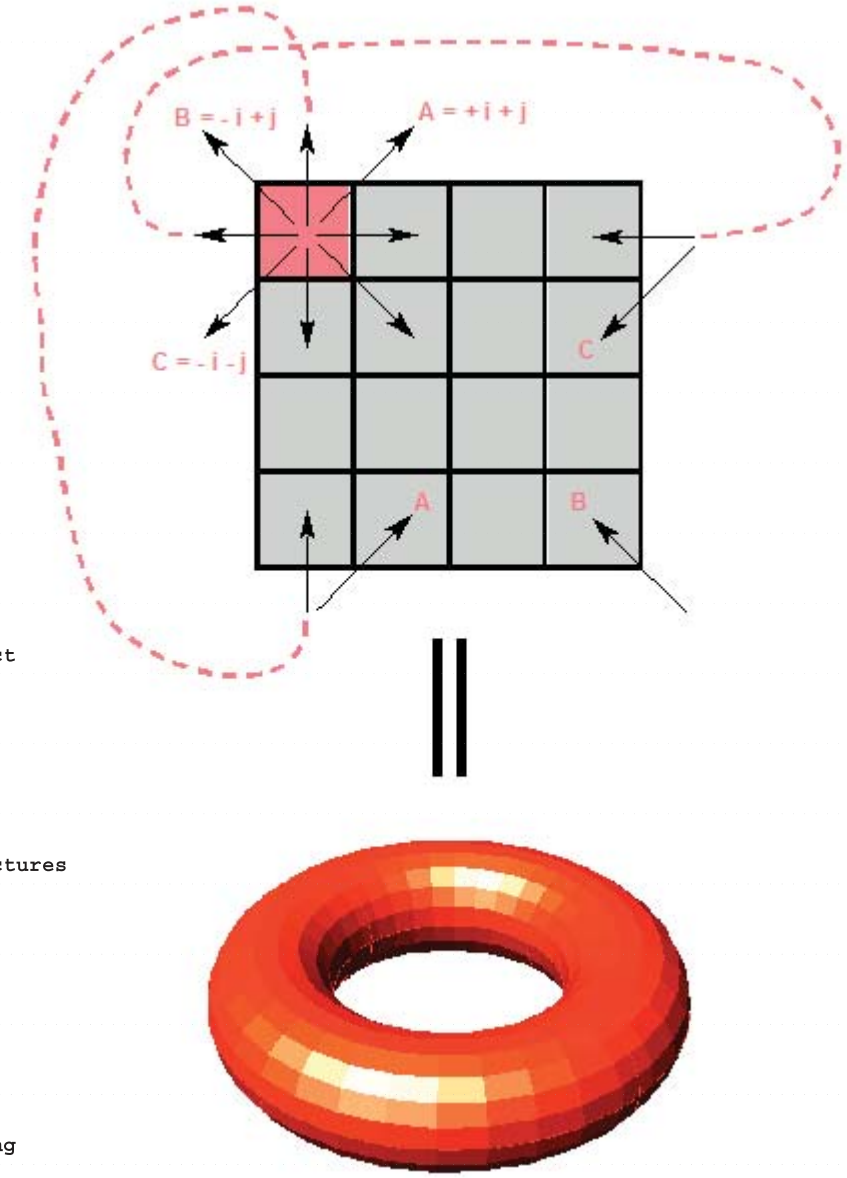


pseudo infinity: toroidal univers

Pseudo Infinity

On last weeks hand-out, I was mentioning the three different methods to describe the edge of the universe for the CA. For this week the task was to alter the state of the universe from the limited to the pseudo infinite.
Last week's code scripted the universe to be a ring of dead cells around the edge, so that the cells would not 'fall of the edge' and generate a *Run Time Error* called *Subscript out of Range*, meaning that a loop addressed an array index that doesn't exist.

The pseudo infinite universe metaphorically stitches the right edge of the CA to its left, the top to the bottom and the deep to the shallow (see figure above). In two dimensions we can imagine a torus or donut; whereas in three dimesions we can't quite imagine the resulting morphology.

Through this operation we will be able to use all cells of the CA again, as opposed to the previous method whereby we could address only the cells removed by one index position from the edge. Therefore, you have to adjust all array dimensions and loop indeces to the same range. Instead of:

```
For i = 1 To row -1
        For j = 1 To col-1
                For l = 1 To level-1
```

as before, you have to loop the complete array:

```
For i = 0 To row
        For j = 0 To col
                For l = 0 To level
```

Having dimensioned all arrays and loops to the same range,

```
ReDim grid(row, col, level) As cell
ReDim limbo(row, col, level) As Integer
```

we have to tell the program in the function *counthem()* where the edge occurs and where to search the next neighbour if the edge of the array has been reached. The extract below shows the scripting method to adjust to the wrapped edge. These lines need to be placed between the loops in *counthem()*, which look in the immediate topological neighbours for their state:

```
ii = i
If (i > row) Then ii = 0
If (i < 0) Then ii = row

jj = j
If (j > col) Then jj = 0
If (j < 0) Then jj = col

ll = l
If (l > level) Then ll = 0
If (l < 0) Then ll = level

neighs = neighs + grid(ii, jj, ll).state
```

The variables ii, jj and ll are new function internal variables that serve as a substitute for the loop indeces i, j and l. Thus, we don't have to manipulate the indeces directly, which would lead to messing up the loops . You can see from the example above that in general the substitute variables take the value of the loop indeces. Only in the exceptional cases of detecting an index that doesn't exist in the array constituting the CA, will the loop index be replaced with either the highest value of the specific dimension of the array - that is either the maximum row value (1-dim), the maximum column value (2-dim) and/ or the maximum depth value (3-dim) - or with the lowest value of the specific dimension, which is 0 in all three dimensions.

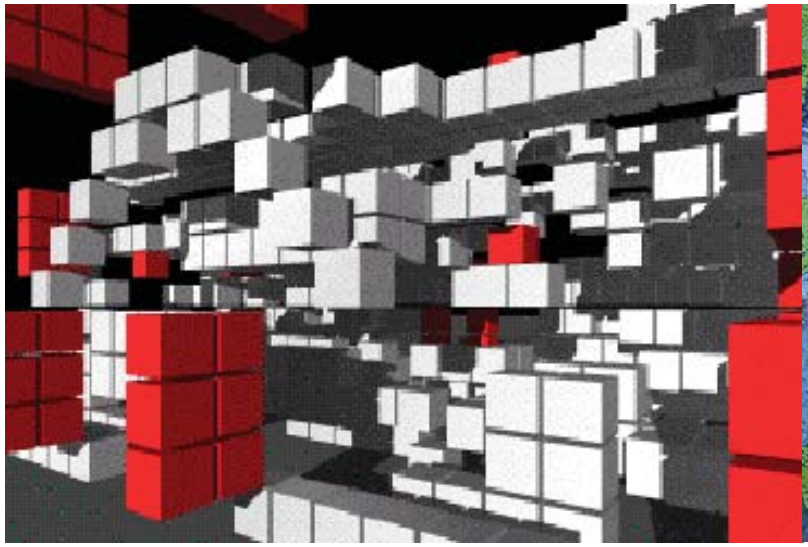
If the loop index grows bigger than the array dimension, i.e. i > row, the substitute variable will be set to the minimum and thus considers the neighbour cell at the bottom/ left edge. If the loop index falls below the array dimension, i.e. j < col, the substitute variable will be set to the maximum dimension and thus considers the neighbour cell at the top/ right edge.

Exit

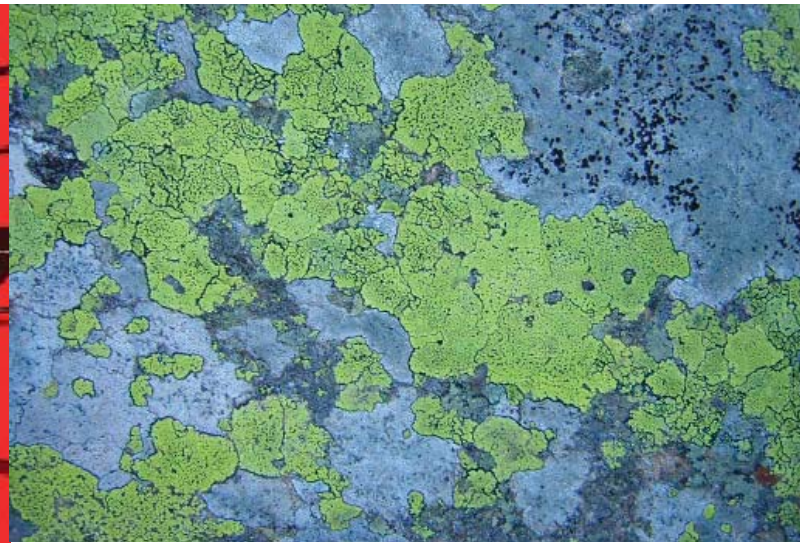
Exit is a function that lets you 'exit' a sub-procedure, function, or any control statement, like a loop or a conditional. You will have to specify after the call to *Exit* what you want to get out of. So, in our example of the CA we want to exit an entire sub-procedure since we are really only looking for one element in the array. If we found it we don't want to look through the rest:

```
Sub newstate(...)
   For i = 1 To row
      For j = 1 To col
         If (id = grid(i, j).id) Then
            ...
            Exit Sub
         End If
      Next j
   Next i
End Sub
```

We could of course also exit either the For Loop or the If statement; but then we would have to specify that accordingly after the 'Exit'!

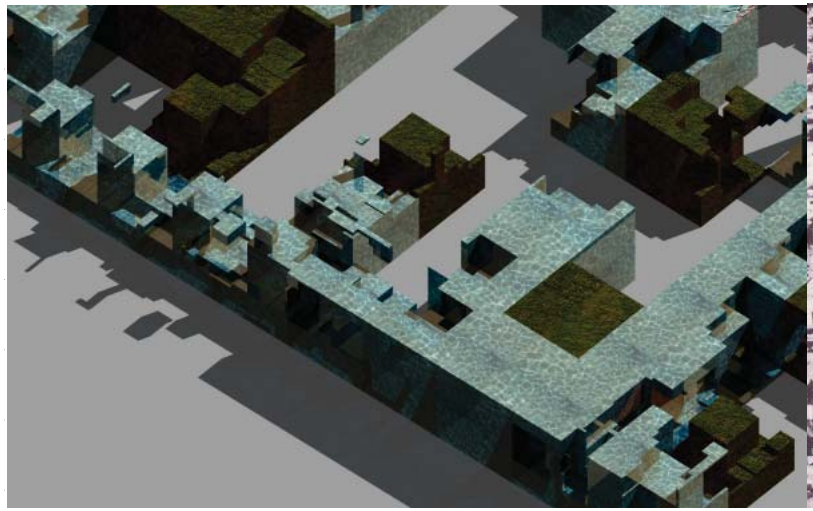


Coates, Healy & Lamb, 1997



moss growth

Derix, 2004



Mexico City sprawl