2006-07-12

Random City Generator Technical Report

Brandon Call
brcall@byu.net

RANDOM CITY GENERATOR TECHNICAL REPORT

BRANDON R. CALL

ABSTRACT. The Brigham Young University (BYU) Multiple Agent Intelligent Coordination and Control (MAGICC) research lab researchs many aspects of small unmanned air vehicles (UAV). To facilitate this research, students have created a UAV simulator called Aviones. In order to increase the capability of Aviones to simulate an urban environment, the ability to draw buildings, streets and vehicles have been added. This document outlines how cities are created and how a researcher can use them in conjunction with Aviones.

1. Introduction

The Brigham Young University (BYU) Multiple Agent Intelligent Coordination and Control (MAGICC) research lab researchs many aspects of small unmanned air vehicles (UAV). Some of the research projects currently under development are video stabilization, video mosaicing, obstacle avoidance, path planning, fire surveillance, convoy surveillance, road following, multiple video stream use, multiple vehicle coordination, multiple vehicle user interface, attitude estimation, and adaptive controllers for small UAVs. To facilitate this research, students have created a UAV simulator called Aviones. In order to increase the capability of Aviones to simulate an urban environment, the ability to draw buildings, streets and vehicles have been added. This document outlines how cities are created and how a researcher can use them in conjunction with Aviones and provides implementation details about the Random City Generator program.

The random city generator was initially began as a project for ME 570 - Computer-Aided Engineering Software. This project was chosen because it was desirable for both path planning and computer vision research. The cities created by the random city generator can be loaded by the path planner and used for path planning as well as loaded into Avoines and displayed in the environment.

In addition to cities, targets were added to Aviones. A generic target is a sphere that can be placed anywhere in the environment and can move in various ways. A Utah Transportation Authority (UTA) bus has also been added and can move in various ways. These targets are not part of random city maker program, but are specified in an XML file and loaded into Aviones separately. See Section 4 for a discussion of Aviones targets.

2. How to create/use cities

Cities are created using the Random City Maker program. This program was written using the Qt user interface library (version 3.2.0 Educational) and has been compiled for Microsoft Windows. The random city maker creates a city with blocks

Key words and phrases. documentation, software, simulation, city, building, aviones, path planning, computer vision, UAV.

and streets in a spoke wheel layout and textures the city buildings with photos of real world buildings. The city is saved as an xml file with a .cit extension. This section describes how to use the Random City maker application to create cities that can be used in Aviones and the Path Planner.

2.1. **The Random City Maker User Interface.** The user interface for the Random City Maker is a dockable window environment. Dockable windows give the user the ability to configure the user interface in a way that works the best for them. The windows that can be docked are

- City Properties - a window containing the variables that can be tuned when creating random cities
- Overhead view - a window displaying the overhead view of the city with North upward. This window is used to select buildings
- Building Properties - a window containing the properties of the currently selected building
- Music Controls - a window that can be used to play music while you create cities. This was added as a joke for the ME 570 teacher. It demonstrates how easy Qt is to use because this entire widget took only a couple of hours to code and debug.

In addition to the dockable windows, the main view contains a three-dimensional view of the city that can be rotated and zoomed. Figure 1 shows a screen shot of the dockable windows and the main view. The main view is moved using either mouse button and moving the mouse. To rotate the city, click either the right or left mouse button and move the mouse left to right. To zoom in and out, hold the right mouse button and move the mouse up and down. To change from pedestrian view to overhead view, use the mouse wheel or hold the left mouse button and move the mouse up and down.

2.2. **Create a city.** Creating a new city with the random city maker is as easy as clicking the Generate Random City button on the city properties window or by choosing Generate Random City from the City menu. Either of these actions will create a new random city and display it to the user. The city is created using the settings in the city properties window. The city is layed out with major roads as spokes extending from the center and minor roads creating the grid between spokes. City block size determines how wide the buildings are. Max building height is the height of the tallest possible building in the city. This doesn't guarantee that there will be a building of this height, but sets a limit that nothing will be higher than this height. When modifying this parameter, realize that the tallest building in the world is 500 meters. Size X is the size of the whole city in the East direction and size Y is the size of the city in the North direction. The height function drop down box has three options for the height function used to determine the skyline, gaussian, exponential, and bimodal-gaussian. Each height function uses the height function parameter and the size of the city to determine how sharply the building heights fall off. Values proven to give reasonable results are summarized in Table 1.

After creating a city, each building can be altered individually. The building is selected by clicking on the building in the overhead view. When the building is clicked the building's properties appear in the building properties window. Each property is visibly updated when the user enters text and then pushes enter. The texture for the building is specified by the drop down box.

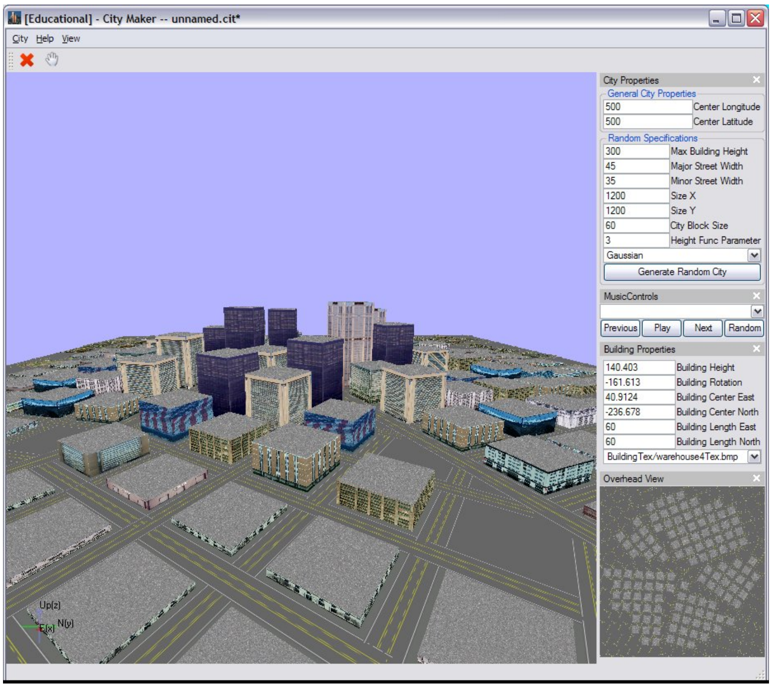

FIGURE 1. A screen shot of the Random City Maker

Height Function	Value
Gaussian	3
Bi-Modal Gaussian	3
Exponential	8

TABLE 1. The values of the height function parameter that give realistic results.

After the user has obtained the desired look, the city can be saved as an XML *.cit file by choosing Save City from the City menu. XML provides a clean, readable file that is also easy for a computer to parse. The XML file is stored with a base city tag containing neighborhood tags and road tags. Neighborhood tags contain road tags and building tags. There are two types of building tags, buildings and flat-buildings.

3. Using the City files for research

3.1. **How to use cities in Aviones.** To load a city in the BYU MAGICC lab UAV simulator, Aviones, choose Load City from the File menu. The program will load the textures onto the graphics card and display the city in the environment. The buildings are drawn on top of the terrain starting at the height of the lowest corner. Roads are divided into small triangles and each segment is drawn slightly above the terrain at that location. Figure 2 shows a city that has been loaded into Aviones. The building textures can be downloaded from the MAGICC lab group space at 'groups/magiccuav/Projects/Random City Maker/textures.zip'. These textures are required to be in the working directory of Aviones in a folder called textures and must have the same subdirectory structure as contained in the textures.zip file on the group space. Adding a city and then changing the terrain map has not been tested and may give undesirable results.

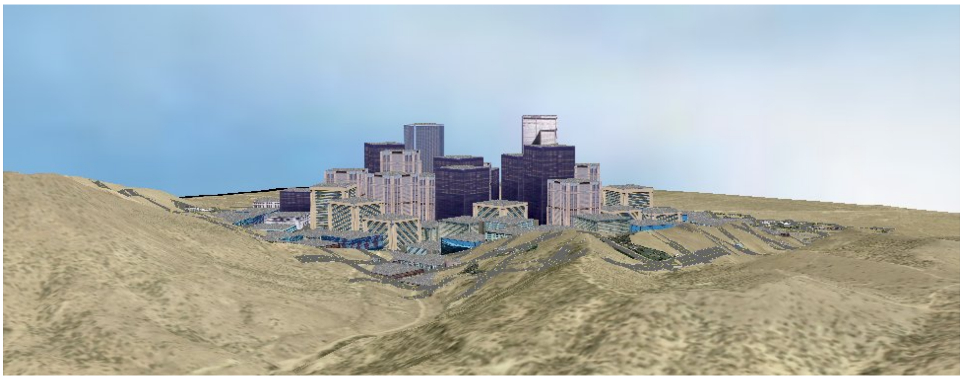

Figure 2. A screen shot of a city loaded into the simulator Aviones and drawn on top of a real terrain model of the west desert.

The cities east and north location as stored in the xml file is a feature that is yet to be implemented in Aviones. It was planned to be a relative coordinate for the city so that it may be drawn off center from the terrain map. This has not been useful for any research in the past so it has not been implemented but it would not be hard to do so.

3.2. **How to use cities in the path planner.** City XML files can also be used in the MAGICC lab path planner software. To add a city to the path planner, a terrain map must be loaded first. To load a terrain, click the browse button and choose the terrain map file (*.tmf). When the terrain maps are loaded, they are downsampled using the resolution in the terrain resolution drop down box so that there will be enough resolution to add a detailed city. To add the city after the terrain is loaded click the add city button and choose the *.cit file. The four corners of each building in the city file are converted to four terrain map coordinate corners using the terrain dimensions and scaling and the size and rotation of the building. Then the height of the building is added to the terrain inside the building corners using a polygon fill and a line drawing technique from Computer Graphics with OpenGL [1].

4. How to create/use Aviones targets

Aviones has the ability to add spherical targets of any color as well as a Utah Transit Authority Bus. These targets are useful for testing target localization and target tracking. Targets are defined in an XML file with a *.tgt extension. Targets are loaded into Aviones by choosing File→Load targets. A spherical target is specified using the following syntax

```
<target center-east="0" center-north="0" height="100" radius="5"
color="ff002a" />
```

All measurements are in meters and the color is specified in the same way as HTML. HTML represents RGB color with six hexidecimal characters. The first two are the red component middle two are the green component and the last two are the blue component.

Targets can be given various movement patterns. The following XML demonstrates how a sphere can be set to move in a circle.

```
<target center-east="0" center-north="0" height="100" radius="5"
color="ff002a">
    <move-mode type="circle" velocity="30" radius="80"
    center-east="-5" center-north="5" />
</target>
```

The other types of available movements are straight line, circle, square, rectangle and random. The syntax for specifying each type of movement mode can be found in Appendix A.

5. City Generation Algorithm and Implementation

The random city maker generates a city in a spoke and wheel pattern and fills in city segments with city block grids. Building height is determined using one of three height functions. This section describes the algorithm for generating these random cities. This program was created using C++ and the Trolltech Qt graphical user interface library. The program was compiled using Microsoft Visual Studio .net 2003, but could be compiled on any compiler that Qt supports including gcc and Borland. This documentation assumes that the reader knows C++ and Qt.

5.1. **Internal C++ Classes.** The Random City uses the C++ class structure to organize the code. The classes can be combined into two major classes, user interface classes and object classes. User interface define how the user interact with the program and are as follows.

- **CityDisplayGL:** The openGL widget that displays the three dimensional view of the city
- **OverheadGL:** The openGL widget that displays the overhead view of the city
- **CityProperties:** The window that contains the properties of the city
- **BuildingProperties:** The window that contains the properties of the default buildings
- **FlatBuildingProperties:** The window that contains the properties of the flat block textures
- **MusicControls:** The completely unnecessary music player user interface

The non-openGL widgets are easily modified using Qt Designer, a user interface for creating dialogs and widgets.

Object classes are classes that define the physical structures. Each class contains information about location, size and description of the object. Each class knows how to render itself in an openGL context and how to read itself from an XML file.

- **BCity:** A class containing city attributes
- **BNeighborhood:** A class that records the polygon outline of a neighborhood and manages the buildings and roads within that neighborhood
- **BBuilding:** A base class that has default behavior to draw a textured building in the given block

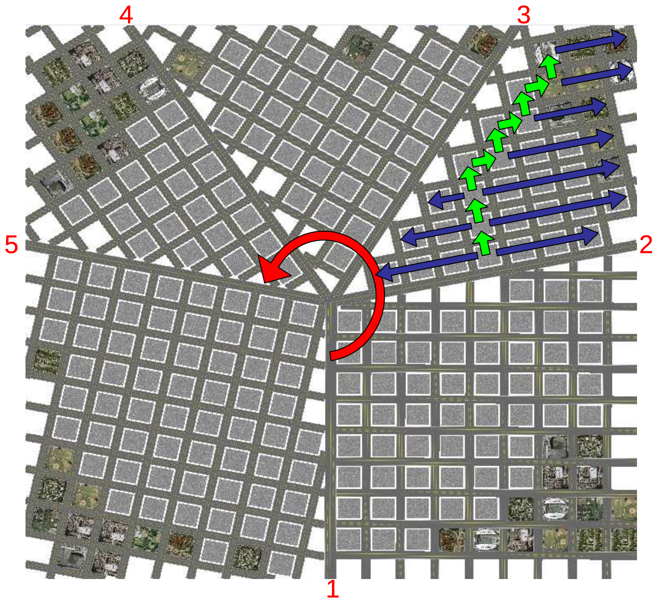

FIGURE 3. An overhead view of a generated city. Main roads are labeled in red and neighborhood building creation is shown in green and blue between main roads 2 and 3.

BBuildingFlat: A class that inherits from BBuilding and draws a flat texture on a block instead of a tall building

BBuildingTex: A class that contains a building texture and an associated acceptable height range for that texture

BRoad: A class that can load one of two different textures depending on if it is a major road or a minor road

Lastly, the program uses tinyXML which is a small XML library that was downloaded and used to read XML files.

5.2. **City Generation Algorithm.** When the Generate Random City button is pressed, the BCity::createRandom() function gets called which in turn calls BCity::createRandomBuildings(). The createRandomBuildings() function divides the city area rectangle radially by randomly generating main roads. Main roads are described by their angle measured in degrees on the set $[-180, 180]$. The angle of the first road is selected randomly between $[-180, -90]$. Subsequent angles are determined by adding a random value from the set $[30, 120]$ to the previous angle as seen in red in Figure 3. If the previous angle plus 120 is larger than 180 then the the random number is chosen from the set $[prevAngle + 30, 180]$ and the program stops generating roads.

Neighborhoods are defined as the area between main roads and inside the city boundaries. Neighborhoods are represented by a list of points comprising a polygon that outlines the neighborhood boundaries. The list of points is calculated by determining which city boundary the first main road intersects and traversing the city border counterclockwise recording city corners until another main road is encountered. Then the city center is added to the list and a new neighborhood is created and continues traversing the city border. When the first main road is encountered again, all neighborhoods have been created.

After all neighborhood boundry points have been calculated, the buildings and minor roads inside the neighborhood are created. The first two points on the neighborhood's point list are the center $(0, 0)$ and the intersection of a main road and the city border. The building grid for the neighborhood is aligned parallel with this first main road. The first building is placed between the center of the city and the edge of the city along this first main road. Then the streets that are perpendicular to this main road are incrementally placed while the streets are inside the neighborhood. The streets that are parallel to the main street are placed in the same fashion.

Next, the buildings are placed in the neighborhood. First, buildings are placed along the first main road to the right and to the left of the first building. Then a new row is created by starting from the location of the first building and stepping away from the main road. If this new location is inside the neighborhood, a new row of buildings is created. If this new location is outside the neighborhood then five blocks to the right or left are checked to see if they are in the neighborhood. The first block entirely inside the neighborhood is used as the new row center block. Building creation is shown in Figure 3 between main roads 2 and 3. If no such block can be found, then the neighborhood is complete. The rotation of each building is the angle of the main road and the height of the building is calculated as described in Section 5.3

5.3. **Height Function.** The user may specify one of three height functions for the buildings that are created. Three height functions are available because most actual cities have a gaussian distribution, but some cities with a few very tall buildings look more exponential, and others where two cities have grown together, have a bi-modal gaussian height function. Examples of the three height functions in the Random City Maker can be seen in Figure 4 In each case, the height of any location is computed, then the height is decreased by a random percentage of the original height chosen from the set $[0\%, 70\%]$. If the height function returns a value less than 10 meters, a flat building texture is used for that block.

In the following height functions, h is the height returned for the building, h_{max} is the maximum height for any building, (x, y) is the location of the building in the city, (c_x, c_y) is the size of the city in each dimension, (s_x, s_y) is the horizontal shift of the distribution and v is a user specified height decay value. The bivariate normal distribution,

$$\begin{aligned}
h &= h_{max} * \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} e^{-\frac{z}{2(1-\rho^2)}} \qquad (1) \\
\textit{where} \\
z &= \frac{(x-s_x)^2}{\sigma_x^2} - \frac{2\rho(x-s_x)(y-s_y)}{\sigma_x\sigma_y} + \frac{(y-s_y)^2}{\sigma_y^2} \\
\sigma_x &= \frac{c_x}{v} \\
\sigma_y &= \frac{c_y}{v}
\end{aligned}$$

(a) An example of a city with a gaussian height function

(b) An example of a city with an exponential height function

(c) An example of a city with a bimodal gaussian height function

FIGURE 4. Examples of three different height functions implemented in the Random City Maker

was used to calculate the gaussian height function. Figure 5 provides a graphical representation of Equation 1. For gaussian distribution calculations, ρ is set to 0.1. The exponential function is

$$h = h_{max} e^{-\sqrt{\frac{v}{c_x * c_y}(x^2+y^2)}}, \tag{2}$$

which creates a circular exponential cone as seen in Figure 6. Values for v that produce good height decay for each height function are included in Table 1.

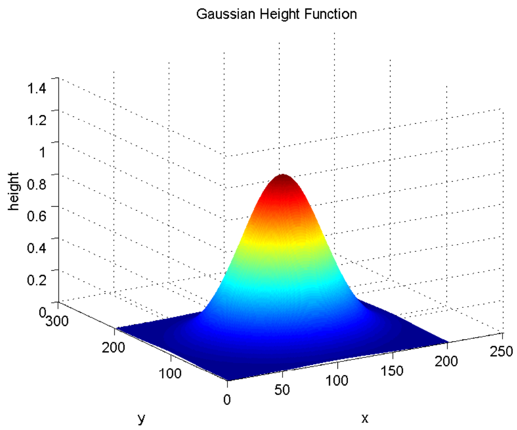

FIGURE 5. A graph of Equation 1, the gaussian height function.

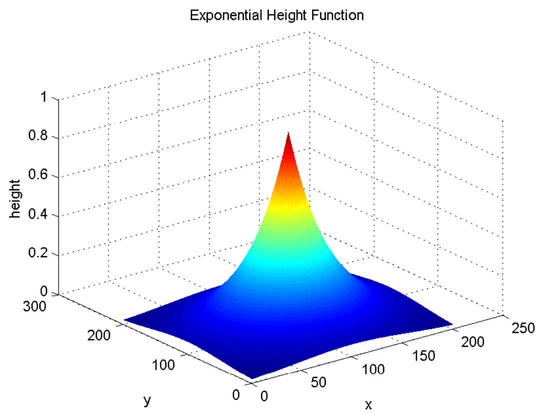

FIGURE 6. A graph of Equation 2

The bimodal gaussian function shifts two gaussian distributions using (s_x, s_y) and adds them together.

6. Conclusion

The Random City Maker performs a sufficient job of creating a city and saving it to an XML file so that it can be rendered in Aviones.

6.1. **Future Work.** There are many improvements that could be added to the Random City Maker program. The most useful feature to add is the ability to create single buildings and roads and position them one at a time. This would allow the user to create cities in whatever layout is desired. This would involve converting mouse clicks to world coordinates. Additionally, a dynamic framework for building texture management would be desirable. Currently the executable must be recompiled in order to add building textures. The main difficulty would be associating a height range with each building textures. It would also be desirable to only load the textures that will be used. Random City Maker and Aviones both load all the textures regardless of whether they will be used or not. This could be accomplished using a separate text file containing the name of the image and the associated height range.

If more realistic cities are desired, add custom buildings of different shapes and sizes including general quadrilateral buildings and triangular buildings for locations where the block is not square. Furthermore, pedestrians could be added to the Aviones target capabilities. Also, the road drawing function in Aviones could be improved so that the road is mapped directly onto the terrain and never underneath.

Lastly, it would be nice to have the city maker take command line arguments so that it can be used for monte carlo simulations

Appendix A. Example Target File

```
<!-- All speeds are in meters/second -->
<!-- All distances are in meters -->
<!-- All angles are in degrees -->
<targets>

    <!-- SPHERES -->
    <!-- A yellow sphere that doesn't move -->
    <target center-east="-5" center-north="5" height="100"
        radius="5" color="ffff00" />

    <!-- A red sphere that goes a constant velocity into
        the sunset -->
    <target center-east="0" center-north="0" height="20"
        radius="30" color="ff0000" >
        <move-mode type="straight" velocity-east="10"
            velocity-north="-10" />
    </target>

    <!-- A magenta sphere that moves in a circle around
        (5,5) -->
    <target center-east="5" center-north="5" height="50"
        radius="20" color="ff00ff" >
        <move-mode type="circle" velocity="30" radius="80"
            center-east="-5" center-north="5" />
    </target>

    <!-- A white sphere that moves in a square centered
        at (-5,5) -->
    <target center-east="500" center-north="100" height="50"
    radius="20" color="ffffff" >
        <move-mode type="square" velocity="25" center-east="-5"
            center-north="5" width="60" />
    </target>

    <!-- A gray sphere that moves in a rectangle defined
```

```
        below -->
    <target center-east="-300" center-north="100" height="50"
        radius="20" color="888888" >
        <move-mode type="square" velocity="25" east-max="80"
            east-min="-50" north-max="50" north-min="-50" />
    </target>

    <!-- A blue sphere that moves randomly -->
    <target center-east="10" center-north="-10" height="100"
        radius="8" color="0000ff">
        <move-mode type="random" velocity="15" max-turn-radius="30"
            heading-hold-time="1" />
    </target>

    <!-- BUSSES -->
    <!-- A UTA bus at (0,0) that doesn't move -->
    <bus east="0" north="0" heading="0" altitude="4" />

    <!-- A UTA bus that moves in a circle -->
    <bus east="40" north="40" heading="-50" altitude="4" >
        <move-mode type="circle" velocity="60" center-east="80"
            center-north="-100" radius="30" />
    </bus>

    <!-- A UTA bus that moves in a square centerd at (50 -80) -->
    <bus east="60" north="-100" heading="90" altitude="10" >
        <move-mode type="square" velocity="15" center-east="50"
            center-north="-80" width="30" />
    </bus>

    <!-- A UTA bus that moves randomly -->
    <bus east="10" north="-10" heading="10" altitude="10">
        <move-mode type="random" velocity="15" max-turn-radius="30"
            heading-hold-time="1" />
    </bus>

</targets>
```

References

1. Donald Hearn and M. Pauline Baker, *Computer graphics with opengl*, 3rd ed., Pearson Prentice Hall, 2004. 3.2

459 Clyde Building, Provo, UT 84602
E-mail address: brandoncall@gmail.com