Automatic Derivation of Electronic Maps from X3D/VRML Worlds

Lucio Ieronutti[(1)]
HCI Lab
Dept. of Math and Computer Science,
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Roberto Ranon[(2)]
HCI Lab
Dept. of Math and Computer Science,
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Luca Chittaro[(3)]
HCI Lab
Dept. of Math and Computer Science,
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Abstract

Maps of physical environments and geographical areas are pervasively exploited in many human activities. Electronic maps of virtual worlds have been studied and proven to be useful as navigation aids to help users in finding their way through the 3D world. The contribution of this paper is twofold. In the first part, we propose an automatic method for the derivation of electronic maps from Web3D worlds which is based on standard X3D/VRML nodes, and can thus be run by any X3D/VRML browser. In the second part, we discuss how we are using the proposed method for the study and support of users' navigation in virtual worlds. In particular, we show: (i) how the derived maps simplify the implementation of various navigation aids, and (ii) how we are using the derived maps to visualize usage data from users' visits to 3D Web sites.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques -- Interaction techniques. I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism -- Virtual reality. H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems -- Artificial, augmented, and virtual realities.

Keywords: Navigation Aids, Electronic Maps.

1. Introduction

Maps of physical environments and geographical areas are pervasively exploited in many human activities. Besides their familiar use for navigation, maps are also used for many other purposes, including planning and information presentation in fields as diverse as geology, military operations, and urban planning.

[(1)]e-mail: ieronutt@dimi.uniud.it
[(2)]e-mail: ranon@dimi.uniud.it
[(3)]e-mail: chittaro@dimi.uniud.it

A map is a (visual) representation showing spatial relationships among different types of information, enabling their display for (human) interpretation and analysis. *Electronic maps* of virtual worlds (i.e., the electronic analogues of the paper maps commonly used by people) have been studied and used (mainly) as navigation aids to help users in finding their way through virtual environments (see e.g. [Aretz, 1991][Darken and Peterson, 2001][Darken and Sibert, 1996][Ruddle et al. 1999]).

In this paper, we consider the problem of automatically deriving an electronic map for a X3D/VRML world, and propose a method that is: (i) effective for many Web3D worlds, and (ii) simple to implement, being based on standard X3D/VRML nodes (other proposed approaches to the problem, as discussed in Section 2.1, require instead the access to low-level graphic primitives). More specifically, the approach we propose is able, starting from a X3D/VRML world, to derive a two dimensional matrix such that: (i) each cell of the matrix corresponds to an area of the virtual world and, (ii) the value of the cell indicates whether the corresponding area contains a geometry or not. Moreover, it is also possible to derive *personalized* maps for specific avatars (or humanoid characters) representing as free cells those areas that the character can travel *by walking*.

To show the practical usefulness of the method, in the second part of the paper we describe how we are using it for the study and support of users' navigation. In particular, we will: (i) outline an algorithm to visually present the electronic map to the user during world navigation, and review some important choices for effective visualization of maps for navigation purposes; (ii) show how our approach can simplify, in the Web3D context, the implementation of many navigation aids proposed in the literature; (iii) describe how we are exploiting the derived maps in a software tool that visualizes various kind of information about the behavior of visitors of Web3D sites (e.g., what are the most traveled areas of the world). This can be useful, for example, in the usability evaluation of a Web3D site.

The paper is structured as follow: Section 2 describes in detail the proposed approach to derive electronic maps from X3D/VRML worlds, discussing also limitations and related work; Section 3 illustrates how we are exploiting the proposed approach; finally, Section 4 concludes the paper and illustrates future work.

2. Automatic derivation of electronic maps

In this Section, we first briefly review existing methods for automatically deriving electronic maps of virtual worlds, and then propose a novel approach to automatically derive an electronic map of a X3D/VRML world.

2.1 Related work

Existing approaches to the automatic derivation of electronic maps of virtual worlds are typically based on rasterization algorithms [Bandi, 1998][Lengyel et al. 1990] that project onto a 2D surface the objects (e.g., walls, buildings) of the virtual world. For example, in the technique proposed by [Kuffner, 1998], an orthographic virtual camera is positioned above the scene, pointing down along the negative normal direction of the "floor" of the world, with the clipping planes of the camera set to the vertical extents of the volume of the scene one wants to reproduce in the map. The specification of this volume is needed, for example, because objects above the head of the avatar and below its feet should not be considered an obstacle to navigation. All geometry inside the specified volume is then projected and rendered into the back buffer or an offscreen bitmap (either using software rendering or using the graphics hardware for increasing speed) in such a way that pixels corresponding to areas with objects are rendered in a different color than pixels corresponding to free areas.

However, this kind of techniques cannot be easily applied in the Web3D context, since Web3D plug-ins and authoring tools do not typically allow the required level of control on the rendering pipeline.

2.2 Our approach to electronic map derivation: main features and limitations

As anticipated in the introduction, the electronic map our approach derives is a two dimensional matrix such that: (i) each cell of the matrix corresponds to an area of the virtual world and, (ii) the value of the cell indicates whether the corresponding area contains a geometry or not. This is the same kind of representation derived by previously proposed approaches. Unlike those approaches, our method is implemented using only a combination of X3D/VRML nodes, and can thus be simply run by a standard X3D/VRML browser.

To give the flexibility of obtaining electronic maps at different sizes and levels of accuracy, we allow the possibility to specify the level of detail of the derived map (i.e., how many cells in the map will correspond to one square meter area of the world). As we will describe in the next section, it is also possible to derive a *personalized map* that highlights navigable areas for a specific avatar. Since the capability of traveling a certain area (e.g., passing through a door) depends on the size of the avatar, then, in general, different avatars can lead to different personalized maps of the same world. For example, consider the very simple world in Figure 1a. The personalized map for avatar *A*, shown in Figure 1c, correctly indicates the possibility for the avatar of passing through the door. On the other hand, the electronic map for avatar *B*, shown in Figure 1b, does not indicate any passage, since the size of *B* does not allow it to walk through that door. The capability of generating personalized maps allows one to present the user with a map that highlights as free *only* those areas that the user is able to travel by walking.

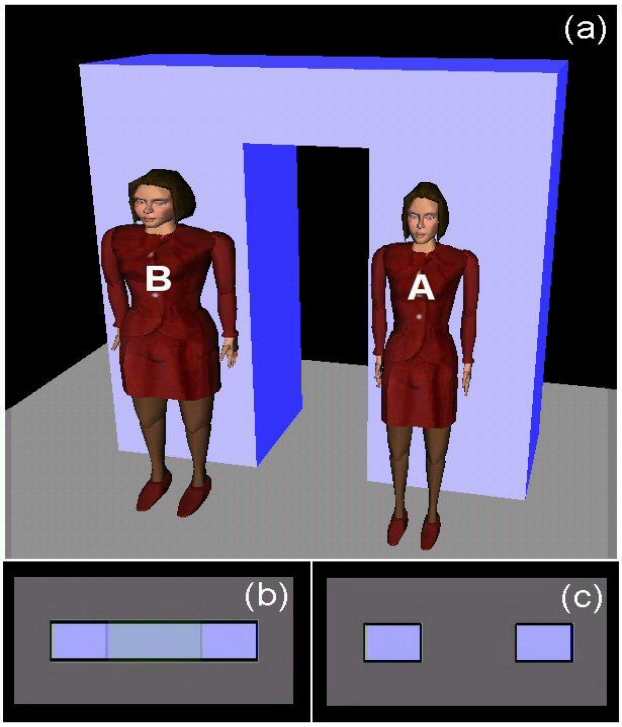


Figure 1. (a) A simple world and two avatars of different sizes; (b) personalized map for avatar *B*; (c) personalized map for avatar *A*.

A limitation of our approach is that accurate maps can only be derived for worlds where the floor is flat. In our approach, any kind of geometry that is above the floor and below the head of the avatar will be represented in the derived map as an obstacle to navigation: as a consequence, all geometries that could be overcome by the avatar will be (incorrectly) represented as obstacles to navigation. However, this limitation is common to the other approaches to the same problem mentioned in Section 2.1. It has also to be noted that this limitation does not hold for multi-floor worlds (e.g., a building with many floors), provided that each floor is flat: in these cases, it is possible to derive a separate map for each floor, and then properly combine the different maps together.

2.3 Deriving the electronic map

The basic idea of our approach is to determine whether a cell of the map should indicate the presences of a geometry that prevents navigation, by checking if the corresponding area in the world can be traveled by an avatar created for the purpose. This is done by automatically moving a `ViewPoint` through the world (i.e., having the `ViewPoint` point *scan* the world) and detecting each collision of the viewpoint itself with any geometry. Whenever a collision is detected, the cell of the map that corresponds to the current position of the viewpoint is marked as containing a geometry. We now describe in detail how the proposed approach is implemented.

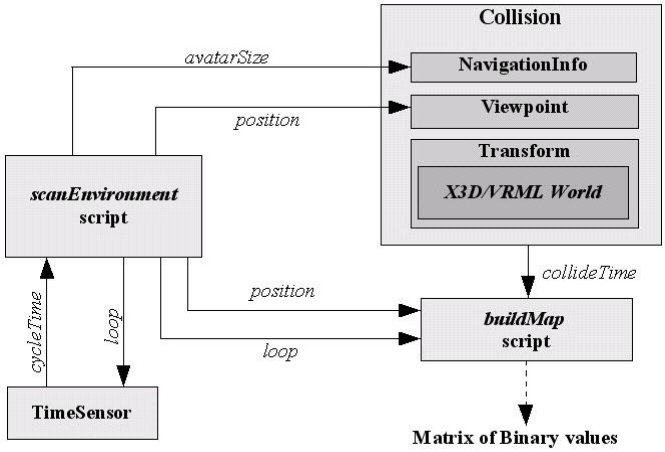


Figure 2. X3D/ VRML Nodes and routing of events in our approach.

The set of required X3D/VRML nodes, together with the needed events, is shown in Figure 2. A `TimeSensor` node starts and stops the map derivation process, and controls the speed of scan. The derivation of the map is carried out by two VRML scripts, called *scanEnvironment* and *buildMap*. The first script computes and updates the position of the viewpoint to scan the entire world (or the part of it whose map we want to derive). The second script receives detected collision events and updates the cell of the map that corresponds to the current position of the viewpoint. Collision events are generated by a `Collision` node, whose children include a `NavigationInfo` node (defining the size of the avatar used for scanning), a `ViewPoint` node (specifying the current position of the avatar), and a `Transform` node, whose children is the X3D/VRML world whose map we want to derive.

We now describe in more detail the *scanEnvironment* and *buildMap* scripts.

2.3.1 Scanning the world

To define the part of the world that has to be represented in the map, one needs first to define the *scan volume,* i.e., a box whose size and position will identify the part of the world that will be represented in the map. More particularly, the scan volume (see Figure 3 for example) is defined by:

- its size (specified by the *xLen, yLen* and *zLen* variables), in VRML length units;
- the position, specified by the *xPos, yPos* and *zPos* variables, of the top-left corner of its bottom side (with respect to the world coordinate system); *yPos* will typically correspond to the *y* coordinate of the world floor;

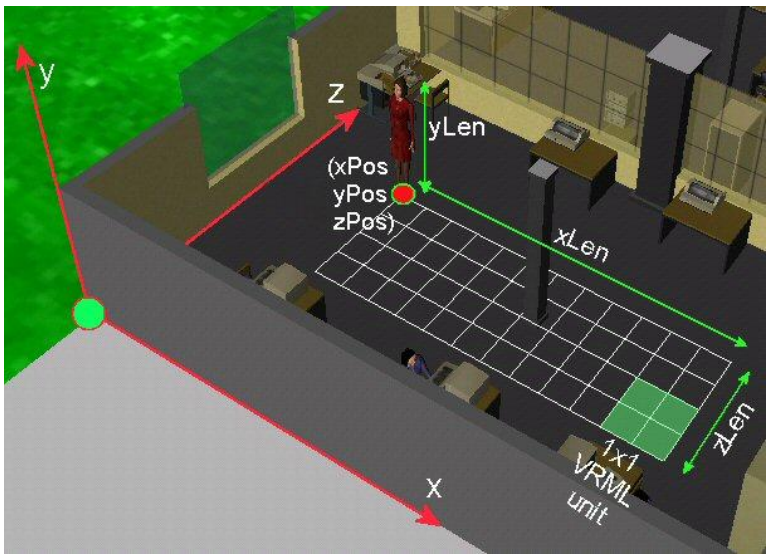


Figure 3. A graphical representation of a scan volume superimposed on a VRML world representing a building.

Second, one needs to define the *accuracy* of scanning, which we define as the length (in VRML units) of the side of each cell. The value of *accuracy* will directly influence the accuracy of the derived map. In general, a smaller value allows one to obtain a more detailed representation of the world, but the obtained map will be larger in size and thus will require more time for on-line computations (e.g., on-line path planning).

For example, in Figure 3, we have set *xLen*, *yLen, zLen*, *xPos, yPos* and *zPos* to obtain the electronic map of a subpart of the world, and then we have set *accuracy* to *0.25* so that an area of 1 square meter is represented by 4 map cells.

All the parameters that define the scan volume and the *accuracy* parameter are fields of the *ScanEnvironment* script (whose VRML code is shown in Figure 4). The script computes a list of coordinates, corresponding to the sequence of viewpoint positions needed to scan the specified scan volume with the desired accuracy, using the algorithm outlined in the following.

At initialization time, the script sets the fields of the `NavigationInfo` node that will be bound to the `ViewPoint` node during the scan. More specifically, the function `setAvatarSize` (called in row 16 of Figure 4):

- sets the height of the `NavigationInfo` node to the value of *yLen* (i.e., the height of the scan volume);
- sets the width of the `NavigationInfo` node (i.e., the radius of the bounding cylinder of the avatar) to half the *accuracy* value (i.e., half length of the side of a map cell). In this way, the bounding cylinder of the avatar we create for scanning purposes is exactly contained into each cell of the map (when positioned in the centre of it). This is done to ensure that, even in cases when an obstacle is only partially included into a cell, a collision is guaranteed to be detected.

Moreover, the `position` field of the script (which will specify the position of the `ViewPoint` during the scan) is initialized, in rows 18-20, to the centre of the top left cell of the map, while its *y* coordinate is set to *yPos* + *yLen* (i.e., the position of the head of the avatar used for the scan).

Then, at constant intervals of time (controlled by the `TimeSensor`), the `cycleTime` function of the script computes and updates (rows 25-32) the next value of the `position` field so that the `ViewPoint` is moved to the centre of subsequent cell that needs to be considered in the scan volume. Each time the function is called, the `Viewpoint` is moved to the cell at the right of the current cell (rows 25-27) or to the first cell of the next row, if the end of the current row has been reached (rows 30-32).

2.3.2 Building the map

The purpose of the *buildMap* script is to create a data structure that stores the map, i.e. a 2D array (called *map*) of binary values (1 for not navigable cells, and 0 for navigable ones). The code of the *buildMap* script is shown in Figure 5.

At initialization time, the script initializes the size of the map required for the scan (rows 13-16 in Figure 5).

The script receives, through its `position` input event, the current position of the `ViewPoint` (sent by the *scanEnvironment* script) at constant time intervals, and transforms it into a pair of indexes (rows 22-23 in Figure 5), that identify the cell of the map where the `ViewPoint` is currently located. Whenever a collision between the avatar and an object geometry is detected, the `Collision` node sends an event to the *buildMap* script; as a consequence, the script marks the current cell of the map as not navigable by setting its value to 1 (row 19 in Figure 5). When all cells of the map have been considered (i.e., a `loop` event is received by the *buildMap* script), the construction of the map ends.

The derived map can either be printed into the browser console (and then copied and saved for later use), or directly exploited inside the world, as we will discuss in Section 3.1.

2.3.3 Personalizing the derived map

The *buildMap* script is also able to personalize the derived map on the basis of the size of a specific user's avatar. When the `personalized` field of the script is `TRUE`, the map is post-processed as follows: all the cells that do not contain geometries but are within a circle of radius *r* from each not navigable cell (where *r* is the radius of the bounding cylinder of the user's avatar, whose size is set in the `userAvatarSize` field of the *buildMap* script) are marked as not navigable cells.

```
1 DEF scanEnvironment Script{
2   eventIn SFTime cycleTime
3   field SFInt32 xLen 0
4   field SFInt32 yLen 0
5   field SFInt32 zLen 0
6   field SFInt32 xPos 0
7   field SFInt32 yPos 0
8   field SFInt32 zPos 0
9   field SFInt32 accuracy 1
11  eventOut MFFloat avatarSize
12  eventOut SFBool loop
13  eventOut SFVec3f position
14  url  "vrmlscript:
15      function initialize(){
16        setAvatarSize();
17
18        position[0] = xPos+(accuracy)/2;
19        position[1] = yPos+yLen;
20        position[2] = zPos+(accuracy)/2;
21      }
22      function cycleTime(){
23        if (position[2]<(zPos+zLen))
24          if (position[0]<(xPos+xLen)){
25           position[0] = position[0]+(accuracy);
26           position[1] = yPos+yLen;
27           position[2] = position[2];
28           }
29           else{
30           position[0] = xPos+(accuracy)/2;
31           position[1] = yPos+yLen;
32           position[2] = position[2]+(accuracy);
33           }
34        else loop = false;
35        }"
36    }
```

Figure 4. The VRML *scanEnvironment* script.

This way, only areas of the world that allow the user's avatar to travel are shown in the map as navigable, while too narrow

passages are shown as not navigable areas. This can be useful for two purposes: first, it simplifies the activity of algorithms that reason with the map (e.g., path planning for a virtual character); second, it produces maps that are simpler to interpret by users. As an example, Figure 6 shows a VRML world, its map derived with *accuracy=0.25* (i.e. 1 VRML unit correspond to the length of 4 cells in the map), and a personalized map for an avatar with radius *r=0.25*.

2.4 Experimental results

We experimentally evaluated the approach on several VRML worlds, using the Cortona 4.1 VRML player, on a 2.4 Ghz Pentium 4 PC equipped with a 128 Mb GeForce4 Ti4600 graphics board.

```
1 DEF buildMap Script{
2   eventIn SFTime collideTime
3   eventIn SFVec3f position
4   eventIn SFBool loop
5   field SFInt32 xLen 0
6   field SFInt32 zLen 0
7   field SFInt32 xPos 0
8   field SFInt32 zPos 0
9   field SFInt32 accuracy 1
10  field SFBool personalized FALSE
11 field MFFloat userAvatarSize [0,0,0]
12 field MFInt32 map []
13  url  "vrmlscript:
14   function initialize(){
15     n = 1/accuracy
16     map[][]=new array[xLen*n][zLen*n];
17   }
18   function collideTime(){
19     map[i][j] = 1;
20   }
21   function position(p){
22      i=(p[0]-xPos)/(accuracy);
23      j=(p[2]-zPos)/(accuracy);
24   }
25   function loop(){
26     if (personalized)
27     /*process the map as described  in Section 2.3.3*/   }"
28  }
```

Figure 5. The VRML *buildMap* script.

The time required to derive the map varied from a few seconds for maps with tens of cells, to some minutes for maps with thousands of cells. For example, the map displayed in the centre in Figure 6 contained about 3000 cells and took five minutes to compute. The time required for the calculation of a map depends on the number of cells of the map and on the value given to the `cycleInterval` field of the `TimeSensor` that controls the map derivation process (roughly it corresponds to number of cells multiplied by `cycleInterval`). One would like to set the smallest possible `cycleInterval` to minimize the time required to derive the map; unfortunately, we noticed that with values smaller than 0.1, one obtains a map that is not accurate (i.e., some cells that do not contain object geometries are marked as not navigable anyway). This is probably due to the fact that, with very small values of `cycleInterval`, some collision events are lost.

3. Applications of the proposed method

In this section, we discuss how we are using the proposed approach both for the study and the support of users'navigation in Web3D worlds.

3.1 Implementation of navigation aids

One of the most relevant usability issues for a Web3D site is the navigational support provided by its user interface. In current Web3D sites, people often become disoriented and tend to get lost. Inadequate support to user navigation is also likely to result in users leaving the world before reaching their targets of interest, or to leave users with the feeling of not having adequately explored the visited world.

A possible solution is to provide the user with electronic navigation aids to augment her capabilities to explore and learn (see e.g., [Darken and Peterson, 2001][Li and Ting, 2000][Ruddle et al. 1999]). In the following, we show how the proposed map derivation method helps in implementing different kinds of navigation aids.

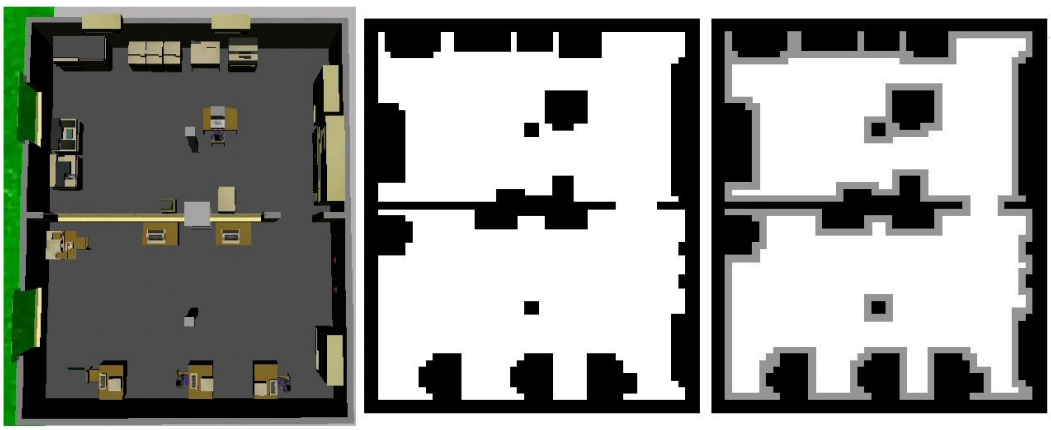


Figure 6. A VRML world representing a building (left), the map derived by our method (centre), and the personalized map for an avatar with *r=0.25* (right). Additional not navigable cells in the personalized map have been highlighted in grey.

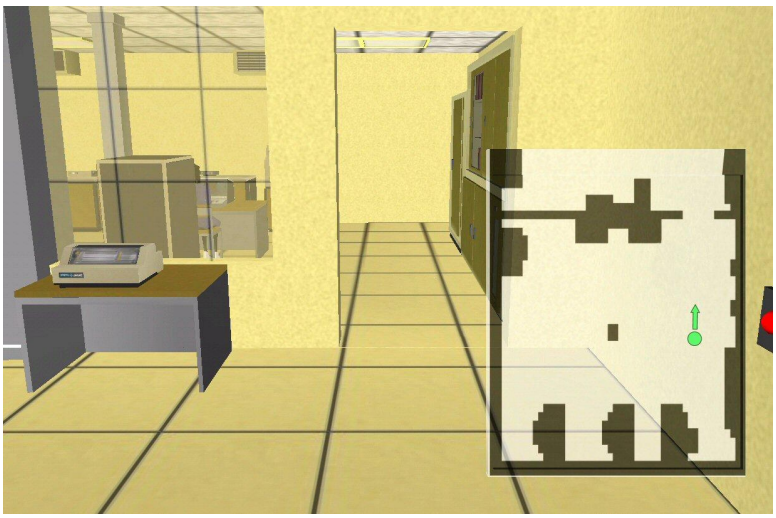


Figure 7. Displaying the map of the world to the user.

3.1.1 Displaying the map of the world to the user

A well-known navigation aid that we implemented by exploiting the output of the proposed method is the electronic map of the world displayed to the user, e.g. as in Figure 7.

The graphical representation of the map can be automatically built by considering each set of not navigable adjacent cells of the map, and building an `IndexedFaceSet` node for that set with:

- `Coordinate` node containing points the coordinates of cell corners in the perimeter of the set of cells;
- `coordIndex` field listing the cell corners in the order one obtains by sequentially following the perimeter of the set of cells.

The set of obtained `IndexedFaceSet` nodes is then placed on a (possibly semi-transparent) panel, scaled as desired and displayed to the user (see, e.g. Figure 7).

The advantage of electronic maps is that they help users in rapidly forming *survey navigational knowledge* about the virtual world, i.e., establishing relationships among locations, which allows them, for example, to evaluate alternate routes. On the other hand, a common problem with electronic maps is that users need to translate the *exocentric* view of the map into their *egocentric* view: this requires a considerable mental effort, and can lead to incorrect interpretations if the map is not properly displayed. For example, the orientation of the map can greatly influence the performance of users. The orientation of a *north-up* map is fixed, while in a *forward-up* map the orientation dynamically changes in such a way that the upward direction of the map always shows what is in front of the viewer (this can be easily achieved by using a `ProximitySensor` that tracks the user's orientation and rotates the map accordingly). Studies by Aretz and Wickens [1992] concluded that forward-up maps are better for navigation purposes, while north-up maps are more indicated for exocentric tasks (e.g., urban planning). However, users that have experience with videogames (where north-up maps are more common) seem to prefer (and obtain better results) with north-up maps [Darken and Peterson, 2001]. A third possibility is to use a north-up map with *visual momentum* (e.g., a triangle showing the actual direction of the user on the map), an approach that has shown to combine the advantages of north-up and forward-up maps [Aretz, 1991] (again, this can be obtained by using a `ProximitySensor` that tracks the user's orientation and rotates the triangle accordingly).

The size of the map becomes an issue with large worlds (e.g., virtual cities), since it may not be possible to show the entire map on the area of the screen devoted to this purpose. One possible solution is to maintain a global map by downscaling. This approach can effectively support search and path finding; however, downscaling the map can hide details that are crucial for navigation (e.g., narrow passages could not be perceived). A second solution is to adopt a local map, showing only part of the environment (typically, an area around the user's current position) with a good level of detail. The disadvantage of this solution is that the user might have to integrate the spatial knowledge acquired from different locations to carry out a navigation task. Ruddle, Payne and Jones [1999] have investigated the efficiency of local and global maps. According to their study, the combination of a global and a local map in a large-scale environment yields the most effective results in a minimum-path search; when only one map can be displayed, however, it seems that a local map is more suited to first-time users, while a global map is more effective when users develop some knowledge of the environment.

3.1.2 Indicating the path to be followed

Navigation aids that explicitly indicate the path the user should follow can be used whenever a given destination for the user is specified, either because the user herself has specified it (e.g., by selecting it from a list of destinations) or because it is set by the designer of the world or because it is proposed or predicted by the system (e.g., on the basis of mouse movements of the user, as in [Li and Ting, 2000]). The purpose of the navigation aid is then to help the user in following a path to the destination. This can be achieved in various ways:

- using *constrained navigation* techniques, where the user's control of avatar's movement is limited to avoid erroneous trajectories and collisions with object. For example, Igarashi

et al. [1998] have presented a simple interaction technique for walk-throughs in which the user draws the intended path directly on the scene, and the avatar automatically moves along the path.

- using *signs*, such as arrows, breadcrumbs, audio directional hints, etc. to suggest the walking direction along the path at each time.
- using *virtual guides*, i.e., animated characters that travel towards the destination and help the user by showing her the way. For example, we have recently proposed the exploitation of H-Anim guides in Web3D worlds [Chittaro et al. 2003].

Each of these approaches requires calculation of (or manual specification of) the path to be followed. A natural solution is to employ a path planning algorithm [Latombe, 1991], i.e., an algorithm that is able to derive an optimal (e.g., shortest) collision-free path from a starting position to a goal position. Among the various approaches to path planning that have been proposed in the fields of robotics and virtual environments, *grid-based* path-planning algorithms [Latombe, 1991] are able to compute the path starting from a so-called *occupancy grid*, i.e. a map of the environment such as the one derived by the approach we propose.

As a result, by using the electronic map in combination with a grid-based path planning algorithm, one can implement the above mentioned navigation aids and easily guarantee that the path the user needs to follow will be optimal and free from collisions. Moreover, since typically the computation of the optimal path must be carried out very quickly, the possibility of deriving simpler maps (e.g. with a suitable number of cells) allows one to constrain the time needed for path planning.

3.2 Building tools to study users' navigation in Web3D sites

Designing Web3D sites that are easy to navigate is not, in general, an easy task. This is mainly due to the lack of both proven design guidelines and proper tools that could help the designer in identifying and correcting possible navigation problems. On the other hand, in the context of traditional 2D Web sites, there are both extensive guidelines and commercial tools devoted to this purpose.

As a result, the main way to discover usability/design problems in Web3D sites is to observe how users interact with the site itself. In particular, one interesting aspect of study is to what extent users are able to navigate the virtual world and access and see locations and objects of interest. For example, certain places in the world may be unreachable or go unnoticed (especially with novice users) because of narrow passages, hardly detectable paths, and so on. Observing users can also be useful to simply determine which parts of the world are most visited (for example, this could be important in a 3D e-commerce site).

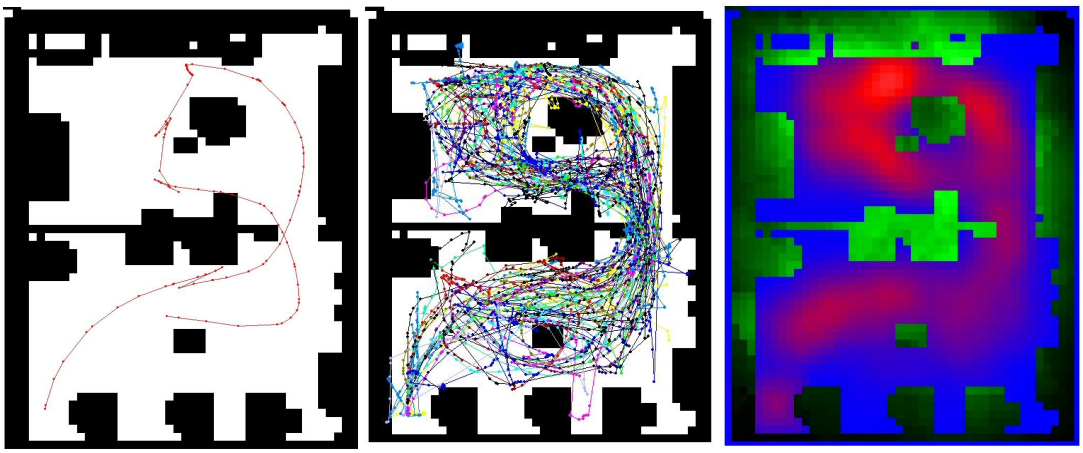
A current limitation in the usability evaluation of Web3D sites is the lack of software tools to support this activity, and thus the Web3D developer is forced to implement her own tools, or simply record usage data with paper and pencil.

The maps derived by the proposed method are an important building block to develop such topic. For example, we are using them as a basis to visualize recorded usage data of virtual worlds. Usage data is obtained by sampling information from various X3D/VRML sensors in the world, and recording it into a database (for example, recording of users' movements can be carried out using a `ProximitySensor` that constantly monitors users' position and orientation), as described in [Chittaro and Ranon, 2002].

In the following, we describe four of the different types of visualizations of usage data on electronic maps we developed, discussing also how they can help the Web3D developer in classifying and evaluating how users navigate in a Web3D site.

3.2.1 Visualizing usage data on the map

The first visualization we discuss simply highlights the trajectories followed by one or more users during a visit. This is accomplished by plotting on the map each sampled user's position, and connecting the obtained dots with lines, as in Figure 8a.



(a) (b) (c)

Figure 8. Using the map from the world in Figure 6 to visualize users' navigational behaviour: (a) visualizing a user's trajectory; (b) visualizing several users' trajectories; (c) visualizing areas where the user stayed for more/less time and more/less seen geometries.

By marking each user's trajectory with a different color, one can compare the paths followed by different users. However, plotting all detailed users' paths on the map can be visually confusing, as demonstrated by Figure 8b.

To better support an evaluator in studying the behaviour of a population of users in a Web3D site, one can develop more complex visualizations based on color-coded areas. For example, in Figure 8c (see color version), colors on the map indicate how much time users *stayed* on each navigable area: red represents the areas visited by users (the brighter the shade of red, the more time users stayed on the area; blue represents areas not traveled by users).

With similar conventions, one can also color map areas according to how many times users *crossed* them; in this case, the more users walk on an area, the brighter the shade of red used to paint it.

Coloring map areas according to how many times users crossed them is a time-independent visualization (i.e., it takes into account only information on paths followed by users), and thus is not influenced by the traveling speed of users. The other proposed visualization is instead time-dependent (i.e., take into account the temporal dimension of recorded data), or, in other words, depend on the speed at which users travel the world.

By combining different usage data, one is also able to visualize more/less seen geometries in the world**.** Since knowing both position and orientation of a user in time allows one to estimate how much time the different areas of the world have fallen inside her field of view, it is possible to visualize this information in the map using a suitable color coding. For example, in Figure 8c black represents unseen geometries; green represents the geometries seen by users (the brighter the shade of green, the more time the geometry has been seen).

3.2.2 Exploiting map-based visualizations: some examples

We now describe two scenarios that can benefit from the proposed visualizations.

The first scenario considers a situation where the Web3D developer is interested in detecting navigation problems in a Web3D world organized in more than one room. In this case, data about the time spent by users into each room can be confusing, because it can depend on the users' interest towards the room contents. The time-independent visualization, instead, allows one to easily identify the less visited parts of the world without being heavily affected by users' interests. Consider, for example, the visualization in Figure 9 (see color version), which highlights more/less traveled areas: one can easily notice that the top left room is the less visited one, although its entrance is close to the initial position of users. This is probably due to the fact that the entrance of that room cannot be seen from the initial position of the user. The visualization also highlights other aspects of users' behavior, e.g. the fact that the majority of users started the visit by passing through the closest available door.

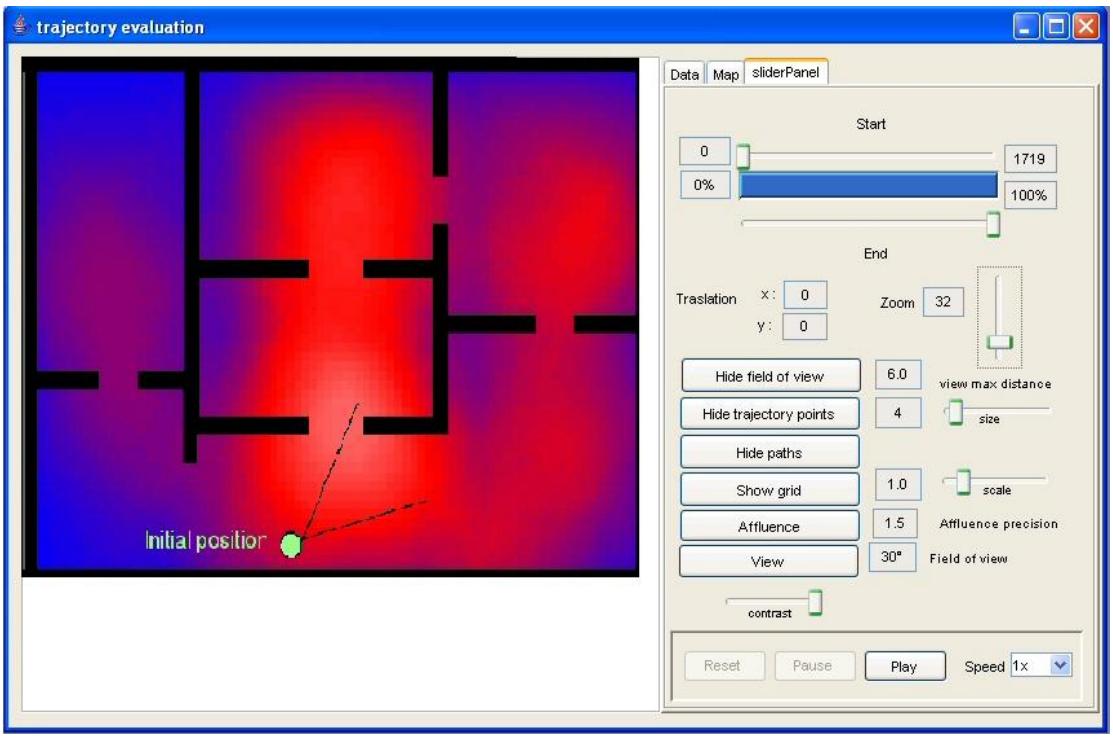


Figure 9. Identifying the less visited room using a time-independent visualization with color-coded areas.

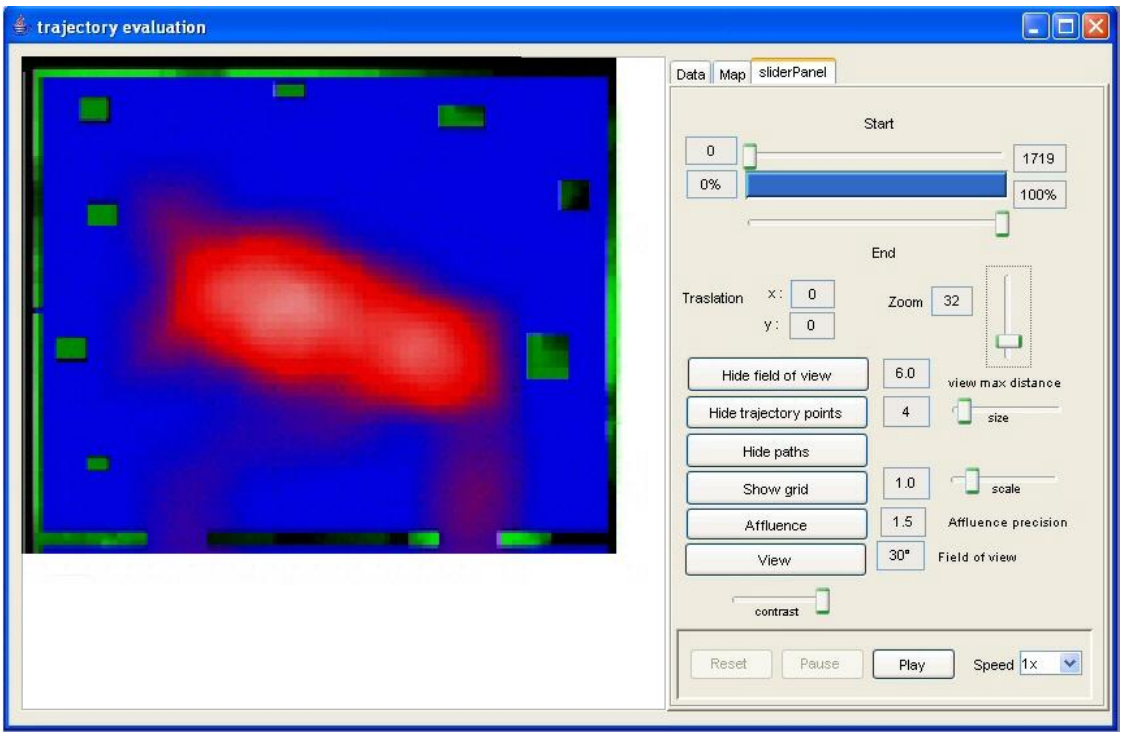


Figure 10. A representation of a "fish" visiting style using a time-dependent visualization with color-coded areas.

The second scenario considers a situation where the Web3D developer is interested in identifying visiting styles and users' interests in a Web3D site. It is interesting to note that, in the real world (e.g., in museums), such kind of evaluation have led to classifications of typical visiting styles, such as [Veron and Levasseur, 1983]. This specific classification allows one to identify four main categories of visitors: ants, fishes, butterflies and grasshoppers, with each category exhibiting a typical navigational behavior (for example, fish visitors tend to stay in the centre of a room, and equally devote their attention to all the objects in the room). By using a time-dependent visualization with color-coded areas, a similar classification can be exploited in Web3D sites. For example, Figure 10 (see color version) allows one to usually detect a typical fish visiting style by visualizing areas where the user stayed for more/less time and more/less seen geometries.

4. Conclusions

This paper proposed a method to automatically derive the map of a X3D/VRML world. The method simplifies the development of navigation aids and of tools that visualize usage data from past users' visits to Web3D sites. The proposed method could be also employed to support on-demand generation of site maps of Web3D worlds by content providers or even by browser vendors, for example as a navigation aid for users.

With respect to future goals of this project, we intend to extend its applicability to X3D/VRML worlds that include obstacles that can be overcome by the avatar, such as slopes and stairs. Performing multiple scans of the world at different heights (and then combine all the derived maps into a single map) may be not a satisfactory solution in the general case, since it could require a high number of scans. In particular, it would be interesting to encode elevation information in the map (as done in some real-world maps), allowing one to always determine if the avatar can move from its current cell to a given neighboring cell. This way, one can determine, for example, that the avatar can easily climb a stair, while it cannot climb a steep mountain in the virtual world.

5. Acknowledgements

This work is partially supported by the MIUR COFIN 2003 program (project "User Interfaces for the Visualization of Geographical Data on Mobile Devices") and by the Friuli Venezia Giulia region (Regional Law 3/98, project "3D Web Sites for the Promotion of Tourism and Cultural Heritage").

6. References

ARETZ, A.J., AND WICKENS, C.D. 1992. The Mental Rotation of Map Displays. Human Performance, 5(4), 303-328.

ARETZ, A.J. 1991. The design of electronic map displays. Human Factors, 33, 85-101.

BANDI, S. 1998. Discrete object space methods for computer animation. PhD thesis, Swiss Federal Institute of Technology, Lausanne, Switzerland.

CHITTARO L., RANON R., AND IERONUTTI L. 2003. Guiding Visitors of Web3D Worlds through Automatically Generated Tours. In Proceedings of Web3D 2003: 8th International Conference on 3D Web Technology, ACM Press, New York, 27-38.

CHITTARO L., AND RANON R. 2002. Dynamic Generation of Personalized VRML Content: a General Approach and its Application to 3D E-Commerce. In Proceedings of Web3D 2002: 7th International Conference on 3D Web Technology, ACM Press, New York, 145-154.

DARKEN, R.P., AND PETERSON, B. 2001. Spatial Orientation, Wayfinding, and Representation. In Stanney, K., ed., Handbook of Virtual Environment Technology, Laurence Erlbaum Associates, New Jersey.

DARKEN, R.P., AND SIBERT, J.L. 1996. Wayfinding Strategies and Behaviors in Large Virtual Worlds. In Proceedings of CHI '96, ACM Press, New York, 142-149.

KUFFNER, J.J. 1998. Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control. In Proceedings of CAPTECH '98: Workshop on Modelling and Motion Capture Techniques for Virtual Environments, Lecture Notes in Artificial Intelligence 1537, Springer-Verlag, Berlin, 171-187.

IGARASHI, T., KADOBAYASHI, R., MASE, K., AND TANAKA, H. 1998. Path drawing for 3d walkthrough. In Proceedings of UIST, 173-174.

LATOMBE, J.C. 1991. Robot Motion Planning. Kluwer Academic Publisher, Boston, MA.

LENGYEL, J., REICHERT, M., DONALD, B.R., AND GREENBERG, D.P. 1990. Real-time robot motion planning using rasterizing computer graphics hardware. In Proceedings of SIGGRAPH '90, ACM Press, New York, 327-336.

LI, T., AND TING, H. 2000. An Intelligent User Interface with Motion Planning for 3D Navigation. Proceedings of the IEEE Virtual Reality Annual International Symposium. New Brunswick, NJ, USA, 177-184.

RUDDLE, R.A., PAYNE, S. J., AND JONES, D.M. 1999. The effects of maps on navigation and search strategies in very-large-scale virtual environments. Journal of Experimental Psychology: Applied, 5(1), 54-75.

VERON, E., AND LEVASSEUR, M. 1983. Ethnographie de l'Exposition. Paris, Bibliothque publique d'Information, Centre Georges Pompidou.