## Techniques in implementation.

1. Load and save:  
I slightly modified the loading and saving function. It now load the image form \image\load and save to \image\saved. The path is hardcoded.  
  
2. To\_Grayscale()  
Just iterating through each pixel and turn RGBA to grayscale. Index of R in RGBA: (. Notice that I defined “*RGBtoGray()*” function for future convenience.  
一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

3. Quant\_Uniform()  
As it requires, we need 8, 8 ,4 shades for R, G, B, which means they need 3, 3, 2 bits to stored. for 8-bit original data, right shift 5, 5, 6 for R, G, B and left shift back for visualization.  
  
4. Quant\_Populority()  
The algorithm is clear. Some implementation details is: for 32 shades of each channel, we can represent a color as 32-base number and convert it to 10-base integer to store. For example, is . this makes it suitable for array.  
  
5. Dither\_Threshold()  
Just call “*RGBtoGray()*” and apply the threshold of 128(256 \* 0.5).   
  
6. Dither\_Random()  
Just add a random multiplier between [-0.2, 0.2].  
  
7. Dither\_FS()  
Just implement the algorithm.  
  
8. Dither\_Bright()  
Store grayscale values of each pixel in reversed order. The thresh hold is the grayscale value on the index ((avgBrightness / 256.0) \* (width \* height)).   
  
9. Dither\_Cluster()  
 Just apply the threshold of **ditherMatrix**[i % 4][j % 4].  
  
10. Filtersystem  
def “**enum paddingType {ZERO\_PADDING = 1, MIRROR\_PADDING = 2}**;”  
and implement **bool applyFilter(const std::vector<std::vector<float>>& filter, int filterSize, paddingType padding, unsigned char origData[],int width, int height)**then the resting is to make the corresponding filter.  
10. Resize(float scale)  
While the new size is (height \* scale, width \* scale), for each pixel in the new image data, we find (i / scale, j / scale)[backward mapping] and apply filter on the point. The filter applied is described in the previous problem statement.   
  
These are all operation I implemented.