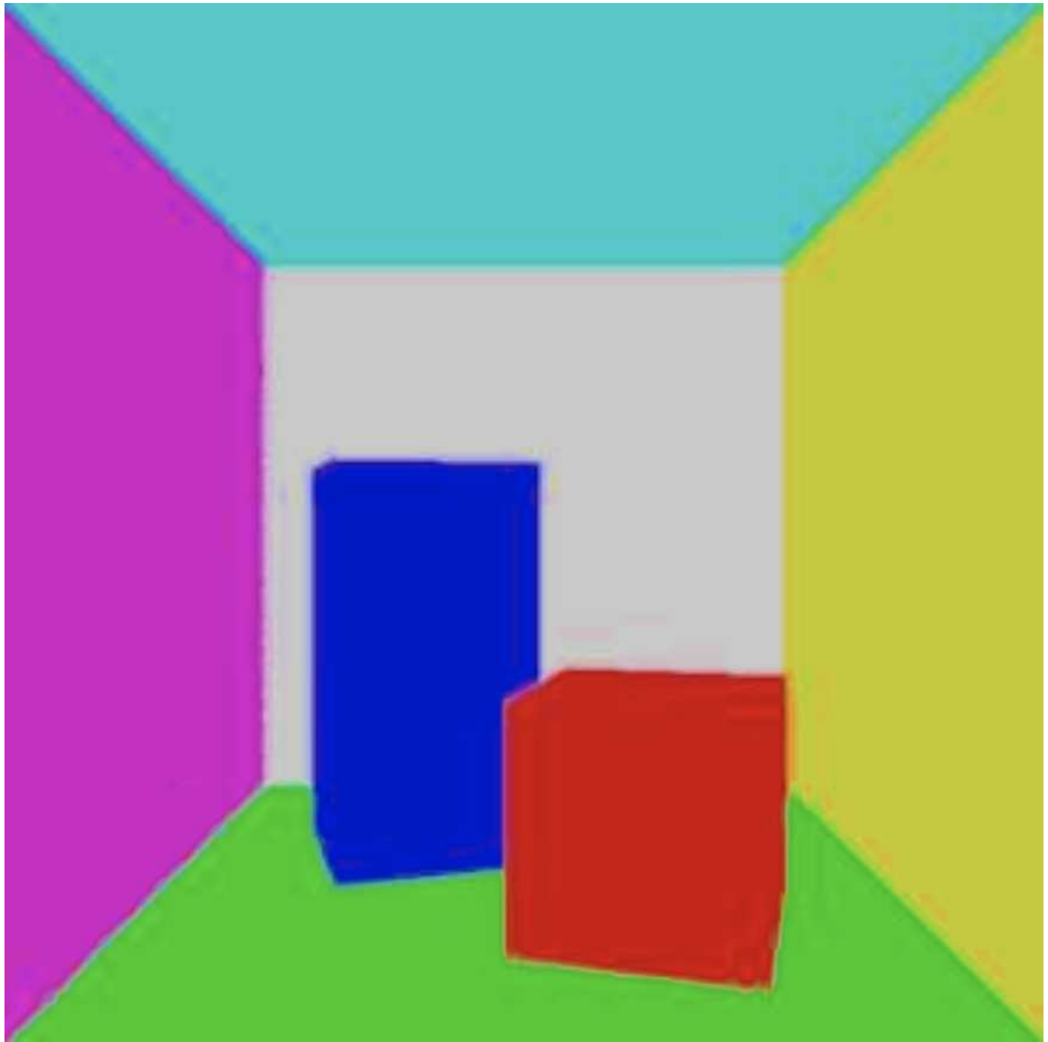
Computer Graphics - Week 7

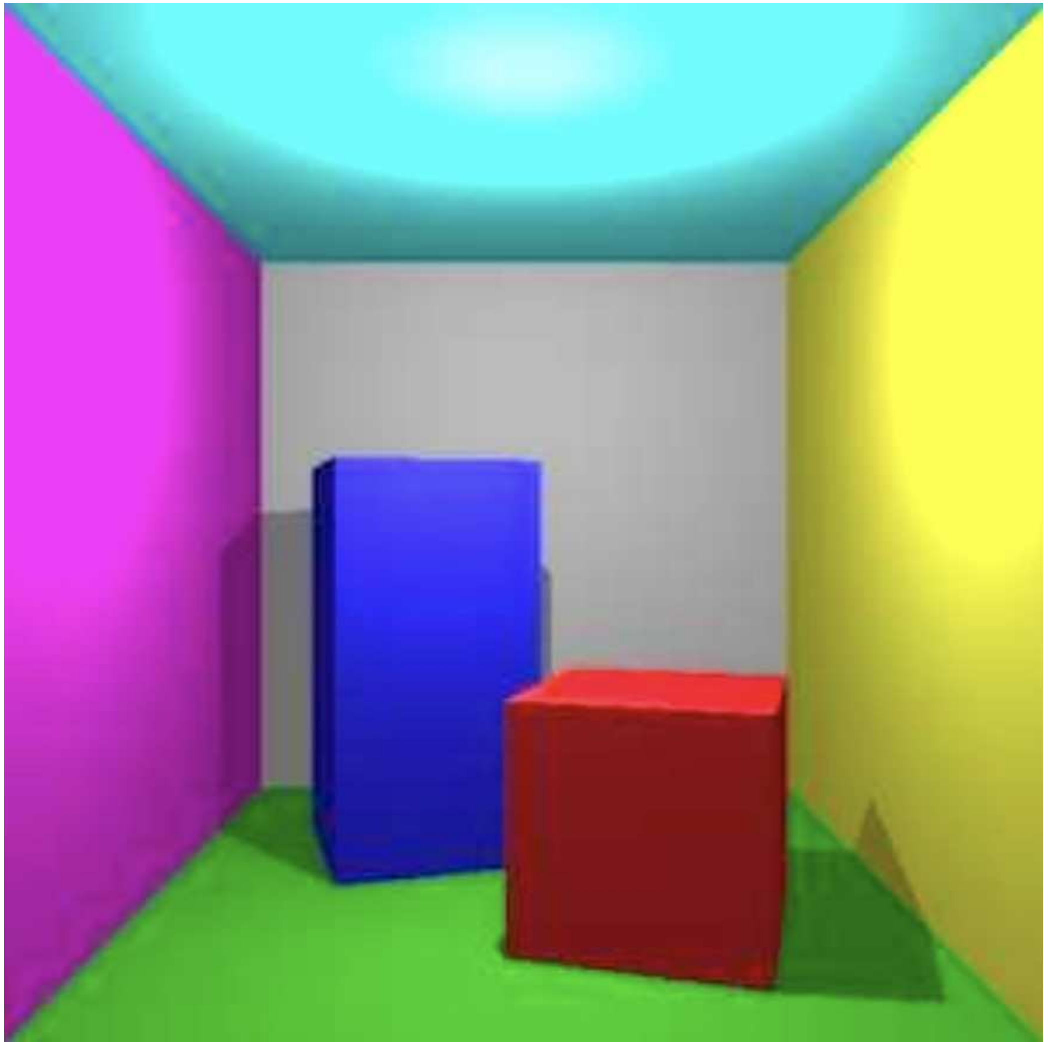
Lighting and Shading

Dr Simon Lock

How could we improve the below render ?



How about this ?



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 produces some "plausible" looking results :o)

Diffuse Light

Think of this as a "primary" or "core" light effect
To calculate, we must consider BOTH the following:

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

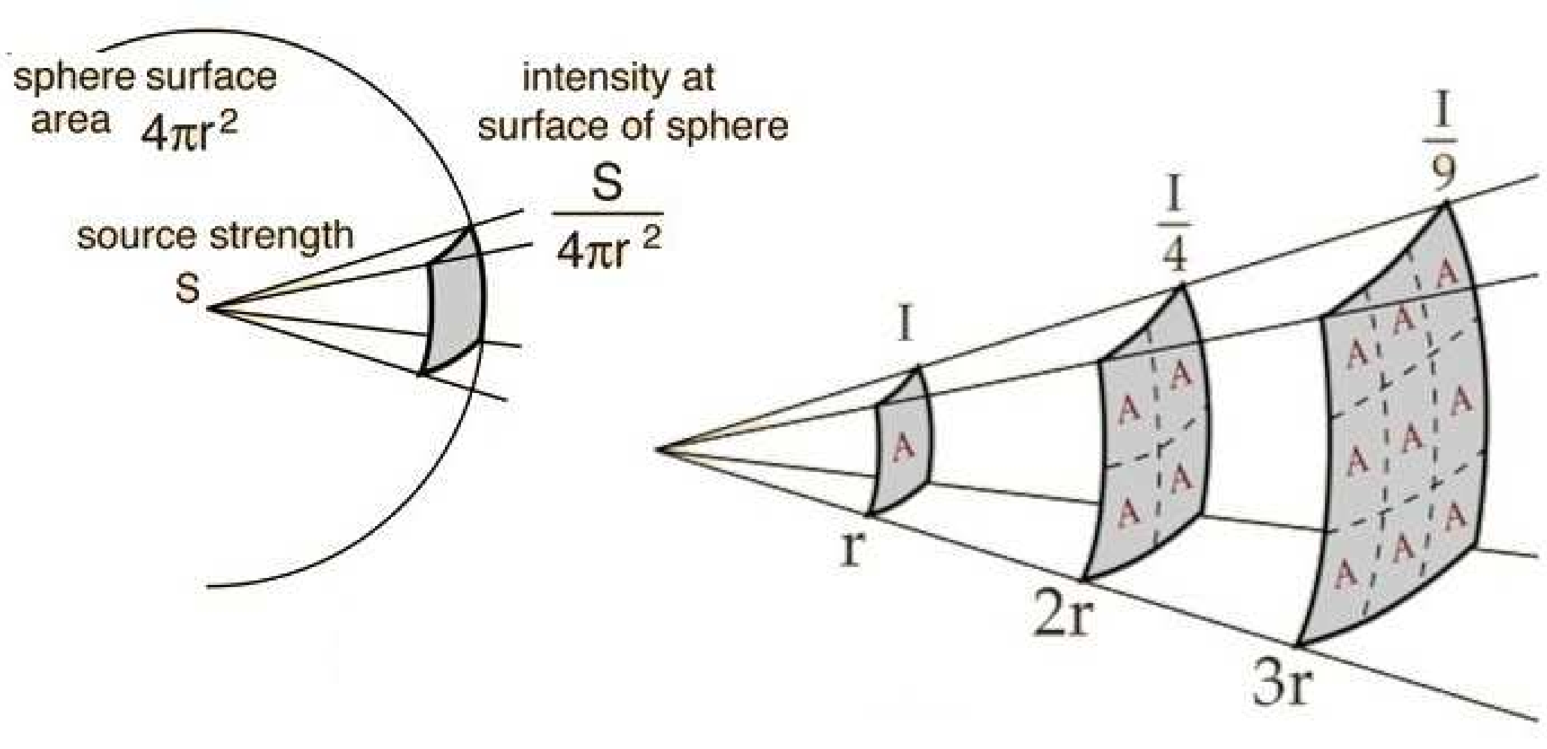
These need to be weighted together carefully
(In order to ensure "convincing" illumination)

Let us consider each of the above two in turn...

Light Intensity Drop-off (Dissipation)

Light from a source disperses in ALL directions

So the spread can be modelled as a sphere



Light Angle-of-Incidence

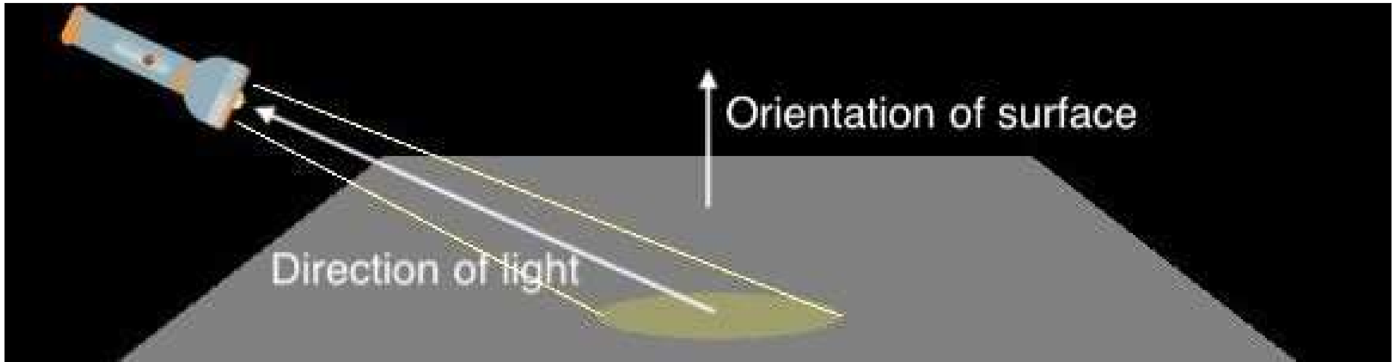
Must take into account "Angle of Incidence" (AoI)

More obliquely a light strikes, further it spreads

In order to calculate the angle of incidence...

We need to know the relative position of the light

Also orientation of the surface being illuminated

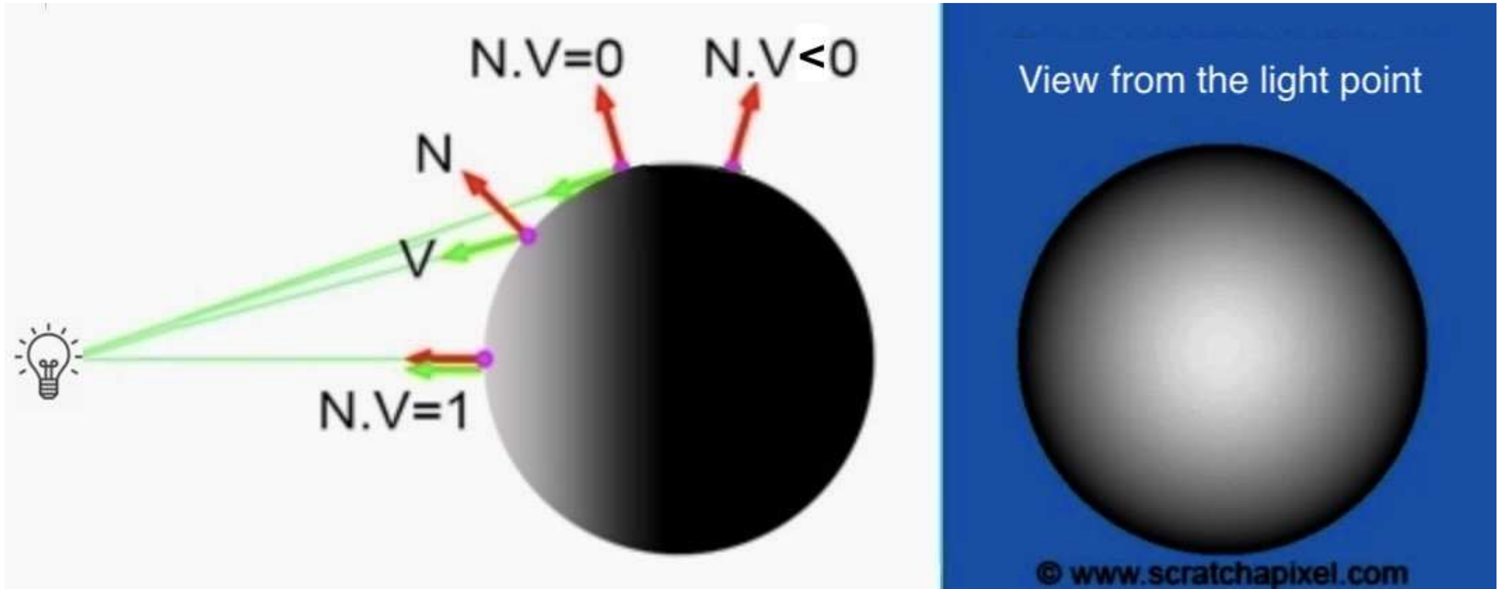


Consider a Spherical Model

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

- If they're parallel, dot product will be 1 (fully bright)
- If they're perpendicular, dot product is 0 (fully dark)

Anywhere inbetween, brightness is between 0.0-1.0

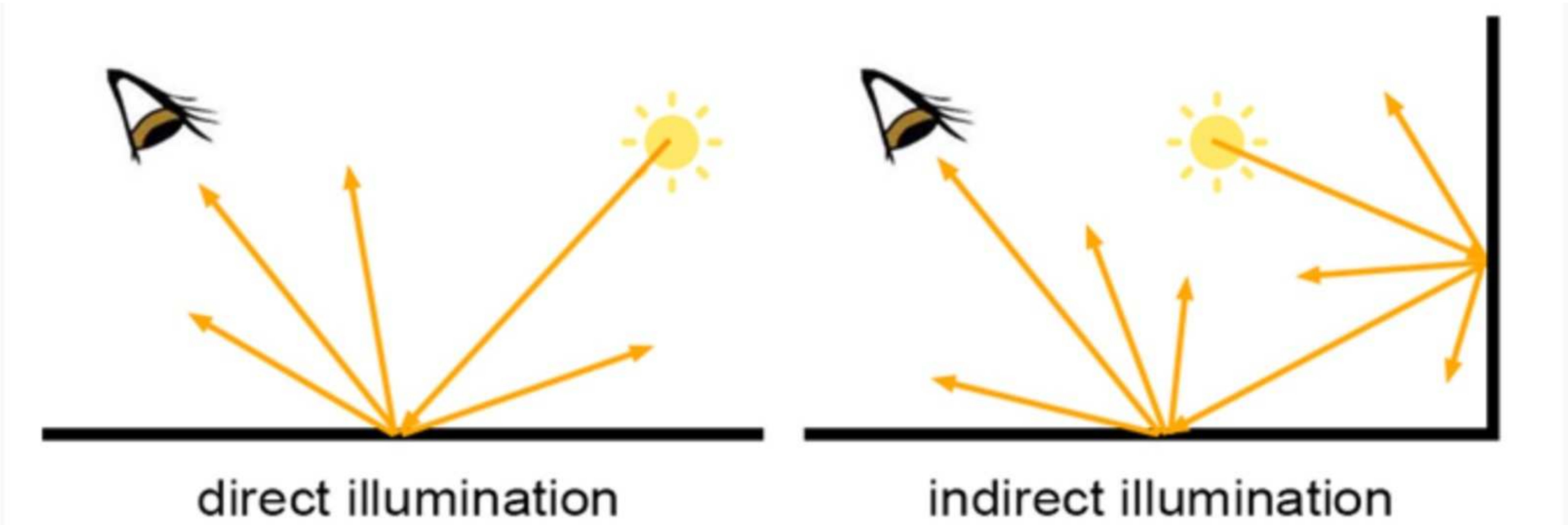


Indirect Illumination

In the real world, light bounces off all present objects

A ray can bounce many times before reaching a surface

Imagine trying to calculate ALL of these bounces !



Faking Indirect Illumination

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

<u>One easy solution: "Ambient lighting"</u>

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

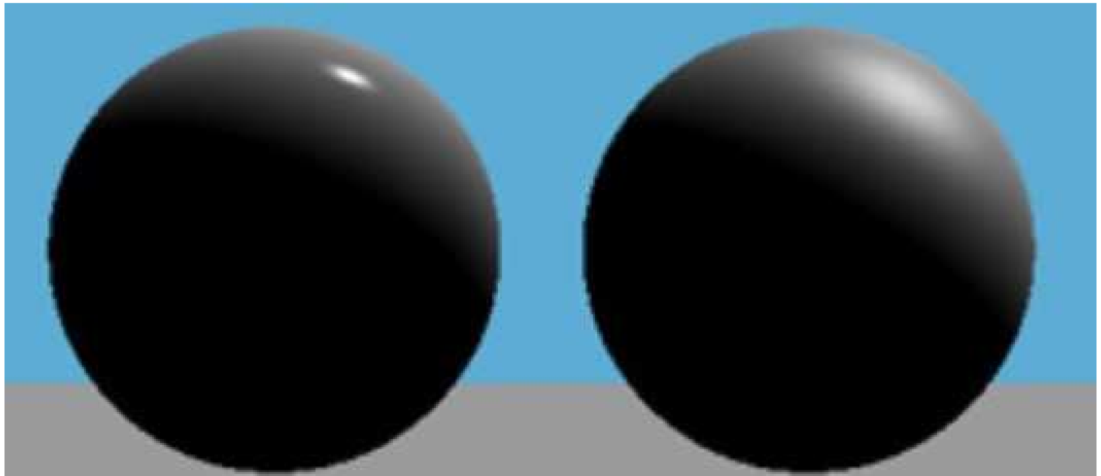
Specular varies depending on nature of material

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

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

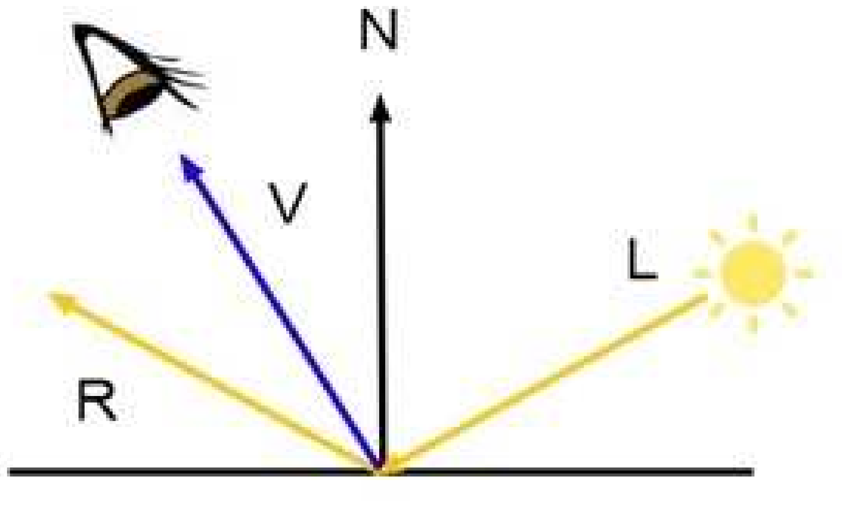
Roughness / imperfections on surface scatter light

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 the point on the surface will appear !



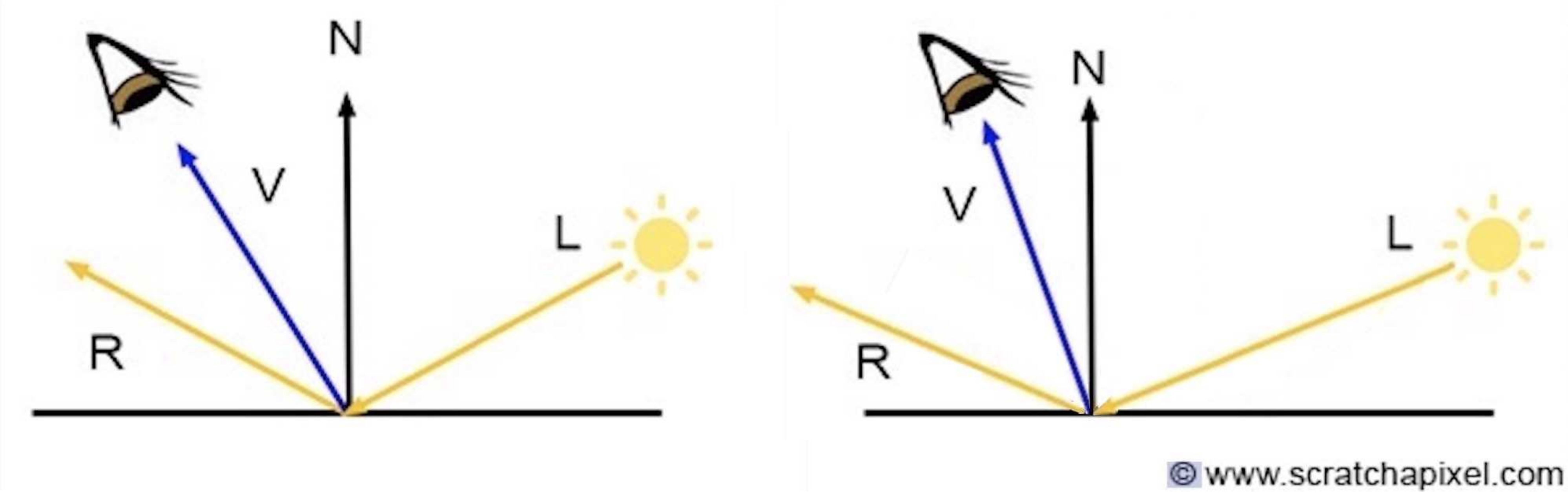
© www.scratchapixel.com

Important Note

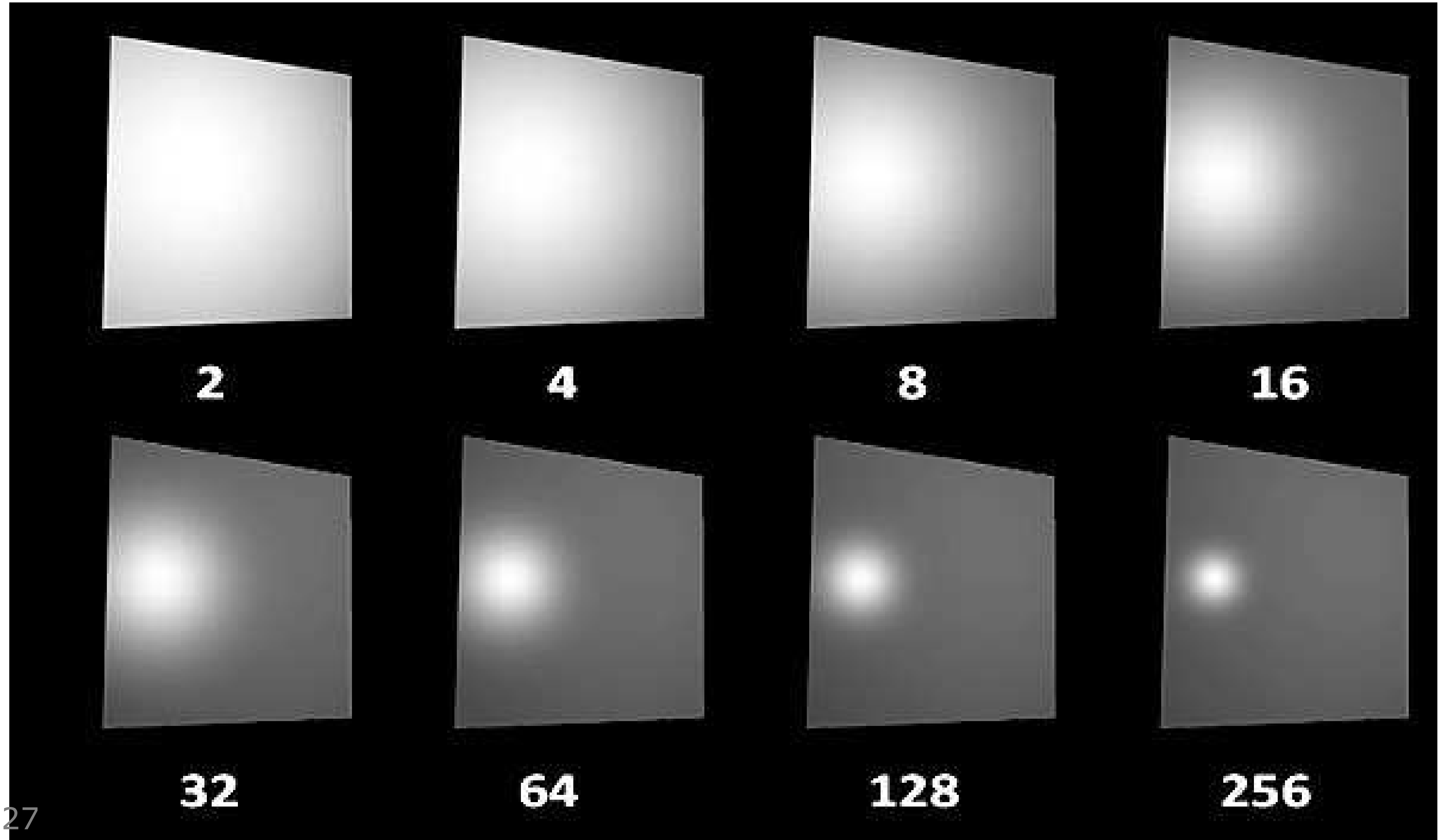
We are NOT moving the camera or light around in space

We're calculating angles for EVERY POINT on the surface

Each point will have *different* L, R, V direction vectors



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 ?

What if we could shade faces more "intelligently" ?

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

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

Gouraud Shading

Gouraud shading in one such intelligent approach !

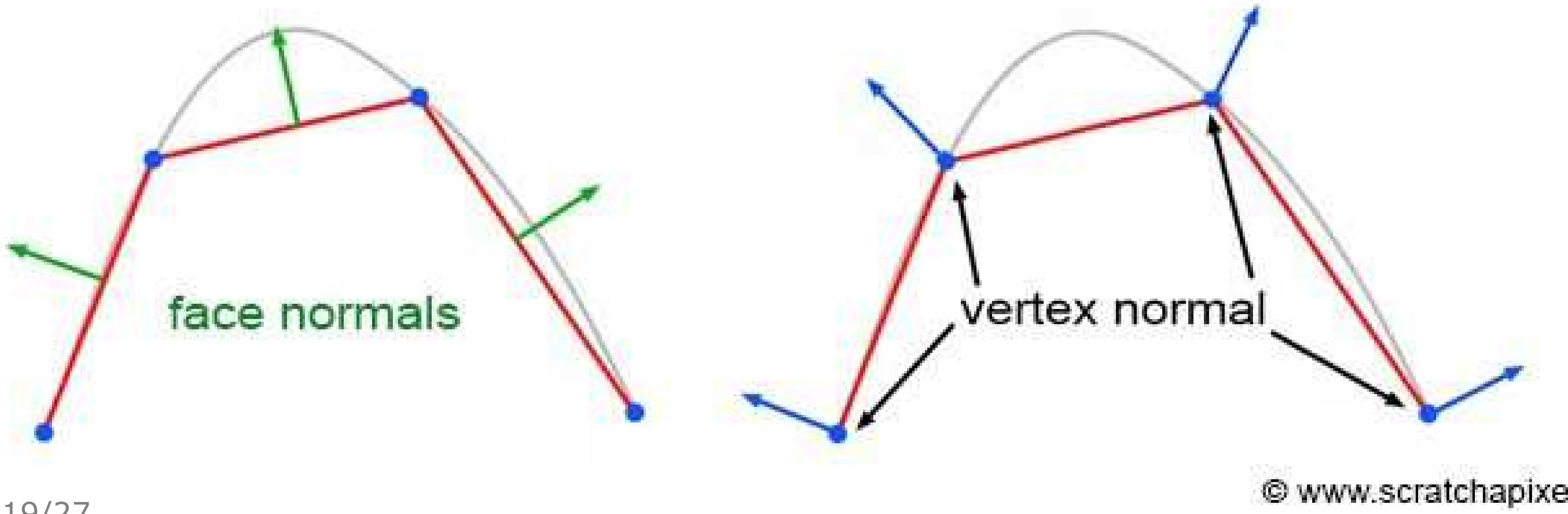
Rather than shading whole triangle with a flat colour
(based on the triangle's single surface normal)

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



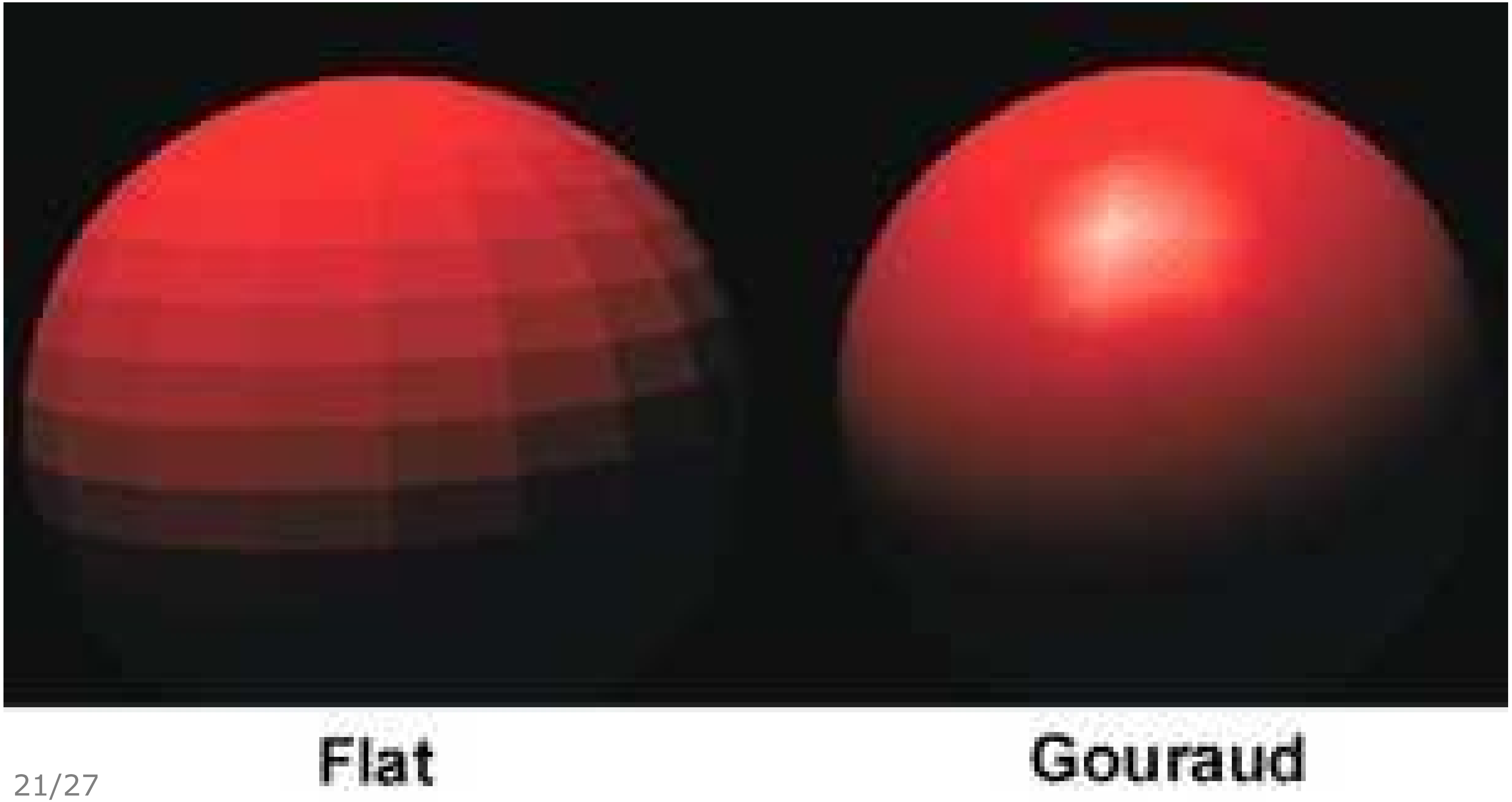
Lighting Calculation

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

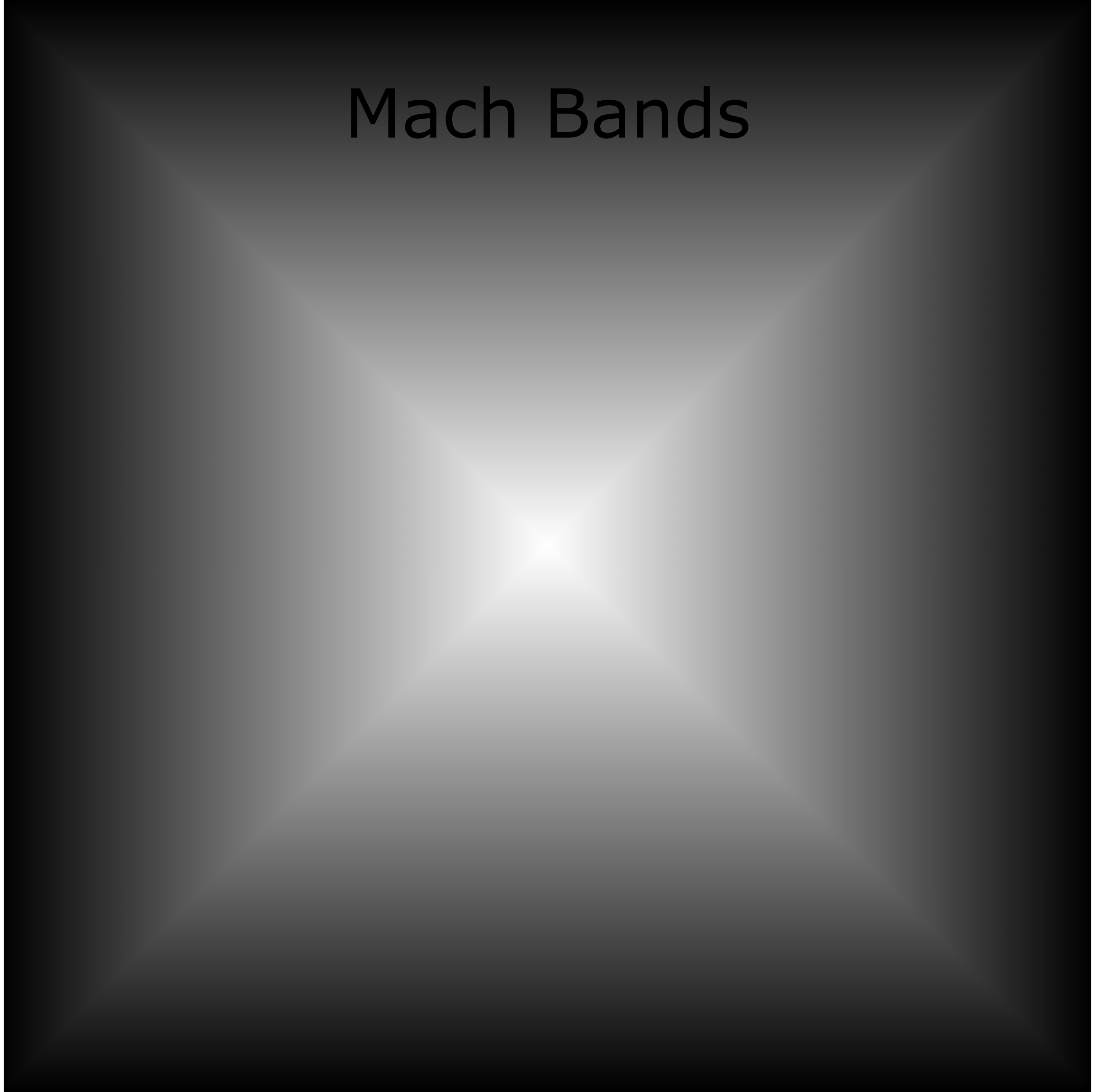
Calculate unique brightness for every pixel
Weighted average of light at each vertex
Based on distance of pixel from vertices
Our old friend: Barycentric Coordinates

BarycentricWeighting

Same Geometry - Different Shading



Mach Bands



Spatial High-Boost Filtering resulting from Neural Lateral Inhibition

Still not ideal

Gouraud is clearly more attractive than flat shading

But there is still much room for improvement:

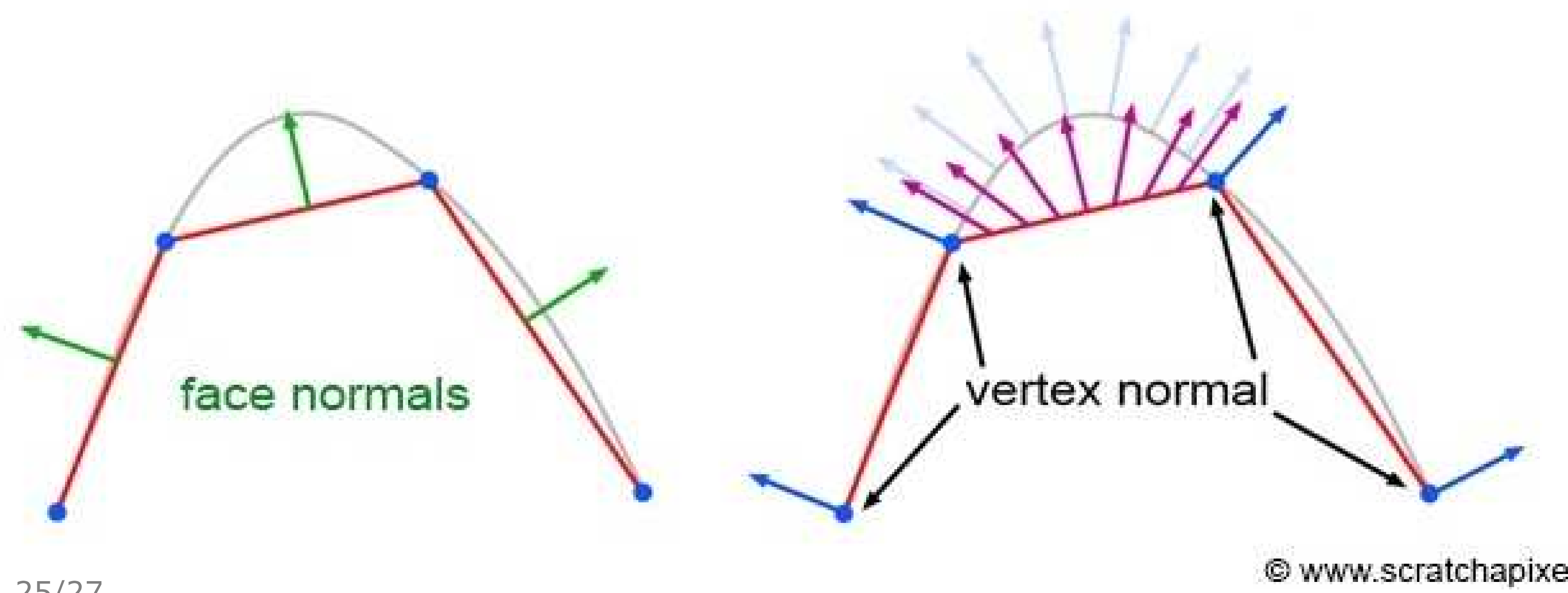
- We still see some rectangular/triangular surfaces
- Perceived Mach bands over-emphasise edges
- Specular highlights can be missed entirely

If only there were a way to achieve
EVEN more elegant shading...

Phong Shading - Even Smoother !

Interpolate *vertex normals* across the entire surface

Only THEN calculate unique brightness - for each pixel

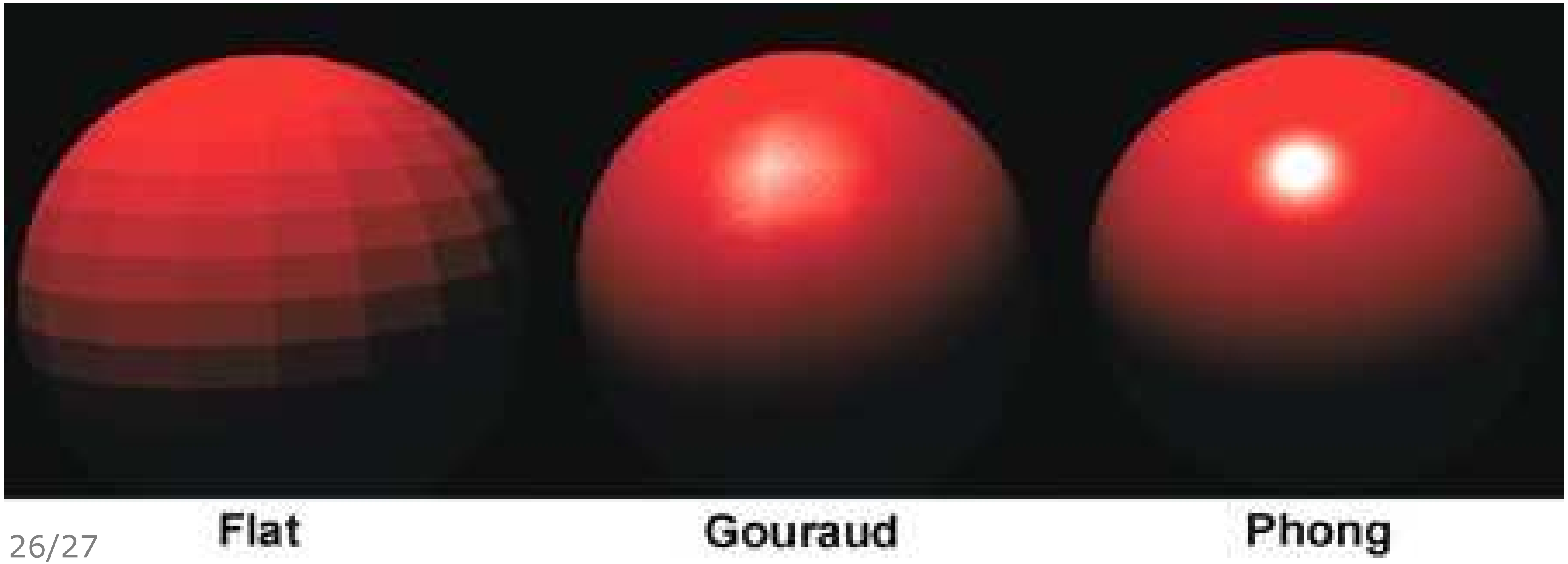


Comparison

Interpolating normals (rather than just brightness)

Produces very much smoother overall object shading

At expense of rendering speed (as you might expect)



A New Model !

Some light effects are hard to see with Cornell box

So an additional model has been provided for testing

Note: this is lower res than the sphere in the slides

(renders faster but also makes bugs easier to spot !)