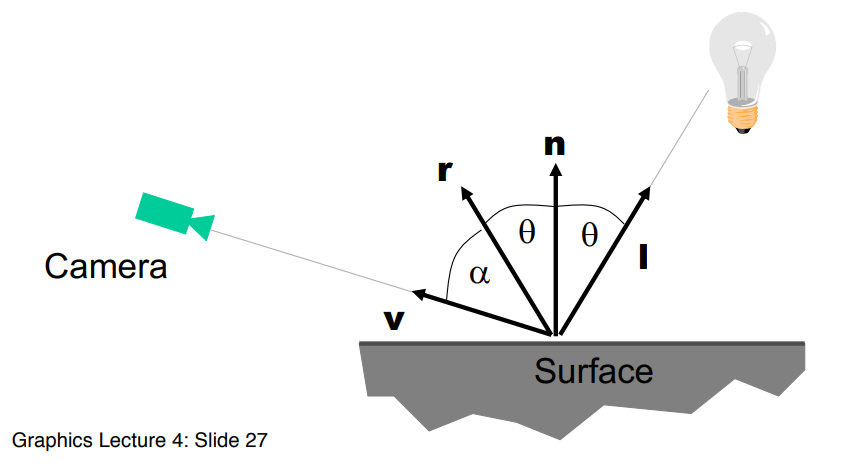
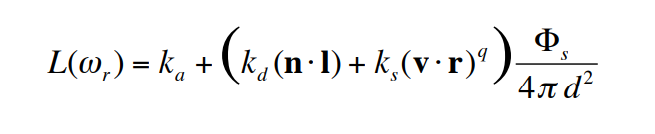
1

a





b

on the same surface, diffuse and specular components are both constant

c

i saturation

ii value

D

i

M\_1 = [[0.6, 0.2, 0.1],

[0.3 0.6 0.1],

[0.1 0.2 0.8]]

M\_2 = [[0.6 0.1 0.15],

[0.25 0.7 0.1],

[0.15 0.2 0.75]]

ii

M2-1 M1 = [[1.01 0.2 -0.106],

[0.82 0.784 0.028],

[-0.092 0.016 1.078]]

iii

M\_{2}^{-1} M\_{1} c = [0.7146 0.1386 0.045] ^ T

2

A

i)

Find p(0), p(1), then p’(t), p’(0), p’(1). The p(1) and p’(1) will be functions in a, b and the other p(.)s. You can solve these by simultaneous equations to get values for a and b.

ii

Not sure - do you have to find t at p2 and t at p1 then find their respective equations and then p2-p1?

Do the coefficients count as parameters? Wording is unclear but this makes sense

a = [0 1]^T

b = [0 -1.5]^T

c = [1 0.5]^T

d = [2 2]^T

b

i

Take derivative and double derivative of the equation, plugging in 0s to get p’(0) and p’’(0). Write the four equations p(0) = ... p’(0) = ... p’’(0) = ... and p(1) = ... then write that in matrix form as (p(0) p’(0) p’’(0) p(1))^T = matrix \* (a b c d)^T

Find inverse of matrix then write in terms of (a b c d)^T = matrix^-1 (p(0) p’(0) p’’(0) p(1))^T

Ii

Use p0’’ ~= ½(p1’-p-1 '), p-1' is a free choice? Use p-1'=-p1’?

c [no idea with this q - if C0 continuous, I think it is also G0 continuous]

i C0 continuous as it agrees on the matching endpoints by definition

Ii not (necessarily) C1, as no matching on gradients is done (only used at one endpoint)

Iii C0=G0, so yes.

Iv not G1? Similar reasons to ii.

3

4 [very rough please correct]

a

Z-buffer: picks nearest fragment (to the viewer) and corresponding colour per pixel. Updates if it finds nearer fragment.

bi

In ray tracing notes I think. Note GLSL so no recursion, must be iterative formula.

bii

* Maximum ray trace depth reached
* No object hit i.e. no intersection
* Colour attenuation beneath a certain threshold

ci

Shadows generated using a shadow intersection. Have an intersect() function which takes in a light ray and shadow intersection. If the shadow has been hit, return black, else return the original colour attenuated by the additional distance it has travelled.

cii

Artefacts occur because of slight differences when computing exact values. Calculations are exact but sphere isn’t (because pixels so discrete approximation to real world) so the ray can actually end up inside the sphere when it should touch the surface. We add epsilon\*intersection\_normal so that it gets out of the sphere and produces the correct shadow.

Ciii

One method is to use an area light source instead of a point light source. Can simulate this effect when calculating shadow rays by shooting multiple slightly tilted rays towards the light area. Computationally expensive to shoot more rays at each ray trae.