Technical Considerations on the Visualisation of Virtual Environments

F. Tecchia, C.A. Avizzano, A. Brogni, C. Evangelista, G. Di Pietro, M. Bergamasco

PERCRO, Scuola Superiore S.Anna
Via Carducci, 40
Pisa, 56127 (PI), ITALY

Abstract

The problem of dealing with the graphical representation of 3D environments by means of computer systems is introduced. The hardware and software tools that allow the 3D visualisation of virtual models by starting from an abstract representation of the Virtual Environment are presented. The objectives for the design of a software module allowing high performance interactive visualisation of complex Virtual Environments are given.

Keywords:

1 Introduction

A virtual environment can be considered as a multisensory immersive experience. The basic idea is that of designing a scenario that it does not exist in reality. The person who lives through a virtual environment should perceive sensorial stimuli in such a way as not to distinguish between the real and the virtual environment in which he/she is immersed. In order to achieve such a goal, the virtual environment system must be designed by presenting a human/machine interface possessing efferent and afferent filters appropriately conceived and realized.

As it happens for a real experience, one of the most important components of the afferent filter is the visual subsystem which goal is that of generating a 3D graphical representation in real time.

An adequate performance of the visual subsystem can be achieved by exploiting graphical workstations capable to perform in real time all the required operations needed to generate the correct response to the subject's action without any perceivable delays (general rules set a rate of at least 30 frames per second for the 3D graphical representation). Another feature to be integrated in the design of the visual subsystem is the capability of stereoscopic representation in order to generate depth perception of the virtual objects belonging to the simulated environment.

By taking into account the determinants specifying the conditions of sense of presence in a virtual environment [1], it can be seen how the development of high definition graphical rendering modules working in real time represents a mandatory line of research in order to achieve adequate levels of extent of information exchange in the afferent filter.

This paper describes the baseline motivations of the work carried out at PERCRO finalised to 3D visualisation of Virtual Environments (VE) and to the graphical control of its contents.

The complexity of the geometrical models utilised for VE applications is often very high and, consequently, a real time visualisation in which all the descriptive data of the environment are considered for the rendering process is prevented. To obtain this, an advanced management of the geometrical data base is required. The geometrical data base must be arranged in such a way that, in every step of the rendering process, it is possible to efficiently discern the information actually involved in the visualisation from those, at least temporarily, not needed.

Moreover, the realistic visualisation of specific phenomena that occur in the virtual scenario, e.g. the interactive deformation of objects, requires ad hoc methodologies and algorithms. These specific algorithms, by focusing on the performances of the visualisation system, succeed to optimise their management and to achieve the best results under the very strong temporal constraints of the application.

The advanced management of the geometrical information is then a very complex activity that cannot be taken in charge of the specific applications. The optimisation of the rendering process must be directly implemented inside the visualisation module that must

separate the application from the necessary computing activities. The software module developed for 3D visualisation performs such an organisation and management process of the geometrical data base by integrating inside all the techniques presently available in the field of the optimisation of real time 3D rendering.

The visualisation of 3D environments performed by an application is a process articulated in different phases. The scenario to be traced on the video is represented by storing different types of data specifying the geometrical aspects of the environment and the visual characteristics of its materials, such as colour and opac-

ity. The set of these data is called graphical data base or, due to the specific geometrical description utilised, polygonal database. The process of visualisation is the set of activities that, by starting from the above logic representation, is capable of visualising on a screen a graphical representation of the environment.

The visualisation process can be schematically divided in the following phases:

- The application retrieves, from the polygonal database, data on the objects to be visualised and requires their visualisation by means of suitable library functions.
- The information given by the application undergoes a series of hardware and software processing. The complex of the processing activity is restricted in a single logical entity called graphical pipeline or rendering_pipeline. Along the path inside this logical entity data are converted into an image which is stored in a dedicated memory area, the framebuffer.
- The stored raster image is visualised on the video.

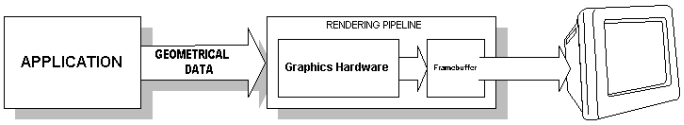

Figure 1: Visualisation process

2 Visualization Hardware

The simplicity of the processing devoted to build a raster image and also the high degree of parallelism by which the input geometrical data can be handled allow to perform at least a part of the activities internal to the pipeline by means of a dedicated hardware. At present, the large part of graphical workstations is capable to process with extreme efficiency very simple geometrical entities, such as polygons with three edges, lines as well as points. These are, in real time graphics, the more suitable geometrical primitives to be used for the representation of the environment to be visualised.

The above indicated dedicated systems utilise the illumination model developed by Phong [2] to simulate the presence of light sources in the scene and the correspondent shading effects. In the approximated Phong model the reflection of light radiation is modelled by three components: diffused, specular and ambient, each one separately modelled and defined for both light sources and materials. For each geometrical primitive a material must be specified; the parameters of the material can be uniformly applied to the primitive surface or varied by means of a texture specified through a bitmap 2D image.

In terms of design, the use of such elementary primitives allows to optimise the management of the processing resources by dedicating them to a restricted set of operations. To this aim, every geometrical entity sent to the rendering pipeline is preliminary decomposed in triangles, segments and points. In order to avoid the computational overhead of this operation, it is preferable to store all the objects directly in the form utilised inside the rendering pipeline; each group of primitives representing a part of the scenario is called polygonal mesh.

To allow the visualisation of the environment, the rendering pipeline receives as input the information on the graphical primitives setting up the scenario. For each of them the following parameters must be specified:

- Spatial position of the vertices with respect to an absolute frame of reference (world coordinates);
- Characteristics of the materials utilised in the scenario;
- Orientations of the polygons in space;
- Information on the textures, if present.

At the same time, also the point of view from which the scenario must be seen, the sight direction (position and orientation of the camera), the position of light sources inside the scenario as well as a value specifying the span of the field of view must be given. The activities performed inside the rendering pipeline can be divided in three different steps, as depicted in 2.

The first step, called geometry_engine, performs all the geometrical transformations needed for positioning the polygons in the scenario and the calculations related to light effects. By starting from the input data, the following operations are performed:

- The primitives coordinates are transformed in the camera frame of reference (eye_coordinate);
- The different parameters related to the illumination of the primitive by the light sources (lighting process);
- The primitives which are invisible with respect to the sight direction and field of view span are removed from the pipeline (view_frustum_culling process);
- The operation of perspective correction is applied and the primitives coordinates are transformed in 2D by projecting them on the sight plane (the monitor screen).

In the second step of the rendering pipeline activities (triangle_setup) all the data necessary for rasterisation are prepared. Calculations of the slopes and shading

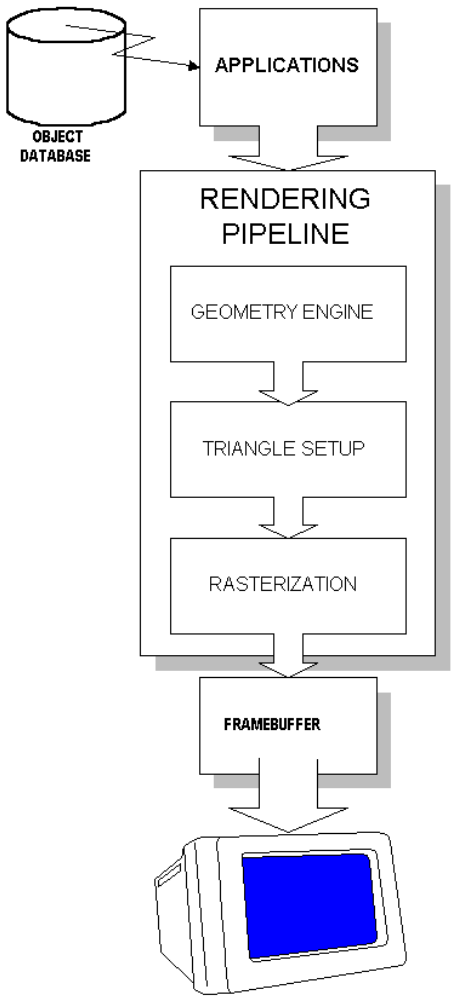

Figure 2: Activities of the rendering pipeline

parameters of the lines and edges of the triangles are performed.

In the third step all the pixels internal to the triangles (rasterisation) are traced. To each pixel are assigned:

- A colour, calculated by considering the illumination conditions of the triangle, the associated material, the texture (if specified) and other possible features (e.g. transparency);
- A depth value, intended as the distance from the camera position, that will be utilised later in the z-buffer algorithm for removing, via hardware, the hidden surfaces.

All the described activities need the calculation of mathematical functions and , consequently, an adequate computing power that limits the amount of data that the rendering pipeline is able to process in the unit time. Usually the performances of the rendering pipeline are measured by the number of triangles processed per second: since the three steps are strictly sequential, the saturation of either of the three implies a bottleneck for the whole system and, consequently, limits the achievable performances. The causes of saturation are:

- Limited computing power for the calculations performed in either the geometry_engine, triagle_setup and rasterisation;
- Limited bandwidth for data transfer from the rasterisation step to the framebuffer.

Which of the above is the cause for the limit of performances it depends on the type of required visualisation. The following classification applies:

- When the cause of saturation is due to the amount of geometrical calculations required in the different steps (e.g. for polygonal mesh with a large number of polygons) the application is called geometry_limited;
- When it is the number of pixel to be traced that limits the computing speed (it often happens when the video resolution is high), the application is called fill-rate_limited.

2.1 Workstation Performances for 3D Graphics

The performances of a graphical workstation are difficult to be defined since they are influenced by different factors, such as the dimensions of the primitives to be visualised, the number and type of the light sources, the sizes of the textures to be applied to the triangles, etc. Moreover, the evaluation should consider also the quality of the produced image but for this it is difficult to define an objective parameter. It is possible to use as evaluation criteria the number of triangles to be

rendered per second, the speed for tracing the pixels belonging to the different triangles (fill_rate) and the amount of memory dedicated to store the textures (texture_memory).

The capability of each system for graphical processing largely depends on the strategy by which the rendering pipeline has been realised. Two solutions have been adopted (see 3):

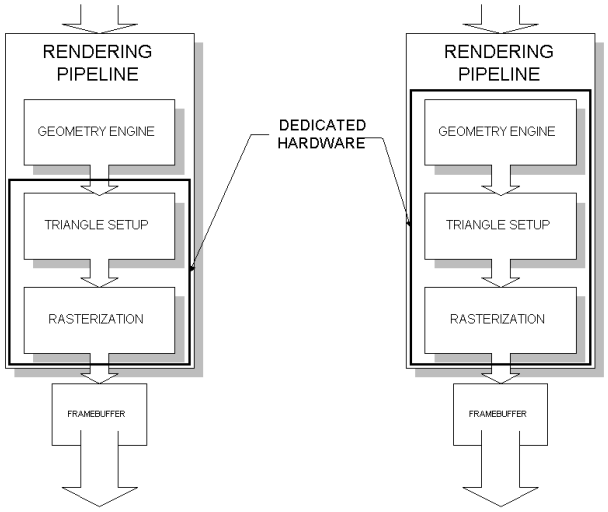

Figure 3: Two different strategies for the rendering pipeline

- In the first solution only the geometry_setup and rasterisation steps are implemented in hardware. The task of executing all the operations required in the geometry_engine are left to the processor(s). This approach is finalised to cost reduction on which the impact for hardware implementation of the GE is evident;
- In the second solution all the three steps of the rendering pipeline are realised by hardware. In this solution the efficiency of the system is maximised but the costs are also increased.

At present, the range of performances for commercially available workstations is very wide. The lower limit is represented by the platforms Intel PC equipped with low cost 3D graphical boards, in which the first strategy is implemented with a computing power of one million of triangles per second. On the other hand, the highest performances are obtained with Unix workstations by Silicon Graphics that, by utilising massively parallel architectures for the realisation of each step of the pipeline, are capable of generating more than 80 millions of triangles per second.

Notwithstanding the high gap of performances between the two architectures, at present the hardware dedicated to 3D graphics on Intel platforms is subjected to a very rapid evolution (two times the computing power per year). Progresses can be registered also in terms of quality of the image: in a very short time the 15 bit per pixel representation moved to a 24 or 32 bit colour depth. Moreover the utilisation of antialiasing techniques for the generated image are always more frequently used. All these features have been matched with an increase of the video memory (for both the framebuffer and texture memory): from the initial 512k, at present the average value is of 16 Mb with an availability on the market of low-medium cost boards equipped with 100Mb of memory. An example of the technical innovations devoted to the management of the large bulk of data required by 3D graphical applications has been the Intel AGP (Accelerated Graphics Port) bus that offers an higher bandwidth with respect to the previous PCI bus and the implementation of particular functionalities, such as the Direct Memory Execution for the data exchange between the processor, the memory and the graphical hardware.

By taking into account the evolution of low cost platforms, the development of future visualisation modules for VE applications must be design by taking into account the characteristic of independence from the utilised platform. This objective influences also software development choices. To this aim and, at the same time, to exploit the advantages of the object oriented programming, the implementation language can rely on C++ while the management of the graphical hardware can be performed through the use of the standard library OpenGL.

3 The Graphics Library OpenGL

In order to allow a generic application to access the rendering pipeline, the availability of a dedicated library (API, Application Programming Interface) is required in order to manage the underlying hardware by releasing the application from a direct communication with the driver.

Several libraries are available to the above scope. Some of them are specific for a given hardware; other are specific for a given operating system (e.g. the DirectX library developed by Microsoft).

In the framework of scientific visualisation the most diffused API is OpenGL, introduced in the market by Silicon Garphics as an evolution of the previous GL (Graphics Library). This library has been designed to be independent from the platform. At present it exists for almost every operating system. Its high spread has also conditioned the hardware production, where the trend is now that of directly integrating "on chip" all the functionalities of OpenGL.

OpenGL renders available a series of functions exclusively dedicated to the treatment of 3D graphics. It allows to trace on video either simple primitives, such as points, lnes and triangles, and either convex polygons

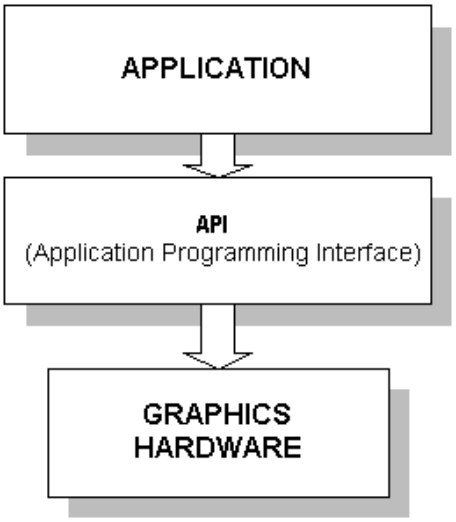

Figure 4: Access to the graphical hardware

composed of an arbitrary number of edges.

Moreover it allows the visualisation of analytical curves and of complex surfaces (NURBS, Non Uniform Rational B-Spline). To treat these type of entities it is the same library that decompose them in segments and polygons.

The application must specify all the geometrical primitives and related materials; after that the OpenGL library is able to visualise the scenario by utilising an orthographic projection or a perspective projection (see 5).

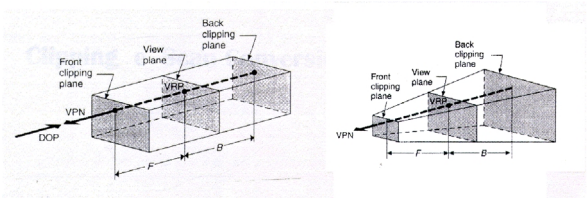

Figure 5: Sight volumes for an orthogonal projection and perspective projection.

OpenGL assumes as visible those primitives that, at least partially, are included inside the sight volume (view_frustum); in the case of perpspective projection this volume is defined by the observer's position in the scene, by the sight direction, by the span of the field of view and by the position of the two planes (front_clipping_plane and back_clipping_plane) delimitating the allowed distance for the primitives to be visualised. The entities outside this volume are not visible and for this reason are discharged from the rendering pipeline immediately after the geometry_engine step in order to avoid unnecessary processing by the following steps (View_frustum_culling process).

The geometrical transformations needed for the visualisation are specified to OpenGl by the application by means of three matrices model_matrix, view_matrix ad perspective_matrix (see 6).

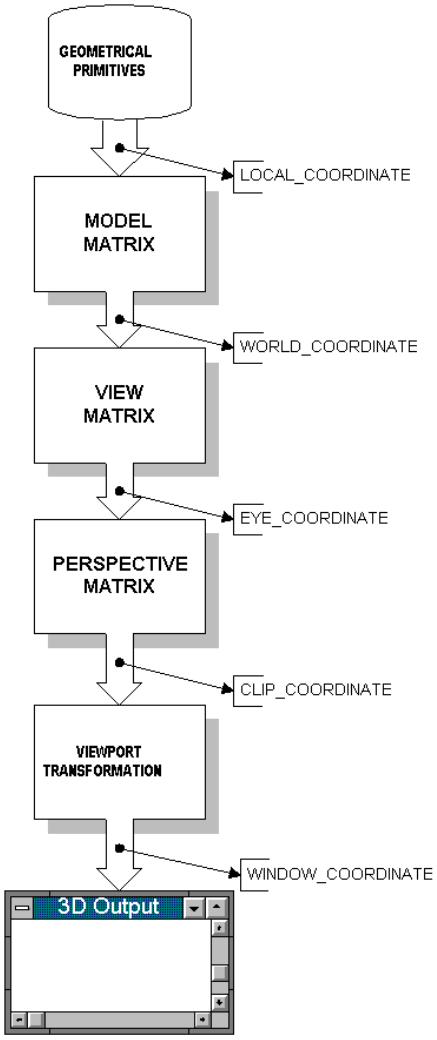

Figure 6: The transformation pipeline used by OpenGL

- In general, it is assumed that the primitives to be visualised can be specified in a local frame of reference; OpenGl transforms the local coordinates to the absolute frame of reference (world_coordinate) by means of the model_matrix.

- The second matrix (view_matrix) takes care of the position and orientation of the camera. All the primitives expressed in world_coordinate are transformed in the frame of the camera (eye_coordinate).

- By means of the third matrix (perspective_matrix) OpenGL performs the perspective correction on the primitives. The coordinates obtained from this step are called clip_coordinates.

The primitives that after the transformation in clip_coordinates are outside the sight volume, are discharged from the processing flow. The other primitives undergo other further elaborations with the goal of obtaining the pixel coordinates for the visualisation on the screen (window_coordinates).

OpenGL allows also to specify an illumination for the scenario. In this case the illumination model is very similar to the Phong's one. Three types of light sources are considered: directional, point and spot-like, here presented in terms of increasing computational costs. The introduction of a new source in the description of the scenario implies an increase in the amount of processing performed in the pipeline.

OpenGL allows to trace primitives with uniform materials or to specify a bitmap image to be used as a texture. Textures are a very powerful tool and allow photorealistic visualisations also in real time graphics.

4 The Graphical Subsystem for VE Applications

After having introduced the available hardware and software resources, the problem of visualisation is considered by focusing on the specific requirements for Virtual Environments applications.

Although the specific sector of VE shares a lot of aspects with the general problem of 3D visualisation, it is necessary to consider the critical objectives that can raise in this particular application scenario. The design guidelines for the development of a graphical module for the visualisation of polygonal environments with a high level of complexity can be summarised in the following points:

- Maximum efficiency: as pointed out before, the quality of the experience with VE systems is strictly correlated to the degree of interactivity of the Virtual Environment. The system must be able to react in very short time to efferent data (e.g. position sensors); this implies that the maximum speed in the visualisation process must be obtained. This goal can be achieved, when necessary, also by utilising preprocessing techniques for the analysis of the polygonal data base and for the construction of data structures in such a way of accelerating the graphical rendering when required by the interaction with the user.

- Maximum modularity: in VE application, the 3D visualisation in real time is only one of the necessary activities in order to obtain an immersive experience. Other computing processes contribute to the simulation: e.g. force feedback , collision detection and, in general, processes for the acquisition of data. Each of these processes shares with the rendering process the characteristic of being a time_critical process, i.e. in which the time required for the data processing is the main parameter for the evaluation of the quality. Among all the previous activities there is a large exchange of information , for which the same efficiency considerations apply. Then it is needed that the software module for the visualisation can integrate with all the other activities and foresee efficient mechanisms of interaction with them. Moreover, due to the extreme variety of application fields considered in VE, the visualisation module must be open to the introduction to new functionalities, by maintaining a logical architecture capable of being expandable without strong changes in the whole architecture.

- Maximum flexibility of use: the contents and typologies of the applications in VE are very different; for this reason, in order to allow a general use, the visualisation module must possess a high flexibility of use. In particular the utilisation and efficacy of its functionalities must be guaranteed in order to tackle different ranges of condition of use.

As described in the previous Sections, the visualisation of a virtual means to send to the rendering pipeline data coming from a polygonal data base which has been previously modelled; at the same time geometrical operations (such as roto-translation transformations) can be performed on the basis of external data. In the realisation of a software module that allows an interactive visualisation, the following considerations must be taken into account:

- The scene to be visualised is the result of a composition of several objects, each of them is represented by a polygonal mesh;

- Every mesh possesses several attributes in terms of colours, materials and textures;

- It must be possible to give to each object, and independently from the others, a position and an orientation in space. Such geometrical information must be modified in time, in order to simulate the motion of the object;

- In the same scenario meshes composed by few polygons must coexist with other meshes possessing a high degree of complexity. No a priori assumption can be done about the complexity of the models;

- For the illumination of the scene one or more light sources must be defined, each of them freely movable in the space and possessing specific characteristics in terms of colour, intensity, etc.;

- In order to easily represent articulated characters or complex mechanical parts, the use of kinematic chains must be foreseen. Each chain defines the hierarchy among the objects and the application of

coordinate transformation must correctly evolve in the chain;

- In order to augment the range of possible appliactions, the scenario can contain both rigid and deformable objects. To this aim it could be extremely interesting the use of interactive procedures in order to deform some of the polygonal meshes used;
- To simulate the immersion of the user inside the virtual environment, the scene must be visualised according to arbitrary points of view and orientations.

to be overcome for the realisation of a visualisation module possessing the above characteristics can be summarised in the following three points:

1. To succeed to interactively visualise polygonal databases possessing a complexity of several degrees of magnitude higher with respect to those that can be managed with the available computing resources. Examining in parallel the course of graphical capabilities of the computers and the complexity of the polygonal models utilised in the field of Virtual Environments, it can be noted that the models have always been more complex than those that could have been treated at the required speed by the available hardware. Moreover the increase in the complexity of the utilised models has been higher than the increase in computing power. This fact means that the management of large polygonal databases cannot be solved by only increasing the performances of the graphical hardware;

   The main difficulties

2. To study and implement an algorithm that allows the real time deformation of arbitrary polygonal meshes. Due to the large amount of geometrical data to be manipulated in very short times, it is necessary to identify a strategy allowing to obtain the best ratio between the needed computing power and the visible obtainable results.

3. To realise a logical architecture of the visualisation module in which, besides being satisfied the previous requirements, high flexibility and easiness of use can be maintained.

5 Conclusions

In the paper the objective of designing an adequate software module for the interactive visualisation of complex Virtual Environments has been introduced. The work performed at PERCRO in this framework has produced an innovative graphics library that has been successfully tested and implemented for the application of Virtual Environments technologies to the field of Cultural Heritage.

References

[1] Sheridan T., "Telerobotics and Human Supervisory Control", MIT Press, 1992.

[2] Phong Bui-Thong, "Illumination for Computer-generated Pictures", Comm. ACM 18(6), June 1975.