

BITM3213 · INTERACTIVE COMPUTER GRAPHICS

Module 2

Graphics Programming

The Sierpinski Gasket · the OpenGL API · primitives, attributes & colour

Faculty of Information & Communication Technology (FTMK)

Universiti Teknikal Malaysia Melaka (UTeM)

Session 2025/2026

Learning Outcomes



Describe the Sierpinski Gasket

Explain how recursive subdivision turns one triangle into a fractal pattern.



Explain what the OpenGL API is

Understand OpenGL as a state machine that issues drawing commands to the GPU.



Identify primitives & attributes

Recognise points, lines and triangles, and the data each vertex carries.



Apply the RGB colour model

Use additive colour and understand how the GPU interpolates it across a face.

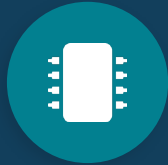
WHY THIS MODULE MATTERS

From one triangle to a whole world of shapes



Infinite detail

A simple rule, repeated, creates endless complexity — the heart of procedural graphics.



Talk to the GPU

Every shape you draw is really a list of vertices handed to the graphics card.



Colour as data

Colour isn't painted on — it travels with each vertex and is blended by hardware.



Foundation for Lab 2

Today's ideas become the shader & uniform code you'll write this week.

The Sierpinski Gasket

Depth 0 • 1 triangle



Depth 1 • 3 triangles



Depth 2 • 9 triangles



Depth 3 • 27 triangles



A classic fractal:

start with one filled triangle, find the midpoint of each side, remove the middle triangle, and repeat on the three that remain.

Each level triples the triangle count ($1 \rightarrow 3 \rightarrow 9 \rightarrow 27 \dots$) yet uses the same simple rule.

💡 A bit of history

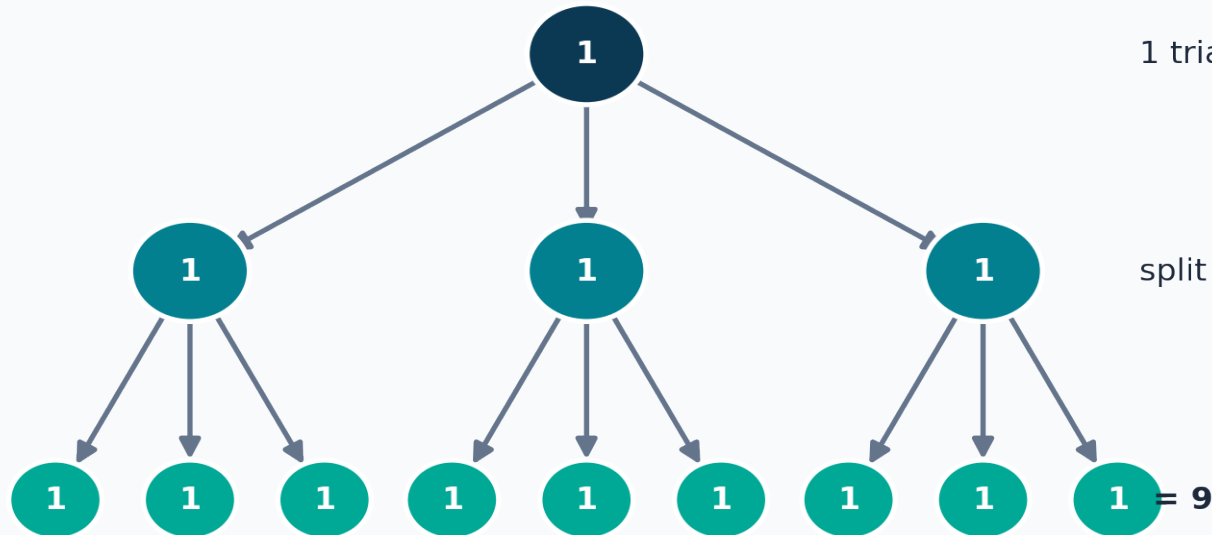
Named after Waclaw Sierpiński, who described it in 1915. It became an icon of fractal geometry — a shape with infinite edge detail but zero area.

Why we use it here

It's the perfect excuse to learn how to feed vertices to OpenGL — small, visual, and instantly rewarding.

Recursion: triangles inside triangles

Recursion: every triangle becomes three smaller triangles



1 triangle

split into 3

The gasket is built by a function that calls itself.

Given one triangle, it:

- computes the three edge midpoints
- makes three corner triangles
- calls itself on each one
- stops at the chosen depth (the base case)

Analogy

Like a set of Russian dolls — open one and find three smaller ones inside, each opening into three more, until you decide to stop.

The OpenGL API

OpenGL is not a library you install — it's a specification.

It defines a set of commands (an Application Programming Interface) that your program uses to tell the GPU what to draw. Your graphics driver provides the actual implementation.

Key traits:

- cross-platform & cross-vendor
- language-neutral (we call it from C++)
- functions are prefixed `gl...` (e.g. `glDrawArrays`)

💡 Analogy: a restaurant menu

OpenGL is the menu — a fixed list of things you can order. You don't cook; you ask the kitchen (the GPU + driver). Different kitchens, same menu.

A few commands you'll meet

`glGenVertexArrays` / `glGenBuffers` — create GPU storage

`glBufferData` — upload vertex data

`glUseProgram` — activate a shader

`glDrawArrays` — issue the draw

OpenGL is a state machine

OpenGL is a state machine — you set state, then draw



💡 What 'state' means

You set things once (which buffer is bound, which shader is active, the clear colour) and they STAY set. Every draw call quietly uses whatever state is currently in effect — so the order of your commands matters.

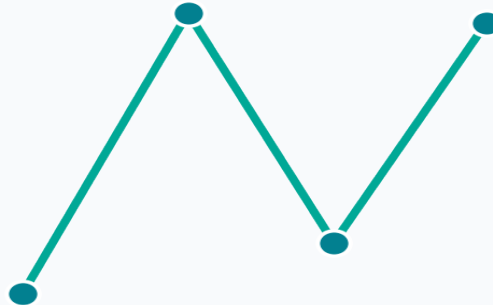
Primitives: the shapes OpenGL can draw

GL_POINTS



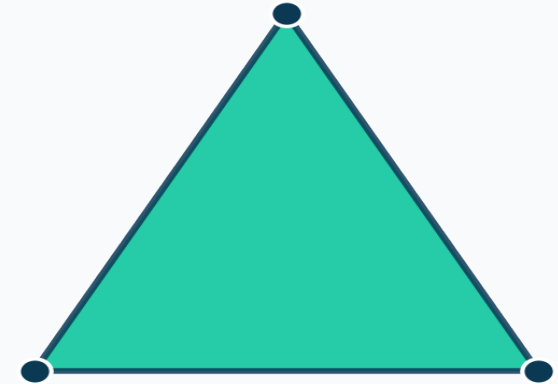
each vertex = a dot

GL_LINES



vertices joined by edges

GL_TRIANGLES

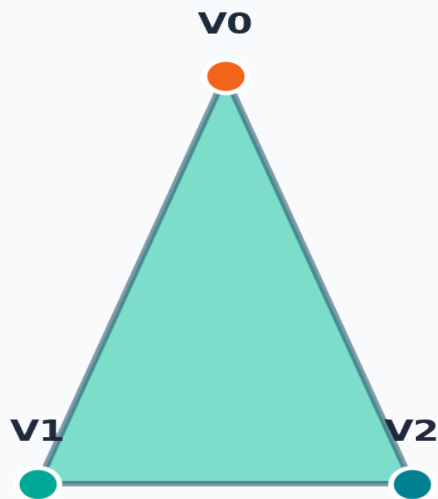


3 vertices = a filled face

Everything — every model, every fractal — is ultimately built from these basic primitives. **The triangle is king: GPUs are optimised to fill triangles incredibly fast, which is why even curved surfaces are approximated with thousands of them.**

Vertex attributes: data that rides along

Every vertex carries attributes — here: position + colour



	x	y	z	r	g	b
v0	0.x	0.x	0.x	1	0	0
v1	0.x	0.x	0.x	0	1	0
v2	0.x	0.x	0.x	0	0	1

position attribute

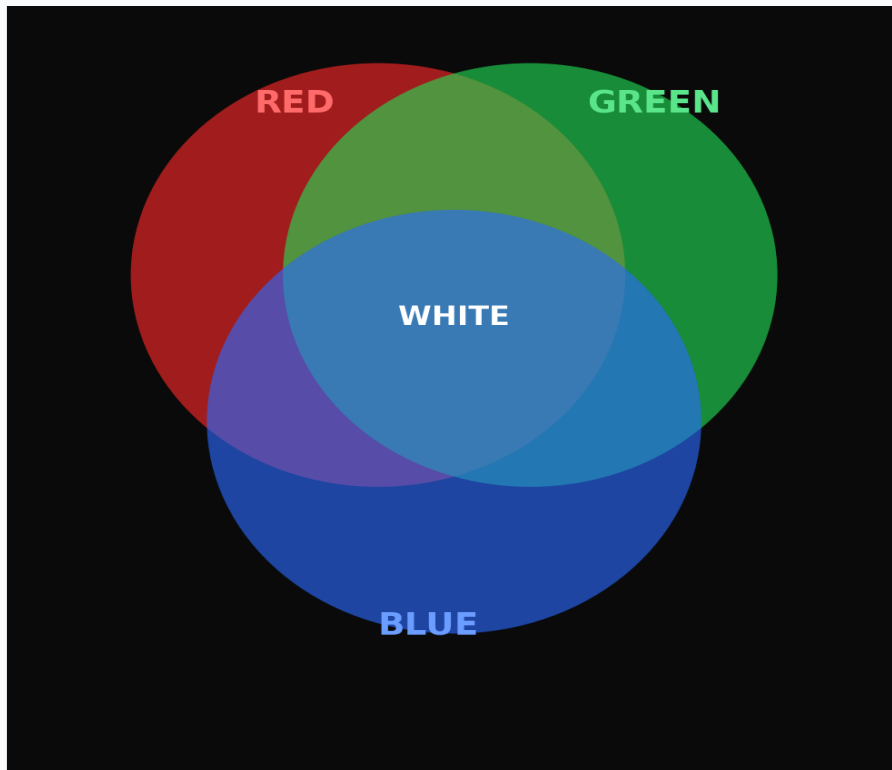
colour attribute

💡 Position is not the only attribute

Each vertex can carry extra data — colour here, and later texture coordinates and normals. These attributes are read by the vertex shader, one vertex at a time.

Colour: the RGB model

RGB is additive — mixing light, not paint



Screens emit light, so colour is additive.

Every colour is a mix of three channels — Red, Green, Blue — each from 0.0 to 1.0:



red

(1, 0, 0)



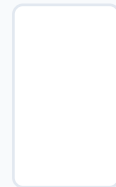
green

(0, 1, 0)



blue

(0, 0, 1)



white

(1, 1, 1)



black

(0, 0, 0)



yellow

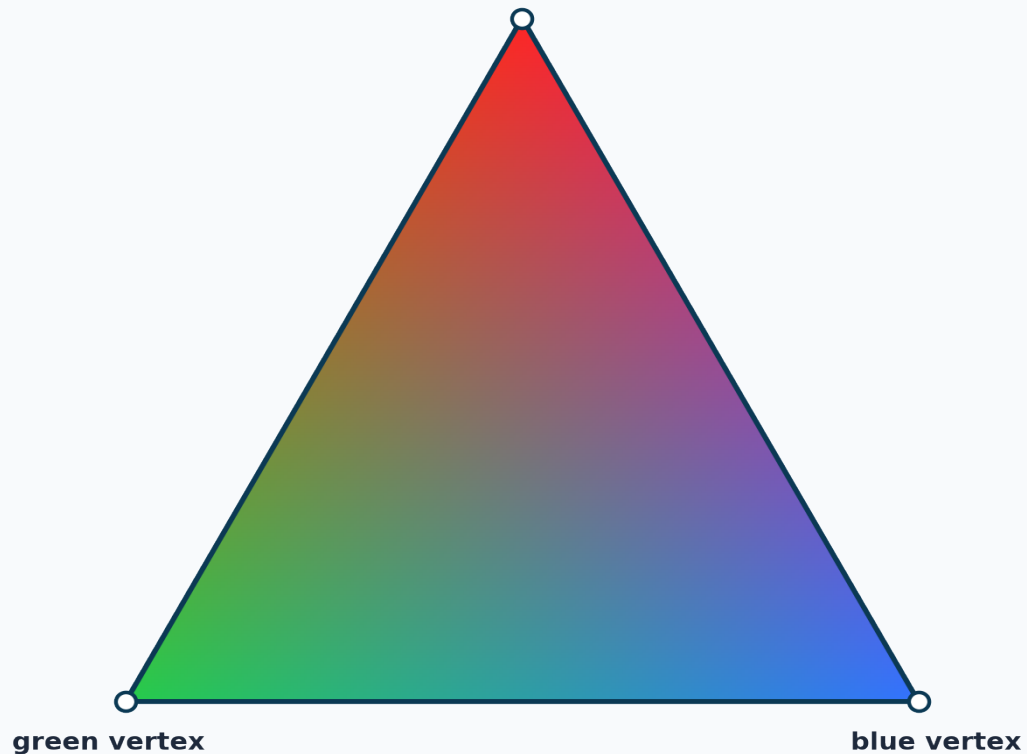
(1, 1, 0)

💡 Paint vs light

Mixing paint is subtractive (more pigment → darker). Mixing light is additive — red + green + blue at full strength gives white.

How the GPU blends colour

The GPU blends vertex colours across every pixel
red vertex



You only set colour at the three vertices.

The GPU fills every pixel in between by smoothly blending those vertex colours — this is called interpolation (specifically, Gouraud-style across the face).

It happens automatically, in hardware, for free — between the vertex shader and the fragment shader.



You've seen this already

This is exactly why your Lab 1 triangle had a smooth rainbow — three corner colours, blended across the whole surface by the GPU.

A first look at viewing



Parallel projection

Lines stay parallel; no shrinking with distance. Like an engineering blueprint — great for measuring, not realistic.



Perspective projection

Distant things look smaller; parallel lines converge. Like a photograph or the human eye — what we'll use for 3D scenes.

💡 Callback to Module 1: the pinhole camera

Remember the pinhole model? Perspective projection is the maths that recreates it — rays from the scene converging to a single eye point. We'll build the full camera in Module 5 and Lab 4.

Control functions & the render loop

Control functions: the GLFW window & render loop



render loop — repeats every frame until window closes

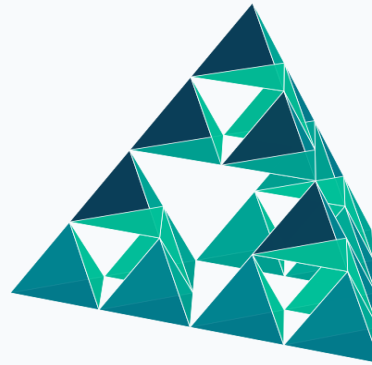
Modern framing: we use GLFW (not legacy GLUT) to create the window, read input and run the loop. Each frame we clear, draw, and swap buffers — over and over until the user closes the window.

From the 2D gasket to the 3D gasket

**2D Gasket
(triangle subdivision)**



**3D Gasket
(tetrahedron subdivision)**



Same idea, one dimension up.

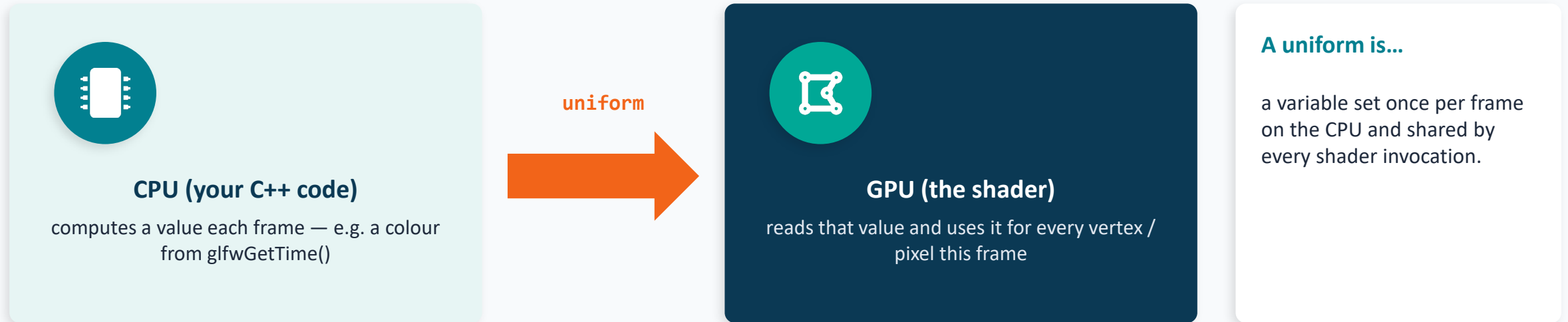
Swap the triangle for a tetrahedron (4 faces). Subdivide it the same recursive way and you get the 3D Sierpinski Gasket — built from many small triangles.

The recursion logic barely changes — only the geometry does.

💡 **Polygons → triangles**

Any polygon a GPU draws is broken into triangles first. The gasket skips the middle-man — it's triangles all the way down.

The bridge to Lab 2: uniforms



💡 What you'll do in Lab 2

You'll create a uniform colour, drive it with glfwGetTime() using sin(), and watch your triangle cycle smoothly through the rainbow — your first real CPU-to-GPU animation.

Lab 2 — Shaders & GLSL · Lab 3 — the Gasket

1

Re-run your Lab 1 triangle Start from the working RGB triangle — same setup, no new libraries.

2

Add a uniform to the shader Declare a uniform `vec4` colour the fragment shader will use.

3

Animate it with time Drive the colour with `sin(glfwGetTime())` for a smooth RGB cycle.

4

Guided modifications Pulse the size, slide it sideways, split colour zones, toggle with SPACE.

5

Independent task (Lab 3) Generate the Sierpinski Gasket by recursion — your assessed challenge.

Key Terms

Fractal

A self-similar shape with detail at every scale (e.g. the Sierpinski Gasket).

Recursion

A function that calls itself, with a base case that stops it.

API

A defined set of commands a program uses to talk to a system — here, the GPU.

Primitive

A basic drawable: point, line or triangle.

Attribute

Per-vertex data (position, colour, ...) read by the vertex shader.

Interpolation

Smooth blending of vertex values across a primitive by the GPU.

TEST YOURSELF

Self-Check Questions

Q1

State the construction rule of the Sierpinski Gasket in your own words.

Q2

How many triangles are in a gasket of depth 4? (Hint: 3^n)

Q3

Why does a recursive function need a base case?

Q4

Is OpenGL a library you install? Explain what it really is.

Q5

What colour is RGB (1, 1, 0) on screen — and why isn't it black?

Q6

Difference between a vertex attribute and a uniform?

Recap



The Sierpinski Gasket is a fractal built by recursive triangle subdivision.



Recursion = a function calling itself, stopped by a base case.



OpenGL is a cross-platform specification — a state machine of gl... commands.



Everything is drawn from primitives; the triangle is the workhorse.



Vertices carry attributes (position, colour ...) read by the vertex shader.



RGB is additive; the GPU interpolates vertex colours across each face.

LOOKING AHEAD

What's next



Module 3 — Input & Interaction

Interaction models, input devices, event-driven input with GLFW callbacks
— leading into the camera you'll build in Lab 4.

Questions? Let's build something.

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 — learnopengl.com

learnopengl.com · glad.dav1d.de · github.com/g-truc/glm