

Viewing

Classical & Computer Viewing · Positioning the Camera · Simple Projections · Projections in OpenGL

BITM3213 — Interactive Computer Graphics

Lecture 8 · Session 2025/2026 · Ts. Dr. Hamzah Asyrani Sulaiman · FTMK, UTeM

THIS WEEK YOU WILL BE ABLE TO

Learning Outcomes

1

Distinguish classical viewing from computer (OpenGL) viewing

2

Position a camera in a 3D scene using the view matrix

3

Explain `glm::lookAt()` — eye, target, and up vector

4

Describe simple projections: parallel vs perspective

5

Build a projection in OpenGL with `glm::perspective()`

Why does VIEWING matter?

Modelling decides WHAT is in the world. Viewing decides WHAT YOU SEE of it.



Games

First-person & third-person cameras flying through levels



CAD / Engineering

Orthographic top/front/side views with true measurements



Film & VFX

Virtual cameras with lenses, focal length, depth



Simulation

Your research — orbiting and fly-through views of data

Classical vs Computer Viewing

CLASSICAL VIEWING

(drafting / by hand)

- Object on paper
- Fixed standard views
- Parallel lines stay parallel
- One projection at a time

same idea,
automated



COMPUTER VIEWING

(OpenGL pipeline)

- Camera placed anywhere
- View + projection matrices
- Any angle, any lens
- GPU does it per frame

💡 Same goal, new tools

Draughtsmen used fixed, standard views (front, top, isometric). OpenGL keeps the SAME idea but lets you place a camera anywhere and pick any projection — and recomputes it every single frame on the GPU.

The Three Ingredients of a View



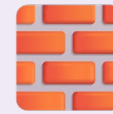
WHERE is the camera?

Its position and the direction it points — the VIEW matrix (`glm::lookAt`).



WHAT lens?

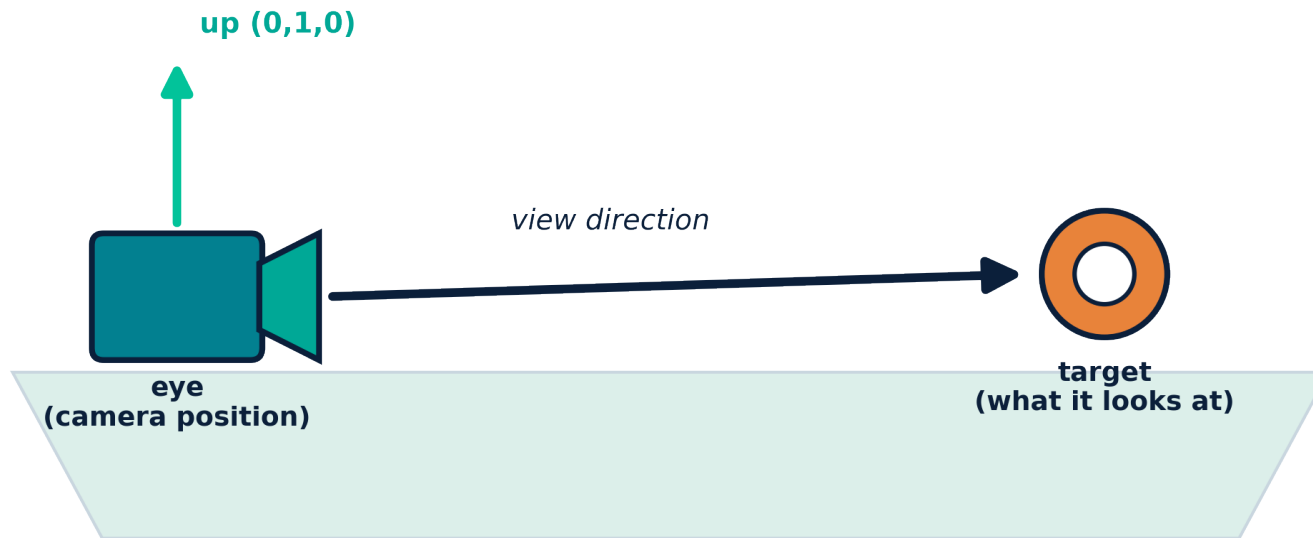
Parallel or perspective, and how wide — the PROJECTION matrix (`glm::ortho` / `perspective`).



WHAT is in front?

Which surfaces are visible vs hidden — handled by the depth test (Week 10).

Positioning the Camera: lookAt



`glm::lookAt(eye, target, up)` → the VIEW matrix

Three inputs

- eye — where the camera sits
- target — the point it aims at
- up — which way is “up” (usually 0,1,0)

```
glm::mat4 view =  
    glm::lookAt(  
        eye, target, up);
```

lookAt in Practice

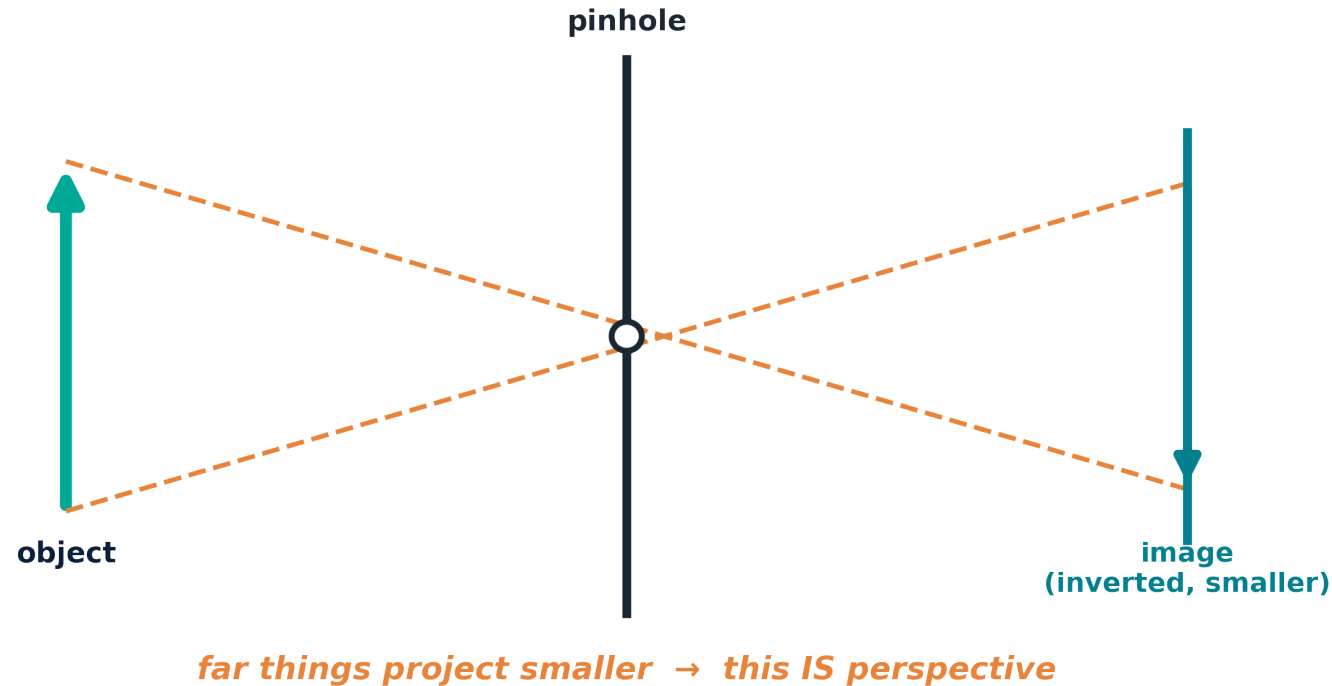
```
// VIEW: camera at (0,0,8) looking at origin
glm::mat4 view = glm::lookAt(
    glm::vec3(0.0f, 0.0f, 8.0f), // eye
    glm::vec3(0.0f, 0.0f, 0.0f), // target
    glm::vec3(0.0f, 1.0f, 0.0f)); // up

glUniformMatrix4fv(viewLoc, 1,
    GL_FALSE, glm::value_ptr(view));
```

💡 Eye must be **BEHIND** the scene

In Lab 4 the cubes sit around $z = 0$ and -6 . The eye is at $+8$ looking toward $-Z$, so it sees them. Put the eye in front (negative z) and you stare into empty space — the classic “black screen” bug in the troubleshooting table.

Simple Projections: the Pinhole



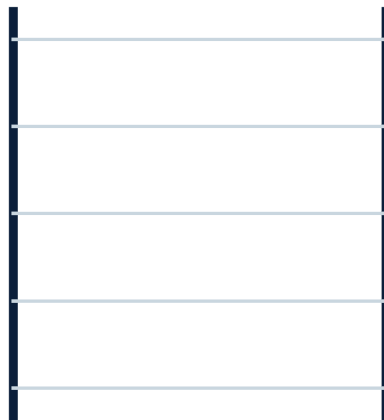
💡 Callback to Module 1

Remember the camera obscura? Light through a tiny hole projects an inverted image. “Far objects look smaller” is not a trick we add — it falls out naturally from rays through a single point. Perspective projection is just the maths of this pinhole.

Parallel vs Perspective

PARALLEL

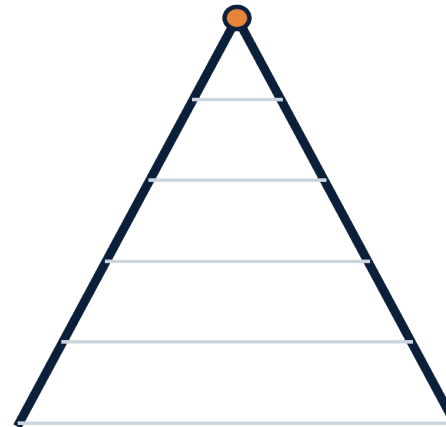
"blueprint"



rails never meet
sizes preserved

PERSPECTIVE

"photograph"
vanishing point



rails converge
far = small

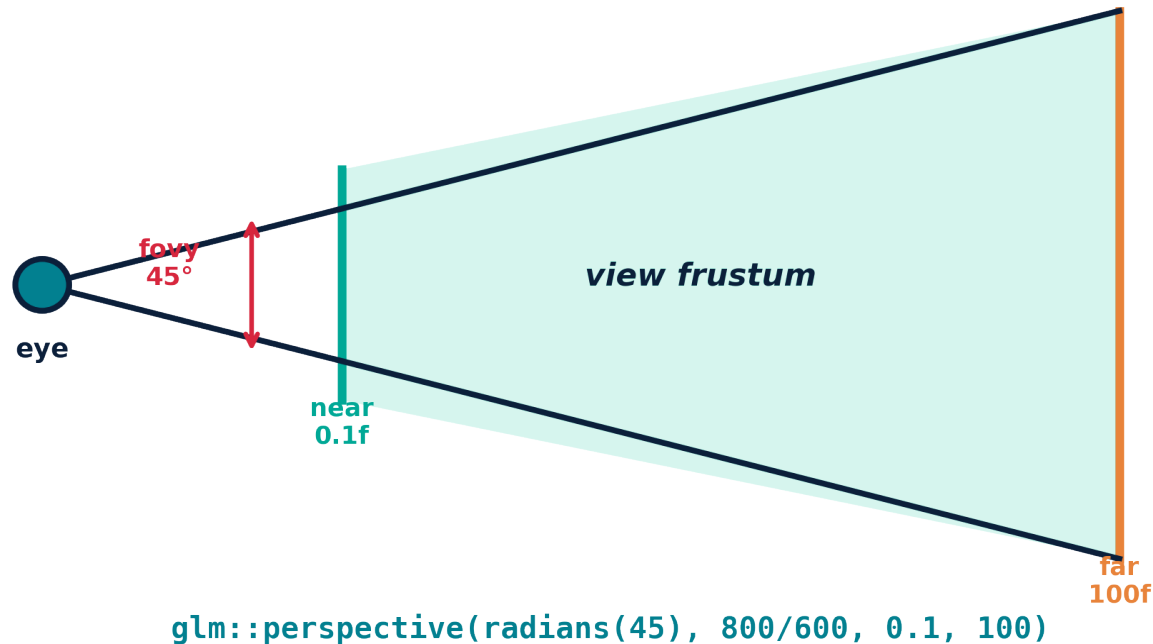
Parallel = "blueprint"

Sizes preserved, lines stay parallel. Great for CAD and measuring. `glm::ortho()`.

Perspective = "photograph"

Far things shrink toward a vanishing point. Looks real. `glm::perspective()`.

Projections in OpenGL: the Frustum



The four arguments

- fovy — vertical field of view
- aspect — width / height
- near — closest visible (0.1f)
- far — farthest visible (100f)

```
glm::perspective(  
    radians(45.0f),  
    800.0f/600.0f,  
    0.1f, 100.0f);
```

Field of View (fovy)

20°

Telephoto

Narrow, zoomed in. Flattens depth. Like a sniper scope.

45°

Normal

The Lab 4 default. Natural, close to human central vision.

90°

Wide angle

Sees a lot, but edges distort. Fast / action feel.

💡 Lab 4 link — scroll-wheel zoom

In Lab 4 Task 2.5 the scroll wheel changed fovy (clamped 1°–75°) and fed it back into `glm::perspective` each frame. Smaller fovy = zoom in. That IS changing the lens live.

Where View & Projection Sit: MVP



```
gl_Position = projection * view * model * vec4(aPos, 1.0); // read RIGHT to LEFT
```

Common Pitfalls



Degrees instead of radians

`glm::perspective` and `rotate` want RADIANS. Always wrap angles in `glm::radians()`.



Eye in front of the scene

If the eye looks away from your objects you get a black screen. Keep the eye behind, looking toward -Z.



near = 0.0f

Near plane must be > 0 (e.g. `0.1f`). Zero causes depth precision to collapse.



Aspect ratio hard-coded

If you resize the window but keep `800/600`, the scene stretches. Recompute aspect from the real size.

From slides to your screen

- 1 Build the VIEW** — `glm::lookAt(eye, target, up)` – place and aim the camera
- 2 Build the PROJECTION** — `glm::perspective(radians(fov), aspect, 0.1, 100)`
- 3 Send both to the shader** — `glUniformMatrix4fv` with `glm::value_ptr`, `GL_FALSE`
- 4 Multiply in order** — `gl_Position = projection * view * model * vec4(pos,1)`

Lab 4 = camera + perspective live. Lab 6 = these matrices applied to a lit 3D scene.

Key Terms

View matrix

Transforms world $\rightarrow$ camera space. Built by lookAt.

Projection matrix

Transforms camera $\rightarrow$ clip space. ortho or perspective.

Frustum

The truncated pyramid of space the camera can see.

fovy

Vertical field of view — the lens angle.

Near / Far plane

Closest and farthest visible distances.

Vanishing point

Where parallel lines appear to meet in perspective.

Self-Check

Q1

What three vectors does `glm::lookAt()` take, and what does each do?

Q2

In projection * view * model, why do we read right to left?

Q3

Does a smaller fovy zoom IN or OUT? Why?

Q4

Which projection preserves true sizes — parallel or perspective?

Q5

Why must the near plane be greater than zero?

Recap

- ✓ Classical viewing = fixed standard views; computer viewing generalises it
- ✓ A view = WHERE the camera is + WHAT lens it uses + WHAT is visible
- ✓ VIEW matrix comes from `glm::lookAt(eye, target, up)`
- ✓ Projection turns 3D into 2D: parallel (blueprint) vs perspective (photo)
- ✓ `glm::perspective(fovy, aspect, near, far)` builds the frustum
- ✓ Order in the shader: `projection * view * model` (right to left)

Next: Week 10 — Viewing (cont.)



Hidden-surface removal — The z-buffer / depth test in detail



Walking through a scene — FPS camera — yaw, pitch, WASD



Projection matrices — Parallel & perspective, the actual maths



Projections & shadows — Projecting an object onto the ground

References

Angel & Shreiner (2014)

Interactive Computer Graphics, 8e — Ch. on Viewing

Gordon & Clevenger (2024)

CG Programming in OpenGL with C++, 3e

learnopengl.com

Camera • Coordinate Systems