

Viewing

(continued)

Hidden-Surface Removal · Walking Through a Scene · Projection Matrices · Projections & Shadows

BITM3213 — Interactive Computer Graphics

Lecture 9 · Session 2025/2026 · Ts. Dr. Hamzah Asyrani Sulaiman · FTMK, UTeM

Where We Left Off

VIEW matrix

`glm::lookAt(eye, target, up)` — placed and aimed the camera

PROJECTION matrix

`glm::perspective(fovy, aspect, near, far)` — chose the lens

The order

`gl_Position = projection * view * model * vec4(pos,1)`

*Today we add the last ingredient — what is **VISIBLE** — and make the camera move.*

Which surface wins?

When two triangles cover the same pixel, the GPU must choose. Get it wrong and far objects paint OVER near ones.



Without depth handling

Draw order decides — late shapes cover earlier ones even if they are behind

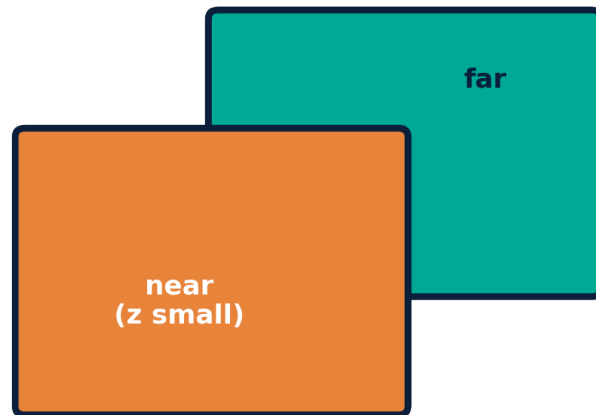


With the z-buffer

Nearest fragment always wins — order no longer matters

Hidden-Surface Removal

DEPTH TEST (z-buffer)



GPU keeps the nearest pixel
per fragment — order-independent

PAINTER'S (sort first)



sort objects back-to-front by hand
— breaks when shapes intersect

💡 z-buffer beats sorting

The painter's algorithm sorts shapes back-to-front by hand — and breaks the moment two shapes intersect. The z-buffer works per PIXEL: it stores the nearest depth seen so far and rejects anything farther. Order-independent and free on the GPU.

How the z-buffer decides

For every pixel: keep the smallest depth



```
glEnable(GL_DEPTH_TEST);    glClear(... | GL_DEPTH_BUFFER_BIT);
```

Two buffers, every frame

Colour buffer holds the pixel colours. Depth buffer holds a 0–1 distance per pixel. A fragment is kept only if its depth is SMALLER (nearer) than what is already there. Clear BOTH each frame.

Depth Test in Code

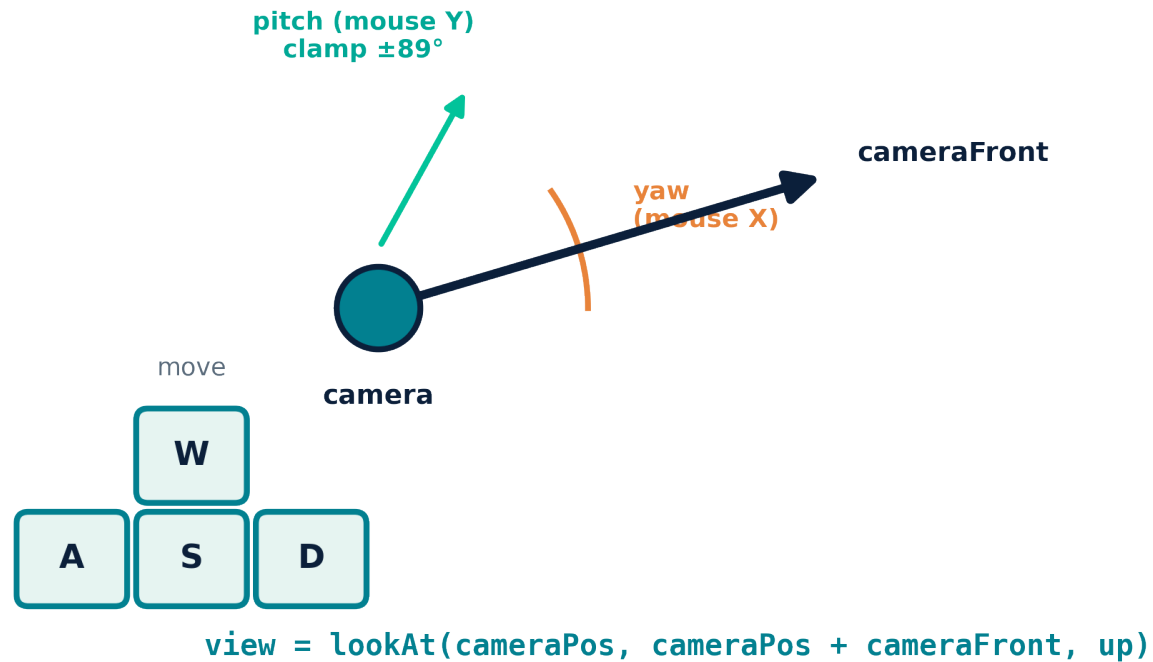
```
// once, at setup
glEnable(GL_DEPTH_TEST);

// every frame, before drawing
glClear(
    GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT); // both!
```

💡 Forget the clear → chaos

In 2D (Lab 1–2) you only cleared the colour buffer. In 3D you must also clear the depth buffer each frame. Skip it and stale depths from the previous frame make cubes flicker or disappear — exactly the symptom in the Lab 4 troubleshooting table.

Walking Through a Scene



FPS camera = 2 ideas

- WASD shifts cameraPos along front & right vectors
- Mouse changes yaw (look left/right) and pitch (up/down)

Rebuild the view each frame

Because cameraPos and cameraFront change, you call lookAt every frame — the view is no longer constant.

Yaw, Pitch & the $\pm 89^\circ$ Clamp

```
yaw += xoffset * sensitivity;
pitch += yoffset * sensitivity;

// stop the world flipping over
if (pitch > 89.0f) pitch = 89.0f;
if (pitch < -89.0f) pitch = -89.0f;

dir.x = cos(radians(yaw))*cos(radians(pitch));
dir.y = sin(radians(pitch));
dir.z = sin(radians(yaw))*cos(radians(pitch));
cameraFront = normalize(dir);
```

yaw starts at -90°

At yaw 0 the camera faces +X. Start at -90° so it looks down -Z toward the scene.

Clamp pitch $\pm 89^\circ$

At $\pm 90^\circ$ the front vector aligns with up and lookAt breaks — the world flips.

firstMouse guard

On the very first mouse event, snap lastX/lastY to avoid a huge jump.

Why deltaTime?

The problem

If you move the camera a fixed amount per frame, a fast PC (300 FPS) moves 5× faster than a slow PC (60 FPS). The same key feels different on every machine.

```
float now = glfwGetTime();
deltaTime = now - lastFrame;
lastFrame = now;

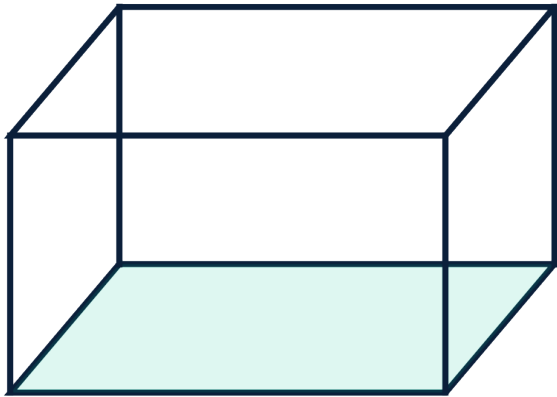
float speed =
    3.0f * deltaTime; // units / second
cameraPos += speed * cameraFront;
```

💡 Same idea returns in Lab 9

Multiplying movement by deltaTime makes speed time-based, not frame-based. The exact same trick drives gravity and bouncing in the Lab 9 physics simulation.

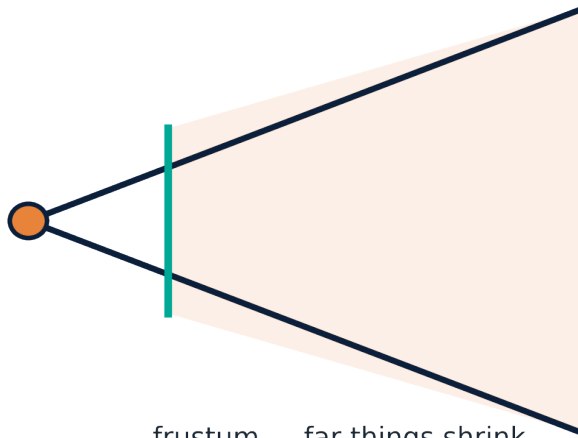
Projection Matrices

`glm::ortho`



box volume — no shrinking
sizes stay true

`glm::perspective`



frustum — far things shrink
the 3D illusion

Parallel → `glm::ortho`

Maps a box to clip space. No divide by depth, so sizes are preserved.

Perspective → `glm::perspective`

Maps a frustum. The w-divide shrinks far things — the 3D illusion.

The Perspective Divide

1

Vertex $\times$ matrices — gives clip coordinates (x, y, z, w)

2

GPU divides by w — $x/w, y/w, z/w$ — the 'perspective divide'

3

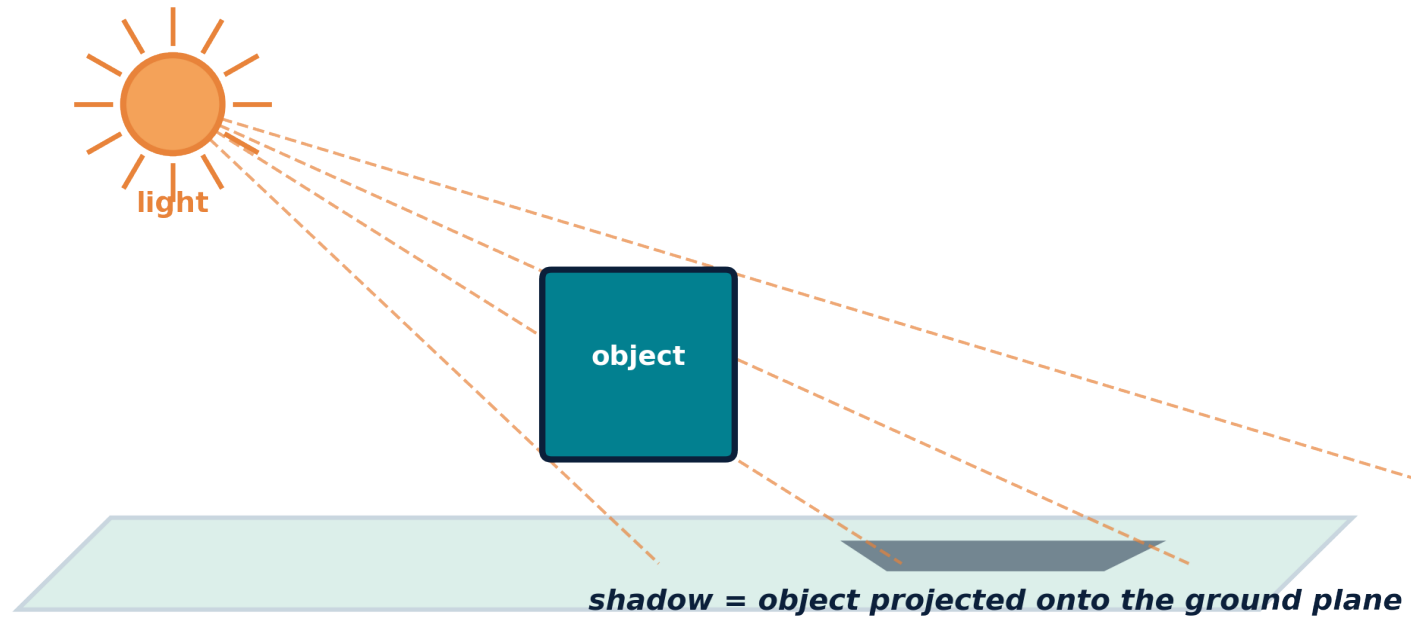
Bigger w = smaller result — far vertices have large w , so they shrink on screen



Parallel projection has no divide

In ortho, w stays 1, so nothing shrinks — that is exactly why parallel views preserve true sizes.

Projections & Shadows



💡 A shadow is a projection

Flatten an object onto the ground by projecting every vertex along rays from the light. The same projection maths that builds a camera image can squash geometry onto a plane — that is a planar shadow. A taste of Module 6 (Lighting & Shading).

The Viewing-Aware Render Loop

```
while (!glfwWindowShouldClose(window)) {  
    float now = glfwGetTime();  
    deltaTime = now - lastFrame;  lastFrame = now;  
  
    processInput(window);           // WASD moves cameraPos  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // HSR  
    glm::mat4 view = glm::lookAt(cameraPos, cameraPos+cameraFront, up);  
    glm::mat4 proj = glm::perspective(radians(fov), aspect, 0.1f, 100.0f);  
    // send view + proj, then draw every object  
    glfwSwapBuffers(window);  glfwPollEvents();  
}
```

Build the camera you can fly

2.1

Move the eye by hand — `change cameraPos, rebuild lookAt each frame`

2.2

WASD movement — `poll glfwGetKey, move along front & right × deltaTime`

2.3

Mouse look — `callback updates yaw/pitch → cameraFront`

2.4

Clamp pitch $\pm 89^\circ$ — `stop the world flipping when looking up/down`

2.5

Scroll zoom — `scroll changes fov, clamp  $1^\circ$ – $75^\circ$`

Part 3 (independent): a 6+ object scene you fly through, with live camera position in the title bar. Sprint-on-SHIFT is the bonus.

Key Terms

Hidden-surface removal

Deciding which surfaces are visible vs blocked.

z-buffer / depth buffer

Per-pixel store of nearest depth seen so far.

Depth test

GL_DEPTH_TEST — keep a fragment only if nearer.

Yaw / Pitch

Left-right / up-down camera rotation from the mouse.

deltaTime

Seconds since last frame — for smooth, FPS-independent motion.

Perspective divide

Dividing by w so far things shrink.

Self-Check

Q1

What two buffers must you clear every frame in a 3D scene, and why?

Q2

Why does the painter's algorithm fail where the z-buffer succeeds?

Q3

Why must pitch be clamped to $\pm 89^\circ$ and not $\pm 90^\circ$?

Q4

What does multiplying movement by deltaTime achieve?

Q5

Which step makes far objects appear smaller — and which projection skips it?

Recap

- ✓ Hidden-surface removal: the z-buffer keeps the nearest fragment per pixel
- ✓ Enable `GL_DEPTH_TEST` once; clear COLOR + DEPTH bits every frame
- ✓ Walking = move cameraPos (WASD) + change cameraFront (mouse yaw/pitch)
- ✓ Clamp pitch $\pm 89^\circ$; multiply movement by deltaTime for smooth motion
- ✓ Projection matrices: ortho (box, no divide) vs perspective (frustum, w-divide)
- ✓ Shadows are projections too — squashing geometry onto a plane

Next: Module 6 — Lighting & Shading



Light & matter — How surfaces respond to light



The Phong model — Ambient + diffuse + specular



Materials — Per-surface shininess and colour



Shading the sphere — Smooth vs flat shading

References

Angel & Shreiner (2014)

Interactive Computer Graphics, 8e —
Viewing & HSR

Gordon & Clevenger (2024)

CG Programming in OpenGL with C++, 3e

learnopengl.com

Camera • Depth Testing

Lab this week: complete Lab 4 (camera + input). Lab Test One also runs this week.