Intelligent visibility-based 3D scene processing techniques for computer games

Dimitri PLEMENOS[1], Jérôme GRASSET[2], Benoît JAUBERT[1], Karim TAMINE[1]
[1]University of Limoges, MSI laboratory, 83, rue d'Isle, 87000 Limoges (France)
[2]Institut d'Ingénerie Informatique de Limoges, Rue Sante-Anne, Limoges (France)
plemenos@unilim.fr, grasset@3il.fr, jaubert@msi.unilim.fr, tamine@msi.unilim.fr

Abstract

In this paper we present intelligent visibility-based techniques allowing efficient processing of 3D scenes in order to be used in real time in computer games. The presented techniques allow real time use of 3D scenes either by simplifying the scene models (by suppressing useless details or by using improved image-based modelling) or by off-line automatic camera path pre-computation allowing fast exploration of the game environment.

Keywords: *Visual scene complexity, Good point of view, Virtual world exploration, Image-based modelling, Object simplification.*

1. INTRODUCTION

In computer games, there are two important parameters to be taken into account. The first one is real time motion and changes of the game environment. This requirement may be obtained both by intelligent simplification of the game environment as well as by preprocessing techniques applied to path computing for the camera or for virtual actors of the game. The second parameter is realism in representation of the game environment and its virtual actors. This requirement needs also simplification and preprocessing techniques to be reached.

It is easy to understand that these two important requirements, real time motion and realistic rendering, are contradictory to each other. More the motion is fast and fluid, more the program may use additional time to improve the realism of the game. More time is required for motion, less time is available for realistic rendering.

In this paper we are mainly concerned by real time motion, whereas intelligent techniques are studied to improve image-based modelling in order to obtain more realistic rendering. Three kinds of techniques will be studied, where artificial intelligence-based methods are used to improve results: Intelligent object simplification; intelligent viewpoint computing for image-based scene modelling; off-line pre-computing of a path for a virtual camera exploring the game environment.

In section 2 we will present an aggressive simplification method for moving objects of the game environment, based on suppression of potentially non visible polygons. Section 3 will be concerned by techniques for computing a minimal set of camera positions, in order to improve image-based modelling and rendering while in section 4 preprocessing techniques will be proposed for computing paths for a virtual camera exploring the game environment. In section 5 we will temporary conclude and will present possible future work.

2. OBJECT SIMPLIFICATION

One of the means for reducing computation time during a game, in order to obtain fluid motion, is object simplification. In a game, there are several parts of its environment which are not always visible or they are visible from very far. In such cases, it is important to simplify the models of these parts, in order to accelerate the game display.

One of the well known simplification techniques is the level of details (LOD) technique. A scene, or part of scene, is modelled in various levels of details. For parts of the scene that cannot be well seen, a low level of detail may be chosen, whereas for the well visible parts a high level must be used for display.

In this section we will present another kind of scene simplification which could be used in computer games. This technique is based on rough determination of potentially non visible parts of the scene and suppression of these parts. It is an aggressive technique because potentially non visible parts of moving objects of a scene are roughly determined and some of them may be visible in some cases. This scene simplification technique is presented below.

2.1 Visibility culling

Visibility culling is the generic term for the various techniques aiming at removing the invisible polygons from the list of polygons sent to the display module. It includes *view frustum culling*, *back face culling* and *occlusion culling*.

The existing approaches are dedicated to static scenes. During a pre-processing stage the visibility from "cells" (delimited 3D areas of the scene) is evaluated to get a collection of Potentially Visible Sets (PVS) of polygons. Then, when the camera travels in the scene only the polygons of the PVS associated with the cell where the camera is are displayed.

Current researches focus on several problems, among which:

- how to build an accurate PVS, containing all the potentially visible polygons and only them? The exact methods ([Dur02] and [Nir02]) need lots of computations and memory and moreover are difficult to implement in a robust way. The practical solutions are usually

"conservative": all the visible polygons are included in the PVS, but some invisible are too. As a consequence many researches deal with the possibilities to reduce this number of invisible polygons.
- how to store the visibility data and swap quickly from a PVS to another?

In computer games, visibility culling is used too (see [Tell91] or [Lue95]). For this kind of application the above two problems are critical and may need specifically adapted solutions from general visibility culling algorithms.

Interested readers may refer to [Coh03] for quite an exhaustive survey of occlusion culling techniques and to [Dur00] for a more general survey of visibility problems.

2.2 Visibility culling for moving objects

Existing methods focus on very large but also "very static" scenes. With current approaches, moving objects in static scenes are never simplified, although it is very important for computer games. The purpose of the method presented here is simplification of moving objects.

The method is based on the bounding box of the object to be simplified (see [Gra02] and [Gra05] for first results). It uses a straightforward observation: when we look at an object only the polygons visible through the visible faces of its bounding box can be seen. Thus, if the polygons visible through each face of a bounding box were precomputed the object could be easily simplified "on the fly".

The main goals of the method are:
- give an easy way to compute visibility,
- define an efficient implementation scheme to change the PVS very quickly in order to be applicable to moving objects in real time animations without any freezing,
- optimise the memory cost of the visibility data.

These three goals have the same objective: the method must be practical for real time 3D animation such as in computer games. It is important to note that visibility is not computed for the whole scene but for each moving object independently.

The proposed visibility culling is working as follows:

- Each of the 6 "primary cells" is defined as the half space that is delimited by the plane supporting a face and that does not contain the bounding box. These cells are unusual because they overlap.

- The visibility is computed for any viewpoint of the half space but only through the face of the box defining its border. Here we use the hardware Zbuffer, considering sampling viewpoints on a half ellipsoïd surrounding this face. Each polygon is given a color, and the visibility is computed by reading the colors of each rendered image. For more details on visibility calculation see also section 3.

- The visibility data are coded as a sort of the polygons of the object. 64 sets of polygons are defined, binary numbered from 000000b to 111111b. A polygon is placed in a set according to the faces of the bounding box it is visible through, each digit corresponding to a face. 0b denotes the polygon is invisible through the face, 1b denotes it is visible.
  With this data structure each polygon belongs to one and only one set. It has been implemented with OpenGL display lists: the structure cost is nearly null, it is only the use of 64 display lists.

To perform visibilty culling before the rendering of each frame the visible from the current viewpoint faces of the bounding box of each object are determined and only the corresponding display lists are used.

The above method is an "aggressive" one because two kinds of sampling are used:
- sampling of viewpoints space because only some points of an infinite space are considered,
- sampling of the rendered view used for preprocessing the visibility, due to the finite resolution of the pixel buffer.

For this reason some visible polygons may be missing. In fact, with reasonable sampling rates (about 40 views for each faces and a 1280x1024 resolution) no error is noticed on ordinary 3D objects.

The behaviour of this visibility culling method for moving objects is generally interesting:
- The method is very fast. The preprocessing stage is easy and, thanks to the data structure, there is no freezing at all in the animation when the list of displayed polygons has to change. There is no need of extra time to compute which faces of the bouding box are visible (6 dot products) neither to build the corresponding list of polygons (64 logical "and" operations). So the acceleration ratio is entirely the simplification ratio. This is a very interesting feature. The precomputation time is less than 1 minute for objects of up to 100000 triangles, with a common PC computer.
- The memory cost is negligible: 64 definitions of display list instead of a raw list of polygons, this has no impact on memory. But there may be some drawbacks that should be studied. First of all the approach needs 64 display lists for each object: if there are lots of objects there may be too many display lists for the graphic card abitility. More specifically, display lists are often used to group polygons having the same textures to improve display performance, so if these polygons have to be sorted according to their visibility also it may give quite an intricate data structure.
- The simplification ratio is sometimes not satisfactory. When the list of polygons to be displayed is computed all the polygons visible through at least one of the sampling viewpoint of one on the visible bounding box face are used. Let's suppose that 3 faces of the bounding box are visible, the observer standing on the line supporting a diagonal of the box. In such a case the method displays polygons that may be visible from an nearly opposite viewpoint, just because they are seen through a face that is visible in both cases.

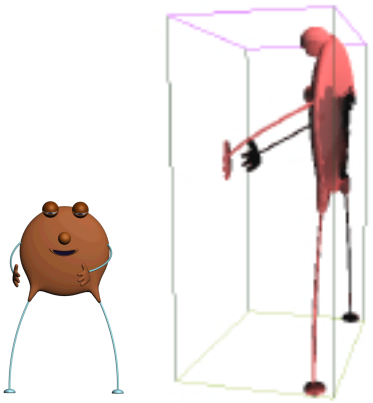

Figure 1: Simplification of a "Condenser"
In front view 32.6% of the polygons are culled.
In diagonal view 21.1% are culled

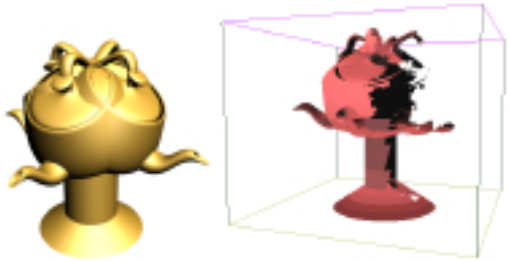

Figure 2: Simplification of a "Tea Fountain"
In front view 65.1% of the polygons are culled.
In diagonal view 39% are culled

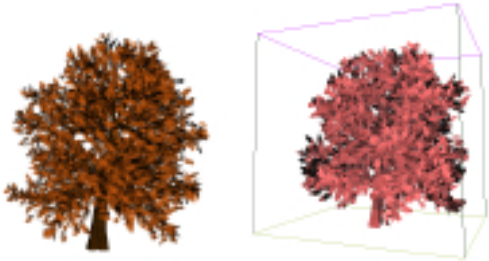

Figure 3: Simplification of a "Tree"
In front view 37.4% of the polygons are culled.
In diagonal view 15.2% are culled

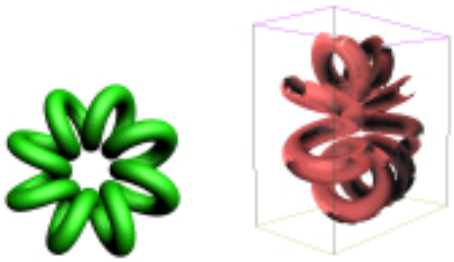

Figure 4: Simplification of a "Rose"
In front view 26% of the polygons are culled.
In diagonal view 11.1% are culled

Figures 1, 2, 3 and 4 show some results of the simplification method, where"front view" is a view from a typical viewpoint showing only one face of the bounding box, and "diagonal view" is the worst case of 3 faces of the bounding box being visible. In these figures diagonal views show the remaining part of the scene after visibility culling for the front views.

3. INTELLIGENT IMAGE-BASED SCENE MODELLING

The purpose of image-based modelling is to replace a scene by a set of images. Display of these images, together with other ones, obtained by interpolation, replaces the classical time consuming visibility determination and hidden surface removal process. How many images are needed to well represent a scene? Searching and loading images may become a time consuming process if too many images are used to model and render the scene. To avoid this problem it would be interesting to compute a minimal set of camera positions allowing to compute a high quality initial set of images for modelling and rendering. It is obvious that a uniform discretisation of the viewpoints space is not a satisfactory solution because the the choice of points of view does not depend on the visual complexity. With uniform discretisation, the discretisation step must be very short in order to be sure that no visible parts of the game environment are lost. Such a method produces too many images , what is prejudicial to the game reactivity. The only reasonable solution is a visual complexity-based selection of camera positions.

Computation of a minimal set of camera positions is not an easy work because, in this computation, we have to choose positions allowing to see as many new details of the scene as possible. In this section we present a method of minimal camera position set computing, based on the notion of visual complexity. The notion of visual (or viewpoint) complexity has received a formal definition in [PSF04] but it was implicitly used since several years in many papers [And04, BDP00a, BDP00b, BDP99, Col88, Feixas02, Feixas99, KK88, PB96, Ple03, PPV01, PPV02, PPV03, PPV03b, PPV03c, PPV03d, PPV03e, Rigau00, Rigau02a, Rigau02b, Sbert02]. Some satisfactory methods to face this problem were proposed, especially in [PPV03d] and [PSF04]. All these methods only take into account geometry of the scene. Of course, other aspects, like lighting, texturing etc., may be important to well understand a game or a scene but geometry is the most important aspect.

We are convinced that light source placement is also an important aspect and we are working to this goal but the methods available today are not satisfactory. Some of the existing methods are based on inverse lighting techniques, where light source positions are determined from the expected lighting result. Among these methods we can mention [PF92], [PRJ97] and [JPP02]. None of these methods is entirely satisfactory, especially because it is not easy to well describe and formalise the expected lighting results. Design Galleries [MAB97] is a general system to compute parameters for computer graphics but computation is not fully automatic. Another not fully automatic system to compute light source positions is presented in [HM03]. The method presented in [Gum02] is based on the notion of *light entropy* and automatically computes lighting parameters but results are

not entirely satisfactory without the help of the user.

In the method of automatically computing of a minimal set of camera position described below, only geometry is taken into account.

Computation of a minimal set of camera positions is performed in 4 steps:

1. Compute an initial set of camera positions by applying uniform sampling on the surface of a sphere surrounding the scene.
2. Using an evaluation function, evaluate each camera position of the initial set and sort the set in decreasing order, starting from the best position and finishing with the the worst position.
3. Starting from the sorted initial camera position set, create a new set by suppressing redundant camera positions, that is, positions allowing to see only details visible from the already selected positions in the set.
4. If the number of camera positions in the final set seems too important, apply once again step 3 to this set in order to get a really minimal set.

The obtained final set of non redundant camera positions can be sorted according various criteria: quality of view, distance from the previous position, etc. The best sorting criterion for image-based modelling is probably distance from the previous position.

3.1 Computing an initial set of camera positions

This step consists in computing a sufficient number of possible camera positions by sampling the sphere surrounding the scene. Uniform sampling of the sphere is performed by increasing the values of two angles θ and φ. Increments $\Delta\theta$ and $\Delta\phi$ determine the sampling precision. In figure 5 one can see three different sampling precisions on the surrounding sphere of a scene.

Figure 5: Sampling of the surrounding sphere to determine initial set of camera positions.

3.2 Evaluating the initial set of camera positions

For each camera position of the initial set, the quality of this position is computed by an evaluation function. The used evaluation function is based on the notion of visual complexity of a scene from a point of view [PSF04] and computes the quality of view on the scene from a camera position.

3.2.1 Evaluation function

Although a quality of view criterion is difficult to define, because the notion of "good view" is partially subjective, a number of objective elements may be retained to closely approximate this notion. The chosen elements are: *number of visible polygons*, *number of visible objects* and *area of projected visible part* of each polygon. The notion of *object* is used together with the notion of surface (or polygon) because very often a scene is composed of triangles with low semantic value. The notion of object, obtained by grouping polygons, is more intuitive and useful because it is not necessary to see all the polygons of an object to consider that the object is visible.

The evaluation function proposed in [PSF04] is the following:

$$I(V) = \frac{\sum_{i=1}^{n} \left[\frac{P_i(V)}{P_i(V)+1}\right]}{n} + \frac{\sum_{i=1}^{n} P_i(V)}{r}$$

where: I(V) is the importance of the view point V,
$P_i(V)$ is the projected visible area of the polygon number i obtained from the point of view V,
r is the total projected area,
n is the total number of polygons of the scene.

In this formula, [a] denotes the smallest integer, greater than or equal to a.

Another evaluation function, proposed in [Sbert02] and based on information theory is very close to the above one:

$$H_p(X) = -\sum_{i=0}^{N_f} \frac{A_i}{A_t} \log\frac{A_i}{A_t}$$

where N_f is the number of faces of the scene, A_i is the projected area of the face i and A_t is the total area covered over the sphere surrounding the scene.

3.2.2 Computing the evaluation function

To compute the main quantities required by the evaluation function, that is number of visible polygons, number of visible objects and area of projected visible part of a polygon, two techniques may be used. The first one is a harware based precise technique whereas the second uses a fast approximated estimation.

Based on the use of the OpenGL graphical library and its integrated z-buffer, the first technique [BDP99, Ple03] works as follows. If a distinct colour is given to each surface of the scene, displaying the scene using OpenGL allows to obtain a histogram (figure 6) which gives information on the number of displayed colours and the ratio of the image space occupied by each color.

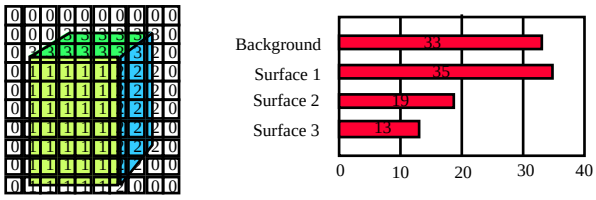

Figure 6: Fast computation of number of visible surfaces and area of projected visual part of the scene by image analysis

As each surface has a distinct colour, the number of displayed colours is the number of visible surfaces of the scene from the current position of the camera. The ratio of the image space occupied by a colour is the area of the projection of the visual part of the corresponding surface. The sum of these ratios is the projected area of the visible part of the scene. With this technique, the two good view criteria are computed directly by means of an integrated fast display method.

In some cases, accurate visual complexity estimation is not requested, either because of need of real time estimation of the viewpoint complexity or because a less accurate estimation is enough for the application using the viewpoint complexity.
In such a case, it is possible to apply a second technique to roughly estimate the visual complexity of a scene from a given point of view, as follows:
A more or less great number of rays are randomly shot from the point of view to the scene and intersections with the surfaces of the scene are computed. Only intersections with the closest to the point of view surfaces are retained (Figure 7).

Now, we can approximate the quantities used in viewpoint complexity calculation. We need first to define the notion of visible intersection. A *visible intersection* for a ray is the closest to the point of view intersection of the ray with the surfaces of the scene.

- *Number of visible surfaces* = number of surfaces containing at least one visible intersection with a ray shot from the point of view.
- *Number of visible objects* = number of objects containing at least one visible surface.
- *Visible projected area of a surface* = number of visible intersections on the surface.
- *Visible projected area of an object* = sum of visible intersections on the surfaces belonging to the object.
- *Total visible projected area* = number of visible intersections on the surfaces of the scene.
- *Total projected area* = total number of rays shot.

The main interest of this method is that the user can choose the degree of accuracy, which depends on the number of rays shot. Selective refinement techniques [Ple87, Rigau02b] may be used to speed up the visual complexity estimation.

3.3 Computing a minimal set of camera positions

Starting from the sorted initial set a new reduced set of camera positions is created in the following manner:

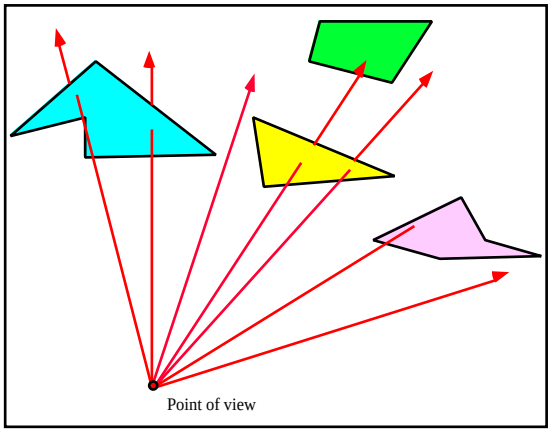

Figure 7: Approximated estimation of visual complexity

The current position in the initial set is compared to all the positions of the reduced set of camera positions. If all the visible details (surfaces and objects) from the current position are visible from the camera positions already stored in the reduced set, the current position is ignored. Otherwise, the current position is stored in the reduced set of camera positions.

The obtained reduced set is generally not minimal because every position of the initial set is only compared to the already existing camera positions in the reduced set. So, the first positions of the initial set have a higher probability to be retained for the reduced set than the last ones. However, it is possible that some of these positions become redundant after addition of new position in the reduced set.

To face this problem, an additional processing of the elements of the reduced set is required. Every position of the reduced set is compared to all the other positions of the set and, if it doesn't allow to see more details than the other positions of the set, it is suppressed.

4. OFF-LINE PATH PRE-COMPUTING

For a computer games player it is important to well understand the game and its environment. The camera has to move taking into account the quality of view and the player's comfort. This requires a smooth movement of the camera. A camera path with brusque changes of direction may be very perturbing for the game player. What is needed is a set of interesting points of view and a path to connect these points of view. As the game environment is generally not changing, it is possible to pre-compute the camera path in order to be able to use it in real time during the game.

Of course, the camera movement may be influenced by the game action but, in all cases it is important that the camera manages its movement with as main goal to make the player able to well understand the game.

In order to be able to compute an interesting path for the camera it is necessary to first compute a set of interesting points of view. So, the process of off-line camera path pre-

computing may be composed of two steps [Jau04]:

a. Computation of a set of interesting points of view for the scene.
b. Computation of a smooth trajectory allowing the camera to reach these points of view.

4.1 Computing a set of points of view

Techniques presented in section 3 may be used to compute the required set of points of view. This has the advantage to use alredy defined modules allowing other kinds of improvements. These techniques may used to obtain a minimal or a non minimal set of points of view. We think that a minimal set of points of view is better for obtaining a smooth path for the camera.

The obtained viewpoints have to be ordered according to some ordering criterion. Two kinds of ordering have been tested: *Proximity-based* ordering and *minimal path-based* ordering.

4.2 Computing a path for the camera

At the end of the ordering process, applied to the reduced set of viewpoint, it is possible to compute a camera path.

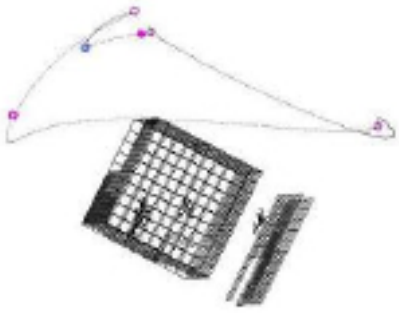

Figure 8 : smooth path for scene exploration

The method has to avoid brusque changes of direction in the camera movement. To do this, the following technique is used.

Let us suppose (figure 9) that the current camera position on the surface of the sphere is A and the viewpoint to reach is B.

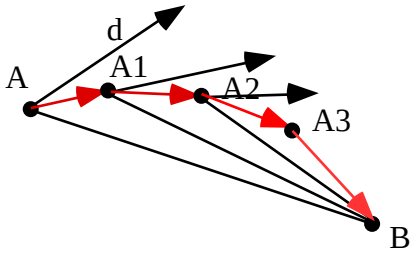

Figure 9 : Computing a smooth camera path

If the current direction of the camera movement at point A is d, a new direction vector AA1 is computed inside the angle (d, AB), where angle (AA1, AB) is equal to the angle (d, AB)/2, and the next position A1 of the camera movement is defined on the vector AA1. The length of AA1 is equal to the current step of the camera movement. The same process is repeated for each new step. If the distance of the current camera position from the position B to reach is less than a threshold value, the position to reach becomes the next camera position. In figure 9, A, A1, A2, A3 and B are successive positions of the camera.

In figure 8 one can see an illustration of smooth camera path, obtained with the above technique. In many cases, minimal path-based ordering allows to obtain better camera paths. On the other hand, this kind of ordering is more time consuming than proximity-based ordering.

5. CONCLUSION

Some intelligent techniques, generally based on visual complexity evaluation, have been presented. The purpose of these techniques is allow implementation of new tools improving already existing ones for computer games.

The proposed techniques improve image-based modelling and rendering as well as simplification of moving objects, in order to get fast rendering. They also improve pre-computing of camera motion, allowing fluid changes of the game environment.

We think that similar techniques, based on heuristic search and visual complexity evaluation, should be applicable to other problems of computer games. One possible application could be automatic estimation of the visual complexity of game environments or objects, in order to allow intelligent fluid selection of level of details (LOD) for the various objects of the game.

Lighting of game environments is another important problem because the player has to well see and understand the game environment. How to choose light source positions [PPV03d] and intensities? Intelligent heuristic search-based techniques, evaluating the visual complexity of the different parts of a game environment should allow to find interesting light source positions. As light source positions may be pre-computed for game environments, the selection process has not to be very fast. We really think that techniques for automatic selection of light source placements could be very promising for computer games.

ACKNOWLEDGMENTS

This project has been supported and financed in part with funds of the European project GameTools. It has also been partly supported and financed by the Limousin Region (France). The authors would like to thank all people and organisations which have supported in any manner this project.

REFERENCES

[And04] Carlos Andújar, Pere Pau Vázquez, Marta Fairén. Way-

Finder: guided tours through complex walthrough models, Computer Graphics Forum (Eurographics 2004), 2004.

[BDP00a] P. Barral, G. Dorme, D. Plemenos. Intelligent scene exploration with a camera. International Conference 3IA'2002, Limoges (France), May 3-4, 2000.

[BDP00b] P. Barral, G. Dorme, D. Plemenos. Scene understanding techniques using a virtual camera. Eurographics 2000, Interlagen (Switzerland), August 20-25, 2000, Short papers proceedings.

[BDP99] P. Barral, G. Dorme, D. Plemenos. Visual understanding of a scene by automatic movement of a camera. International Conference GraphiCon'99, Moscow (Russia), August 26 – September 3, 1999.

[Coh03] D. Cohen-Or, Y. Chrysanthou, C. Silva, F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics* 2003.

[Col88] C. Colin. A System for Exploring the Universe of Polyhedral Shapes. Eurographics'88, Nice (France), September 1988.

[Dur00] F. Durand. A multidisciplinary survey of visibility. ACM Siggraph course notes Visibility, Problems, Techniques, and Applications 2000

[Dur02] F. Durand, G. Drettakis, C. Puech. The 3D visibility complex. ACM Trans. Graph. 2002, 21, 176-206.

[Feixas02] Miquel Feixas, An Information Theory Framework for the Study of the Complexity of Visibility and Radiosity in a Scene. PhD thesis, Technical University of Catalonia, 2002.

[Feixas99] M.Feixas, E.Acebo, Philippe Bekaert and M.Sbert. An information theory framework for the analysis of scene complexity, Eurographics'99.

[Gra02] J. Grasset. Techniques for improving rendering: maps and boxes. PhD thesis, Limoges (France), July 8, 2002 (in French).

[Gra05] J. Grasset, D. Plemenos. Visibility-based Simplification of Objects in 3D Scenes. Proceedings of WSCG 2005 short papers.

[Gum02] S. Gumhold. Maximum entropy light source placement. Visualization 2002 International Conference (October 2002).

[HM03] M. Halle, J. Meng. Lightkit: A lighting system for effective visualization. IEEE Visualization (2003).

[Jau04] B.Jaubert. Off-line automatic exploration of virtual worlds. MSc report (in French), Limoges (France), July 2004.

[JPP02] V. Jolivet, D. Plemenos, P. Poulingeas. Inverse direct lighting with a Monte Carlo method and declarative modelling. Lecture Notes in Computer Science, 2002.

[KK88] T. Kamada, S. Kawai. A Simple Method for Computing General Position in Displaying Three-Dimensional Objects. Computer Vision, Graphics and Image Processing, vol. 41, 1988.

[Lue95] D. Luebke, C. Georges. Portals and mirrors: simple, fast evaluation of potentially visible sets. *Proceedings of the 1995 symposium on Interactive 3D graphics* 1995, 105-ff.

[MAB97] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. International Conference SIGGRAPH 97.

[Nir02] S. Nirenstein, E. Blake, J. Gain. Exact from-region visibility culling. Proceedings of the 13th Eurographics workshop on Rendering 2002, 191-202.

[PB96] D. Plemenos, M. Benayada. Intelligent Display Techniques in Scene Modelling. New Techniques to Automatically Compute Good Views. International Conference GraphiCon'96, St Petersburg (Russia), 1-5 of July 1996.

[PF92] P. Poulin, A. Fournier. Lights from highlights and shadows. Computer Graphics 25, 2, March 1992.

[Ple03] D. Plemenos. Exploring Virtual Worlds: Current Techniques and Future Issues. International Conference GraphiCon'2003, Moscow (Russia), September 5-10, 2003.

[Ple87] D. Plemenos. Selective refinement techniques for realistic rendering of 3D scenes. International Journal of CAD and Computer Graphics, vol. 1, no 4, 1987, in French.

[PPV01] P.P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint Selection Using Viewpoint Entropy. Vision, Modeling, and Visualization 2001 (Stuttgart, Germany), pp. 273-280, 2001.

[PPV02] P.P. Vázquez, M. Feixas, M. Sbert, and A. Llobet. Viewpoint Entropy: A New Tool for Obtaining Good Views for Molecules. VisSym '02 (Eurographics - IEEE TCVG Symposium on Visualization) (Barcelona, Spain), 2002.

[PPV03] Pere Pau Vázquez, PhD thesis. On the Selection of Good Views and its Application to Computer Graphics. Technical University of Catalonia, 2003.

[PPV03b] Pere-Pau Vázquez and Mateu Sbert. Fast adaptive selection of best views. Lecture Notes in Computer Science, 2003 (Proc. of ICCSA'2003).

[PPV03c] Pere-Pau Vázquez and Mateu Sbert. Perception-based illumination information measurement and light source placement. Lecture Notes in Computer Science, 2003 (Proc. of ICCSA'2003).

[PPV03d] P.P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Automatic View Selection Using Viewpoint Entropy and its Application to Image-Based Modeling. Computer Graphics Forum, desember-2003.

[PPV03e] Pere-Pau Vázquez and Mateu Sbert. Automatic indoor scene exploration. In International Conference on Artificial Intelligence and Computer Graphics, 3IA'2003, Limoges, May 2003.

[PRJ97] P. Poulin, K. Ratib, M. Jacques. Sketcing shadows and highlights to position lights. Computer Graphics International 97, June 1997.

[PSF04] D. Plemenos, M. Sbert, M. Feixas. On viewpoint complexity of 3D scenes. STAR report. International Conference GraphiCon'2004, Moscow (Russia). September 5-10, 2004.

[Rigau00] J. Rigau, M. Feixas, and M. Sbert. Information Theory Point Measures in a Scene. IIiA-00-08-RR, Institut d'Informàtica i Aplicacions, Universitat de Girona (Girona, Spain), 2000.

[Rigau02a] J. Rigau, M. Feixas, and M. Sbert. New Contrast Measures for Pixel Supersampling. Advances in Modeling, Animation and Rendering. Proceedings of CGI'02 (Bradford, UK), pp. 439-451, 2002. Springer-Verlag London Limited, London, UK.

[Rigau02b] J. Rigau, M. Feixas, and M. Sbert. Entropy-Based Adaptive Sampling. Graphics Interface 2003 (Halifax, Canada), june-2003.

[Sbert02] M. Sbert, M. Feixas, J. Rigau, F. Castro, and P.P. Vázquez. Applications of Information Theory to Computer

Graphics. Proceedings of 5th International Conference on Computer Graphics and Artificial Intelligence, 3IA'2002 (Limoges, France), pp. 21-36, May 2002.

[Tell91] S. J. Teller, C. H. Séquin. Visibility preprocessing for interactive walkthroughs. Proceedings of the 18th annual conference on Computer graphics and interactive techniques 1991, 61-70.

About the authors

Dimitri Plemenos is a full professor at the University of Limoges (France). His research area is intelligent techniques in computer graphics, including Declarative Modelling, Intelligent Rendering and Intelligent Virtual World Exloration. He is author or co-author of several papers and member of the IPC of many international conferences and journals. Dimitri Plemenos is the organiser and general chair of the 3IA international conference on Computer Graphics and Artificial Intelligence.

Jérôme Grasset is professor in Computer Science at the High School of Computer Science Engineering. of Limoges (France). His research area is Computer Graphics.

Benoît Jaubert is a PhD Computer Science student at the MSI laboratory of the University of Limoges (France). He is working on defining new tool for computer games.

Karim Tamine is an associate professor in Computer Science at the University of Limoges (France). His research areas are Computer Graphics and Network Security.