6.837
Computer Graphics

Final Project Report

Cellular Modelling

Jaroslav Blagojevic, Xavier A. Guajardo

Abstract:

When modelling organic objects, we desire natural look and variety (e.g. a forest where no two trees are the same). Many methods have been proposed, from sparse approximations in forms of fractals and L-systems to the sophisticated parametric objects and the simulations of Nature itself. We propose an alternative--to create a cell of a pyramidal shape that multiplies over time based on the "DNA" within it, that is replicated in each produced cell. When a desired spread is achieved, it can be encapsulated under a name, then used as a daughter cell. The cell can grow into almost every possible solid, thus giving user the full control, yet still being able to account for variability. We also achieve an observable real-time growth.

Applications:

Applications of the Universal Stem Cell could be:

1) modelling natural objects (trees, leaves, flowers, creatures)
2) simulating natural objects (e.g. a plant that grows towards sunlight)
3) reconstructing objects from instrumented photographs, (e.g. an object that grows to best fit an image)
4) having fun watching the thing grow

API:

The environment (a.k.a. Simulator) includes an OpenGL hardware accelerated based visualization system; the user can interact via a console that enables it to add and remove objects and objects inside objects and change their variables. The user can save a form of an object into a text file, or load it from the file; alternatively, the file can be edited inside a text editor.

e.g. inside the environment

```
~/sim
   new object Plant
   cd Plant
   new stemcell Seed
   cd Seed
   spread Tentacle
   new daughter Petal
   cd Petal
   new stemcell Seed
   cd Seed
   spread Radial
```

... etc.

by typing "alive", "freeze" and "reset", the user can watch the growth of the cell in real-time. The animation could be exported into a specific format, but we do not deal with it for now.

or in the file e.g.

~/Forms/Flower1:

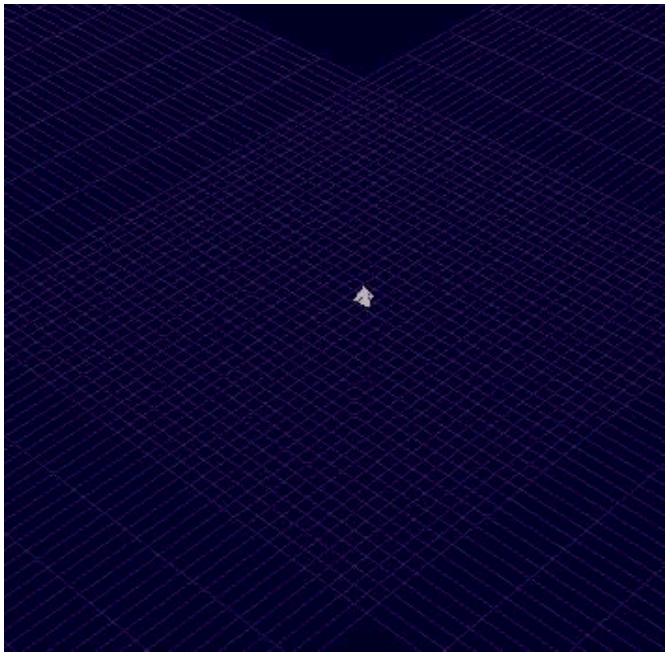
```
stemcell Flower1
{
   spread Tentacle
   radlife 6 6 0 0.2
   daughter Thorn
    {
       pos Random
       stemcell Seed { ... }
    }
   daughter Petal
    {
       pos Terminal
       petal Seed
    }
}
```

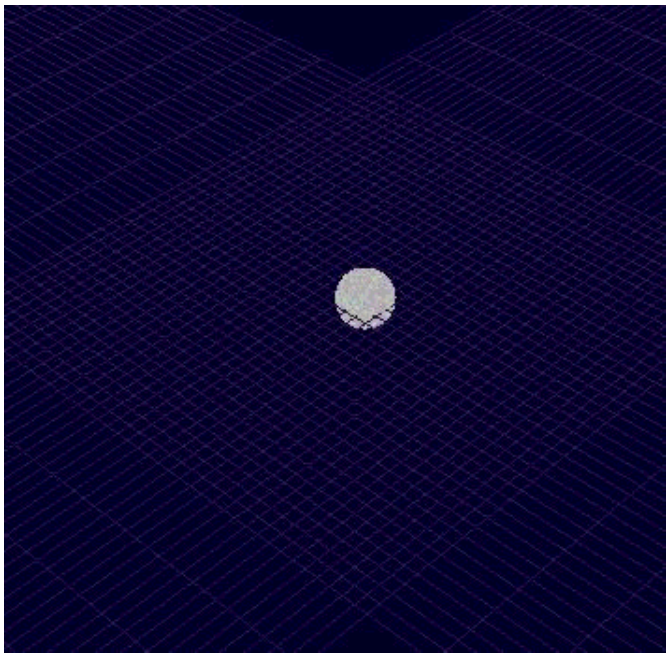
The Simulator is an environment in which objects communicate via messages (e.g. TimeTick), can be dynamically alived and destroyed, and ones created inside others; thus we end with a decent C++ code, that would otherwise be much longer and cumbersome to implement. The environment works on any UNIX or Win32 system.

The final model can be exported in Inventor IV format. We stay away from utilizing commercial software to do the rendering, as it is very non-academic, counterproductive and does not teach anything new.

Stem Cell Definition:

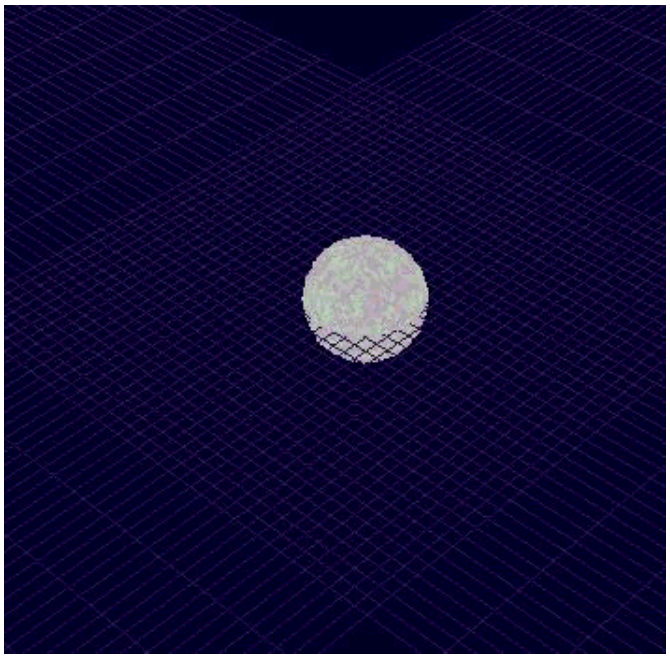
Every object grows from a single cell:





Time =

0 some time after



some time after some time after end product

Figure 1. A simple radial spread; note the roughness and color variability

We experimented with a number of parameters, ultimately ending in the set of the most important ones. To achieve the ability to grow into every form, the cell must be able to grow

Radially

```
spread = Radial
```

Radial spreads are basic, first-order spreads. The parameters are

```
radlife   = <x> <y> <z> <variability>
roughness = <roughness> <variability>
```

Where parameters after '=' are double-precision floating point values. Roughness adds to "spikiness" of the spread. Zero roughness produces smooth forms.

Roughness might also be enhanced to include x y z coordinates.

Directionally

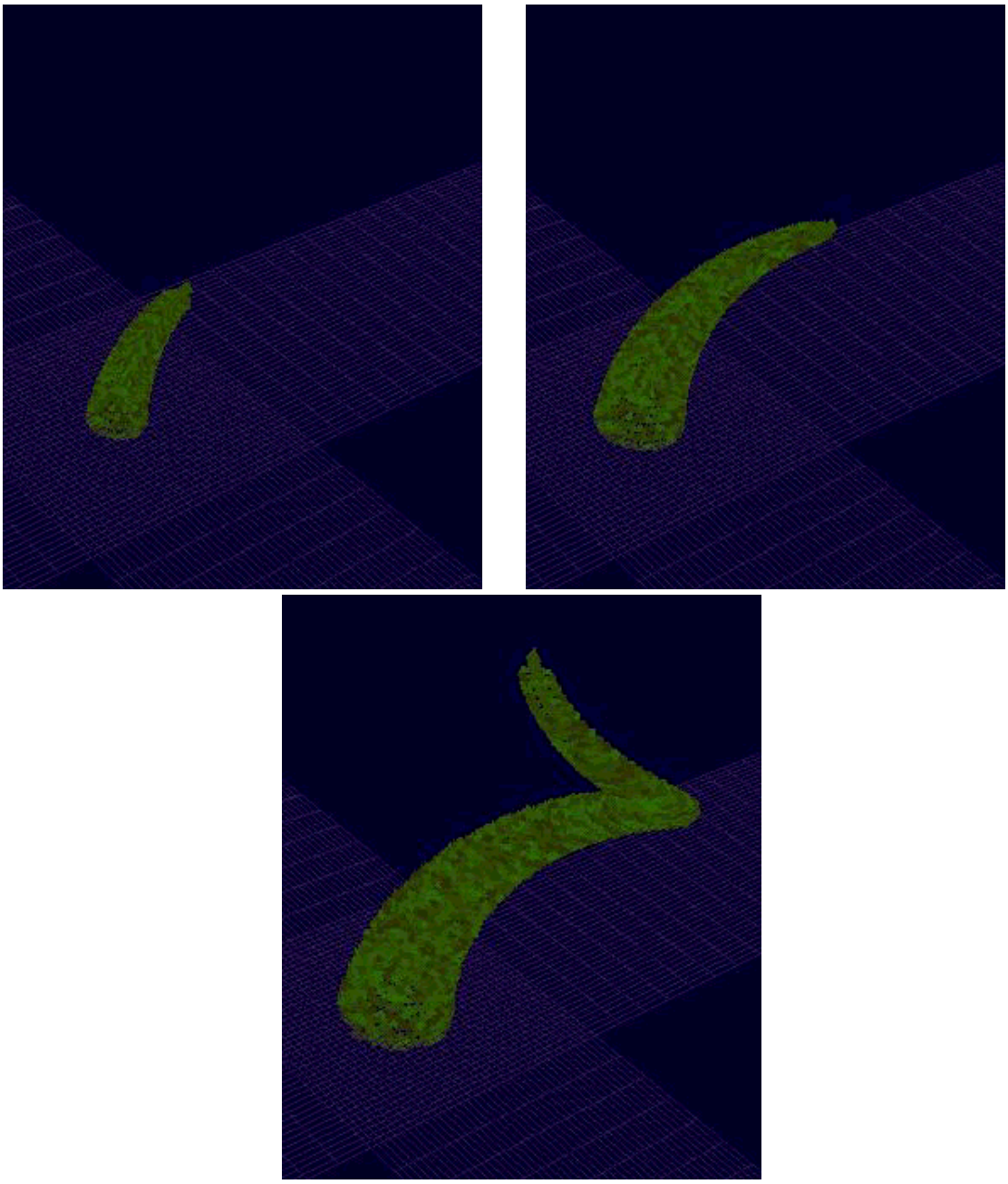


Figure 2. A tentacle spread

```
spread = Tentacle, Sheet, Block
```

Directional spreads are second-order spreads. At each tick of time, the spread progresses towards the direction and creates a daughter stemcell which spread is:

for Tentacle - Radial, with RadLife.z = 0
for Sheet - Linear
for Block - Sheet

The parameters are:

```
dirlife   = <life> <variability>
direction = <x> <y> <z> <variability>
normal    = <x> <y> <z> <variability>
```

The parameters Direction and Normal establish a coordinate system for a specific cell.

```
twist         = <x> <y> <z> <variability>
twist_decay   = <x> <y> <z> <variability>
radlife_decay = <x> <y> <z> <variability>
```

One can experiment with these; the project time is too short to try every variation of a directional spread.

Daughter spreads

A cell can contain a number of daughter cells inside it, that can spawn based on the chance. A branch could spawn a branch that could spawn a branch, and so we get a tree; a branch can spawn a petal and the thorns, and we get a flower. The parameters are:

```
pos    = Random, Terminal, Radial, Radial2D
chance = <chance>
```

The position of the daughter spread and the odds of happening.

```
amount = <amount>
```

The number of daughters spreading from the position.

```
<type> Seed  { ... }
```

Seed is a stemcell of a particular type, dynamically created inside the cell. <Type> can be a stemcell or any filename in /Forms directory, that contains a previously saved cell. The parameters such as "Direction" in the Seed are interpreted as relative to the mother cell.

e.g.

```
StemCell Branch
  {
    spread Tentacle
    daughter Branchie
     {
       pos Random
       chance 0.02
       Branch Seed {
                      RadLife 3.0 2 0 0.2
```

```
                    }
                }
            }
```

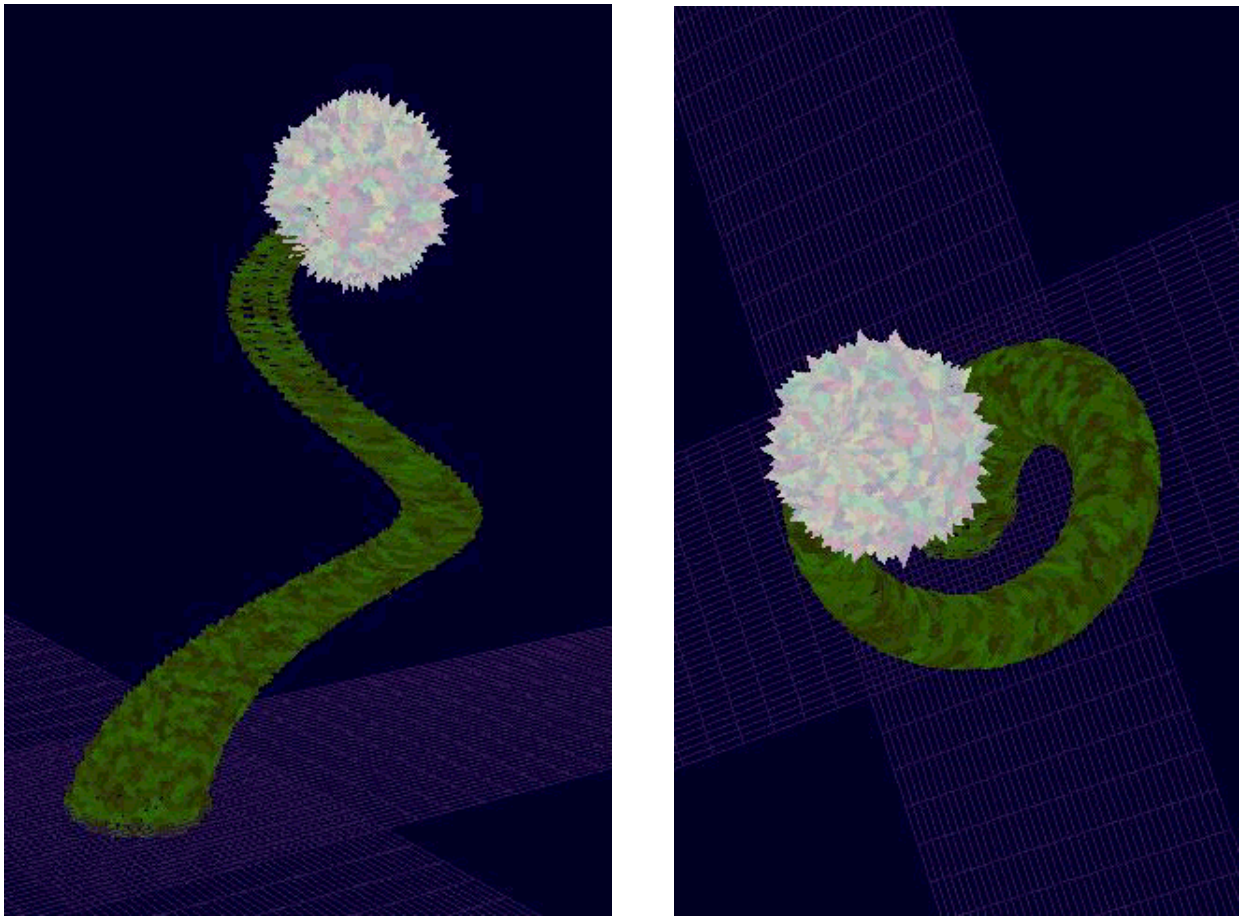


Figure 3. A twisting tentacle spread with a simple daughter

Other parameters

```
color           = <r> <g> <b> <a>
color_variation = <r> <g> <b> <a>
```

Color variation adds to the look. It is currently based on pure chance; the more mature system would use the daughters stemcells to achieve a specific skin.

```
is_wireframe    = T/F
```

DNA transcription:

DNA is transcribed by grouping it into triplets starting at the first start codon. Each daughter is separated by a stop codon. The organism's scope is ended when there are no further start codons in the

string.

Each daughter contains a vector of daughters (linked list) which are its daughters. A daughter has the parameters filled by associating specific codons with specific parameters, for instance, AAC corresponds to the color parameter. Each parameter contains further parameters and these values are retrieve to several mathematical functions. Since there are many involved, they are not described here.

Upon completion of all daughters, the data is saved in a text format which is output into a file that is accepted by the simulation environment. Since the environment can have a degree of randomness even when presented with a given set of static values, it can more closely simulate a natural organism.

Project Evolution:

At the beginning, the cell was supposed to spread such that the produced cells are adjacent to the surfaces of the mother cell. That would create a decent hull and make the object easily exportable into other graphic formats. The cells, however were sometimes overlapping each other, and before we tried to use spatial data structures to check for neighborhoods and eliminate the overlaps, we found out that the fuzziness adds to the desired look. The smooth, polygonal, almost-Bezier-like surfaces are far from natural. Thus we left the idea of smoothing the thing out, and continued with the cells that do not necessarily have adjacent surfaces. After the growth is achieved, we made an algorithm to eliminate the inner surfaces, for the purpose of export and fast rendering. Alternatively, the system could work with a voxel model, that takes more memory, but is much more capable and accurate. The 6.837 class potentiated polygon/texture/approximated lighting models, that are artifacts made to compromise for speed, and achieve an approximate visualization that can work on the currently available hardware. We present the achieved models in flat/wireframe, and let one imagine how it could look in a realistic visualization engine. At the end, we added the ability for the user to specify a random DNA string similar to the real one, that translates into the cell parameters and creates a random form. The idea could also be applied to artificial creativity, if ever such an area becomes popular.

Achievements:

The objective was to develop a pure simulator that could accept as a parameter a string of DNA that would then be simulated in real-time. The team was actually very close to accomplishing this goal but a very significant challenge remained.

The problem is developing the simulation such that the object simulated is wholly dependent only on the DNA string. This has shown to not be the case since that would amount to what can be best described as a continuous system while our environment is best described as being discrete.

An example of what is meant is that all DNA values a specific default value. Every difference must be programmed by hand so attempting to have a totally natural selection would amount to having a very large number of DNA triplets. And even this solution is not complete since it still has a finite number of options.

To put this in perspective, there is one parameter called color that accepts three reals as parameters. It is possible to have the color 1 1 1 & the color .99 .99 .99. But trying to program such a difference is not pragmatic. At best, we would have to have discrete values such as 1, .95, .90, .85, etc. which is not a

very natural option. We are unable to have the proper spectrum of values needed to properly simulate nature.

Attempting to have a natural way of doing this is not really possible with discrete values. There are many other examples of this in our project where it is just not feasible to simulate a proper natural environment.

But achievements that were made was that the simulator is functional and it can accept a DNA string. It is now possible to simulate the growth of an object given its dna pattern and the object will grow to completion.

Lessons Learned:

There were several particular items that became apparent as the project proceed. These were generally items that we had not considered prior to the commencement of the project.

Academically, some of the concepts practiced were the following. We had to experiment with spatial distortion/placement through the exploitation of proper spatial coherence. One of the issues involved was rendering time, so eliminating unneeded objects were a significant key so we had otherwise the efficiency of the system suffered a lot.

A second item involved proper modular design. It is very difficult to understand a innards of a large project without having a key understanding of what each module should accomplish. It is therefore a very useful tool to follow proper design standards in order to avoid confusion and miscommunication. Further, this helped elminate redundant work since by not having proper documentation, a lot of unneeded work was made that was later discarded.

A third item learned that seems trivial at first but which can become a very significant problem if not properly practiced is communication. One of the problems that arose early on was keeping the team properly informed of such things as division of labor, scheduling, etc. This was the result of a failure of the team to properly communicate all necessary information. Often, the result was confusion, loss of work, and redundancy.

The most difficult problem was adapting to a project that was very well defined from the start. The reason that this was a challenge was that it made the division of labor and the development of the project very challenging. This was to be expected and in hindsight, is a good thing since every programmer has to be prepared for this situation when out in the industry. It was nonetheless, a very challenging an on occasion, somewhat hectic thing.

Credits:

Blagojevic: stem cell definition, spread coding
Guajardo: algorithm for eliminating inner surfaces, code-string DNA specification
Both: suggesting ideas, experimenting

Examples:

The possibilities are endless, and we still experiment to get the forms for the final presentation. We will do trees and flowers.

Conclusion & Further Development:

We did not have time to develop the system to an ultimate extent, where it could model the complicated forms (e. g. with skin, bones etc.). From the experimentation we conclude that a stem cell has a great potential for:

1) practice: an artist<->machine interface based on a description of the objects closer to a human view of the world; a good modeller of variable and organic-type forms
2) theory: evolution-based algorithms (e. g. genetic).

One can evolve the system further to improve the first or suit the second purpose.

Source for the Forms/Simple1 (Figure 3.) :

```
stemcell Seed {

    spread Tentacle
    radlife 4.500000 4.500000 0.000000 0.100000
    direction 0.500000 0.500000 0.500000 0.200000
    normal 0.000000 1.000000 0.000000
    dirlife 100.000000 0.000000
    twist 0.030000 0.000000 0.000000 1.000000
    twist_decay -0.010000 0.000000 0.000000 0.200000
    radlife_decay 0.020000 0.020000 0.020000 0.200000
    color_variation 0.000000 0.100000 0.000000 0.000000
    roughness 50.000000 0.400000
    color 0.200000 0.300000 0.050000 1.000000
    daughter Petal {
        pos Terminal
        amount 1
        chance 0.010000
        stemcell Seed {
            spread Radial
            radlife 5.000000 5.000000 5.000000 0.100000
            direction 0.000000 0.000000 1.000000 1.000000
            normal 0.000000 1.000000 0.000000
            dirlife 100.000000 0.000000
```

```
                twist 0.000000 0.000000 0.000000 0.200000
                twist_decay 0.000000 0.000000 0.000000 0.200000
                radlife_decay 1.000000 1.000000 1.000000 0.200000
                color_variation 0.100000 0.100000 0.000000 1.000000
                roughness 30.000000 0.400000
                color 0.700000 0.700000 0.700000 1.000000
            }
        }

}
```