

MODULE 6 · WEEK 12 · LECTURE 11

Materials & Global Rendering

Light Sources in OpenGL · Material Specification · Shading the Sphere · Global Rendering

BITM3213 — Interactive Computer Graphics

Faculty of Information & Communication Technology (FTMK)

Universiti Teknikal Malaysia Melaka · Sesi 2025/2026

Module 6, continued — Prepared by: Ts. Dr. Hamzah Asyrani Sulaiman

Where we left off



Phong model — ambient + diffuse + specular, summed per pixel



Four vectors — N (normal), L (to light), V (to eye), R (reflected)



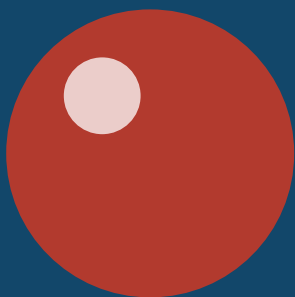
Normal matrix — $\text{mat3}(\text{transpose}(\text{inverse}(\text{model})))$ keeps normals correct



This week's leap Last week: ONE light, ONE object, ONE colour. This week: each surface gets a full MATERIAL, and we light it with directional + point + spot all at once.

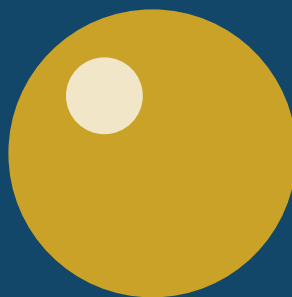
THE BIG IDEA THIS WEEK

Same light. Different material. Completely different look.



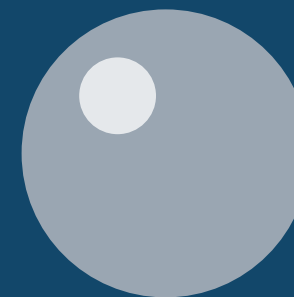
RUBBER

matte · soft · low shininess



GOLD

warm · shiny · medium shininess

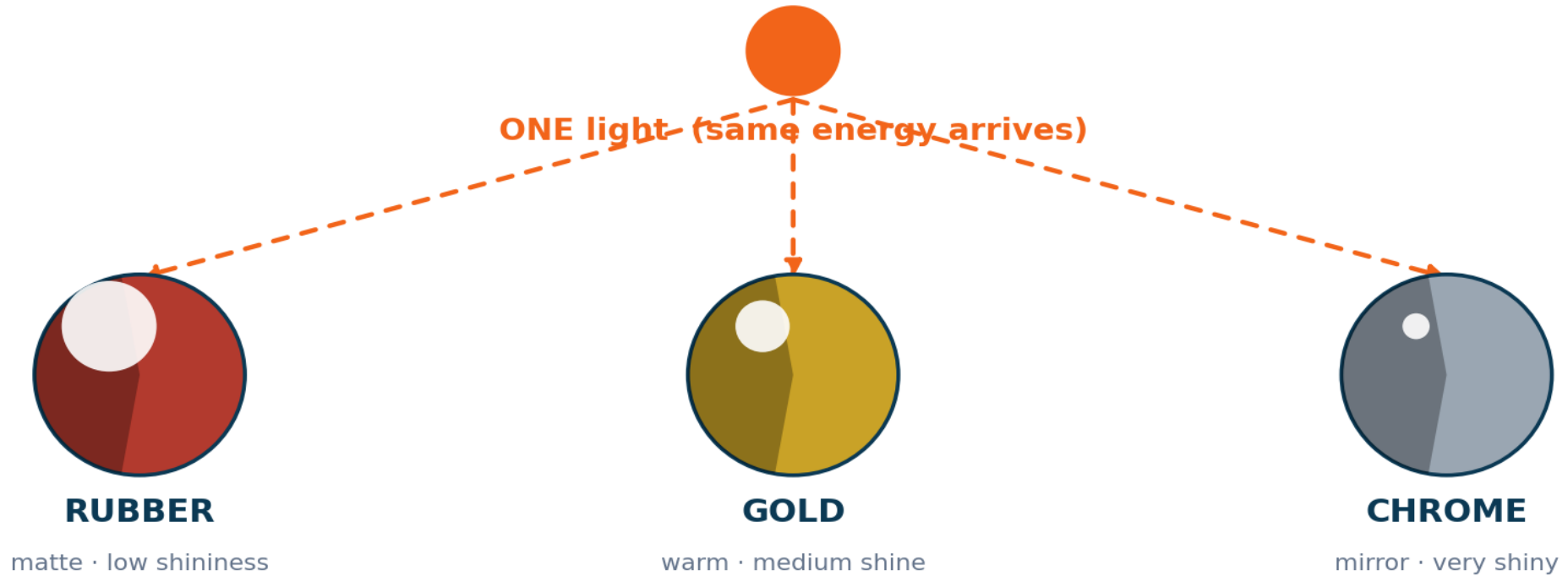


CHROME

mirror-like · very high shininess

Light defines what energy ARRIVES. Material defines how much the surface REFLECTS.

Material vs Light



One sentence to remember Same light + different material = different look. A material is just four numbers per surface: ambient, diffuse, specular reflectivity, and a shininess exponent.

A surface = four reflectivity properties

ambient

how much ambient light it reflects (its colour in shadow)

diffuse

its main colour under direct light — the dominant look

specular

colour & strength of its highlight (grey for plastic, tinted for metal)

shininess

highlight tightness — 10 = matte rubber, 76.8 = chrome

```
struct Material {  
    vec3  ambient;  
    vec3  diffuse;  
    vec3  specular;  
    float shininess;  
};
```

```
// in C++ set per object:  
material.diffuse = ...  
material.shininess = 51.2f;
```

Material values you'll actually use

Material	diffuse (r, g, b)	specular (r, g, b)	shininess
Rubber (red)	0.5, 0.4, 0.4	0.04, 0.04, 0.04	10
Plastic (red)	0.5, 0.0, 0.0	0.7, 0.6, 0.6	32
Gold	0.75, 0.61, 0.23	0.63, 0.56, 0.37	51.2
Silver / metal	0.5, 0.5, 0.5	0.77, 0.77, 0.77	89
Chrome	0.4, 0.4, 0.4	0.77, 0.77, 0.77	76.8



Where these come from These are from the classic OpenGL material table (devernay.free.fr/cours/opengl/materials.html). Notice metals tint their specular; plastics keep it grey. Lab 7 ships rubber, gold and chrome ready to use.

Directional light — the sun

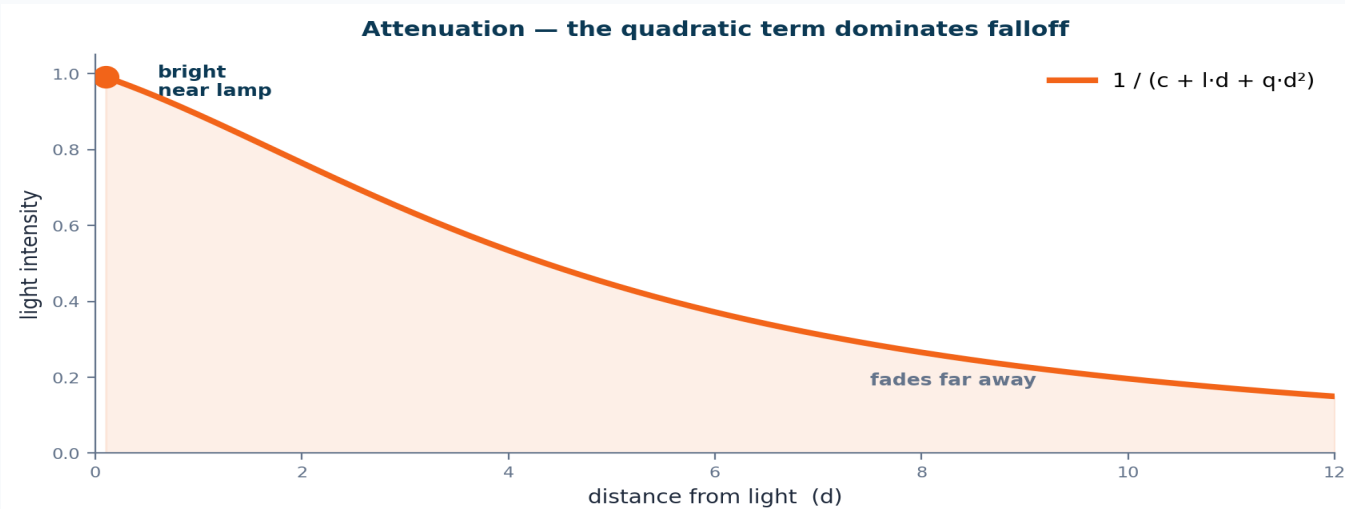
A directional light is infinitely far away, so all its rays are **parallel**. It has a **direction but no position** — perfect for the sun or moon. Distance doesn't matter, so it never fades.

```
struct DirLight {  
    vec3 direction;  
    vec3 ambient;  
    vec3 diffuse;  
    vec3 specular;  
};  
  
// the light direction is negated:  
vec3 lightDir = normalize(-light.direction);
```



Why negate the direction? The struct stores which way the light TRAVELS (e.g. downward from the sky). But the Phong maths needs the direction FROM the surface TOWARD the light. Those are opposite, so we use `normalize(-direction)`. This is Lab 7's `CalcDirLight`.

Point light — a bulb that fades with distance

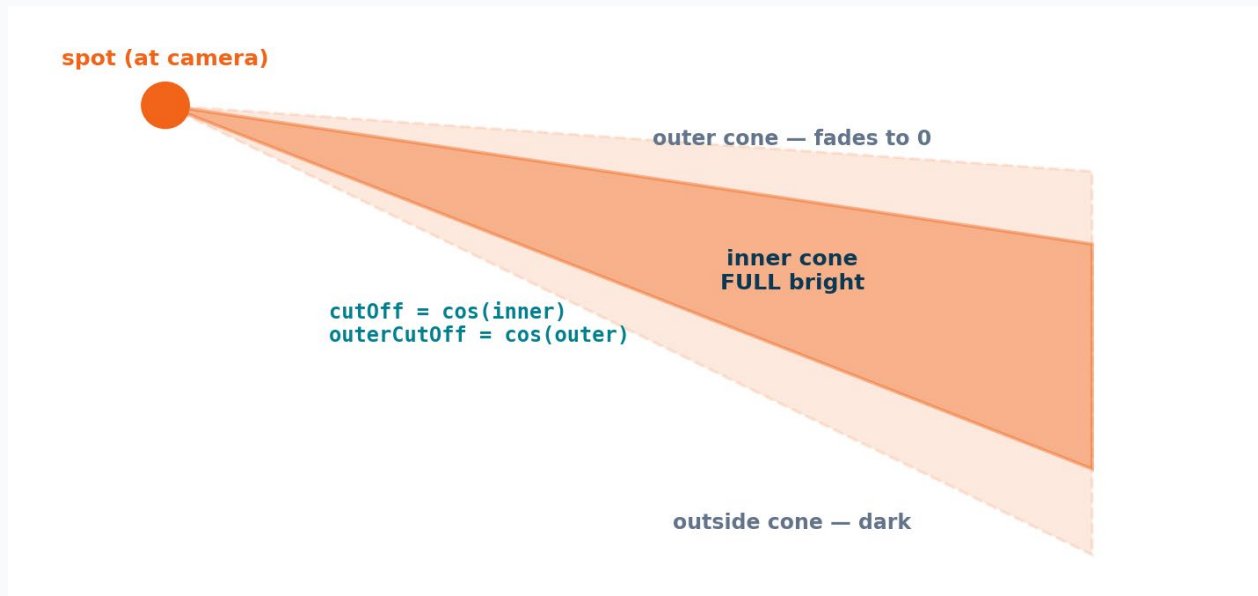


```
float d = length(light.position - fragPos);  
  
float att = 1.0 / (constant + linear*d + quadratic*d*d);  
  
ambient *= att;  diffuse *= att;  specular *= att;
```



Real light obeys inverse-square A point light has a position and radiates everywhere — but it must fade with distance. The quadratic term dominates the falloff. Good mid-range values: constant 1.0, linear 0.09, quadratic 0.032. For a cosy lamp pool, use linear 0.14, quadratic 0.07. This is Lab 7 Task 2.2.

Spotlight — a cone like a torch



```
float theta = dot(L, normalize(-spotDir));

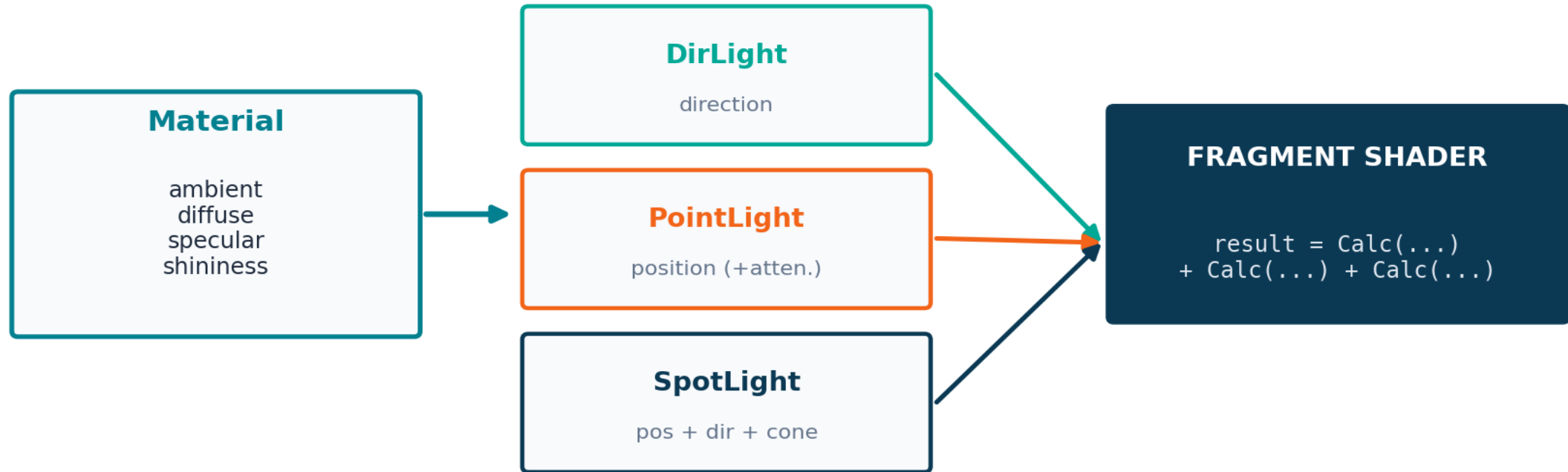
float eps = cutOff - outerCutOff;

intensity = clamp((theta-outerCutOff)/eps, 0, 1);
```



Inner & outer cone A spotlight has a position, a direction and a cone. `cutOff` is the inner cone (full bright); `outerCutOff` is the wider outer cone (fades to zero). We compare COSINES, not angles — it's faster on the GPU, and a bigger cosine means a smaller angle (closer to the centre). In Lab 7 the spotlight sits at the camera and points where you look — a torch effect. This is Lab 7 Task 2.3.

Stack them: one function per light type



The standard pattern One helper per light type — `CalcDirLight`, `CalcPointLight`, `CalcSpotLight` — each returns a colour. `main()` simply ADDS them. Want more point lights? Store them in an array and loop. This is the learnopengl.com structure used in Lab 7.

Why a sphere is the lighting test object

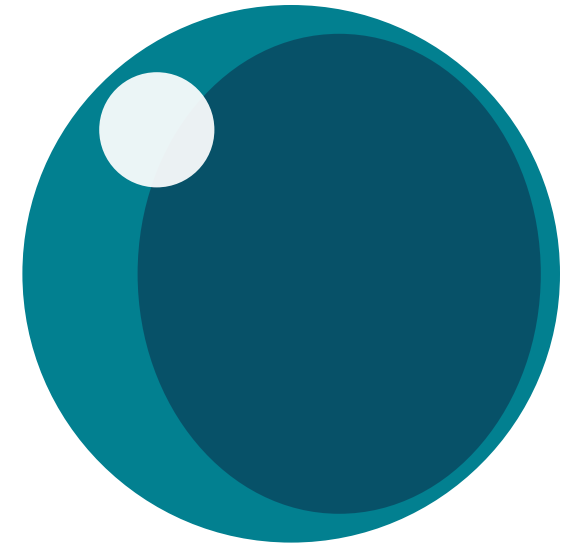
A sphere is the classic surface for testing lighting because its normals point in **every direction at once**. You instantly see the full gradient — a bright lit cap, a smooth diffuse falloff, a dark terminator, and a crisp specular dot — all on one object.

Bright cap — faces the light — diffuse near maximum

Smooth falloff — $\text{dot}(\mathbf{N}, \mathbf{L})$ eases off as the surface curves away

Terminator — the line where light meets shadow

Specular dot — tiny bright highlight where $\mathbf{R}$ points at the eye



normals point everywhere → the full lighting story on one surface

Phong is LOCAL — what are we missing?

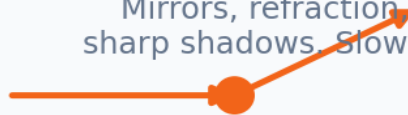
LOCAL (what we do)

Phong per object.
Fast, real-time.
No true shadows/bounce.



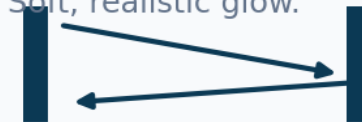
RAY TRACING

Trace rays from eye.
Mirrors, refraction,
sharp shadows. Slower.



RADIOSITY

Diffuse light bounces
between surfaces.
Soft, realistic glow.



Local vs global Phong is a LOCAL model — each surface is lit on its own, ignoring the rest of the scene. That's why it's fast enough for real-time. GLOBAL methods (ray tracing, radiosity) account for light bouncing between surfaces — far more realistic, far more expensive.

Ray tracing vs radiosity

RAY TRACING

- Trace rays from the eye into the scene
- Follows reflections & refractions
- Sharp shadows, mirrors, glass
- Per-ray — expensive, view-dependent
- Now real-time on RTX GPUs

RADIOSITY

- Simulates diffuse light bouncing between surfaces
- Soft, realistic colour bleeding
- Great for interiors & architecture
- View-independent — precompute once
- Matte surfaces only (no mirrors)

How Lab 7 puts it all together

1

Define materials on the CPU — a `CpuMaterial` struct holds ambient/diffuse/specular/shininess presets

2

Set lights once per frame — fill `dirLight` + `pointLight` (+ `spot`) uniforms before drawing

3

Loop over objects — for each cube: `setMaterial(...)` then upload its model matrix

4

Draw — same VAO, new material — one cube mesh, drawn several times with different materials

5

Draw a lamp marker — a tiny bright cube shows where the point light sits

Sending a material to the GPU

```
struct CpuMaterial {  
    glm::vec3 ambient, diffuse, specular;  
    float shininess;  
};  
  
void setMaterial(unsigned int prog, CpuMaterial m){  
    glUniform3fv(loc("material.ambient"), 1, &m.ambient);  
    glUniform3fv(loc("material.diffuse"), 1, &m.diffuse);  
    glUniform3fv(loc("material.specular"), 1, &m.specular);  
    glUniform1f (loc("material.shininess"), m.shininess);  
}  
  
// per object, before glDrawArrays:  
setMaterial(shader, gold);
```



The key habit Call `setMaterial()` INSIDE the per-object loop, just before each draw. Forget it and every object reuses the last material — the classic Lab 7 'all objects look identical' bug.



LAB 7 — MATERIALS & MULTIPLE LIGHTS

What you'll build this week

Part 1

Three cubes — rubber, gold, chrome — under a sun + an orbiting lamp

Task 2.1

Swap materials live with keys 1 / 2 / 3 (plastic / metal / rubber)

Task 2.2

Add attenuation so the point light fades with distance

Task 2.3

Build a spotlight torch that follows where you look

Task 2.4–2.5

Toggle each light (F / G); add an emissive self-glow cube

Part 3

Independent: a NIGHT SCENE — moonlight + 2 streetlamps + torch, 5+ materials

Expected output: three cubes that react differently to the same lights — matte rubber, warm gold, mirror chrome.

Key terms — must-know for the exam

Material

four reflectivity values per surface (ambient/diffuse/specular/shininess)

Directional light

parallel rays, a direction but no position (the sun)

Point light

has a position, radiates out, fades with distance

Spotlight

a point light limited to a cone (inner + outer cutoff)

Attenuation

light fading with distance: $1/(c + l \cdot d + q \cdot d^2)$

Global rendering

light transport across the whole scene (ray tracing, radiosity)

Self-check questions

1

What four properties make up a material, and which one sets the highlight size?

2

How does a directional light differ from a point light?

3

Write the attenuation formula and say which term dominates far from the light.

4

What are the inner and outer cutoffs of a spotlight for?

5

Phong is a 'local' model — what does global rendering add that Phong cannot?

Recap — what to take away

- ✓ Light = energy arriving; material = how much each surface reflects.
- ✓ A material is four numbers: ambient, diffuse, specular, shininess.
- ✓ Three light types: directional (sun), point (bulb), spotlight (torch).
- ✓ Point & spot lights fade with distance via attenuation $1/(c+l\cdot d+q\cdot d^2)$.
- ✓ Stack lights by summing one helper function per type.
- ✓ Phong is LOCAL; ray tracing & radiosity are GLOBAL (whole-scene).

Module 7 — Hierarchical & Object-Oriented Graphics

With lighting mastered, Module 7 turns to **structure**: symbols & instances, hierarchical models, trees & traversal, and scene graphs — the parent-child transforms behind robot arms and solar systems (callback to your Lab 3 solar system). Your lit, materialled objects become nodes in a scene.

References

- Gordon & Clevenger (2024). Computer Graphics Programming in OpenGL with C++, 3rd Ed. Mercury Learning.
- Angel & Shreiner (2014). Interactive Computer Graphics, 8th Ed. Pearson.
- de Vries (2020). Learn OpenGL — Materials, Lighting Maps & Multiple Lights. learnopengl.com.
- Devernay OpenGL material table — devernay.free.fr/cours/opengl/materials.html

Module 6 complete — you can now light and texture a believable 3D world.