Generative design of landforms with Dynamo in Civil 3D

Andreas Luka
Consultant and Director Digital Technologies at Next Generation Landscape Design Beijing

Description

In the first part of this session, we will create an associative parametric model to introduce fundamental concepts that underpin computational design while exploring visual programming in Dynamo software. We will enhance by parameters retrieved from AutoCAD software and Civil 3D objects. We will show how to organize and export a point cloud representing the landform and link it into Civil 3D. There we will integrate this landform as a triangulated irregular network (TIN) model into the model of the existing environment and perform several analyses, including grading, stormwater, and cut/fill balance. Based on this analysis, we will optimize the parametric model. As the model is dynamically linked, we can check the improvement immediately in Civil 3D and iterate the process conveniently. We will also show how to use Dynamo to link landforms created by designers as NURBS with TIN surfaces that engineers can use.

Learning Objectives

- Learn how to import elements from Civil 3D into Dynamo and extract parameters from these elements.
- Learn how to create landforms driven by parameters and formulas inside Dynamo, and how to export the landforms through point lists.
- Learn how to import the point list into Civil 3D, and create and analyze the TIN surface.
- Learn how to establish a workflow for an iterative design process with Dynamo and Civil 3D.
Speaker

Andreas is a German landscape designer with more than 25 years working experience. During his career he worked on high profile projects in Germany and China as designer, project manager and design director. He is passionate about new technologies, started as an early adopter of CAD and now committed to help landscape teams to master their transition to LIM, the adoption of BIM in landscape design. His approach is pragmatic, collaborative and focussed on fast realisation of BIM/LIM's benefits.

He trains related software, implement standards, templates and procedures. He is also the developer of landscape design add-ins like the LX-Stair application and a frequent contributor to the Autodesk Civil 3D developer forum. He currently works on dynamic trees for Revit and Civil 3D in cooperation with the landscape departments of the Eastern Switzerland University of Applied Science in Rapperswil and Tsinghua University Beijing.

http://www.luka-consult.de/about.html
Introduction

Parametric, Computational and Generative Design

The parametric design process is based on rules and parameters to define and describe design intent and design response. In a computational design process, the design is described in a programming environment with a programming language. The generative design extends this process by iterations (varying the parameters) and constraints to generate a certain number of valid outputs. The designer or a learning algorithm than fine-tune the feasible region by changing minimal and maximal values of an interval in which the parameters meet the set of constraints, to reduce or augment the number of outputs to choose from (fit survivor) and create the next generation.

<table>
<thead>
<tr>
<th>Traditional Design</th>
<th>Parametric Design</th>
<th>Computational Design</th>
<th>Generative Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drafting, 3D Modeling</td>
<td>BIM Modeling, Algorithms encapsulated</td>
<td>Modeling, Algorithms user defined, Automated repetitive tasks</td>
<td>Option generation, Optimization</td>
</tr>
</tbody>
</table>

Dynamo for Civil 3D, DesignScript, Python and Zero Touch Nodes

**Dynamo nodes**

Dynamo is an environment for visual programming, that allows users without a programming background to build algorithmic processes to implement as tools and resources. Rather than by typing code, in Dynamo programs are created by manipulating graphic elements called 'nodes'.

Each node performs a specific task. Nodes have inputs and outputs. The outputs from one node are connected to the inputs on the other node using 'wires'. The program or 'graph' flows from node to node through the network of wires. The result is a graphic representation of the steps required to achieve the end design. This approach to programming is better suited for visually oriented types, like architects, designers, and engineers.
Dynamo for Civil 3D extends this concept by providing nodes that access Civil 3D objects like TINS, feature lines, and corridors thru the Civil 3D API (Application Programming Interface) in a more accessible manner than typical programming languages like C# or C++ and allow them to automate repetitive tasks, access civil objects data, generate multiple design options, and test performance.

**DesignScript**
Visual data flow programming uses incredibly simple rules for connecting nodes together. All nodes follow this simple graph-node convention and allowing users to discover more functionality even from unfamiliar nodes. As a result, visual programming has an intuitive access quality, so users with the minimum of training are immediately productive and able to explore new functionality. However, this initial 'ease of use' advantage of visual programming can also become a disadvantage as projects getting more complex.

By implementing a series of intermediate programming techniques, DesignScript is a 'soft transition' between visual data flow programming and text-based scripting. Thus DesignScript provides a gentle learning curve that allows the gradual introduction of more advanced programming concepts and notation. DesignScript connects the pragmatic world of design automation and production to the world of computer science and algorithmic thinking. With these new possibilities, users can move beyond automating existing design processes to new forms of computational design and architectural expression. From a technical perspective, DesignScript is a multi-paradigm domain-specific end-user language and modeling environment for architectural and engineering computation.

DesignScript was originally developed at the Autodesk Singapore Research and Development Centre and is now the computational engine within Dynamo. Nodes or groups of nodes with wired logic can be automatically translated to DesignScript. Civil 3D and AutoCAD elements are exposed thru 'out of the box nodes'.

**Python**
The third way of creating programs in Dynamo is using Python, a high-level, general-purpose, interpreted open-source programming language. Using Python gives the user access to a wide range of modules and packages for mathematical and data science tasks. Python can access AutoCAD and Civil 3D objects thru APIs.

**Zero Touch Nodes**
Zero Touch nodes are custom, ready to use nodes written and compiled in C#. Writing custom nodes in C# improves type-safety and robustness of code and enable the development of programs that would be difficult to achieve using Python and DynamoScript. As programs written in C# compiled before runtime, they execute quicker than Python, making it an ideal option for projects that demand a high degree of performance. Zero Touch Nodes provide access to Civil 3D objects currently not accessible thru the 'out of the box nodes'.

Page 4
Land forms in engineering, landscape design and art

Grading and terrain modeling is a key part of site engineering, landscape design, and land art. Shaping the Earth’s surface must not only developing the aesthetic potential but also requires ecological sensitivity and profound technical competency. The successful collaboration between artists, designers, and engineers rely on the integration of tools and data for the form-finding process engineering software for analytical tasks as well as tools for the generative design.

This lecture will break down the process into smaller tasks and a flexible approach. In the first part, the focus is on AutoCAD's native parametric design elements (NURBS and procedural surfaces) and their integration in TIN surfaces. The second part has a focus on computational modeling techniques for the parametric form-finding process. The third part focuses on workflows to leverage the strength of each tool.

Learning Resources:

https://primer.dynamobim.org
https://developer.dynamobim.org
https://dynamonodes.com
https://github.com/DynamoDS/Dynamo/wiki

AU University classes from Samir Rezk, Andrew Milford, Stacey Morykin, Dylan Kahle and many more to come
1. Connecting NURBS and ruled surface to Civil 3D TIN surfaces

The drawing contains two paths and three staircases (build with my LX-Stair Add-On), between these five elements two gentle undulated landform one as ruled surface and on as NURBS.

Open the nurbs-1.dwg.

Let's open Dynamo from the ‘Manage’ tab and start. All available nodes are in groups and subgroups in the left hand 'Library' pane selectable. With two 'Select.Object' node (from the AutoCAD selection node group) we can access the two surfaces (or any other drawing object). Many AutoCAD objects can be translated to Dynamo objects. But currently, this node has no implementation for NURBS or ruled surfaces. So I build a ZeroTouchNode that can directly extract points from these objects instead. The node can be installed via the package manager. After installing the ALC-ZTN you have the 'NURBS.GetPoints' and the 'Surface.GetPoints' node available. So let’s use the ‘Select Object node and select the NURBS surface from the Civil 3D drawing and add a ‘NURBS.GetPoints’ node. After connecting the nodes a field of Dynamo points is created that represents the NURBS. The node calculates the points at the ‘u’ and ‘v’ parameter of the NURBS. We can increase the density of the points (and accuracy of the surface representation) by two 'Integer.slider' nodes for the ‘udiv’ and ‘vdiv’ values from the surface.

The output of this node is a structured list of Dynamo points, as we can see in the ‘Watch’ node, but not in the format expected for a point list in Civil 3D CSV import. With the ‘Point.X’, ‘Point.Y’ and ‘Point.Z’ nodes we can extract the values only in three separated lists. We combine the three list in to one with a ‘List.Create’ node, displayed in a ‘Watch node’. We need
to transpose the list to get the right order and rank of list, suitable to connect to ‘Data.ExportCSV’ node. This node also needs an input for the file location, provided by a String Input node. We duplicate all nodes by Copy/Paste, change the file name in the copied input string and using the copied ‘Select Object’ node to select the ruled surface. Dynamo gives us a warning as the copied ‘NURBS.GetPoints’ node can’t take a ruled surface as input. We need to replace this node with a ‘Surface.GetPoints’ node and rewire accordingly.

For better readability we rearrange and group nodes. The node creates two CSV files

nurbs.csv
surface.csv

We are now able to use his two CSV files will as definitions for a TIN surface inside Civil 3D, so we can make use of all tools from Civil 3D to analyze the surfaces. If necessary we can adjust the surfaces, run Dynamo again, and rebuild your surface.

Final package and dwg:
nurbs.dyn
nurbs-2.dwg
2. Creating land forms driven algorithmically inside Dynamo and adding them as point groups to a surface definition

Creating single point (peak)

We start the process starts with an empty metric drawing. We also set the Dynamo mode to Automatic.
We develop the graph backward from output to the necessary inputs. The output to Civil 3D will be a single Cogo Point using the 'Point.ByCoordinates' node (from the Civil 3D nodes). The inputs for this node are document, layer, geometry, and raw description. We can get these values with the 'Document.Current' node and 'Document.CurrentLayer' (both from the AutoCAD nodes) and lace them to the 'CogoPoint.ByGeometry node'. Geometry means a Dynamo geometry object, in this case, a Dynamo point, created with the 'Point.ByCoordinates' node (from Dynamo Geometry nodes). The Point node requires three input values. We using a 'Number sliding' node for the height of the point, laced to the Z input of the 'Point.ByCoordinates'. We can keep the values for X and Y by the node’s standard values (0). The last thing missing piece is the raw description. A 'String Input' node will provide this value.

Developing a graph backward from the output is a reasonable approach, as this is also the logic, how Dynamo is evaluating this graph at runtime. In the last step, we rearrange the graph for better readability.

By using the slider the point we move the point in in Dynamo and Civil 3D accordingly. This point will be later the peak of the cone.

GRAPH FOR SINGLE COGO POINT
Creating a single cone

A sequence of points will approximate the circle for the bottom of the. Therefore we need a 'Circle.ByCenterPointRadius' node (from the Geometry group) and a second 'Point.ByCoordinates' node as well as an additional 'Number slider' for the radius. If we would try to lace the 'Circle.ByCenterPointRadius' node directly to 'CogoPoints.ByGeometry' node we would get an error message, as the node can only accept points. We need to use a node that can create a sequence of points along the path of the circle. The node 'Curve.PointsAtEqualSegmentLength' can solve this problem. The number of points can be easily adjusted by adding another 'Integer slider' node with a reasonable value (start 6, end 90, step 6 i.e.). The output of this node is a list of points. By lacing the list output into the CogoPoint node, the CogoPoint node takes each of the values provided by the Curve.PointsAtEqualSegmentLength and process it. To get the peak point, as well as the bottom points, processed, we just combine them to a single list with the 'List.Create' node. We need to add either the start or endpoint of the curve to complete the circle using the 'CurvE.StartPoint' or 'Curve.EndPoint' node. As a circle is a closed curve the points are identical, so it doesn't matter which one. We lace this point also to the list. We turning off the preview for the center point and the circle and rename similar nodes for better readability.

As now all points for a single cone are created we can define a TIN in Civil 3D using the Cogo-points.
In the next step, we are positioning the cone relative to the origin of the project by moving it in x and y-direction. For testing, let's move the cone 50 m in X respective 10 m in Y provided by two number input nodes.

One option would be to move the complete list with a translate node, but it is easier to lace the x and Y values to the input side of the point nodes.

Creating multiple cones

We continue now by placing the cones along a curve (arc). It is much easier to use cylindrical coordinates for points at a curve than cartesian coordinates. Therefore we replace then node ‘Point.ByCoordinates’ for the center of the circle with a ‘Point.ByCylindricalCoordinates’. We can use this point's X and Y values to control the ‘Point.ByCoordinates’ node for the peak. The ‘Point.ByCylindricalCoordinates’ has four inputs. We can leave the cs (coordinate system) and elevation empty, using standard values. For the radius, we use the ‘Number.Input’ node.

To the input angle, we will provide instead of a single value a list of angles from a 'Range' node. The 'Range' node requires three inputs provided by three ‘Number.Input’ nodes. With 0 for the start value, 180 for the end, and 30 for the step, we create a sequence of 6 angles. We can monitor this by a 'Watch' node.

As the number of points increases now, I recommend to change from Automatic run to Manual run mode, as well as unlace the ‘CogoPoint’ node and only lace the node back after the geometry creation works perfectly. Thus we keep Dynamo running fast.
Using functions with DesignScript
We can implement equations like spirals more efficiently in DesignScript than with visual programming. The logarithmic spiral with the polar equation:

\[ r = a \cdot e^{k \cdot \phi} \]

can be represented in Cartesian coordinates by:

\[ x = a \cdot e^{k \cdot \phi} \cdot \cos(\phi) \quad \text{and} \quad y = a \cdot e^{k \cdot \phi} \cdot \sin(\phi) \]

and in DesignScript (‘Code Block’ node. Code block nodes a quickly created by double clicking.):

```designscript
def PointAtSpiral (angleRadians, a, k)
{
    angleDegree = Math.RadiansToDegrees(angleRadians);
    r = Math.Pow(Math.E, angleRadians * a) * k;
    x = Math.Sin(angleDegree) * r;
    y = Math.Cos(angleDegree) * r;
    return Point.ByCoordinates(x, y, 0);
}
```

Note that the spiral function needs angle in Radians, the Math.Sin() and Math.Cos functions (respectively nodes) expect the angle in degree. This function we call in another ‘Code Block’ node. Values for a and k we provide by ‘Number Input’ nodes the angle (in radians) by a ‘Sequence’ node. A ‘Sequence’ node creates similar to the ‘Range’ node a
list of values. But the node expects a start value, a step value (increment), and the number of elements.
The start value and end values are again calculated in the DesignScript block based on windings, provided by 'Number Sliders' nodes. As the size of the cones should change with the distance from the center, we calculate the distance to the ‘Point.Origin’ for each of this point by the ‘Geometry.DistanceTo’ node. In an additional code block, we divide them by the maximum distance (the distance of the last point) and multiply them with the maximum height or radius. We wiring than the points and the radius to a ‘Circle.ByCenterPointRadius’ node for a quick test.

As the spiral works well, now instead into circles the values for radii, heights and positions are laced to ‘Circle.ByCenterPointRadius’ node of the cone.
Now we should change the nodes for creating the cones to DesignScript, using Dynamos Node to Code function. We refactored the code to extract a function that takes the center point, height, radius, and position to build a cone. In a second code block, we call this function with our lists for angles, radii, and distance twice to build the sequences of upright and inverted cones. We rotate the inverted cones by an 180-degree angle.
Cone-7.dyn

The boundary of the artwork are again defined by points on a curve, this time right in DesignScript.

Cone-8.dyn
Placing the cones on the Civil 3D terrain surface

Until now, we placed the landform around the origin of Dynamo and Civil 3D. In this step, we derive the position by interrogating a Civil 3D/AutoCAD object. Before the open the drawing:

cone-landscape.dwg

Dynamo must be closed, as Dynamo only interact with only with one document. After opening the Civil 3D drawing, Dynamo can be restarted and the Script reopened.

---

Dynamo must be closed, as Dynamo only interacts with only one document. After opening the Civil 3D drawing. We can now restart Dynamo and reopen our script.

The Civil 3D drawing contains terrain and circle, marking the proposed position of the landform. We can connect to this circle in Dynamo by a 'Select.Object' node and the 'Document.Current' node (both from the AutoCAD group). The 'Object.Geometry' node changes the AutoCAD circle into a Dynamo circle. We can query the circle’s center-point and monitor the result in a ‘Watch’ node.

We pass the center point's X, Y, and Z values to the ‘Geometry.Translate’ node. This node takes the complete list of points and passes them on to the ‘CogoPoint.ByGeometry’ node. We need to adjust the settings for geometry scaling to 'Extra large' (in the Dynamo menu) as the drawing is in GK coordinates. We also turn off the preview for all nodes except the ‘Geometry.Translate.’ node, so that Dynamo’s ‘Zoom to Fit’ zooms to the points at transformed points.
The landform is also aligned in Civil 3D. With the circle

Establishing a workflow for iterative design process with Dynamo and Civil 3D

We are now able to rebuild a surface from our landform and grade from the edge to the existing surface. To balance the Cut/Fill can consider the inside as flat (balanced between upright and inverted cones).
Using grading tools would allow us to find the Cut/Fill Volume in Civil 3D automatically. On the other hand, grading with corridors from the edge feature line allows better control of the interaction between our edge and the existing terrain. With some iterations, we can find the equilibrium and then move the circle accordingly and with a rerun in Dynamo the cones move to the new height so we visually control our design.

**Outlook**

Automate the Cut/Fill calculation by Dynamo and then using Refinery to move of the feature line to a list of sample point (works already), tilting and rotating the ground circle on position (works too) to minimize to Cut and Fill.