1. A To display the image undistorted and to just fully fit within a square OpenGL window by mapping it onto a rectangle, we note that the left and right sides would touch the edge of the OpenGL window. The x value of the 4 corners would be either 1 or -1. We use the width to height ratio of the image to determine the y coordinate of the 4 corners. Since the 3648 pixels fit within 2 units of the OpenGL window, we compare the ratios to get

$$\frac{2437}{3648} = \frac{z}{2}$$

Solving for z, which represents the height of the triangle, we note that it is equal to 4/3. If we center the rectangle, the (x,y) normalized device coordinates of the four corners would be

$$(-1, 2/3), (-1, -2/3), (1, 2/3), (1, -2/3)$$

Hilroy

B. To determine the coordinates after a 30° counterclockwise rotation around the center, we first note the distance of each point to the center of rectangle remains constant as we rotate.

We use Pythagorean Theorem to find length from center of each vertex.

$$l = \sqrt{(1)^2 + (4/6)^2}$$
$$= \sqrt{13}/3$$
$$\approx 1.20$$

We then use arctan to find that the angles associated with each vertex are approximately 33.7°, 146.3°, 213.7° and 326.3°.

A transformation of 30° counterclockwise rotation means adding 30° to each of these value, giving new angles of

63.7°, 176.3°, 243.7° and 356.3°.

We now use these angles and the lengths (hypotenuse) to find the associated x and y coordinates using sin and cos. Doing this, we get the new coordinates

(.533, 1.077), (-1.199, .077), (-.533, -1.077), (1.199, -.077)

2. A. To achieve the same perceived intensity by the average human observer, we note the following value corresponding to the relative sensitivity of the human visual system to light stimulus of blue, green and red light from the provided graph.

Red (620 nm)	:	0.375	$\bar{y}(\lambda)$
Green (530 nm)	:	0.860	$\bar{y}(\lambda)$
Blue (470 nm)	:	0.090	$\bar{y}(\lambda)$

The ratio of "pure red" to "pure green" needed is therefore around $0.860 / 0.375 = 172 / 75$.
That is, 172 parts of red to 75 parts of green result in same perceived intensity.

The ratio of "pure green" to "pure blue" needed is around $0.090 / 0.860 = 9 / 86$.
Around 9 parts of green to 86 parts of blue result in same perceived intensity.

B. If an image encoded in RGB with the above primary colours, then a reasonable function to convert to greyscale would take into account the relative sensitivity of human perception. We would need to keep the ratios in part 2A the same. Adding $0.375 + 0.860 + 0.090$, we get 1.325. Dividing each value by 1.325, we get the function

$$L = 75/53\,R + 172/265\,G + 18/265\,B$$
$$\approx 0.283\,R + 0.649\,G + 0.0679\,B$$

Hilroy

3. A. To display the photo in 1A to fit in a 1000 x 1000 pixel window, there would be more image pixels for every display pixel, since the original image is larger.

We thus have to fit a 3648 x 2432 pixel wide image into 1000 x 667 pixels to not distort. This means that the width and height of original image have 3648/1000 = 3.648 times more pixels. Fitting it to 1000 x 1000 window means that 3.648 x 3.648 = 13.30794 image pixels fit into every display pixel.

B. The above discrepancy does affect the output of the Sobel edge filter on the program. That is, if we resample the same photo to a size of 1000 x 667 pixels, then apply the Sobel edge filter to it, this means that we are applying Sobel to an image that has lost some of the original data. For instance consider a black edge of pixels with white on both side:

White	Black	White
White	Black	White
White	Black	White
White	Black	White
White	Black	White
White	Black	White
White	Black	White

By resampling, these pixels are collapsed, thus losing an edge which was originally present. Sobel would then not be able to detect an edge which was orginally present.

If we instead apply the Sobel filter to the orginal image of 3648 x 2432 pixels, Sobel will be able to detect an edge. Hilroy

If we then display it on the OpenGL window, we would know that it would be scaled to fit the window after the sobel filter has been applied. Depending on the image originally used, this may produce an image that differs from the one obtained by rescaling first then using sobel.

For instance, if we applied the above effects to an image composed of pixels all of value 0, we would see no difference at all. If we applied the effects to a more complex image however, we may notice minute differences due to the application of Sobel before fitting to OpenGL window versus resampling to a size of 1000x 667 before applying Sobel.

C. By applying the unsharp mask filter twice on an image, the effect is enhanced. When using the unsharp mask filter on the image the second time, the effect is applied on top of the already modified image, thus compounding the first application of the unsharp mask.

A 2D convolution kernel that would create the same effect as applying the 3x3 unsharp mask kernel twice could be found by noting that convolution is associative. We can therefore apply the convolution kernel to itself.

We recall that convolution is the process of multiplying each element of the image with its local neighbours weighted by the kernel. To accomplish this, we flip both the rows and columns of kernel, multiply locationally similar entries and take the sum.

We note that after flipping the rows and columns of unsharp mask, we still have the unsharp mask. We then pad around the image (of unsharp mask) with 0's and apply convolution.

Thus, we have $\left(\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}\right) =$

$$\begin{bmatrix} 2 & -10 & 2 \\ -10 & 29 & -10 \\ 2 & -10 & 2 \end{bmatrix}$$

We note for instance that entry

$$[0,0] = (0\times5) + (-1\times-1) + (5\times0) + (-1\times-1) = 2$$
$$[2,1] = (0\times-1) + (-1\times0) + (5\times-1) + (-1\times0) + (0\times-1) + (-1\times5) = -10$$
$$[1,1] = (0\times0) + (-1\times-1) + (0\times0) + (-1\times-1) + (5\times5) + (-1\times-1) + (0\times0) + (-1\times-1) + (0\times0)$$
$$= 29$$

with symmetric entries in the other locations. Thus, $\begin{bmatrix} 2 & -10 & 2 \\ -10 & 29 & -10 \\ 2 & -10 & 2 \end{bmatrix}$

is the 2D convolution kernel that would apply the same effect as applying the 3×3 unsharp mask kernel twice. If we do not remove the edge data and considered the image padded by an edge of 0's, then we note we would obtain instead the 5×5 kernel

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & -10 & 2 & 0 \\ 1 & -10 & 29 & -10 & 1 \\ 0 & 2 & -10 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

4. A. At $\sigma = 1.4$, the Gaussian Function $G_7(x)$ evaluates to the following for the select values of x

x	$G(x)$
0	0.284959
1	0.220797
2	0.102713
3	0.0286865

B. To find the value of σ that was used to generate the 5-point Gaussian kernel, we note that at $x=0$, $G_7(0) = 0.4$. Solving for σ, we get $\pm 5/(2\sqrt{2\pi})$, which is approximately $\approx \pm 0.997$

We note that at $x=0$, $G(0) = 0.4$. Thus, the equation becomes

$$0.4 = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{0}{2\sigma^2}}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}}$$

$$2/5 = \frac{1}{\sqrt{2\pi\sigma^2}}$$

$$\sqrt{2\pi\sigma^2} = 5/2$$

$$2\pi\sigma^2 = 25/4$$

$$\sigma^2 = \frac{25}{8\pi}$$

$$\sigma = \pm \sqrt{\frac{25}{8\pi}}$$

$$= \pm \frac{5}{2\sqrt{2\pi}}$$

Hilroy

c. To describe the procedure for creating a one-dimensional discrete n-point Gaussian kernel, we note that as σ increases, the bell curve flattens. Thus, as n increases, our value of σ would similarly have to increase so that the discrete Gaussian kernel accurately approximates the continuous Gaussian kernel.

We know from question 4A, that comparing those results with the approximation of 7-point discrete Gaussian in Part IV of programming assignment, $\sigma = 1.4$ corresponds to 7 point discrete Gaussian. From 4B, we know $\sigma \approx 1$ corresponds to 5 point discrete Gaussian. By solving for σ of 3-point Gaussian kernel of Part IV by letting $x=0$, $G(x) = 0.6$, we get $\sigma = 5/(3\sqrt{2\pi}) \approx 0.66$.

From these values, we note that for an n-point Gaussian kernel, the σ is around $(1/5)n$. This choice of σ provides an appropriate Gaussian kernel of the specified size, since it seems to provide discrete points for which the continuous curve closely approximates. By using this method, and taking the discrete value to (1 significant digit for least/greatest x, and 2 significant digits for all other x with G(0) equally whatever is left), we

± 4 in this case

get the following value for a 9-point Gaussian kernel.

We use $\sigma = 1.0$ to get

x	-4	-3	-2	-1	0	1	2	3	4
G(x)	0.02	0.055	0.12	0.19	.23	0.19	0.12	0.055	0.02

Thus, the general procedure would be to find σ ($\frac{1}{5}$ of n), then use this to calculate G(x) for $x = -(n-1)/2$ to $(n-1)/2$ with G(x) for least/greatest x being to 1 sig dig, the rest being 2 sig dig, and G(0) being what is left to total 1 (unity).