A Cell-DEVS Geographic Visualization Framework Using Google Earth *

With A Land Use and Land Cover Model Prototype Implementation

Yu Wang Peiwen Chen

University of Ottawa
School of Electrical Engineering and Computer Science
161 Louis Pasteur
Ottawa, Ontario K1N 6N5, Canada
ambitionwang@gmail.com
pchen082@uottawa.ca

Abstract

Cell DEVS could be used to model many environmental and social issues. With more and more online geographic information, the simulation results from some specific Cell DEVS models have the opportunities to be visualized geographically . In this paper, a framework based on Google Earth is proposed to visualize the Cell DEVS simulation results. An land use and land cover Google Earth visualization prototype is implemented to validate this framework. It supports two dimensional and three dimensional Cell DEVS simulation visualization and is flexible and easy to make new configurations. With this tool, more realistic simulation display will accelerate the corresponding research domains. Additionally, due to the limiting rendering capabilities of Google Earth, a cell merging algorithm is implemented to reduce the rendering elements. Simulations results show that this algorithm can work efficiently and make Google Earth simulation run more smoothly. Finally, it is pointed that the scale of the simulation is bounded the size of rendering elements available in Google Earth.

Categories and Subject Descriptors I.6.7 [*Simulation and Modeling*]: Simulation Supporting Systems—Environments; H.5.1 [*Information Interfaces and Presentation*]: Multimedia Information Systems—Animations

General Terms Algorithms, Design

Keywords Google Earth, GIS (Geographical Information System), Cell DEVS

1. Introduction

Geovisualization has been found many applications in real world situations, especially related to geographic environment, such as wild land firefighting, forestry, archaeology, environmental studies, urban planning. While a variety of software, known as Geographic Information System (GIS) could visualize, analyze, interpret and understand georeferenced information by allowing different operations with maps. GIS and Geovisualization could provide a directly way for spatial data changes analysis by allowing for more interactive maps compared to traditional, static maps, including the ability to explore different layers of the map, to zoom in or out, and to change the visual appearance of the map. Many GIS has already contained embedded simulation capabilities, however, it just supports several specific modeling and simulation focused on several kinds of geographic information. On the other hand, simulators running separated from GIS tools could provide precise simulation results but they have no connections to georeferenced environment. Therefore, sophisticated and intuitive interfaces for visualizing updated geographic information is in need.

2. Background

DEVS[1] is a discrete-event formalism that allows a hierarchical and modular description of the models, which describes behavioral (atomic) or structural (coupled) models. Cell-DEVS[2] formalism extended DEVS allowing modeling systems represented as executable cell spaces by defining each cell as a DEVS atomic model and the space as a DEVS coupled model. CD++[3] is an open source environment for executing DEVS and Cell-DEVS models. Originally, CD++ produces a text file (.log file) as the simulation result for analysis, we need to retrieve the text message line by line to get the state transitions and port messages. A visualization tool is in need to understand directly the behavior of DEVS and Cell-DEVS. CD++ Modeler could both show 2D and 3D simulation results in a comparable natural way, however, sometimes we need a more extensive interpretation and construction to see what is exactly happening during the simulation. Recently, CD++ was provided with facilities for 2D or 3D visualization using Virtual Reality Modeling Language(VRML), Autodesk Maya[4], Open Graphics Library (OGL)[5], and Blender[6]. [4] shows how advanced DEVS models could be visualized using an advanced generic visualization environment like Maya. DEVSView, as an Open GL-based toolkit, provides a GUI and a text format for the creation of visual models which translates CD++ log file into animations. DEVSView[7] uses these visual models to represent the simulation process, while users could set up the rules, to trigger state changes and control the animations. MAPS[8] is a graphical modeling and visualization toolkit exclusively designed for simulation of urban traffic.

* This work was supported by SYSC5104

SYSC5104 '12 December 16–17, 2012, Ottawa, Ontario, Canada.

All of these graphical modeling and visualization toolkits in CD++ are applicable to general DEVS and Cell-DEVS models except that MAPS only focuses on urban traffic Model and Simulation. However, in terms of wide application of GIS, we need a specific visualization tool for analysis and interpretation of the Cell-DEVS models containing geographic information. This project focuses on how to build up a visualization of CD++ models sensitive to geoinformation.

3. Model defined

3.1 The overview of the geographic visualization framework

GIS has been found many applications in real world problem, such as Defense and Intelligence, Health and Human Services, Transportation, Natural Resources and Utilities and Communications. A geographic dataset always include a comprehensive collection of vector, or raster, or imagery data covering some parts on earth. For vector dataset, it often includes hydrography maps, geological maps, soils, administrative boundaries and others; for raster dataset, it often includes elevation, slope, aspect, landuse and geology; for imagery dataset, it often includes 1 m resolution orthophoto, several LANDSAT-TM5/7 scenes and a MODIS daily LAND Surface Temperature(LST) time series[9]. Any of the above geo-information could be applied to CD++ modeling and simulation for analysis.

The modeling and simulation of problems sensitive to geo-information in Cell-DEVS always interprets one specific area as a cell and the whole region as the cell space. To integrate GIS and Cell-DEVS modeling, we need to map the geo-information from real world geo-ordinates to cell space ordinates, separates the whole region into cells and store as the initial file (.val file) for CD++. Then, running CD++ will produce a log file(.log file) as the simulation results. Lastly, we need to analyze the text line by line to get simulation process and map it back to real world geo-ordinates, and visualize in Google Earth.

We developed a framework(Figure 1 on page 2) to integrate Cell DEVS models sensitive to real world geo-information with a powerful Geospatial Visualization System (GVS) based on [10], and the following will demonstrate how to prove the usefulness of this proposed structure by taking land use change as an example in this project. Firstly, geographic information of a specific area will be retrieved from GRASS GIS based on GDAL then the defined geo-information sensitive Cell-DEVS model will read this initial information. M&S in CD++ generates log file. To visualize the M&S process, we feed the log file to Google Earth.

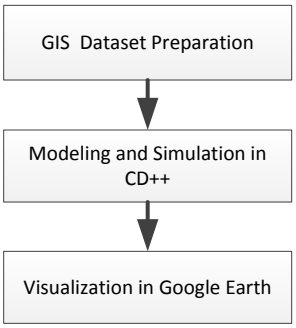

Figure 1. The structure of the geographic visualization framework

3.2 GIS Dataset Preparation

GRASS is one of the most popular GIS and could handle with raster, topological vector, image processing and graphic data. GRASS GIS use Geospatial Data Abstraction Library (GDAL)[11] for raster/vector import and export. Geographic information could be read from raster maps based on the data model of GDAL, such as coordinate system, affine geotransform and raster band stored information. We choose a sample raster dataset named North Carolina (NC, USA) in GeoTIFF format provided by GRASS.

NC dataset covers parts of North Carolina (NC), USA , prepared from public data sources provided by the North Carolina state and local government agencies and Global Land Cover Facility (GLCF). This dataset offers raster, vector, LiDAR and satellite data. We choose this dataset in raster format for the purpose of retrieving information from the dot matrix data structure. And we chose GeoTIFF[12] as the standard file format, as it is supported by most GIS, including GRASS, for both the input and output purposes. The raster map set in GeoTIFF format includes elevation, slope, aspect, watershed basins, geology, and landuse, while this paper focuses on landuse changes.

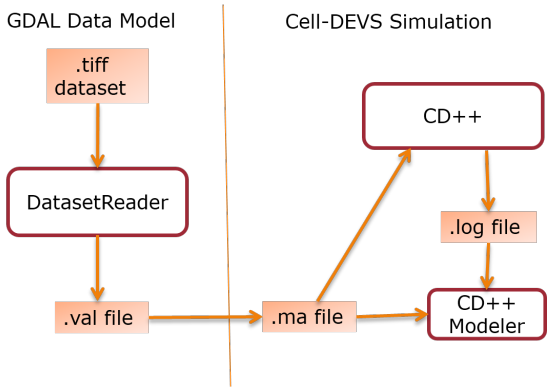

Figure 2. The architecture of data retrieval and Cell-DEVS simulation

The dataset chosen contains the geographic information in a specific area and GRASS GIS serves as GUI to display all the information in this dataset. We need to retrieve information from GRASS GIS (either via an API or import/export files) to communicate with the simulator (CD++), while GDAL presents a single abstract data model to the calling application for all supported formats. The abstract data model includes dataset, Coordinate System, Affine GeoTransform, GCPs, Metadata, Subdatasets Domain, Image_Structure Domain RPC Domain, XML Domains, Raster Band, Color Table and Overviews. Georeferencing information could be properly read from GRASS by taking advantage of this data model of GDAL.

As showed in Figure2 , in order to model and simulate in CD++, firstly we need to retrieve initial information from GRASS GIS based on GDAL. "DatasetReader" will read the dataset geo-information in GeoTITFF format and output it as .val file. Then, with the defined transition rules showing interaction between land use and population, this Cell-DEVS model will show land use changes and do predictions by generating .log file as the simulation result. With CD++Modeler, we have a directly way to visualize the S&M process.Raster dataset defined in GDAL consists of several raster bands which contain some information in common. One raster band contains the map size and GDAL data type, which could be Byte, Float32 or Int32. Block is a preferred and efficient access chunk size in GDAL data model. Also, the raster band contains a color table consisting of a zero or more color entries. To associate

a color with a raster pixel, the pixel is used as a subscript into the color table. And we need to check the color interpretation table to get the specific meaning of that color in this band. Below shows the steps to retrieve data from a raster dataset.

For given "landuse.tiff" dataset:

- For each raster band in this dataset:
  - Get this raster band information: Xsize, Ysize, BlockSize.
- For each block in this band:
  - Get block information: valid block size, stored block data;
  - Read block data and output each pixel data in this block.

The "landuse.tiff" file in the whole map set will be used for geographic information retrieval, and land use will be the first layer in our model. This raster dataset contains only one raster band, while the width and height in pixels are separately 179,165 for this raster band, and it could be divided into four blocks.

Reading the block data of the raster band of "landuse.tiff", we could get the color value of each pixel. Then we compared the color entries with interpretation table, found that only seven landuse types of 24 types exist in this raster dataset. It is shown below.

Landuse color entry Value	Landuse type
1	High Intensity Developed
2	Low Intensity Developed
4	Managed Herbaceous Cover
7	Evergreen Shrubland
15	Southern Yellow Pine
18	Mixed Hardwoods/Conifers
20	Water Bodies

Table 1. The land use types exist in the "landuse.tiff" raster dataset

The whole raster band size is 179 ×165, while limited by the CD++ tool computation capability, we choose one small area in the data map, the resized area is only 8 x 63, which contains six of these seven land use types in the whole dataset.

With this specifically chosen area, we retrieve its pixel color value as the initial landuse type after compared to color interpretation table, and store it as initial information file (.val file) for CD++ modeling. "landuse.val" contains information for land use types at the beginning.

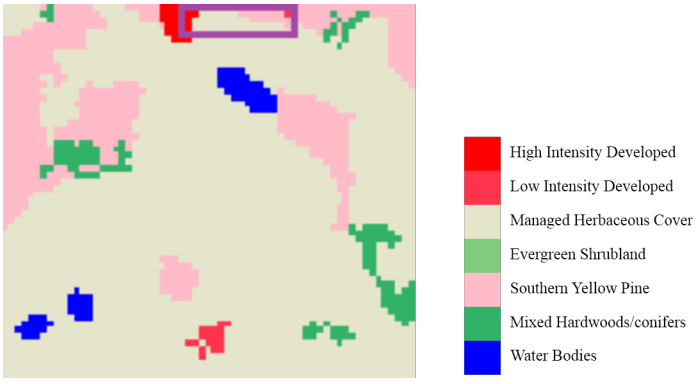

Figure 3. The " landuse" map (left) and the selected area for simulation

3.3 Land Use and Land Cover Cell Model in CD++

Changes to land use has been popular in urban planning, engineering, geography, urban economics, and related fields. Some other networks, such as transportation, population, soil distribution, can exert a strong influence on land use pattern changes; also, new patterns have influence on the developments of such networks.

To forecast land use changes, several methods have been developed. The simplest types of models are Markovian models such as Markovian chain models, which treats land use changes as a stochastic process. A changing probability, which is almost constant retrieved from large amount of history recorded data, exists between two land use types always. With Markovian models, we could reach a distribution of land uses by forwarding to a future time. This kind of models can hardly incorporate other geoinformation networks except some special methods are adopted, such as incorporating artificial neural network into Markovian models [13]. However, it makes the process too complex.

Another method drawing much attention is cellular and agent-based models in simulating land use changes. Cellular automata models emphasize neighbor effects and dynamic interactions between agents, while other models are often coupled into this land use simulation model by adding other simulation layers.

In order to simulate interaction between land use types and population, we chose cellular automata method while adding population information to land use information, instead of adopting the Markovian chain models.

Model formalism: The following box contains the formal specification for the Cell-DEVS Population model.

$$CD =< X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, D >$$

$$X = \emptyset, Y = \emptyset$$

$$S = \{1, 2, 4, 7, 15, 18, 20\}$$

$$N = neighborhood = \{$$
$$(-1,-1,0),(-1,0,0),(-1,1,0),$$
$$(0,-1,0),(0,0,0),(0,1,0),$$
$$(1,-1,0),(1,0,0),(1,1,0),$$
$$(-1,-1,1),(-1,0,1),(-1,1,1),$$
$$(0,-1,1),(0,0,1),(0,1,1),$$
$$(1,-1,1),(1,0,1),(1,1,1)\}$$

$$d = 100ms$$

$$\tau : N \rightarrow S$$

For the first layer:

$S = 1$ if $(0,0,0) = 2$ and more than 1 neighbor state is 1;

$S = 7$ if $(0,0,0) = 4$ and more than 2 neighbor state is 7;

$S = 2$ if $(0,0,0) = 7$ and more than 2 neighbor state is 2;

$S = 2$ if $(0,0,0) = 20$ and more than 3 neighbor state is 2;

$S = (0,0,0)$ for others

For the second layer:

$S = (0,0,0)$ remain unchaged

4. The design of Google Earth visualization

Google Earth[14] is a powerful GVS supporting KML elements (place, marks, images, polygons, 3D models, textual descriptions, etc.) to manage three-dimensional geographic data. For the ease

of use of Google Earth, visualization based on it could enhance the communication of results to non-technical users and provide instantaneous access to a database of information layers.

4.1 A brief introduction to KML

Google Earth uses KML to display geographic data. Based on the XML standard, KML uses the tags and its attributes to descript the geography information. With an script language with regular expresions match in-built features, it is easy to manipulate the KML files. In our implementation, we use perl as our script lanuages for Google Earth Visualization. In our Implementation, we use an subset of the elements defined in KML, including `<Document>`,`<Style>`, `<Folder>`,`<Placemark>`, `<Polygon>` and `<Timestamp>`.

The following program code is cut from our implementation. It has the main features and basic structures as more more complex KML output in our visualization. Line 1 is an XML header. Line 2 is an KML namespace declaration. Line 3 states the beginning the `<Document>` and Line 39 concludes the the `<Document>`. We use `<Document>` as our Cell DEVS simulation unit shown in Google Earth. Line 6 states an Style, which descripts an polygon style. We could define different colors for differenct cell states. Every style may have its state name as its style name. In this design, we use an cell state configure file called `cell.style` to descript the cell state style we want. Line 12 begins an `<Folder>` element, which stands for a layer in our Cell DEVS model. Line 38 conclude the `<Folder>` element. If the model has more than one layer, `<Folder>` and `</Folder>` pairs will be iterated the same times as the number of the layers in the sepecific models. It could use layer number as its name. The `<Folder>` element contains lots of `<Placemark>` elements. In this design, a `<Placemark>` element depicts a cell region, which could be an squre or irregular polygon. In this simple example, it contains an `<Polygon>` element for an square. It should be noted that the first coordiate is the same as the last one in KML `<Polygon>` element specification. The `<tessellate>1</tessellate>` guarentees that the geography will be tessellated on the ground and gives a better animation effect. In addition, the `<Placemark>` element contains the `<Timestamp>` element. It give the `<Placemark>` bobing-up time.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <kml xmlns="http://www.opengis.net/kml/2.2">
3  <Document>
4  <name>visualization</name>
5
6  <Style id="CellState">
7  <PolyStyle>
8  <color>aabbggrr</color>
9  </PolyStyle>
10 </Style>
11
12         <Folder id="#id layer">
13 <name>#id layer</name>
14
15 <Placemark>
16 <name>cell name</name>
17 <styleUrl>#CellState</styleUrl>
18
19 <Polygon>
20 <tessellate>1</tessellate>
21 <outerBoundaryIs>
22 <LinearRing>
23 <coordinates>
24         -78.7707183652255,35.809591457038,0
25         -78.770717636844,35.8098483367093,0
26         -78.7704022759328,35.8098477429906,0
27         -78.7704030053161,35.8095908633212,0
28         -78.7707183652255,35.809591457038,0
29 </coordinates>
30 </LinearRing>
31 </outerBoundaryIs>
32 </Polygon>
33
34 <TimeStamp>
35 <when>2010-01-01T00:00:00.000</when>
36 </TimeStamp>
37
38         </Folder>
39 </Document>
40 </kml>
```

Severl important notes should be made clear here. Firstly, the relationship between the cell region and an placemark containing an polygon. Figure 4depicts an two dimensional cell space with three kinds of states. In this case, we could use 16 placemarks to represent every cell respectively. Because every placemark has only one style and we could draw an more complex polygon shown in 5, it is possible to use four placemarks to depict the sates of the cell space at that time. Actually, the `<Polygon>` supports inner ring. Therefore, in Figure 5, we could merge the two externel placemarks. But it is hard to implement this optimization in design, which will be explained in 4.4. Secondly, at timestamp, the correspoding placemark will appear on Google Earth. It will keep its appearance until 1) the end of the simulation and 2) part of the region will be covered by the appearance of another placeamrk at a later stamptime , which is explained in Figure 6. Therefore, when an cell state changes, we just put that placemark with the correspoding style at that specific time. It will stay in that style until next change occur. Finally, the maximum uncompressed KML file size is limited to 10MB currently. Google says that this limitation will be changed at any time. It is certaint that if we passed too many elements into KML file, Google Earth wouldn't work. Therefore, it is important to reduce the number of Google Earth elements when the KML is generated. This will be explained clearly in 4.4.

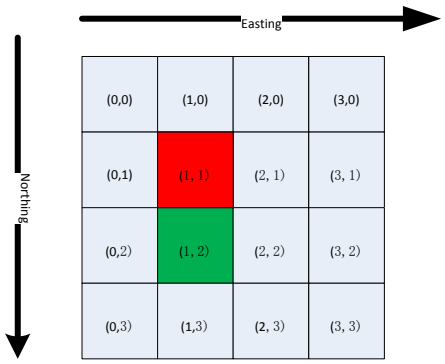

Figure 4. An two dimensioanl cell space with three kinds of sates

4.2 The Goegraphy Coodination System Conversion

For validation our visualization framework, we implement a land cover and land use model. The `landuse96_28m.tif` form grass, an opensouce online GIS dateset, is used to retrieve the geography information. The Figure7 is a subset of this file. Every color in this file stands for an kind of land cover. Thus, a pixal with a specific color stands for a square of land with corresponding use. When the

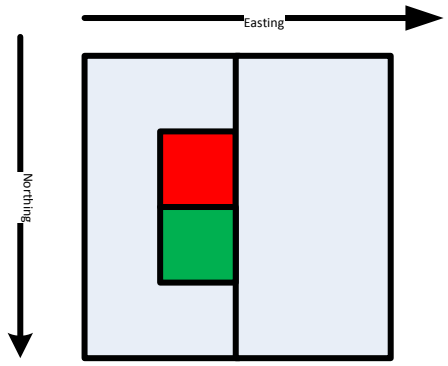

Figure 5. An possible palcemark configuration for Figure 4

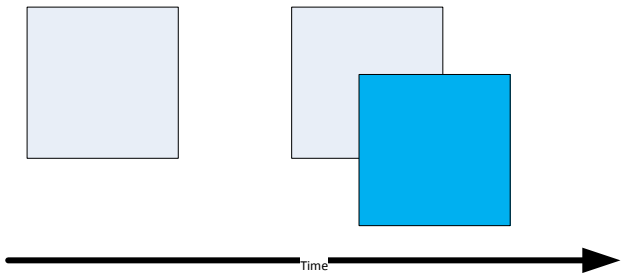

Figure 6. An placemark is covered by the later apearance of another placemark with a different style

land use information is retrieved and fed into the CD++ simulation tool as the initial value file. The geography information of each cell should be kept in a separate file. In our implementation, the geography information is stored in a file called `geo.info` . With this information, the geography visualization could be achieved.

We could use the command `gdalinfo` from gdal to check the geography information, which is shown in the Figure 7. This file use Lambert Conformal Conic geography project system, which is defined by two ellipsoidal parameter, semimajor a and flattening factor f ; the latitudes of the north and south standard parallel, $\phi_N$and ϕ_S; the latitude of the origin, λ_0;the central meridian ϕ_0; the false easting and northing positions E_0, $N_b$in XY coordinate plane. Also, this plane size is fixed by the coordinates of upper left, lower left, upper right, lower right. We could see that every pixel stands for an area of square. But Google Earth uses an different kind of geography reference system called World Geodetic System 1984(WGS84). WGS84 uses longitude and latitude pair (ϕ, λ) to define the unique position on the earth. So when we visualize the simulation results on Google Earth, the geography information corresponding the initial value file stored in geo.info has to be converted to longitude and latitude pairs (ϕ, λ) for each cell.

In our implementation, each cell has four coners. For eveyr corner, an position (E, N) is corresponding to that corner. Thus, each cell use four positions to descript its geography information. In Figure 8, there is an two dimensioanl cell space, in which every red circle stands for an position. In order to visualize the results, geography references conversion is mandatory. In our case, we need covert from Lambert Conformal Conic to World Geodetic System 1984(WGS84).For every position (E, N) on the plane, we

```
E:\devs-visual\geo>gdalinfo landuse96_28m.tif
Driver: GTiff/GeoTIFF
Size is 526, 474
Coordinate System is:
PROJCS["Lambert Conformal Conic",
    GEOGCS["NAD83",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS 1980",6378137,298.2572221010002,
                AUTHORITY["EPSG","7019"]],
            AUTHORITY["EPSG","6269"]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4269"]],
    PROJECTION["Lambert_Conformal_Conic_2SP"],
    PARAMETER["standard_parallel_1",36.16666666666666],
    PARAMETER["standard_parallel_2",34.33333333333334],
    PARAMETER["latitude_of_origin",33.75],
    PARAMETER["central_meridian",-79],
    PARAMETER["false_easting",609601.22],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]]]
Origin = (629980.000000000000000,228527.250000000000000)
Pixel Size = (28.500000000000000,-28.500000000000000)
Metadata:
  AREA_OR_POINT=Area
Corner Coordinates:
Upper Left  (  629980.000,  228527.250)
Lower Left  (  629980.000,  215018.250)
Upper Right (  644971.000,  228527.250)
Lower Right (  644971.000,  215018.250)
Center      (  637475.500,  221772.750)
Band 1 Block=526x3 Type=Int32, ColorInterp=Gray
  NoData Value=-9999
```

Figure 7. The geography information from landuse96_28m.tif

need the corresponding (ϕ, λ) . The formulation could be found in [15] and is rewritten as follows.

$$W(\phi) = \sqrt{(1 - e^2 \sin^2(\phi))}$$

$$M(\phi) = \frac{\cos(\phi)}{W(\phi)}$$

$$T(\phi) = \sqrt{\frac{1 - \sin\phi}{1 + \sin\phi}(\frac{1 + e\sin\phi}{1 - e\sin\phi})^e}$$

$$w_1 = W(\phi_S)$$

$$w_2 = W(\phi_N)$$

$$m_1 = M(\phi_S)$$

$$m_2 = M(\phi_N)$$

$$t_0 = T(\phi_0)$$

$$t_1 = M(\phi_S)$$

$$t_2 = M(\phi_N)$$

$$n = \sin\phi_o = \frac{\ln m_1 - \ln m_2}{\ln t_1 - \ln t_2}$$

$$F = \frac{m_1}{nt_1^n}$$

$$R_b = aFt_0^n$$

$$E^{'} = E - E_0$$

$$N^{'} = R_b - (N - N_b)$$

$$R = \sqrt{E'^2 + N'^2}$$

$$t = (\frac{R}{aF})^{1/n}$$

$$\gamma = \tan^{-1}\frac{E^{'}}{N^{'}}$$

$$\chi = 90^{\circ} - \tan^{-1} t$$

$$\phi = 90^{\circ} - 2\tan^{-1}[(\frac{1 - e\sin\phi}{1 + e\sin\phi})^{e}]$$

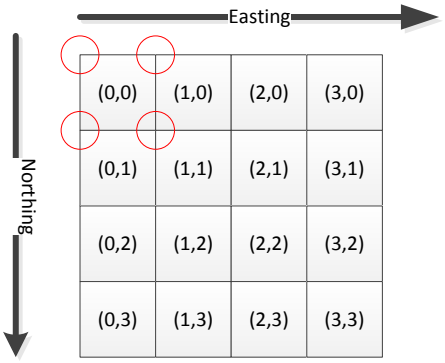

Figure 8. The plance ordinate sytem of the cell model

For other GIS datasets, different geography reference systems could be used. If the Cell DEVS model used data sources with another kind geography coordinates, the corresponding conversion script should be written. In the future, more geography reference conversion scripts will be integrated into this framework.

4.3 The overview structure of the visualization procedure

The Google Earth visualization procedure consists of two phases, the preparation phase and the generation phase. The preparation phase has the following stages:

1. Collecting gepgraphy information (ϕ, λ) for each cell;
2. Gathering the state style information for display on Google Earth;
3. Collecting the cell model information from MA file;
4. Analyzing the CD++ log file and fething the events information.

For the preparation, every cell will have four postions to describe where it shows on Google Earth. Intuitively, every cell stands for a region. In our assumption, it represents an square. You can also define your own geometry for this cell, but you have to provide your own coordinates. For now, we only store four positoins to depict the region, thus the region is limited to a rectangular. In Google Earth, different colors are used to annoate to different states. As times go, the changing color means the changing state. You could define your own state style in the configuration file `cell.style`, which is shown in the following code, in which 7 state styles are defined. You need provide the keyword style, the style index, the RGB hex code, its name for every style. Also, you have to check whether the number of state styles you defined is more than the states you defined. Otherwise, the script will output an error.

```
style 1   63B8FF   Population3
style 2   B0171F   Population1
style 3   FF3E96   Population2
style 4   00008B   Population4
style 5   008B45   Forest
style 6   FFF68F   NonForest
style 7   8B8970   Urban
```

The CD++ log file is analyzed and it is easy to fetch the state change information using the regular expression. Figure 9show the format of the matched message. The message always begin with `Mensaje Y` followed by the timing and cell position information. And a new state value is after the `/out/` substring. After analysis, the information that the new state value and its timing will be retrieved.

The generation stage will has the following stages:

1. Generating the KML header;
2. Writing the KML body;
3. concluding the KML.

In the KML header, the state style defined in `cell.style` will be rewritten as polygon style used to dye the polygon in each placemark. In the design of how to write the KML body, two versions is implemented. The basic version is simple and direct. Firstly, for each cell, an `<Placemark>` element with its corresponding state will be created. A `<timestamp>` with the initial time will be configured for it. Then, in the following time, from the first event to the end of the simulation, an `<Placemark>` element with its corresponding new state and its `<timestamp>` will be inserted. An `<Placemark>` element with a later `<timestamp>` will cover an couterpart with an early `<timestamp>`, if they have the same region fully or partially. In the implementation, it is should be noted that the time granularity of CD++ tool and Google Earth is different. The time interval has to be enlarged hundreds or thousands times in order to show in Google Earth.

The improved version is based on the observation shown in Figure 5. It is possible to use an single placemark to represent several cells with the same property to reduce the Google Earth elements. In the improved one, an every timepoint, two additional stages, that is `merge_cell` and `generate_graph`, are taken before the tags are inserted. According to the merging rules, `merge_cell` will merge the cell with the same state and generate new polygons. `generate_graph` will generate the corresponding coordinates depicting the boundary. Thus, in the improved version, more complex polygons are inserted. The alogrithm specifics will be talked about in the next subsection.

4.4 The cell merging algorithm

The cell merging algorithm is divided into two phases : `merge_cell` and `generate_graph`. In the merge_cell stage, different merging rules will define different mreged polygons. The <Polygon> supports both inner and outer boundary. But it is not easy to utilize this property to reduce the number of elements drastically. Figure 10 descript such a dilemma. If the merging rules supports including an different cell state inside. It is possible that two cell with different state will be inside. In this case, there are two cells with state 1 surrounding by cells with state 0. According to the rules, all the cells with state 0 will be merged together. But the difficulty is that this geometry is not supported by KML. Therefore, it is important to know what the geometry properties are, given certain merging rules. Another diffculty is that if the merged geometry is too complex, it is hard to retrive the coordinate information. Therefore, an moderate merging rules will be beneficial to our implementation.

For each merged geometry, a fake state is assigend to this geometry. Also, a table will be created to map the fake state to real state defined in cell. The algorithm will scan from left to right and

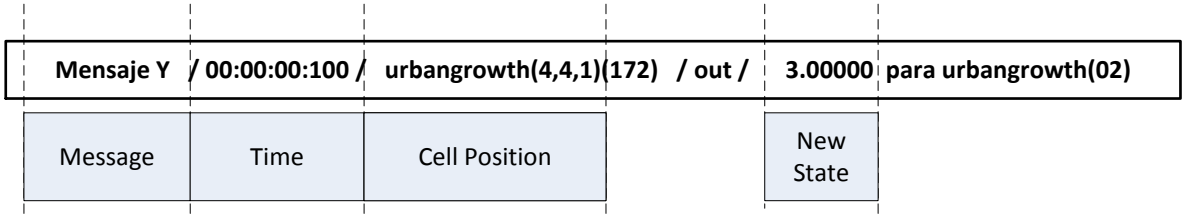

Figure 9. The matched message format in the CD++ log file

from up to down. According to the rules, each cell will be merged into the existing geometries or incur an new geometry. The merging rules is shown in 1.

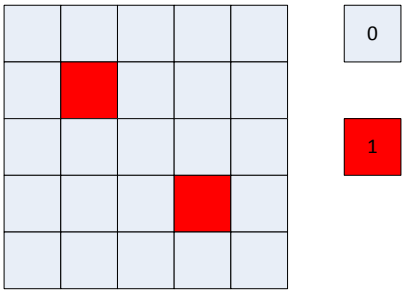

Figure 10. An difficult situation for cell merging

The merged geometry has several properties under these rules. It is easy to prove that the the merged geometry has no inner ring definately and its left edge is always straight, which makes it easy to retrieve geography information from each cell. Figure 11is a typical example in this implementation. `generate_graph` will generate the necessary coordinates for this geometry. The algorithm will scan from top to down, get the right most column element position. Then, in the second pass, it will compare the column number with the underlying one. If they are not equal, two coordinate pairs are needed to be inserted to annotate this change. Otherwise, it will insert nothing. By the end of the second pass, the geograpy information of the right boundary of the merged geometry will be generated. Because it has a straight left edge, it is easy to insert the corresponding coordinates.

5. Simulation Results

5.1 Simulation Results in CD++

Figures 13 and 14 below shows simulation results in CD++ modeler and the colors of the defined pallette represents the same land use type as in the dataset color table.

Seen from the initial state and final state for landuse layer, state transitions happen almost from 4 to 7, from 7 to 2, just as what defined in our model. To relate the landuse changes in cell space with real geo-system, we need to feed our simulation results to GVS.

5.2 Google Earth Simulation Result

Google Earth version 7.0.1 is used to validate our idea. In the basic version, 1787 geometries are needed. After using the cell merging algorithm, 1157 geometries are needed. The number of geometry reduction depends on the change pattern. If the cell states are randam, this cell merging algorithm couldn't make significant

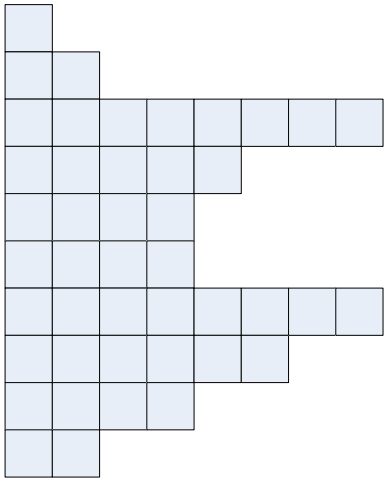

Figure 11. The merged geometry in Google Earth visualization

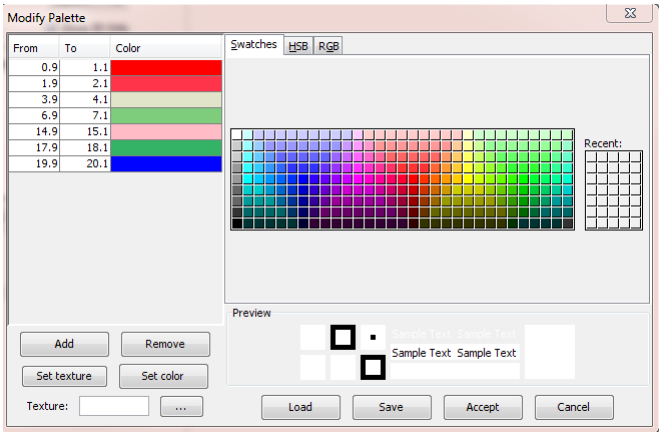

Figure 12. The defined palette for landuse model

improvements. Figure 13,16,17shows the layer 0 Google Earth Visualization, while Figure 18,19shows the layer 1 Google Earth Visualization.

6. Conclusions

We develop a framework to integrate GIS, CD++ modeling and simulation, and visualization in Google Earth in this project. To

Algorithm 1 The merging rules used in thi prototype

```
#rule1
if($column_count==0 and $row_count==0){
  create an new merged geometry;
}
#rule2
if($row_count==0 and $column_count!=0){

        if( the current cell has the same state as its left one ) {
      merged into its left geometry;                              }
        else{
     create an new merged geometry;  }
}
#rule3
if($column_count==0 and $row_count!=0){
    if ( the current cell has the same state as its upper one ){
        merged into its left geometry;}
    else {
       create an new merged geometry;}
}

if($column_count!=0 and $row_count!=0){
#rule4
   if ( the current cell has the same state as its upper and left neighbour and
        its left and upper neighbour has the same fake states)  {
       merged into its left geometry;}
#rule5
   elsif ( the current cell has the same state as its upper and left neighbour and
        its left and upper neighbour has the different fake states)  {
       merged into its left geometry;        }
#rule6
   elsif ( the current cell has the different state from its upper and left neighbour) {
       create an new merged geometry;}
#rule7
   elsif ( Only its left neighbour has the same state) {
       merged into its left geometry;}
#rule8
   elsif( the left neighbour has different state from the current cell and
         the current cell has the same state as its upper neighbour){
      create an new merged geometry; }
}
```

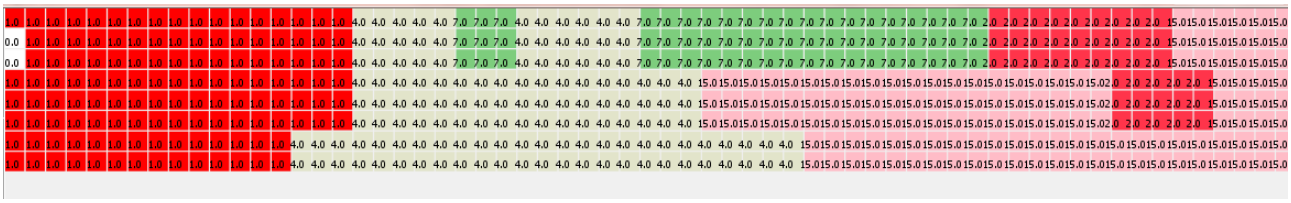

Figure 13. Initial state for landuse layer

prove the usefulness of this integration, we choose the topic landuse. Actually, this framework should work well in other applications, such as natural resources, transportation, utilities. What we need to do is retrieve the information from a corresponding dataset, then model and simulate in CD++, and visualize in GVS. GRASS is chosen as GIS in our project, but not the must, other GIS also should work well with this integration only if corresponding operations supported by it. Also, Google Earth serves as a powerful GVS, but not the must, some other kinds of GVS should work well, such as Bing Maps only if corresponding transforming relationship is found.

In the future, two or more Cell DEVS models will be supported in this visualization framework to see the coupled effects among Cell DEVS. We also could use more complex geometry in Google Earth to further the reduction in Google Earth, which will expand the simulation space for this framework. Finally, we could integrate more Cell DEVS models to validate the general usability of this framework.

Acknowledgments

We would like to thank the technical help from Sixuan Wang and the guide from Professor Gabriel A. Wainer.

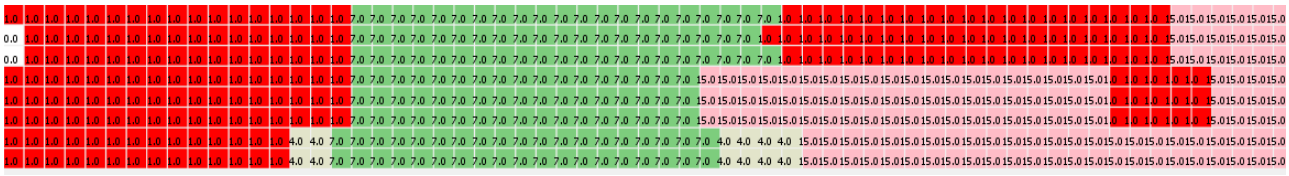

Figure 14. Final state for landuse layer

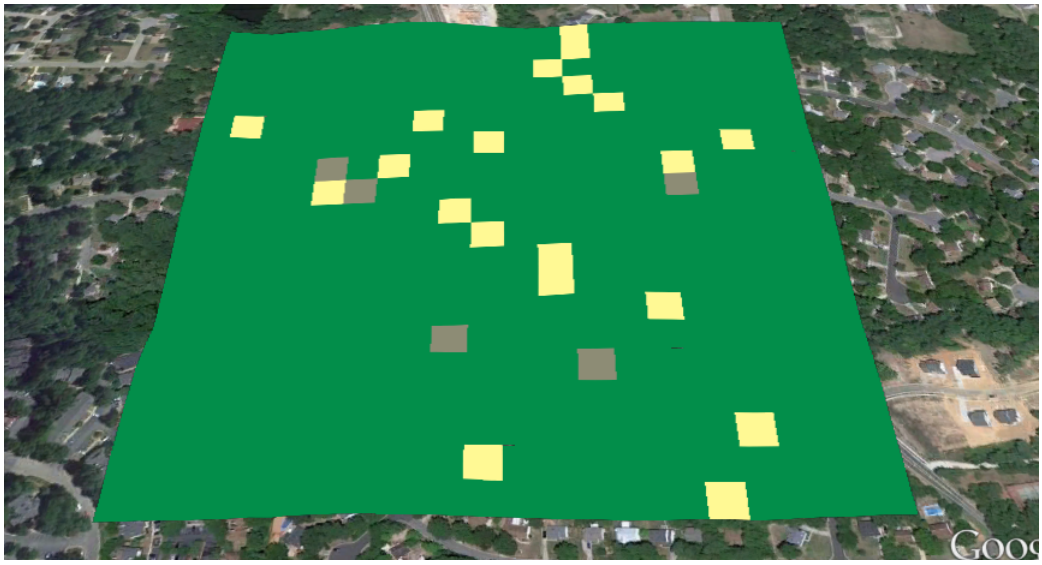

Figure 15. The initial state distribution on layer 0 in Google Earth Visualization

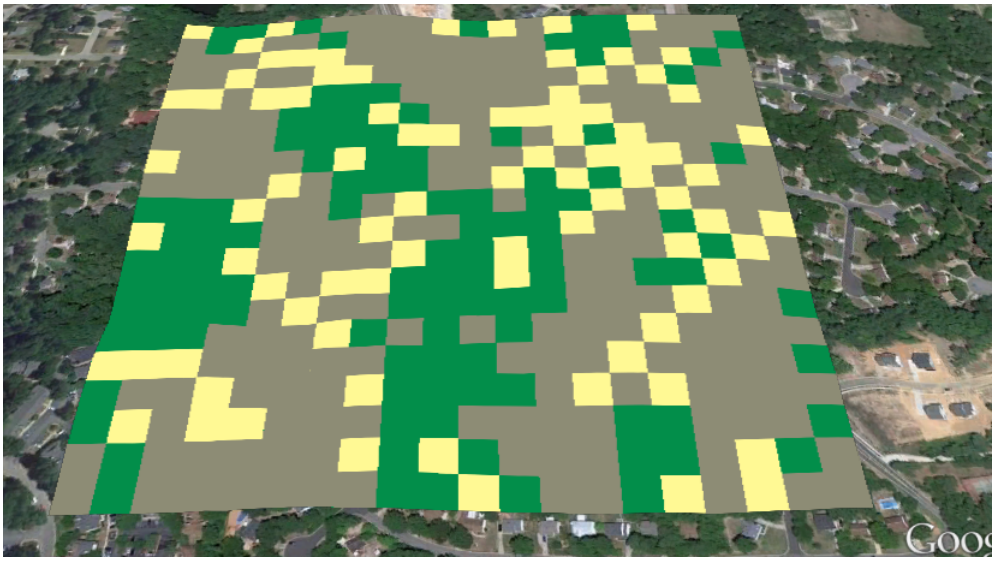

Figure 16. The middle state distribution on layer 0 in Google Earth Visualization

References

[1] B.P. Zeigler, H. Praehofer, and T.G. Kim. *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. Academic Pr, 2000.

[2] G.A. Wainer and N. Giambiasi. N-dimensional cell-devs models. *Discrete Event Dynamic Systems*, 12(2):135–157, 2002.

[3] G. Wainer. Cd++: a toolkit to develop devs models. *Software: Practice and Experience*, 32(13):1261–1306, 2002.

[4] A. Khan and G. Wainer. A visualization engine based on maya for devs models. In *Proceedings of SISO Fall Interoperability Workshop. San Diego, CA. USA*, 2005.

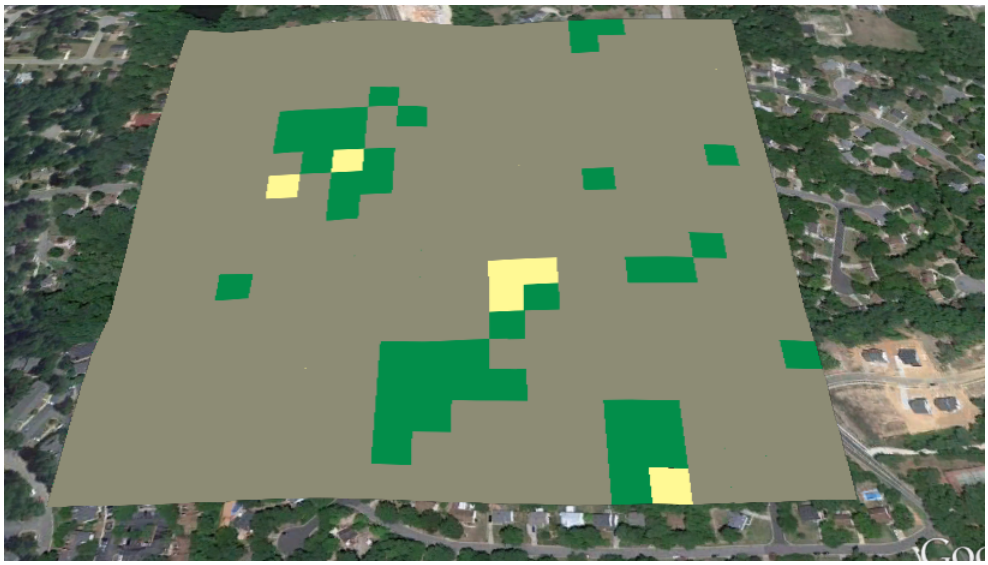

Figure 17. The final state distribution on layer 0 in Google Earth Visualization

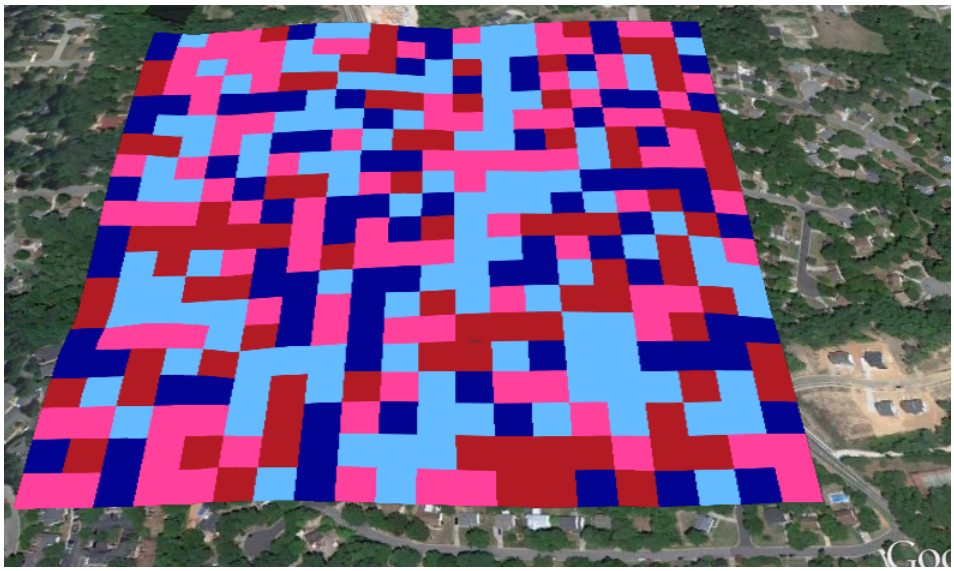

Figure 18. The initial state distribution on layer 1 in Google Earth Visualization

[5] M. Segal and K. Akeley. The opengl r graphics system: A specification, version 2.1, 12 2006.

[6] T. Roosendaal and S. Selleri. *The Official Blender 2.3 guide: free 3D creation suite for modeling, animation, and rendering*. No Starch Press, 2004.

[7] W. Venhola and G. Wainer. Devsview: A tool for visualizing cd++ simulation models. *SIMULATION SERIES*, 38(1):133, 2006.

[8] S. Borho, J. Pittner, and G. Wainer. Defining and visualizing models of urban traffic. In *Proceedings of the SCS 1st Mediterranean Multiconference on Modeling and Simulation*, 2004.

[9] GRASS GIS. Geographic resources analysis support system, 12 2012.

[10] M. Zapatero, R. Castro, G. Wainer, and M. Houssein. Architecture for integrated modeling, simulation and visualization of environmental systems using gis and cell-devs. In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, pages 997–1009. IEEE, 2011.

[11] GDAL. Geospatial data abstraction library, 12 2012.

[12] OSGeo Foundation GeoTIFF Spec. Geotiff format specification, 12 2012.

[13] X. Li and A.G.O. Yeh. Neural-network-based cellular automata for simulating multiple land use changes using gis. *International Journal of Geographical Information Science*, 16(4):323–343, 2002.

[14] Inc. Google. Google earth, 12 2012.

[15] C.D. Ghilani. *Adjustment computations: spatial data analysis*. Wiley, 2010.

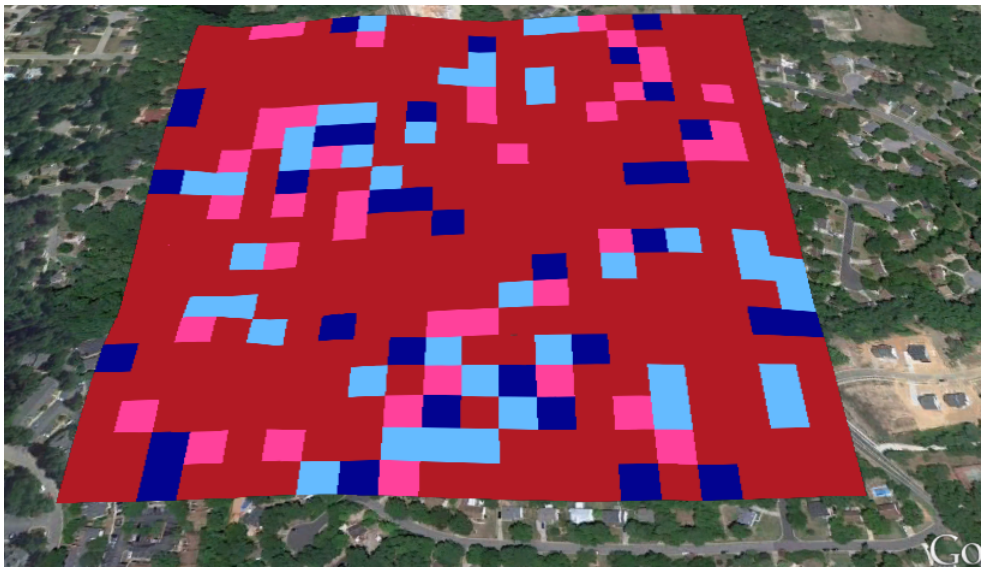

Figure 19. The final state distribution on layer 1 in Google Earth Visualization