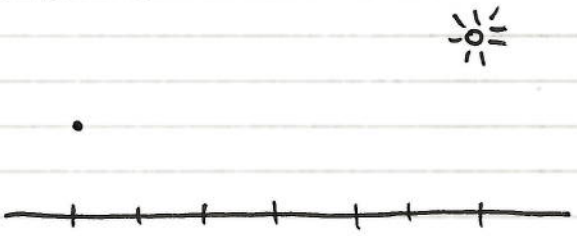
GRAPHICS 2015

1 a)



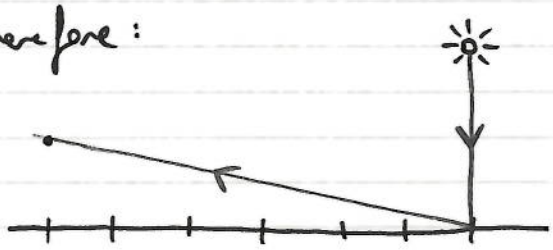
i) The plane only has a diffuse component:

$$I_{diff} = I_{in} \cdot K_{diff} \cdot \cos\phi$$

↳ for maximum $\cos\phi$, $\phi = 0$

$\vec{L}$ is parallel to $\vec{N}$

Therefore:



↑ this will be the point in the line that will appear brightest to the viewer.

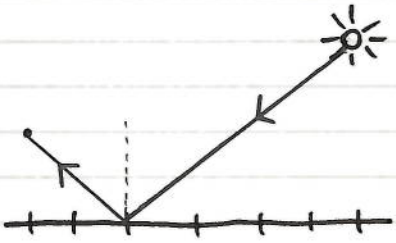
ii) The plane only has specular (reflective) component:

$$I_{spec} = I_{in} \cdot K_{spec} \cdot \cos^{n}\theta$$

↳ to maximise I_{spec}, must maximise $\cos\theta$

for max. $\cos\theta \Rightarrow \theta = 0$

which means the ideal reflection angle and viewing direction are aligned

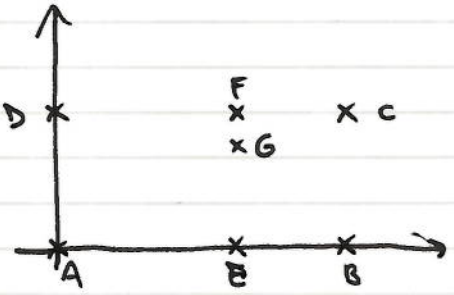


iii) Plane only has diffuse component and light attenuates with $1/d^2$:

$$I_{diff} = K_{diff} \cdot \cos\phi \cdot$$

1. b)

Vertex	Coordinates	Colour
A	(0, 0)	(250, 0, 0)
B	(10, 0)	(250, 250, 0)
C	(10, 5)	(0, 250, 0)
D	(0, 5)	(0, 0, 0)



i) $E\ (6, 0) = 0.6 \times B + 0.4 \times A$

ii) $F\ (6, 5) = 0.6 \times C + 0.4 \times D$

iii) $G\ (6, 4) = 0.8 \times F + 0.2 \times E$

iv) $RGB_E = 0.6 \cdot RGB_B + 0.4 \cdot RGB_A = (250, 150, 0)$

$RGB_F = 0.6 \cdot RGB_C + 0.4 \cdot RGB_D = (0, 150, 0)$

$RGB_G = 0.8 \cdot RGB_F + 0.2 \cdot RGB_E = \boxed{(50, 150, 0)}$

v)

1. c) The Blinn-Phong model uses the Phong model as a base, but performs the calculations differently so as to improve performance. Instead of having to calculate the reflection vectors, the Blinn variant uses the halfway vector $\vec{H}$, where

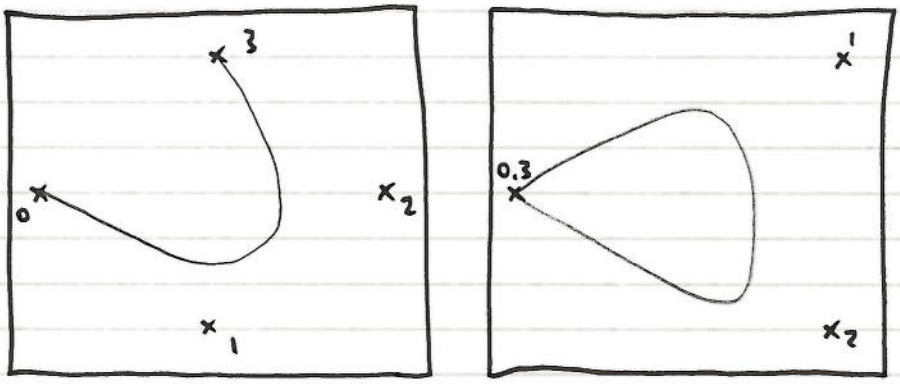
$$\vec{H} = \frac{\vec{V} + \vec{L}}{|\vec{V} + \vec{L}|}$$

$\vec{V}$ = normalized vector from point to viewer

$\vec{L}$ = normalized vector from point to light source.

This vector can now be used to calculate the specular coefficient. This is less computationally expensive and yields similar results.

2. a)



2. b) $$P(\mu) = \sum_{i=0}^{3} a_i(\mu) P_i$$

$$= (1-\mu)^3 P_0 + 3\mu(1-\mu)^2 P_1 + 3\mu^2(1-\mu) P_2 + \mu^3 P_3$$

2 c)

2. d) :)

3. a)

```
polygons → list
z_buffer [x, y] = -∞
result [x, y] = 0

begin
for polygon in polygons:
    for pixel (x,y) that intersects polygon:
        depth = z-depth of polygon at (x,y)
        if depth < z_buffer [x, y]:
            result [x,y] = intensity of polygon at (x,y)
            z_buffer [x,y] = depth

display result
end
```

3. b)

3. c) Super-sampling is an anti-aliasing method where you compute the picture at a higher resolution than the display area. These "super samples" are then averaged to find the final pixel value. This makes boundaries between polygons appear blurred, but uniform areas of colour are unaffected. This method works well for scenes made of filled polygons but at high computational cost. It also does not work for line drawings.

Convolution is another anti-aliasing method that is less computationally expensive and uses filters to blur the image. Each pixel is replaced by a local weighted average with the pixels around it (this is called a mask). The operation is fast, and can be parallelised and/or executed in hardware. However, this process degrades the image.

includes d)

3. e) i) alias effect ??

ii) obvious approach ???

4. a) orthographic projection $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

perspective projection $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.1 & 0 \end{pmatrix}$

identity $\begin{pmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{pmatrix}$

rotation $\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

uniform scaling $\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

non-uniform scaling $\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 \end{pmatrix}$

4. b) $ax + by + cz + d = 0 \Rightarrow \boxed{y = -y_0}$

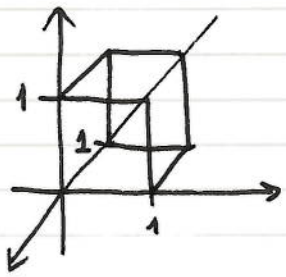
axis unmoved: $\vec{q} = \vec{i}$, $\vec{r} = \vec{j}$, $\vec{s} = \vec{k}$.

origin transformation: from $(0, 0, 0) \Rightarrow (0, 0, -2y_0)$

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2y_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. c) transformation of x axis:

4. d)



:) A perspective projection matrix can achieve this transformation:

$$M_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$$

ii)

$$M_p \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} =$$