A Comparative Study of Acceleration Techniques for Geometric Visualization

Pascual Castelló, José Francisco Ramos, and Miguel Chover

Department of Computer Languages and Systems, Universitat Jaume I,
Campus Riu Sec, 12080 Castellón de la Plana, Spain
{castellp, jromero, chover}@uji.es
http://graficos.uji.es

Abstract. Nowadays computer graphics hardware presents a series of characteristics, such as AGP memory, vertex cache, etc., that can be used for real-time rendering. The aim of this paper is to conduct a comparative study of different techniques that are shown in the OpenGL graphics standard together with hardware features that enable the visualization of the geometry of complex objects to be accelerated. These techniques are applied to the multiresolution modeling, which requires techniques that can be implemented with dynamic geometry.

1 Introduction

Computer graphics systems present a pipeline architecture [12] in which 3D data about the scene that is to be displayed cross at different stages. Depending on the type of application or the nature of the data, it is possible that some of these stages become a bottleneck, thus drastically reducing the overall performance of graphics devices. It is therefore essential to locate these bottlenecks so that the graphics hardware can operate properly.

Figure 1 shows, in diagram form, the different stages of a graphics pipeline architecture that OpenGL presents [13] [7].

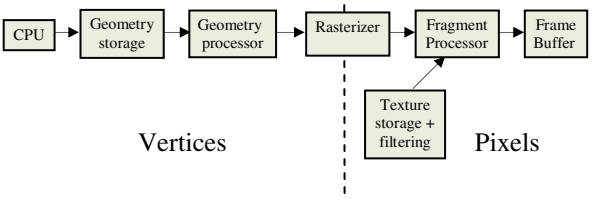

Fig. 1. Simplified OpenGL graphics pipeline

In this architecture, the main bottlenecks that can be distinguished are as follows:

- CPU. There are problems with the application or the driver.
- AGP/PCI Express. There are too many data for the bus.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 240 – 247, 2005.

- Geometry. There are too many vertices or too much computation per vertex.
- Rasterization. There are also too many large-sized triangles.
- Texture. Textures are too big, texture cache is underutilized and the filters are expensive.
- Fragment Shader. There are too many fragments or too much computation per fragment.
- Frame buffer. There is too much read/write to the frame buffer.

The first step is to find the bottleneck and later eliminate it, if possible. If this cannot be done, we then have to try to balance the pipeline.

Traditionally the problem of representing large objects is solved by using multiresolution models [4] [11]. In the multiresolution modeling an object is represented by different approaches, each from a different level of detail (LOD), which is going to necessarily generate bottlenecks in the processing of vertices. There are two types of multiresolution models: continuous, in which geometry changes in runtime as a result of the variation of the level of detail, and discrete, where a few levels are precalculated and one level or another are shown according to the visualization requirements of the scene. The first one requires techniques for dynamic geometry and the second can benefit from techniques for the visualization of static geometry.

The specific problems within geometry are usually the following:

- Wrong transfer due to the use of non-native data types, inadequate sizes or calls to the graphics libraries using less efficient primitives.
- Geometry transformation that gives rise to bottlenecks due to the number of vertices and to the calculations based on them, or because of an inefficient use of the vertex cache.

Within this framework we intend to perform an in-depth analysis of the problems related to geometry, the solutions currently available and their possible application to dynamic geometry.

In section 2, several solutions offered by OpenGL are presented, as well as general solutions related to the drawing primitive, such as the use of extensions that make use of video memory and the vertex cache included in modern graphics processors (GPU). Section 3 describes the solutions applied to several test cases in this paper, and in section 4 the results that were obtained are reported. Finally, conclusions and future work are presented in section 5.

2 Solutions Overview

Different solutions are available with which to resolve the bottleneck problems in the geometry subsystem. They can be classified as specific solutions, which are directly linked to the graphics library that is used, and general solutions, which are more related to the type of primitive employed in the visualization of 3D objects or to the particular characteristics of modern GPUs.

2.1 Specific Solutions

Among the specific solutions, the following can be considered in OpenGL [7]:

- **Immediate mode.** This consists in drawing geometry by means of glBegin()...glEnd() operations. This is the primitive with lowest performance and the least recommended, as is shown below.
- **Display list.** A display list is simply a command group and OpenGL arguments that have been saved in a suitable fashion for later execution by the hardware. Its fundamental characteristic is that it cannot be modified. Obviously the main limitation of this solution is that in principle it is not adapted to dynamic geometry because any change in geometry implies the creation of a new display list. However, it is very well adapted to the representation of static geometry or discrete multiresolution models.
- **Vertex array.** This consists of storing the associated vertex data in special arrays, so that these arrays can be used to specify several geometries with the execution of only one command. The array of vertices can only contain the vertices of the geometry and a list of indexes is needed to generate the primitives based on that array. This scheme is suitable to exploit the characteristics of dynamic geometry because the array of vertices is generated only once and it is then possible to modify the list of indices at runtime.
- **OpenGL extensions.** Extensions enable new characteristics and capabilities to be added to OpenGL. In general, what the extensions related to geometry acceleration attempt to do is to load the data concerning the vertices into the local memory of the graphics card and try to avoid using the main memory.

  Of the extensions that have been developed by manufacturers, the following are perhaps the most noteworthy:

  - ATI_vertex_array_object (8/2002)
  - NV_vertex_array_range (9/2001)

  An extension that has recently appeared with the same intention as the previous ones and which has been officially approved by OpenGL is ARB_vertex_buffer_object (2/2003). This is the extension that was used in this study; its use allowed the behavior of OpenGL vertex arrays to be modified by letting them reside in the local memory of the graphics card.

2.2 General Solutions

Graphics processors have hardware characteristics that allow them to represent 3D objects suitably by means of triangles.

In order to avoid sending repeated vertices to the graphics system other structures such as triangle strips or triangle fans can also be employed. In a triangle strip, the first triangle is drawn and then only one vertex is added for each new triangle that is added to the strip. In a triangle fan, all the triangles share one vertex they have in

common. Consequently, the visualization of an object by means of triangle strips or fans achieves greater acceleration because less information is sent to the graphics system.

In this paper we have analyzed the following algorithms that allow us to obtain a strip-based representation:

- Stripe [3] (Stony Brook University)
- NvTriStrip [8] (NVIDIA Corporation)

Both Stripe and NvTripStrip allow strips to be generated, but only NvTriStrip takes advantage of the vertex cache. Current GPUs have a series of registers that allow us to store the last 16 or 24 vertices used. It is possible to try to provide the graphics system with information about vertices ordered in such a way as to produce the smallest number of page faults. Several studies have been carried out that try to take advantage of this characteristic. These include Chow [2], who presents a method for the compression of geometry; Hoppe [6], who develops an algorithm for the generation of strips while taking advantage of the vertex cache in a transparent way; and Bogomjakov and Gotsman [1], who present a method for the optimization of the vertex cache applied to the Progressive Meshes [5] multiresolution model.

The NvTriStrip library [8] allows both triangles and triangle strips to be generated, taking advantage of the GPU vertex cache and, in addition, it allows for the generation of one single triangle strip (by using degenerated triangles) or several triangle strips.

The Stripe utility [3] allows triangle strips to be generated from objects represented by means of the obj format. This utility always generates several strips, so it does not make use of degenerated triangles.

3 Description of the Comparison

The objective of this study was to measure the performance of all the techniques described above for 3 test meshes: cow (5804 triangles), sphere (30 624 triangles) and bunny (69 451 triangles). These measurements consisted in obtaining the frames per second (fps) rates, that is to say, the number of times per second the system is able to draw the scene.

These techniques were applied to models represented as triangle lists and triangle strips, and the extension that refers to local memory buffers and GPU vertex cache was also tested.

First, we assessed the use of techniques classified as specific solutions:

- Immediate mode
- Display list
- Vertex array
- Vertex buffer object

These techniques were applied to the general solutions. The following aspects were compared within the triangles model:

- Original model triangles (respecting the order in which they appear in the obj file)
- Optimized triangles obtained by NvTriStrip
- One strip with degenerated triangles obtained by NvTriStrip
- Several strips obtained by NvTriStrip
- Several strips obtained by Stripe

Finally, the effect exerted by the vertex cache was measured. It was tested by using NvTriStrip with remapping (vertices are rearranged in the same way as indexes). In particular, times with and without cache were measured using different specific solutions:

- NvTriStrip optimized triangles
- NvTriStrip strips (one strip with degenerated triangles and several strips)

Lastly, the results obtained using the different techniques were compared.

4 Results

The experiments were conducted using different PCs: Pentium IV 2600 MHz, Pentium Xeon 2800MHz and dual Pentium III with 1 Gb RAM and several GPUs: NVIDIA GeForce FX5900[1] 128 Mb, GeForce FX5700[2] 256 Mb and GeForce 6600[1] 256 Mb with two operating systems: Windows XP Professional SP1 and Linux SuSE 9.1. They involved measurement of the number of frames per second (fps) that the system reaches using each of the techniques. The implementation was performed in C++ using the OpenGL graphics library. Ms Visual C++ 6.0 SP5 was employed as a compiler for windows and gcc 3.3.3 for Linux.

The results obtained for the 3 test meshes are shown in Tables 1 to 3.

Table 1. Results (fps) obtained for Cow

Cow	GPU	V. cache	t / s	Kb	IM	DL	VA	VBO
Triangles: original model	*FX5700*	no	5804	136	397.60	1315.68	707.68	1050.95
	Ge6600				850.00	1319.00	1259,70	1268.00
	FX5900				816.19	1951.05	1421.58	1913.09
Triangles: NvTriStrip	*FX5700*	yes	5804	136	371.63	1288.71	384.62	1031.97
	Ge6600				796,20	1488,50	1420,60	1430,60
	FX5900				774.23	1958.04	1385.61	1888.11
One strip: NvTriStrip	*FX5700*	yes	1	102	543.90	1408.59	831.17	1239.76
	Ge6600				1084,90	1494,50	1436,60	1434,60
	FX5900				1089.91	2154.88	1860.20	2081.92
Several strips: NvTriStrip	*FX5700*	yes	551	98	427.72	1368.63	701.30	1105.89
	Ge6600				1015.00	1430,60	1441,60	1427,60
	FX5900				981.02	2010.99	1550.45	1674.33
Several strips: STRIPE	*FX5700*	no	101	98	622.38	1592.41	833.17	1259.74
	Ge6600				1009.00	1520,50	1440,60	1450,60
	FX5900				966.03	2350.65	1995.00	2283.72

[1] Tested under Windows.
[2] Tested under Linux.

Table 2. Results (fps) obtained for Sphere

Sphere	GPU	V. cache	t / s	Kb	IM	DL	VA	VBO
Triangles: original model	*FX5700*	no	30 624	717	86.14	584.83	119.64	328.02
	Ge6600				216,10	834,33	758,20	828,17
	FX5900				205.38	1121.88	397.60	1143.86
Triangles: NvTriStrip	*FX5700*	yes	30 624	717	79.60	748.50	118.53	296.70
	Ge6600				212,40	861,14	810,20	849,15
	FX5900				204.80	1239.76	394.61	1197.80
One strip: NvTriStrip	*FX5700*	yes	1	512	132.60	830.17	178.64	590.82
	Ge6600				394,20	860,14	812,20	847,15
	FX5900				384.23	1287.61	772.23	1221.78
Several strips: NvTriStrip	*FX5700*	yes	1763	496	111.00	780.22	176.47	520.96
	Ge6600				324,40	773,23	721,20	764,24
	FX5900				335.33	1271.73	645.35	986.01
Several strips: STRIPE	*FX5700*	no	172	481	165.41	785.21	213.14	601.40
	Ge6600				336,30	831,17	863,20	765,23
	FX5900				296.11	1149.85	870.13	1049.95

Table 3. Results (fps) obtained for Bunny

Bunny	GPU	V. cache	t / s	Kb	IM	DL	VA	VBO
Triangles: original model	*FX5700*	no	69 451	1630	31.40	218.13	29.01	113.10
	Ge6600				94,53	256,50	112,10	172,70
	FX5900				75.40	345.96	187.62	247.75
Triangles: NvTriStrip	*FX5700*	yes	69 451	1630	36.31	420.00	32.35	162.67
	Ge6600				99,50	512.00	337,70	506.00
	FX5900				93.91	763.24	185.25	741.26
One strip: NvTriStrip	*FX5700*	yes	1	1218	56.10	527.42	55.56	256.23
	Ge6600				171,50	501,50	366,90	495,50
	FX5900				160.20	778.22	360.64	758.48
Several strips: NvTriStrip	*FX5700*	yes	6194	1168	44.12	485.03	69.86	217.13
	Ge6600				121.00	423,70	274,50	455,10
	FX5900				130.61	768.23	270.73	438.12
Several strips: STRIPE	*FX5700*	no	917	1176	58.88	562.87	73.05	286.71
	Ge6600				119,80	499.00	401,80	431,70
	FX5900				105.89	715.57	335.33	614.39

Table 4. Techniques for geometry acceleration in LodStrips, results mesured in milliseconds

Object	Test	Vertices	Strips	Mb	IM	VA	VBO
Cow	*Linear*	2 904	136	0.183	3.74	2.56	1.01
	Exponential				4.60	1.29	1.06
Bunny	*Linear*	34 834	1 229	2.111	19.06	5.85	5.19
	Exponential				24.74	7.47	6.65
Horse	*Linear*	58 485	1 964	3.098	26.38	8.54	7.57
	Exponential				35.51	13.17	9.85

Finally, we have also experimented with the use of acceleration techniques in a continuous multiresolution model, based on strips, called LodStrips [9]. We have applied the linear test and the exponential test [10] for the following test meshes:

Cow, Bunny and Horse. We have considered the total times (extraction + rendering) in milliseconds to measure the performance.

As shown in Table 4, if we consider the object of highest complexity (Horse), it can be seen that the vertex array technique improves performance by up to 300% with regard to the immediate mode. The use of VBO provides an increase in performance about 20% with regard to the vertex array technique.

5 Conclusions and Future Work

Of the different visualization techniques used by OpenGL, the worst technique is the immediate mode. Vertex array leads to a gain of at least 300%, although the technique that achieves the best performance is the display list, which offers a gain of 800% with regard to the immediate mode. The use of VBO remarkably improves the performance of the vertex array by up to 700% with regard to the immediate mode, although it is lower than display lists. The use of the vertex cache significantly improves VBO performance in complex 3D models, and almost reaches the level of display lists. These proportions remain the same regardless of the drawing primitive used, that is, whether they are strips or triangles.

In relation to the drawing primitive, the use of strips instead of triangles improves performance by at least 100% if optimized triangles are used. If we employ original model triangles, however, this can reach 600% when the drawing primitive is a display list. On the other hand, the memory requirements of strips are also about 30% lower than in the case of triangles.

Finally, on comparing NvTriStrip strip generation with that of Stripe, it can be observed that the benefits deriving from the best NvTriStrip case (one single strip with degenerated triangles) and Stripe are similar, although NvTriStrip is significantly improved in VBO. Stripe generates fewer strips than NvTriStrip. For this reason, it achieves higher performance than NvTriStrip with several strips.

Therefore, it can be concluded that using strips as a drawing primitive and display lists for static geometry is the best way to obtain the highest performance. In a case involving dynamic geometry, VBO may be employed since its use in conjunction with the vertex cache offers an increase in performance of 775%, compared with the immediate mode, and this is close to the value achieved by display lists (an increase of 800% with regard to the immediate mode).

Lastly, it can also be concluded that the application of the described techniques to multiresolution modeling improves performance. However, these techniques impose

Table 5. Summary of performance in relation to the immediate mode

Primitive	DL	VA	VBO	VBO with NvTriStrip
Original triangles	300%	150%	225%	-
Optimized triangles	775%	200%	650%	700%
Triangle strips	800%	300%	700%	775%

special requirements when working with data structures. This is notable at the time of defining a new multiresolution model, and we must consider these restrictions if we want to reach optimal performance. LodStrips is a multiresolution model in which these techniques are applied easily.

An extension to this paper would be the application of the techniques described above to different multiresolution models in order to increase their performance.

Acknowledgments

We are sincerely grateful to Cem Cebenoyan of NVIDIA Corporation for his helpful feedback concerning the NvTriStrip library.

This work was partly financed by projects GameTools IST-2-00463 from European Comission, Mater TIN2004-07451-C03-03 from the Spanish Ministry of Science and Technology, VisualCAD FIT-3501101-2004-15 from the Spanish Ministry of Industry, Tourism and Trade, and Juegos P11B2004-22 (Fundació Caixa Castelló - Bancaixa).

References

1. A Bogomjakov, C Gotsman. “Universal Rendering Sequences for Transparent Vertex Caching of Progresive Meshes”, Computer Graphics Forum, (21:2) (2002) pp. 137-148.
2. M M Chow. “Optimized Geometry Compression for Real-time Rendering”, Proceedings of the IEEE Visualization ’97 (1997), pp. 347–354.4. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996).
3. F Evans, S Skiena and A Varshney. Optimising Triangle Strips for Fast Rendering, IEEE Visualization ’96, pp. 319-326, 1996. http://www.cs.sunysb.edu/~stripe
4. M Garland. Multiresolution modeling: survey & future opportunities. State o the Art Reports of EURO-GRAPHICS ’99, 1999.
5. H Hoppe. “Progresive Meshes”, Proc. of SIGGRAPH ’96 (1996) 99-108.
6. H Hoppe. “Optimization of Mesh Locality for Transparent Vertex Caching”, *ACM SIGGRAPH 1999*, pp. 269-276.
7. R Kempf and C Frazier. OpenGL reference manual. Reading, Addison-Wesley Developers Press, 1997.
8. NvTriStrip Library, NVIDIA Corporation (2002). Available on the Internet at the following URL : http://developer.nvidia.com/object/nvtristrip_library.html
9. F Ramos, M Chover, LodStrips, Lecture notes in Computer Science, Proc. of Computational Science ICCS 2004, Springer, ISBN/ISSN 3-540-22129-8, Krakow (Poland), vol. 3039, pp. 107-114, June, 2004.
10. J Ribelles, M Chover, A López and J Huerta, A First Step to Evaluate and Compare Multiresolution Models, Short Papers and Demos of EUROGRAPHICS’99 (1999) 230-232.
11. J Ribelles, A López, O Belmonte, I Remolar, M Chover. Multiresolution modeling of arbitrary polygonal surfaces: a characterization. Computers & Graphics. Elsevier Science. 2002.
12. J Spitzer, “OpenGL Performance Tuning”, NVIDIA Corporation, GameDevelopers Conference 2003.
13. M Woo, J Neider, T Davis. OpenGL programming guide. Reading, MA: Addison-Wesley Developers Press, 1997.