Hierarchical Modelling

> **Complete guide for Section 6 + Lab 2 coding**
> Includes: Subroutines, Scene Graphs, Transformations, Save/Restore, Animation
> Tips: Use subroutines for simple hierarchies; scene graphs for reuse/complexity. Always save/restore to isolate transforms. Test incrementally.

---

1. Introduction to Hierarchical Modelling

Break complex objects into simpler parts in a tree structure.

- **Benefits**: Reuse, easy animation, modular code.
- **Coding Tip**: Start from leaf nodes (basic shapes) and build up to root (full scene).
- **Example Scene**: Windmill (base + rotating blades), Cart (body + wheels).
- **Common Mistake**: Forgetting to isolate transforms – leads to unwanted propagation (e.g., rotating blades rotate the ground).

---

2. Subroutine Hierarchy (Section 6.2)

Build with functions – each draws a part, calls sub-parts.

2.1 Basic Subroutine Example

```
function drawRectangle(g, x, y, width, height, color) {
    g.fillStyle = color;
    g.fillRect(x, y, width, height);
}
```

> Tip: Parameters for flexibility (position, size, color).
> Technique: Define basics first (rect, circle, triangle).

2.2 Creating Hierarchical Structure

Parent calls children.

```
function drawHouseBase(g, x, y, width, height) {
    drawRectangle(g, x, y, width, height, 'brown');  // Walls
    drawRectangle(g, x + width/3, y + height/2, width/3, height/2, 'black');  //
Door
}

function drawRoof(g, x, y, width, height) {
    g.fillStyle = 'red';
    g.beginPath();
```

```
    g.moveTo(x, y);
    g.lineTo(x + width / 2, y - height);
    g.lineTo(x + width, y);
    g.closePath();
    g.fill();
}

function drawHouse(g, x, y, width, height) {
    drawHouseBase(g, x, y, width, height * 0.75);
    drawRoof(g, x, y, width, height * 0.25);
}
```

> Tip: Use relative coords (0,0 center) for easy transforms.
> Technique: Test each function alone in draw().

2.3 Applying Transformations

Modify coordinate system for positioning.

```
function drawHouse(g, x, y, width, height, angle) {
    g.save();
    g.translate(x, y);       // Position
    g.rotate(angle);         // Orientation
    g.scale(width/100, height/100);  // Assume base size 100
    drawHouseBase(g, -50, -50, 100, 75);  // Centered at 0,0
    drawRoof(g, -50, -50, 100, 25);
    g.restore();
}
```

> Order: Scale → Rotate → Translate (fixed in Canvas).
> Tip: Use `g.transform(1, shearY, shearX, 1, 0, 0)` for skew.

2.4 Save and Restore (Importance & How-To)

Isolate transforms to prevent leakage.

Importance: Without it, child transforms affect siblings/parents (e.g., wheel rotation rotates cart body).
How-To: `save()` before transforms, `restore()` after drawing sub-object.
Stack Behavior: Push (save) / Pop (restore) – match every save with restore.

Example (Nested):

```
function drawWindmill(g) {
    g.save();                // Save for base
    g.translate(0, 2);       // Position base
    drawBase(g);
    g.restore();
```

```
    g.save();                // Save for blades
    g.translate(0, 2);       // Position top
    g.rotate(frameNumber * 0.1);  // Animate blades
    drawBlades(g);
    g.restore();
}
```

> Tip: If bugs, log transform state or draw debug axes.
> Technique: For animation, update inside subroutine (e.g., rotate based on frameNumber).

---

3. Scene Graphs (Section 6.3)

Object-oriented data structure – nodes for objects/transforms.

3.1 SceneGraphNode (Base)

Abstract – override `doDraw(g)` for custom drawing.

Example (Line):

```
let line = new SceneGraphNode();
line.doDraw = function(g) {
    g.beginPath();
    g.moveTo(-0.5, 0);
    g.lineTo(0.5, 0);
    g.stroke();
};
```

3.2 CompoundObject (Group Sub-Objects)

```
function CompoundObject() {
    SceneGraphNode.call(this);
    this.subobjects = [];
}
CompoundObject.prototype.add = function(node) {
    this.subobjects.push(node);
    return this;  // Chainable
};
CompoundObject.prototype.doDraw = function(g) {
    this.subobjects.forEach(sub => sub.draw(g));
};
```

> Tip: Use for groups (e.g., wheel = circle + spokes).

3.3 TransformedObject (Apply Transforms)

```
function TransformedObject(object) {
    SceneGraphNode.call(this);
    this.object = object;
    this.rotationInDegrees = 0;
    this.scaleX = 1; this.scaleY = 1;
    this.translateX = 0; this.translateY = 0;
}
TransformedObject.prototype.setRotation = function(deg) {
    this.rotationInDegrees = deg;
    return this;
};
TransformedObject.prototype.setScale = function(sx, sy) {
    this.scaleX = sx; this.scaleY = sy;
    return this;
};
TransformedObject.prototype.setTranslation = function(dx, dy) {
    this.translateX = dx; this.translateY = dy;
    return this;
};
TransformedObject.prototype.doDraw = function(g) {
    g.save();
    g.translate(this.translateX, this.translateY);
    g.rotate(this.rotationInDegrees * Math.PI / 180);
    g.scale(this.scaleX, this.scaleY);
    this.object.draw(g);
    g.restore();
};
```

Chain example:
`.setScale(2,2).setRotation(45).setTranslation(100,100)`

> Tip: Transforms auto-isolated with save/restore.

3.4 Building a Scene (Car Example)

```
let world = new CompoundObject();

// Wheel
let wheelTemp = new CompoundObject();
wheelTemp.add( new TransformedObject(filledCircle).setScale(2,2) );
for (let i = 0; i < 12; i++) {
    wheelTemp.add( new TransformedObject(line).setRotation(i*30) );
}
let wheel = new TransformedObject(wheelTemp);

// Cart
let cartTemp = new CompoundObject();
```

```
cartTemp.add( new
TransformedObject(wheel).setScale(0.8,0.8).setTranslation(-1.65,-0.1) );
cartTemp.add( new
TransformedObject(wheel).setScale(0.8,0.8).setTranslation(1.65,-0.1) );
cartTemp.add( new
TransformedObject(filledRect).setScale(6,1.5).setTranslation(0,1) );
let cart = new TransformedObject(cartTemp).setScale(0.3,0.3);

world.add(cart);
```

> How to Make Your Own: Define templates (CompoundObject), wrap in TransformedObject for positioning. Add to root.

3.5 Traversal & Stack

Push/Pop: Auto in `doDraw()` (save/restore).
Importance: Ensures child transforms don't affect siblings.

> Tip: For complex, visualize tree – draw on paper first.

---

4. Animation (Section 6.4)

Update in `updateFrame()` – redraw full scene.

Basic Update

```
function updateFrame() {
    frameNumber++;
    cart.setTranslation(Math.sin(frameNumber * 0.05) * 2, 0);  // Oscillate
    wheel.setRotation(frameNumber * 3);  // Spin
}
```

> Tips: Use `%` for loops, `Math.sin/cos` for smooth motion.
> Techniques: Procedural (physics-like), keyframe interpolation (lerp between poses).
> Advanced: Inverse Kinematics (IK) for limbs – calculate joint angles from end position.

Example: Ferris Wheel (Ex 3 Idea)

```
let wheel = new TransformedObject(wheelTemp).setRotation(frameNumber * 0.05);  //
Spin wheel
for (let i = 0; i < 8; i++) {
    let seat = new TransformedObject(seatTemp)
        .setTranslation(0, r)
        .setRotation(-frameNumber * 0.05)  // Counter-rotate seat to stay
horizontal
        .setRotation(i * 45);  // Position on wheel
```

```
    wheel.add(seat);
}
```

> Make Your Own: Think real-world (sailboat rocking, airplane flying). Add multiple instances (e.g., 2 carts).

---

5. Exam Tips & Techniques

- **Subroutine vs Scene Graph:** Subroutines for quick/simple; Graphs for reuse/animation (easier updates).
- **Save/Restore Importance:** Prevents "bleed" – always bracket transforms.
- **How to Code Your Own:**
    1. Sketch hierarchy tree.
    2. Define basics (leaf nodes).
    3. Build composites (parents call children).
    4. Add transforms/animation last.
    5. Test: Comment out parts, draw one level at a time.

> **Common Errors:** Mismatched save/restore, wrong order (translate last), forgetting relative coords.
> **Lab 2 Style Questions:** Build cart/windmill, animate rotation/translation, create custom scene (ferris wheel, swing).