Carrots and a vole: Simple interactions between plants and animals

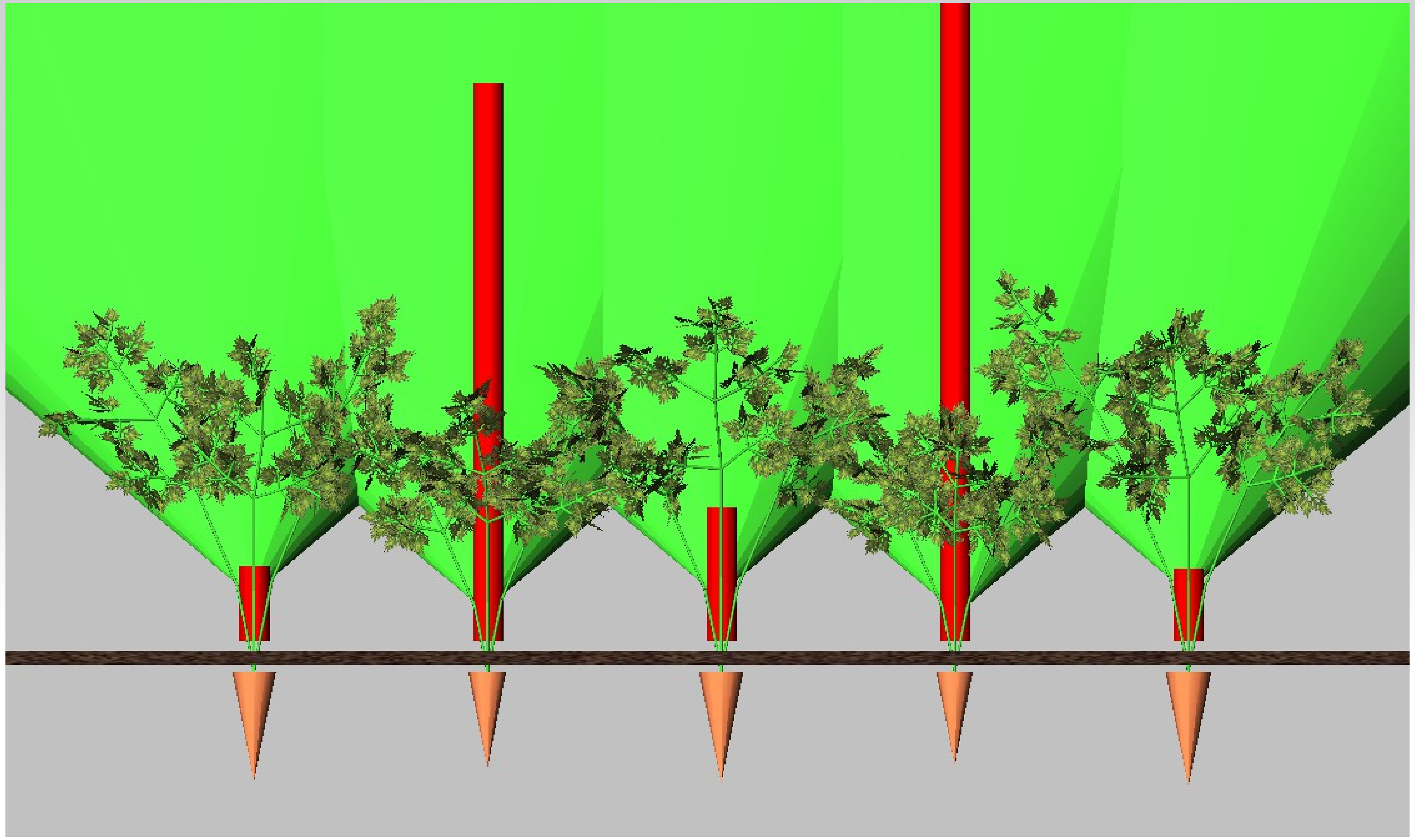
Ole Kniemeyer

Summer School “Modelling of Ecosystems by Tools from Computer Science“
16 – 20 September 2013, Prague

The actors

The carrot model

- Simple light competition model based on amount of obstacles in light cone



- Carbon production in leaves according to the light interception
- Carbon flow towards root leads to internode elongation and root growth

The vole model

The vole has two states:

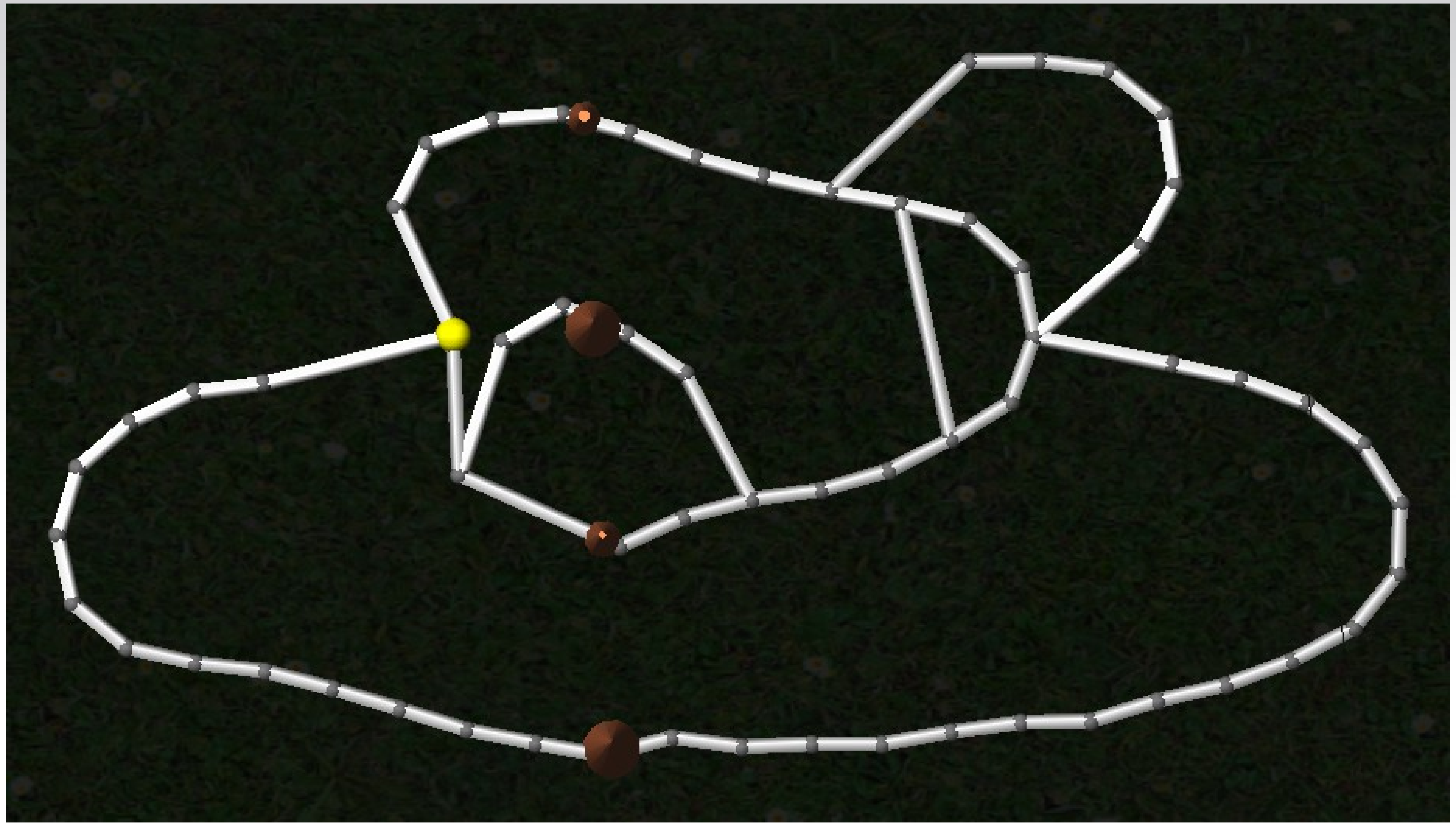
- Non-digging: Run along existing tunnels, gnaw at carrot roots if possible
- Digging: Dig a new tunnel and try to reach a carrot or an existing tunnel

Non-digging rules:

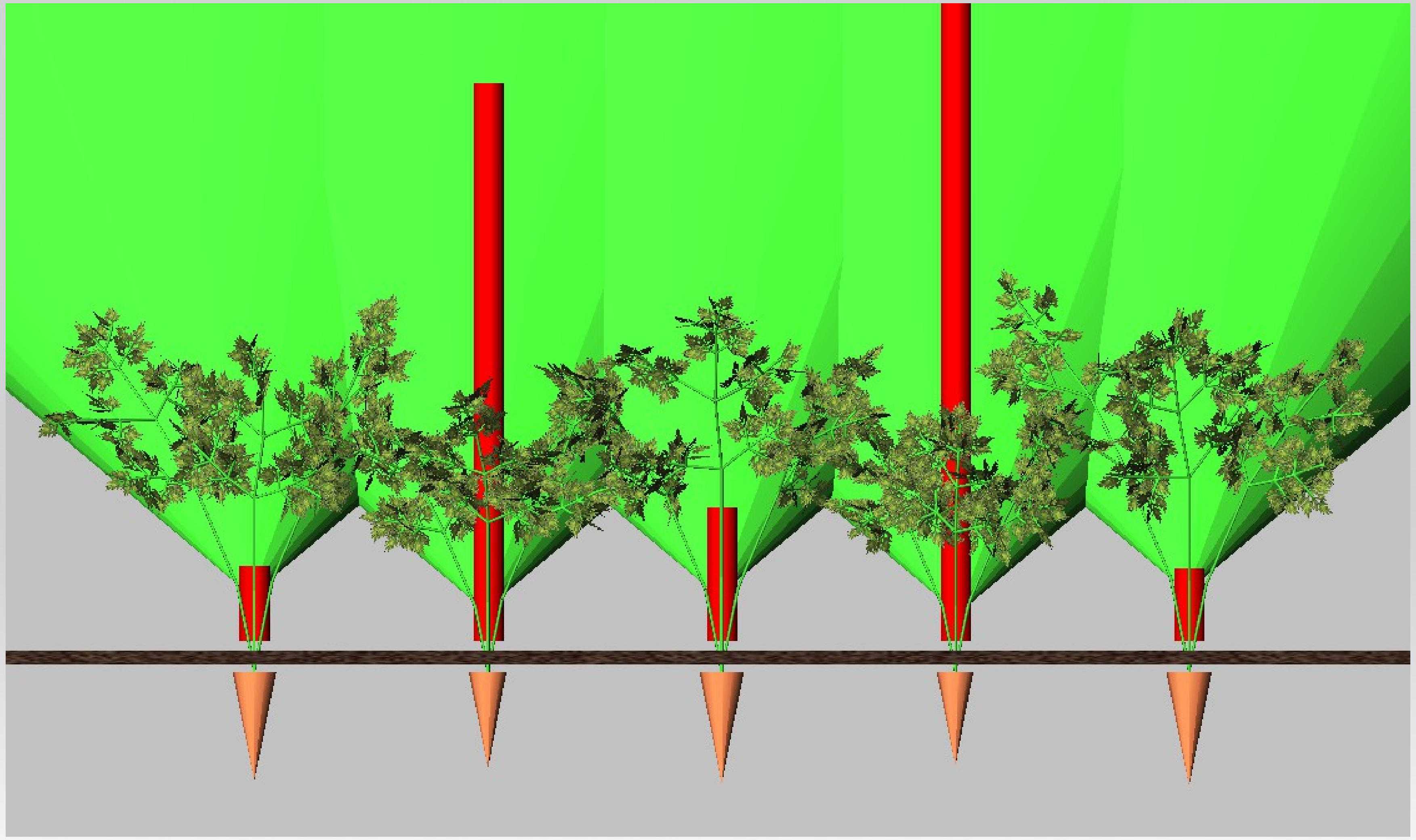
- Reinforce a scent-mark at current tunnel node
- If a root is nearby, gnaw at it
- Move to a connected tunnel node, favouring those with less intensive scent-marks
- Or from time to time, if no root is nearby, become a digging vole

Digging rules:

- Find closest nearby tunnel node in front of vole
- Dig to this node and become a non-digging vole
- Or, if no node was found, find the closest carrot and dig in its direction



Implementation of plant model

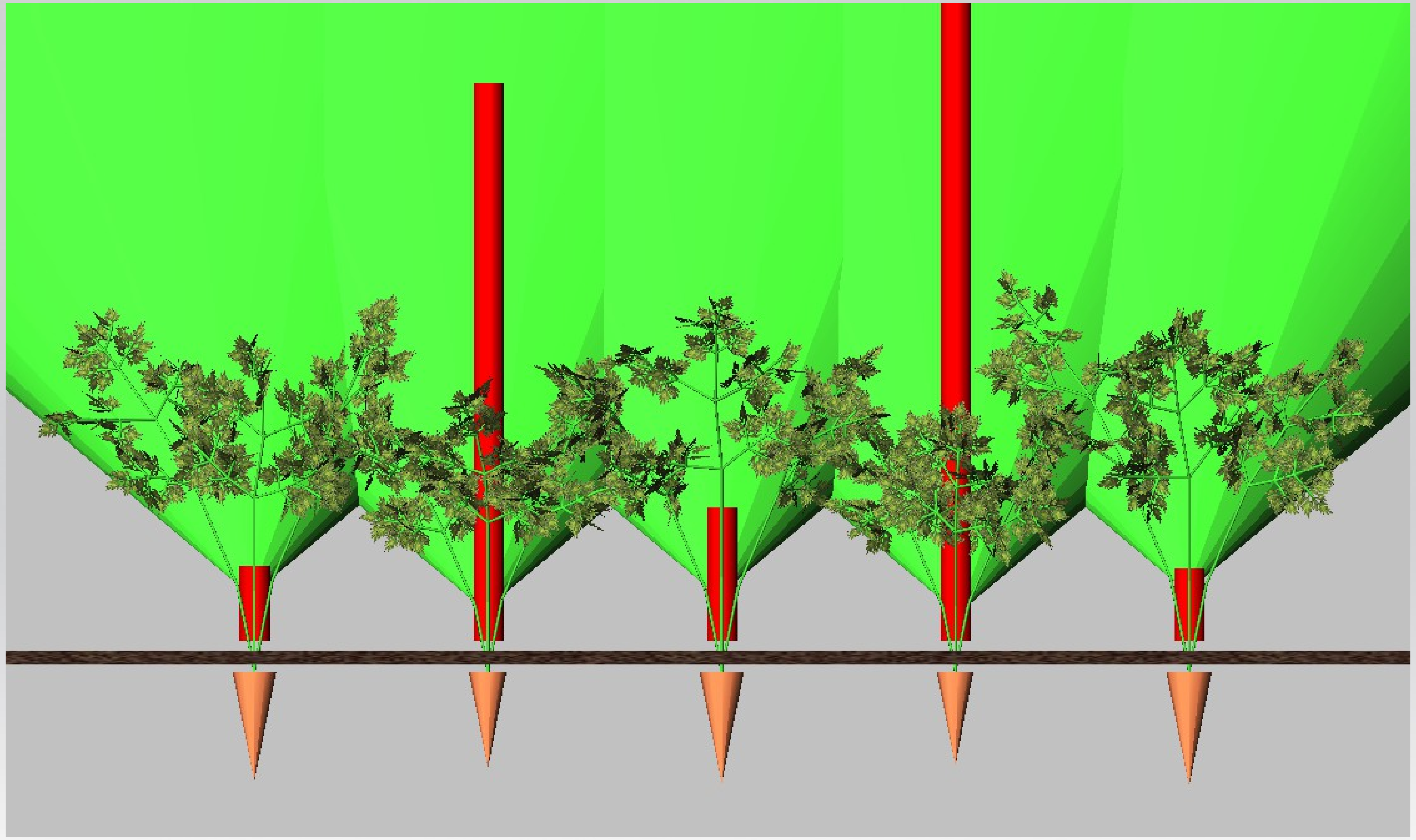


Computation of shading:

```
c:Carrot ::>
{
  Tuple3d m = mean(location((* c (-->)* Leaf *)));        // Compute the centre of c's leaves
  m.z *= 0.3;                                              // and reduce it along the z-axis.

  c[shadow] := sum
    ((* d:Carrot, ((d != c) && (distance(c, d) < 3)),     // For every neighbouring d
        d (-->)* f:F,                                      // find all internode descendants f
        (f in cone(m, HEAD, 50))                           // within a light cone around m
     *)[length]);                                          // and sum up their length.
}
```

Implementation of plant model



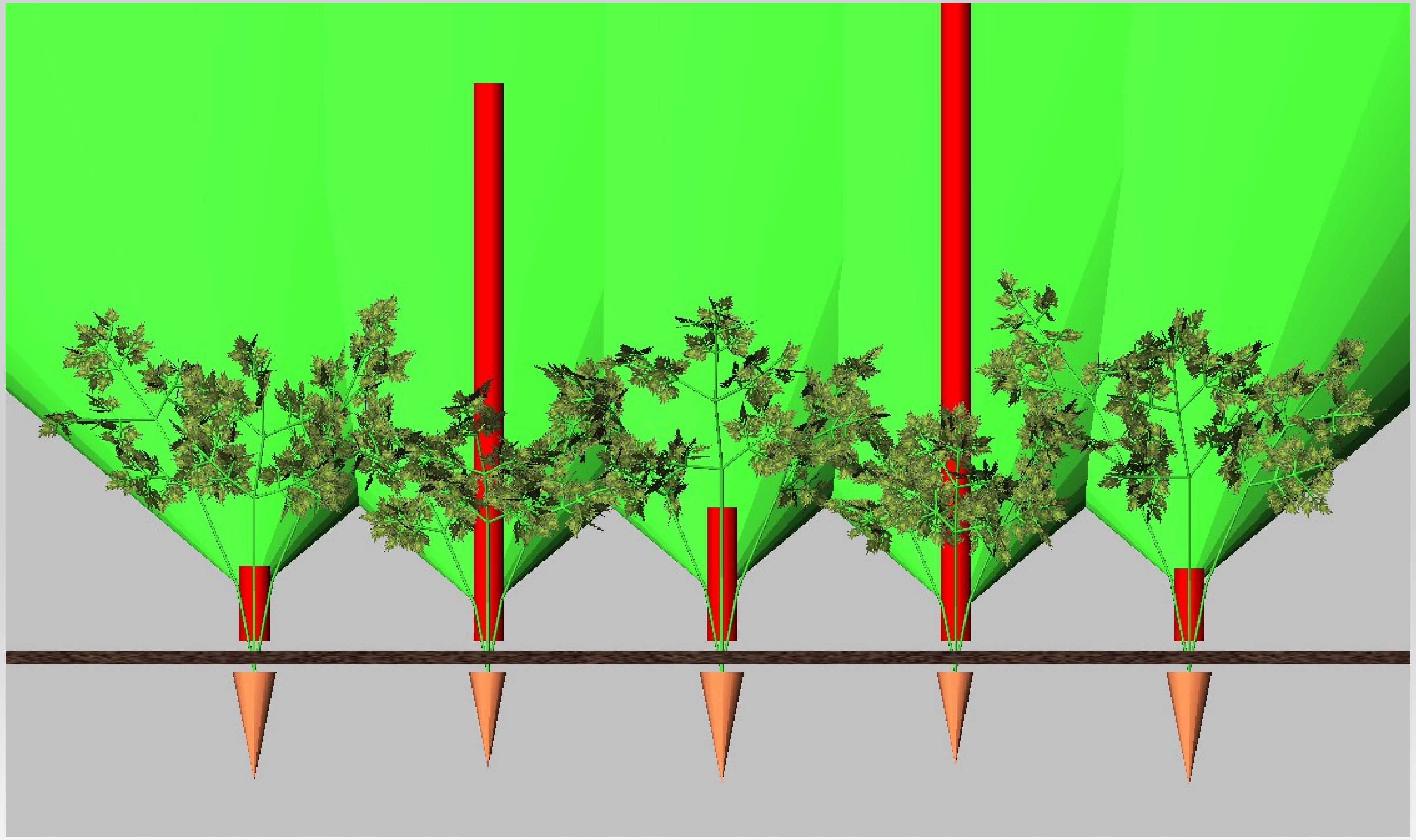
Carbon production:

```
x:Leaf ==>>
  if (probability(0.95)) (break)
                                                                 // With a probability of 5%,
                                                                 // append a Carbon particle to the leaf.
  x [Carbon(0.03 / (1+0.8*first((* x -ancestor-> Carrot *)[shadow])))];
```

Carbon transport:

```
n:Node [c:Carbon] -ancestor-> a:F ==>> n, a [c];          // Move c downwards to next F
```

Implementation of plant model



Carbon allocation:

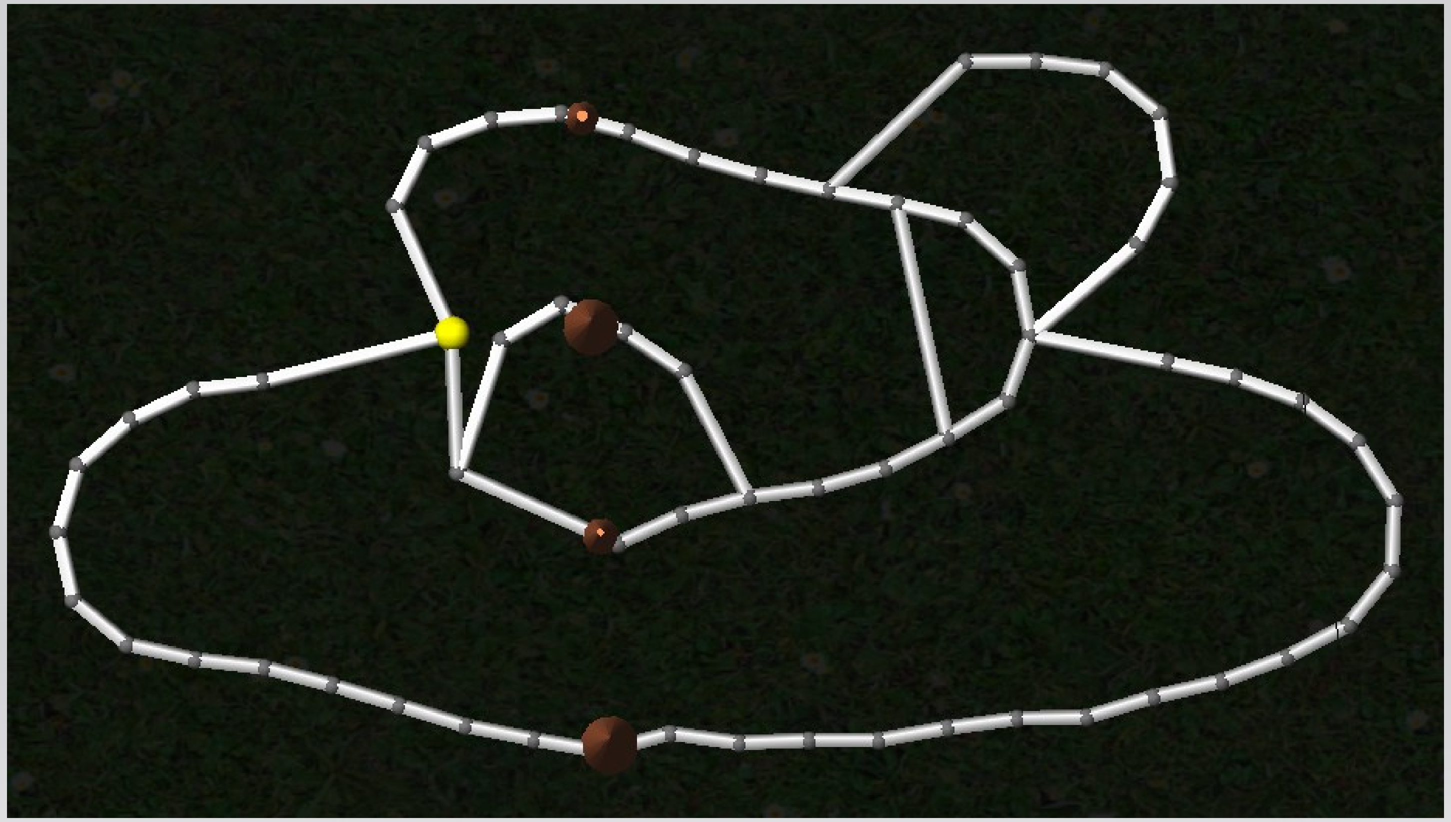
```
n:F [c:Carbon] ==>>
  {Carrot m = first((* n <-- Carrot *));}
  if (m != null) (
    n
    {m[size] :+= 0.014 * c[value];}
  ) else {
    float v = 0.2 * c[value];
    c[value] :-= v;
    n[length] :+= v;
    break;
  };
```

```
// If n is the immediate successor of a carrot m
// keep n in the graph, but delete c
// and let the carrot grow.

// Else allocate an amount v of carbon

// and elongate the internode n.
// Do not apply any structural changes to the graph.
```

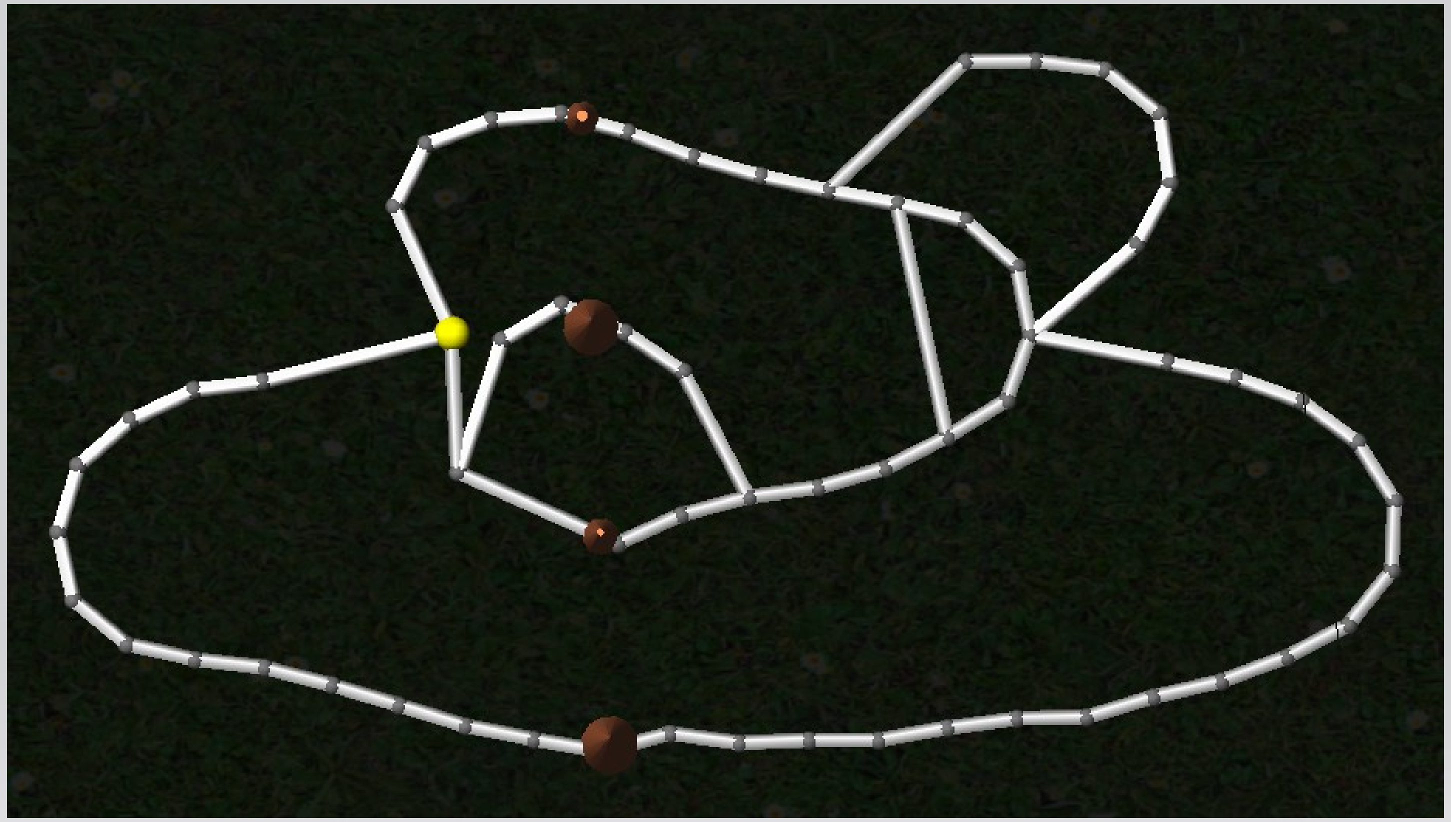
Implementation of vole model



Non-digging vole:

```
t:TunnelNode [v:Vole] ==>>
  {
    t[scent] += 3;                                          // Re-inforce scent mark.
    Carrot c = first((* x:Carrot, (distance(v, x) < 0.5) *));
    if (c != null)                                          // There is a close carrot,
      c[gnawed] :*= 0.9;                                    // so gnaw off a bit.
  }
  if ((c != null) || probability(0.9))                      // At a carrot or with high probability
  (                                                         // choose a neighbouring node n of t
    {                                                       // favouring nodes with less intensive scent marks.
      TunnelNode next = selectRandomly((* t -tunnel- n:TunnelNode *), Math.exp(-n[scent]));
      if (next == null) next = t;
    }
    t, next [v]                                             // and move to that node.
  ) else (
                                                            // Otherwise, start digging in a random direction.
    t RU((irandom(0,1)*2 - 1) * random(60, 120)) M(1) [DiggingVole]
  );
```

Implementation of vole model



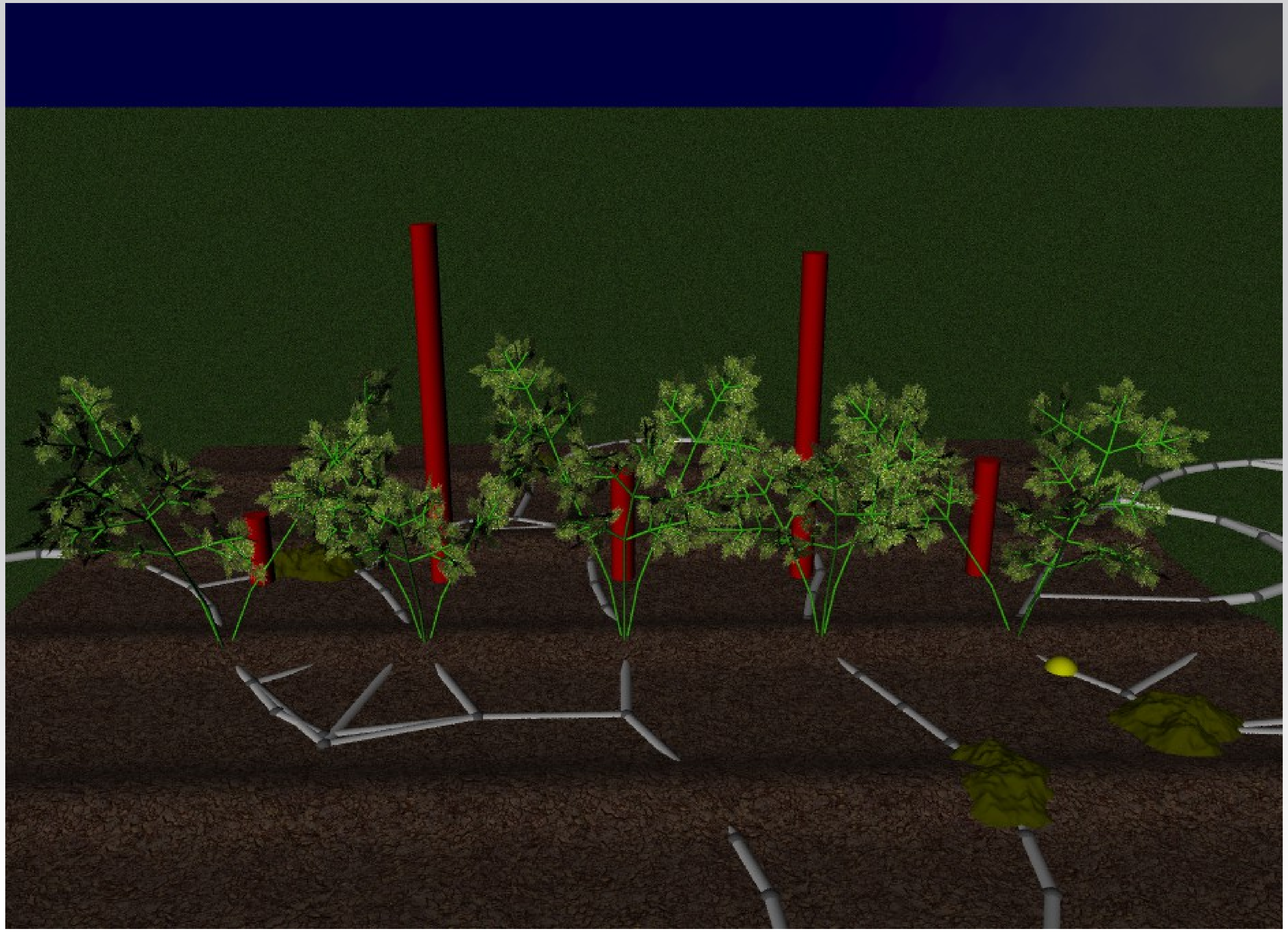
Digging vole:

```
v:DiggingVole -ancestor-> t:TunnelNode ==>>
                                                                  // Of all nodes z that are close to v
  {TunnelNode x = selectWhereMin((* z:TunnelNode, ((z != t) && (distance(z, v) < 1)
                                                                  // and lie within the vole's forward cone,
                                                           && (z in cone(v, false, 60))) *),
                                                                  // select that node x with minimal distance to v.
                                 distance(z, v));}
  if (x != null) (                                                // If such a node x exists,
    t -tunnel-> x, x [Vole]                                       // dig a tunnel to it and become a non-digging vole.
  ) else (
    n:TunnelNode(1) [<-tunnel- t]                                 // Otherwise, create a new node n with tunnel from t,
    moveIncoming(v, n, -1)                                        // embed n at the graph location of v,
                                                                  // select the closest carrot c
    {Carrot c = selectWhereMin((* d:Carrot *), distance(d, v));}
    tropism(v, c, Math.exp(-0.1*distance(c, v)))                  // and turn towards c.
    RU(random(-10,10)) M(0.5) [v]
  );
```

Putting it all together

Sub-models have different time resolutions:

```
public void model()
{
  if ((globalStep % 5) == 0)
  {
    carrots();
  }
  vole();
  globalStep++;
}
```



Thanks for your attention!