1. Introduction

The ultimate goal of all 3D graphics systems is to render 3D objects on an inherently two dimensional surface, which may be a plotter output, a screen or anything similar. One way of achieving this is to start from the medium itself, and to cast so–called *rays* (that is half–lines), starting from the viewer's eye and determined by each raster point of the display, back into the scene. The intersections of these rays with the three dimensional objects will determine the visible raster dots on the view surface. This is basically what the technique called *ray–tracing* does. Although ray–tracing gives the possibility to perform complicated shading, transparency, translucency, and shadow calculations on all intersection points and can therefore lead to highly realistic images, the computational requirements of this approach are still so high that it cannot be used for interactive applications (for a more consistent description of ray–tracing see for example [Fole90] or some other standard textbooks on computer graphics).

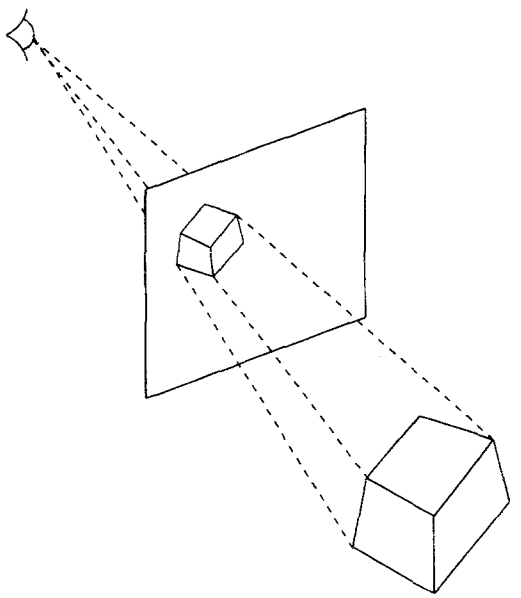

Figure 1.1.

Another approach, which is adopted by most 3D graphics systems as well as the different ISO documents on 3D graphics (for example GKS–3D, PHIGS, PHIGS PLUS, CGM Addenda etc; additionally to the ISO documents themselves, the reader may refer for example to [Arno90] for a good overview of these standards) is to design a system which performs a central or a parallel projection of the objects to render onto the view surface. As this projection is done on the geometrical level (that is by describing the projected lines/polygons etc. on the 2D surface), this

approach requires fewer calculations than ray–tracing. As a consequence however, these three dimensional systems have to make use of the mathematical results of *projective geometry*.

Early three dimensional systems explicitly implemented the parallel or central projections as shown in figure 1.1, that is they calculated the intersection points of the projection lines with the view surface to determine the outlines of the projected objects. In more modern systems as well as in all the cited ISO documents the approach is different. These systems define and perform a transformation of the whole scene, so that the central projection becomes a simple parallel one onto the $z=0$ plane. More precisely, the transformation should result in a projection such that:

$$(x,y,z)^T \rightarrow (x,y)^T \tag{1.1}$$

This means, mathematically, that the transformation to be applied is such that the view point is moved to the "point at infinity on the z–axis". The reason for this approach is that performing the parallel projection after this transformation creates the possibility to perform the so called Hidden Surface/Hidden Line calculations efficiently, that is to determine which part of the scene is effectively visible on the view surface (this presupposes, however, that the algorithms make use of the relative magnitude of the z values only). It also makes the necessary clipping processes computationally simpler. Furthermore, efficient hardware can also be used to perform these computations (for example the so–called *Z–buffer* calculations) easily and effectively (for the details, see the cited ISO documents in [ISO88], [ISO88b], [ISO89], [ISO89a] or any of the general works on computer graphics like [Newm79], [Fole84], [Magn86], [Mudu86], [Salm87], [Watt89], [Fole90]).

The mathematically precise formulation of what is stated above is to introduce the notion of ideal points, which are the mathematically precise definitions of the "points at infinity". *Projective geometry* is the branch of mathematics which provides a uniform description of ideal and Euclidean points. One may also speak of projective planes (ie Euclidean planes extended with ideal points) and projective spaces (ie Euclidean space extended with ideal points).

Using these notions one can say that all 3D graphics systems tend to perform their internal calculations in the projective space rather than the Euclidean one. The fact that a projective space has its natural coordinate system by means of homogeneous coordinates makes this approach feasible: if a homogeneous coordinate system is chosen, all projective transformations may be described by (homogeneous) 4×4 matrices and the effect of the transformation is then a matrix–vector multiplication. The use of homogeneous coordinates also gives a uniform way to describe efficiently all transformations usually used in graphics systems: rotations, scaling, shearing and translations.

Although a number of calculations (ie line/line intersections and the like) can be performed easily with homogeneous coordinates, the fact of working in projective rather than Euclidean space is the source of additional complications. A number of graphical output primitives are described inherently in a linear fashion, that is

using different linear combinations of the points and vectors involved in 3D. Examples are the description of colour patterns (usually a rectangular array is to be calculated) and high precision text (described as a set of small line segments). While a "traditional" transformation of a Euclidean space does keep this linear structure, this is definitely not the case for projective space and projective transformations, which might lead to distortions which are not easily describable by linear means. As a consequence, the implementor of a graphics system might choose to perform all such generation (that is to generate the series of points approximating the high precision text) before the transformation and to transform, as a second step, the generated points. This leads clearly to a loss of efficiency: the number of generated points may be very high and therefore the computational cost of the transformation itself may be significant.

The use of projective transformations and projective space may lead to other problems. In the course of a projective transformation it may happen that some of the (originally Euclidean) points of the objects to be rendered become ideal points (in homogeneous terms this means that the last coordinate value, usually denoted by w, will become 0). A graphics system must have some means to deal with such situations which can lead, if not properly handled, to computational singularities and unexpected visual effects on the screen.

A common framework to handle these problems may be found if the exact mathematical behaviour of projective transformations in graphics systems is carefully analysed. The derivation of such a framework is the central subject of this thesis. It will be shown that a mathematically precise description of the projective geometrical nature of a 3D (or even 2D) graphics system leads not only to a deeper understanding of the system but also to new approaches which result in faster or more precise algorithms.

The thesis aims to be self–consistent for all those computer scientists who have a general knowledge about computer graphics. However, no preliminary knowledge about projective geometry is needed; instead, an extensive introduction to the subject with all necessary notions and theorems will be given. It is well–known that projective geometry is (unfortunately) missing from the usual curriculum for computer scientists and apart from the excellent book of Penna and Patterson ([Penn86]) and the tutorial at the Eurographics'88 Conference ([Herm91]) there are no other textbooks available for specialists of computer graphics. It is of course true that a large number of textbooks are available as an introduction to projective geometry; some of them are listed among the references ([Coxe49], [Coxe74], [Crem60], [Fisc85], [Hajó60], [Heyt63], [Kell63], [Keré66], [Lanc70], [Rose63], [Stru53]) and the list is certainly not exhaustive. (The author's knowledge of this area originates mostly from the lectures made by Prof. M. Bognár at the University of Budapest in the year 1971, from which only hand–written notes are available.) A common problem with these books is that they were written *by* mathematicians *for* mathematicians; in other words they stress different aspects of projective geometry from those needed for the purposes of computer graphics. Whereas they present very general and important theorems, the *computational*

aspects of the mathematical structures tend to be disregarded; in other words, it is sometimes quite difficult to extract the specific information which is necessary for the purposes of computer graphics. Those readers, however, who have the necessary background in projective geometry, may bypass chapter 2 altogether and start at chapter 3 directly.

The material included in this thesis has been admittedly influenced by the algorithmic problems arising when implementing one of the ISO 3D graphics standards. Some of the ideas have been implemented when the author took part in a GKS–3D and a CGI–3D implementation in the years 1986–87 at Insotec Consult GmbH in Munich (Federal Republic of Germany) (see also [Herm88b]). PHIGS and PHIGS PLUS have also generated a number of additional problems while some of the works presented here (and having been published already elsewhere, like [Herm89]). However, most of the problems presented in the sequel are not exclusively proprietary to the standardisation activities; in fact, they are inherent to most 3D graphics systems in use and/or in development such as the Doré package of Ardent Computers [Arde87], RenderMan of PIXAR [Pixa88] and others still to come[†].

[†]The main outlines of all the cited ISO standards are however considered as known and will not be detailed in what follows. The reader should refer to the relevant ISO documents or the existing textbooks and tutorials on the subject, as for example [Hopg83], [Salm87], [Arno90], [Hubb90], [Howa91] or [Hopg91].