Hierarchical Modelling with Scene Graph Construction

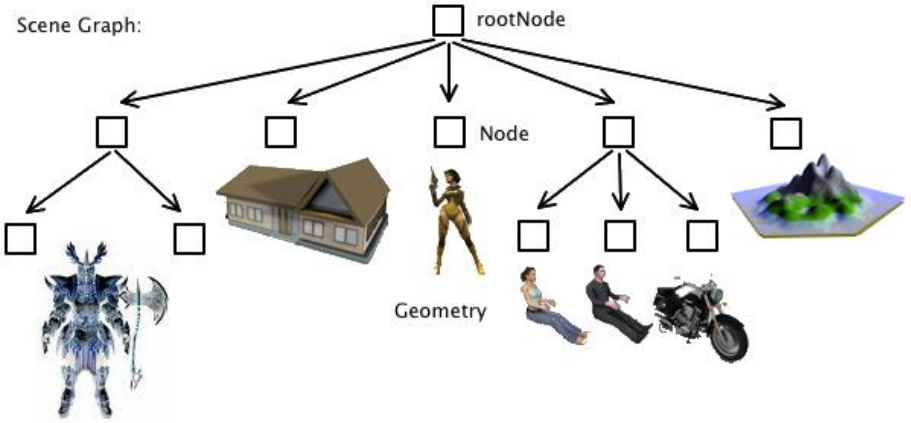
Frederick Li

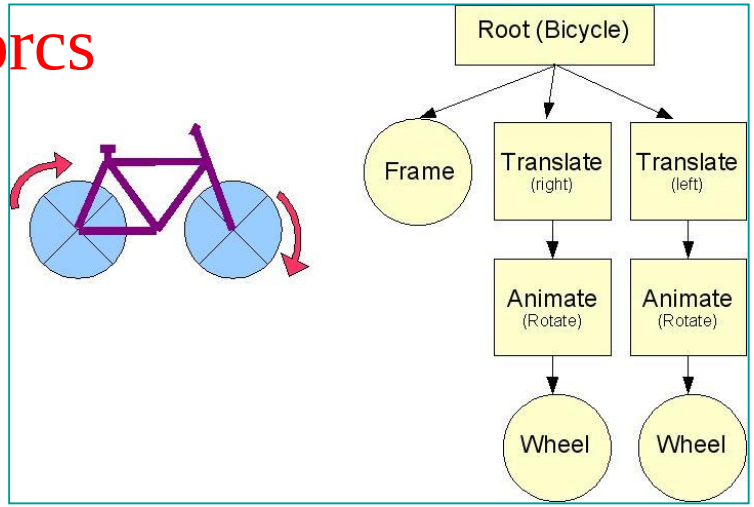
This Lesson

➢ Scene graph construction

Scene Graph:

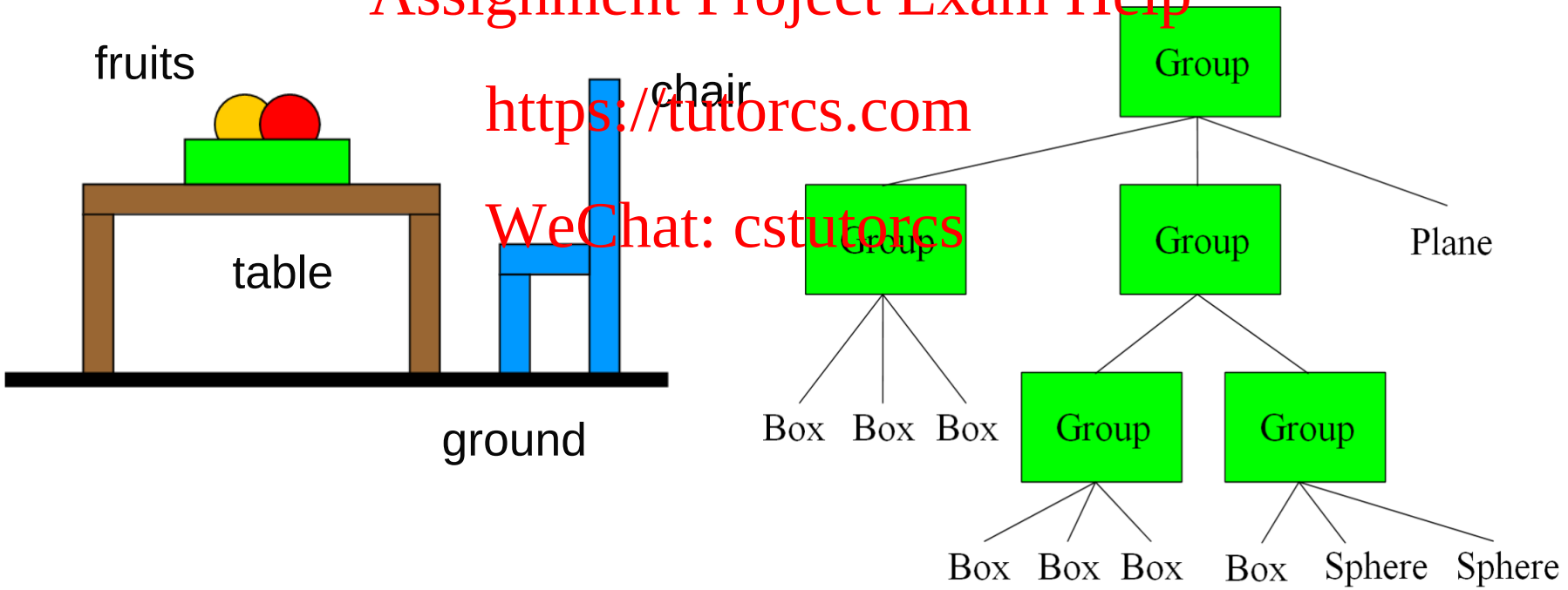
- is a collection of nodes in a graph or tree structure
- A tree node may have many children but often only a single parent, with the effect of a parent applied to all its child nodes
- An operation performed on a group automatically propagates its effect to all of its members





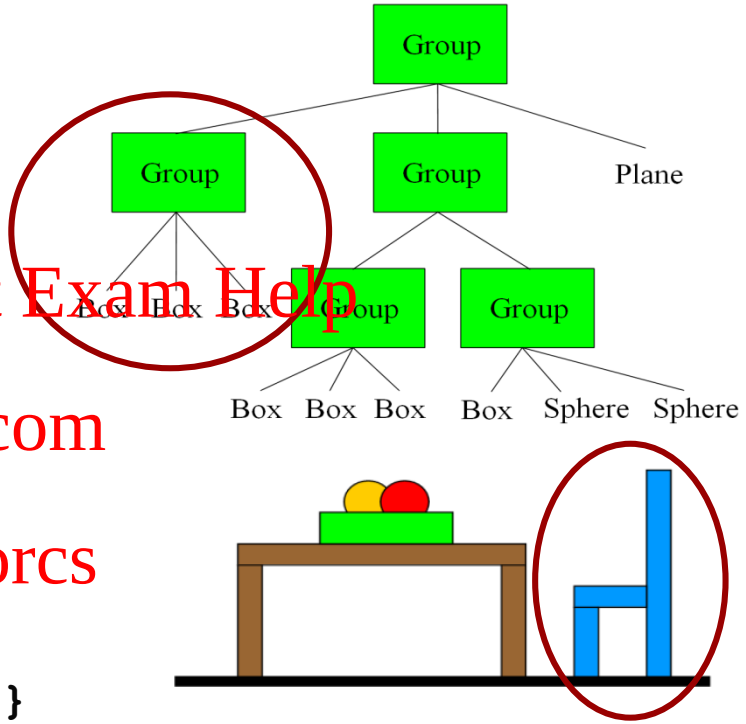
Scene Graph

- Data structure arranges the logical and spatial representation of a graphical scene
- Hierarchical Grouping of Objects



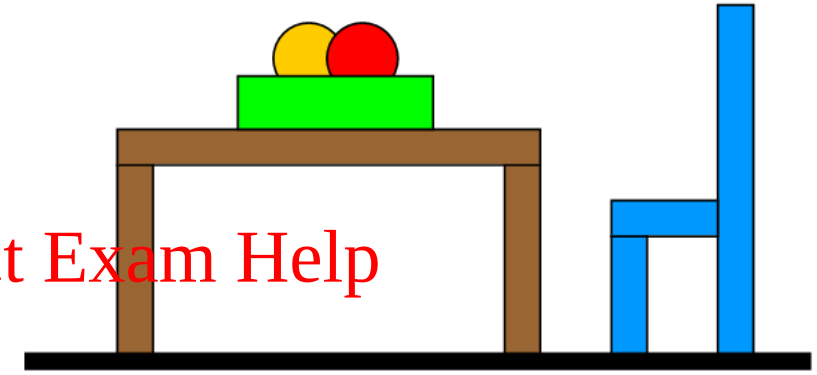
Pseudocode for a Scene Graph

```
Group {
     numObjects 3
     Group {
          numObjects 3
          Box { <BOX PARAMS> }
          Box { <BOX PARAMS> }
          Box { <BOX PARAMS> } }
     Group {
          numObjects 2
          Group {
               Box { <BOX PARAMS> }
               Box { <BOX PARAMS> }
               Box { <BOX PARAMS> } }
          Group {
               Box { <BOX PARAMS> }
               Sphere { <SPHERE PARAMS> }
               Sphere { <SPHERE PARAMS> } } }
     Plane { <PLANE PARAMS> } }
```

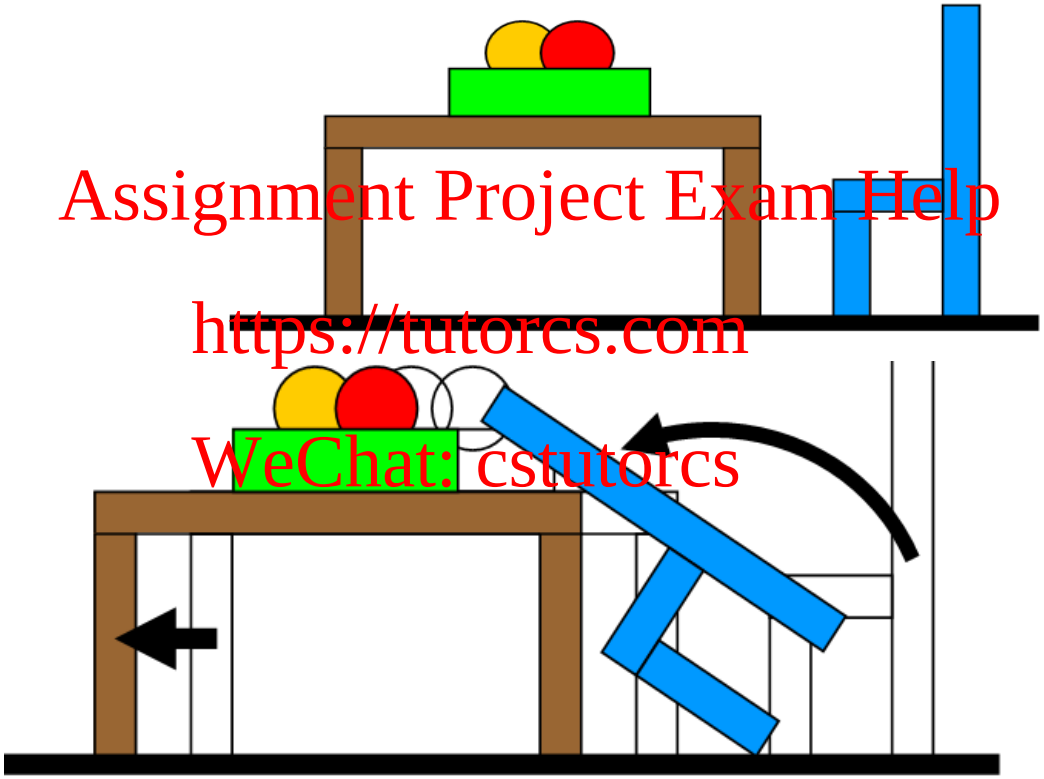


Adding Materials

```
Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> } } }
            Material { <BLACK> }
    Plane { <PLANE PARAMS> } }
```

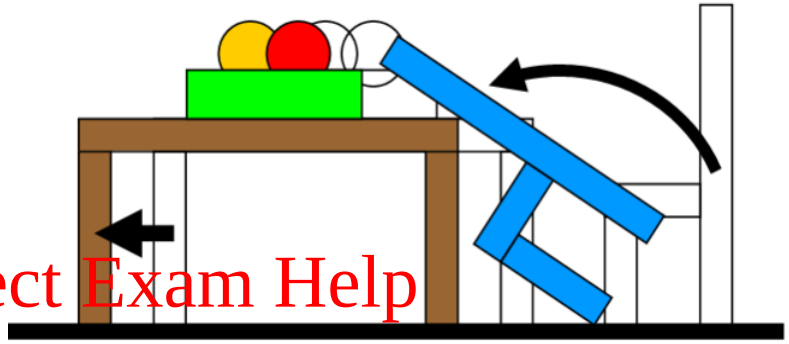


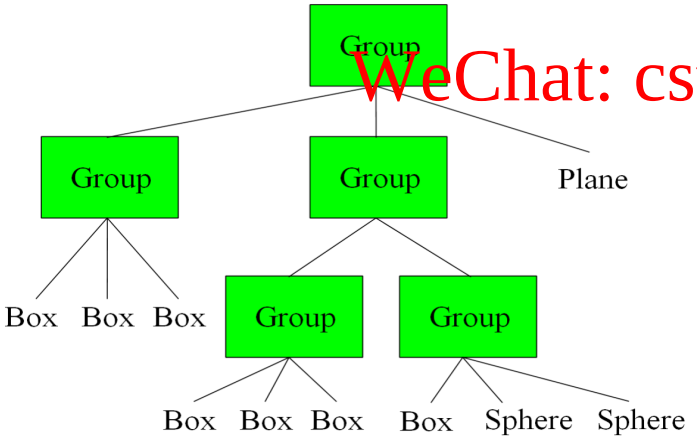
Adding Transformations



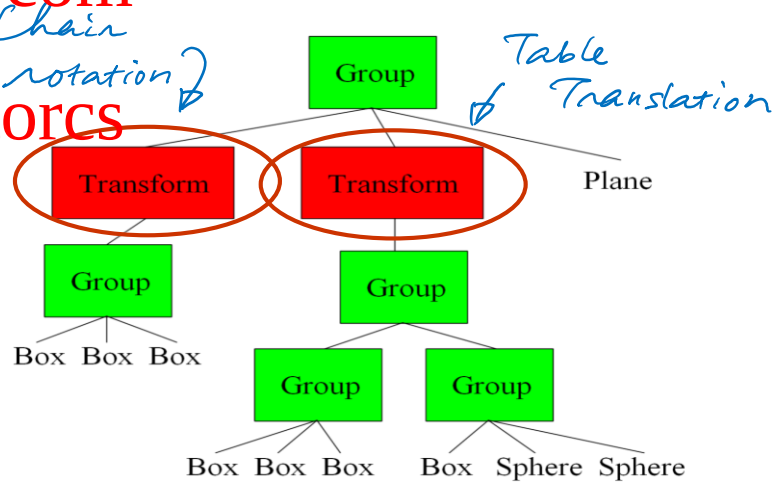
Hierarchical Transformation of Objects

- Apply geometric transformations to logical groups of objects within the scene





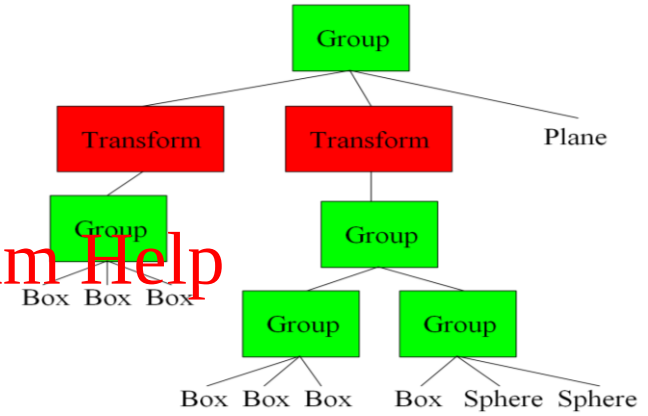
Simple Scene Graph

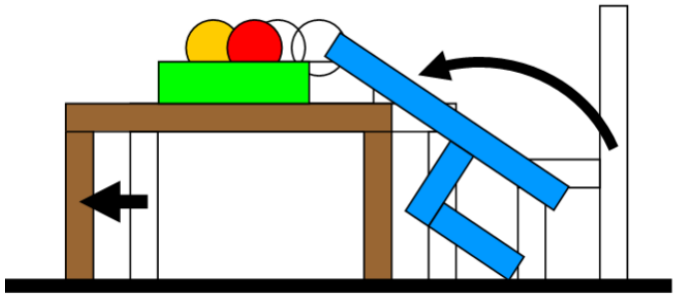


Scene Graph with transformations

Scene Graph with transformations

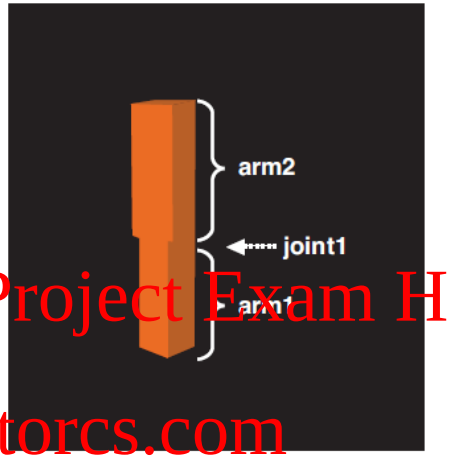
```
Group {
    numObjects 3
    Transform {
        ZRotate { 45 }
        Group {
            numObjects 3
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } } }
    Transform {
        Translate { -2 0 0 }
        Group {
            numObjects 2
            Group {
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> } }
            Group {
                Box { <BOX PARAMS> }
                Sphere { <SPHERE PARAMS> }
                Sphere { <SPHERE PARAMS> } } } }
    Plane { <PLANE PARAMS> } }
```





Sample Program (JointModel.js)

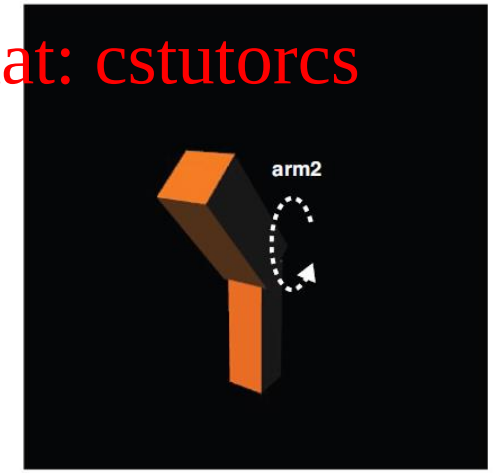
Simple robot arm
with two components



←→: arm1 rotation(y-axis), ↑ ↓ : joint1 rotation(z-axis)

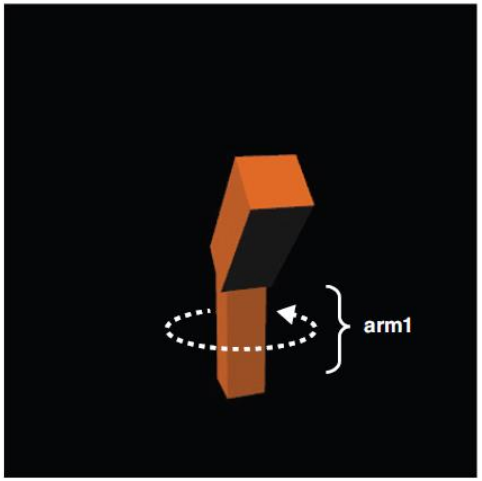
What is the scene graph?

Each component supports
a different type of rotation



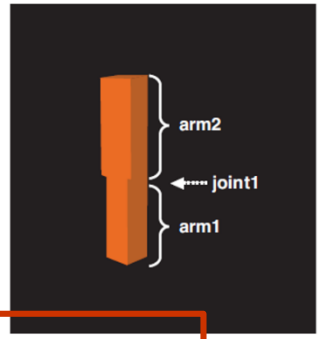
←→: arm1 rotation(y-axis), ↑ ↓ : joint1 rotation(z-axis)

Every time when arm 1
rotates, the same rotation
applies simultaneously to
arm 2



←→: arm1 rotation(y-axis), ↑ ↓ : joint1 rotation(z-axis)

WebGL Implementation




```
181   // Arm1
182   var arm1Length = 10.0; // Length of arm1
183   g_modelMatrix.setTranslate(0.0, -12.0, 0.0);
184   g_modelMatrix.rotate(g_arm1Angle, 0.0, 1.0, 0.0);   // Rotate y-axis
185   drawBox(gl, n, viewProjMatrix, u_MvpMatrix, u_NormalMatrix); // Draw
186
187   // Arm2
188   g_modelMatrix.translate(0.0, arm1Length, 0.0);      // Move to joint1
189   g_modelMatrix.rotate(g_joint1Angle, 0.0, 0.0, 1.0);// Rotate z-axis
190   g_modelMatrix.scale(1.3, 1.0, 1.3); // Make it a little thicker
191   drawBox(gl, n, viewProjMatrix, u_MvpMatrix, u_NormalMatrix); // Draw
```


Hierarchical Operations:

- Draw arm 2 (upper part of robot arm); scale, then rotate, then translate it **(implicitly all transformations applied for arm 1 also apply to arm 2)**
- Draw arm 1; rotate, then translate it

Construct individual Component

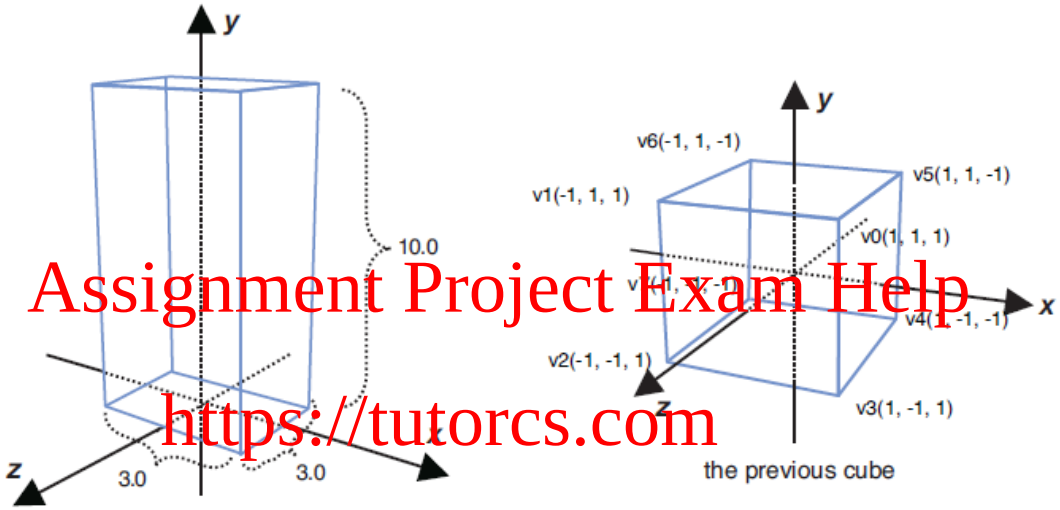


Figure 9.6 A cuboid for drawing the robot arm

Code for the building block (cuboid)

```
var vertices = new Float32Array([
   1.5, 10.0, 1.5, -1.5, 10.0, 1.5, -1.5,  0.0, 1.5,  1.5,  0.0, 1.5, // v0-v1-v2-v3 front
   1.5, 10.0, 1.5,  1.5,  0.0, 1.5,  1.5,  0.0,-1.5,  1.5, 10.0,-1.5, // v0-v3-v4-v5 right
   1.5, 10.0, 1.5,  1.5, 10.0,-1.5, -1.5, 10.0,-1.5, -1.5, 10.0, 1.5, // v0-v5-v6-v1 up
  -1.5, 10.0, 1.5, -1.5, 10.0,-1.5, -1.5,  0.0,-1.5, -1.5,  0.0, 1.5, // v1-v6-v7-v2 left
  -1.5,  0.0,-1.5,  1.5,  0.0,-1.5,  1.5,  0.0, 1.5, -1.5,  0.0, 1.5, // v7-v4-v3-v2 down
   1.5,  0.0,-1.5, -1.5,  0.0,-1.5, -1.5, 10.0,-1.5,  1.5, 10.0,-1.5  // v4-v7-v6-v5 back
 ]);
```

Draw a Component

Code for Javascript main()

```
function drawBox(gl, n, viewProjMatrix, u_MvpMatrix, u_NormalMatrix) {
  g_mvpMatrix.set(viewProjMatrix);     // projection, view transforms
  g_mvpMatrix.multiply(g_modelMatrix); // translate, rotate, scale
  gl.uniformMatrix4fv(u_MvpMatrix, false, g_mvpMatrix.elements);
  ...
  gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
}
```

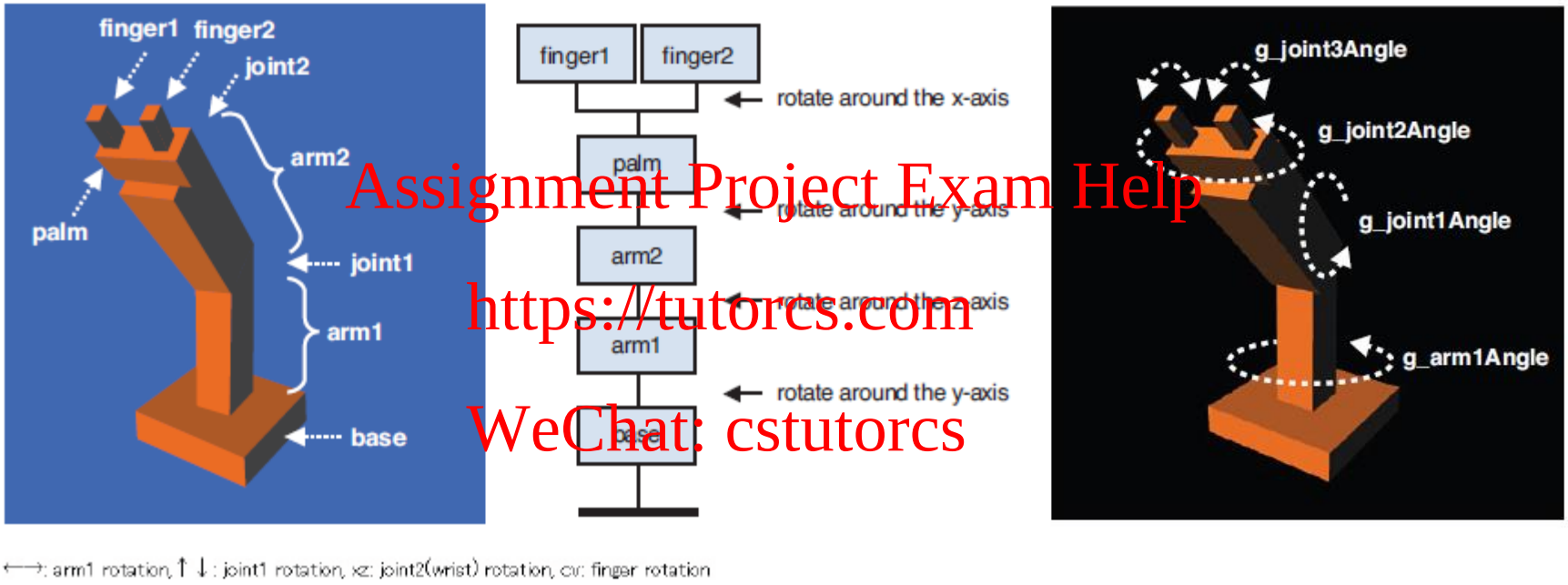


```
V.SHADER: gl_Position = u_MvpMatrix * a_Position;
```

Idea:

- Transformations composition are pre-computed in the CPU
- Use GPU to apply a simple composited transformation on every vertex

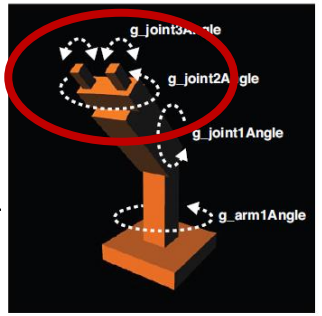
A Complicated Model



←→: arm1 rotation, ↑ ↓ : joint1 rotation, xz: joint2(wrist) rotation, cv: finger rotation

Figure 9.8 The hierarchical structure of MultiJointModel

Draw the Palm (Upper Part)



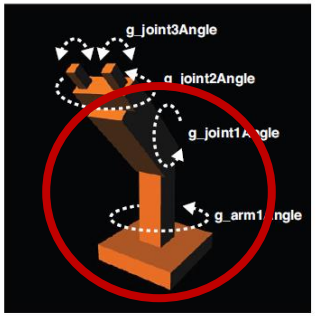

```
// A palm
 var palmLength = 2.0;
 g_modelMatrix.translate(0.0, arm2Length, 0.0);       // Move to palm
 g_modelMatrix.rotate(g_joint2Angle, 0.0, 1.0, 0.0);  // Rotate around the y-axis
 drawBox(gl, n, 2.0, palmLength, 6.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);

 // Move to the center of the top of the palm
 g_modelMatrix.translate(0.0, palmLength, 0.0);

 // Draw finger1
 pushMatrix(g_modelMatrix);
   g_modelMatrix.translate(0.0, 0.0, 2.0);
   g_modelMatrix.rotate(g_joint3Angle, 1.0, 0.0, 0.0);  // Rotate around the x-axis
   drawBox(gl, n, 1.0, 2.0, 1.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);
 g_modelMatrix = popMatrix();

 // Draw finger2
 g_modelMatrix.translate(0.0, 0.0, -2.0);
 g_modelMatrix.rotate(-g_joint3Angle, 1.0, 0.0, 0.0);  // Rotate around the x-axis
 drawBox(gl, n, 1.0, 2.0, 1.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);
```

Draw the Arm (Lower Part)




```
// Draw a base
var baseHeight = 2.0;
g_modelMatrix.setTranslate(0.0, -12.0, 0.0);
drawBox(gl, n, 10.0, baseHeight, 10.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);

// Arm1
var arm1Length = 10.0;
g_modelMatrix.translate(0.0, baseHeight, 0.0);      // Move onto the base
g_modelMatrix.rotate(g_arm1Angle, 0.0, 1.0, 0.0);   // Rotate around the y-axis
drawBox(gl, n, 3.0, arm1Length, 3.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);

// Arm2
var arm2Length = 10.0;
g_modelMatrix.translate(0.0, arm1Length, 0.0);        // Move to joint1
g_modelMatrix.rotate(g_joint1Angle, 0.0, 0.0, 1.0);   // Rotate around the z-axis
drawBox(gl, n, 4.0, arm2Length, 4.0, viewProjMatrix, u_MvpMatrix, u_NormalMatrix);
```


Storing a matrix:

```
var g_matrixStack = [];
function pushMatrix(m) {
  var m2 = new Matrix4(m);
  g_matrixStack.push(m2);
}
```

```
function popMatrix() {
  return g_matrixStack.pop();
}
```

Enhance the drawBox() Function

```
function drawBox(gl, n, width, height, depth,
                 viewProjMatrix, u_MvpMatrix, u_NormalMatrix) {

  pushMatrix(g_modelMatrix);    // Save the model matrix

    // Scale a cube and draw
    g_modelMatrix.scale(width, height, depth);
    // Calculate the model view project matrix and pass it to u_MvpMatrix
    g_mvpMatrix.set(viewProjMatrix);
    g_mvpMatrix.multiply(g_modelMatrix);
    gl.uniformMatrix4fv(u_MvpMatrix, false, g_mvpMatrix.elements);

    ...

    // Draw
    gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
  g_modelMatrix = popMatrix();    // Retrieve the model matrix
}
```

Summary

- Scene graph construction
- WebGL Implementation
  - Draw building blocks (object components)
  - Transform individual components
  - Apply a series of transformations

- **Reference:**
  WebGL Programming Guide [Ch. 9]