Dynamo List Masterclass

Andrew Heumann
Design Technology Specialist, Woods Bagot

Join the conversation #AULondon
Introduction
Introduction

Abu Dhabi
Adelaide
Auckland
Beijing
Brisbane
Canberra
Doha
Dubai
Hong Kong
Kuala Lumpur
London
Melbourne
New York
Perth
San Francisco
Shanghai
Singapore
Sydney
Outline

- **List Basics:** default behaviors, list matching, replication
- **Your List Toolkit:** List Lacing, List.Map, List@Level, DynamoScript
- **Practical Examples:** Real-world problem solving
- **List Deep Dive:** The Isovist Problem
List Basics
List Basics

- Kinds of Lists:
List Basics

- Index-Access:
Types of functions:

- **One to One**: Creates a straight line between two input points.

- **One to Many**: Returns points spaced equally along the curve length based on the input number of divisions.

- **Many to One**: Make PolyCurve by connecting points. Set the 'connectLastToFirst' input to true to close the PolyCurve.
Matching/Replication

- What happens if the rank of the input data doesn’t match the rank of the function’s inputs?
## Basic Replication

<table>
<thead>
<tr>
<th>function</th>
<th>input structure (rank):</th>
<th>output structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-many</td>
<td>i</td>
<td>[]</td>
</tr>
<tr>
<td>one-to-one</td>
<td>i</td>
<td>[]</td>
</tr>
<tr>
<td>many-to-one</td>
<td>X</td>
<td>i</td>
</tr>
</tbody>
</table>
Detailed Replication

- What happens if the rank of the input data doesn’t match the rank of the function’s inputs?
  - Step 1: Calculate “Rank Delta” = \( d_k \) for all inputs.
  - Step 2: If two or more of the inputs have a \( d_k \) that’s greater than 0, perform “Zip replication” on all arguments on those positions, and then reduce \( d_k \) by one. Repeat until one or no parameters is \( d_k > 0 \)
  - Step 3: If there’s a parameter with \( d_k > 0 \), perform Cartesian Replication on that argument and reduce \( d_k \) by 1 - repeat until no more parameters have \( d_k > 0 \).
Detailed Replication

\[ V = \frac{1}{3} \pi r^2 \cdot h \]

\[ y = ax^2 + bx + c \]

\[ (x_1, x_2) = -\frac{b \pm \sqrt{b^2 - 4ac}}{2a} \]
Detailed Replication

**zip replication**

A → 1 → A1
B → 2 → B2
C → 3 → C3
D → 4 → D4

**cartesian replication**

A → 1 → A1
B → C → B1, C1
D → A, B, C, D1
Detailed Replication

- Step 1: Calculate “Rank Delta” = $d_k$ for all inputs.

```
rank delta

rank = 2  rank = 1  rank = 2  rank = 1
Point[]  Point[]  Curve[]  Curve[]
rank = 0  rank = 0  rank = 1  rank = 0
Point      Point      Curve      Curve
rank delta = 2-0 = 2  rank delta = 1-0 = 1  rank delta = 2-1 = 1  rank delta = 1-0 = 1
```
Detailed Replication

- Step 2: If two or more of the inputs have a \( d_k \) that’s greater than 0, perform “Zip replication” on all arguments on those positions, and then reduce \( d_k \) by one. Repeat until one or no parameters is \( d_k > 0 \).
Detailed Replication

- Step 3: If there’s a parameter with $dk > 0$, perform Cartesian Replication on that argument and reduce $dk$ by 1 - repeat until no more parameters have $dk > 0$. 
Detailed Replication

Line.ByStartPointEndPoint

0. Initial State

PointsA
RANK=2

PointsB
RANK=1

RANK=(0,0)

dk = (2,1)
Detailed Replication

Line.ByStartPointEndPoint

0. Initial State

PointsA RANK=2

PointsB RANK=1

1. Zip

RANK=0

PointsA

PointsB(0)

RANK=1

PointsA

PointsB(1)

PointsB(2)

PointsB(3)

dk = (2,1)

dk = (1,0)
Detailed Replication

Line.ByStartPointEndPoint

0. Initial State

PointsA
RANK=2

PointsB
RANK=1

1. Zip

RANK=0
PointsA

RANK=1
PointsA

RANK=2
PointsA

2. Cartesian

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=0
PointsB

RANK=1
PointsA

RANK=2
PointsA

RANK=3
PointsA

RANK=(0,0)

rk = (2,1)

dk = (1,0)

dk = (0,0)
Detailed Replication

Line.ByStartPointEndPoint

0. Initial State

1. Zip

2. Cartesian

3. Execute

PointsA
RANK=2

PointsB
RANK=1

dk = (2,1)

dk = (1,0)

dk = (0,0)
The Many Methods

- List Lacing
- List.Map
  - Custom Nodes
- List@Level
- DynamoScript Techniques:
  - Replication Guides
  - Custom Functions
  - List@Level in DynamoScript
List Lacing

- Lacing determines how Lists of different lengths match up with each other.
- Three methods:
  - Shortest List
  - Longest List
  - Cross Reference
- Lacing is applied at the very end, after list replication.
List.Map

- Less necessary than it used to be, thank goodness
- A node with an empty input passes \texttt{itself} as a function to its outputs, and List.Map can call that function over a list.
- List.map repeats a given function for every item in the OUTERMOST list of input data.
- In order to apply a composite function made up of multiple nodes, you often have to make it a custom node.
  - If you do this, make sure to strongly type your inputs!
List.Map

PointsA

List.GetItemAtIndex

List.Map

empty

List.GetItemAtIndex

f()
List@Level

- Allows you to select the level on which you want your function to operate - and "replicates" over the rest.
- "Keep List Structure" will keep results organized by the designated input’s data structure
- Eliminates 80-90% of need for List.Map (IMO)
Node to Code is your friend!

Replication guides support everything lacing does, and then some.

- \( F(a,b) = \) shortest list lacing
- \( F(a<1L>,b<1L>) = \) longest list lacing
- \( F(a<1>,b<2>) = \) Cross reference lacing

Building custom function definitions to operate on single items makes it easy to apply those functions to lists later on - normal replication behavior applies.

@-N after an input = “list at level” N - use @@-N for L@L w/ preserve. You must use a lacing “channel” for this to work!
Practical Examples
Get Levels by Name List

List.Map Approach

Need to retrieve the level elements that correspond to a particular list of level names, maintaining their order.

List@Level Approach

Need to retrieve the level elements that correspond to a particular list of level names, maintaining their order.
Change Curtain Panel Type at Ends

Select all walls

Filter wall elements for Tower Curtainwall type

Get curtain panels from wall via curtain grid

Glazed Operable System Panel Glimpse Grid: Family Type
List Deep Dive: The Isovist Problem
Other approaches:
Other approaches:

Credit: Vikram Subbaiah
- Other approaches:

   - Obstructions

   - Centerpoint and perimeter

Credit: Zach Kron
Other approaches:

Credit: Vikram Subbaiah
Final Thoughts

- Copy-Paste is a sin
- Get in the habit of “reading” the structure of your data and the nodes you use
- List structures are information-rich - preserve them where you can
- Avoid List.Map in favor of List@Level
- Design your custom nodes for the simplest structure possible - let Dynamo handle lists + replication
Additional Resources

- Introducing List@Level

- Replication Details

- Dynamo Script Language Guide

- Isovist Challenge Discussion
  https://forum.dynamobim.com/t/isovist-challenge/11510