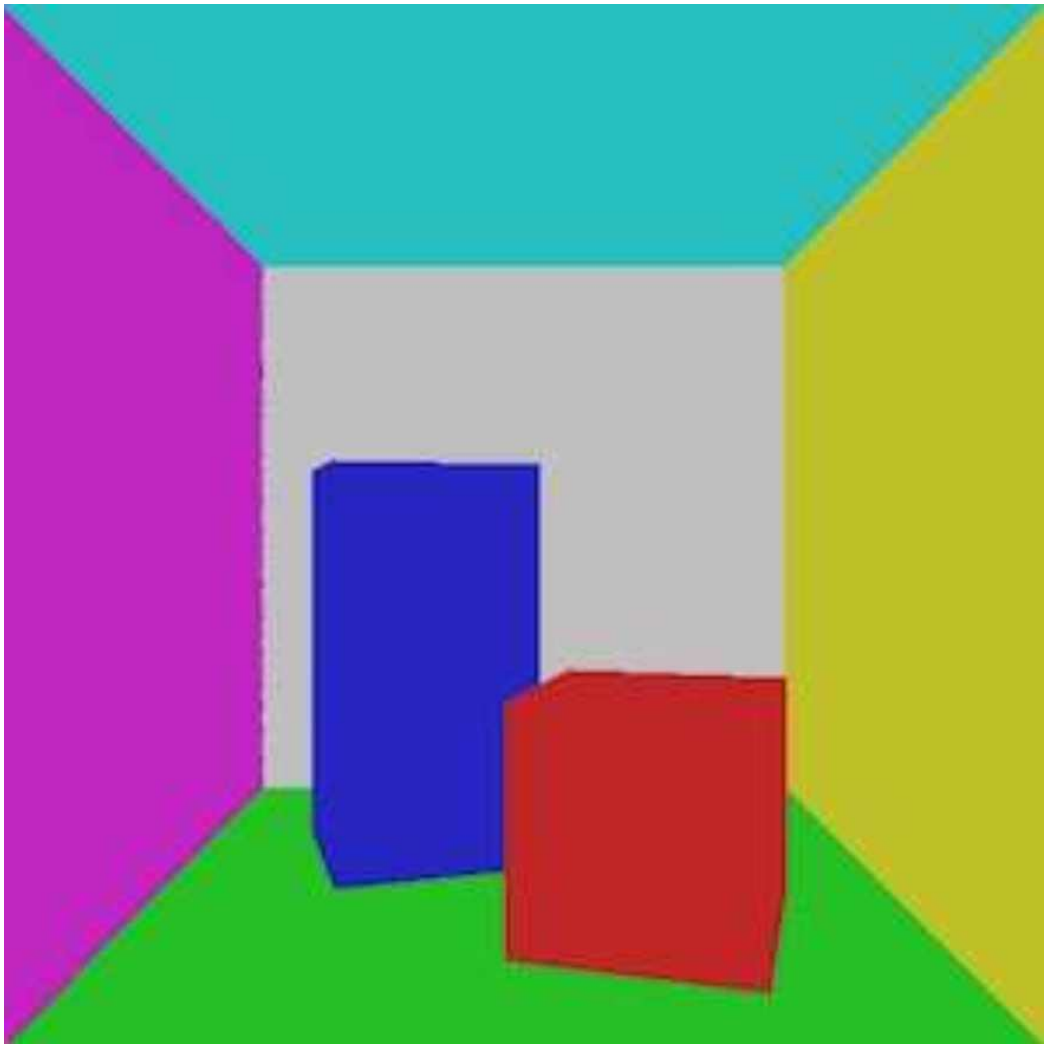
Computer Graphics - Week 7

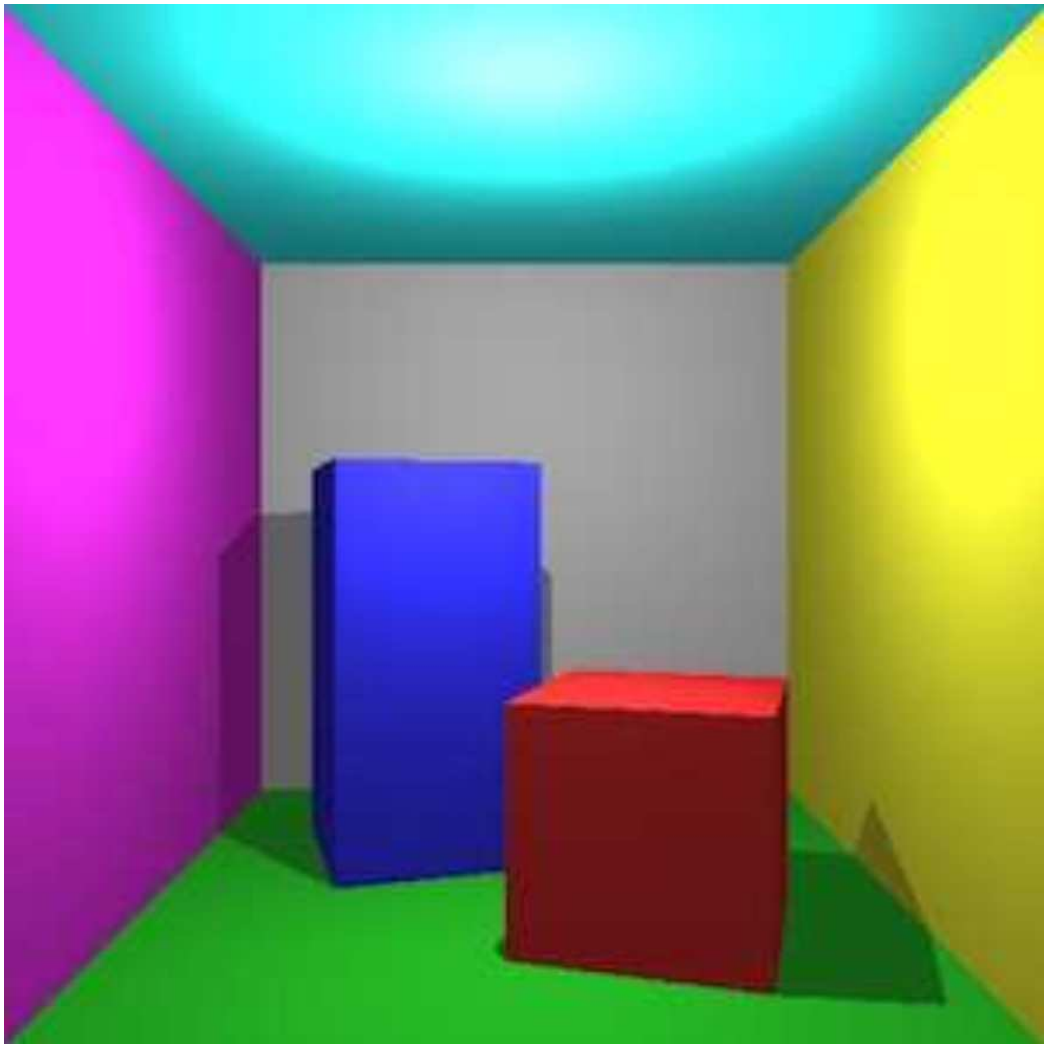
Lighting and Shading

Dr Simon Lock

How could we improve the current render ?



How about this ?



What do we need to be able to do this ?

If we know the location of the light in 3D space
Then we can "easily" calculate:

- Reachability/visibility of surfaces from lights
- Distance of surfaces from lights (dissipation)
- Angle that light falls onto surfaces (obliqueness)

Calculating all of these requires 3D spatial data
Problem with rasterising is that a lot of this is lost...
When we "flatten" the scene onto the image plane
With Raytracing however, we retain all of this data !

Limiting Complexity

To ease calculation, let's make two simplifications:

1. There is only ONE light for the 3D scene/world
2. It is a "single point source" (no width/height)

This will make our job a lot more straight-forward
But will still produce some nice looking light effects

We might consider more complex lighting later on
(If you're doing the coursework variant of this unit)

3 Types of Light !

It is "convenient" to consider 3 types of light:

- Diffuse: Direct illumination of surfaces
- Specular: Mirrored reflection of the light source
- Ambient: All around, background light level

ThreeTypesOfLight

This concept (3 types) does NOT occur in nature
A quick "approximation" (rather than "simulation")
But it does give some "plausible" looking results :o)

Diffuse Light

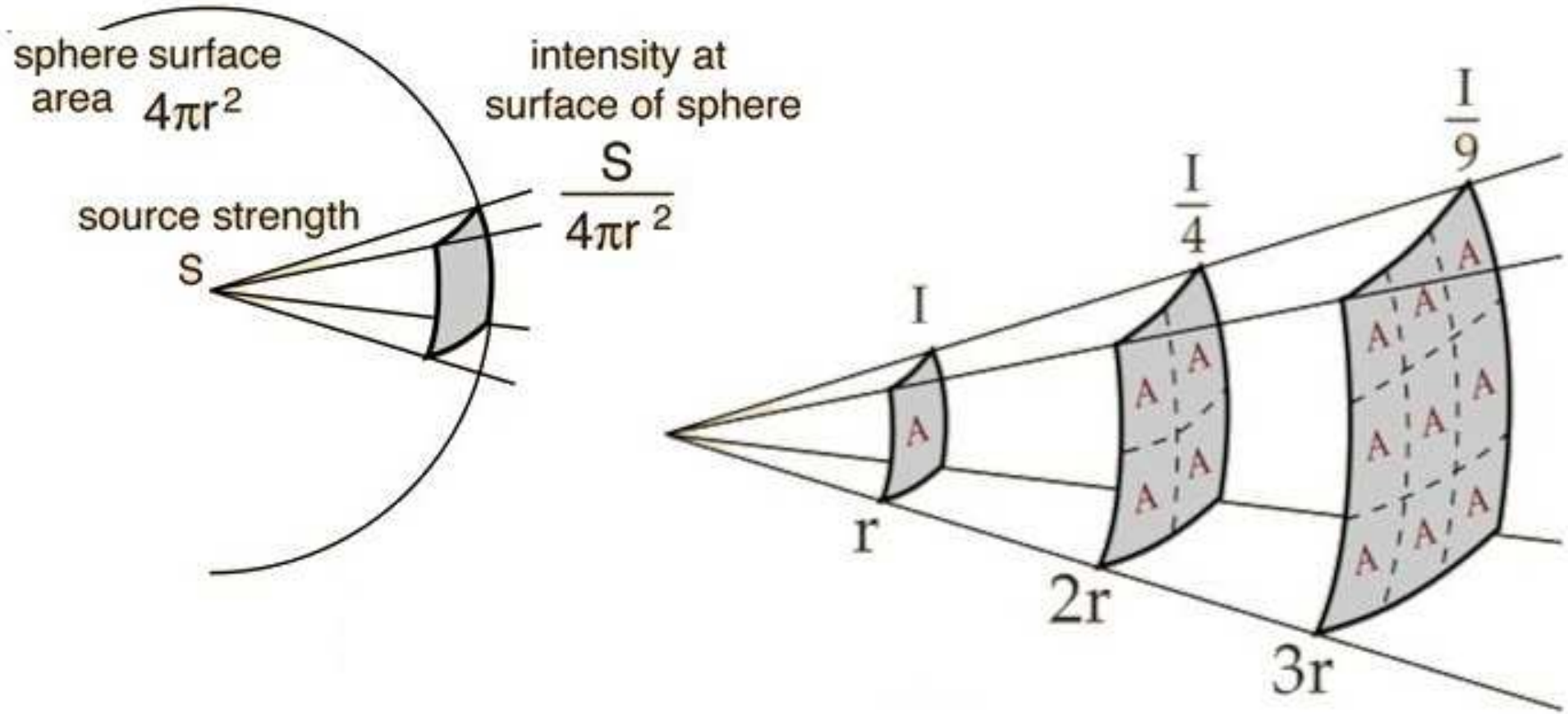
Think of this as a "standard" or "core" light effect

To calculate we must consider BOTH the following:

- The DISTANCE of the surface from the light
- The ANGLE the light falls onto the surface

Let's take a look at each of these in turn...

Light Intensity Drop-off (Dissipation)



Light Angle-of-Incidence

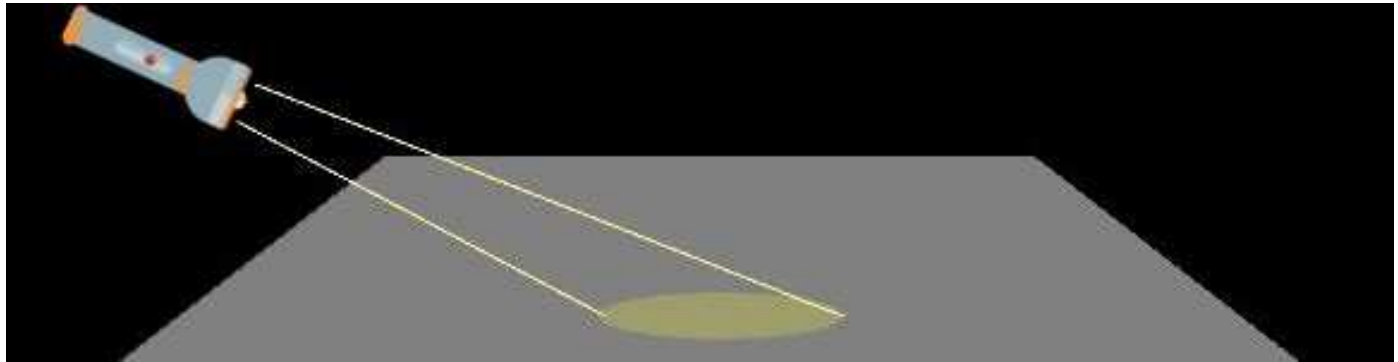
Must take into account "Angle of Incidence" (AoI)

More obliquely a light strikes, further it spreads

In order to calculate the angle of incidence...

We need to know the position of the light

And orientation of the surface being illuminated

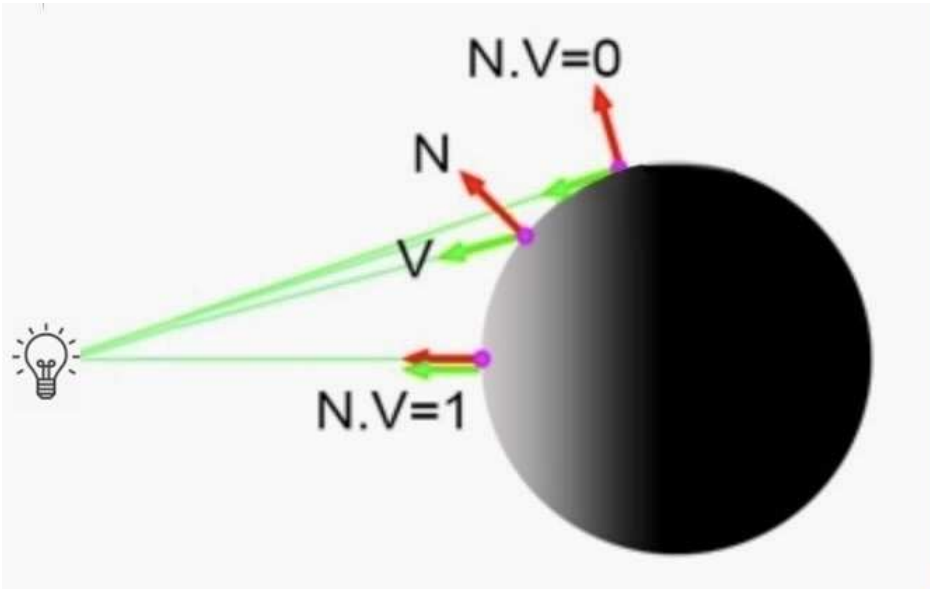


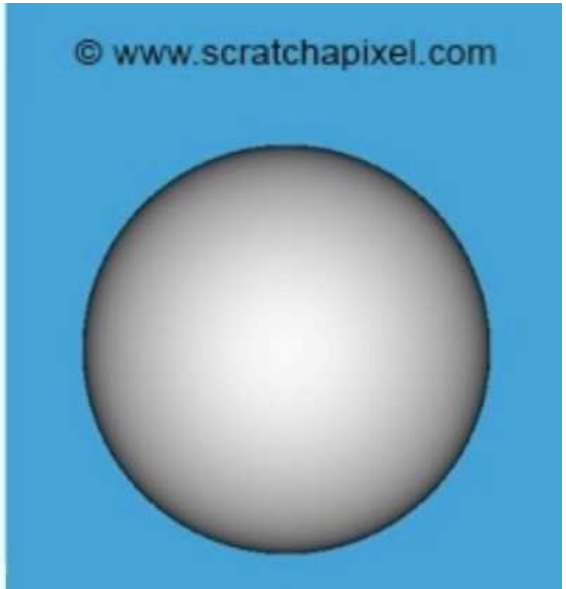
Calculating Brightness Due to AoI

Dot product normal N with surface-to-light vector V

- If they are perpendicular, dot product is 0 (fully dark)
- If they are parallel, dot product is 1 (fully bright)

Anywhere in between, brightness is between 0.0-1.0



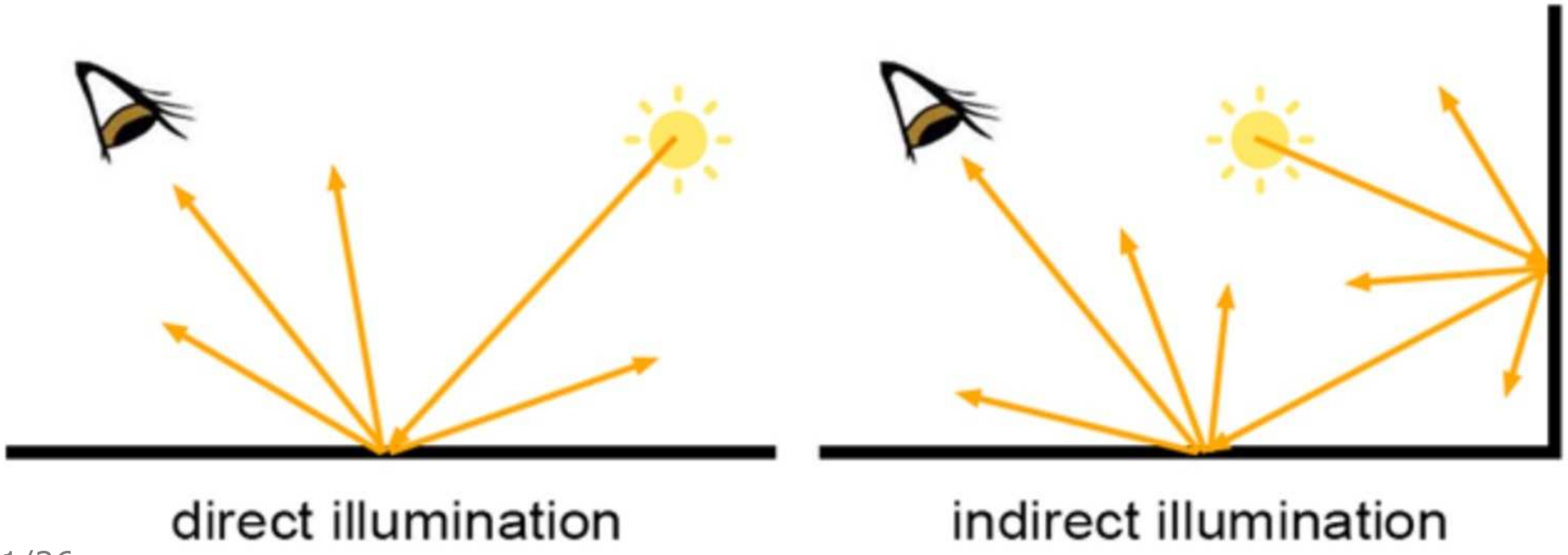


Indirect Illumination

In the real world, light bounces off other objects

Ray can bounce many times before reaching surface

Imagine trying to calculate all of this !



Faking Indirect Illumination

We could try to calculate FULLY bouncing light
But this can be VERY computationally expensive

Possible solution: "Ambient lighting"

Quick & dirty way to fake bounce
A "background" level of lighting
Min threshold for all surfaces

"don't let the brightness...
...drop below 0.2"

Specular Lighting

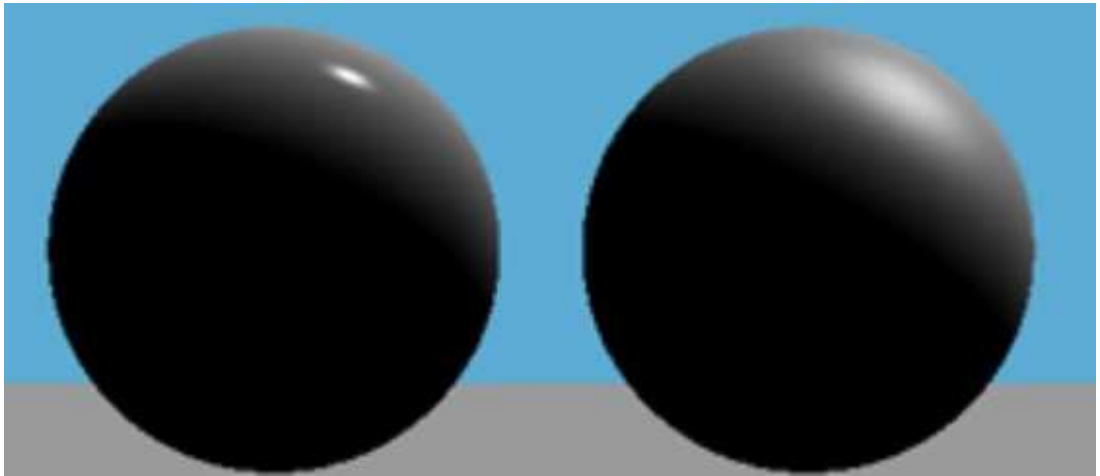
Specular varies depending on nature of material

The left-hand sphere looks very glossy ("shiny")

The right-hand sphere looks quite matt ("frosted")

Roughness / imperfections on surface scatter light

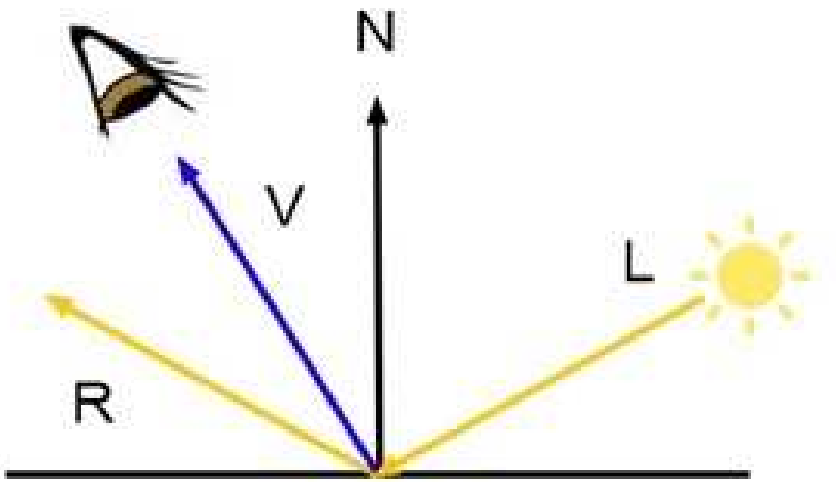
And cause the specular highlight spot to "spread"



Specular Highlight - The Theory

Need to determine how far off the view angle "V" is from the "perfect mirror" light reflection vector "R"

The closer the view angle is to reflection direction, the brighter you paint the pixel !

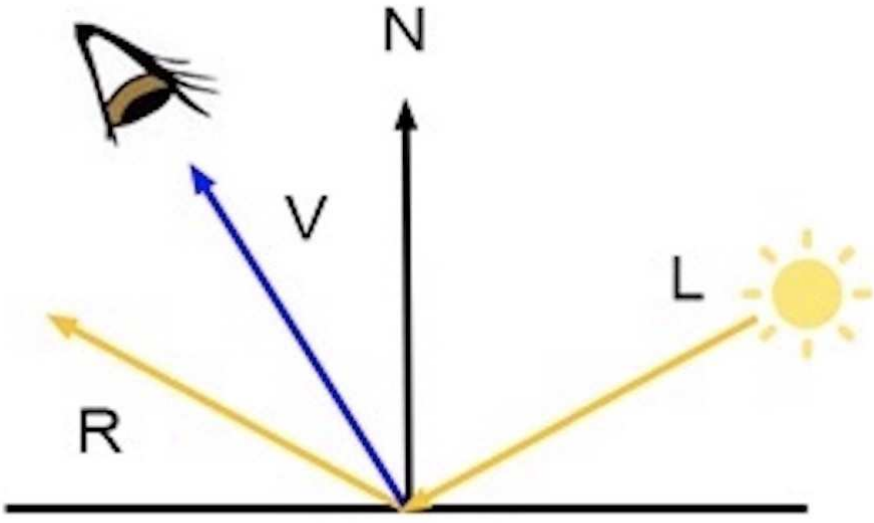


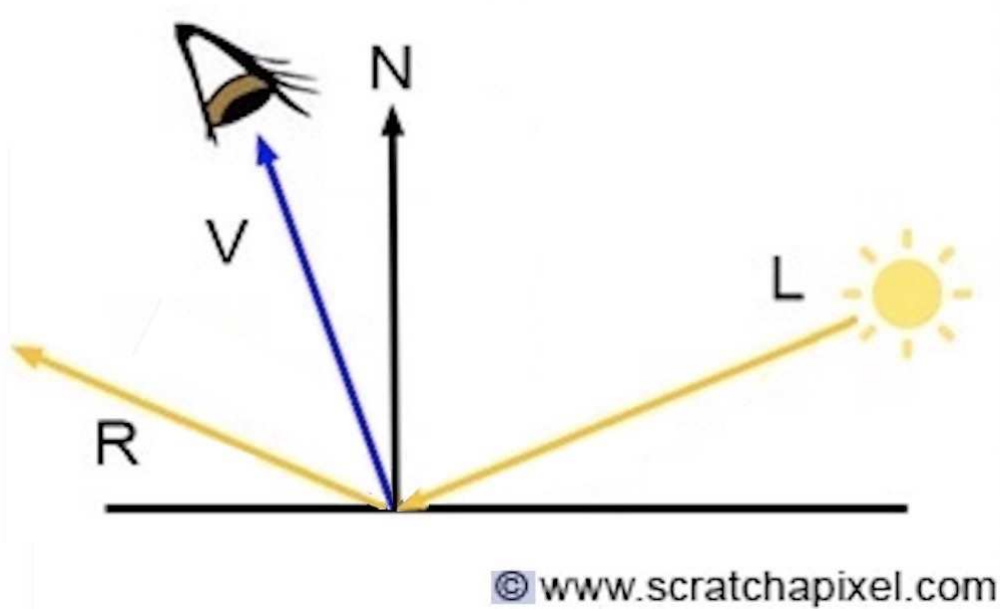
Important Note

We are NOT moving the camera or light around in space

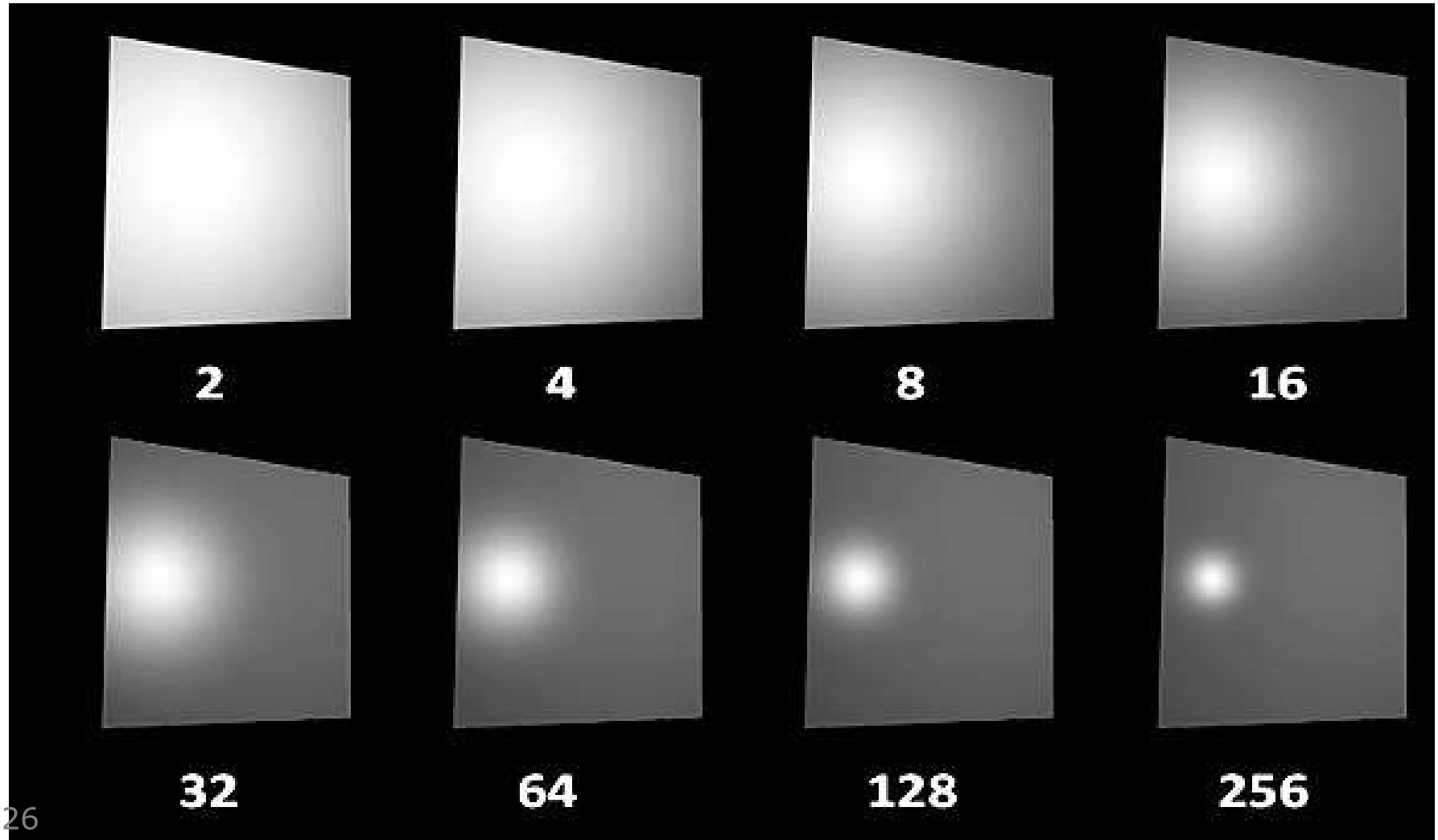
Instead calculating angles for EACH POINT on surface

Each point will have different L, R and V directions





Adjust parameters for different materials



"Blocky Model" Problem

All our models are composed of a set of triangles
When shading these triangles to illustrate lighting,
they are BOUND to look like flat surfaces
(because that is what they are !)

This can make the model look
quite "low-res" and "blocky"

Particularly if there aren't
that many triangles !

Intelligent Shading ?

Could we shade the triangles more "intelligently" ?

Not just looking at each triangle in isolation
But also at the *surrounding* triangles

Detect the "trend" of the surface
Smooth off those sharp corners
"Blend" the triangles together
Avoid those ugly flat faces

Gouraud Shading

Gouraud shading in one such intelligent approach !

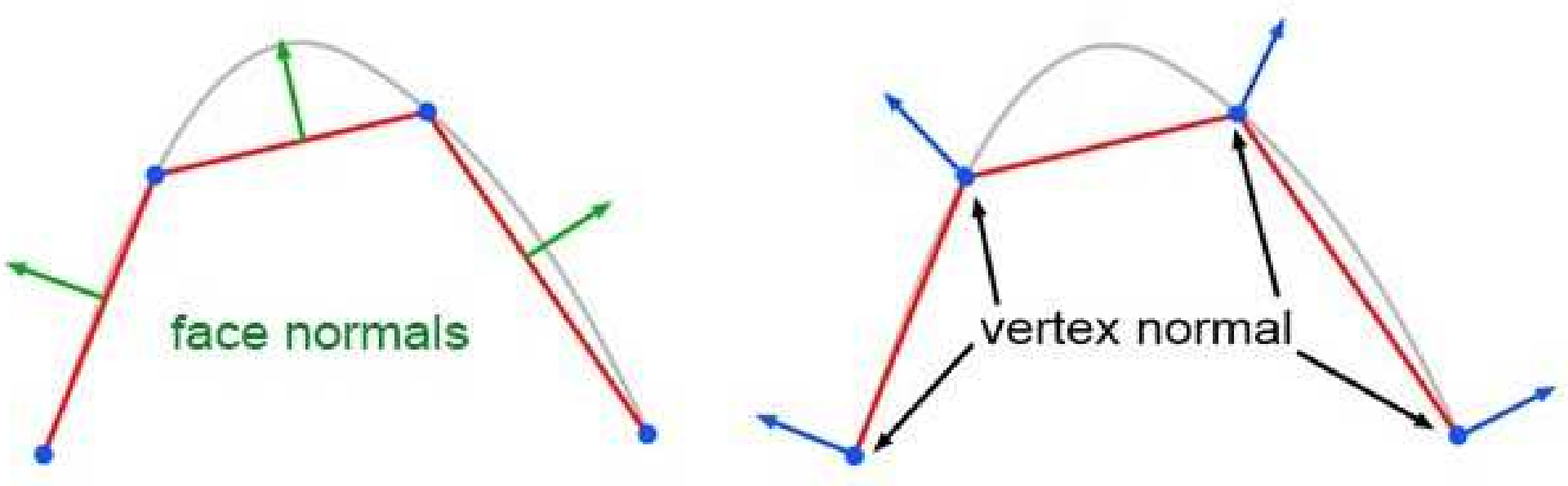
Rather than shading whole triangle with a flat colour (based on the triangle's single surface normal)

We shade every single pixel on the triangle uniquely (Depending on where on the triangle the pixel sits)

Our calculation will be based on "Vertex Normals"...

Vertex Normals

A "Vertex Normal" is calculated by taking the average of all "Face Normals" of surfaces that share that vertex



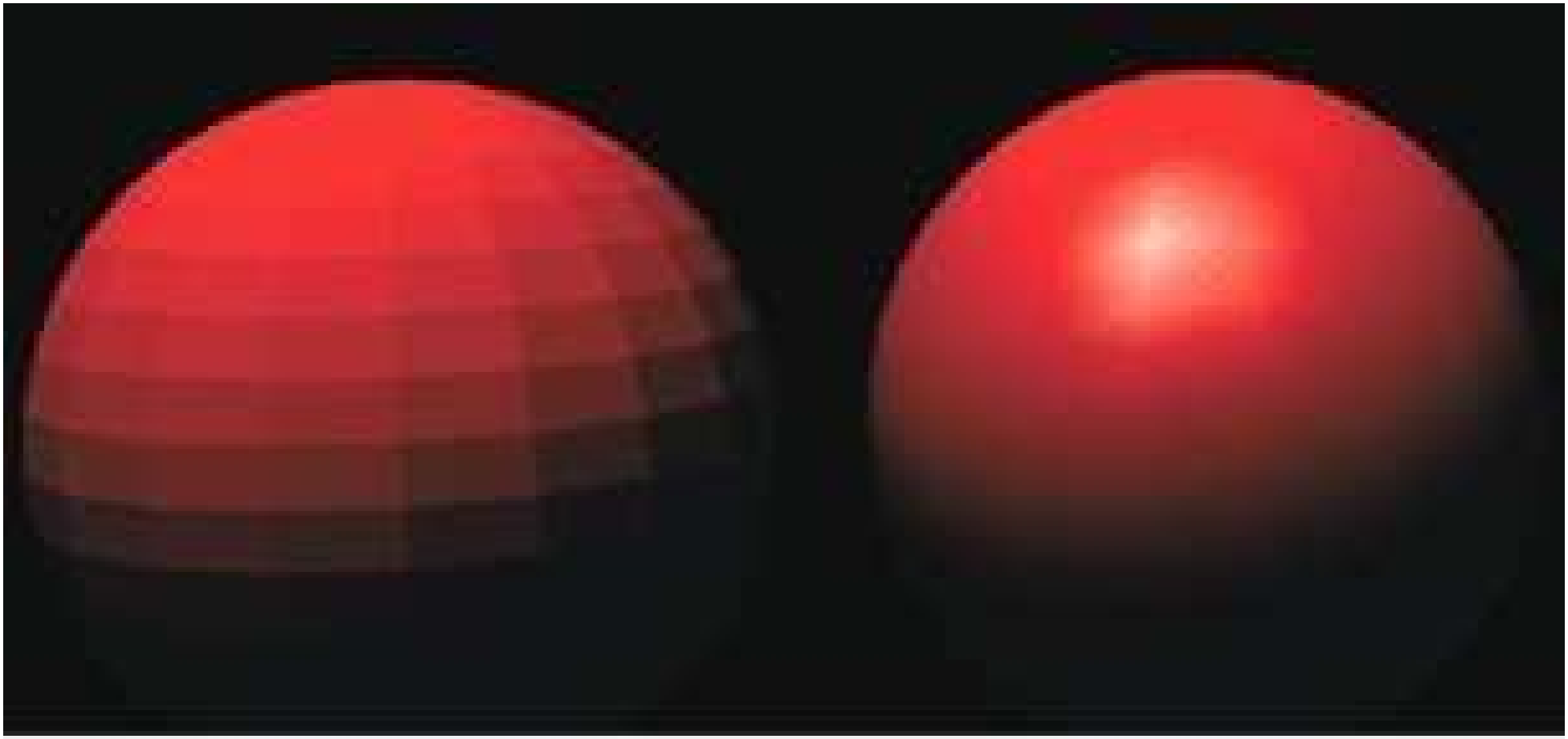
Lighting Calculation

Each triangle now has 3 normals (not just 1) !
Each of which can be used to calculate lighting
(angle-of-incidence, specular highlight etc.)

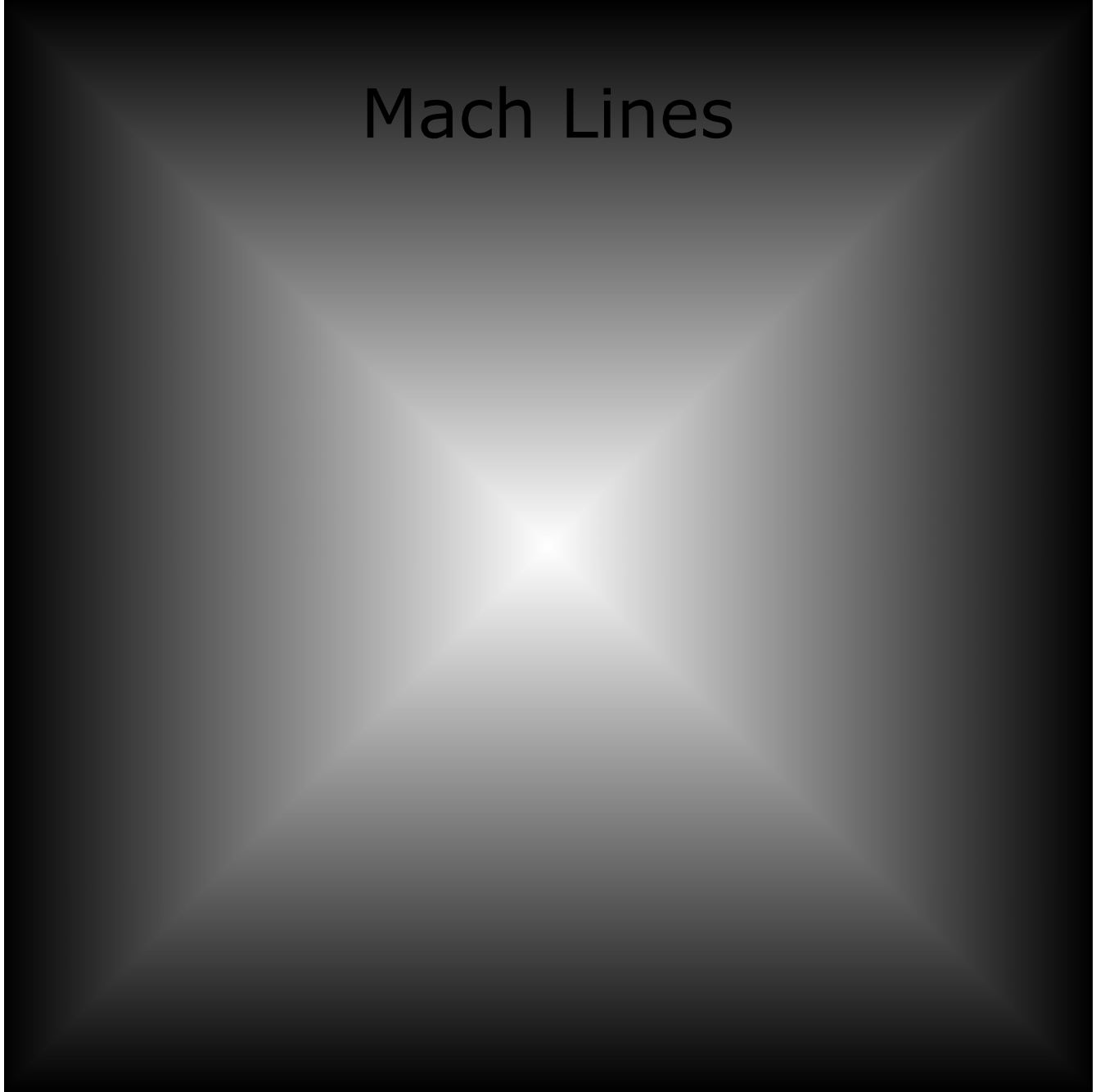
Calculate unique brightness for every pixel
Weighted average of light at each vertex
Based on distance of pixel from vertices
Makes use of Barycentric Coordinates

BarycentricWeighting

Same Geometry - Different Shading



Mach Lines

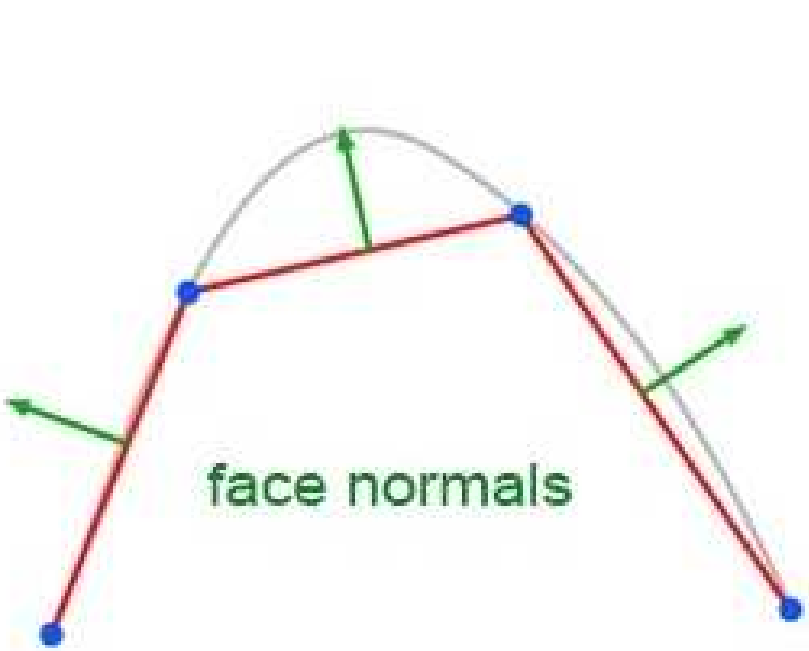


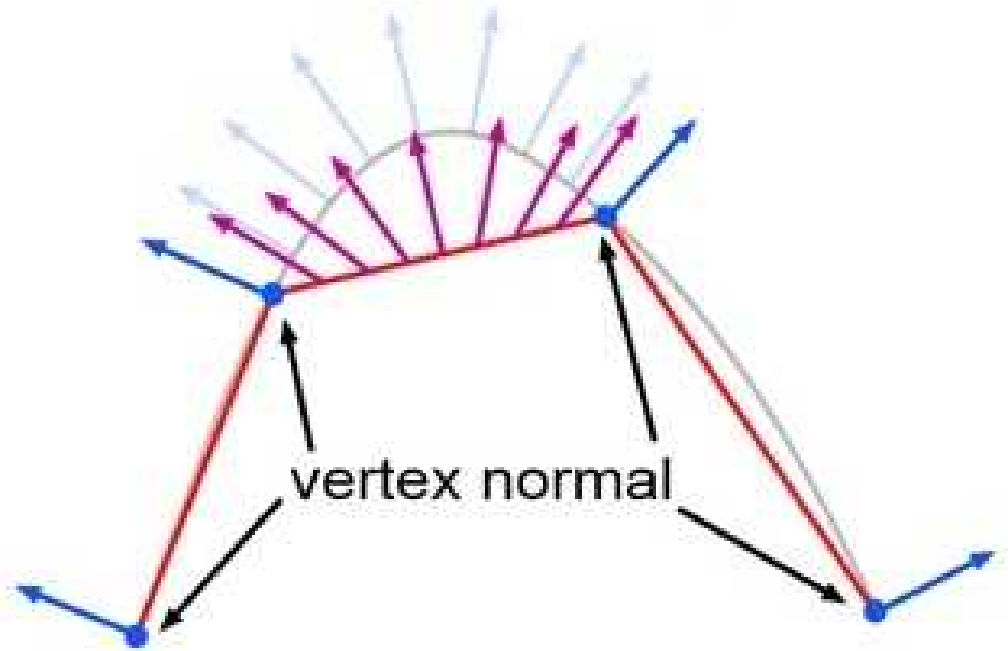
Spatial High-Boost Filtering
resulting from
Neural Lateral Inhibition

Phong Shading - Even Smoother !

Interpolate the *vertex normals* across the surface

THEN calculate a unique brightness for each pixel





Comparison

Interpolating normals (rather than brightness)

Produces much smoother overall object shading

At expense of rendering speed (as you might expect)