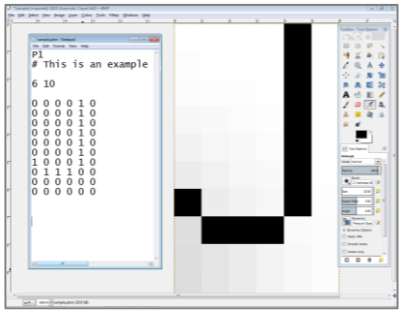
Practical activities to introduce file formats, file structure and contents. An exploration of how to write an image file using a text editor. An investigation into image compression techniques.

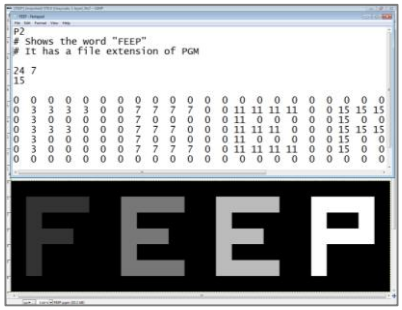
Preparation required:
GIMP installed on all computers. Sample file resources available in a shared repository.

Writing and Rendering Image Files

In this section we use GIMP (gimp.org) – a free graphics editor very similar in capabilities to Photoshop. In the presentation, we open plain text files (.txt) and save them with a new extension. We can then render them using GIMP. We use the NETPBM format: several file formats that allow a bitmap to be written in plain text. They use file extensions .pbm, .pgm and .ppm. Sample files are included in the resources.

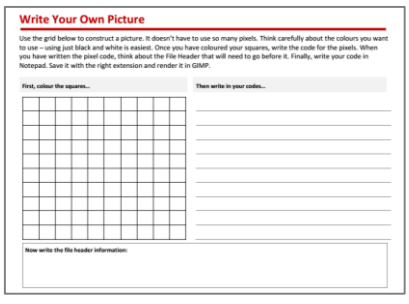


Using Notepad, open the file 'sample_letter.txt' from the Text Files folder. Notice the layout of the characters highlighted. These represent pixel values 0 for white, 1 for black. Save the file with a .pbm extension. GIMP, and some other image editors, will be able to render the pixel information. The pbm format renders in black and white only. Open the .pbm file in GIMP and zoom in. It is only 60 pixels so is very small. Looking at the file in Notepad, as well as the pixel values, there is other information at the top. Ask pupils what they think this is for. The image editor will ignore comments when rendering the file. The other details make up the file header – essential information for rendering the picture to the right dimensions – 6 pixels per row, and 10 rows. P1 is header detail that tells the computer to render each ascii value (1 and 0) as black and white. You'll find more about the file specification (and example files) at goo.gl/VKSASv. Notice that the pixel values do not need to be arranged as they are displayed, as the rendering information (6 by 10) is contained in the file header. To make the point, ask pupils to open 'diamond_v1.txt' in Notepad and predict what will be displayed. Then ask them to predict 'diamond_v2.txt'. It is the same data, and will render the same because of the header information.



Open 'feep_sample.txt' in Notepad, and study it. Notice it has different header information. P2 specifies a different way to interpret the data that follows. Ask pupils to study it and predict what will be displayed. Notice it has an extra number, 15 in the file header – what might that be related to? Only display the rendered image once they have made predictions. Save the file, but this time with a file extension .pgm. This will render as a grayscale image, with 16 values of grey. Pupils should experiment by changing the maximum grey value, and also changing the file extension to .pbm and observe what happens.

Finally, pupils should open 'six_coloured_pixels.txt', study it before saving with the extension .ppm. They could use a colour picker utility to predict how it would render.


Write Your Own Picture

Use the grid below to construct a picture. It doesn't have to use so many pixels. Think carefully about the colours you want to use – using just black and white is easiest. Once you have coloured your squares, write the code for the pixels. When you have written the pixel code, think about the File Header that will need to go before it. Finally, write your code in Notepad. Save it with the right extension and render it in GIMP.

First, colour the squares...

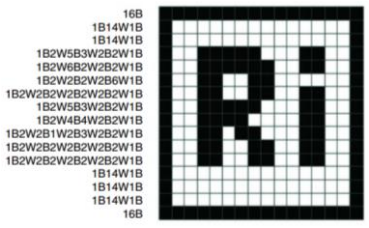
Then write in your codes...

Now write the file header information:

A class activity could be to write their own picture. Easier and hard version are included, a 10 by 10 grid being challenging enough for most. More able could attempt full colour images. The key point to stress is that when looking at bitmaps previously, pixels were rendered according to their binary value. Here the image is rendered according to ascii characters typed in Notepad. The notion of file header information has also been introduced. Further investigation of file sizes using, say Paint will reveal image files are larger than just the sum of pixel values due to the data in the file header.

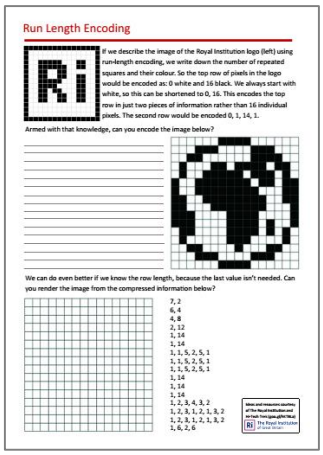
Compressing Images

Lempel-Ziv compression looked for repeated patterns in the text. The same approach can be applied to pictures too. Often a picture will have a continuous run of the same colour pixels. Why store each value repeatedly? This introduces the idea of run length encoding. It reinforces the point that all data is a collection of bit patterns. How those are interpreted determines what is displayed.



Display the example of run length encoding and ensure pupils understand how it works. Then ask whether it could be reduced further. Is there any need to store B & W. Could we just store the 'run' size? If we did, how would we know which was first? A protocol would be required, usually starting with a white run. The following slide gives the same example, with no reference to W or B, but notice how each row starts with a 0 run of white pixels. An online interactive (goo.gl/bvMxDS) from the CS Field Guide lets students build a picture and generate the run length coding. It gives the comparison between the raw bitmap size and the run length encoding. Could it be reduced further? Yes – the final value isn't required, so long as we know the length of the row.

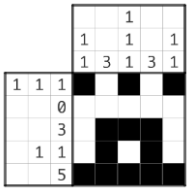
Using the slide, the class can work through the first part of the exercise. Note rows 6 to 11 MUST start with a 0 run of white. Note also that the final run on each row isn't included. For this, the header would need to store the row length once. The uncompressed file requires 256 bits. The run length version would require 86 numbers to be stored – saving around 2/3rd of the space. This is the simple explanation, adequate at KS3. However, some students may point out that this is flawed. Can a number be stored in 1 bit? As the biggest number required is 8 (1000 in binary), 4 bits per number are required, so the total file size would be 344 bits – an increase of nearly 40%! Compression algorithms might not always result in a smaller file size! If students have pointed this out, encourage them to think about the sort of files that would compress well, and those that would not. A plain white image would be a 'best case'. What would be a worst case?



Lossless and Lossy Compression

An image compressed by run length encoding can be restored to its original form. It is an example of lossless compression. Lossy compression techniques, on the other hand, rely on the removal of detail to shrink a file. Compression methods such as JPEG reduce the size without impacting on the image unnecessarily. In the example, the uncompressed image (left) is compared to two smaller versions. Both reduce the file to a third of its size. The middle one uses lossy techniques – the difference being imperceptible. The right hand version achieves the reduction by reducing the ***colour depth***. Instead of 24 bits per pixel, it uses 8 bits, thus reducing the number of colours it can represent to 256. Reducing the number of bits (the colour depth) is so crude it isn't compression, just a low quality representation but it allows the notion of colour depth to be explored.

Methods like JPEG and PNG take advantage of patterns in an image to get good size reduction without losing more quality than necessary. Once compressed though, the original cannot be recovered. The example comes from an excellent chapter in the CS Field Guide, recommended as further reading: goo.gl/GEBfUU. For those who are time pressed, CS4Fn also has a couple of good articles which would make good homework reading for children. The link (goo.gl/OfB0tI) takes you to an accessible explanation of JPEG. Having read the article, follow the link at the end to a further explanation of MPEG video compression.



A different method for rendering an image is also introduced at the end. The presentation steps through how to logically deduce the image from the information given. As well as encouraging logical thinking (an exercise is included), the key point to draw out is that there are lots of ways in which data can represent a picture. Software that renders images, such as GIMP must be able to interpret the various file formats.