Applying 3D Techniques to Compute Views Provided by Video-Cameras

Roberto Yus, Eduardo Mena, and Sergio Ilarri

IIS Department, University of Zaragoza
Maria de Luna 1, 50018, Zaragoza, Spain
{ryus,emena,silarri}@unizar.es

Abstract. There exist contexts where managing several video-cameras is required, for example in the broadcasting of sports events. To identify the objects that a certain camera views (e.g., when there are multiple objects in the focus) or obtaining the kind of view (front, top, etc.) that a camera is capturing of an object are challenging tasks if we want to perform them automatically.
In this paper, we describe a set of 3D techniques that can be applied to obtain the view of a camera. More specifically, they can be used to obtain the objects viewed by a camera, the kind of view of an object retrieved by the camera, and the percentage of the object or the view of an object that the camera is capturing. These techniques use a 3D model of the scenario and a 3D engine. Many dynamic factors are taken into account, such as the location of interesting objects in the environment, camera parameters, and possible occlusions, among others.

Keywords: Multiple camera management, 3D view analysis, content selection in run-time, data management in 3D spaces

1 Introduction

Nowadays, a lot of video-cameras with competitive prices are available in the market. Thus, those video-cameras can be part of information systems and provide rich multimedia content. It is possible to build a system that provides information about a multi-camera environment, which could be useful for example to help technical directors in broadcasting sports events. Such users could want to select among the views provided by the different cameras. For example:

- To retrieve automatically the objects viewed by a camera. An automatic identification of the several objects captured by a camera could be very useful for example when it is difficult to identify them due to the long distance.
- To obtain a certain kind of view of an object. When dealing with a multi-camera environment, it is possible that several cameras view the same object, but from different angles. Thus, front/top, rear, right side/front, etc. views of the same object can be available from different cameras.

– To view at least a certain percentage of an interesting object. A partial view of the object can happen 1) when it is occluded by other objects or 2) when it does not fit the field of view (for example if the target is too close).
– To view at least a certain percentage of a given kind of view of an interesting object. This case is a combination of the two previous ones, for example when the technical director is interested in cameras viewing at least 50% of the top of an object.

Developing such a system implies facing a number of challenges. Not only the information about the features of cameras must be taken into account, but also the location of both objects and cameras and, as explained before, the possible occlusions.

In this paper, we propose a 3D approach that overcomes these difficulties. The main features of our proposal are:

– It manages information about the features of video-cameras to decide which of them satisfy the requirements of the user.
– It allows the user to obtain the objects that a camera is currently focusing, the kind of views (top, front, etc.) of the objects provided, and the percentage of the object or the view of an object of interest viewed.
– It considers the 3D representations and locations of the objects defined by the user.
– It supports the detection of occlusions caused by other objects or by the camera limitations.

To perform these tasks, the system manages a 3D model and uses an existing 3D engine to obtain 2D projections of such a 3D model. The rest of the paper is structured as follows. In Section 2, we introduce some concepts about video-cameras and objects in the scenario and how they are modeled in our approach. In Section 3, we explain how to obtain the kind of view of an object that a camera is capturing. In Section 4, we present 3D techniques to compute the percentage of a target object that a camera is viewing. In Section 5, we explain how to compute the percentage of a certain view of a target object that a camera is viewing. In Section 6, we review some related work. Finally, conclusions and future work appear in Section 7.

2 Monitoring Multimedia Data: Management of Video-Cameras

In this section, we first introduce some basic concepts about video-cameras, explaining their parameters and limitations. Then, we explain how multi-camera scenarios are modeled and managed in our approach.

2.1 Video-Camera Features

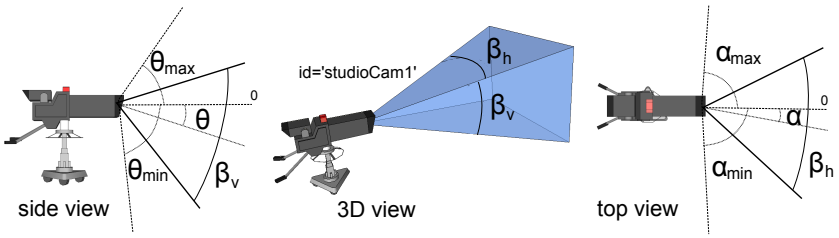
There are several parameters that define a video-camera. Among them, the *field of view* is one of the most important, as it determines the extent that the camera

will see. The field of view of a camera is called in 3D computer graphics *viewing frustum*, and denotes the region of space in the modeled world that the camera is capturing.

Some cameras are also defined by its horizontal turning capability (called *pan*) and its vertical turning capability (*tilt*), along with the corresponding pan and tilt speeds. The pan and tilt range of a camera are physically limited by its maximum and minimum pan and tilt. Cameras have other parameters (such as zoom, focus, etc.), but they will not be considered in the proposal presented in this paper.

2.2 Modeling the Scenario

We model a camera c as shown in Figure 1. In the figure, we identify several elements: id is a unique identifier; β_h and β_v are the horizontal and vertical angle of view, respectively; α, α_{max}, α_{min}, and α_{speed} are the current pan, the maximum pan possible, the minimum pan possible, and the pan speed (degrees/second) of such a camera, respectively; finally, θ, θ_{max}, θ_{min}, and θ_{speed} are the current tilt, the maximum tilt possible, the minimum tilt possible, and the tilt speed (degrees/second), respectively. Considering both the pan and the tilt of a camera is needed to manage a 3D space containing 3D objects.



$$c =< id, \alpha, \alpha_{max}, \alpha_{min}, \alpha_{speed}, \theta, \theta_{max}, \theta_{min}, \theta_{speed}, \beta_h, \beta_v >$$

Fig. 1. Modeling a video-camera

We model objects in the scenario by representing them in a 3D model, considering their *extent*. This extent is represented by a 3D mesh created by using an external modeling package (e.g., *Autodesk 3ds Max*, *Blender*, etc.). Also the locations of the objects are needed. This information can be obtained from a database that supports efficient real-time updates in dynamic environments. Besides, the normal front and top vectors are defined for each object in order to distinguish between the different kinds of views of the object (top/bottom, front/rear, left/right side, or any combination of two or three of these elements from different pairs). As an example, Figure 2 shows a 3D mesh representing a rowing boat, its top and front vectors, and some examples of views that we could obtain for that object.

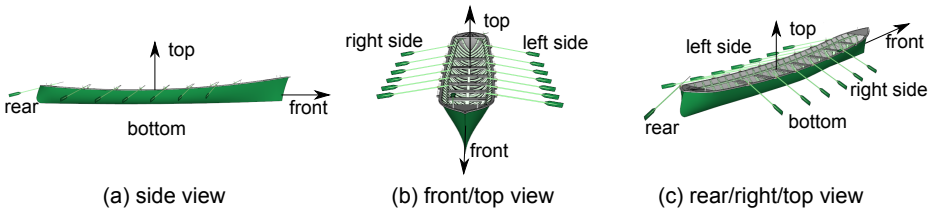


Fig. 2. Different views of an object

Notice that our approach is applicable to any scenario where the location of (moving) objects (no matter if they are vehicles or people) and their extents can be obtained (somehow) in real-time. To compute what a camera is viewing, we recreate the scene in a 3D engine and use 2D projections (obtained by rendering the 3D scene) to calculate the percentage of an object that is viewed by each camera. For this purpose, we have used *JMonkeyEngine* (JME)[1], a Java game engine with advanced graphics capabilities supported through *OpenGL 2.0* via *LWJGL*[2]. JME enables us to model cameras by setting the parameters explained in Section 2.1. For example, we can set the field of view of a camera by defining the planes that compose its *viewing frustum*. It also allows us to add 3D models of objects using several 3D formats (*OGRE XML*[3], *Wavefront OBJ*[4], etc.). Thus, with this engine a real scenario can be simulated.

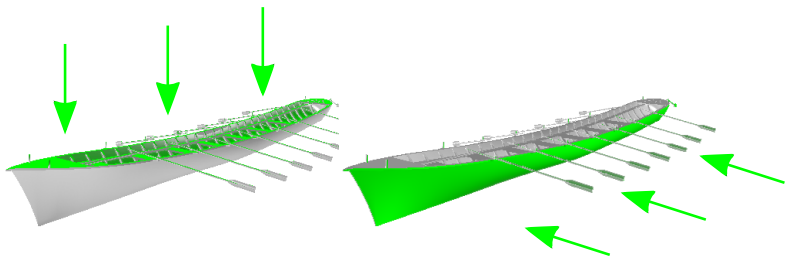
Identifying the objects that a camera is viewing can be achieved by computing intersections between the 3D mesh and the planes composing the viewing frustum of such camera (usually 3D engines provide methods for this). However, this is not the main goal of our proposal. Instead, we present methods to retrieve the kind of views of an object that a camera is obtaining in Section 3, the percentage of an object that a camera is covering in Section 4, and a combination of the two previous ones in Section 5.

3 Computing the Kind of View of an Object

A user could be interested in obtaining the kind of views that a camera is retrieving of a target object. For example, it could be interesting to obtain the cameras that are providing a rear/right side/top view of a rowing boat, as seen in Figure 2. The 3D model of the objects can be complex, and therefore selecting the parts of the target that belong to a view can be a difficult and time-consuming task. So, we have used an approach based on *light sources* and *illumination*. Our proposal is to use a directional light source[5] as the only light in the scenario,

[1] `http://www.jmonkeyengine.org/`
[2] *Lightweight Java Game Library*: `http://www.lwjgl.org/`
[3] `http://www.ogre3d.org/`
[4] `http://en.wikipedia.org/wiki/Wavefront_.obj_file`
[5] Directional light sources have no position, only a direction. They are considered "infinitely" far away and send out parallel beams of light.

setting its direction according to the vector corresponding to the desired view. In this way, we illuminate only the parts of the object that belong to the view. The procedure is similar to the use of a flashlight in a dark room: if we point the flashlight to an object inside the room, only the parts being illuminated will be visible.



(a) Top view selection (b) Left side view selection

Fig. 3. Selecting the pixels of the image belonging to the top view of the target object (a) and to its left side view (b)

Figure 3 shows an example of this technique, where a light source has been added to "select" the parts of the target object that the camera is focusing and belong to the top view (Figure 3.a) and to the left side view (Figure 3.b). Notice that the rest of the object, that is not visible, has been colored in gray to help understand the current view.

4 Computing the Percentage Viewed of a Target Object

A challenging problem when dealing with video-cameras is to take into account possible occlusions. This is an important issue because considering only the positions of the objects could lead us to add cameras to the answer set that are focusing the object but are not having a good view of it (even they could not view the target at all). For example, a camera could be focusing an object but a wall between them would make it invisible for the camera. An incomplete view of an object occurs in two situations:

1. The target is partially or fully occluded by another object.
2. The target does not fit the field of view of the camera (because of its size or the camera *viewing frustum*).

In the following, we detail these cases.

4.1 Occlusions Caused by Other Objects

We need a mechanism to deal with occlusions caused by other objects located between the camera and the target. For this purpose, we use pixels as metric for

counting the amount of the target that a camera is viewing. A first approach one can think of consists of the following steps:

1. Obtain an image of the current view of the camera, setting the scene with the target object alone.
2. Count the number of pixels of the target being shown.
3. Add all the other objects to the scene.
4. Count the number of pixels of the target still being shown.
5. Calculate the percentage by comparing both counts.

The main problem of this method is that it needs to use two images rendered from the 3D scene, and obtaining them is an expensive task that involves both the graphic card and the CPU. Thus, for a highly dynamic environment the number of such expensive tasks has to be minimized.

Therefore, the approach proposed uses a single image rendered from the 3D scene to compute both the pixels of the target being shown and the pixels occluded. This is achieved by setting different colors for the target object (blue) and the others objects (green) in the scenario and setting the transparency of these other objects to 100%.

We consider a pixel as composed of four color channels (*red, blue, green* and *alpha*). Thus, if a certain pixel has its blue channel distinct from zero it will belong to the target object, and if the green channel is also distinct from zero, it will be an occluded pixel. If the same part of a target is occluded by two or more objects there is no problem, as the green channel of an occluded pixel will be the same for any number of occlusions (for an example, see Figure 4.a.2, where the method computes that the camera is viewing a 19% of the target object).

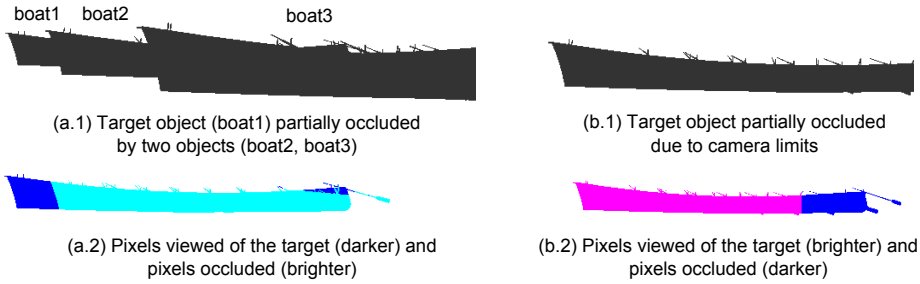


Fig. 4. Incomplete view of an object caused by: occlusions by other objects (a) and the target does not fit the viewing frustum (b)

4.2 Objects that Do Not Fit the Viewing Frustum

If the object intersects the viewing frustum it means that we have to somehow compute the total number of pixels of the object, even if they are not in the shot,

to be able to compare them with the pixels currently being viewed. Assuming that the pixels inside the viewing frustum are all the pixels of the object would lead us to erroneously obtain the percentage being viewed; more specifically, we would consider that the camera is focusing a greater percentage that the real one.

There are several approaches to deal with a situation like this one where the target intersects the camera frustum. Rotating the camera to cover the whole object by parts is a natural way to count the total number of pixels of the target object, but a rotation of the camera will change the angle between it and the target, and thus the view will be modified. So, the only constraint to consider is not to rotate the camera. We briefly indicate two possible approaches:

1. *Increase the size of the frustum until it contains the full object.* With this approach, the idea is to count the number of pixels being viewed by the current camera, then increase the size of the frustum, and finally count the total number of pixels of the object.
2. *Move the camera backwards until the target is completely shown.* In this case, the idea is first to occlude the parts of the object that the camera is viewing, then move the camera until the target is completely shown, and finally perform the counting[6].

Increasing the size of the frustum is not always possible, as it depends on the 3D engine used. For example, with the JME, a change of the planes that define the frustum means a change of the resolution of the JME application, which would make the two images obtained not comparable. So, we use the second approach, which is based on a movement of the camera backwards:

1. The objects behind the camera have to be removed from the scene to avoid possible new occlusions.
2. The parts of the object that the camera was viewing have to be "selected" (to distinguish between the pixels of the object occluded and the pixels viewed). To "select" those parts, a 3D representation of the frustum of the camera has to be added to the scene and its color has to be set to red and its transparency to 100%.
3. The camera is moved backwards until the object fits within the frustum, as explained before.

Figure 4.b.2 shows the parts of the target object that the camera is viewing (brighter) and occluded (darker), and the method returns that the camera is currently viewing a 76% of the target object. Notice that a pixel is visible when all the following statements occur:

- Its blue channel is distinct from zero.
- Its green channel is equal to zero.
- Its red channel is distinct from zero (if the object did not fit the viewing frustum).

[6] The percentage being viewed will be $\frac{\#occluded\ pixels}{total\ \#\ pixels} * 100$.

4.3 Method Proposed

We present a method developed to deal with occlusions and objects that do not fit the viewing frustum, in order to obtain the percentage of the target object being viewed by a camera. This method first obtains the situation of the target from the camera point of view (by calling *cam.checkObjectVision(target)*). There are three possibilities:

- *Outside the viewing frustum*: the camera is not currently focusing this object.
- *Intersecting the viewing frustum*: the object does not fit the viewing frustum.
- *Inside the viewing frustum*: the camera is focusing the object and the object fits the viewing frustum.

Then, considering this information, we deal with occlusions to obtain the total number of pixels of the object and the number of pixels of the object viewed by the camera.

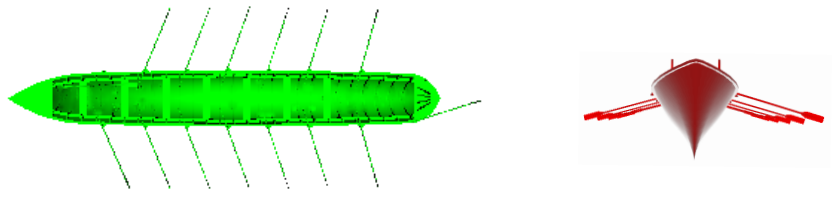
```
program obtainPercentageViewed(target, objects, cam)
--Requires:
--           target: target object of the query
--           objects: all the objects in the scenario
--           cam: camera to check
--Returns:
--           percentage of the target that cam is viewing

  --Check the position of the object in the frustum of the camera
  --(inside, outside or intersects)
   visionOfTarget=cam.checkObjectVision(target);
   if(visionOfTarget == "outside")  return 0;  end if;

   intersects=false;

   if (visionOfTarget == "intersects")
      for all object in objects
         if (cam.objectBehind(object)) --Objects behind the camera are made
            object.setVisible(false);  --invisible
        end if;
      end for;
      cam.createFrustumObject(red,100%); --Occlude the current view of the camera
      while (visionOfTarget != "inside")
         cam.moveBackwards();
         visionOfTarget=cam.checkObjectVision(target);
      end while;
      intersects=true;
   end if;

   for all object in objects
      if (object == target)  object.setColor(blue); object.setTransparency(0%);
      else object.setColor(green); object.setTransparency(100%);
```

```
      end if;
   end for;

   view=cam.renderScene();
   occludedPixels=0; objectPixels=0; visible=0;

   for all pixel in view.pixels   --Analyze the color channels of the
      if (pixel.blueChannel != 0) --pixels of the image
         objectPixels++;
         if (pixel.greenChannel != 0) occludedPixels++; end if;
         if (pixel.redChannel != 0) visible++; end if;
      end if;
   end for;

   if (intersects == true) return (visible-occludedPixels)*100/objectPixels;
   else return (objectPixels-occludedPixels)*100/objectPixels;
   end if;

end
```

5 Computing the Percentage Viewed of a Target View of an Object

Our proposal also takes into account that the user could want to query the system about specific views of an object. For example, he/she could be interested in cameras providing a view that covers a certain percentage of the top of an object. The challenge for this type of queries is the need to obtain the parts of the target that belong to the view (explained in Section 3) and to obtain the shot that would cover 100% of the view selected (in order to compute the percentage actually covered).

To obtain the shot that covers 100% of the target view, we need to set a camera pointing in the same direction as the desired view but at the same distance to the target object than the real camera that we are considering (to have comparable values). Figure 5 shows the shots that would cover 100% of the top view (Figure 5.a) and front view (Figure 5.b) of the target object.



(a) Top view (b) Front view

Fig. 5. Shots covering 100% of the top (a) and front (b) views of a target object

We have developed a method to solve both problems, which computes the percentage that a camera is viewing of a target view of an object. This method obtains first what the camera is viewing, taking occlusions into account:

1. Set the color of the objects that are not the target to black. The parts of the target object being occluded have to remain invisible.
2. Create directional lights for the views. This method is able to check several views at the same time by using different colors for the light sources (using a similar approach to the one explained in Section 4.1 about analyzing the color channel of every pixel).
3. Move the camera in the direction of the view and at the same distance to the object, and obtain an image that covers 100% of the view.
4. Return the percentage that the camera is covering of each view entered as a parameter.

As explained before, the method can consider several views per query at the same time. The example of Figure 6 shows an image rendered by the system for the current view of a camera focusing the target object when the user wants to obtain a certain percentage of its front and top. Notice that there are different intensities of red and green in the image, as depending on the normal of the corresponding polygon the *illumination method* (*Phong* is used in JME) makes it look darker or brighter. This is not a problem for our approach because it counts pixels that have a nonzero value for that specific channel. It is interesting to notice also that some parts of the paddles are green (those parts belong to the top view, as seen in Figure 5.a) and others are red (because they belong to the front view). Using this image and the ones in Figure 5 (representing the shot that covers 100% of the top (a) and front (b) parts of the object, the system computes that the camera is currently viewing a 41% of the front and a 78% of the top views of the target.

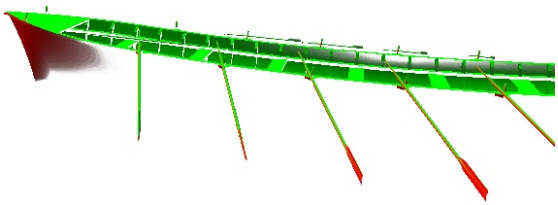


Fig. 6. Image showing the 41% of the front and the 78% of the top views of an object

A more detailed version of the method proposed is included on the following:

```
program obtainPercentageOfViewPoints(target, objects, cam, viewPoints)
--Requires:
--           target: target object of the query
```

```
--           objects: all the objects in the scenario
--           cam: camera to check
--           viewPoints: views that are going to be checked
--                              (e.g., front and top)
--Returns:
--           percentage of every view in viewPoints of the target
--           that cam is viewing

   for all object in objects --Take into account possible occlusions
      if (object != target)
        object.setColor(black); object.setTransparency(0%);
      end if;
   end for;

   colors={blue,green,red}; i=0;

   for all viewPoint in viewPoints --Create directional light sources
      createLight(viewPoint, colors[i]); i++;
   end for;

   currentView=cam.renderScene();

   --Count the number of pixels of every viewPoint being viewed
   for all pixel in currentView.pixels
      if (pixel.blueChannel != 0) pixelsViewing[0]++; end if;
      if (pixel.greenChannel != 0) pixelsViewing[1]++; end if;
      if (pixel.redChannel != 0) pixelsViewing[2]++; end if;
   end for;

   d=cam.distanceToObject(target); i=0;

   --Obtain the shot that will cover 100% of every viewPoint and
   --compute the percentage being viewed
   for all viewPoint in viewPoints
      cam.align(viewPoint,d);
      perfectView=cam.renderScene();
      for all pixel in perfectView.pixels
        if (pixel.colorChannel[i] != 0) totalPixels[i]++; end if;
      end for;
      percentages.add(pixelsViewing[i]*100/totalPixels[i]); i++;
   end for;

   return percentages;

end
```

6 Related Work

In this section, we briefly describe some related work. We consider works related to the problem of positioning a camera in a virtual world, works that deal with object detection, tracking, and occlusions, as well as works related to image and video retrieval.

The *Virtual camera composition (VCC)* problem (positioning a camera in a virtual world) has been widely studied in *Computer Graphics* (some examples are [1–5, 8]). Where to position a camera to get a desired shot is not an easy task in a 3D environment and these works try to solve this problem. These works are based on the definition of constraints on the camera variables and the numerically solving an optimization problem (obtaining the location of the camera). Although our problem is not exactly the same (in our case the locations of the cameras are already set and obtained by querying a database that tracks the locations of the moving objects and their cameras), there are some common difficulties, such as the need to take into account occlusions or consider when an object is inside the field of view of a camera [1].

Other related works are those that deal with object detection and tracking, for example *Computer Vision* applied to surveillance (some examples are [7, 10, 13, 14]. These works are based on the analysis of the images provided by a real camera, which is a great challenge. The objects involved in these scenarios are usually unknown in advance and the system has to use different techniques to identify them. Moreover, the analysis can be affected by several difficulties, such as the variation of lighting conditions, the possibility of failures in foreground detection, etc. The idea of *closed-worlds* [10], which are regions of space and time in which the specific context of what is inside is assumed to be known, is used to minimize those problems. Occlusions are also an important issue because when a camera is tracking an object it could be occluded for a certain amount of time and this has to be taken into account to estimate trajectories accurately. In [7, 13, 14] a model of the background is used to help the system reason about the occlusions of objects by scene elements. However, obtaining this model can be difficult.

In the field of image and video retrieval in information systems there are numerous works. These works deal with the same problems that the ones explained in Computer Vision [11]. Visual information (as for example color, texture, shape, etc.) is usually used to compute the similarity between shots [12, 15]. This technique allows the users to express queries such as "images containing 35% red and 27% blue". Other techniques use semantic content associated to the videos to support queries defined in a more declarative manner [6].

Although our work shares some problems with the proposals described in this section, there are also some differences. Specifically, in our scenario the locations, directions of movement, extents (given by the 3D model attached), and other parameters of the interesting objects are known. Thus, we can recreate entirely the scene in a 3D virtual environment avoiding the problems related to Computer Vision regarding the analysis of real images. So, for these scenarios we can obtain more accurate answers and support queries that will be difficult to compute

by using techniques based on image comparison or visual information analysis. Nevertheless, the works performed in Computer Vision on trajectory estimation could be considered and adapted to our context to improve the answers provided to the user.

7 Conclusions and Future Work

In modern information systems, multimedia data represent an interesting source of information to take into account. In this paper, we have presented a general method based on several 3D techniques that can be used to compute what a camera is viewing automatically. The approach is based on the use of 3D models for the objects in the scenario along with information about their locations. The main contributions of this work are the techniques proposed to support several tasks:

- To retrieve automatically the objects viewed by a camera.
- To obtain a certain kind of view of an object, even defying the percentage of the view or the object.
- To take into account occlusions caused by other objects or by the camera viewing frustum.
- To optimize the time required for computations by using efficient methods that minimize the number of expensive tasks needed.

The 3D techniques implemented support obtaining the information about the cameras in a flexible way in order to retrieve cameras that can provide the required views. Our approach is applicable to any scenario where the locations of (moving) objects (no matter if they are vehicles, people or others) and their extents can be obtained (somehow) in real-time.

Furthermore, we have performed several tests (not included here due to space limitations) to prove the methods explained in this work. In these tests[7], check the cameras viewing at least a 70% of an object take between 0.3 and 0.6 seconds (0.4 seconds on average) in a scenario with seven cameras and four moving objects.

As future work we plan to use the techniques explained in this paper to improve the continuous location query processing system explained in [9]. In this system the cameras fulfilling the queries expressed by a technical director are shown to him/her and updated every second. So, the potential and the efficiency of the techniques explained in this paper become very important.

Acknowledgments. This research work has been supported by the CICYT project TIN2010-21387-C02-02. We also thank Francisco J. Serón for his technical support.

[7] Tests run on an *Intel Core i5-480M* with graphics card *NVIDIA GeForce GT 540M.*

References

1. W. H. Bares, J. P. Grégoire, and J. C. Lester. Realtime constraint-based cinematography for complex interactive 3D worlds. In *15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 1101–1106. American Association for Artificial Intelligence, 1998.
2. W. H. Bares, S. McDermott, C. Boudreaux, and S. Thainimit. Virtual 3D camera composition from frame constraints. In *8th ACM International Conference on Multimedia*, MULTIMEDIA '00, pages 177–186. ACM, 2000.
3. J. Blinn. Where am I? What am I looking at? *Computer Graphics and Applications, IEEE*, 8(4):76 –81, July 1988.
4. D. B. Christianson, S. E. Anderson, L.-w. He, D. H. Salesin, D. S. Weld, and M. F. Cohen. Declarative camera control for automatic cinematography. In *13th National Conference on Artificial Intelligence*, AAAI'96, pages 148–155. AAAI Press, 1996.
5. M. Christie and J.-M. Normand. A semantic space partitioning approach to virtual camera composition. In *Annual Eurographics Conference*, volume 24, pages 247–256, 2005.
6. M.-S. H. Cyril Decleir and J. Kouloumdjian. A database approach for modeling and querying video data. In *15th International Conference on Data Engineering*, ICDE '99, pages 6–13. IEEE Computer Society, 1999.
7. T. Ellis and M. Xu. Object detection and tracking in an open and dynamic world. In *IEEE CVPR Workshop on Performance Evaluation of Tracking and Surveillance*, PETS '01, 2001.
8. L.-w. He, M. F. Cohen, and D. H. Salesin. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *23rd Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 217–224. ACM, 1996.
9. S. Ilarri, E. Mena, A. Illarramendi, and G. Marcos. A location-aware system for monitoring sport events. In *8th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2010)*, pages 305–312. ACM Press, Austrian Computer Society (OCG), November 2010.
10. S. S. Intille, J. W. Davis, and A. F. Bobick. Real-time closed-world tracking. In *Conference on Computer Vision and Pattern Recognition*, CVPR '97, pages 697–703. IEEE Computer Society, 1997.
11. M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2:1–19, February 2006.
12. C.-W. Ngo, H.-J. Zhang, and T.-C. Pong. Recent advances in content based video analysis. *International Journal of Image and Graphics*, 1(3):445468, 2001.
13. A. Senior. Tracking people with probabilistic appearance models. In *IEEE Workshop on Performance Evaluation Tracking Surveillance*, PETS '02, pages 48–55, 2002.
14. C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:780–785, 1997.
15. Y. Wu, Y. Zhuang, and Y. Pan. Content-based video similarity model. In *8th ACM International Conference on Multimedia*, MULTIMEDIA '00, pages 465–467. ACM, 2000.