Virtual Navigation of Multimedia Maps

A versatile map generator and viewer

Mountain Brook High School

Software Composition and Modeling Laboratory
SoftCom
Department of Computer and Information Sciences
University of Alabama at Birmingham

Robert Smyly and Oliver White

Mentor: Dr. Jeff Gray

Abstract

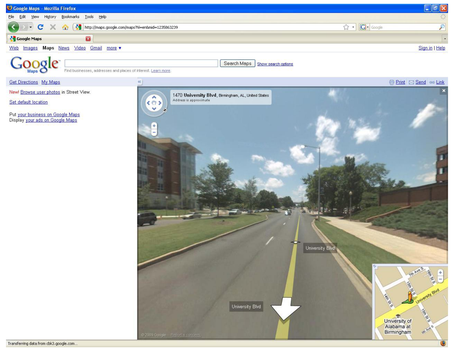
Many navigation utilities, such as Google Street View, provide users with an interactive three-dimensional environment that represents certain predetermined locations. This type of utility has an advantage over traditional two dimensional maps in that they allow users to see the locations as if they were actually there and recognize landmarks that will help them navigate with a greater degree of confidence. However, current systems only work within a specific context and only the program's creators are able to add content to the maps. This project addresses this problem by allowing end-users to create their own virtual environments in a very flexible and expandable way.

The software developed through this investigation provides a way to easily map an area of any type (e.g., the inside of a large building or a series of convoluted trails in the wilderness). The maps can then be exported into a viewer, which enables users to navigate through any area in a natural and intuitive way. Several algorithms were developed for this project, allowing users to accomplish such tasks as clicking on the part of the map they want to explore or loading prebuilt maps from configuration files. For experimental assessment, the program was used to map several locations in the McWane Science Center.

Google Street View (right) is a web-based application that allows users to navigate through streets at eye level, as if they were actually driving down the road. However, it has several limitations: primarily, it can only be used for viewing roads and only Google can add new content. This project extends the concept of Street View to allow customized navigation of user-defined locations.

1. Overview

- The project investigates the creation of a versatile mapping utility and associated algorithms that aid in virtual navigation.
- Similar tools, such as Google Street View, are not adaptable and are fixed to a specific type of navigation and domain.
- This project seeks to create a tool that can create maps for a wide variety of areas and allow end-users to create their own content to be used by the tool.
- The project is divided between two separate programs: the navigation map editor, and the navigation viewer.
- The map editor enables users to create their own maps and export them in a format that is readable by the viewer.
- The navigation viewer is responsible for displaying the data in a way that provides a navigation context for a specific location.
- The project was tested by mapping sections of the McWane Science Center and portions of the UAB CIS Department.

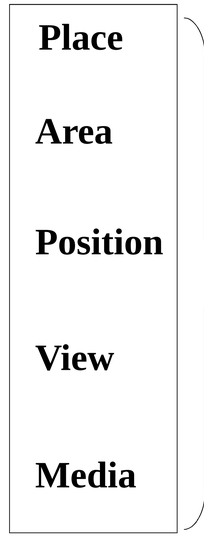


2. Design of the Editor and Viewer Programs

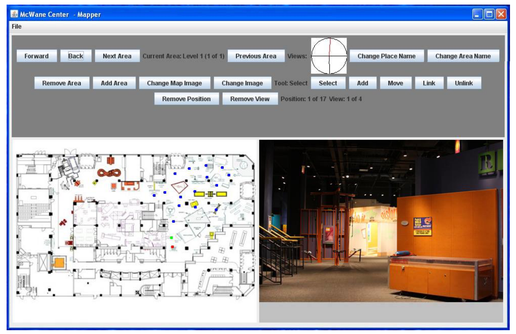
Both the viewer and the editor are built on top of the same set of classes. These classes contain all of the data structures used to store the information needed by the program, such as images and locations. The classes also simplify interaction with these data structures by providing an additional layer of abstraction. This reduces the amount of code needed in the project by using the same core features in each program.

This consolidation of code allows the viewer program to act as just an interface. The viewer program takes input from the user and sends it to the simplified methods in the class files. The class files then send the result back to the viewer, which displays it to the user.

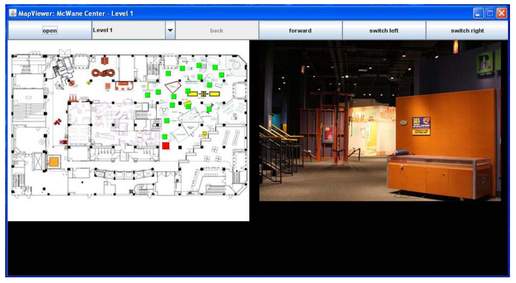
The designer program is used to create and edit the maps used by the viewer. The designer provides a graphical way to change the data stored within the core class files and then writes the data to a file that can be opened by the viewer through the core classes. Like the viewer program, the editor interacts with the core class files. When the user changes a property in the editor, it then sends the information to the class files, which make the changes to the data.



Core Class Files



Navigation Editor



Navigation Viewer

3. Algorithms and Techniques Used

In order to export the maps from the editor to the viewer, the core class files need methods to write to and read from a standard configuration file. The core class files handle this feature. To save the map, each object creates a string to represent itself. All of the strings are then combined and the result is written to a file. To load the map, a file created by the editor is opened by the core class files. The program then reads through the file and extracts the original strings. These strings are then used to construct the navigation map.

```
1,McWane Center
level1.jpg,17,Level 1
314,210,4
86,1,-1.0,-1.0,314.0,193.0
IMG_9806.jpg
0,1,-1.0,-1.0,-1.0,-1.0
IMG_9807.jpg
272,1,312.0,192.0,-1.0,-1.0
IMG_9808.jpg
181,1,-1.0,-1.0,-1.0,-1.0
IMG_9809.jpg
312,192,4
88,1,314.0,210.0,315.0,161.0
IMG_9810.jpg
1,1,-1.0,-1.0,-1.0,-1.0
IMG_9811.jpg
274,1,316.0,160.0,314.0,212.0
IMG_9812.jpg
…
```

This is an excerpt of the file that contains the map's data.

```
public void load(File fileName) throws IOException {
          Scanner input = new Scanner(fileName);
          String placeString = input.nextLine();
          StringTokenizer placeToken = new StringTokenizer(placeString, ",");
          int areaCount = Integer.parseInt(placeToken.nextToken());
          setName(placeToken.nextToken());
          areaList = new ArrayList(areaCount);
          for(int areaI=0; areaI<areaCount; areaI++) {
               Area tempArea = new Area();
               String areaString = input.nextLine();
               StringTokenizer areaToken = new StringTokenizer(areaString, ",");
               tempArea.setFileName(new File(fileName.getParent()
                + File.separator + areaToken.nextToken()));
               int positionCount = Integer.parseInt(areaToken.nextToken());
               tempArea.setPositionList(new ArrayList(positionCount));
               tempArea.setName(areaToken.nextToken());
               areaList.add(areaI, tempArea);
               …
```

This is a fragment of the code that is used to read the configuration file and import the data it contains.

In order for the user to be able to move forward or backward, the core class files need to know which points are in front of the current points and which ones are behind. This is done by giving each point a forward and back reference point. An algorithm we designed finds the closest existing position from the reference point, determines the corresponding view, and moves the user to the position.

```
public int findLinkedPosition(Point linkPoint) {
          if(linkPoint.getX() < 0.0)
               return -1;
          Point[] points = getPoints();
          double minDistance = points[0].distance(linkPoint);
          int minIndex = 0;
          for(int i=0; i<points.length; i++) {
               if(linkPoint.distance(points[i]) < minDistance) {
                    minDistance = linkPoint.distance(points[i]);
                    minIndex = i;
               }
          }
          return minIndex;
     }
```

This code is the method that finds the closest existing point to the location that the current point links to.

4. Future Work

There are several ways this project could be improved in the future:

- Support for true panoramic pictures

  This would allow for a much more immersive experience with the project. Users would be able to seamlessly change the angle that they are viewing.

- Support for sound and video

  This would also make the project more immersive. Users would be able to hear and see the environment in a way that better portrays the area.

- Implementations for mobile platforms such as Android or iPhone OS

  This feature would allow users to take advantage of the viewing program when they do not have quick access to a desktop computer.