Techniques for Computing Viewpoint Entropy of a 3D Scene

Pascual Castelló[*], Mateu Sbert[+], Miguel Chover[*], Miquel Feixas[+]

[*]*Departamento de Lenguajes y sistemas Informáticos, Universitat Jaime I, Campus Riu Sec, 12080 Castellón de la Plana, Spain*
[+]*Institut d'Informàtica i Aplicacions, Universitat de Girona, Campus Montilivi, 17071 Girona, Spain*

Abstract

Viewpoint entropy is a metric that allows measuring the visibility goodness of a scene from a camera position. In this work, we have studied different software and hardware assisted techniques to compute the entropy. The main objective of this study is to identify which of these techniques can be used in real time for 3D scenes of high complexity. Our results show that interactivity can be obtained with occlusion query technique and that for real time we need a hybrid software and hardware technique.

Key words: Viewpoint entropy, Graphics hardware, Occlusion query

1 Introduction

Recently, several methods have been developed to compute the goodness of a viewpoint. These methods have in common the use of the viewpoint complexity concept [Barral et al. (1999, 2000), Sbert et al. (2002), Vázquez et al. (2001, 2002, 2003), Rigau et al. (2000)]. The notion of viewpoint complexity is used in several areas of Computer Graphics such as scene understanding and virtual world exploration, radiosity and global illumination, image-based modelling and rendering, etc.

Among the metrics that have been introduced for the complexity calculation, the viewpoint entropy has been the most fruitful metric up to date [Vázquez (2003)]. However, the viewpoint entropy computation can be very expensive, especially when a very complex scene and multiple viewpoints have to be evaluated.

In this paper, we will assess different alternatives for its calculation on several geometric models with increasing complexity. For our calculation, we will make use of the facilities of modern hardware cards such as OpenGL histogram and Occlusion query as well as the new symmetric bus PCI Express [Segal et al. (2004), Wilen et al. (2003)].

This paper is organized as follows. In the next section we will give a viewpoint entropy definition. In Section 3 we will present several techniques for its calculation and in Section 4 we will evaluate those techniques with different models. In section 5 we will show our conclusions and future work.

2 Viewpoint Entropy

The viewpoint entropy, based on Shannon entropy definition, was introduced by Vázquez et al. (2001, 2003) and is a measure of the information provided by a point of view.

The Shannon entropy of discrete random variable X, with values in the set $\{a_1, a_2, ..., a_n\}$, is defined as

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i, \qquad (1)$$

where $p_i=Pr[X=a_i]$, the logarithms are taken in base 2 and 0 log 0 = 0 for continuity. As -log p_i represents the information associated with the result a_i, the entropy gives the average information (or the uncertainty) of a random variable. The unit of information is called a bit.

To define viewpoint entropy we use as probability distribution the relative area of the projected faces (polygons) over a sphere of directions centered in the viewpoint. Thus, given a scene S and a viewpoint p, the entropy of p is defined as

$$I(S, p) = -\sum_{i=0}^{N_f} \frac{A_i}{A_t} \log \frac{A_i}{A_t}, \qquad (2)$$

where N_f is the number of polygons of the scene, A_i is the projected area of polygon i over the sphere, A_0 represents the projected area of background in open scenes, and A_t is the total area of the sphere. In a closed scene, the whole sphere is covered by the projected polygons, and thus $A_0=0$.

The maximum entropy is obtained when a certain point can see all the polygons with the same relative area. So, in an open scene, the maximum entropy is log (N_f + 1) and, in a closed scene it is equal to log N_f. Among all the possible viewpoints, the best is the one that has maximum entropy, i.e. maximum information captured.

3 Techniques for Computing the Entropy

In order to compute the entropy, we need the number of pixels covered for each visible triangle from a particular camera position. This number will give us the projected area. Next we analyze several techniques that allow us to compute those areas.

3.1 OpenGL Histogram

The OpenGL histogram was first used to compute the entropy in Vázquez et al. (2004). The OpenGL histogram let us analyze the colour information of an image. Basically, it counts the appearances of a colour value of a particular component. However, we can also use it to calculate the area of triangles that are visible from a viewpoint, without reading the buffer. Since version 1.2, OpenGL includes an extension called glHistogram. This extension is part of the image processing utilities. The OpenGL histogram is hardware-accelerated, although there are just a few graphics cards that actually support it (for instance, 3DLabs WildCat). Usually, it is a driver manufacturer's responsibility, as generally happens with the OpenGL extensions, and it is often implemented in software.

In order to obtain the area of each visible triangle, by means of the OpenGL histogram, we need to assign a different colour to each triangle. An important limitation is that histograms have a fixed size, normally of 256 different values. This is the most common value in many graphics cards. The glGetHistogram command returns a table that counts each colour value separated into channels. If we use the 4 RGBA colour channels, a 256 item table of 4 integer values will be returned, where each integer is the number of pixels this component has. Thus, if we want to detect a triangle, this should be codified using one single channel. This gives us a total of 1020 different values. That is to say, for channel R (1,0,0,0) up to (255,0,0,0), for channel G (0,1,0,0) up to (0,255,0,0), for channel B((0,0,1,0) up to (0,0,255,0) and finally for channel A (0,0,0,1) up to (0,0,0,255). The value (0,0,0,0) is reserved for the background.

Obviously the main drawback of this technique is that for objects with more than 1020 triangles, several rendering passes are needed. In each pass, we will obtain the area of 1020 different triangles of the object.

Using histograms with a higher number of items, and making a rendering off-screen, will increase the number of colours, and therefore making necessary less rendering passes. However, this possibility is outside the OpenGL specification

and it is hardware dependent. It was not possible for us to use a larger size histogram in the several graphics cards tested.

3.2 Hybrid Software and Hardware Histogram

The OpenGL histogram allows us to obtain the area of each visible triangle. However, as we have seen in previous section, several rendering passes are needed for objects with more than 1020 triangles. Currently, new symmetric buses have appeared such as the PCI Express. In this new bus the buffer read operation is not as expensive as before. Therefore, it is possible to obtain a histogram avoiding making several rendering passes. The way to get it is very simple. A different colour is assigned to each triangle and the whole object is sent for rendering. Next, a buffer read operation is done, and we analyze this buffer pixel by pixel retrieving data about its colour. Using a RGBA colour codification with a byte value for each channel, up to 256*256*256*256 triangles can be calculated with only one single rendering pass.

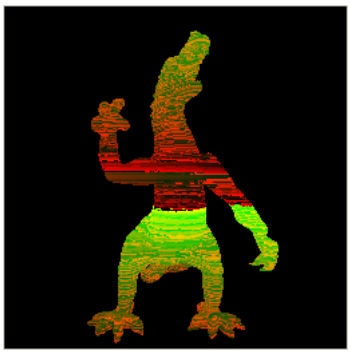

Fig. 1. Entropy (I = 3.140178) for the Dragon model obtained using the Hybrid Software and Hardware Histogram.

3.3 Occlusion Query

This OpenGL extension is normally used to identify which scene objects are hidden by others, and therefore we shouldn't send them to render. In fact, what we do is just to render the bounding box of an object and, if it is not visible, the object is not send for rendering. However, it can also be used to compute the area of the triangles that are visible from a particular camera position.

The OpenGL GL_ARB_occlusion_query extension returns the number of visible pixels. This feature allows us directly obtain the area of the visible triangles. In order to compute the area of each visible triangle from an object using this technique we will proceed as follows. First, the whole object is sent

for rendering and the depth buffer is initialized. Second, we independently send each triangle for rendering. With this procedure it is necessary to make n + 1 rendering passes, being n the number of triangles in an object. We must mention that only in the first pass the whole geometry is rendered. In the following passes, only one triangle is rendered. However, a high number of renderings can significantly penalize this technique. In order to improve the results, this extension can be used asynchronously in contrast to its predecessor HP_OCCLUSION_QUERY. That is to say, it does not use a "stop-and-wait" execution model for multiple queries. This allows us to issue many occlusion queries before asking for the result of any one. But we must be careful using this feature because, as we mentioned above, this extension was not designed to deal with thousands of multiple queries. Thus, we can have some limitations depending on the graphics card.

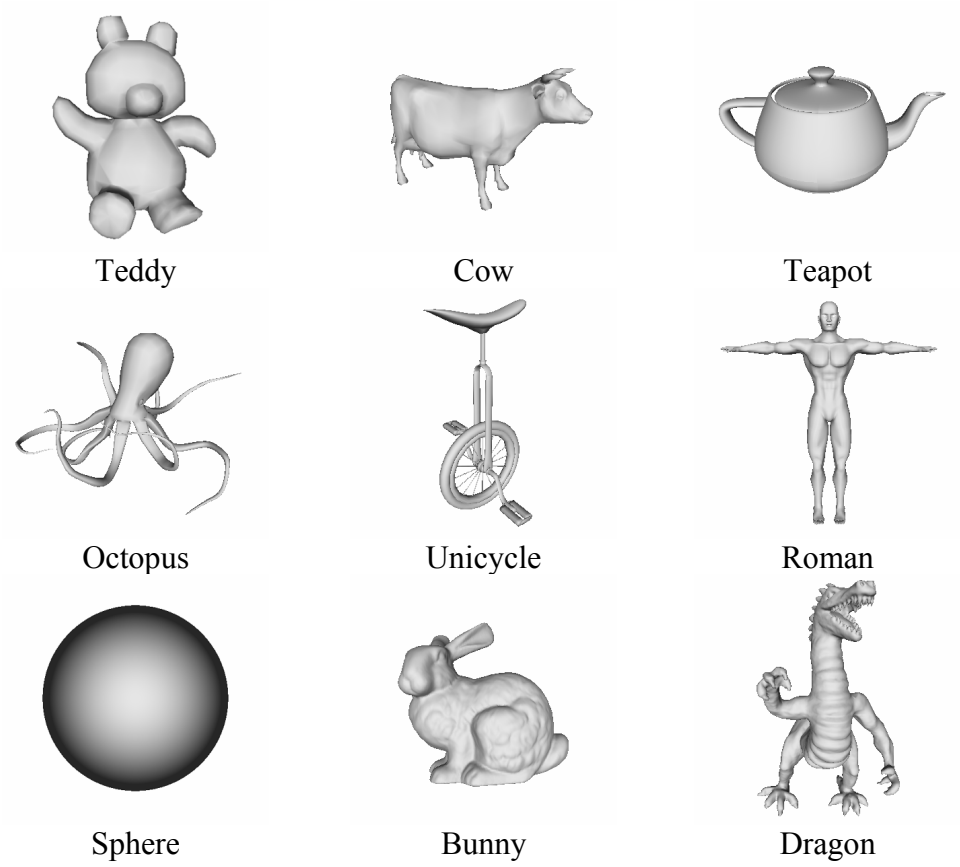

Fig. 2. Models used in our experiments.

4 Results

We have calculated the entropy from 6 camera positions, regularly distributed over a sphere that covers the object, using the different techniques we have described above. In order to compare them, we have measured the time needed to compute the entropy from those cameras. As test models, we have used several objects of different complexities (see Figure 2). These models have been rendered in a 256x256 pixels resolution using OpenGL vertex arrays.

The results have been obtained using a PC Xeon 2,4Ghz 1GB RAM and with 2 different GPUs PCI Express NVIDIA Quadro NVS 280 64 MB and an ATI X800 XT 256 MB. We must emphasize that between the 2 analyzed GPUs, only the NVIDIA card supports the OpenGL histogram.

Model	Vertices	Triangles	Rendering passes	OpenGL Histogram (ms)
Teddy	1598	3192	4	2317.252
Cow	2904	5804	6	3497.404
Teapot	3644	6320	7	4082.005
Octopus	4242	8468	9	5261.136
Unicycle	6973	13810	14	8243.972
Roman	10473	20904	21	12496.437
Sphere	15314	30624	31	18771.404
Bunny	34834	69451	69	44372.069
Dragon	54296	108588	107	72488.374

Table 1. Results obtained for the viewpoint entropy calculation using the OpenGL Histogram.

Model	Vertices	Triangles	Rendering passes	SW+HW Histogram (ms)
Teddy	1598	3192	1	17.893
Cow	2904	5804	1	19.107
Teapot	3644	6320	1	19.374
Octopus	4242	8468	1	20.696
Unicycle	6973	13810	1	23.241
Roman	10473	20904	1	29.965
Sphere	15314	30624	1	36.752
Bunny	34834	69451	1	74.936
Dragon	54296	108588	1	84.438

Table 2. Results obtained for the viewpoint entropy calculation using the Hybrid Software and Hardware Histogram.

Model	Vertices	Triangles	Rendering passes	Occlusion Query (ms)
Teddy	1598	3192	3193	33.122
Cow	2904	5804	5805	52.403
Teapot	3644	6320	6321	59.397
Octopus	4242	8468	8469	78.523
Unicycle	6973	13810	13811	129.849
Roman	10473	20904	20905	190.203
Sphere	15314	30624	30625	284.798
Bunny	34834	69451	69452	657.747
Dragon	54296	108588	108589	1010.673

Table 3. Results obtained for the viewpoint entropy calculation using Occlusion Query.

In Table 1, we can examine the results obtained for the entropy calculation using the OpenGL histogram. These times are too high to allow an interactive calculation, even for objects with a low complexity. This is fundamentally due to the several rendering passes of the whole object that we have made, using objects with several thousand triangles. The main cost component is the OpenGL histogram operation. These results have been obtained using the previously described NVIDIA card.

In Table 2, we show the results for the entropy calculation using the hybrid software and hardware histogram. In this table we can see that the measured times are quite low even if the complexity is increased, mainly because we have made only one rendering pass and the buffer read operation has a very low cost. These results have been obtained by the previously described ATI card.

In Table 3, we can see the results obtained using the Occlusion Query technique. In this table we can clearly observe that the measured times increase proportionally in relation to the complexity of the analyzed model. In the same way as the previous technique, the ratio remains unchanged here because the number of rendering passes is proportional to the number of triangles. A complete rendering of the object is only done at the first pass. The results shown in this table have been obtained by the previously described NVIDIA card.

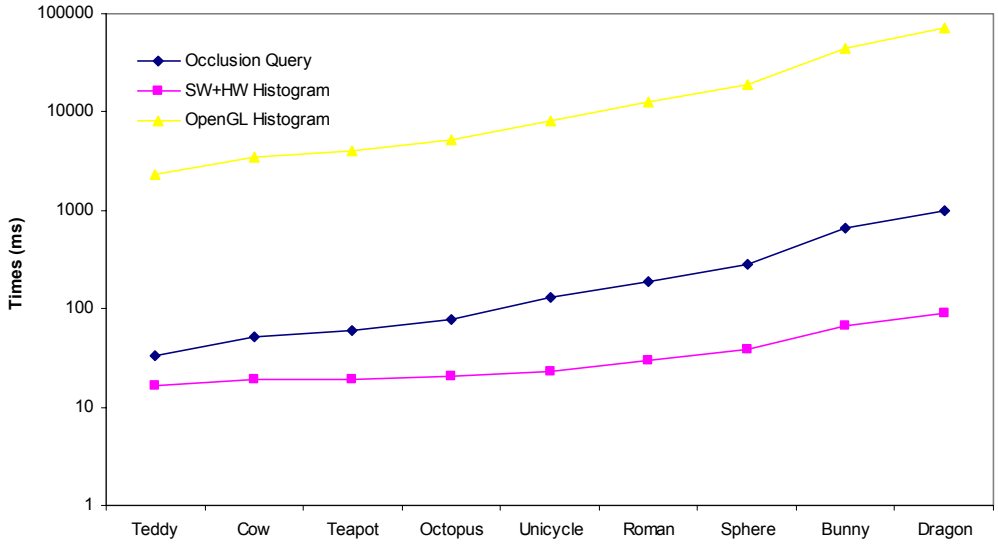

Fig. 3. Comparison of results obtained with the different analyzed techniques.

Finally, in Figure 3 we can see as a summary a performance comparison among the different techniques. The results show clearly that using the hybrid software and hardware histogram we can calculate the entropy in real time and even for complex objects (100 000 triangles), because times increase very slowly as complexity goes up. The next best technique is the Occlusion Query. Note that its cost grows as the object complexity increases, being unapproachable for complex objects for real time. Lastly, the OpenGL histogram technique is surprisingly worst than the two others. This technique is useless for real time, unless we use objects of low complexity (1 000 triangles).

5 Conclusions

The viewpoint entropy is a metric that has been mainly used to determine the best viewpoint of a 3D object. In this work we have studied several hardware assisted techniques to allow computing the viewpoint entropy in an efficient way. Among the different analyzed techniques, the viewpoint entropy calculation by means of hybrid software and hardware histogram has the best performance, followed by the use of occlusion query based technique. By using the hybrid software and hardware histogram technique we can practically achieve the entropy calculation in real time even for complex objects, while using occlusion query technique allows us to obtain only interactivity.

References

Carlos Andújar, Pere Pau Vázquez, Marta Fairén, Way-Finder: guided tours through complex walthrough models, Computer Graphics Forum (Eurographics 2004), 2004.

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

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.

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

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.

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.

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

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.

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

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.

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'02 (Limoges, France), pp. 21-36, 2002.

P. P. Vázquez, M. Sbert. On the fly best view detection using graphics hardware 4th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2004).

M. Segal, Kurt Akeley. The OpenGL Graphics System: A Specification (Version 2.0 – October 22, 2004). Silicon Graphics, Inc., 2004.

A.Wilen, J.Schade, R.Thornburg. Introduction to PCI Express. A Hardware and Software Developer's Guide. Intel Press, 2003.