



Hierarchical & Object-Oriented Graphics

Module 7 — Symbols, instances, scene graphs & parent-child motion

Scene graphs • Hierarchical models • Trees & traversal • the matrix-stack idea

Week 13 • Lecture 13 • FTMK • Session 2025/2026

Learning Outcomes



Think in scene graphs

Represent a scene as a tree of nodes where children inherit their parent's transform.



Build hierarchical models

Drive a robot arm / solar system using nested model matrices — parent times local.



Use the matrix-stack idea

Recreate push/pop in modern OpenGL with plain C++ matrix variables.



Reuse with symbols & instances

Define geometry once and draw it many times with different transforms.



Traverse a tree to render

Walk parent to child, carrying the accumulated transform down the branches.

Move the car — and the wheels, doors, and driver should move **WITH** it.

Flat list

Every part has its own world position. Move the car? Update every single part by hand. Miss one and a wheel floats away.

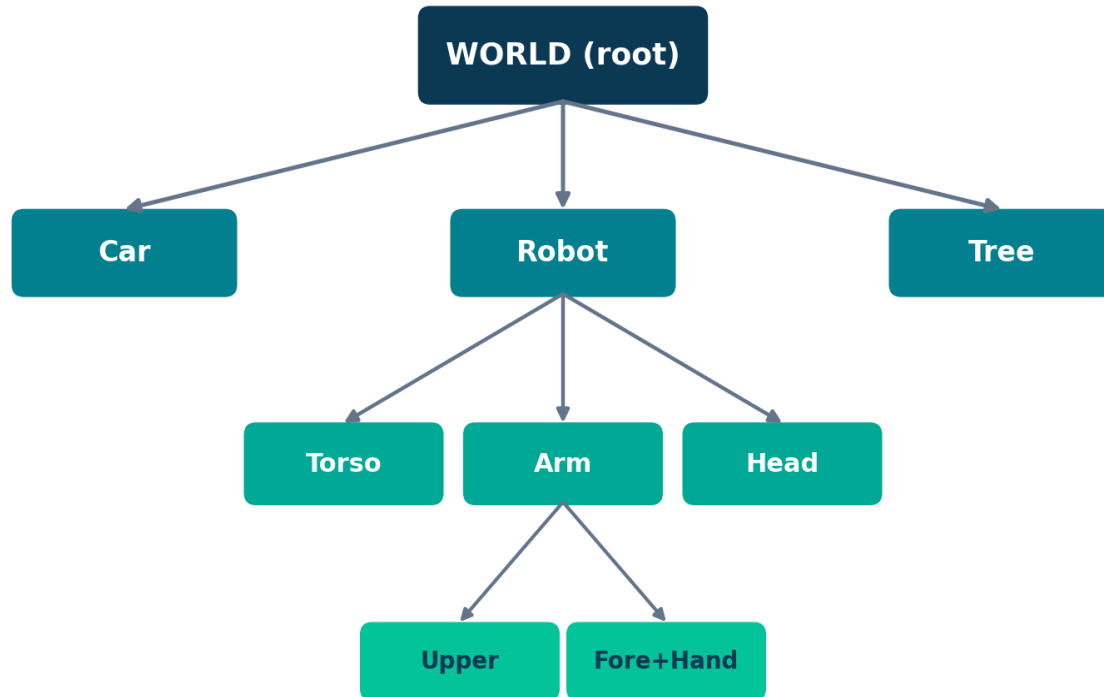
Hierarchy

Parent the parts to the car. Move the car node once — children follow automatically. One change, whole car moves.

A hierarchy turns 'update everything' into 'update one node'.

A Scene Is a Tree of Nodes

A scene graph: every object is a node; children inherit their parent's transform



Move the
Robot node →
everything
under it moves



Tree vocabulary Root = the world. Each node has one parent and any number of children. A node's final transform = its parent's transform combined with its own.

Why a tree?

Trees have no loops, so traversal always ends. Parent-before-child gives a natural draw order.

The One Equation: $\text{child} = \text{parent} \times \text{local}$

Each node stores only its LOCAL transform — where it sits relative to its parent.

To draw it, you need its WORLD transform. You get that by multiplying the parent's world matrix by this node's local matrix:

```
glm::mat4 world = parentWorld * localTransform;
```

Matrix order matters: parent on the LEFT, child's local on the RIGHT.



It chains down the tree The world matrix you compute for a node becomes the 'parent' for its own children. Depth 3? Three multiplies. The chain carries the whole ancestry.

Remember from Module 4

Matrix multiply is NOT commutative. $\text{parent} * \text{local}$ and $\text{local} * \text{parent}$ give different results. Parent always goes first (on the left).

Local Space vs World Space

LOCAL space

Where a part sits relative to its PARENT. A car wheel is 'at the front-left corner of the body' — that never changes, even as the car drives around.

WORLD space

Where the part ends up in the SCENE, after every ancestor's transform is applied. This is what you draw with.

The one-liner to remember

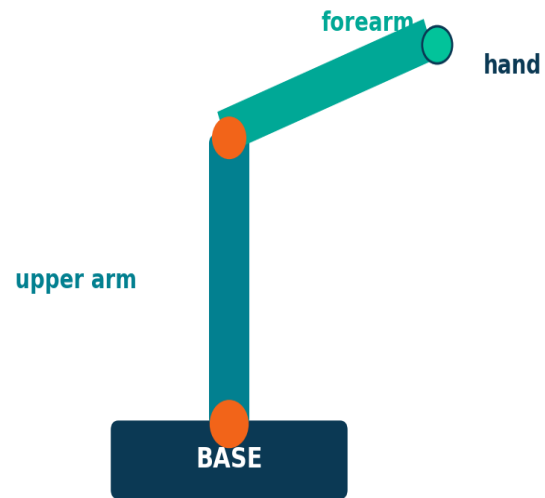
You author in LOCAL.

The tree converts to WORLD.

You only ever describe each node relative to its parent — the easy, stable part. Multiplying down the tree turns all those little local placements into final world positions automatically.

```
// local: never changes as the car moves  
wheel.local = T(frontLeft);  
// world: recomputed every frame  
wheel.world = car.world * wheel.local;
```

The Robot Arm



Each joint's transform is **RELATIVE** to its parent

```
mat4 base  = T(basePos) * R(baseAngle);  
mat4 upper = base * R(shoulder);  
mat4 fore  = upper * T(len) * R(elbow);  
mat4 hand  = fore * T(len) * R(wrist);
```

Rotate the shoulder → forearm + hand follow, because they are built ON TOP of 'upper'.



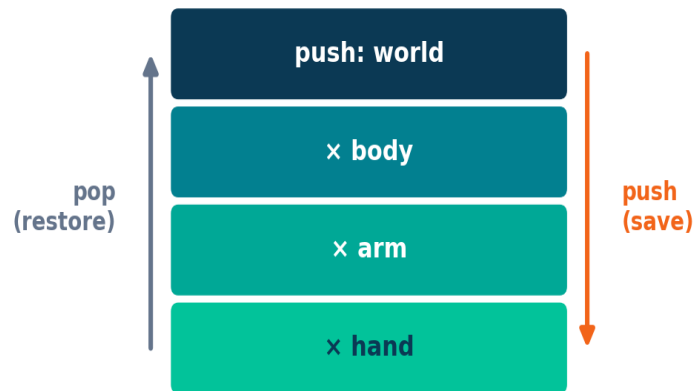
Why this is the classic demo Rotate the shoulder and the forearm and hand swing too — but rotate the wrist and only the hand moves. The hierarchy decides what follows what.

Try it in your head

Bend your elbow: your hand moves, your shoulder doesn't. Your arm IS a parent-child hierarchy.

Push, Pop... and the Modern Way

The Matrix Stack idea



Modern OpenGL has NO built-in stack.

You ARE the stack — just keep parent matrices in C++ variables and multiply:

```
glm::mat4 child = parent * local;
```

'Save' = keep the parent in a variable.

'Restore' = reuse it for the next child.



Old OpenGL did this for you Legacy OpenGL had `glPushMatrix / glPopMatrix` — a built-in stack. Core profile removed it. Now YOU hold parent matrices in variables.

The mental model

'Push' = save the parent in a variable before you descend. 'Pop' = come back up and reuse that saved parent for the next sibling.

Drawing the Tree: Depth-First Traversal

To render the whole scene, walk the tree from the root, carrying the accumulated transform down each branch.

Visit a node, draw it, then recurse into each child — passing this node's world matrix as their parent.

```
void draw(Node n, mat4 parent){  
    mat4 world = parent * n.local;  
    drawMesh(n.mesh, world);  
    for (Node c : n.children)  
        draw(c, world);    // recurse  
}
```



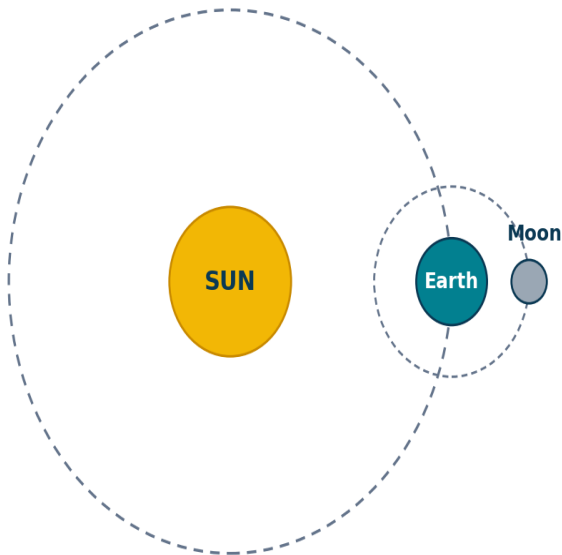
Recursion mirrors the tree The call stack naturally remembers each parent for you. One recursive function renders a scene of any depth or shape.

Depth-first = follow one branch to its tip

Go all the way down one branch (root to leaf) before backing up to the next branch. The recursion handles the bookkeeping; you just pass 'world' as the next parent.

You've Already Built One: The Solar System

Parent-child motion: the Moon orbits the Earth, which orbits the Sun



```
mat4 sun = R(t*0.3);  
mat4 earth = sun *  
    T(orbit)*R(t);  
mat4 moon = earth *  
    T(small)*R(t*2);
```

= Lab 3 Task 2.5

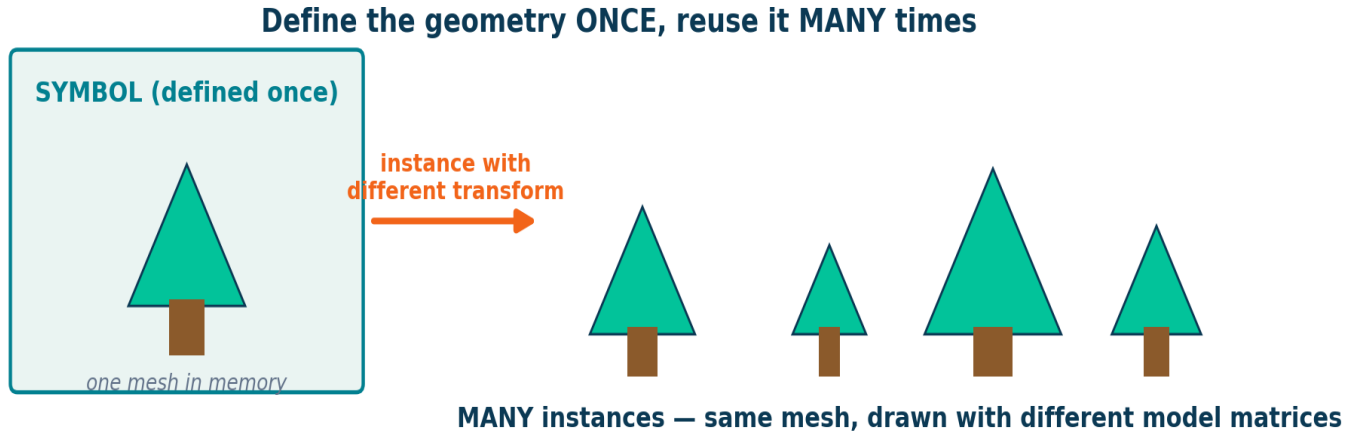


Nested rotation = orbit The Moon's matrix is built on the Earth's, which is built on the Sun's. Each level adds one rotation, so the Moon orbits the Earth while both orbit the Sun.

Same idea, different scene

Robot arm or solar system — it's the same parent×local chain. Only the local transforms differ.

Define Once, Draw Many: Instancing



Symbol vs instance A SYMBOL is the master geometry, stored once (one VAO/VBO). An INSTANCE is that symbol drawn with a particular model matrix. 100 trees, one mesh.

Why it matters

Memory: store the mesh once, not 100 times. This is the OO idea — a class (symbol) and its objects (instances) with their own state (transform).

Modelling Objects as... Objects

A SceneNode class

- mesh / geometry
- local transform (mat4)
- list of children
- material / colour

Methods it offers

- `addChild(node)`
- `update(dt) // animate`
- `draw(parentMatrix)`
- `setTransform(...)`

Why OO fits graphics

- each object owns its state
- behaviour travels with data
- reuse via composition
- mirrors the scene tree

Animating a Hierarchy Over Time

To animate, change a node's LOCAL transform each frame — then re-traverse.

Feed time into the local rotations and the whole branch below moves correctly, because the world matrices are rebuilt every frame from the new locals.

```
float t = glfwGetTime();  
// only the LOCAL angles change:  
shoulder.local = R(sin(t)*30deg);  
elbow.local    = R(sin(t*2)*45deg);  
// then draw(root, identity) as before  
// → forearm + hand follow for free
```



Animate parents, not children Rotate the shoulder a little and the whole arm sweeps. You get rich motion from a few animated joints because children inherit the movement.

This is the link to Lab 9

There, time drives PHYSICS (gravity, velocity) instead of joint angles — but the pattern is identical: update state each frame, then redraw. A hierarchy says where; a loop says when.

From Static Trees to Moving Worlds

So far

- Objects arranged in a hierarchy
- $\text{child} = \text{parent} \times \text{local}$
- Traverse the tree to draw
- Reuse meshes via instances



Lab 9 (this week)

Physics Simulation — Basics

- Separate UPDATE from RENDER
- `deltaTime`: same speed everywhere
- Euler integration: $v += g \cdot dt$; $p += v \cdot dt$
- Bounce with restitution

Lab 9 — Physics Simulation

1

Update, then render

Every frame: step the physics first, draw second. Keep the two jobs apart.

2

Scale by deltaTime

Measure seconds since last frame; multiply all motion by it for frame-rate independence.

3

Euler in two lines

`velocity += gravity*dt; then position += velocity*dt;`

4

Bounce with restitution

Hit the floor → flip vertical velocity, multiply by 0.8 to lose energy.

5

Build a ball-pit

Part 3: 10 balls, random starts, gravity + walls. A mini simulation.

Common Hierarchy Pitfalls

Wrong multiply order

`child = local * parent`

Parent goes on the LEFT: $\text{world} = \text{parent} * \text{local}$. Reversing it detaches the child.

Forgetting to pass 'world' down

`children use the root, not their parent`

In the recursion, pass the node's computed world matrix as the child's parent.

Mutating a shared matrix

`all instances move together`

Each instance needs its OWN model matrix; don't edit one shared matrix in place.

Deep nesting, no resets

`transforms accumulate forever`

Build each node's local fresh from identity; let the chain do the accumulation.

Key Terms — Know These Cold

Scene graph

A tree of nodes representing a scene; children inherit their parent's transform.

Local transform

A node's placement relative to its parent, not the world.

Traversal

Walking the tree (depth-first) to draw every node in order.

Symbol

Master geometry defined and stored once.

Node

One element of the tree: geometry plus a local transform plus a list of children.

World transform

parent's world $\times$ this node's local; where the node actually ends up.

Matrix stack

Push/pop scheme for saving parents; in modern OpenGL, just C++ variables.

Instance

A symbol drawn with its own model matrix; reuse without copying.

Self-Check Questions

- 1 Write the equation for a node's world transform in terms of its parent.
- 2 Why must the parent matrix go on the LEFT of the multiplication?
- 3 How does modern OpenGL replace `glPushMatrix` / `glPopMatrix`?
- 4 In the solar system, why does the Moon orbit the Earth automatically?
- 5 What is the difference between a symbol and an instance?
- 6 What kind of traversal draws a scene graph, and why does it always finish?

Recap — Screenshot This Slide



A scene is a tree of nodes; children inherit the parent's transform.



The one equation: $\text{world} = \text{parent} \times \text{local}$ (parent on the left).



Traverse depth-first; the call stack remembers each parent.



Matrix stack push/pop = saving parent matrices in variables.



Robot arm and solar system are the same $\text{parent} \times \text{local}$ chain.



Symbols define geometry once; instances reuse it with transforms.



OO graphics: each node is an object owning its state + behaviour.



Next: Lab 9 makes the scene MOVE with a physics update loop.

Module 8 — Visualization



Why visualize

Turning data and simulations into pictures the eye can actually reason about.



Visual metaphors

Mapping numbers to position, colour, size, and motion.



Handling complexity

Realism vs clarity, and taming huge or messy data.

Bring your Lab 9 simulation — it's our first visualization example.

Reading: learnopengl.com • Angel & Shreiner Ch. on hierarchical modelling • Gordon & Clevenger 3rd Ed.