This evolution is shown in Figure 11.4.2 which is not close to settled. Our last image of the evolution of the Game of Life shows the glider gun considered in Section 7.3. The POV-Ray file is created as follows and the image is shown in Figure 11.4.3.

```
   (30 pad |:A2) life_evo_pov
povpath,'life_c.pov'
```

Notice the streams of gliders heading upward to the right and the periodic structure between two columns that generates those streams.

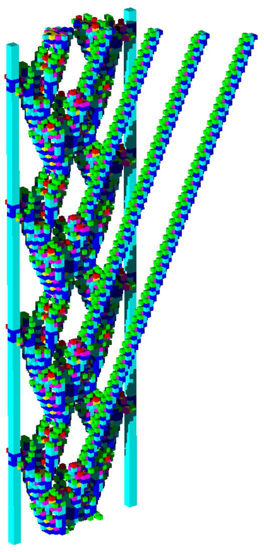

Figure 11.4.3 Life on a Glider Gun

11.5 Cyclic Cellular Automata

We saw in Section 7.5 that cyclic cellular automata evolved through phases into a periodic state with self-organized spirals. Recall that a cell in these automata evolves by increasing to the next state (cyclically) if it has a neighbor in that next state. In this section we investigate a three dimensional analog of these automata. Thus, we will use the third dimension to represent the three dimensional array of states, rather than the time evolution as in the previous section.

If we consider 3 by 3 by 3 neighborhoods in a three dimensional array of states, we see the center cell has index 13 and the neighbors that share a face (Von Neuman neighbors) have indices 4, 10, 12, 14, 16, and 22. Here we will look at an example with 36 states. Notice the local rule has the same definition as was used in Section 7.5 while the global rule needs periodic extension in a third direction and the tessellations are 3 by 3 by 3.

```
   <"2 i.3 3 3
+-----+--------+--------+
|0 1 2| 9 10 11|18 19 20|
|3 4 5|12 13 14|21 22 23|
|6 7 8|15 16 17|24 25 26|
+-----+--------+--------+

   ns=:36

   cen=:13

   nei=:4 10 12 14 16 22

   perext3=:(perext"_2)@:(perext"_1)@:perext

   lcca=:(ns|cen&{ + (ns|1+cen&{)e. nei&{)@,

   cca3d=: 3 3 3&(lcca;._3)@perext3
```

We iterate this automaton on an 80 by 80 by 80 random array of states. We save the result after 300 and 500 iterations and visualize every 4th state using POV-ray.

```
   a0=:?.(3#80)$ns

   a300=:cca3d^:300 a0

   a500=:cca3d^:200 a300
```

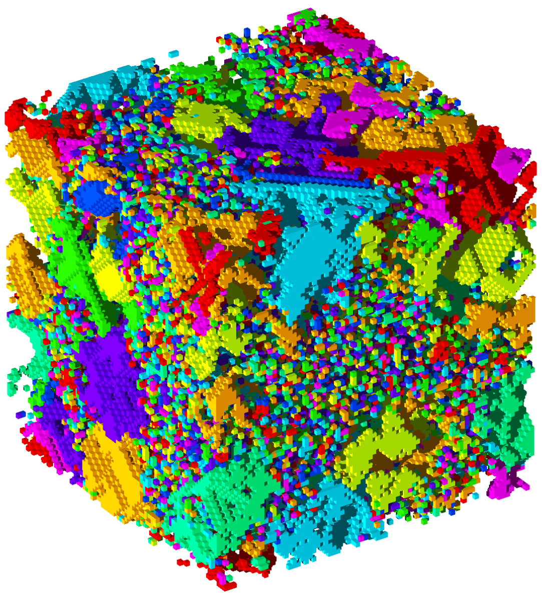
Figure 11.5.1 Cyclic Cellular Automaton Droplets

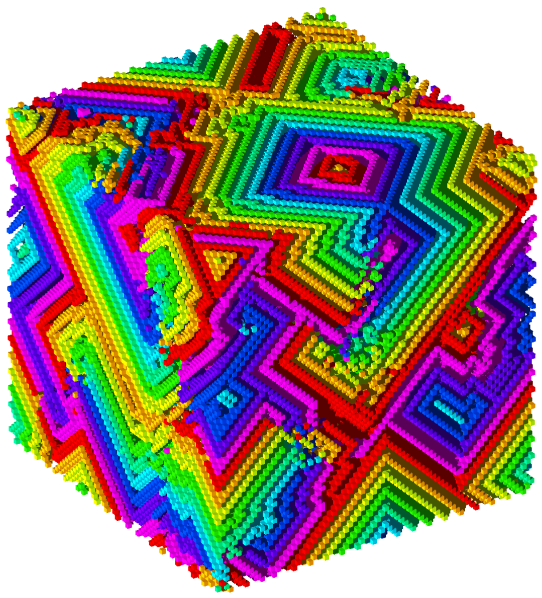
Figure 11.5.2 Cyclic Cellular Automaton Self-organized Spirals

```
   view_pars_cca3d
camera{
location <40,60,40>
angle 2.2
up <0,0,1>
right <0,1,0>
sky <0,0,1>
look_at<0,0,0>
      }
#default{finish{ambient 0.35}}
object{light_source{<200,100,80> color rgb<2,2,2>}}
background {color rgb<1,1,1>}
```

```
   cca3d_pov
1 : 0
:
colors=.Hue1 5r6*(i.%<:)#x
view_pars_cca3d fwrite y
's0 s1 s2'=.$m
dx=.2%>./$m
xyz=.,/,/_1+dx*(i.s0) ,"0 1/ (i.s1),"0/i.s2
for_k. i.#x do.
  M=.,m=k{x
  ((k{colors)fmtbox (, dx&+)"1 M#xyz)fappends y
end.
fsize y
)
```

```
   (4*i.9) a300 cca3d_pov povpath,'vn36_300.pov'
11320021
```

```
   (4*i.9) a500 cca3d_pov povpath,'vn36_500.pov'
11127799
```

Figure 11.5.1 shows the result after 300 iterations. Notice the droplet patches of constant state. Figure 11.5.2 shows the results after 500 iterations and nested three dimensional spirals have self-organized.

11.6 Rendering Surfaces

We return to the topic of surface plotting that we first discussed in Section 10.3. Here we use some utilities from *povkit.ijs* that we have not used before; namely, utilities for formatting a triangle and breaking a quadrilateral into two triangles. The definitions may be seen in that script which we assume has been loaded. We will illustrate their use here. Unlike earlier formatting utilities, these take a matrix right argument (the matrix lists the vertices).

```
   1 0.5 0 fmttri 1 1 1,2 2 2,:3 5 7
object{triangle{<1,1,1>,<2,2,2>,<3,5,7>}pigment{rgb<1,0.5,0>}}

   0 1 1 fmtquad 1 1 1,2 2 2,3 5 7,:0 0 0
object{triangle{<1,1,1>,<2,2,2>,<0,0,0>}pigment{rgb<0,1,1>}}
object{triangle{<2,2,2>,<3,5,7>,<0,0,0>}pigment{rgb<0,1,1>}}
```

Now we construct our sum of three sines function from Section 5.3, sample it at 129 points in each direction and then rearrange the data to obtain the necessary quadrilaterals.

```
   sin=: 1&o.

   f1=: +&sin"0

   f2=: sin@(+&.*:)

   f3=: (f1 + f2)"0 f.                   the three sines function

   $x=: y=: _14+28*(i.%<:)129            sample points
129

   $xyz=: x ([,],f3)"0/ y                array of x-y-z points
129 129 3

   quad=: 0 1 3 2&{@(,/)
```

The function `quad` rearranges 2 by 2 arrays of points into a list of 4 points which is a quadrilateral suitable as an argument to `fmtquad`. We then apply `quad` to the 2 by 2 tesselations of the *x*-*y*-*z* data to get the polygons we want to plot, we compute the average *z*-coordinate so that we can false-color the height in the same way we false-colored contour plots of this function in Section 5.3, and we use `cile` from that section to break the colors into 1000 distinct hues. The function `Hue1`, which is defined in *povkit.ijs*, creates hues on a 0-1 scale.

```
   $polys=: ,/,/2 2 quad ;._3 xyz              the polygons
16384 4 3

   $avgz=: 2 2(+/%#)@, ;._3 {:"1 xyz           the average heights of the polygons
128 128

   H1K=: Hue1 5r6*(i.%<:)1000                  a thousand hues

   $colors=: ,/H1K{~ 1000 cile avgz            color using hue for height
16384 3

   fn=: povpath,'3sines.pov'                   POV-Ray file name

   view_pars_3sines
camera{
     location <-40,-60,40>
    angle 40
    up <0,0,1>
      right <0,1,0>
```