Module: MSC MD (INTMMT) **Seminar 2**

Introduction to Computer graphics and Vector graphics

Introduction

One of the key attractions of any multimedia production is its visual appeal and this is due to the graphics used. They may simply delineate the page or be an integrated part of a presentation, they may still or moving, coloured or monotone however they all add the most significant value to the whole. This seminar is about graphics in general and the software and hardware that makes them possible. We will leave video and animation to Seminar 3. So, for us, we take graphics to mean the use of still images to convey meaning or add value in a multimedia presentation. Graphics are a means of visual communication used throughout mankind's development, from the early cave paintings to modern stereoscopic displays but what are we trying to communicate? This is of fundamental importance to our choices for style, software or hardware used and the design of the image. Look at a young child's book. The images are bright and clear using few colours because the child's ability to understand subtle shading and complex images is not fully developed. Now look at any Renaissance painting with their vivid colours and complex settings. Finally, examine a photograph from a fashion magazine. Each of these demands a different set of tools and each sets a minimum level for the technology used to create them and on which they are created. A child's book will require limited use of colours which should be bold and with no fussy detail. This can be achieved with a cheap paint package on a relatively cheap machine. The Renaissance painting requires more control and greater colour ranges which means we need to go up-market for our tools. Lastly, the photograph in a fashion magazine requires the computer and its software to match the original colours precisely and allow manipulation of light and shade to create atmosphere. Here we will need to use expensive packages. The previous examples look only at two dimensional images but we can find similar examples in the 3-D world.

To appreciate the power of graphic presentation I would like to introduce you to what Edwards Tufte described in his classical book "The Visual Display of Quantitative Information" (Tufte, E.R 2001) and the website (Posters, Anon) as the "Probably the best statistical graphic ever drawn". This is a superb example of what can be achieved through a graphic image detailing the timeline and the events of napoleon invasion of Russia through the use a graphic presentation.

During this seminar we will examine Vector and bit mapped graphics, 2-D and 3-D approaches. However we will leave a more detailed discussion of Bitmaps and how we can manipulate them using application software until Seminar 3. During the module we will introduce you to freeware packages of multimedia related applications and will assign to the same labs as only by playing with the tools will you truly appreciate their capabilities and differences.

Computer graphics has a history almost as old as computers (Carlson 2003, Shoaff 2000) but for many years graphics was a side show to text. In fact many early graphics were made up of text for output to a printer. Perhaps the first useful graphics were those used in line drawings for design purposes. These used a technology which is rarely seen today. They were known as Vector Graphics systems (Vector, Anon) and they drew line drawings directly onto the video display, similar to a pen plotter, but unlike our modern screens which require the image to be scanned first. Vector machines were limited in their use of colours and could not cope very well with filled sectors. They were also very expensive. The mass production of the television ensured that by the time the personal computer generation came along all our screens were raster scan devices.

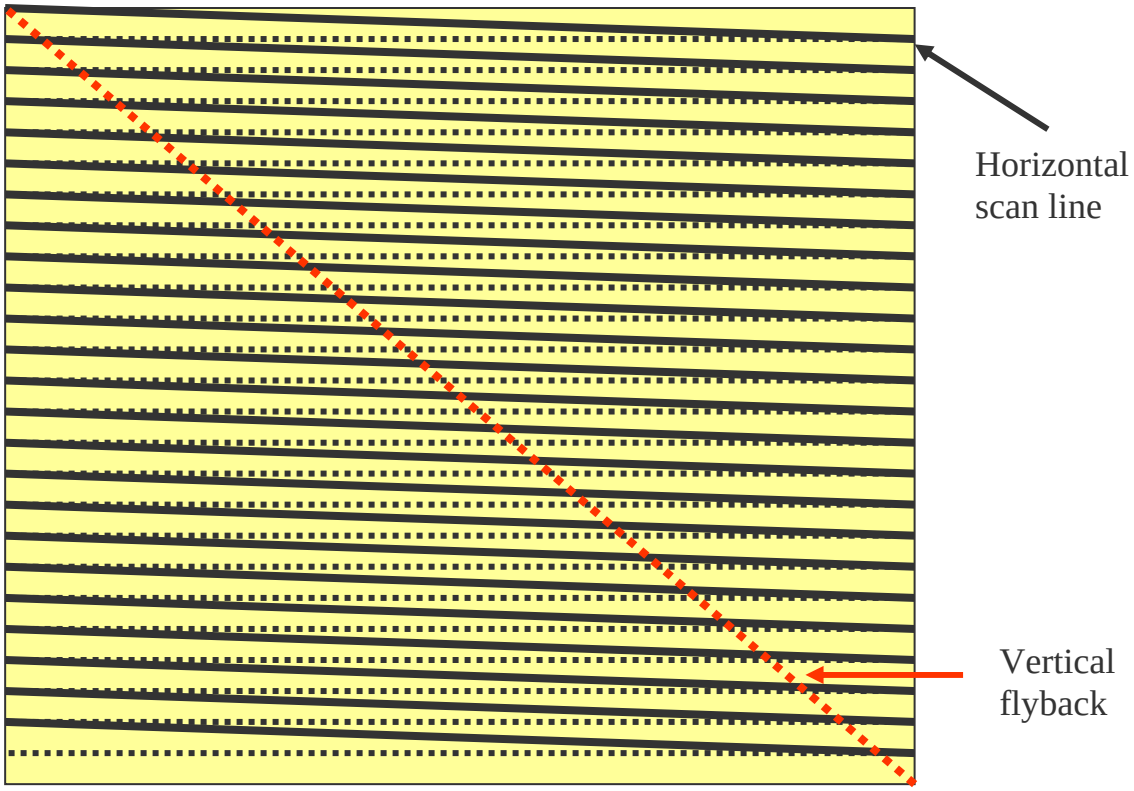

Figure 1. Unseen to us the electronics on a raster device is tracing across the screen. We make a picture by switching the electronic gun beam on and off redrawing the picture between 60 and 100 times per second. As you can see this must lead to distortion.

Today the words Vector Graphics have come to mean the production of images from vectors stored in the computer while the images are displayed as physical picture elements, (pixels). See Figure 1.

It is tempting to think that with modern technology we can produce images to whatever fidelity we wish but let me sound a few words of caution. No matter what we do to improve the software, we have chosen a technological path that is limiting. The screens work by scanning consecutive lines one at a time and this produces artefacts (we shall discuss this later). To overcome this we may use more lines but this just changes the composition of the artefact. The fundamental problem remains. Any increase in scan lines requires an increase in circuit complexity and fundamental changes to the quality of the screen. This has obvious cost implications. The use of high fidelity images implies more pixels and more colours (Seminar 3 discusses colour) which will require more memory. Our screens use only three phosphors to produce the whole range of colours for our PC's and as we shall see later, this sets a limit on the achievable range. The networks have limited capacity and we have limited funding to pay for what there is. This further restricts the style and accuracy of the graphics we can use in an Internet based Multimedia presentation. This leads us to ask what we can do to maximise the effectiveness of our graphics under a given set of circumstances.

Firstly, we must look at the types of graphics currently in use. We use simple graphics to interpret data, sometimes 2-D and sometimes 3-D. These images are generated by the data and designed specifically for display on a terminal. They lack the complexity of 2-D images or the even greater complexity of a 3-D model. Generally they use a limited subset of colours, (Typically eight) and are used for charting results of data processing e.g. simple pie charts or histograms. They are also used in Supervisory, Control and Data Acquisition systems (SCADA) in public utilities where central control of distribution is required. This type of graphic does not concern us in this module. This seminar will specifically examine the use of Vector graphics, bit-mapped graphics and associated 3-D modelling (which requires a perspective translation to be rendered onto a display).

As previously mentioned, Vector graphics is the use of a series of vectors and attributes to store the image. This allows us to set the final image size irrespective of the output technology. It is also a very condensed form of image definition. Alternatively, bit-mapped graphics require us to store a pixel map of the image. This pixel map defines the ideal resolution for the image since each point in the map defines a point on a specific screen size. Increase the number of pixels on the screen and the image will appear smaller (without corrections) decrease the number of screen pixels and the image will appear larger. By now, we should be able to see that each type has its uses, so let us examine them more closely.

Another way to look at vector graphics is as though we were to trace the picture and produce an outline. Each separate piece (such as the petal of a flower or the

wheel of a car) is stored as a descriptor of the outline and its attributes e.g. the colour to fill the outlined area with. This works well for simple drawings but as the image complexity increases so does the number of vectors. Also as colour changes become more subtle it is difficult to delimit the areas accurately. We can see that Vector graphics is good for the creation of cartoons and poster art but lacks discrimination for detailed work.

With a bit-mapped (for bit mapped you can substitute pixel map if you prefer but strictly speaking they are not the same. Pixel is short for Picture – Element and should be reserved for the final output.), image we define a colour value for each element so that it is harder to determine an outline of a given shape. While vector graphics retains objects and can manipulate them separately, the bitmap graphics regards the entire picture as one single object. In figure 2, you can see an endoscope image of a developing ulcer. Would you be able to define the ulcer's limits for storage as a vector graphic? Even non-intelligent automatic systems will fail since there is no continuous path of a single colour present. Bit maps are suited to detailed images where there are continuous tonal changes such as photographs or complex scenes.

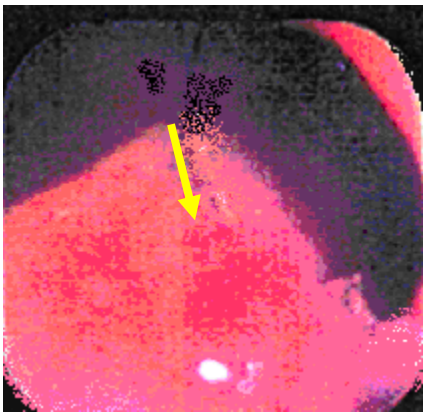

Figure 2. This is an enhanced Endoscopic view of an Ulcer. You should just be able to see different shades of reds which form an irregular pattern. The yellow arrow points to one such area.

Modern bit maps are usually stored with 24 bits for colour and an eight bit alpha channel. The 24 colour bits are separated into 8 bits for each component colour (red, green and blue) and the alpha channel is used for special effects such as shadows or translucency to be applied to that pixel.

Since the two types of graphic are so different it should not be a surprise to find that the tools used to create them differ as do the file formats used to store them.

Drawing packages are often oriented towards the production of Vector graphics whereas Paint packages usually produce bitmapped images. When you draw, you are creating an image from lines so that the source material for a vector

system is implicit. When you use a paint package then you are offered a selection of brushes which produce different effects. In fact, Microsoft Windows uses the concept of brushes when producing graphical output, e.g. a window. Of course there is nothing to stop us mixing the vector graphic with a bitmapped graphic. This is often done when we wish the focus to be cartoon like or posterised against a realistic background. Packages accomplish this using layers to allow easy editing. You can imagine a fish swimming against a coral reef background. The background contains a lot of detail and the designer decides to use a bitmap which is static and requires no manipulation whereas the fish is the focus and star of a cartoon and is dynamically moving. Manipulating vector objects is a simple mathematical operation. Layers allow the designer to move the fish about in the picture without having to re-edit the background every time.

Paint and drawing packages often allow saving to file in a variety of formats to allow user choice and to facilitate compatibility with other packages. (Formats, Anon)

There are many ways to save images in files. Fundamentally, what do you need? We may think that a two dimensional array of storage locations would suffice for a bitmap image providing we can read the locations sequentially and put each locations value on the screen in order. Unfortunately we need a few other items such as the length of the file, its name and the order the elements are in. These items go in the file header. In turn, the structure of the header may limit the size of the file. Several file formats incorporate compression schemes and this must be recognisable to the imaging system. This is why your system may read one type of bitmap and not another. Information in the header tells your system that it requires certain facilities in order to be read; if they are not available then often a downloadable plug-in can be found. File systems such as the Tagged Image File Format (TIFF) were designed to be open and accessible to all but the header was extensive. With the TIFF format you could describe keywords which effectively ensured privacy to a special interest group or header offsets which allowed mixing of images in the one file. Tiff is a very flexible file format but designing readers for it can be complex. Simpler systems such as GIF provide compression but limit the number of colours. Since GIF has a built-in proprietary compression scheme its days are probably numbered. Many new file formats are predicated on the assumption that the device producing them is digital e.g. a digital camera, video or scanner.
In the next two sections we will take a closer look a how Vector and Bitmapped graphics work.

Vector Graphics

Vectors are the basic building blocks for a wide variety of graphical work. From the simple two dimensional posterised work to the complexities of 3D modelling epitomised by computer games and simulators but how do they work? Vectors

are simply numbers which describe a relationship between items. For our purposes, they describe how to reach locations in space. The following is a brief review for those of us who missed Vectors in our mathematics classes.

We can begin by imagining a grid made up of small squares where each square represents a pixel (see figures 3) then if we want to reach a particular point on the grid we simply count the appropriate number of squares in the x and y directions. For example if we wish to move 2 squares in the x direction and two squares in the y direction from the origin we arrive at the yellow square in figure 3a. In mathematics we write, (2,2) or generally (x,y). In some texts, especially those on computer graphics from North America, you may see it written as [2,2]. Now if we want to move we simply need to use a vector to say how far and in which direction. For example we could draw the red vertical line in figure 3a up to t (2,4).
To do this we need simply say Draw 0 squares in x and +2 squares in y from the current position or (2,2) + (0,2) = (2,4). Imagine now a simple machine to which we can give some instructions as follows:

```
Reset to origin
Colour =Red
Brush = Pencil
  Use Incremental
Move (2,2)   /* Without drawing */
Draw  (0,2)  /* Move and draw */
```

In the above pseudo-code we have told the machine to; start at the origin, select a colour to use, select a style of brush (in this case we want to draw a narrow line) move two squares in the x and two squares in y direction, finally draw a line from (2,2) which is 0 in the x direction and 2 in the y direction. See figure 3a. Can you see how this could be extended to draw any straight edged object in two dimensions? Below are the instructions to draw a rectangle.

```
Reset to origin
        Colour =Red
        Brush = Pencil
          Use incremental
        Move (2,2)
        Draw  (0,2)
        Draw (2,0)
        Draw (0,-2)
        Draw (-2, 0)
```

See figure 3b

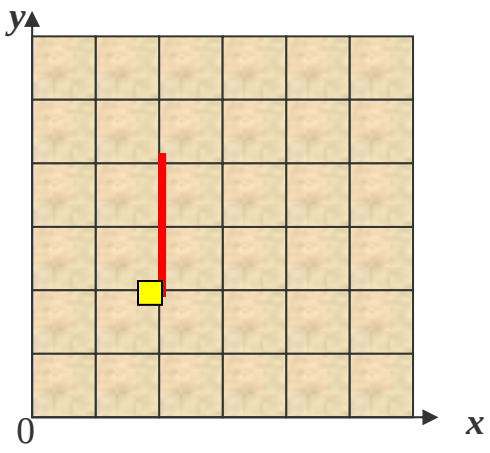

Figure 3a

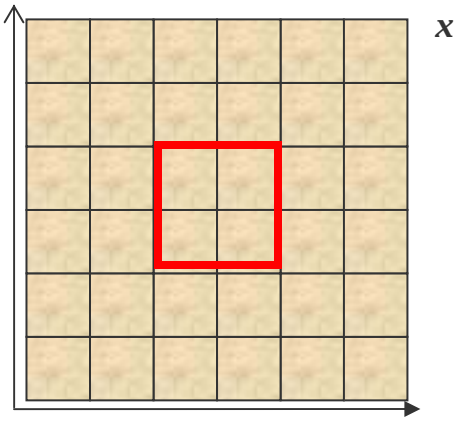

Figure 3b

Notice that the above examples are using relative instructions. We could have used absolute positions, e.g.

Reset to origin
Colour =Red
Brush = Pencil
  Use absolute
Move (2,2)
Draw (2,4)
Draw (4,4)
Draw (4,2)
Draw (2,2)

It has the same effect but without those confusing minus signs. This simplistic system ignores some complex issues.
We are using a coordinate system which starts at (0,0) in the lower left. This is a natural way to think and draw for humans but our common raster scan display draws from the top left to the bottom right. Thus the data presented to it from our memory is read from the lowest array locations first to feed the top right position. We must convert our coordinate system to accommodate this (most graphics packages now do this automatically.) or we must think with (0,0) at the top right and a y axis increasing downwards. For some items this would make no difference but see figure 4.

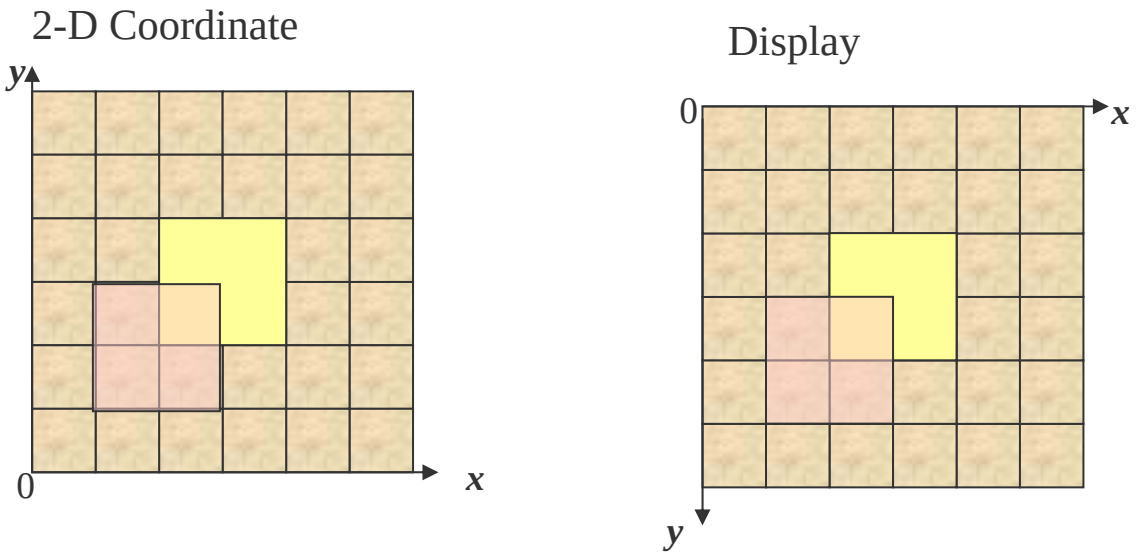

Figure 4 A yellow box at (2,2), (4,2), (4,4), (2,4) does not change but for the pink box …..?

So far we have dealt with integers but that is limited. We can consider a screen to have 640 x 480 pixels and our drawing to have a coordinate system to match but what happens when we change resolution or we wish to have an easily scaled description? We find that we need awkward multipliers. It would be much better if we used a system based on real numbers between 0 and 1. Then we can select any point in between and scale up as required. Another problem with integers occurs because we assign a single colour to the whole area of a pixel. Consider figure xx. There we can see a horizontal line which looks perfectly satisfactory and a vertical line that also looks correct but a 45 line that looks broken and an even worse line at 20 degrees. We have a problem which is described as aliasing due to under sampling of the image. E.g. It would look better if we increased the number of pixels in each direction. We improve this picture by selecting adjacent colours to blend with the line. Our eyes then integrate the sum of these shades to produce the impression of a smooth line. We avoid these problems in our initial drawings by using real numbers with their much larger range. There is then a question of where is the point that represents pixel (0,0)? Surely not at (0,0) in our real number system. It will be at (0 + ½ x , 0 + ½ y). That is, the centre of the pixel. This will allow us to accurately pick our anti-aliasing values from those around the required pixel centre. These problems occur for all shapes and in figure 5 you can see the effects that occur and must be compensated for when drawing a simple circle. Of course the resolution is greatly lowered

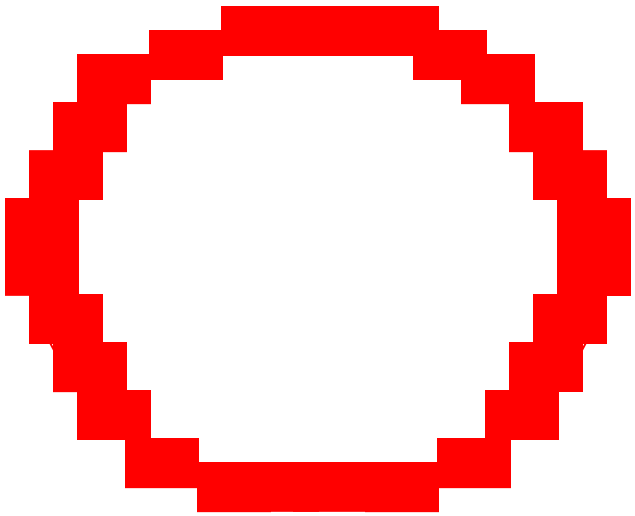

Figure 5. Distorted circle at low resolution. Note the effects on the top and sides. This circle is made from rectangles (a screen pixel is rectangular) and can only follow the underlying true circle to an approximation.

If we can imagine a point, then we can imagine another point and between them we can imagine a line. If we can imagine a line, then we can imagine another line joining it and yet another. If the lines connect, we can imagine a small surface patch, and from that any feasible planar shape. This is the basis of Euclidean geometry. Shapes can be made from lines or from curves which could be approximated by many small lines. Therefore, we can save our images as a set of point descriptions and a connectivity list. There is another way. We could describe our curves (actually a line is a curve with one direction) by mathematical formulae and save considerably on the storage. Complex curves require tools which can be manipulated by humans to produce what is called a ‘fair’ curve e.g. it is ‘smooth’. The most common example is the Bezier tool so often included in drawing and paint packages. A Bezier curve is drawn by controlling basis functions which describe the curve. By manipulating the parameters to the curve it is possible to make the curve bend in the desired manner. In order to experience the Bezier curve you can experiment with it through a Java interactive program (Bezier, Anon).

The actual mathematics is beyond the scope of this seminar however it is instructive to see how parametric descriptions help us produce curves. Let us start with a straight line. In this example we will use a 3D line. The only difference we need concern ourselves with for the moment is the presence of an extra coordinate value which describes the third axis (the z-axis).
A line is defined as a set of points (x,y,z). See figure 6 and we want an efficient way to describe them.

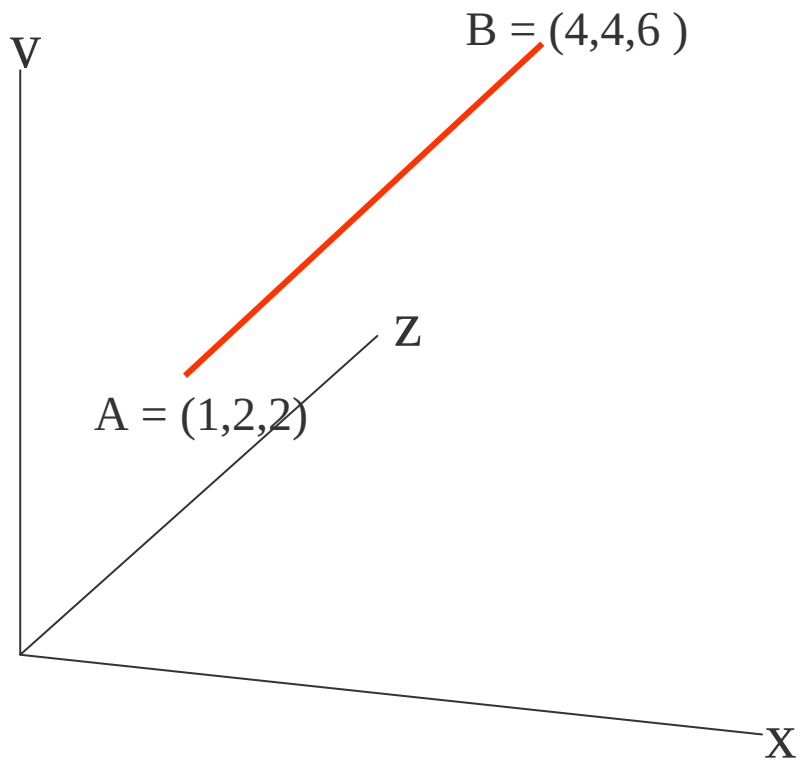

Figure 6. A 3 - dimensional line as a set of points described by their endpoints.

We use a parametric form. In this form we can cause a line (or a curve) to be drawn point by point in a controlled manner. We are used to the formula for a straight line ; y = mx + c. In a similar way we can write ;

$$x = a_x u + b_x$$
$$y = a_y u + b_y$$
$$z = a_z u + b_z$$

Here ‘u’ is the parameter.

We can rearrange the above and insert the ‘i’ designators to give us;

$$a_x u = x_i - b_x$$
$$a_y u = y_i - b_y$$
$$a_z u = z_i - b_z$$

Notice how the parameter ‘u’ is involved in finding all three coordinates; if we change ‘u’ we change the value for all three coordinates.

We would like our parameter to describe the curve from start to finish. So we set it equal to zero at the start and 1 at the end.

Substitute

$i = u = 0$	$i = u = 1$
$b_x = x_0$	$a_x = x_1 - x_0$ /* Because $a_x1 = x_i – b_x$ →
$b_y = y_0$	$a_y = y_1 - y_0$ and $b_x = x_0$, and the same
$b_z = z_0$	$a_z = z_1 - z0$ for the rest */

We have isolated the coefficients **a** and **b**. So we can substitute them to arrive at a purely parametric form which relies only on our substituting for **u** a value between 0 and 1.

$$x = (x_1 - x_0)u + x_0$$
$$y = (y_1 - y_0)u + y_0$$
$$z = (z_1 - z_0)u + z_0$$

We can describe more complex curves in this manner using quadratics, cubics and higher forms. Enough of the mathematics!
But if you are keen on learning the mathematical manipulations that drive all these translation, scaling and shearing take a look at “Transformations in 3D” (Moldhave 2000).

We have our vectors and our shapes what remains in a two dimensional image is to fill the shapes. In a piece of art work we carefully manipulate the internal area of a shape to produce a more realistic form or a more pleasing form brush stroke by brushstroke. Our computer package can do this for us in many cases quite automatically. We can define a brush to be thick or thin and to apply a particular colour just as in real life but our computer can define a dashed brush or tartan or multi-coloured. We can set an area to fill with shades which vary from corner to corner (a gradient fill). We can also define actions across the whole subset of the image. The usual ones are; translations rotations, shearing and scaling. We do this by dragging the object or selecting a point to act upon.

We have quietly introduced the idea of three dimensional coordinate systems whilst looking at 2-D images but is that all there is to it? Of course not. Let us take a closer look.

The obvious difference is we define coordinates as triples (x y z) and objects as a series of triples. See figure

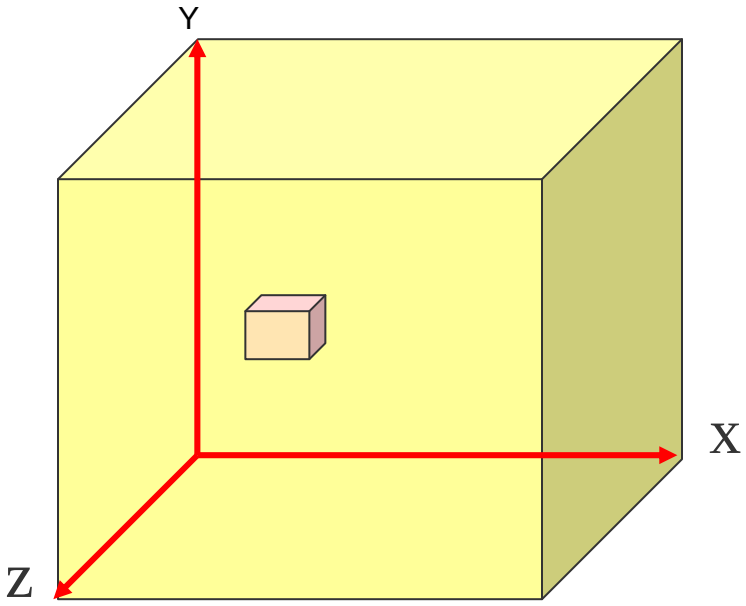

Figure 7. Notice the axis can have negative values. The box inside the yellow area is defined by eight triples, (xyz) just as the yellow box is surrounding it.

Our objects in 3-D are models of the real world and are made from simple primitives such as spheres and boxes which are developed into more complex forms. If we look more closely at our primitives we see simpler shapes; a box has six sides, each side consists of two triangles. We consider our objects to be made from surface patches of varying complexity but at the end of the day we need to put them onto the screen. Let us take a look at the process.

Our models are contained in a database but to get them to the screen requires certain actions to be carried out. We can only show you one possible sequence of events but there are others. In what follows use figure 8 as your guide. We begin with a world coordinate system in which we consider our object to exist and our first job is to transform that to our modelling system. Of all the objects in our world we are only interested in those within our field of view so we can throw away those outside it. This is the trivial accept / reject stage (After all, why process something we have no interest in?). Our next task is to apply a mathematical model of lighting to the objects. In this case the Gouraud model. This is achieved by setting a value for the vertices of each patch and interpolating between the edges to obtain values for each pixel. This can be done later in systems such as those using Ray Tracing. Now we must make the final transformation from 3D to 2D and apply perspective for our view of the

scene to be realistic. Next we clip any partial objects to size. At this point we have only those objects and parts of objects that are visible on the screen. Now we map to the Viewport (on the Pc it is known as a window) on the screen. At last, the process which converts our planar patches to a series of sequential line elements can take place. This is known as rasterisation. Finally we send the individual pixels to the display.

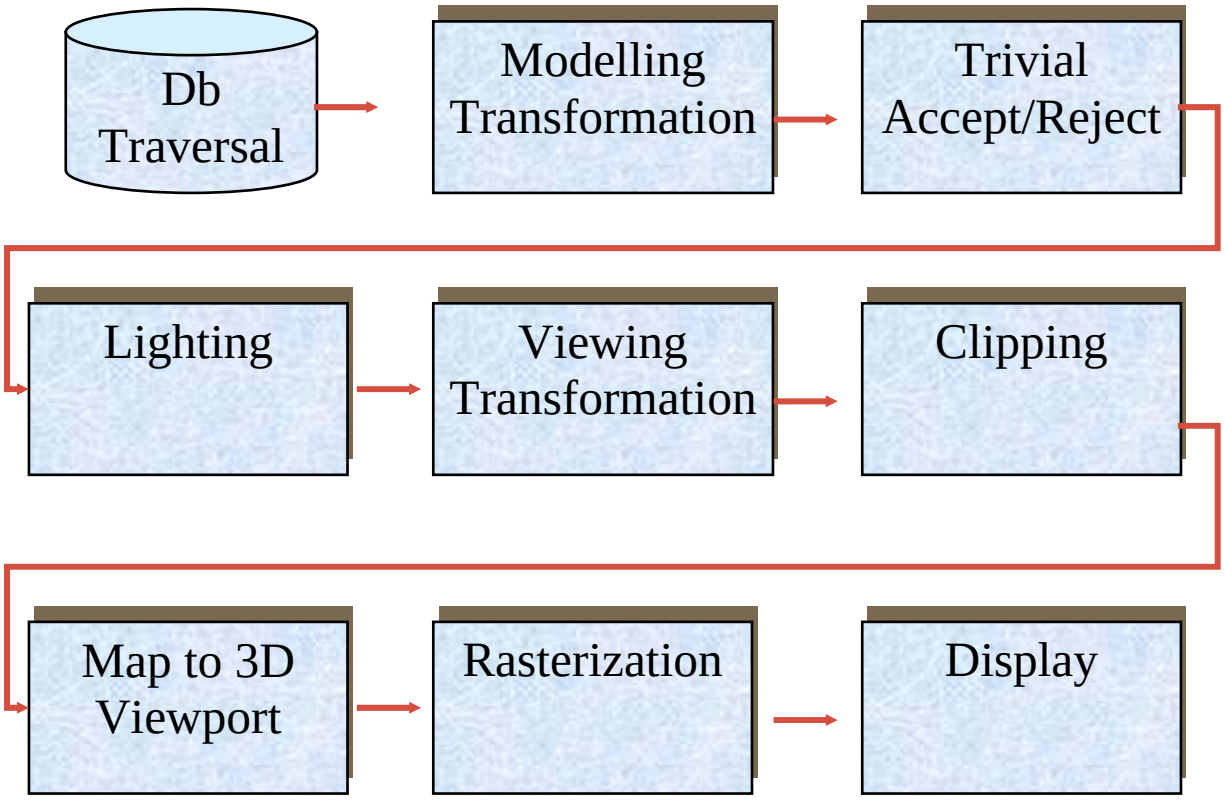

Figure 8. A typical pipeline for Gouraud lighting.

You will read about rendering engines that perform the lighting functions. Modern engines are capable of being modified by knowledgeable programmers to suite the mood desired. One such engine is the DirectX 9 application from Microsoft. For most multimedia work where super realism is not a priority we do not need such knowledge and we accept what is available from the 3D modelling package we are using. If you want to experiment with 3D graphics, you can down load many trial packages. One good clone of the commercial package, 3DSMax, is GMAX from Discrete.

Another approach to modelling, the Procedural Modelling makes use of algorithms rather then equations. This approach had gained a lot of attention since the introduction of the Fractals, which could be defined as an image in self similarity reign. Thus every section of the image is similar to the total image. A

known example is clouds, where a portion of a cloud looks like a full cloud, and a portion of that portion looks again like a cloud and so ad infinitum. Another example can be found in Indian temples were a part of a temple has the same design of the whole temple (Jackson, n.d.).
For a definition and examples of fractals, see (Lorenz 2003)

See the Tools section at the end of this lecture for programs that implement these ideas.

Links and References

Bezier, Anon An interactive Java Bezier applet [Internet]
http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCExamples/Bezier/bezier.html,
(Accessed: 15th September 2008)

Carlson W.E. (2003)
CGI Historical Timeline [Internet]
http://accad.osu.edu/~waynec/history/timeline.html
http://web.archive.org/web/20070519144440/accad.osu.edu/~waynec/history/timeline.html
A timeline history of graphics
(Accessed: 1st May 2007)

Formats, Anon [Internet]
http://www.why-not.com/articles/formats.htm#GIF,
A look at graphics file formats and a link to the file format specifications
http://www.dcs.ed.ac.uk/home/mxr/gfx/2d-hi.html
(Accessed: 15th September 2008)

Jackson W.J. (n.d.) Hindu temples and fractals [Internet]
http://liberalarts.iupui.edu/~wijackso/tempfrac/
(Accessed: 15th September 2008)

Lorenz W.E. (2003) Fractals and Fractal Architecture [Internet]
http://www.iemar.tuwien.ac.at/fractal_architecture/subpages/01Introduction.html (Accessed: 15th September 2008)
A definition, math and examples of fractals.

Molhave T. (2000) Transformations in 3D [Internet]
http://home10.inet.tele.dk/moelhave/tutors/3d/transformations/transformations.html
(Accessed: 15th September 2008)

Posters, Anon [Internet]
http://www.edwardtufte.com/tufte/posters . This site contains the description of the image and what it conveys. For a better sized display see, http://www.edwardtufte.com/tufte/minard
(Accessed: 15th September 2008)

Shoaff W. (2000), A short history of computer graphics [Internet]
http://www.cs.fit.edu/~wds/classes/graphics/History/history/history.html
(Accessed: 15th September 2008)

Tufte, E. R. (2001),
The Visual Display of Quantitative Information (2nd edition), Graphic Press

Vector, Anon [Internet] http://www.cca.org/vector/
A discussion of vector graphics systems
(Accessed: 15th September 2008)

Tools:

***Vector Drawing 2D*:**

Commercial: Free hand, Illustrator
Freeware:
Sodipodi: http://www.sodipodi.com/index.php3?section=home/about
InkSpace: http://www.inkscape.org/ .
This is a spin off Sodipodi and a SVG editor. We will discuss SVG in another seminar.

***3D Graphics*:**

A selection of Graphics books is available at:
http://www.ecs.soton.ac.uk/~msn/book/books.html
Two well known text books on Mathematical Computer Graphics are: Foley, J.D. et al (1995) Computer Graphics: Principles and Practice in C (2nd Edition), Addison-Welsley Heran, D. and Baker, M. P.(2003), , Computer Graphics with OpenGL (3rd Edition), Prentice Hall
Commercial: 3Dmax
Freeware:
Art of Illusion: http://www.artofillusion.org/
Gmax: http://www.turbosquid.com/gmax
Blender: http://www.blender3d.com/

Fractals 2D: A superb fractals program for Windows and DOS (the DOS version is more extensive!):
http://spanky.triumf.ca/www/fractint/fractint.html

Fractals 3D: MojoWorld Transporter
http://www.pandromeda.com/page/products/transporter.html

Bitmap 2D:

Are listed in next week seminar.