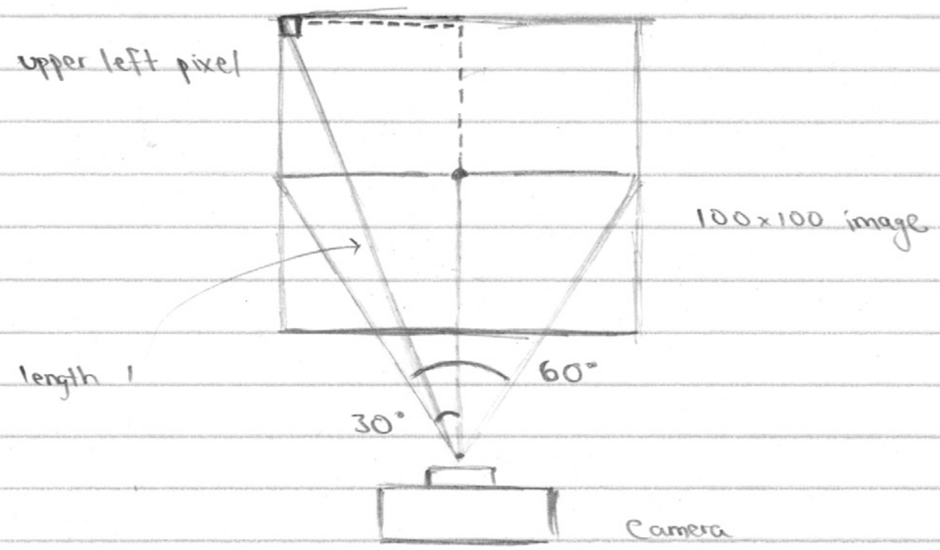
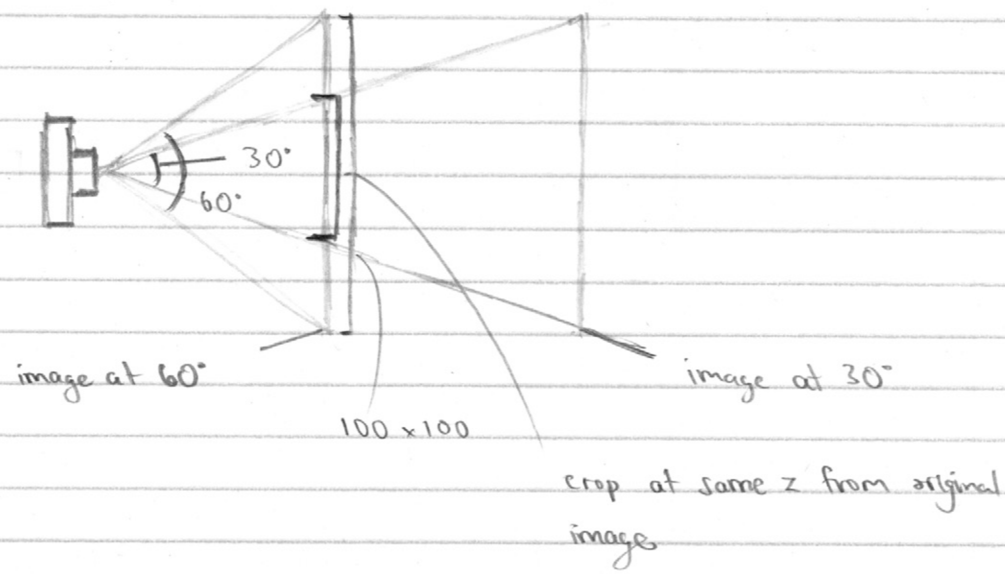
1. A. To determine the normalized device coordinates of upper left pixel, we first note that it is 50 pixels upwards and 50 pixels to the left from the centre. This can be seen in the below image



Thus, the x coordinate is -50, and the y coordinate is 50. We note that coordinates are measured in pixels. To determine the z coordinate, we note that $\tan(30°) = 50/z$, so $z = 50/\tan(30°) = 50\sqrt{3}$. Since away from camera, we take this to be negative z direction, so upper left pixel is $(-50, 50, -50\sqrt{3})$. We want to normalise this, so we divide each component by magnitude, which is $\sqrt{(-50)^2 + (50)^2 + (50\sqrt{3})^2} = 50\sqrt{5}$. Therefore, the normalized coordinates become $(-1/\sqrt{5}, 1/\sqrt{5}, -\sqrt{3/5})$. The ray is therefore $r(t) = (0,0,0) + t(-1/\sqrt{5}, 1/\sqrt{5}, -\sqrt{3/5})$

B. It is possible to crop the image from 1A to form a result that looks as if it were rendered with a 30° field of view. But since this is a cropped image, the result (what we see of original image that is cropped) is not the original image. It would however, look as if this image was rendered with a 30° field of view. To determine how much to crop to achieve this effect, we simply determine the ray to the upper left if the original image was rendered in a 30° field of view. Similar to 1A, we know x coordinate is -50, y coordinate is 50, and z coordinate is now $-50/\tan(15°)$ ← 15 because $\frac{1}{2}$ of 30°. This is $-50/(2-\sqrt{3})$. To determine how much to crop, we need to determine x and y coordinates when we scale the z to match that of original image.



z at 30° is $-50/(2-\sqrt{3})$, while z at 60° is $-\sqrt{3/5}$. Thus, multiply all coordinates by $\left(\frac{-\sqrt{3/5}}{\left(\frac{-50}{(2-\sqrt{3})}\right)}\right)$. Thus, z becomes $-\sqrt{3/5}$, while x becomes ≈ -0.21 and $y \approx 0.21$. To summarize

Uncropped normalized 60° field of view = $(-0.45, 0.45, -0.77)$

Uncropped 30° field of view = cropped = $(-0.21, 0.21, -0.77)$

Since corners are symmetric, this means that to crop image from 1A so that the result appears as if the result has been rendered at 30° field of view, crop so that the xy vertices are now at $(-0.21, 0.21)$, $(0.21, 0.21)$, $(0.21, -0.21)$ and $(-0.21, -0.21)$ from the original $(-0.45, 0.45)$, $(0.45, 0.45)$, $(0.45, -0.45)$, $(-0.45, -0.45)$

However, it is not possible to crop image so that the result is the original image rendered at 30°. This is because the cropped result would have original edges missing, while original rendered at 30° would be the original image scaled down to fit.

2 A. Since the unit sphere is centred around $(0,0,-3)$, the equation for the sphere is $\|p-(0,0,-3)\| = 1$. To find the intersection, we apply the formula $(o+td-c)\cdot(o+td-c) - R^2 = 0$ where the ray $o+td$ is $(0,0,0)+t(0,1,-4)$ and the sphere $\|p-c\| = R$ is given above.

Substituting values, we get

$$((0,0,0)+t(0,1,-4)-(0,0,-3))\cdot((0,0,0)+t(0,1,-4)-(0,0,-3)) - 1^2 = 0$$
$$(0,t,-4t+3)\cdot(0,t,-4t+3) - 1 = 0$$
$$0 + t^2 + 16t^2 - 24t + 9 - 1 = 0$$
$$17t^2 - 24t + 8 = 0$$

Solving for t gives $(12 \pm 2\sqrt{2})/17 \approx 0.540,\ 0.872$. Substituting this t into the ray, we get $(0, (12-2\sqrt{2})/17, -4(12-2\sqrt{2})/17) \approx (0, 0.540, -2.16)$ and $(0, (12+2\sqrt{2})/17, -4(12+2\sqrt{2})/17) \approx (0, 0.872, -3.49)$

The point nearest the camera at $(0,0,0)$ is $(0, 0.540, -2.16)$ which can be determined using the distance formula.

B. To find normal vector, we note that for points P, Q, R forming vertices of triangle, vectors $\overrightarrow{PQ}$ and $\overrightarrow{PR}$ are on plane. That is, $(2,3,-5)-(2,-1,-3) = (0,4,-2)$ and $(-1,-1,-3) - (2,-1,-3) = (-3,0,0)$. The normal vector is given by the cross product of the two vectors above which is $(0,6,12)$.

C. To find the position of intersection between the ray and the plane in 2B, we apply formula with ray $o+td = (0,0,0)+t(0,1,-4)$ and plane $(p-q)\cdot n = (p-(2,-1,-3))\cdot(0,6,12)$ to find the intersection of $(o+td-q)\cdot n = 0$.

Substituting values, we get

$((0,0,0) + t(0,1,-4) - (2,-1,-3)) \cdot (0,6,12) = 0$

$(-2, t+1, -4t+3) \cdot (0,6,12) = 0$

$6t + 6 - 48t + 36 = 0$

$42t = 42$

$t = 1$

Substituting this t into ray gives intersection at $(0, 1, -4)$.

D. To determine barycentric coordinates, we convert by using ray-triangle intersect formula. Since we already know the intersection with plane, we can substitute into formula $o + td = (1-u-v)p_0 + up_1 + vp_2$ to get

$(0, 1, -4) = (1-u-v)(2,-1,-3) + u(2,3,-5) + v(-1,-1,-3)$

$0 = 2 - \cancel{2u} - 2v + \cancel{2u} - v \rightarrow 3v = 2 \rightarrow v = 2/3$

$1 = -1 + u + \cancel{v} + 3u - \cancel{v} \rightarrow 4u = 2 \rightarrow u = 1/2$

$(1-u-v)$ is therefore $(1 - 1/2 - 2/3) = -1/6$. The barycentric coordinates are therefore $(-1/6, 1/2, 2/3)$. The camera ray does not intersect the triangle, since $-1/6$ is not between 0 and 1.

3\. A. We use the diffuse reflection equation $c = c_r c_l \max(0, n \cdot l)$. Since we want to determine angle at which the light reflected from the surface is exactly half of incident intensity, we write equation as $1/2 = n \cdot l$. The dot product $n \cdot l = \|n\| \|l\| \cos\theta$. Solving for θ gives $\theta = \cos^{-1}(1/(2\|n\| \|l\|))$. Taking $\|n\| = \|l\| = 1$, we get $\theta = \cos^{-1}(1/2) = \pi/3$ or $60°$.

B. We use the Phong lighting equation $c = c_l \max(0, e \cdot r)^p$. For angle with half incident intensity, it becomes $1/2 = e \cdot r^p$. Applying the dot product, this becomes $1/2 = (\|e\| \|r\| \cos\sigma)^p$. Thus, $(1/2)^{1/p} = \|e\| \|r\| \cos\sigma$, so $\sigma = \cos^{-1}((1/2)^{1/p} / (\|e\| \|r\|))$. The angle measured from surface normal is therefore $\theta + \sigma$ where

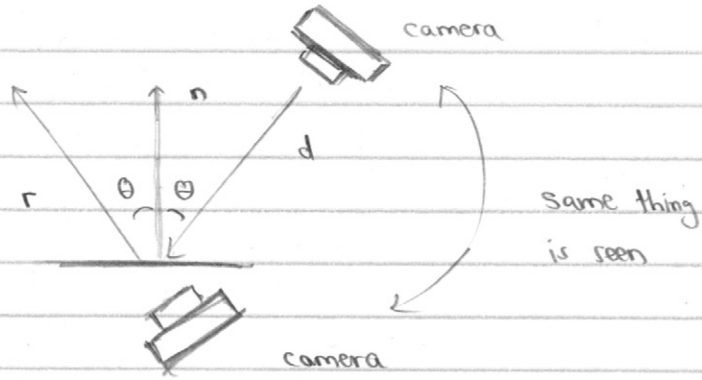
σ is given above. If we let $\theta = 25°$ and $\|e\| = \|r\| = 1$, then σ becomes $\cos^{-1}(((1/2))^{1/8}) \approx 23.51°$ and the angle from surface normal is $\theta + \sigma \approx 48.51°$.

4. A. The viewing or camera ray would be from the eye (or camera) to a point on the screen that corresponds to where an object may be. We can model this with 3D parametric line where **e** denotes the position of the eye and **s** denotes the point on the screen as $p(t) = e + t(s - e)$. We can then check using ray-object intersection on planes, spheres, triangles, when the view ray intersects, thus allowing us to create a ray tracing program that allows us to generate a viewing ray for a given pixel and find the intersection with an object. In an actual implementation, we may also want to return a reference to the object or its properties.

Shadow rays on the other hand, extend from the object being tested to a potential light source. If we are on point P on a surface being shaded, we look towards the direction of the light l. Intersection testing is done here, similar to the view ray, but now instead to determine whether there is an object blocking the light. If $p + tl$ hits an object, then it is in a shadow since light is blocked, otherwise it is not in a shadow since light is not blocked. If multiple light sources exist, we need to ensure that we send a shadow ray and evaluate the diffuse or phong terms for each light source. Additionally, we would take into account any ambient colour that would be present.

thus we do not add diffuse or specular contribution from light

B. Comparing how reflected rays would differ from view/camera rays described in detail above, we note that reflected rays are used to determine the light that is reflected from an object. Note that what is seen in direction d, is what is seen in direction r from the surface, as shown on the next page. Vector r can be found using the Phong equation, thus providing a means to account for the specular reflection.



in a given scene. Unlike view rays, when light reflects off an object, some energy may be lost. For instance, tinted glass reflects some colours more efficiently, so this changes the colour of the object it reflects. We can implement this in a recursive call (making sure it terminates in the case that the ray starts in a room for instance) so that we take into account all reflections of light off of different objects. We would recursively trace the reflected ray in the program, making sure to terminate after some depth limit and taking into account partial reflection by mixing colour from reflected ray with shaded colour of object.