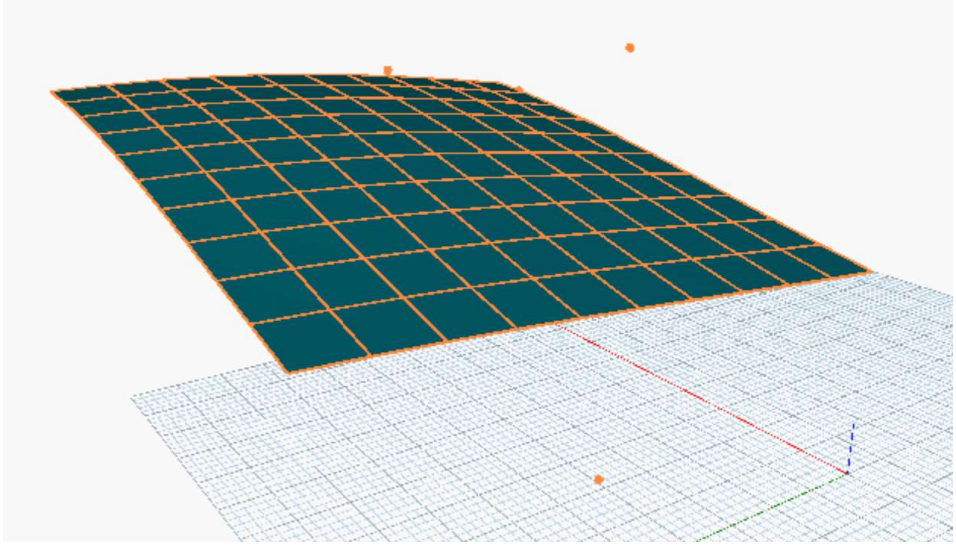
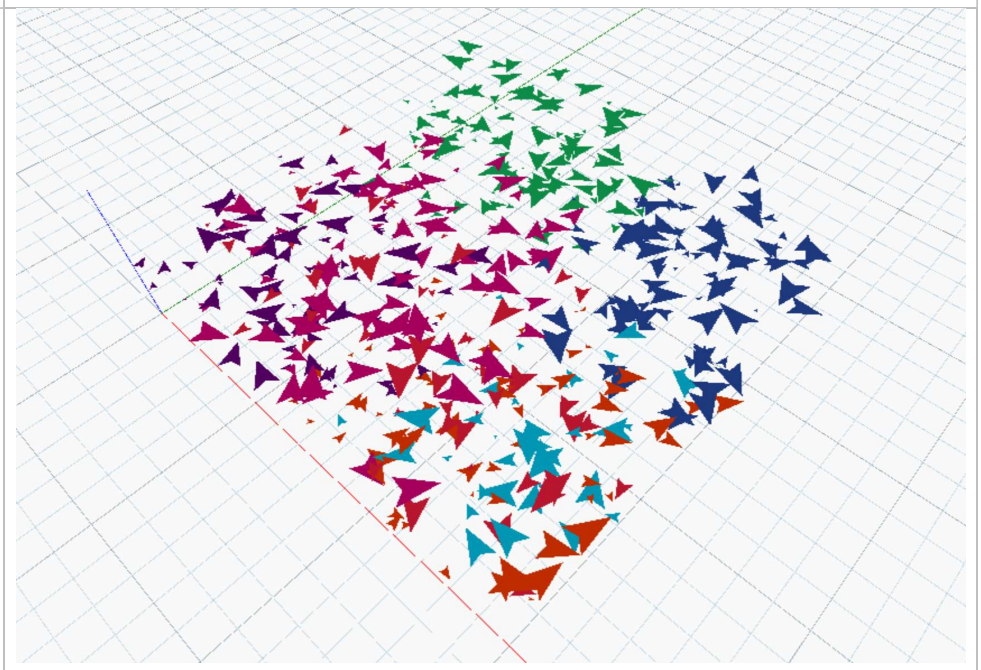
This can be a good starting point for stage design or church design in which the audience views need to be maximized.

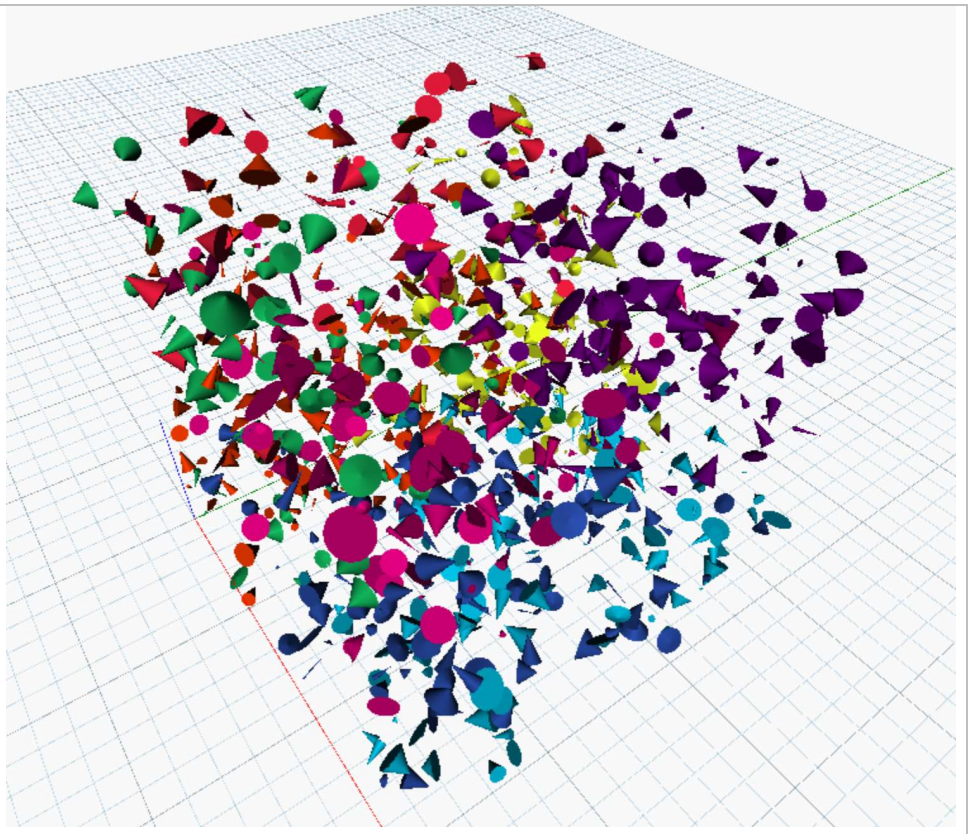
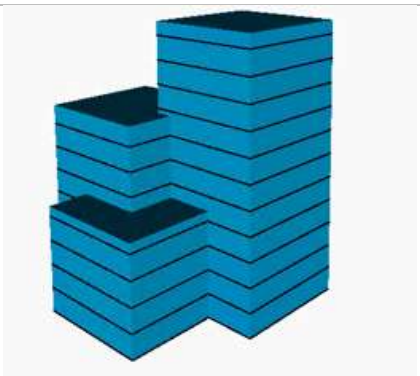
5-AmpitheatreTradeoffs.dyn

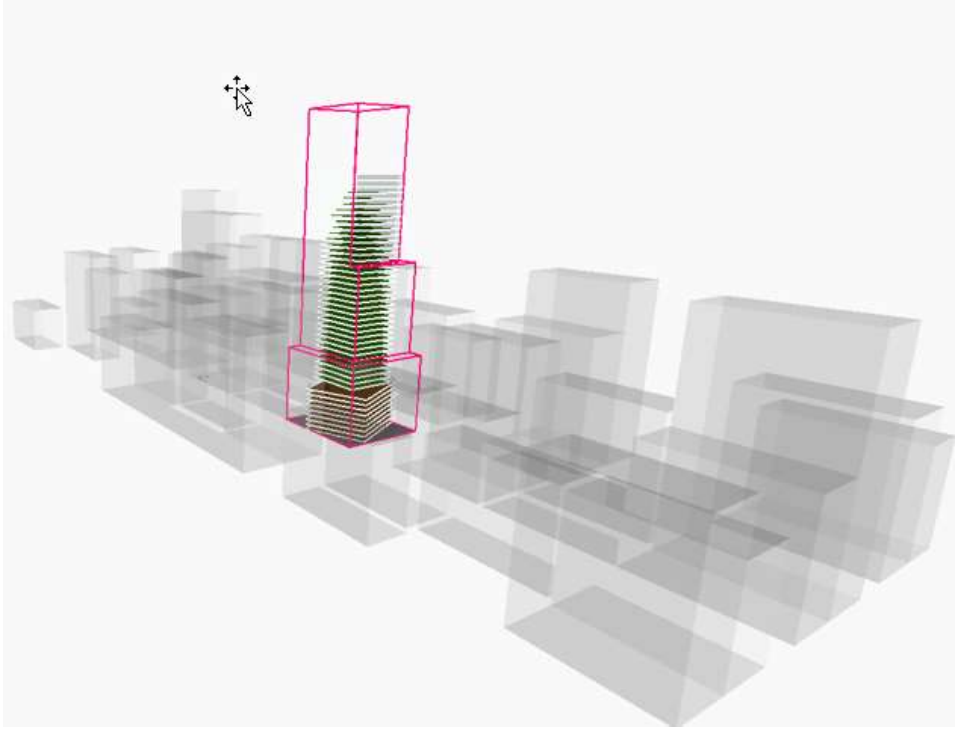


Adjust total width, curvature, and aisle interval at bottom row (subsequent rows offset out). Adjust row count and steepness incline. Maximize number of seats. Minimize average distance to focal point and ratio of obstructed to unobstructed views (clash with heads in front of viewer).

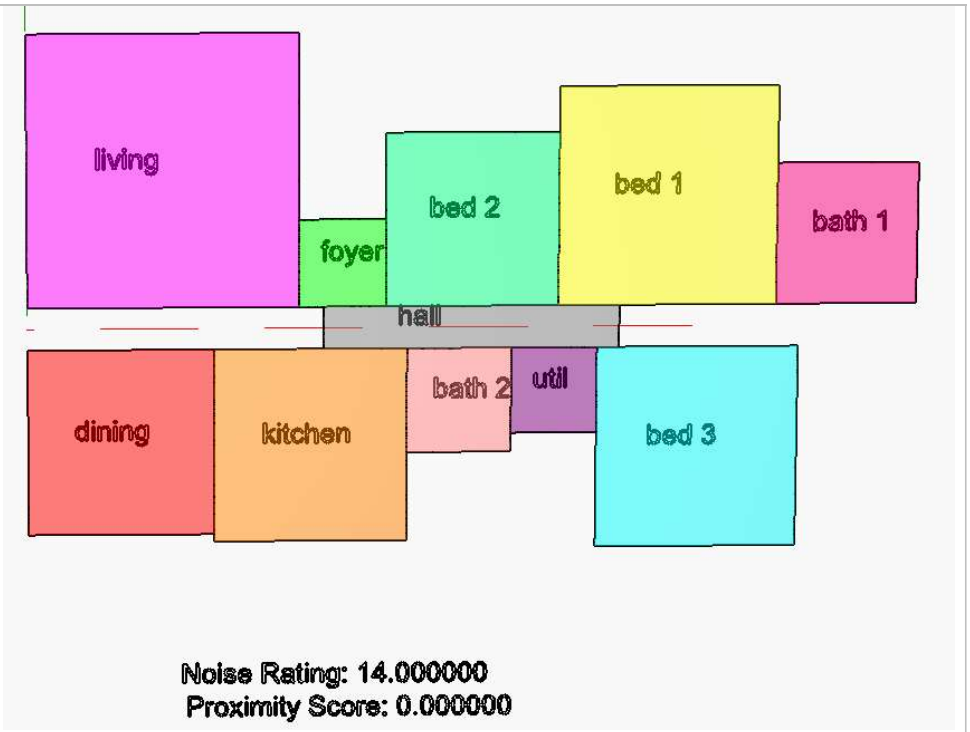
This can be a great tool for stadium design in order to optimize seating for both seat density and viewer sight range.

?-MLSurfaceTest.dyn	 Optimize surface panels using machine learning; outcomes are based on average planarity, average panel area, and average distance from original points. This is a great starting point for panel optimization using LunchBoxML machine learning tools.
3-SchoolOfFish_2D.dyn	 Adjust ML classification deviations by adjusting the importance values. The 'fish' are pointed polygons with values for XY location, XY direction, “swimming” depth, and area fed to the ML Gaussian Mix node. Domains of values are remapped before passing into the ML node; those domain ranges are adjusted by the importance value sliders. Average deviations are analyzed after classification and either minimized or maximized to see a range of input values for output values. This example is more abstract for those who are interested in machine learning optimization problems. This kind of problem is useful for pattern-finding and predictive pattern behavior through data clustering.

ɔ-SchoolOfFish_3D.dyn	 Adjust ML classification deviations by adjusting the importance values. The 'fish' are pointed polygons with values for XY location, XY direction, “swimming” depth, and area fed to the ML Gaussian Mix node. Domains of values are remapped before passing into the ML node; those domain ranges are adjusted by the importance value sliders. Average deviations are analyzed after classification and either minimized or maximized to see a range of input values for output values. This example is more abstract for those who are interested in machine learning optimization problems. This kind of problem is useful for pattern-finding and predictive pattern behavior through data clustering.
*ɜBoxVolumeSurfaceArea_* *ddDetail_SendToRevit.dyn*	 This is an update of the 3BoxVolume sample in the Refinery samples that shows how to split the 3Box model into floors and send the floors into Revit for continued design development. It also shows how you can use Refinery to create random and optimized studies focusing on the goals of maximizing volume and minimizing surface area.

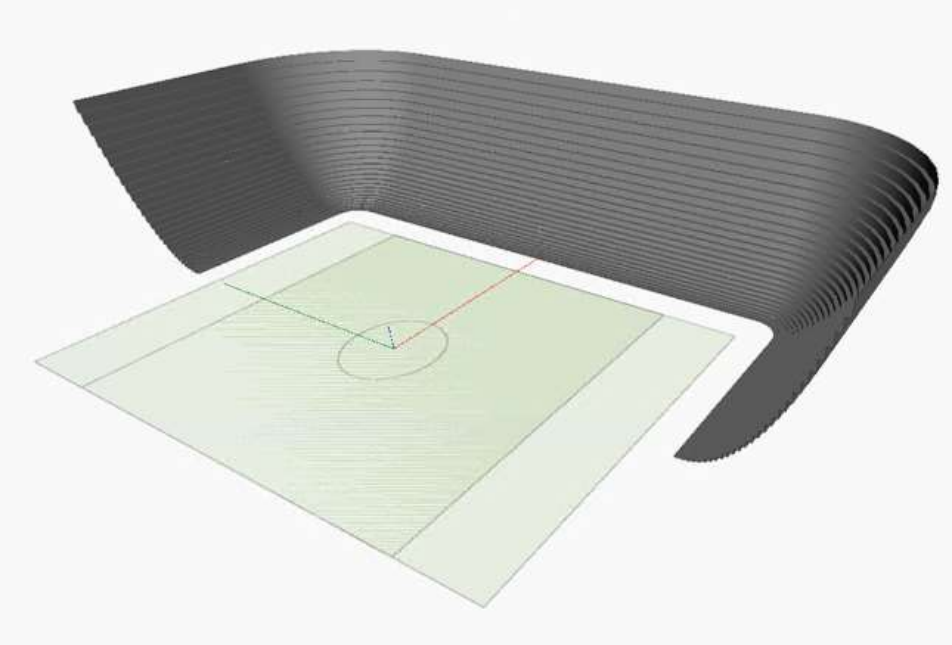
uildingMasser-2programs.dyn	 This example uses site context and a zoning boundary created In Revit and studies the programmatic distribution (office and retail) and configuration for a building on an urban site. It varies the ratio of retail to office and the size and rotation of the retail and office blocks. It has a number of metrics Included that Inform the goals of minimizing cost, maximizing total value per year and minimizing zoning envelop overlap.
olarRadiation.dyn	 This Dynamo graph computes the solar radiatioin on a fixed building geometry accounting for the shading caused by surrounding buildings. The graph rotates and positions the building on the site to minimize solar radiation and maximize total glazing area.

ɔacePlanning.dyn

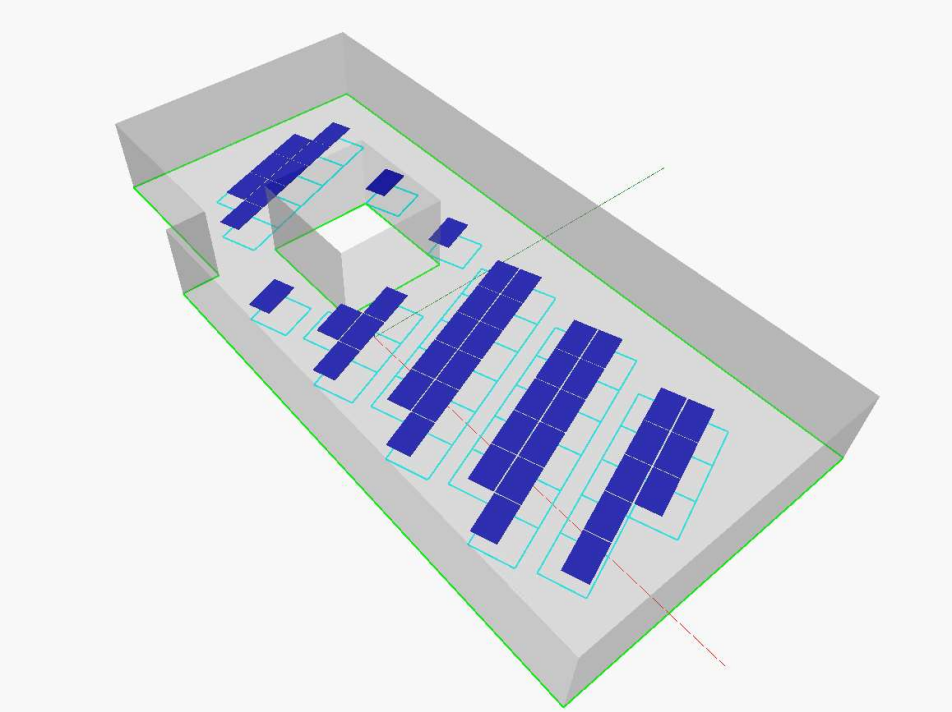


This example generates layouts for a 10-room apartment given a program of rooms, space sizes, desired adjacencies and noise ratings. Refinery can be used to automate design creation and minimize noise rating and proximity score.

:adium.dyn



This graph evaluates the size of a proposed stadium. Varying the height and depth of the seating area, it generates stadium designs that can be maximized for seating area and view quality with the minimal distance to the farthest seat.

penOfficeLayout.dyn	 Given a room from Revit and Inputs about desk size and clearances, this graph will lay out desks in different configurations. It evaluates designs based on desk count, space utilization, area per desk and number of private desks.