Computer Graphics and Human Computer Interfaces

J.R.Gallop
CCLRC Rutherford Appleton Laboratory, Didcot, Oxfordshire, UK

Abstract

This paper provides a brief summary of some of the topics being covered in the lectures on computer graphics at the CERN School of Computing.
Keywords: Computer Graphics.

1. Uses of computer graphics

Computer graphics is now in wide use across a multitude of applications which include science, engineering, education, entertainment and medicine. It can be used:

- as part of the design process: a computer model of a proposed artefact - whether a cassette recording head or a civil airliner - is constructed, assessed, updated and used as the master for consultation, manufacture, maintenance and upgrading.
- to visualize complex data from experimental sources or from a computer simulation.
- to provide information
- to entertain with fast moving and/or complex games

2. A pixellated world

When talking about computer graphics, one commonly sees reference to addressability, RAM, bitmaps, pixels, limited colour maps and so on. These are important, but they represent the limitations inherent in the hardware we have at our disposal.

It is important not to be over-influenced by the discrete limitations of the actual devices. The user wishes to perceive a representation of an idealised model - whether this be an engine part of a civil airliner, a car in a racing game or earth sensing data. The user also wishes to interact with the idealised model - control the view of it, alter it, evaluate it. The task of computer graphics is to allow (indeed encourage) the user's active engagement with the idealised model and to reduce the inhibitions imposed by the limitations of actual devices. These limitations can include:

- spatial resolution,
- number of colours and brightness levels,
- time.

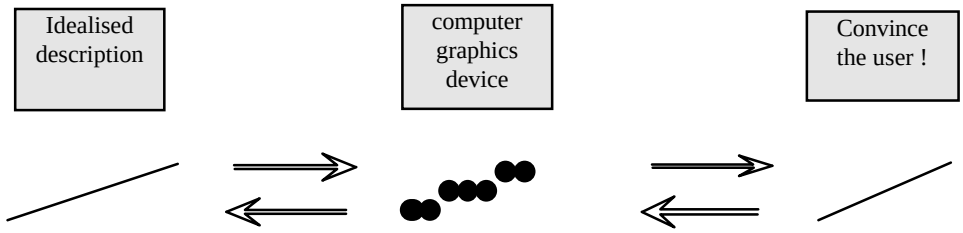

Figure 1: The ideal and the reality

Most devices in use today are raster devices, which rely on being able to store a pixel value at each cell of a raster. The number of available pixels in each of x and y limits the spatial resolution of the device. The maximum value at each pixel restricts the brightness and range of colour that is possible. We show in this table some typical values for commonly found devices.

	x * y	**dpi**	**possible pixel values**
Laser printer		600	2
Dye sublimation printer		300	2^{24}
Publication printer		2400	8
Quality display	2000*1500	150	2^{24}
VR headset	640 * 480	-	

Table 1: Table of device capabilities

How many brightness values are enough? Although the human visual system can detect enormous brightness range. (approximately a factor of 10^{10} to 1 between bright sunlight and the dimmest detectable list source [1]), it does not do so all at the same time, as the aperture˝of the eye compensates. The higher end of the dynamic range of a CRT is approximately 200 and the typical range for black and white printing on coated paper is about 100 (black and white on newsprint is 10).

Supposing that the brightness levels are evenly distributed over the dynamic range of the display device, one could make a calculation of the number of levels required (a more detailed discussion appears in [2]). We must realize that the eye perceives the ratio between two intensity levels not the arithmetic difference: thus the perceived difference between 0.1 and 0.2 appears to be the same as between 0.4 and 0.8. The reproduction on the graphics output medium is perceived to be continuous when the ratio is about 1.01. Thus if the dynamic range is d and the number of levels is n, we have

$$1.01^{n} = d$$

Typical values of n of a good CRT under good conditions are 530 and for black and white on coated paper, 465. But evenly distributing the brightness levels over the dynamic range is difficult to achieve ....

Note that we have been discussing the requirements for continuous tone. If we are using colours to present distinct meanings to the observer, the situation is different. Consider as an example the map of the underground train system of London or Paris. Beyond about 10 colours, distinguishing certain pairs of underground lines is quite difficult.)

The limitations of time are partly those of speed - can the processor achieve the task fast enough. However even if the processor is fast, we also have operating system software problems which may prevent us achieving synchronisation and time accuracy. For some applications (video-conferencing, movie playback), synchronisation between vision and sound is vital. For these and other applications (visualizing a time-varying sequence), time accuracy is essential.

Let us recap. The application programmer has an idealised description of the graphics to be presented and the interactions to be allowed. The task of computer graphics software is to overcome the quantising effects and other limitations of actual devices so that the user can interact with the idealised model as faithfully as possible.

3. 2D graphics

Although 3D graphics is becoming more widely used as hardware capable of supporting it well becomes cheaper, 2D graphics continues to be important for images, schematics, charts and diagrams.

3.1 The idealised form

As we discussed, in most applications there is an idealised model of the geometry under consideration. Somehow this needs to be converted to the actual, discrete graphics device.

Some 2D graphics software allows the application program to specify this ideal form and is therefore responsible for achieving the best possible results on the graphics device. Points and vectors are specified in a coordinate system (usually Cartesian) chosen by the application programmer. Colours are specified as real numbers. An example of this software is the ISO standard for 2D graphics programming, of which the most recent version is GKS-94 [3].

Other (lower level) software allows the application program to have close control over the specific pixels of the graphics device. Xlib (in the X Window System) allows this. It is the responsibility of the application program to map from the application's idealised model to the specific points and colours of the graphics device.

3.2 Output functions

To allow the application to specify the 2D output in a idealised form, what output functions do we need. In principle, we need a 0D function (draw points), a 1D

function (draw lines) and a 2D function (draw areas). In practice this is too spartan and an application programming interface needs more support than this.

It is instructive to summarise the output functions supplied in GKS-94 as these will be regarded as a minimum for other packages in future.

- SET OF POLYLINE: set of curves, each of which is a sequence of connected lines defined by a point sequence
- SET OF NURB: set of curves, each of which is a NURB (discussed later)
- SET OF CONIC SECTION: set of curves each of which is a conic section
- POLYMARKER: set of symbols of one type centred at given positions
- SET OF FILL AREA: set of areas each of which is defined by a closed sequence of connected lines
- elliptic arc primitives
- TEXT
- CELL ARRAY: array of cells with individual colours
- DESIGN: a tiling may be extruded through a stencil which may consist of area boundaries, closed contours and others.

3.3 2D transformations

Having defined 2D objects by means of some output functions, we need to think about how 2D graphics are transformed. There are two main reasons:

- We may be composing a complex schematic such as a floor plan. It is convenient to define an object shape and then to place it in the schematic. Placing it may involve rotating it and scaling it as well as translating the object to its desired place.
- Having defined the complete floor plan, we may wish to view different parts of it at different magnitudes. We can think of a camera panning, zooming and rotating. In 2D this panning and zooming usually achieved when the coordinates are converted from the 2D world of the application to those of the device. However in 3D, viewing becomes a more complex issue as the camera movements can be more complex.

These 2D transformations can be achieved by the application of straightforward 2D matrix operations. We can represent a point by a vector [x,y] - many introductory texts in fact use the convention of a column vector.

To rotate a point about the origin, multiply the point by a rotation matrix.

To scale a point with respect to the origin, multiply the point by a scaling matrix.

But, to translate a point, add a translation vector. This is unfortunate because (for efficiency and simplicity) we would like to transform the point in one operation. If we could only use matrix multiplication for everything, we could win. What can we do about translation?

The answer is to introduce homogeneous coordinates. We can represent a 2D point by a vector

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

By taking care with zero, we can establish the convention that

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ and } \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

are the same point.

We can make scale and rotate work as before by extending them with 0's and 1's:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & t & 0 \\ u & v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We can now create a˝matrix for translation which allows us to use multiplication:

$$\begin{bmatrix} x+tx \\ y+ty \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This is a greatly simplified view and more detail can be found in the introductory texts [2] and [4].

3.4 Object structure

In many applications it becomes important to group objects so that they can manipulated together or separately. A group of objects may need to be identified by the user using some pick mechanism and hence selectively made invisible or highlighted in some way. This becomes even more important when we think about 3D.

Various methods have been used. PHIGS (and PHIGS PLUS [5]) uses a method of hierarchical structures. This design is useful for manipulating scenes made up of objects which are made up of a collection of rigid parts.

GKS-94 uses an approach called Name Sets. Each output primitive created through GKS-94 has a set of names associated with it and filters can subsequently be applied. Instead of the hierarchical approach of PHIGS, this can be thought of as a set of layers which can be handled independently. This approach has promise in the field of cartography.

Open GL also uses a hierarchical scene database.

3.5 2D computer graphics file transfer and archival

At some point it becomes necessary to transfer pictures to another system or to archive them. One way is to create an image and archive and store that. There are many image formats and most readers will be aware of them.

Less commonly known about is the Computer Graphics Metafile (CGM). This is an ISO standard which was revised in 1992 [6]. It contains graphics primitives, not just images. Converters and interpreters are available on PCs, Macintosh's and Unix workstations and a variety of graphics software can create as a CGM as an output driver.

CGM can also fill a hole in the World Wide Web. Images have the problem of not being scalable and also require much network bandwidth even when compressed. Many pictures are better stored in the form of geometric primitives which CGM does.

At present CGM is 2D, but there are moves to produce a 3D version.

4. 3D graphics

3D graphics is now in widespread use. It is in extensive use for Computer Aided Design. Data visualization software often results in an abstract 3D model. Virtual Reality presents a 3D world to the participant and tries to give the best possible sense of "being there". Most home computer have some 3D graphics software.

Although it is usually not necessary to understand the underlying algorithms, people who are creating application software using 3D graphics need some understanding of the many options. Even people who are using pre-written applications software need some grasp to enable better choices to be made. In this section we introduce some of the problems that need to be thought about.

4.1 Models and views

When discussing 2D graphics, we introduced the idea of composing the floor plan and as a separate operation viewing it through a camera. This distinction between the two processes, which is often blurred in 2D systems, is even more important in 3D.

A 3D graphics system allows scenes to be composed of objects. The objects themselves may be defined in a hierarchical way i.e. there are subobjects. A table may consist of a table top, a number of legs and, if the detail matters to the application, the fixings that hold it together. The scene may be a room which may consist of the walls doors and also the contents of the room. At the lowest level each object needs to be defined in terms of some output primitives.

A 3D graphics system also allows scenes to be viewed and one method is to simulate a camera, with a position and a view direction.

4.2 Output primitives

Output primitives fall into a number of categories (PHIGS PLUS is used here as a source of examples).

Some output primitives are inherited from a 2D system but are available in 3D space. Examples of this are polyline, polymarker and fill area.

Other primitives consist of multiple polygons. Examples are triangular set, triangular set and quadrilateral mesh. An object may be represented entirely in terms of triangles in which case it may be represented by the single primitive triangular set, which allows the 3D graphics system to handle shading and hidden surface removal correctly.

The other category is represented by curves and surfaces. Often the ideal for the application is a curved line or surface. Complex meshes often result from an object that has already been approximated.

There are many ways of representing a curve. If at all complex, it has become common to use a piecewise representation, which avoids a polynomial representation of a high degree.

The B-spline concept is one example of this. It can be uniform or non-uniform, which refers to the spacing of the knots - the knots divide the subinterval over which each piece is defined. A B-spline may consist of non-rational or rational polynomials. With the rational form, weights are applied to control parts of the spline more tightly. The most general of these is the non-uniform, rational B-spline i.e. the NURB. This is found in PHIGS PLUS and other systems.

B-splines have a number of advantages:

- Compared with some other types of curve, it is possible to exert local control of the shape of the curve. This is important in many design applications.
- A NURB can be used to model all conics exactly with a small number of control points.
- NURBS are invariant under perspective transformation.

4.3 3D transformations

In 2D graphics, the operations of translate, scale and rotate are achieved using 3 x 3 matrices and, with the modification we described for translate, using matrix multiplication. This allows a user to expect a graphics system to compose a non-trivial transformation without loss of performance.

The same operations can be used in 3D using 4 x 4 matrices and non-trivial transformations may be composed in the same way.

However in 3D we also have to deal with projecting the 3D scene onto a 2D graphics device. We have to assume a viewer or a camera, pointing at a screen through which the scene can be viewed.

camera	screen	object
x,y,z=0	z=v	(X,Y,Z)

Since we have 3D rotations and translations, we can always transform the 3D coordinate system so that the coordinates are simplified as shown. We seek the (x,y) coordinates of the projection of the object on the screen.

For a perspective projection, this is a straightforward division:

$$(Xv/Z,\ Yv/Z)$$

and we retain Z for any calculations relying on depth.

For a parallel projection, it is even simpler as there is no division.

4.4 Rendering, lighting and shading

A 3D scene can be rendered in a number of straightforward ways:

- wire frame: display all edges
- hidden line removal - display only those edges and parts of edges which are not obscured by solid objects
- flat shading (or constant shading).

These have the attraction of being fast, but provide limited information about to the user about the scene. It is difficult to perceive the relative depths and gradients. We therefore need to think about lighting and shading. First a simple principle:

We see things, that are not themselves light sources, by the action of light being reflected off them of transmitted through them.

Therefore we need to understand surfaces (and interiors) and lights.

4.4.1 Conventional model for surfaces and incident light

There is no real agreement on the most suitable surface and lighting model to use. Often the determining factors are pragmatic ones. We outline here a simplified model which is in common use.

We first observe that light can be modelled by considering

- *ambient illumination*, in which there is no variation in intensity of direction
- and a set of specific *light sources*. When light from any of these sources hits a surface, the direction of incident light is defined.

We next model the reflection from a surface by splitting it into these components.

Firstly the response to ambient light depends on:

- a factor for ambient light reflection for this surface (k_a)
- a factor for diffuse reflection for each colour w for this surface (O_{dw})
- the intensity of ambient light (I_{aw})

The response to ambient light is: $k_a\, O_{dw}\, I_{aw}$

Secondly, the diffuse (rough) response to the specific light sources depends on:

- a factor for diffuse light reflection for this surface for each colour (k_d)
- O_{dw} as for ambient light
- the intensity of incident light from each specific light source (I_{sw})
- the angle between the direction of incident light and the normal vector to the surface (θ)

The diffuse response is: $k_d O_{dw} I_{sw} \cos \theta$ summed over all light sources

Thirdly the specular (shiny) response to the specific light sources depends on:

- a factor for specular light reflection for this surface (k_s)
- a factor for specular reflection for each colour w for this surface (O_{sw})
- the intensity of incident light from each light source (the same as for the diffuse response I_{sw})
- for each light source, the angle between the viewing direction and the direction of the maximum reflection (α)
- a specular exponent for the surface, expressing the concentration of the reflected light about the direction of maximum reflection (c)

The specular response is: $k_s O_{sw} I_{sw} \cos^c \alpha$

This specular model was one of Phong's contributions (except for the O_{sw} term) - the Phong specular reflection model.

4.4.2 Shading interpolation

Often the application approximates a curved surface by a large collection of polygons. To calculate the reflected light, we need surface colour information (called the intrinsic colour in PHIGS PLUS) . Also, when shading, we may want to simulate a curved appearance, so in that case we need surface normals everywhere.

Unfortunately if the curve has been approximated by polygons, the surface normal is either available only at the vertices or worse still has been discarded. In this situation, it is necessary to interpolate the normals across each polygon and, at each point where needed, use that to calculate the reflected light everywhere. This is normal vector interpolation or Phong shading.

Since the lighting calculation is expensive, an alternative is to calculate the colour at the vertices and interpolate colours everywhere across the polygon. This is usually much faster and is the principle behind Gouraud shading which is widely implemented in hardware. It is quite sufficient for many purposes, especially if there are no highlights. There can be some visual interference if the intensity changes across the object's surface rapidly.

4.4.3 Ray tracing and radiosity

Algorithms described so far have gained wide acceptance in graphics hardware. However the solutions are far from perfect. One problem is that only single reflections are taken into account. The appearance of most real-life scenes is influenced by complex reflections and deeper algorithms are needed.

In outline, ray tracing algorithms are quite straightforward. A light ray emanates from a light source, it may hit one or more surfaces and may pass to the lens of the camera. However the number of light rays that never appear in the picture is enormous so this would be extremely w wasteful.

Instead a ray is traced backwards. Given a pixel, we have to find which light rays contribute to it and what colour do they contribute? The fundamental operation is what object did this ray come from last, at what point on its surface was it incident and what is its angle of incidence.

Calculating the intersection of each object with each ray is time-consuming as there are many rays and (usually) many objects. Since the first ray-tracing algorithms, many improvements have been made, taking advantage of regularities in the scene or using parallel processing.

A full description of ray-tracing can be found in [7]

Since it is difficult to trace a ray backwards beyond a diffuse reflection, the method is suited to specular reflection, which is way these algorithms usually are usually seen producing highly polished surfaces. Unfortunately most real life scenes have a predominance of surfaces with a high diffuse component. Radiosity algorithms attempt to solve this.

A radiosity algorithm models all the light energy passing between the surfaces in a scene. It allows for light being reflected in many directions off a surface.

5. Summary

In this paper, an attempt has been made to introduce in straightforward terms some of the concepts of 2D and 3D computer graphics.

6. References

[1] S.J.Thorpe, *Image processing by the human visual system,* Eurographics Tutorial 4 (1990).

[2] J.D.Foley, A. van Dam, S.K.Feiner, and J.F.Hughes, *Computer Graphics Principles and Practice,* Addison-Wesley (1990).

[3] ISO, *Information technology - Computer Graphics and image processing - Graphical Kernel System (GKS) Part 1 : Functional description* ISO 7942 (1994).

[4] D.Hearn and M.P.Baker, *Computer Graphics,* Prentice-Hall (1994).

[5] ISO, *Information technology - Computer Graphics and image processing - PHIGS PLUS* ISO 9592-4 (1992).

[6] ISO *Information technology - Computer Graphics and image processing - Computer Graphics Metafile* ISO 8632 (1992).

[7] A.S.Glassner (ed), *An introduction to ray tracing*, Academic Press (1991).