1

a

i

[[1, 0, 0, -10],

[0, 1, 0, -10],

[0, 0, 1, 0],

[0, 0, 0, 1]]

[[0, 0, -1, 0],

[, , 0, 0],

[, , 0, ],

[0, 0, 0, 1]]

Got the answer below:

[[0, 0, 1, 0],

[, , 0, 0],

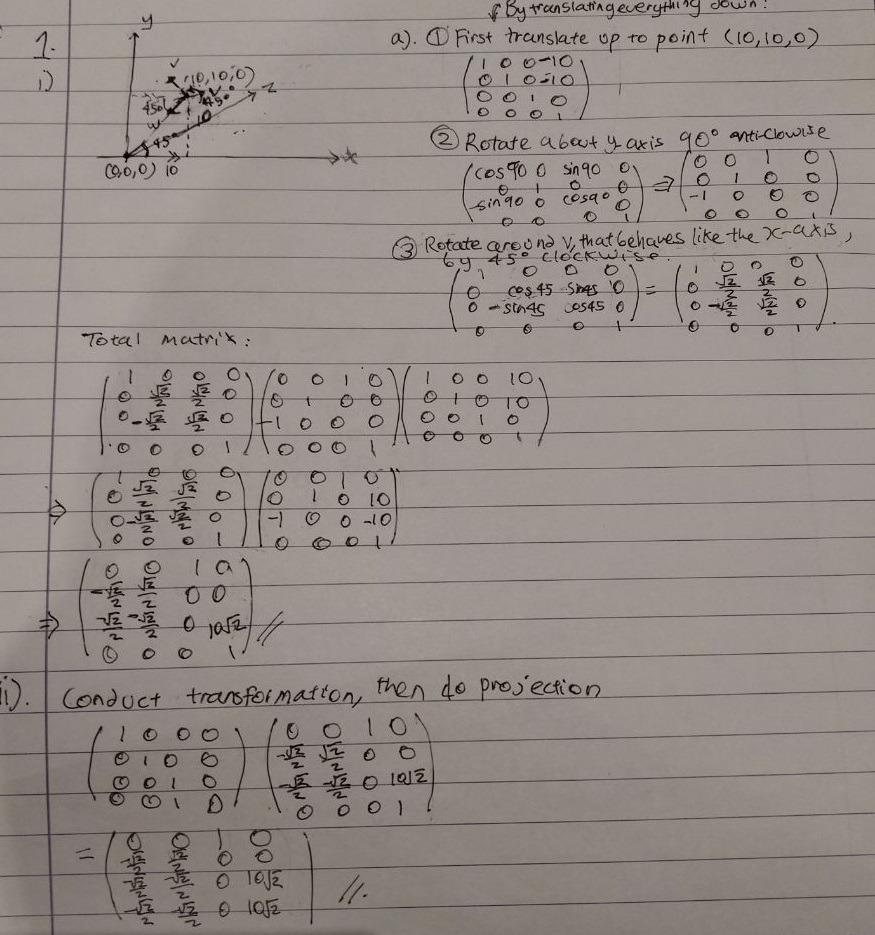
[, , 0, ],

[0, 0, 0, 1]]

**Another alternative answer:**

The camera sits at (10, 10, 0) pointing to the origin, in world space, so we want to make this vector sit at the origin, pointing down the z axis.

This is a 3 step process:

1) Translate by (-10, -10, 0)

2) Rotate in Z-axis clockwise by 45deg

3) Rotate in Y-axis clockwise by 90deg

Matrices of these three transformation:

1)

[[1, 0, 0, -10],

[0, 1, 0, -10],

[0, 0, 1, 0],

[0, 0, 0, 1]]

2)

[[cos45, sin45, 0, 0],

[-sin45, cos45, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]]

3)

[[cos90, sin90, 0, 0],

[0, 1, 0, 0],

[-sin90, 0, cos90, 0],

[0, 0, 0, 1]]

Total transformation: multiply (3)(2)(1)

[[0, 0, 1, 0],

[, , 0, 0],

[0, , 0, ],

[0, , 0, ]]

ii

[[1, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 1, 0]]

b

i)

We calculate the transformation of moving the viewer to the origin and then do the projection.

[[1, 0, 0, 0],

[0, 1, 0, 0], \*

[0, 0, 1, 0],

[0, 0, 1, 0]]

[[1, 0, 0, -10],

[0, 1, 0, -10],

[0, 0, 1, 0],

[0, 0, 0, 1]]

=

[[1, 0, 0, -10],

[0, 1, 0, -10],

[0, 0, 1, 0],

[0, 0, 1, 0]]

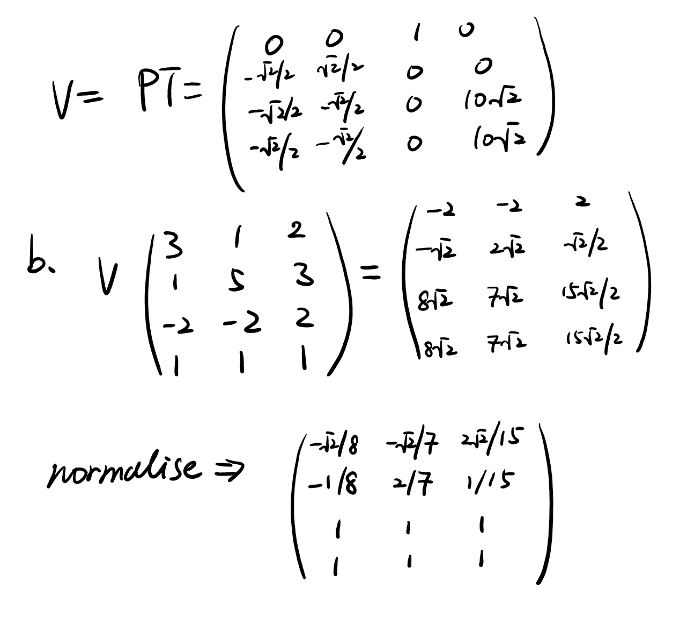
multiply the above matrix to P1, P2 and P3

P1 is (3.5, 4.5, 1) ^ T

P2 is (4.5, 2.5, 1) ^ T

P3 is (-4, -3.5, 1) ^ T

Using the hand written answer from above, it gives



ii)

Calculate the cross product of P1 - P2 and P2 - P1

And then use the dot product to check for the direction. Invert it if necessary.

c

i)

see slide

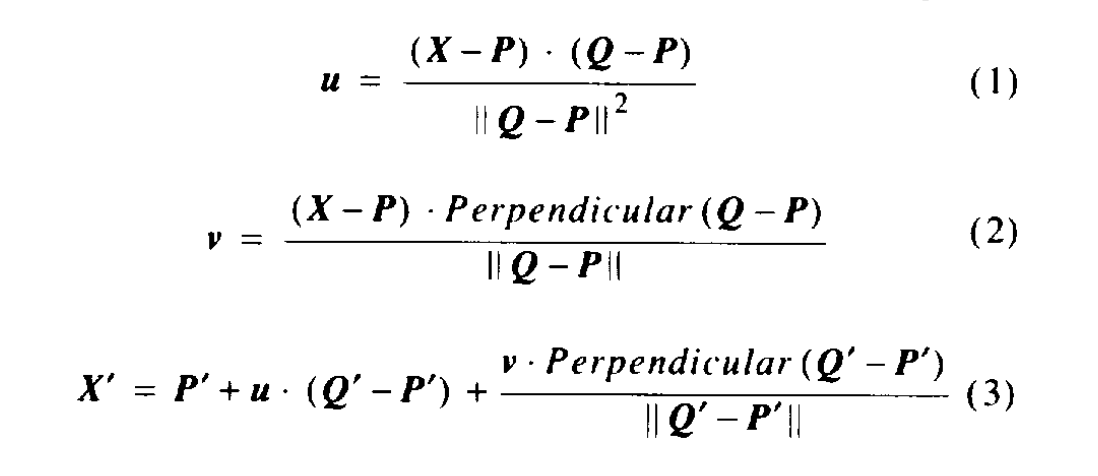
ii)

there should be 3 value for each vertex since the light vector change slightly

8

2a)

i)



ii)

Do we need to prove this explicitly?

Otherwise, to explain it with words:

Translation is achieved by offsetting control line. This will move all coordinates by a point.

Rotation is achieved by turning the control line, by an angle. This will rotate all points around the line.

b)

Two bigger arrows (yes it must be two, scaling only occurs in one dimension)

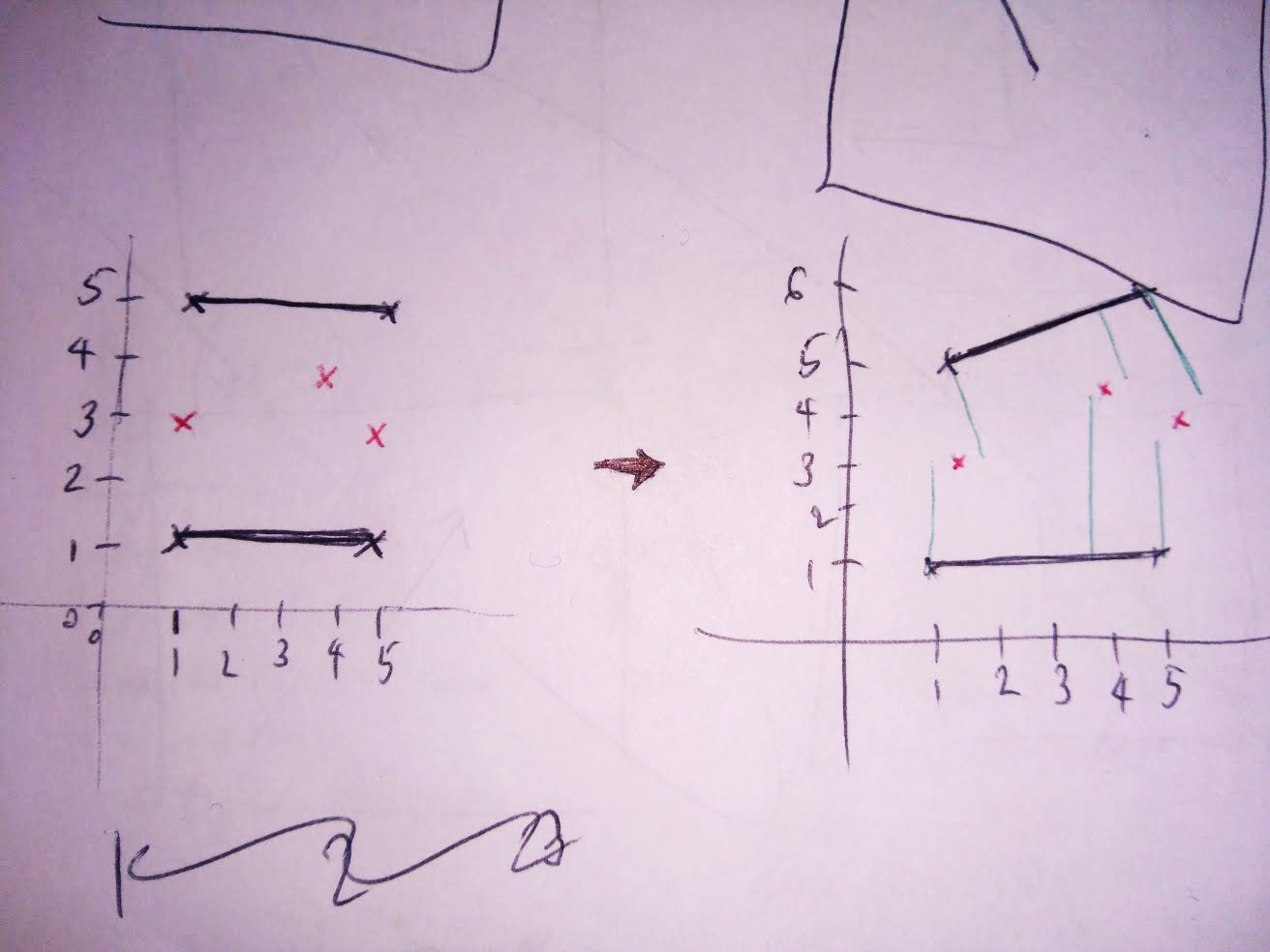
Rotated arrow, normalized

one arrow, flipped in y=0

One arrow, rotated left and scaled up

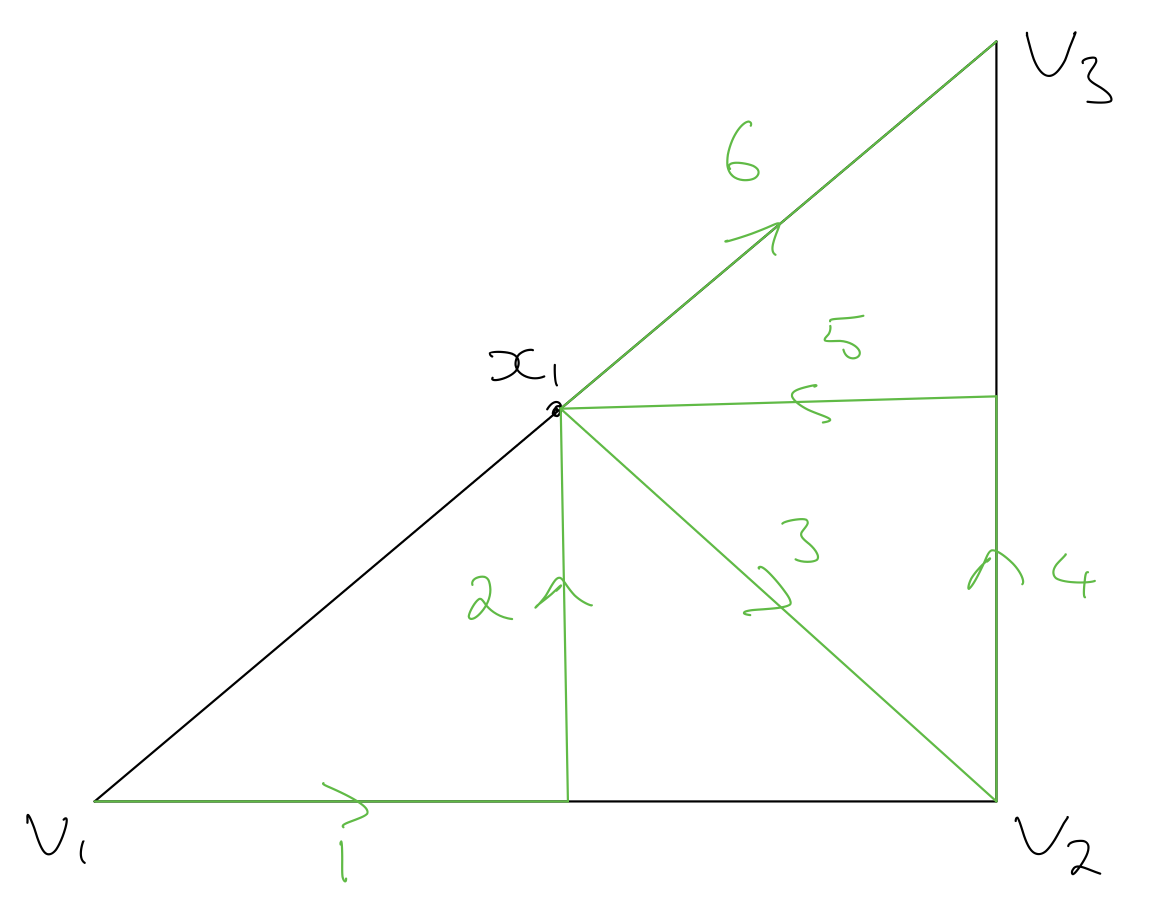
c)

Assuming we can do the whole thing geometrically (i.e. all of u’, v’, u, v, not just u, v) looks something like this



3

a



Non-optimal (from tutorial 4 2019)

b

trilinear and barycentric

ii)

trilinear: use the relative distance to the side

barycentric: use affine combination of vertex

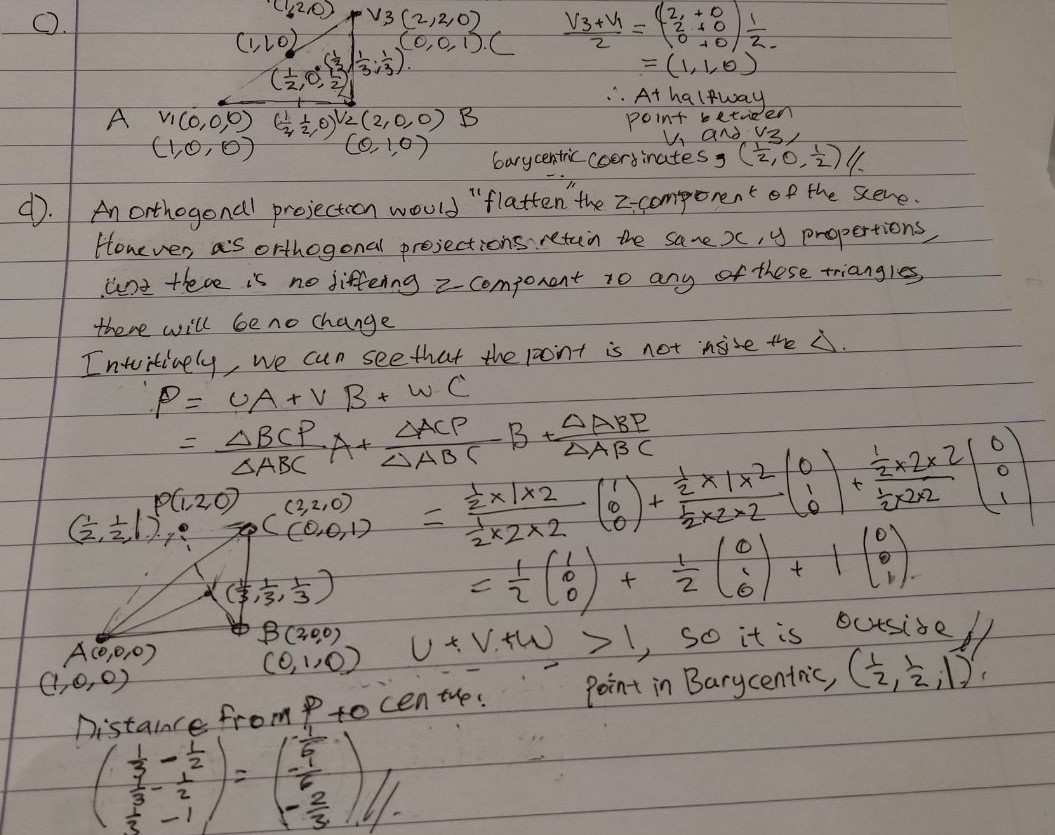
iii)

suppose we have trilinear coordinates of (t\_1, t\_2, t\_3) we will have (at\_1, bt\_2, ct\_3) as barycentric

I.e. multiply by triangle side lengths.

iv)

can check if the point is inside or not



c

(0, 0, 0.5)

d

(0.5 -0.5 1) not inside

incenter is in (1,1,1)

distance vector is (0.5 -0.5 1)

which is sqrt(2.5)

e

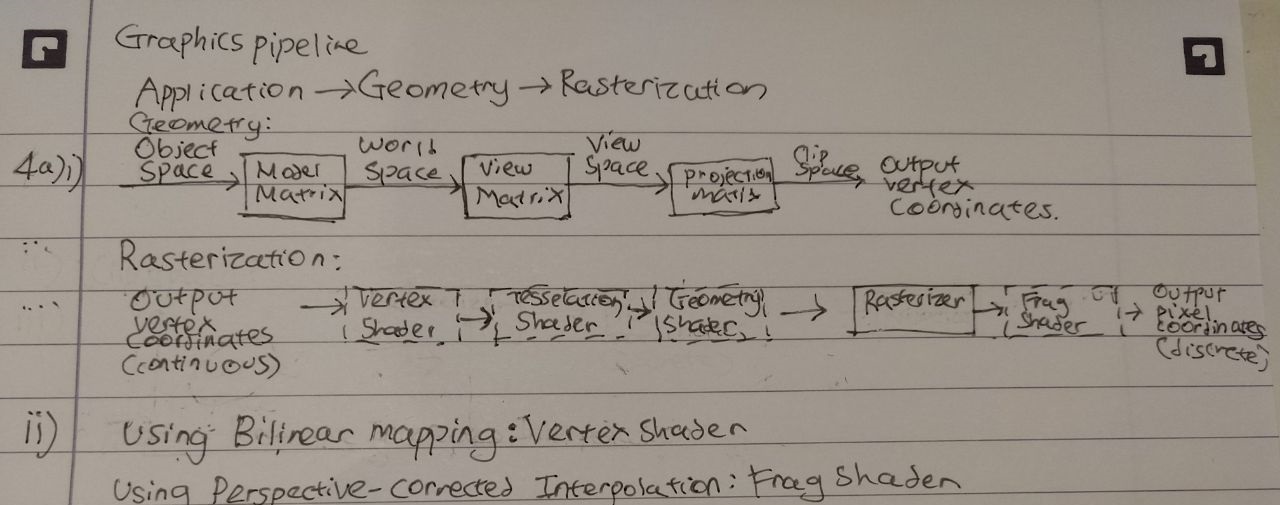
0.5 \* [0 1 0] = [0 0.5 0]

f

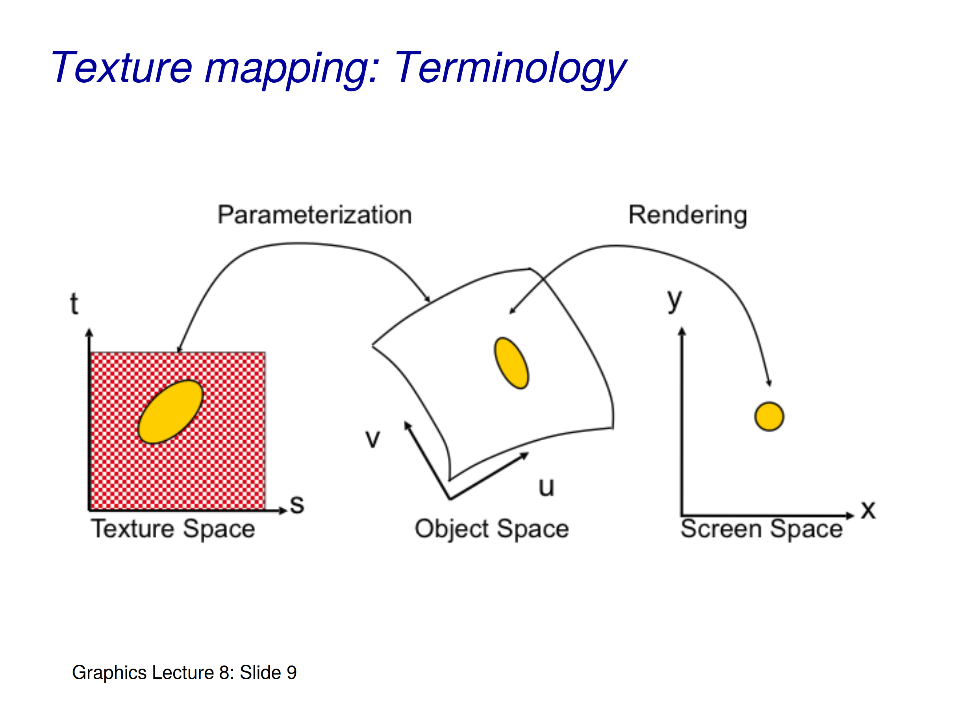
Don’t know

4

a



iii) Explaining the transformations from the following slide:



b

static colour

clamp

repeat

mirror

c

i)

No. The length is not preserved in perspective projection which will distort the texture.

ii)

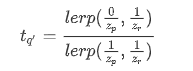
Plane mapping

box mapping

etc.

d

t\_{q’} = \frac{lerp(\frac{0}{z\_{p}}, \frac{1}{z\_{r}})}{lerp(\frac{1}{z\_p}, \frac{1}{z\_r})}



From notes: “*In order to solve this problem, instead of the texture coordinates u and v, the values of u/z and v/z and also 1/z are interpolated linearly, whereby z is the coordinate in 3D space in the direction of sight (z or 1/z must therefore be stored to each projected point of the polygon). In order to calculate the texture coordinates for a pixel, divisions must now be executed: u= (u/z ÷ 1/z), v= (v/z ÷ 1/z)*”

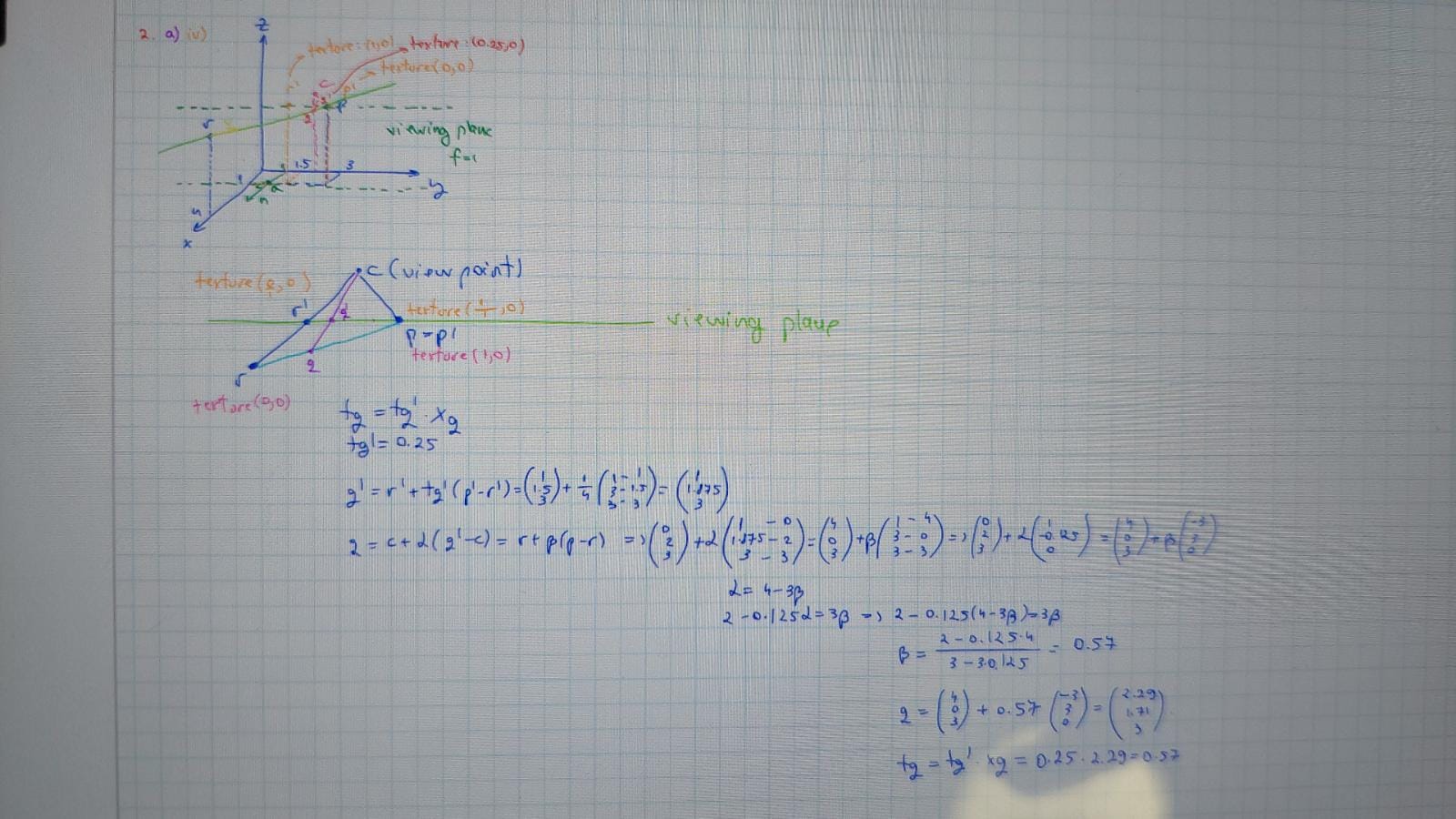
e

viewer position, the point P and the point R is all in the plane z = 3, so you can draw a sketch of that.

Interpolate between ¼ and 1. q’ is (0.25, 0) so I think at q its (0.25/(1-¼), 0) = (⅓, 0). Correct if wrong.  
https://www.comp.nus.edu.sg/~lowkl/publications/lowk\_persp\_interp\_techrep.pdf

---------------

I don’t know how to do the lerp thing but I think solution this is correct since I got beta = tq = 0.57

MSl

------------------------------------------------------------------------------------------------------

I agree with 0.57

tq = lerp(0/4,1/1) / lerp(¼, 1) = [0 + 1 \* 1/4]/[¼ + (1-1/4) \* 1/4] = 4/7 = 0.57

The tp and tr are switched compared to the slide because the former should have t = 0 while the latter should be t = 1

Could someone show the full working for this question? Going right over my head, would really appreciate it :)

F

i)

A normal map

ii)

Replace surface normals with those of the normal map. Geometry is **not** manipulated

iii)

Fake detail without changing the geometry. Significantly &more efficient