Visualizing Spatial and Temporal Dynamics in Google Earth Using Powersim

Abstract

At its genesis, system dynamics (SD) modeling was developed to examine the temporal behavior of interrelated systems. This ability has made SD modeling and analysis the choice for decision and policy makers to do scenario testing and risk analysis. In addition to the conceptual advantages of SD modeling in this realm, are the many SD modeling platforms that have been developed that greatly simplify model creation and more importantly, provide an environment for visualizing the output. However, when making decisions that involve urban planning, electrical and/or water infrastructure, and the like, or for examining impacts of resource development on the environment, the spatial aspects of the decision becomes just as important as the temporal dynamics. To properly support these decisions, simultaneous visualization of the temporal *and* spatial dynamics is needed. This paper presents a methodology for utilizing vbscript from within the SD development platform Powersim to dynamically link Powersim simulations with Google Earth to visualize, in real time, spatial data that change over time. The presentation will describe the logic behind the approach, its capabilities and limitations, and areas for improvement that should be addressed.

Introduction

With its genesis in business applications, system dynamics (SD) modeling is now being used across many different disciplines, from resource management, to urban planning, to medical applications, to space flight. The utility of SD modeling lies in its ability to bring together disparate yet linked systems to capture the feedback and delays that ultimately control the overall systems collective behavior (Forrester 1971). While rarely accurate in an exact numerical sense, the true value of SD models lies in their ability to provide insight into the causal relationships and to clarify the structure of the examined problem (Miezka and Größler 2004). It is only through a clear understanding of the problems structure, and the dynamics of the causal relationships, that true insight can be gained (Forrester 1987).

Historically SD models have mainly dealt with 1-dimensional problems, with that single dimension representing time. Mathematically, these types of problems are represented as an array of first order partial differential equations that only vary in time. Visual output from these models is relatively straight forward; one needs to simply plot how the metric changes as a function of time. An example of this type of output is shown in Figure 1 that shows the temporal dynamics of a predator prey model that represents coyotes and rabbits. Visual examination of Figure 1 illustrates the point made above that while the actual numerical output may be inaccurate (the output in this case being the time dependent rabbit and coyote populations), the model provides great insight into the causal relationships between the two systems. The fact that this insight can be gained with little or no understanding of the underlying numerical model is one of the reasons SD modeling is well suited for decision and policy makers.

Unfortunately, for spatially dynamic systems, visualization is more complex (Dykes 1997). The addition of the spatial component can turn the problem into a 2, 3, or even 4-dimensional problem (the four dimensions being the three spatial dimensions plus time). Gaining insight for SD problems that include one or more spatial dimensions becomes difficult, especially when one considers the fact that most of the SD modeling development platforms lack this capability. An excellent example of spatial visualization is shown in

Figure 2. While spatial visualization and modeling is a key aspect to any geographical information system (GIS), very few tools exist that allow for simulating system dynamics that vary in both space and time (Anselin, Syabri et al. 2004).

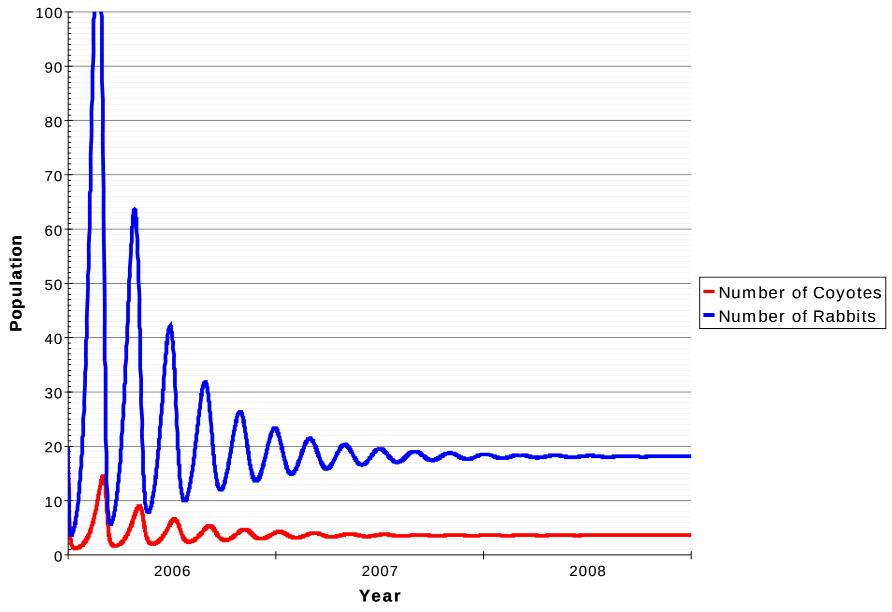

Figure 1 – Sample output from a predator-prey model based on the Lotka-Volterra equations (Volterra 1931) showing the population of coyotes and rabbits.

To address this problem, we have utilized the Visual Basic script (vbscript) capabilities within Powersim to create, initialize, and dynamically update Google Earth (GE) *.kml files. When GE is used simultaneously with the Powersim simulation, the dynamic updating provides a means for visualizing in real time, how spatial data change over time. The remainder of this paper begins with a quick overview of GE and the 'keyhole markup language' (KML), a description of the data needs, details of the vbscript code used in Powersim, and a discussion on improvements that should be addressed.

Google Earth

Over the last decade or so, Google Earth (http://earth.google.com) has progressed from being a 'really cool' computer application with little utility, to a full blown scientific and educational tool. Beyond its ever increasing and extensive database, as well as its satellite imagery, GE's popularity stems from the ease of which spatial data can be visualized, transferred, and shared. This ease of functionality is due to GE's 'keyhole markup language' (KML), which is a XML-based (see http://en.wikipedia.org/wiki/XML for information on XML) language schema for expressing and visualizing geographic information[1]. The name 'Keyhole' stems from the KH (i.e., keyhole) reconnaissance satellites that were the original 'eye-in-the-sky' military reconnaissance system first launched in 1976. KML was originally developed for the parent of

[1] The description of KML in this paragraph as well as the next, comes loosely from http://en.wikipedia.org/wiki/Keyhole_Markup_Language

GE, the Keyhole Earth Viewer, and was kept as the schema of choice when Google acquired Keyhole Inc. in 2004.

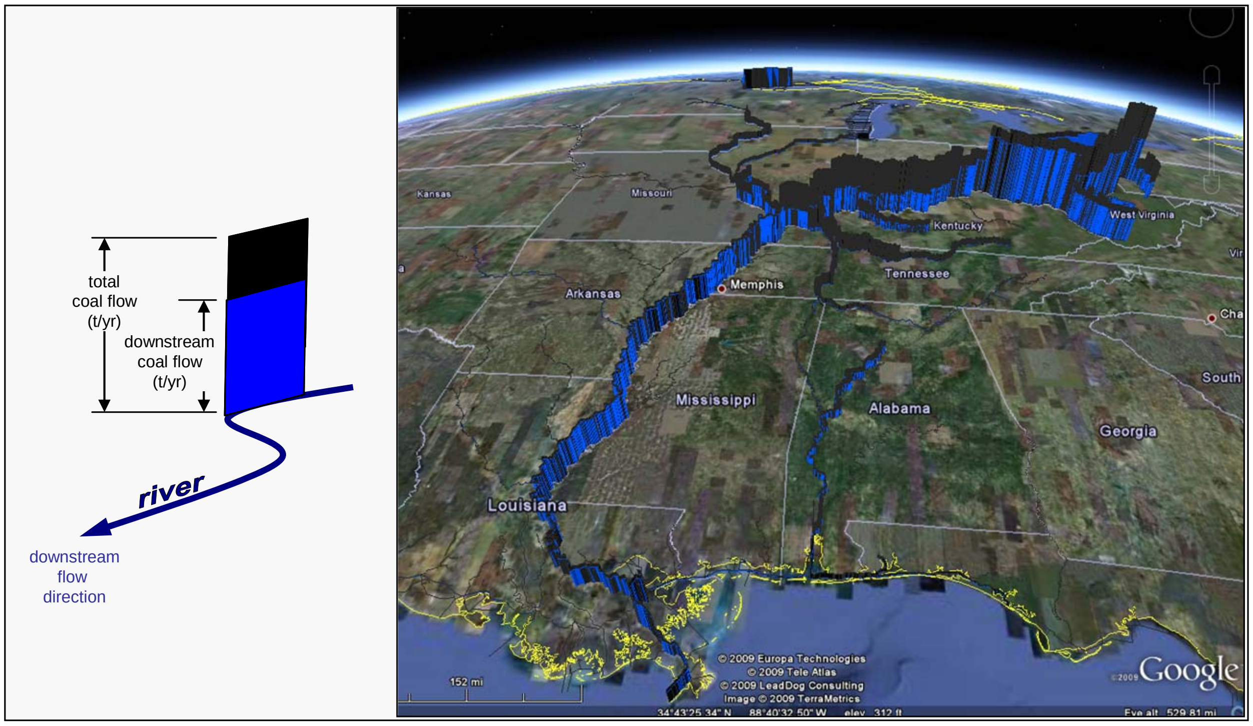

Figure 2 - Google Earth is used to show annual barge commodity flow rates throughout the Mississippi River navigable waterways network (U.S. Army Corps of Engineers).

A KML file (designated as *file_name.kml*) specifies a set of point and/or polygon features that are described by their latitude, longitude, and altitude in that order. Other features such as images, 3D models, textual descriptions, and so on can also be specified but won't be discussed here. With regards to spatial data, other designations can make the view more specific, such as tilt, heading, and altitude, which together define a "camera view". For its structure, KML is very similar to HTML (which is used for webpage creation) in that it utilizes a series of opening and closing tags to describe an attribute. A simple KML file that creates a placemark (point features in GE are known as placemarks) for Old Town Albuquerque, New Mexico is shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
  <Placemark>
    <name>Old Town Albuquerque</name>
    <styleUrl>#msn_ylw-pushpin</styleUrl>
    <Point>
    <coordinates>
      -106.6699003211037,35.09608350388626,0
    </coordinates>
    </Point>
  </Placemark>
</Document>
</kml>
```

Figure 3 - Sample KML file for a point feature (placemark) locating Old Town Albuquerque, New Mexico in Google Earth.

Polygons are defined by listing a series of points that when connected, describe the boundary of the polygon. Figure 4 is an example of a KML file that places a polygon over the Old Town Square in Albuquerque. The polygon attributes are defined by creating a 'style id' that describes the boundary line and polygon fill colors and opacity values, and then referencing the 'style id' using the 'styleUrl' tag within the definition of the polygon. The code in Figure 4 creates a polygon with a solid blue outline and a 65% opacity red fill. The details of these designations are discussed below. A comprehensive KML reference can be found at http://code.google.com/apis/kml/documentation/kmlreference.html.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>

  <Style id="old_town_poly">
    <LineStyle>
      <color>ffff0000</color>
      <width>2</width>
    </LineStyle>
    <PolyStyle>
      <color>a60000ff</color>
    </PolyStyle>
  </Style>

  <Placemark>
    <name>Old Town Albuquerque</name>
    <styleUrl>#old_town_poly</styleUrl>
    <Polygon>
      <tessellate>1</tessellate>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates>
              -106.670318936411,35.09632483935854,0
              -106.6703665689491,35.09590064534395,0
              -106.6695078242527,35.09587266699027,0
              -106.6694674295317,35.09630015648485,0
              -106.670318936411,35.09632483935854,0
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
</Document>
</kml>
```

Figure 4 - Sample code for 4 vertex polygon centered over Old Town Albuquerque, NM. Note that five coordinate points are defined in order to close the loop (the last coordinate is the same as the first).

VBSCRIPT, KML, and Powersim

Powersim has the ability to define an auxiliary using vbscript, which allows a user to create unique functions that are not part of the Powersim library. The use of vbscript also provides a means for Powersim to communicate with the 'outside world' (beyond its ability to work with Microsoft Excel). The GE implementation is done by using vbscript to create and update KML files. While GE is intended to operate in a client/server environment where the data exist on a separate server, the implementation illustrated here assumes that the model, GE, and the data all exist on the same computer. This allows for remote demonstrations of the models where no internet connection is available but, as will be discussed below, comes with a cost of increased computational overhead.

Four separate KML files are created: a link file, a style file, a legend file, and a boundary file. The link and style files are only written once at the beginning of a simulation. The legend and boundary files are updated at each timestep to reflect changes in the underlying data. The link file creates a 'network link' that tells GE what files to open and how often to refresh them. Two separate network links are created, one for the boundary file and one for the legend file. In a client/server set-up, the network links would consist of a server address (i.e. URL) and file name. Since no server is being used in this case, only the file name need be defined. To allow it to only be written at the start of the simulation, an 'IF' statement along with the Powersim Boolean function, 'ATSTART()' is used.

The style file defines the line color as well as the polygon fill color and transparency for each color bin. Colors are defined in GE using a Hexadecimal code of the form TTBBGGRR, where TT is the two digit hexadecimal code for the transparency, and BB, GG, and RR, are the two digit hexadecimal codes of the blue, green, and red components of the color (note that the common order is TTRRGGBB). Hexadecimal codes use a combination of integers (0 thru 9) and letters (A thru F), with one byte (i.e. two digit code) representing a number that varies with 256 steps in the range of 00 to FF (or 0 to 255 in decimal notation), with 00 having the least intensity. Transparency is similarly defined with 00 being fully transparent. A light blue color with 25% transparency would have the hexadecimal code, 41FF8D8D. An issue arises when using Powersim since alpha characters cannot be passed to the vbscript code. To circumvent this issue, we pass the ASCII character code to the script and then use the 'Chr(dd)' designation to write the character. For the letters A to Z, the ASCII codes vary from 65 to 90. For the numbers 0 through 9, the ASCII codes vary from 48 to 57. Thus the letter 'D' would be printed using Chr(68) in the vbscript write statement. To pass a hexadecimal code to vbscript, 8 numbers need to be passed. For example, the following 8 ASCII codes define the light blue hexadecimal color above: 52, 52, 70, 70, 56, 68, 56, 68. This could be avoided by hardwiring each color bin into the vbscript code but this restricts the flexibility to change color scales or the number of bins. Table 1 shows a sample color scale going from blue to red in 11 steps as well as corresponding the decimal and hexadecimal codes, and the ASCII code version of the hexadecimal values.

The boundary files describe the outline of each polygon in the simulation. In our implementation, we begin with a boundary file template that describes all the polygons in the simulation. Within the Powersim auxiliary that creates the boundary file, it first reads the template and then copies it to a file of the same name that is listed in the link file. This setup allows for the user to visualize a subset of the polygons, such as visualizing the counties in one state, instead of every county in every state. Within the boundary file, the 'styleUrl' tag is used to refer to the appropriate color style in the style file. The appropriate color style is determined by the polygons' simulated attribute (i.e. the value that is changing during the simulation) and where the attribute value falls in relation to the full range of values. The 'styleUrl' name is updated in the boundary file at each timestep.

Since the current setup keeps the data and simulations on a local machine, the entire boundary file must be re-written at each timestep since vbscript lacks the ability to only change a single line within a file. This can become computationally burdensome for simulations that include lots of polygons. The way around this is to use a client/server setup where the KML 'Update' tag can be used to update specific parts of a KML file. For computer security purposes, the 'Update' tag cannot be used to alter local files and thus is not used in the current configuration.

Table 1 - Sample color scale with 11 bins. Note that for the Hexadecimal and ASCII codes, the color order is blue-green-red while for the decimal color it is red-green-blue.

Color Example	Decimal Color (RGB)	Hexadecimal Code (BBGGRR)	ASCII (BBGGRR)
🟦	0, 0, 255	FF0000	70, 70, 48, 48, 48, 48
🟦	0, 102, 153	996600	57, 57, 54, 54, 48, 48
🟩	0, 204, 51	33CC00	51, 51, 67, 67, 48 ,48
🟩	51, 255, 0	00FF33	48, 48, 70, 70, 51, 51
🟩	153, 255, 0	00FF99	48, 48, 70, 70, 57, 57
🟨	255, 255, 0	00FFFF	48, 48, 70, 70, 70, 70
🟧	255, 194, 0	00C1FF	48, 48, 67, 49, 70, 70
🟧	255, 133, 0	0084FF	48, 48, 56, 52, 70, 70
🟧	255, 82, 0	0051FF	48, 48, 53, 49, 70, 70
🟥	255, 41, 0	0028FF	48, 48, 50, 56, 70, 70
🟥	255, 0, 0	0000FF	48, 48, 48, 48, 70, 70

To create the legend, a series of small polygons (small enough to not show up on the map) were created that were each assigned a color from the color scale. The colored polygons and the corresponding bin value shows up in the 'Places' window in GE when the link file is loaded. The legend file is updated each timestep to allow for changes in the color scheme and/or range of values and number of bins. This is especially useful if one is visualizing more than one attribute for a particular set of polygons such as population and energy use on a per county basis. If it is known that the legend will remain static then it only needs to be written to once at the beginning of the simulation.

To use the visualization, GE is opened alongside Powersim at the start of the simulation. The link file is then opened from within GE and the model simulation is begun. The simulation speed of the model needs to be limited to be greater than or equal to the refresh rate of the boundary file. 'IF' statements and other conditional operators can be used to update the KML boundary file every n^{th} timestep or when certain events occur or conditions are met. Powersim requires a value to be assigned to each auxiliary, which means that a "Result = X" must be placed within the vbscript code, with the X being some numerical value. Since the auxiliary is only being used to generate output, the value of X is immaterial. Figure 5 shows a screen shot of a demonstration model that simulates changing values for watershed polygons for the whole country. This model has the ability to simulate a subset of the polygons by specifying beginning and ending watershed numbers (controlled by slider bars). A random number generator is used to generate a random number between two user specified numbers and assign a number to each watershed. The 'color bins' auxiliary creates the bin thresholds between the two numbers by which the watershed values can be slotted. Figure 6 shows a screen shot of the GE visualization from the demonstration model at two consecutive timesteps that uses a color scale that varies from white to blue in 20 steps (i.e. bins).

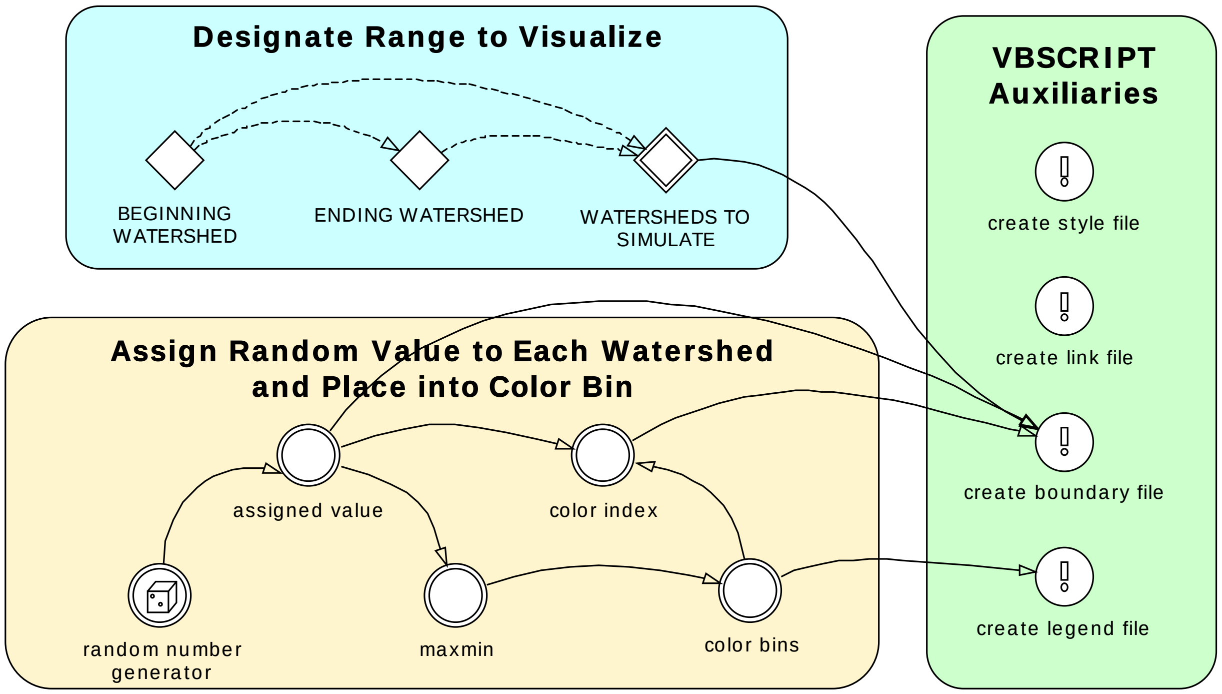

Figure 5 - Demonstration model for illustrating spatial visualization with Google Earth. The simulation assigns a random number to each of the major watersheds in the use at each timstep. The display color of the watershed is dependent on the value of the random number.

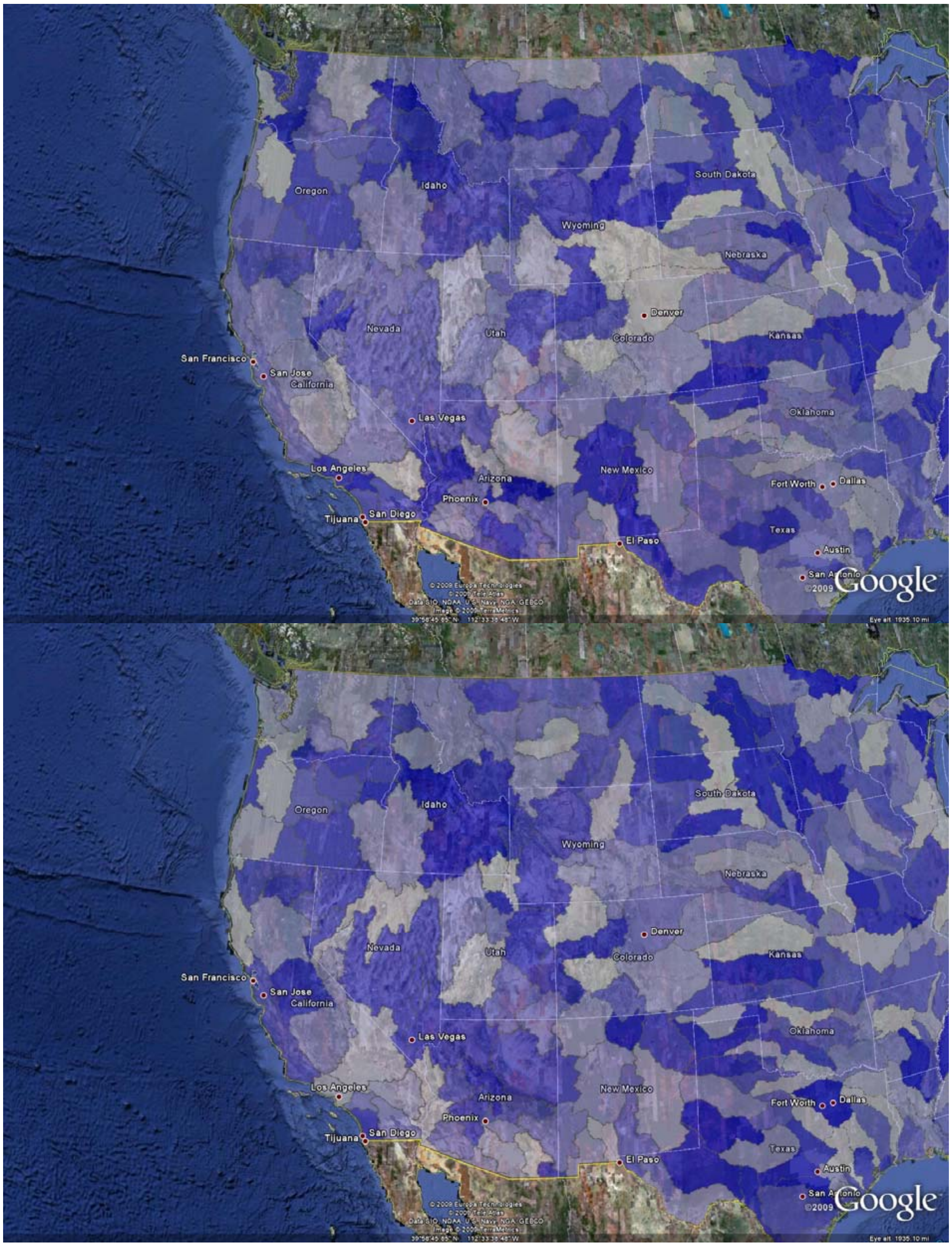

Figure 6 - Screen shots of the Google Earth visualization at two successive timesteps for each major watershed across the western US.

References

Anselin, L., I. Syabri, et al. (2004). GeoDa: An Introduction to Spatial Data Analysis. University of Illinois, Urbana-Champaign, Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics.

Dykes, J. A. (1997). "Exploring spatial data representation with dynamic graphics." Computers & Geosciences **23**(4): 345-370.

Forrester, J. W. (1971). Principles of Systems. Cambridge, MA, Wright-Allen.

Forrester, J. W. (1987). "Obvious Truths." System Dynamics Review **3**(2): 156-159.

Miezka, S. F. L. and A. Größler (2004). Merger Dynamics - A System Dynamics Analysis of Post-Merger Integration Processes. System Dynamics Society 22nd International Conference, Oxford, England.

Volterra, V. (1931). Variations and fluctuations of the number of individuals in animal species living together. Tranlated from 1928 edition by R.N. Chapman., McGraw-Hill.

Appendix A – vbscript code examples from the watershed demonstration model (see Figure 5 for an illustration of the model layout)

Blue text indicates comments within the code.

Powersim auxiliary code for writing the style file. This only has to be executed at the start of the simulation. The “dot” and “w2b” variables are arrays of ASCII code values defining the common file extension ‘.kml’ and a 21 step color scale that ranges from white to blue. The transparency is hardwired to be about 80% (hexadecimal value ‘C8’).

```
VBFUNCTION( | "dot" = 'FILE EXTENSION', "w2b" = 'WHITE TO BLUE' |
        "Dim fso, kmlOut, kmlTemp"
        "Set fso = CreateObject(\"Scripting.FileSystemObject\")"
        "fend = Chr(\+dot(0)\) & Chr(\+dot(1)\) & Chr(\+dot(2)\) & Chr(\+dot(3)\)" //File extension (i.e. '.kml')
        "fname = \"watershed_zones\" & fend"  //State kml file name
        "Set kmlOut = fso.CreateTextFile(\+fname\, 8, 0)"  //Open master kml file for writing
        "kmlOut.WriteLine(\"<\" & Chr(63) & \"xml version=\" & Chr(34) & \"1.0\" & Chr(34) & \" encoding=\" &
        Chr(34) & \"UTF-8\" & Chr(34) & Chr(63) & \">\")"
        "kmlOut.WriteLine(\"<kml xmlns=\" & Chr(34) & \"http://earth.google.com/kml/2.2\" & Chr(34) & \">\")"
        "kmlOut.WriteLine(\"<Document>\")"
        "fzone = -1"
        "bsize = UBound(w2b)" //Find the number of color bins
        "For i = 0 to bsize"  //Loop through each color bin and define a unique style
                "fzone = fzone + 1"
                "kmlOut.WriteLine(\"  <Style id=\" & Chr(34) & \"zn\" & \+fzone\ & Chr(34) & \">\")" //Supply a
        unique style name
                "kmlOut.WriteLine(\"    <LineStyle>\")"
                "kmlOut.WriteLine(\"      <color>\")"
                "kmlOut.WriteLine(\"        C8505050\")"
                "kmlOut.WriteLine(\"      </color>\")"
                "kmlOut.WriteLine(\"      <width>\")"
                "kmlOut.WriteLine(\"        2\")"
                "kmlOut.WriteLine(\"      </width>\")"
                "kmlOut.WriteLine(\"    </LineStyle>\")"
                "kmlOut.WriteLine(\"    <PolyStyle>\")"
                "kmlOut.WriteLine(\"      <color>\")"
//Color code output needs to be in ttbbggrr for GE
                "kmlOut.WriteLine(\"        C8\" & Chr(\+w2b(i,0)\) & Chr(\+w2b(i,1)\) & Chr(\+w2b(i,2)\) &
        Chr(\+w2b(i,3)\) & Chr(\+w2b(i,4)\) & Chr(\+w2b(i,5)\))"
                "kmlOut.WriteLine(\"      </color>\")"
                "kmlOut.WriteLine(\"    </PolyStyle>\")"
                "kmlOut.WriteLine(\"  </Style>\")"
        "next"
        "Result = 2" //This line is needed to provide a value to the auxiliary
        "kmlOut.WriteLine(\"</Document>\")"
        "kmlOut.WriteLine(\"</kml>\")"
)
```

Powersim auxiliary code for writing the link file. This only has to be executed at the start of the simulation. The “dot” variable is an array of ASCII code values defining the common file extension ‘.kml’.   The “refresh” variable is the refresh rate of the boundary file in seconds. The refresh rate of the legend is hardwired to be 2.5 seconds.

.

```
VBFUNCTION( | "stime" = ATSTART(), "dot" = 'FILE EXTENSION', "refresh" = 'REFRESH INTERVAL' |
        "Dim fso, kmlOut"
        "Set fso = CreateObject(\"Scripting.FileSystemObject\")"
        "fend = Chr(\+dot(0)\) & Chr(\+dot(1)\) & Chr(\+dot(2)\) & Chr(\+dot(3)\)" //File extension (i.e. '.kml')
        "fname = \"Link_Watersheds\" & fend"
        "If fso.FileExists(\+fname\) then" //Clean-up directory
                "fso.DeleteFile(\+fname\)"
        "end if"
        "Set kmlOut = fso.CreateTextFile(\+fname\, 8, 0)"  //Create link file
        "kmlOut.WriteLine(\"<\" & Chr(63) & \"xml version=\" & Chr(34) & \"1.0\" & Chr(34) & \" encoding=\" &
        Chr(34) & \"UTF-8\" & Chr(34) & Chr(63) & \">\")"
        "kmlOut.WriteLine(\"<kml xmlns=\" & Chr(34) & \"http://earth.google.com/kml/2.2\" & Chr(34) & \">\")"
        "kmlOut.WriteLine(\"  <Document>\")"
        "kmlOut.WriteLine(\"  <open>\")"
        "kmlOut.WriteLine(\"    1\")"
        "kmlOut.WriteLine(\"  </open>\")"
        "kmlOut.WriteLine(\"  <name>\")"
        "kmlOut.WriteLine(\"    Watersheds\")"
        "kmlOut.WriteLine(\"  </name>\")"
        "kmlOut.WriteLine(\"    <NetworkLink>\")"
        "kmlOut.WriteLine(\"      <name>\")"
        "kmlOut.WriteLine(\"        Simulated Watersheds\")"
        "kmlOut.WriteLine(\"      </name>\")"
        "kmlOut.WriteLine(\"      <Link>\")"
        "kmlOut.WriteLine(\"        <href>\")"
        "kmlOut.WriteLine(\"          Watersheds\" & \+fend\)"
        "kmlOut.WriteLine(\"        </href>\")"
        "kmlOut.WriteLine(\"        <refreshMode>\")"
        "kmlOut.WriteLine(\"          onInterval\")"
        "kmlOut.WriteLine(\"        </refreshMode>\")"
        "kmlOut.WriteLine(\"        <refreshInterval>\")"
        "kmlOut.WriteLine(\"          \" & \+refresh\)"
        "kmlOut.WriteLine(\"        </refreshInterval>\")" //Refresh rate
        "kmlOut.WriteLine(\"      </Link>\")"
        "kmlOut.WriteLine(\"    </NetworkLink>\")"
        "kmlOut.WriteLine(\"    <NetworkLink>\")"
        "kmlOut.WriteLine(\"    <open>\")"
        "kmlOut.WriteLine(\"      1\")"
        "kmlOut.WriteLine(\"    </open>\")"
        "kmlOut.WriteLine(\"    <name>\")"
        "kmlOut.WriteLine(\"      Legend\")"
        "kmlOut.WriteLine(\"    </name>\")"
        "kmlOut.WriteLine(\"      <Link>\")"
        "kmlOut.WriteLine(\"        <href>\")"
        "kmlOut.WriteLine(\"          legend_zones\" & \+fend\)"
        "kmlOut.WriteLine(\"        </href>\")"
        "kmlOut.WriteLine(\"        <refreshMode>\")"
        "kmlOut.WriteLine(\"         onInterval\")"
        "kmlOut.WriteLine(\"       </refreshMode>\")"
        "kmlOut.WriteLine(\"       <refreshInterval>\")"
        "kmlOut.WriteLine(\"         2.5\")" //Refresh rate hard wired to be 2.5 seconds.
        "kmlOut.WriteLine(\"       </refreshInterval>\")"
        "kmlOut.WriteLine(\"     </Link>\")"
        "kmlOut.WriteLine(\"   </NetworkLink>\")"
        "kmlOut.WriteLine(\"  </Document>\")"
        "kmlOut.WriteLine(\"</kml>\")"
        "Result = 1"
        )
```

Powersim auxiliary code for writing the boundary file. This is executed at each timestep. The passed variables, “wsnum”, “np”, “w2s”, “bi”, and “ei” provide the ability to visualize a subset of all the watersheds (i.e. polygons). The “ci” variable is the color bin that each watershed falls into and the “crit” variable are the actual values. The template file is a KML file that contains the boundary information for every polygon. It is read and copied as needed to create the boundary file.

```
VBFUNCTION( | "wsnum" = 'WATERSHED NUMBER', "np" = 'NUMBER OF WATERSHED POLYGONS', "ci" = 'color
        index', "w2s" = 'WATERSHEDS TO SIMULATE',
        "bi" = 'BEGINNING WATERSHED INDICE', "ei" = 'ENDING WATERSHED INDICE', "crit" = 'assigned value',
        "tod" = MONTH() |
        "Dim fso, kmlOut, kmlTemp"
        "Set fso = CreateObject(\"Scripting.FileSystemObject\")"
        "asize = UBound(wsnum)" //Find out size of array that you want to loop through
        "fend = Chr(46) & Chr(107) & Chr(109) & Chr(108)" //File extension (i.e. '.kml')
        "fname = \"Watersheds\" & fend"  //kml file name
        "Set kmlOut = fso.CreateTextFile(\+fname\, 8, 0)"  //Open watershed output file for writing
        "fin = \"Watersheds_master\" & fend"  //watersheds_template.kml file name
        "Set kmlIn = fso.OpenTextFile(\+fin\, 1, 0)"  //Open watersheds_template.kml file
        "for i = 0 to 3"
                "text = kmlIn.ReadLine"
                "kmlOut.WriteLine(\+text\)" //Rewrite the first 4 lines of the template file
        "next"
        "for i = 0 to 74" //Skip the next 75 lines of the template file
                "kmlIn.SkipLine"
        "next"
        "if w2s(0) = 0 then" //if w2s(0) = 0, then simulate all watersheds
                "for i = 0 to asize"
                        "for iline = 0 to 29"
                                "kmlIn.SkipLine" //Skip the next 30 line of the template file
                        "next"
                        "fzone = ci(i)"
                        "kmlOut.WriteLine(\"    <Placemark>\")"
                        "kmlOut.WriteLine(\"     <name>\")"
                        "kmlOut.WriteLine(\"       \" & wsnum(i) & \" - \" & crit(i))"
                        "kmlOut.WriteLine(\"     </name>\")"
                        "kmlOut.WriteLine(\"     <styleUrl>\")"
//Write the styleUrl.  This is the unique style name defined in the style file.
                        "kmlOut.WriteLine(\"       watershed_zones.kml\" & Chr(35) & \"zn\" & \+fzone\)"
                        "kmlOut.WriteLine(\"     </styleUrl>\")"
                        "kmlOut.WriteLine(\"     <MultiGeometry>\")"
                        "if np(i) > 1 then"
                                "kmlOut.WriteLine(\"       <MultiGeometry>\")"
                                "kmlIn.SkipLine"
                        "end if"
                        "for j = 0 to np(i) - 1"
                                "for iline = 0 to 3"
                                        "kmlIn.SkipLine" //Skip 4 more lines in the template file
                                "next"
                                "text = kmlIn.ReadLine" //Read the polygon coordinates from the template file
                                "kmlOut.WriteLine(\"       <Polygon>\")"
                        "kmlOut.WriteLine(\"        <altitudeMode>\")"
                        "kmlOut.WriteLine(\"          clampToGround\")"
                        "kmlOut.WriteLine(\"        </altitudeMode>\")"
                        "kmlOut.WriteLine(\"        <tessellate>\")"
                        "kmlOut.WriteLine(\"          1\")"
                        "kmlOut.WriteLine(\"        </tessellate>\")"
                        "kmlOut.WriteLine(\"        <outerBoundaryIs>\")"
                        "kmlOut.WriteLine(\"          <LinearRing>\")"
                        "kmlOut.WriteLine(\"            <coordinates>\")"
                        "kmlOut.WriteLine(\"              \" & \+text\)" //Write polygon coordinates
                        "kmlOut.WriteLine(\"          </LinearRing>\")"
                        "kmlOut.WriteLine(\"        </outerBoundaryIs>\")"
                        "kmlOut.WriteLine(\"      </Polygon>\")"
                        "for iline = 0 to 2"
                                "kmlIn.SkipLine"
                        "next"
                "next"
                "if np(i) > 1 then"
                        "kmlOut.WriteLine(\"      </MultiGeometry>\")"
                        "kmlIn.SkipLine"
                "end if"
                "kmlOut.WriteLine(\"      </MultiGeometry>\")"
                "kmlOut.WriteLine(\"    </Placemark>\")"
                "kmlIn.SkipLine"
                "kmlIn.SkipLine"
        "next"
        "kmlOut.WriteLine(\"  </Document>\")"
        "kmlOut.WriteLine(\"</kml>\")"
        "Result = 1"
"else" //Else if we are only visualizing a subset, do this loop
        "if w2s(0) = w2s(1) then"
                "i2 = bi - 1"
        "else"
                "i2 = ei - 1"
        "end if"
        "i1 = bi - 1"
        "for i = 0 to asize"
                "if i < i1 or i > i2 then"
                        "for iline = 0 to 29"
                                "kmlIn.SkipLine"
                        "next"
                        "if np(i) > 1 then"
                                "kmlIn.SkipLine"
                        "end if"
                        "for j = 0 to np(i) - 1"
                                "for iline = 0 to 7"
                                        "kmlIn.SkipLine"
                                "next"
                        "next"
                        "if np(i) > 1 then"
                                "kmlIn.SkipLine"
                        "end if"
                        "kmlIn.SkipLine"
```

```
                    "kmlIn.SkipLine"
                "else"
                    "for iline = 0 to 29"
                        "kmlIn.SkipLine"
                    "next"
                    "fzone = ci(i)"
                    "kmlOut.WriteLine(\"    <Placemark>\")"
                    "kmlOut.WriteLine(\"      <name>\")"
                    "kmlOut.WriteLine(\"        \" & wsnum(i))"
                    "kmlOut.WriteLine(\"      </name>\")"
                    "kmlOut.WriteLine(\"      <styleUrl>\")"
                    "kmlOut.WriteLine(\"        watershed_zones.kml\" & Chr(35) & \"zn\" &
\+fzone\)"
                    "kmlOut.WriteLine(\"      </styleUrl>\")"
                    "kmlOut.WriteLine(\"      <MultiGeometry>\")"
                    "if np(i) > 1 then"
                        "kmlOut.WriteLine(\"        <MultiGeometry>\")"
                        "kmlIn.SkipLine"
                    "end if"
                    "for j = 0 to np(i) - 1"
                        "for iline = 0 to 3"
                            "kmlIn.SkipLine"
                        "next"
                        "text = kmlIn.ReadLine"
                        "kmlOut.WriteLine(\"        <Polygon>\")"
                        "kmlOut.WriteLine(\"          <altitudeMode>\")"
                        "kmlOut.WriteLine(\"            clampToGround\")"
                        "kmlOut.WriteLine(\"          </altitudeMode>\")"
                        "kmlOut.WriteLine(\"          <tessellate>\")"
                        "kmlOut.WriteLine(\"            1\")"
                        "kmlOut.WriteLine(\"          </tessellate>\")"
                        "kmlOut.WriteLine(\"          <outerBoundaryIs>\")"
                        "kmlOut.WriteLine(\"            <LinearRing>\")"
                        "kmlOut.WriteLine(\"              <coordinates>\")"
                        "kmlOut.WriteLine(\"                \" & \+text\)"
                        "kmlOut.WriteLine(\"            </LinearRing>\")"
                        "kmlOut.WriteLine(\"          </outerBoundaryIs>\")"
                        "kmlOut.WriteLine(\"        </Polygon>\")"
                        "for iline = 0 to 2"
                            "kmlIn.SkipLine"
                        "next"
                    "next"
                    "if np(i) > 1 then"
                        "kmlOut.WriteLine(\"        </MultiGeometry>\")"
                        "kmlIn.SkipLine"
                    "end if"
                    "kmlOut.WriteLine(\"      </MultiGeometry>\")"
                    "kmlOut.WriteLine(\"    </Placemark>\")"
                    "kmlIn.SkipLine"
                    "kmlIn.SkipLine"
                "end if"
            "next"
            "kmlOut.WriteLine(\"  </Document>\")"
```

```
                    "kmlOut.WriteLine(\"</kml>\")"
                    "Result = tod" //Assign value to the auxiliary.
          "end if"
)
```

Powersim auxiliary code for writing the legend file. This is executed at each timestep. The "dot" and "w2b" variables are arrays of ASCII code values defining the common file extension '.kml' and a 21 step color scale that ranges from white to blue. The variable "cb" is an array of the cutoff values for each color bin.

```
VBFUNCTION( | "dot" = 'FILE EXTENSION', "w2b" = 'WHITE TO BLUE', "cb" = 'color bins' |
        "Dim fso, kmlOut, kmlTemp"
        "Set fso = CreateObject(\"Scripting.FileSystemObject\")"
        "fend = Chr(\+dot(0)\) & Chr(\+dot(1)\) & Chr(\+dot(2)\) & Chr(\+dot(3)\)" //File extension (i.e. '.kml')
        "fname = \"legend_zones\" & fend"  //legend file name
        "asize = UBound(cb)"
        "Set kmlOut = fso.CreateTextFile(\+fname\, 8, 0)"  //Open master kml file for writing
        "kmlOut.WriteLine(\"<\" & Chr(63) & \"xml version=\" & Chr(34) & \"1.0\" & Chr(34) & \" encoding=\" &
        Chr(34) & \"UTF-8\" & Chr(34) & Chr(63) & \">\")"
        "kmlOut.WriteLine(\"<kml xmlns=\" & Chr(34) & \"http://earth.google.com/kml/2.2\" & Chr(34) & \">\")"
        "kmlOut.WriteLine(\"<Document>\")"
        "kmlOut.WriteLine(\"  <visibility>\")"
        "kmlOut.WriteLine(\"    0\")"
        "kmlOut.WriteLine(\"  </visibility>\")"
        "kmlOut.WriteLine(\"  <open>\")"
        "kmlOut.WriteLine(\"    1\")"
        "kmlOut.WriteLine(\"  </open>\")"
        "kmlOut.WriteLine(\"  <visible>\")"
        "kmlOut.WriteLine(\"    0\")"
        "kmlOut.WriteLine(\"  </visible>\")"
        "kmlOut.WriteLine(\"  <name>\")"
        "kmlOut.WriteLine(\"    Legend\")"
        "kmlOut.WriteLine(\"  </name>\")"
        "fzone = -1"
        "for i = 0 to asize" //Loop through and define a style based on each color bin
                "fzone = fzone + 1"
                "kmlOut.WriteLine(\"    <Style id=\" & Chr(34) & \"legend_\" & \+fzone\ & Chr(34) & \">\")"
                "kmlOut.WriteLine(\"      <LineStyle>\")"
                "kmlOut.WriteLine(\"        <color>\")"
                "kmlOut.WriteLine(\"          C8505050\")"
                "kmlOut.WriteLine(\"        </color>\")"
                "kmlOut.WriteLine(\"        <width>\")"
                "kmlOut.WriteLine(\"          2\")"
                "kmlOut.WriteLine(\"        </width>\")"
                "kmlOut.WriteLine(\"      </LineStyle>\")"
                "kmlOut.WriteLine(\"      <PolyStyle>\")"
                "kmlOut.WriteLine(\"        <color>\")"
                "kmlOut.WriteLine(\"          ff\" & Chr(\+w2b(i,0)\) & Chr(\+w2b(i,1)\) & Chr(\+w2b(i,2)\) &
        Chr(\+w2b(i,3)\) & Chr(\+w2b(i,4)\) & Chr(\+w2b(i,5)\))"
                "kmlOut.WriteLine(\"        </color>\")"
                "kmlOut.WriteLine(\"      </PolyStyle>\")"
                "kmlOut.WriteLine(\"    </Style>\")"
        "next"
        "fzone = -1"
        "for i = 0 to asize"            //Loop through and create a small legend polygon for each color bin
                "fzone = fzone + 1"
                "kmlOut.WriteLine(\"  <Placemark>\")"
                "kmlOut.WriteLine(\"    <name>\")"
                "kmlOut.WriteLine(\"      \" & cb(i))"
                "kmlOut.WriteLine(\"    </name>\")"
                "kmlOut.WriteLine(\"    <visibility>\")"
                "kmlOut.WriteLine(\"      0\")"
                "kmlOut.WriteLine(\"    </visibility>\")"
                "kmlOut.WriteLine(\"    <styleUrl>\")"
                "kmlOut.WriteLine(\"      \" & Chr(35) & \"legend_\" & \+fzone\)"
                "kmlOut.WriteLine(\"    </styleUrl>\")"
                "kmlOut.WriteLine(\"        <Polygon>\")"
                "kmlOut.WriteLine(\"          <altitudeMode>\")"
                "kmlOut.WriteLine(\"            clampToGround\")"
                "kmlOut.WriteLine(\"          </altitudeMode>\")"
                "kmlOut.WriteLine(\"          <tessellate>\")"
                "kmlOut.WriteLine(\"            1\")"
                "kmlOut.WriteLine(\"          </tessellate>\")"
                "kmlOut.WriteLine(\"          <outerBoundaryIs>\")"
                "kmlOut.WriteLine(\"            <LinearRing>\")"
                "kmlOut.WriteLine(\"              <coordinates>\")"
                "kmlOut.WriteLine(\"                -106.2859,35.0880917,0 -106.2857,35.0880917,0 -
106.2857,35.0880028,0 -106.2859,35.0880028,0 -106.2859,35.0880917,0\")"
                "kmlOut.WriteLine(\"              </coordinates>\")"
                "kmlOut.WriteLine(\"            </LinearRing>\")"
                "kmlOut.WriteLine(\"          </outerBoundaryIs>\")"
                "kmlOut.WriteLine(\"        </Polygon>\")"
                "kmlOut.WriteLine(\"    </Placemark>\")"
        "next"
        "kmlOut.WriteLine(\"  </Document>\")"
        "kmlOut.WriteLine(\"</kml>\")"
        "Result = 2" //Assign value to the auxiliary
)
```