Entertainment Computing 2 (2011) 123–131

Contents lists available at ScienceDirect

Entertainment Computing

journal homepage: ees.elsevier.com/entcom

Enabling dynamic generation of levels for RTS serious games

Matteo Bardini, Francesco Bellotti *, Riccardo Berta, Alessandro De Gloria

Dept. of Electronics and Biophysical Engineering, DIBE, University of Genoa, Via Opera Pia 11a, Genoa 16145, Italy

ARTICLE INFO

Article history:
Received 6 September 2010
Revised 23 November 2010
Accepted 6 December 2010
Available online 19 December 2010

Keywords:
Serious games
Real time strategy games
A* shortest path computation
Parallel processing
Image processing

ABSTRACT

This paper proposes a new family of Real Time Strategy (RTS) serious games that exploit a dynamic insertion of aerial/satellite images in games, in order to allow covering any target geographic area on user demand and automatically implementing several different gaming sessions. The approach intends to combine the possibility for the user of practicing decision strategies with the development of knowledge about specific geographical areas, which is important in particular to enhance the training of field operation personnel.

This target requires that the system correctly interprets the terrain features – in particular roads – in order to build a semantic correspondence between the game's logic and the background image.

RealPath, a new solution for automatic definition of paths on aerial and satellite images, which reduces the cost for the creation of new game maps. The algorithm exploits the image's pixel values and extracts information on the terrain in order to identify possible paths in the area. A parallel version of the algorithm has been implemented, exploiting multi-resolution pyramidal image processing, in order to allow better exploiting the power of current multicore processing architectures (CPUs and GPUs). The algorithm has been successfully tested on several real-world images, in real-time.

As a proof of concept, we present a RealPath-based RTS serious game that we are developing for training military and civil protection personnel in field operations. Exploitation of the algorithm in 3D virtual environments is also shown.

1. Introduction

Real Time Strategy (RTS) games are a popular category of games where players are challenged to take decisions and move objects on a territory, without turns, but in a continuous time-line, non-stop. They are usually played in a top down or 3/4 view, so that players can easily navigate though maps and scenarios (Fig. 1). Maps can be vectorial, ad-hoc made, or raster images. In this case, streets and tracks may be present, but are not considered by the logic of the game (e.g. Age of Empire,[1] Total War[2]). In the era of Geographic Information Systems (GISs) and Location-Based Services (LBS), with a high availability of aerial and satellite images, this is at least odd. Ubisoft's R.U.S.E.,[3] which has been released last September, is the first commercial game, to the best of our knowledge, that will exploit game paths corresponding to roads in real-world images used as maps [1].

While R.U.S.E. is likely to pre-compute road graphs from such images, our idea is to further develop the approach by using real-time image processing modules in order to introduce the possibility of a dynamic insertion of aerial images and exploit them as game maps by extracting semantic information on-the-fly. This would increase the playability of RTS by extending limitless the availability of game maps and by allowing end-users' customization. Moreover, exploitation of real-world maps would be useful for enhancing the quality of military and civil protection training, by combining real time strategy decisions/features and real-world geography knowledge.

Achieving this target requires facing a number of issues, that we are addressing in the development of an RTS Serious Game (SG) aimed at training military workforces and civil protection personnel on specific target geographic areas.

Developing this SG involves two major types of challenges: the logic of the application (i.e. the artificial opponent's intelligence, the definition of the levels, resources etc.) and the processing of map images so that they can be used as dynamic inputs. In this regard, it is necessary to semantically interpret and annotate the images so that they can be used as a meaningful background with a strong correspondence with the logic of the game. In particular, we identify three major needs:

- Identification of points where game resources (e.g. water tanks, weapons, power stations) could be likely placed in the map.
- Characterization of the territory (e.g. houses, fields, woods, etc.).
- Definition of best paths to reach destinations.

* Corresponding author. Tel.: +39 010 3532795.
E-mail address: franz@elios.unige.it (F. Bellotti).

[1] http://www.ageofempires3.com/.
[2] http://www.totalwar.com/.
[3] http://www.rusegame.com/.

1875-9521/$ - see front matter © 2011 Published by Elsevier B.V. on behalf of International Federation for Information Processing.
doi:10.1016/j.entcom.2010.12.005

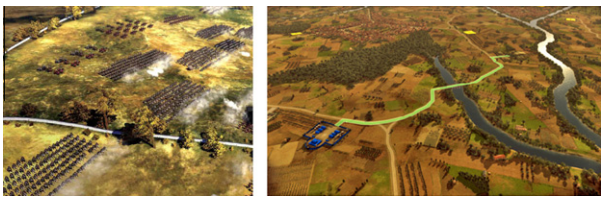

Fig. 1. Snapshots from Total War and a preview of R.U.S.E.

Our study has focused on this last point. The idea is that a player may decide to move objects (e.g. soldiers, tanks, fire-brigade cars, ambulances) from one place to another on the map and the system has to compute the corresponding best path – following the roads and the terrain – and the relevant time cost. The system has thus to compute the travel costs along segments. The cost is computed by considering the actual nature of the segments (i.e. roads, fields, wood), which is obtained through standard image segmentation algorithm [2]. The objective is to find the fastest path to destination.

This paper describes a new automatic procedure – we have called it RealPath – for the identification of best routes solely relying on the real-time processing of grayscale input images. A parallel implementation is also proposed, in order to meet the current computing architectures' paradigm.

This solution is suited to the design of levels of RTS games, a widespread category of games, but may be used for any generic game or geo-referenced application that need defining paths in outdoor maps.

The currently most known and advanced RTS SGs are BreakWay's Mosbe[4] and CityOne,[5] recently released by IBM. This is a sort of city manager, without the possibility, at least until now, of managing resources in real-time. Mosbe, on the other hand, is a platform that allows third parties to develop RTS SGs. It is used for teaching and training in military and civil fields.

The remainder of the paper is organized as it follows: Section 2 presents backrground information and related works. Section 3 describes the algorithm, while Section 4 shows and discusses the test results. The RealPath serious game is presented in Section 5. The final Section 6 draws the conclusions on the work done.

2. Background and related work

The path finder is a necessary Artificial Intelligence (AI) module for almost all types of videogame engines [3], since Non Player Characters (NPCs) management requires taking a short route to destination. In general terms, the path finder module can be applied in a variety of situations (e.g. setting an enemy's path in a First Person Shooter [4] or selecting paths a player team should follow to gather resources in a Real Time Strategy (RTS) game [5]).

Path finding is commonly modeled as a search problem in a graph data structure. The graph consists of a set of nodes, the waypoints, and connecting edges. Waypoints contain the information on the positions available in the game area, while edges model the cost to move from two adjacent waypoints. The path finding module computes if and how it is possible to reach one node from another one. The most used algorithms for path finding are A$*$ [6–8], which is used in different flavors in different game typologies, or IDA$*$ [9]. A major issue concerns the definition of the data structure (i.e. the waypoints and their connections for a specific game map). This is not a simple task for games with complex environments involving roads, buildings, bridges, forests, etc.

Research works on path finding are mostly focused on the problem of processing the graph data in order to find the shortest path, while few papers focus on the problem of extracting automatically the graph structure from the underlying map image.

There are two major types of solutions: grid layouts and waypoints. The former technique consists in covering a game area with polygonal tiles (different type and dimension of tiles impact the accuracy of the path). Graph nodes are extracted from these elements through different procedures, such as the polygonal/tiles movement [10], edge movement or vertex movement. Hybrid solutions increase the accuracy of the path, but increase the nodes' total number.

A different solution involves using the concept of *waypoint*. The main difference is that waypoints represent meaningful points in an environment. Thus, waypoints are to be placed manually (typically using 3D or 2D GUIs). For instance, in the computer games field they are designated by the environment designer [11]. Even the connections among waypoints are often set manually by the game designer [12]. The waypoint approach can drastically reduce the number of nodes (thus reducing the time for the shortest path computation), and increase the efficiency of the control of the movement. However, waypoints have to be set manually by the game designer after studying and testing the most interesting points that allow realizing a more natural movement. This is the apparent drawback of this solution: the work for creating a new game map is significant and it is difficult to expand the system with custom-generated maps.

A last approach is the navigation meshes [13,14], which is a set of polygons that are automatically placed to approximate the surface of the game terrain. This technique is used to map out areas of world that are not blocked or covered by obstacles – areas that can be used by the NPC [15]. This is done by abstracting the polygons from the virtual environment to generate a collection of convex polygons [11]. The centers of these polygons are then used to represent the nodes on a graph [16]. When two polygon's edges are shared, an edge between the respective nodes is created. This solution does not require the manual involvement of the designer, but suffers from the difficulty of deciding a correct approximation of the surface (how many polygons should be placed?). Moreover, the generated mesh needs to be fine-tuned by hand to be effective. Still, there may be places in which the NPC may behave unrealistically (e.g. getting stuck in a wall, passing through a river, etc.).

Recent works have been proposed for improving these two methods. For instance, [17] shows a hierarchical refinement model for the methods based on fixed grid, subsequently improved by introducing a progressive data elaboration [18]. Considering interaction between image processing and pathfinding, [19] shows an algorithm, applied in the automotive field, that exploits a Sober Filter to extract the edges of a map's elements and traces a path going around such edges.

3. The RealPath algorithm

3.1. Input images

The input to the RealPath algorithm is a raster picture representing a portion of urban or rural terrain with roads. Pictures are satellite or (more frequently) aerial photos. Input images may also be symbolic pictures as well. This case is simpler from a computational point of view because symbols already represent abstract entities (e.g. roads, bridges, buildings) and there is no noise. So, the following of our description focuses on images taken in the real world. In particular, in the tests we have used input images taken from Google Maps,[6] with a 1280 $*$ 1024 resolution and size between 2 and 8 MB (e.g. Fig. 2).

[4] http://www.mosbe.com/applications/index.shtml.
[5] http://www.01.ibm.com/software/solutions/soa/innov8/cityone/index.html.
[6] http://maps.google.com/.

Fig. 2. The original map image.

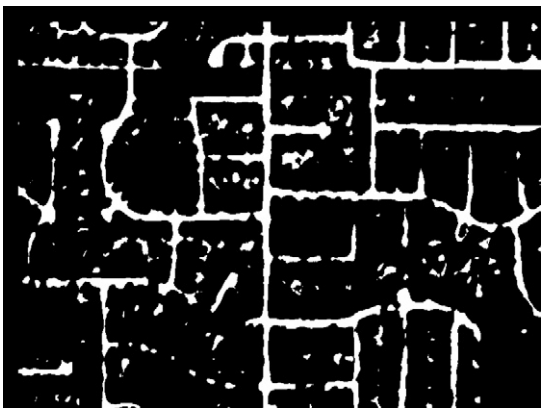

Fig. 4. Road tracing using the second, template matching-based, algorithm.

3.2. Raster pre-processing

Input images are pre-processed when loaded in the game, in order to speed-up the actual execution when computing dynamically the paths.

The first step consists in normalizing the image RGB pixel values on a 256-value scale. Then, all the image's pixels values are squared, in order to emphasize high values. This enhances the gradient values and helps to detect impracticable obstacles that for any reason are included in the admissibility region (for instance an aqua green lake).

The second step consists in a procedure that traces possible road pixels in the source image, with the objective of reducing the number of possible search nodes and thus the computation time. We call it road-tracing. The procedure asks the user to indicate a road point on the map. If this is not possible, a statistic average value is extracted from the history. The algorithm simply assigns the white color to all the pixels within a range centered around the selected road pixel value. We have empirically verified the optimality of a value range of ±30. Fig. 3 shows the results of a sample preprocessing. From empirical tests (Section 4) we have also seen that the road-tracing step is useful on color images only, while gray-scale images, reducing the number of information channels from 3 to 1, do not have enough information and also tend to produce spurious segments.

The fixed value range is a problem for certain types of pictures, in particular, when the road pattern is various (e.g. Fig. 6) or similar to the pattern of buildings close to the road. To overcome this limitation we have investigated a different implementation for the road tracing phase, exploiting template matching, which is typically more robust [20]. In particular, we have defined a simple set of kernels (Fig. 5), to be used for image convolution. The kernels represent rectangular shapes, that are used to highlight the structure of roads as rectangular segments with different orientations. The size of the kernel may be critical, since road sizes are different within a map and across maps. We have seen that small size kernels ($12 * 12$ pixels) are most suited, since they are able to highlight small roads and parts of larger roads that are clustered together. Results (Fig. 4) look better than the pixel-based approach, also for input gray-scale images (quantitative results are shown in Section 4). The trade-off is represented by the execution times. While the pixel-based approach requires less than 3 s, the template matching takes up to 30 s, which is anyway acceptable for a game level loading.

Concerning Figs. 3 and 4, it is important to stress that the tracing just highlights with a white color the detected road segments. The rest of the picture has been blackened for the sake of clarity. But the actual (non black) pixel color remains. This means that

Fig. 5. The kernels used in the template matching-based, algorithm. Given the symmetries, four kernels are sufficient for eight different orientations.

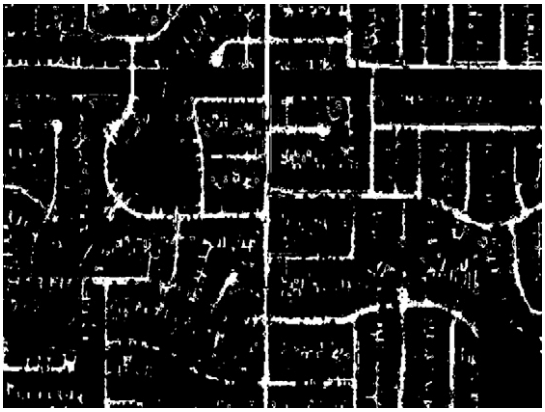

Fig. 3. Road tracing using the first, pixel-based, algorithm.

Fig. 6. Urban area with various road patterns.

black pixles do not mean insurmountable obstacles, but will be processed with the gradient processing described in the next sub-section C.

3.3. Runtime execution

In the game, the pre-processed image is used as the basis for dynamically computing paths. To this end, the PathFinder implements an A* search from the starting point to destination. Every pixel in the image is considered as a possible node. The traveling cost among nodes is proportional to the gradient difference. The cost is weighted with the cost of the terrain nature, which is extracted through terrain recognition procedures, in order to penalize the different terrains. For instance, a forest may be given a higher cost than a grass field.

The typical A* algorithm scrolls all the nodes in the Open List every time it is necessary to know whether a node was already created. This is critical for a real-time response with a huge number of nodes to be considered as in our case. Thus, we have implemented the Open List as a Hash Table, that guarantees a constant computational cost for each element search, independent of the search area. The Hash function relies on the position of the pixel in the image.

It is worth noting that our path definition does not involve optimizations for realizing a natural movement for the NPC, that is important in particular for 3D games. The RealPath algorithm searches the shortest path, which could not be natural for a human user. To address this issue the output can be post-processed by a standard algorithm, to smooth the path and move it away from obstacles. A lot of solutions have already implemented for this in current videogames. Since the number of points onto which computations (e.g. approximations or interpolations) can be performed is much higher than those of waypoint/grid algorithms [21,22].

3.4. Parallel implementation

Given the ever increasing number of parallel processing units in current computer architectures, it is important to implement parallel versions of algorithms in order to speed-up the computation.

Pyramidal algorithms are an established family of parallel algorithms for image processing. The concept of pyramid was introduced in 1984 by Adelson et al. [23], when facing the problem of robustly extracting features from an image in the shortest possible time, minimizing the computation costs.

Pyramidal algorithms rely on the fact that some features can be extracted from images with level of detail and resolution lower than the original. In particular, they noticed that they could obtain excellent results by using images that had been re-sized and filtered with Gaussian filters. A pyramid thus consists of a sequence of scaled and filtered images.

The real image constitutes the basis (or ground level) of the pyramid. The higher levels are obtained through a filtering and scaling of the previous ones.

In this way, different analysis may be performed of the same image, at different resolution levels, and obtain the final result through a refinement procedure, which is the basis concept of a generic pyramidal algorithm.

In our case, A* is sequential, by nature. We did not think of making it parallel. But to design the RealPath algorithm so that it exploits parallel executions of A* processing distinct sets of data obtainable through lower-resolution analysis.

Our idea is to compute a very coarse first path, define an initial set of points and recursively refine the subpaths until converging to the final path in the original, high-resolution map.

A data structure is thus needed that allows performing a first, coarse-grain analysis on a small-size image, so to be able to define a path comparable to a heuristic function slightly more evolved than the sole geometric distance, and subsequent path refinements obtained in ever more detailed maps. The pyramid shape offers these properties. After some experiments, we have concluded that, for a medium-size image (1–4 MB), 3- or 4-level Gaussian pyramids are sufficient.

The first step of the parallel RealPath implementation (Fig. 7) consists in computing the path with the requested start and arrival points on the top level of the pyramid. From this path, a series of n points is chosen. Our tests – we will discuss them in the next section (Fig. 11) – have shown that dividing a path into 30–40 segments usually gives optimal results. But this strongly depends, of course, on the parallelism level of the underlying computing architecture.

Every subsequent couple of such n points constitutes the start and arrival points of $n-1$ sub-paths. Every sub-path is independent of each other, which makes the following of the algorithm parallelizable, since the working data set is separable.

By repeating this procedure for all the levels until level 0 (the basis of the pyramid), the $n-1$ sub-paths are recursively subdivided into sub-paths. The last step computes the sub-paths on the real image. By joining such sub-paths, a sub-optimal path from the origin to the destination is obtained.

With this procedure, every sub-path can be computed on ever smaller image portions, which allows reducing the amount of data to be recorded. Moreover, after the initialization, which is sequential, the algorithm can be executed in parallel, since every sub-paths can be computed independent of the other.

Fig. 8 shows the case of a 3-level pyramid. Performance results are discussed in the next section.

4. Results

4.1. Precision

We have tested the algorithm by taking five satellite Google Maps images and computing 15 different paths for each image, for a total of 75 paths. The first four images are from US cities, while the last one is from Kabul (samples in Figs. 9–13).

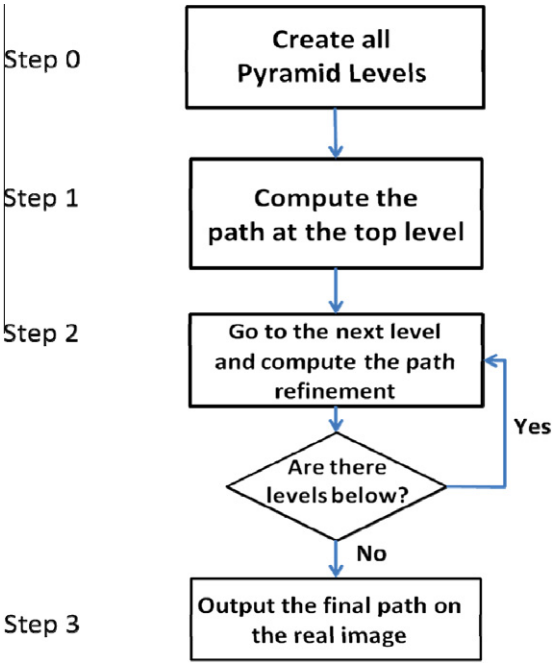

Fig. 7. Workflow of the RealPath's pyramidal implementation.

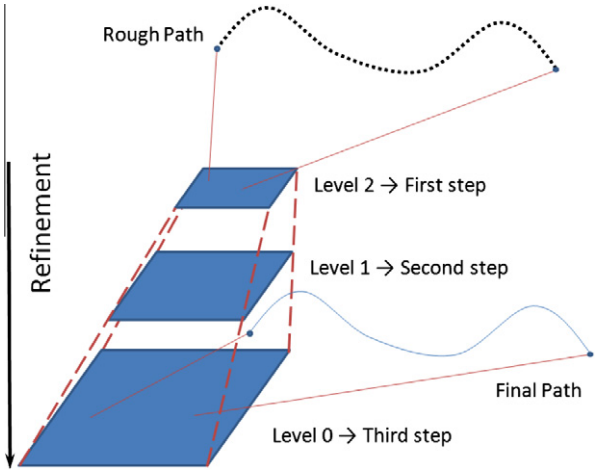

Fig. 8. A view of the pyramidal algorithm, showing the levels used (only 3 levels) and the corresponding output paths.

Fig. 9. Suburban area. Large and well defined roads.

Table 1 shows the success rates (overall: 73%) for a gray-scale version of the images, without using the road-tracing pre-processing. For each map we have selected a mix of easy and difficult paths. The success criterion was whether the path was at least 95% on roads. In all cases, start and arrival points were defined on roads.

Introducing the template-matching-based road-tracing pre-processing leads to a 100% success rate for all the images, and the path is always completely on the roads. This result and improvement is impressive in particular in the case of the Kabul map. The alternative pixel-based pre-processing, instead, produces no improvement.

We have made the initial tests with gray-scale images because in some particular cases (e.g. in war territories) color images may not be easily available. If we use the color versions of the images, again, we get a 100% rate for all the images, and the path is always completely fully on the roads. The difference – with respect to grayscale images – is that pixel-based pre-processing, exploiting three information channels instead one, leads to complete success as well, with a time gain from 30 to 3 s.

Fig. 14 shows a sample path computation in the Kabul area. Even if the street pattern is not very different from the buildings around, the path is never off-road.

As we have anticipated, the algorithm can consider also off-road paths. Fig. 15 shows a case in which the starting and destination points are not connected by any road/track. Even in this case the algorithm is able to recognize the lake and move along the shore instead of crossing it, different from the A$*$ heuristic would suggest. Even if the two points are chosen closer to each other – which increases the effect of the heuristic component in A$*$ – a non lake-crossing path is found again.

Fig. 10. Metropolis. Narrow and winding roads.

Fig. 11. Residential area. Winding, tree avenues (with noise due to shadows).

Fig. 12. Residential area. Narrower, but less winding roads (the original image was not in grayscale).

Fig. 13. Central area in Kabul. Only dirty roads, difficult to distinguish also at naked eye.

Table 1
Success rates for the five test maps (15 paths per map).

Image description	Rate (%)
1 – Suburban area. Large and well defined roads	73
2 – Metropolis. Narrow and winding roads	67
3 – Residential area. Winding, tree avenues (with noise due to shadows)	73
4 – Residential area. Narrower, but less winding roads	93
5 – Central area in Kabul. Only dirt roads, difficult to distinguish also at naked eye	60

Fig. 14. A sample path in the Kabul area.

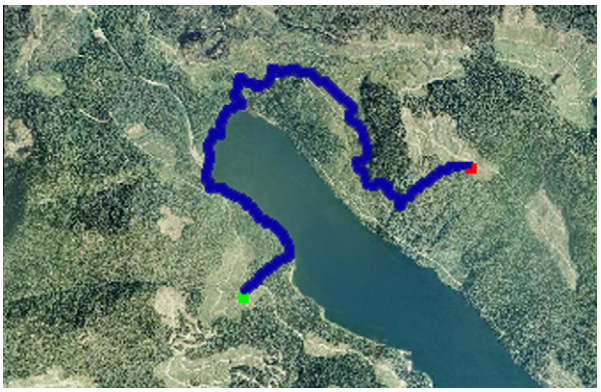

Fig. 15. Applying RealPath in a naturalistic area.

Fig. 16 shows a situation that is problematic in residential areas and rural roads. It is given by the presence of trees that completely cover the street, dividing it in two separated branches that may prevent the identification of the shortest path or even isolate some areas. In 100 paths computed in Fig. 16 with random source and destination points, this problem causes a 30% shortest-path failure.

In order to solve the problem we have introduced the following modification in the pre-processing phase. If a path comes to an end (block) along a direction, then it probes the terrain at a maximum distance of n pixel along that direction. If the terrain there is recognized as a road, then the distance is completely considered as an occluded road. Of course, while this addition may solve some problems, it may also add some spurious paths, for instance by joining two road branches really separated by an obstacle. The maximum length of the distance n is heuristically estimated as equal to the

Fig. 16. Trees occlude the road (in two parallel segments), preventing the identification of the shortest path.

width w of the road in proximity to the point where it reaches the obstruction.

The new version of the algorithm allows reaching an optimal path in the case of 79% of the previously failing paths. To investigate the possible addition of spurious paths, we tried again 100 random test paths, observing 15% spurious paths because of the addition of some small spurious segments.

Fig. 17 shows the image after the tree removal procedure obtained with $n = w$. The figure indicates the removed trees with two violet ovals. We recall that black pixels in the image do not necessarily mean obstacles. They will be processed by the A$*$ algorithm with their own real color (i.e. non pre-processed). Road segments not traced as roads (e.g. in the mid-upper right part of the image) have pixel values near to white and will be used in the A$*$ shortest paths. Tree-covered road segments, instead, were critical and needed the specific additional pre-processing, because they were dark and different from the other road segments close to them.

In order to identify the best value for the heuristic parameter n, we have investigated three cases: (1) $n = w/2$; (2) $n = w$ and (3) $n = 2 * w$, where w is the estimated width of the blocked road. Table 2 shows that good results are obtained with $n = w$. Smaller values tend to solve a lower number of critical cases, higher values of n introduce spurious segments. This is shown in the last column of Table 2, that reports the percentage of spurious paths in an independent 100 random path computation test.

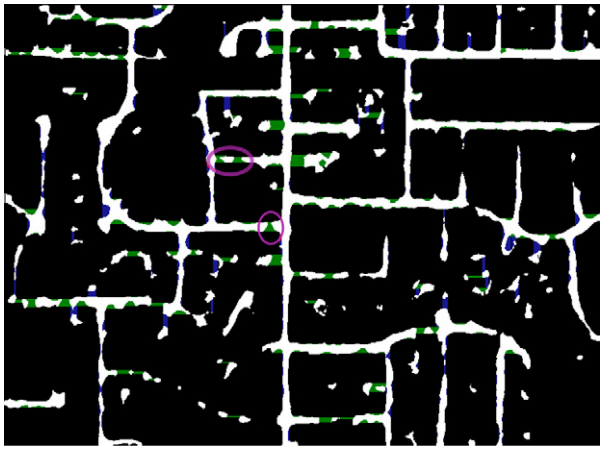

Fig. 17. The tree removal procedure allows to cancel the two obstacles. Green portions indicate horizontal filling, blue vertical. The violet ovals indicate the removed trees.

Table 2
Percentage of paths become correct after the tree-removal procedure using different values of the pixel distance parameter and percentage of added spurious paths in an independent experiment.

Pixel distance	Resolved cases (%)	Spurious paths (%)
$n = w/2$	48	4
$n = w$	79	15
$n = 2w$	91	80

Considering the time performance, the average overhead introduced by the tree-removal procedure is in the order of the second.

4.2. Time performance of the path computation

In order to test the time performance of the path computation algorithm (i.e. not considering the image pre-processing, that is performed before the actual start of the game), we have implemented two additional tests. Results have been obtained on a standard PC platform, with the following features: CPU: Intel Dual Core E6600 2.4 Ghz; Ram: 2 GB, 800 Mhz; 8 MB cache; Operative System: Windows 7; Language used: C++, single thread, single task program.

The first one investigates the advantage of exploiting a hash table for the A$*$ Open List to speed-up the search phase. Results are reported in Table 3, for the case of a 63 pixel distance between the departure and arrival point. The hash table introduces a 3$\times$ speed-up factor, and a 1.4$\times$ memory footprint increase. Execution times show the real-time behavior.

A second test has been defined in order to analyze how the computation time varies with sub-path length. In Fig. 20, the x-axis shows the number of path segments (subpaths) into which every path is subdivided; the y-axis represents the computation time. The lowest computation times occur with a number of sub-paths between 20 and 50. For smaller numbers, we have fewer and longer sub-paths, that imply a low level of parallelization. For larger numbers, we have too many sub-paths with respect to the available functional units. We expect that a higher level of available hardware parallelization (e.g. through the latest GPUs) would allow achieving better results with larger sub-path numbers.

5. Applications

RealPath has been devised as a tool to be integrated in the game level editor of Real Time Strategy (RTS) games. With RealPath, a designer (or an end-user himself) can load geographic pictures (aerial or satellite images or ortophotos, symbolic pictures) to build game maps (e.g. to place resources or define a battle field). The input image may represent a rural environment with trekking paths, or a city with roads and squares, etc.

With current state of the art RTS games, a player's task of collecting resources may become repetitive and boring as soon as he has learnt the position of resources and the existing paths in the map. Also the strategies employed in the same battle fields may become trivial for skilled users. For this reason, RealPath aims at improving the game's longevity by supporting a simple and automatic generation of new game maps.

Table 3
Performance analysis with the hashtable (average on test 100 iterations).

Mode	Time (ms)
With hashtable	1
Without hashtable	3.6

Commercial RTS games are usually enhanced with tools that allow amateur communities to create new contents (e.g. weapons, characters, enemies, models, textures, levels, etc.), to implement modified versions of the games (mods). Since RealPath does not require specific competences (e.g. programming or game design), it can be used to support the User-Generated Content (UGC) modality for map generation. Other path design methodologies, like waypoints manual positioning or grids, need an ad-hoc work for every single map. RealPath users, instead, can simply insert their own images (e.g. their favorite geographic places) in the game to play in new, appealing situations without any design work.

5.1. The RealPath serious game

We have developed the RealPath algorithm in the context of an RTS Serious Game targeting two kinds of applications. The first one concerns military scenarios (Fig. 18); the second one, civil protection management (Fig. 19).

The RealPath SG's mechanics are similar to classic RTS games. There are two competing teams that have a base camp each, where new units (soldiers and workers) are generated. The goal of the player is to keep the camp alive. In order to create units, the player has to gather resources, that can be spent to recruit new soldiers or workers. The former can defend the resources and attack the enemies; the latter can collect resources. As in typical RTS games, the player can choose among various gaming strategies for winning (e.g. more or less aggressive/defensive).

The novelty of the game is given by its ability of dynamically processing input map images. At the beginning, the player is asked to place its base camp on a road. This is the trick we have employed in order to get information for the image pre-processing step. After this phase, the map is divided in sectors, and resources are automatically placed on road segments. This feature makes it strategic for a player to control the main roads and crossroads, which is key in a real-world conflict, but very seldom considered in an RTS. The automatic placement of resources is useful for the longevity of the game, even in the case in which the set of input images were predefined, since it would anyway guarantee a certain degree of variety.

From a pedagogical point of view, the added value of the game is given by the fact that the decision strategy training is combined with the possibility for a player to contextualize his experience in the real target area.

A significant capability of the algorithm is the possibility of integrating information from the satellite image with data from influence maps. An influence map records the game status of each geographic point. Typically, it records the positions of a team's

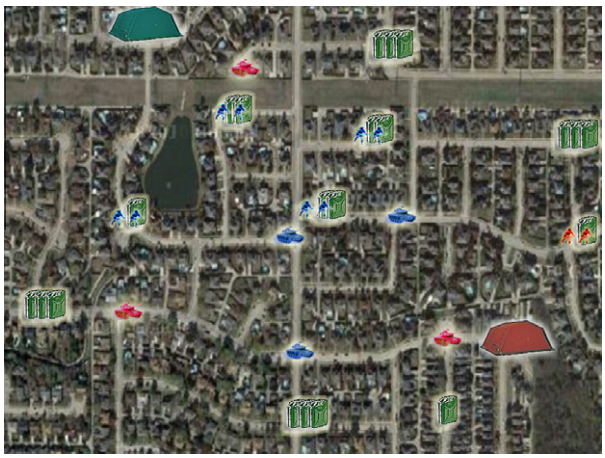

Fig. 18. Snapshot from RealPath SG military scenario.

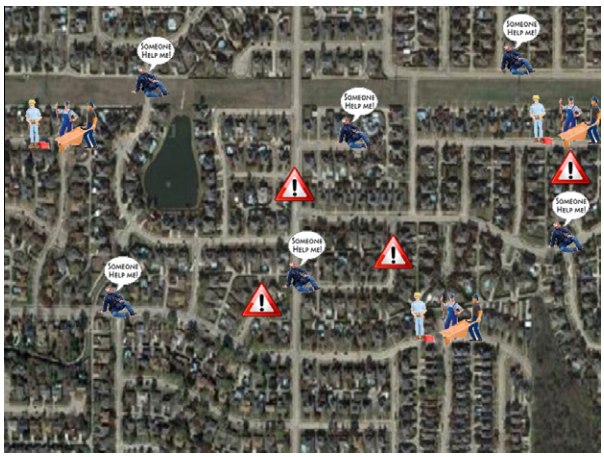

Fig. 19. Snapshot from RealPath SG civil protection scenario.

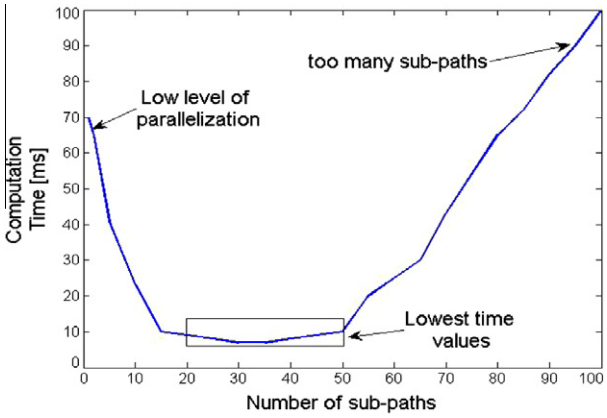

Fig. 20. Execution time vs. number of sub-paths into which every path is divided.

units (e.g. soldiers) and where they are killed. Presence of companions has a positive weight on a point and its neighbors (estimating this as a safe place), while injuries and murders imply a negative weight. Thus, every point in the geographic map is weighted according to the influence map, so that the most critical points (and their neighbors as well) are penalized. This approach allows implementing an AI that adapts to the battlefield conditions and modifies the strategies relying on what it learnt from its previous experience.

The civil protection scenario (Fig. 19) presents a geo-localized rescue operation situation. The pedagogical goal of the game is to train the user in managing resources (rescue teams) in order to save as many injured people as possible. In this case, there is no base camp, but there are few player-moveable rescue teams that are spread on the map by the same algorithm employed for placing resources in the previous scenario. Injured people are generated along road segments as well. The scenario also involves that some roads in the network are blocked, with a symbol simulating the damage of a destructive event (e.g. an earthquake, flood, attack). This, again, is the added value of the game, as it allows realistically training personnel on strategies in a real-world environment. Injured people will be always more than the teams – even in the simpler game sessions. This will draw the player to move his teams considering not just the current move, but also the next ones, since a bad command could avert a rescue team from the majority of the other injured that he will have to assist. This is even more critical because of the presence of obstacles, that may make far to reach destinations that could appear close on the map, if the roads were all working. Moreover, the time allowed for a session is limited, which spurs the player to decide promptly and in a way that he can assign as many missions (i.e. reaching injured people) as possible, considering the traveling times for the teams.

An early prototype version of the game is available online.[7]

5.2. Application in 3D worlds

Beside the employment in 2D RTS applications, we have applied the RealPath algorithm in the context of a 3D virtual environment, by developing a simple XNA[8] demo for a New York city reconstruction. Beyond the definition of paths for Non-Player-Characters (NPCs) in 3D settings, that are ever more used for realistic serious games and simulations, the module has been integrated with two additional components: one for orientating the NPC movement and the other one for avoiding collisions with moving obstacles (static obstacles are directly tackled by the RealPath algorithm).

6. Conclusions and future works

In this paper we have proposed a new family of Real Time Strategy (RTS) serious games that exploit a dynamic insertion of aerial/satellite images in games, in order to allow covering any target geographic area on user demand and automatically implementing several different gaming sessions. The approach intends to combine the possibility for the user of practicing decision strategies with the development of knowledge about specific geographical areas, which is important in particular to enhance the training of field operation personnel.

This target requires that the system correctly interprets the terrain features – in particular roads – in order to build a semantic correspondence between the game's logic and the background image.

RealPath is a new solution for automatic definition of paths on aerial/satellite images, which reduces the cost for creating new game maps, also supporting User Generated Content. A high-performance parallel implementation has also been developed, in order to take advantage of the current computer architectures' and of the additional computational power offered by the latest Graphic Processing Units (GPU), that include hundreds of functional units. The algorithm has been successfully tested with several real-world images.

At present, due to the availability of parallel computation on consumer GPU pipelines (that can be used also for AI, not only for graphics [24]) we are porting the algorithm on NVIDIA's CUDA programming environment [25] in order to further exploit the pyramidal version of RealPath.

A further development will consist in assessing a different parallelization approach, characterized by subdividing the map area in tiles and assigning to each tile a uniform gray scale (or multi-chromatic) value. This could be obtained using a quantization of the image with few color levels. The set of tiles will be used by RealPath as a first map to compute the overall path. Further sub-paths will be computed on each single tile, independent of each other.

Acknowledgments

We are grateful to the anonymous reviewers and the editor for their comments that have significantly contributed to increase the quality of the paper.

[7] http://www.elios.dibe.unige.it/index.php/demo.
[8] http://creators.xna.com.

References

[1] RUSE News – Understanding the game design (Part 2). Available from: <http://ruse.uk.ubi.com/index.php?page=news&newsid=10045> (accessed 9.10.09).
[2] H.D. Cheng, X.H. Jiang, Y. Sun, Jingli. Wang, Color image segmentation: advances and prospects, Pattern Recognition 34 (12) (2001) 2259–2281.
[3] A. Nareyek, AI in computer games, In Queue 10 (No. 1) (2004) 58–65.
[4] M. McPartland, M. Gallagher, Learning to be a Bot: reinforcement learning in shooter games, in: Proceedings of the Artificial Intelligence in Interactive Digital Entertainment (AIIDE), Stanford, USA, 2008.
[5] Y. Bjornsson, K. Halldorsson, Improved heuristics for optimal path-finding on game maps, in: Proceedings of the Artificial Intelligence in Interactive Digital Entertainment (AIIDE), Marina del Rey, California, 2006.
[6] J. Matthews, Basic A∗ pathfinding made simple, in: AI Game Programming Wisdom, Charles River Media, Hingham, MA, 2002.
[7] P.E. Hart., N.J. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, in: IEEE Transactions on Systems Science and Cybernetics SSC4, 2008, pp. 100–110.
[8] Millington, J. Funge, Artificial Intelligence for Games, second ed., Morgan Kaufmann, Burlington, 2009.
[9] R.E. Korf, Depth-first iterative-deepening. An optimal admissible tree search, Artificial Intelligence 27 (1) (1985) 97–109.
[10] P. Yap, Grid Based Path Finding, Proceedings of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence (2002) 44–55.
[11] R. Graham, H. McCabe, S. Sheridan, Pathfinding in Computer Games, Institute of Technology Journal (2003) 1–10.
[12] L. Liden, Using nodes to develop strategies for combat with multiple enemies, in: Artificial Intelligence and Interactive Entertainment: Papers from the 2001, 2001.
[13] G. Snook, Simplified 3D movement and pathfinding using navigation meshes, in: Game Programming Gems, 2000, Charles River Media.
[14] D. Hamm, Navigation mesh generation: an empirical approach, AI Game Programming Wisdom, Charles River Media 4 (2008) 113–124.
[15] A. Patel, Map Representations, in: Theory at Stanford University, June 2009.
[16] J.E. Doran, D. Michie, Experiments with the Graph Traverser, Proceedings of the Royal Society of London 294 (1966) 235–259.
[17] Partial Pathfinding Using Map Abstraction and Refinement, Nathan Sturtevant and Michael Buro, in: Proceedings AAAI-2005, July, 2005.
[18] Improving Collaborative Pathfinding Using Map Abstraction, Nathan Sturtevant and Michael Buro, AIIDE-06, Marina del Rey, CA.
[19] K. Vojco, A. Aleksandar, S. Dragan, S. Mile, K. Tatjana, Intelligent control of an autonomous vehicle: image processing and pathfinding problem, IEEE Automation Congress, 2004, Proceedings, World 17 (2004) 131–136.
[20] R. Brunelli, Template Matching Techniques in Computer Vision: Theory and Practice, Wiley, 2009.
[21] S. Rabin, A∗ aesthetic optimizations, in: Game Programming Gems, Charles River Media, Hingham, 2000, pp. 264–271.
[22] D. Hearn, M.P. Baker, Computer Graphics (C Version), Prentice Hall, 1997.
[23] E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, J.M. Ogden, Pyramid Methods in Image Processing, 1984.
[24] J. Shopf, J. Barczak, C. Oat, N. Tatarchuk, March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on GPU", in: Proceedings of the ACM SIGGRAPH 2008, Los Angeles, CA, 2008.
[25] Nvidia Corporation: CUDA Compute Unified Device Architecture Programming Guide. Available from: <http://developer.nvidia.com/cuda> (accessed March 2009).