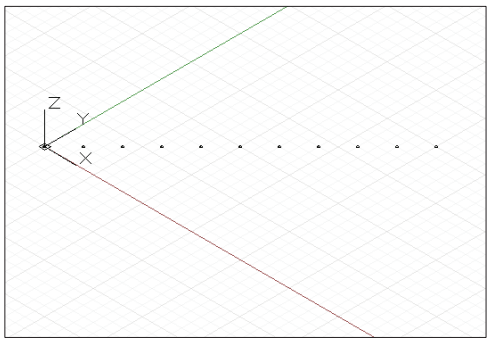
13: Replication Guides

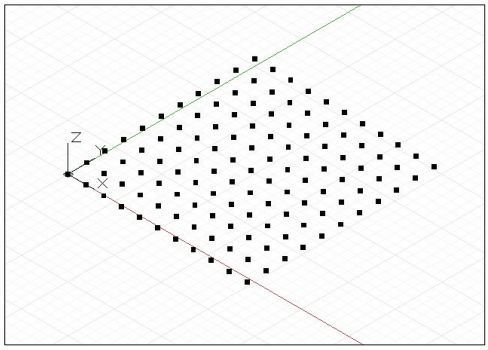
The Dynamo language was created as a domain-specific tool for architects, designers and engineers, and as such has several language features specifically tailored for these disciplines. A common element in these disciplines is the prevalence of objects arrayed repetitive grids, from brick walls and tile floors to façade paneling and column grids. While range expressions offer a convenient means of generating one dimensional collections of elements, replication guides offer a convenient means of generating two and three dimensional collections.

Replication guides take two or three one-dimensional collections, and pair the elements together to generate one, two- or three-dimensional collection. Replication guides are indicated by placing the symbols `<1>`, `<2>`, or `<3>` after a two or three collections on a single line of code. For example, we can use range expressions to generate two one-dimensional collections, and use these collections to generate a collection of points:




```
x_vals = 0..10;
y_vals = 0..10;

p = Point.ByCoordinates(x_vals, y_vals, 0);
```

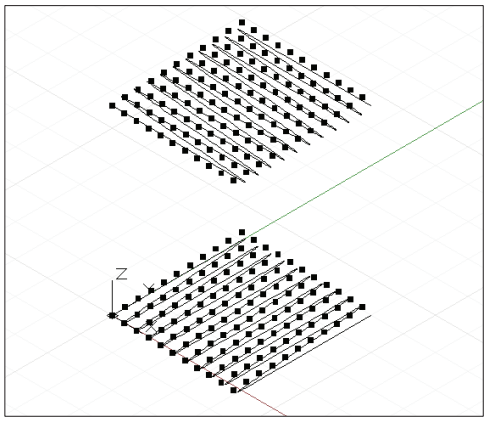
In this example, the first element of `x_vals` is paired with the first element of `y_vals`, the second with the second, and so on for the entire length of the collection. This generates points with values (0, 0, 0), (1, 1, 0), (2, 2, 0), (3, 3, 0), etc…. If we apply a replication guide to this same line of code, we can have Dynamo generate a two-dimensional grid from the two one dimensional collections:




```
x_vals = 0..10;
y_vals = 0..10;

// apply replication guides to the two collections
p = Point.ByCoordinates(x_vals<1>, y_vals<2>, 0);
```

By applying replication guides to `x_vals` and `y_vals`, Dynamo generates every possible combination of values between the two collections, first pairing the 1st element `x_vals` with all the elements in `y_vals`, then pairing the 2nd element of `x_vals` with all the elements in `y_vals`, and so on for every element `of x_vals`.

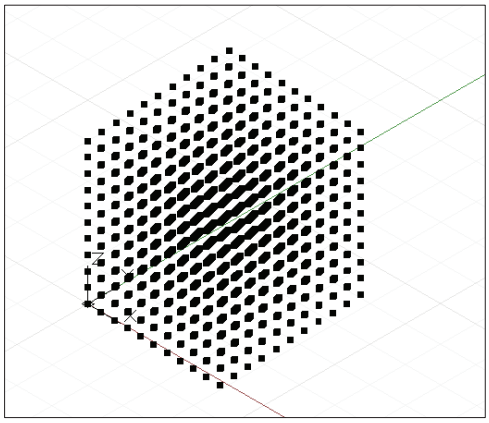
The order of the replication guide numbers (`<1>`, `<2>`, and/or `<3>`) determines the order of the underlying collection. In the following example, the same two one-dimensional collections are used to form two two-dimensional collections, though with the order of `<1>` and `<2>` swapped.




```
x_vals = 0..10;
y_vals = 0..10;

p1 = Point.ByCoordinates(x_vals<1>, y_vals<2>, 0);

// apply the replication guides with a swapped order
// and set the points 14 units higher
p2 = Point.ByCoordinates(x_vals<2>, y_vals<1>, 14);

curve1 = NurbsCurve.ByPoints(Flatten(p1));
curve2 = NurbsCurve.ByPoints(Flatten(p2));
```

`curve1` and `curve2` trace out the generated order of elements in both arrays; notice that they are rotated 90 degrees to each other. p1 was created by extracting elements of `x_vals` and pairing them with `y_vals`, while p2 was created by extracting elements of `y_vals` and pairing them with `x_vals`.

Replication guides also work in three dimensions, by pairing a third collection with a third replication symbol, `<3>`.



```
x_vals = 0..10;
y_vals = 0..10;
z_vals = 0..10;

// generate a 3D matrix of points
p = Point.ByCoordinates(x_vals<1>,y_vals<2>,z_vals<3>);
```

This generates every possible combination of values from combining the elements from `x_vals`, `y_vals`, and `z_vals`.