9. Modeling and Hierarchy

Outline

- Hierarchical Modeling
- Graphical Objects and Scene Graphs
- Other Tree Structures

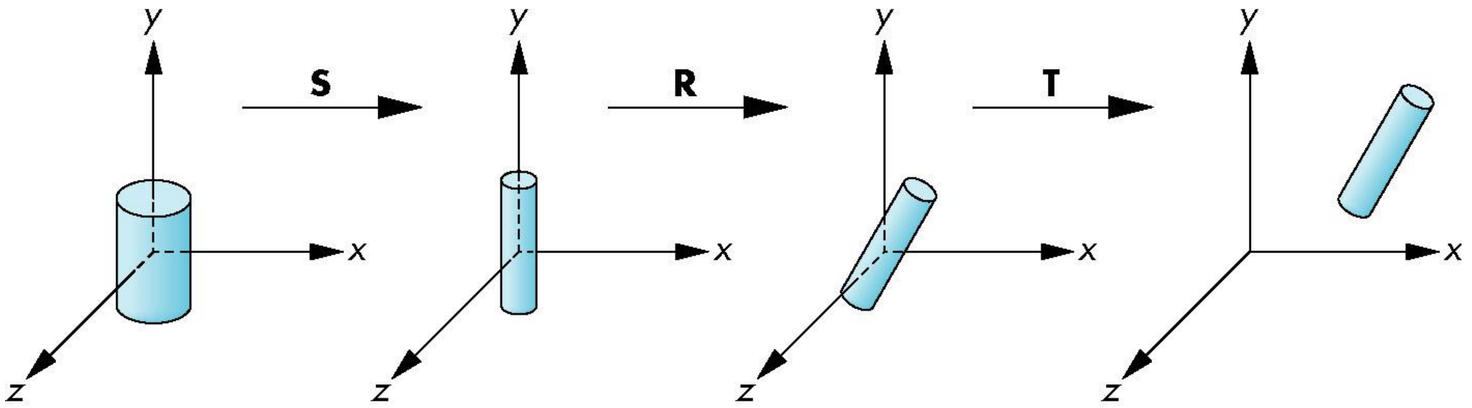
Hierarchical Modeling I

Objectives

- Examine the limitations of linear modeling
  - Symbols and instances
- Introduce hierarchical models
  - Articulated models
  - Robots
- Introduce Tree and DAG models

Instance Transformation

- Start with a prototype object (a *symbol*)
- Each appearance of the object in the model is an *instance*
  - Must scale, orient, position
  - Defines instance transformation



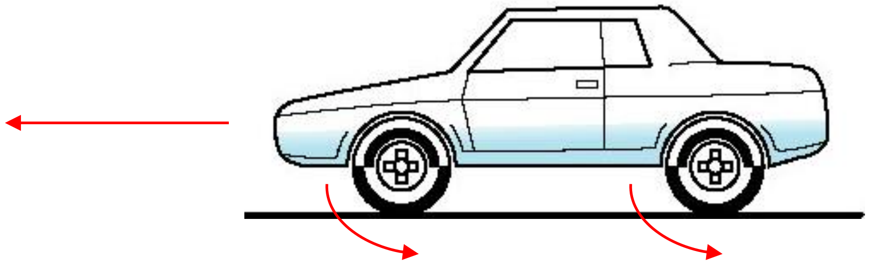
Symbol-Instance Table

Can store a model by assigning a number to each symbol and storing the parameters for the instance transformation

Symbol	Scale	Rotate	Translate
1	s_x, s_y, s_z	$\theta_x, \theta_y, \theta_z$	d_x, d_y, d_z
2			
3			
1			
1			
.			
.			

Relationships in Car Model

- Symbol-instance table does not show relationships between parts of model
- Consider model of car
  - Chassis + 4 identical wheels
  - Two symbols



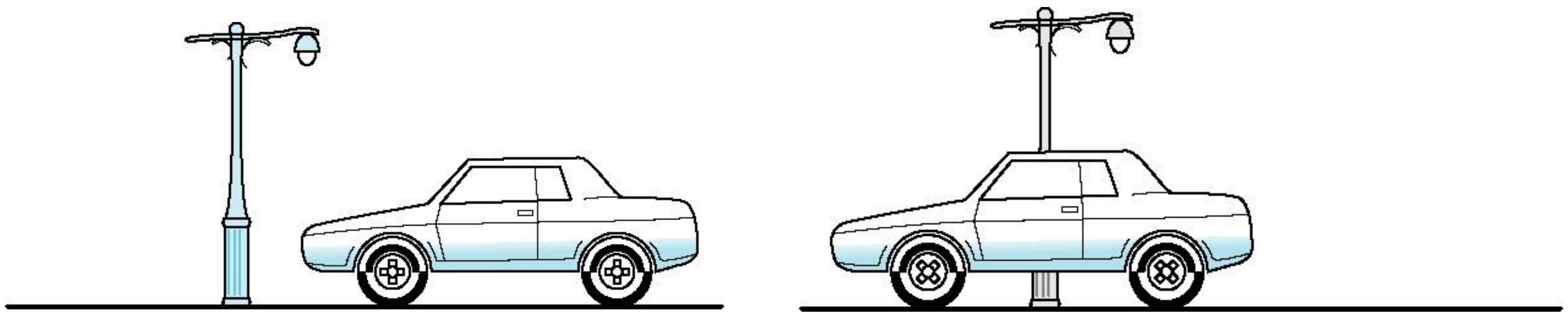
- Rate of forward motion determined by rotational speed of wheels

Structure Through Function Calls

```
car(speed)
{
    chassis()
    wheel(right_front);
    wheel(left_front);
    wheel(right_rear);
    wheel(left_rear);
}
```

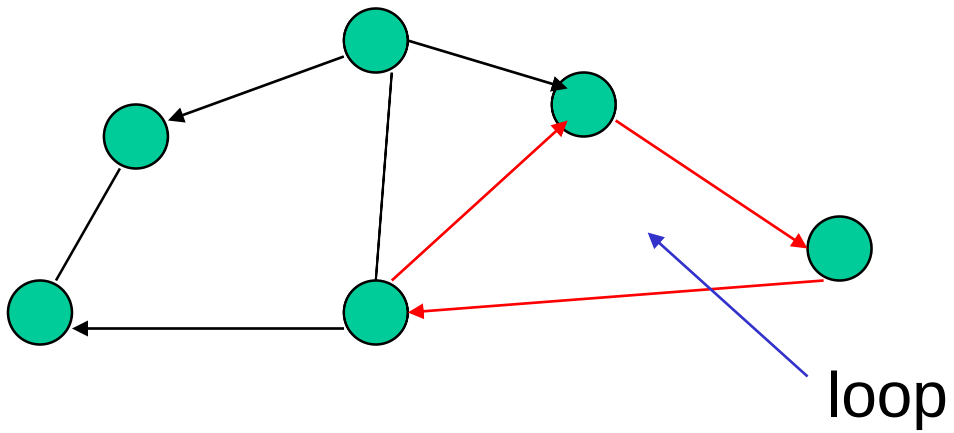
- Fails to show relationships well
- Look at problem using a graph

Two-frame of Animation



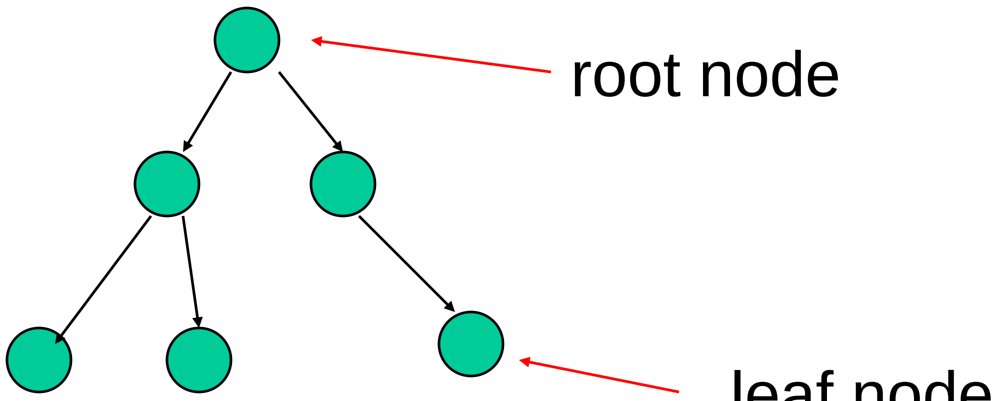
Graphs

- Set of *nodes* and *edges (links)*
- Edge connects a pair of nodes
  - Directed or undirected
- *Cycle*: directed path that is a loop

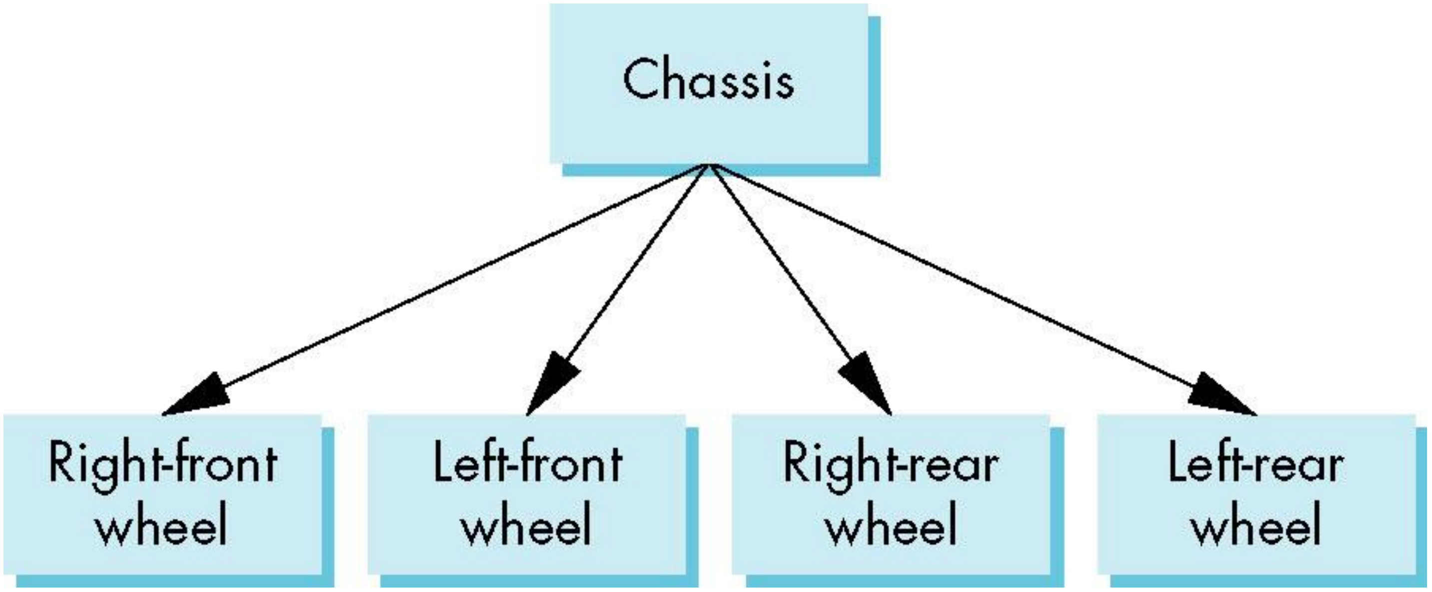


Tree

- Graph in which each node (except the root) has exactly one parent node
  - May have multiple children
  - Leaf or terminal node: no children

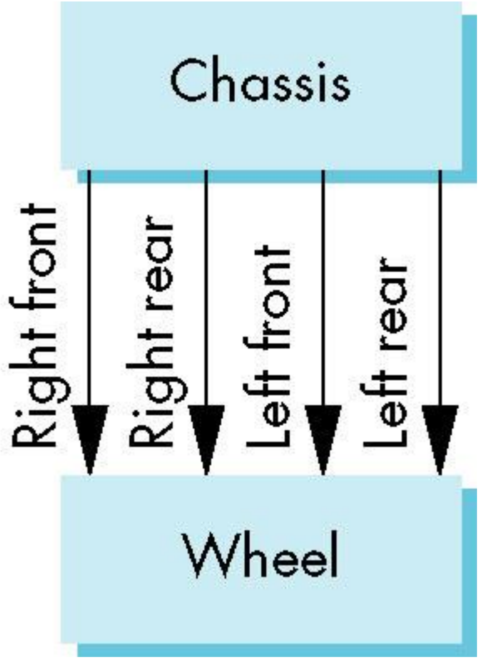


Tree Model of Car



DAG Model

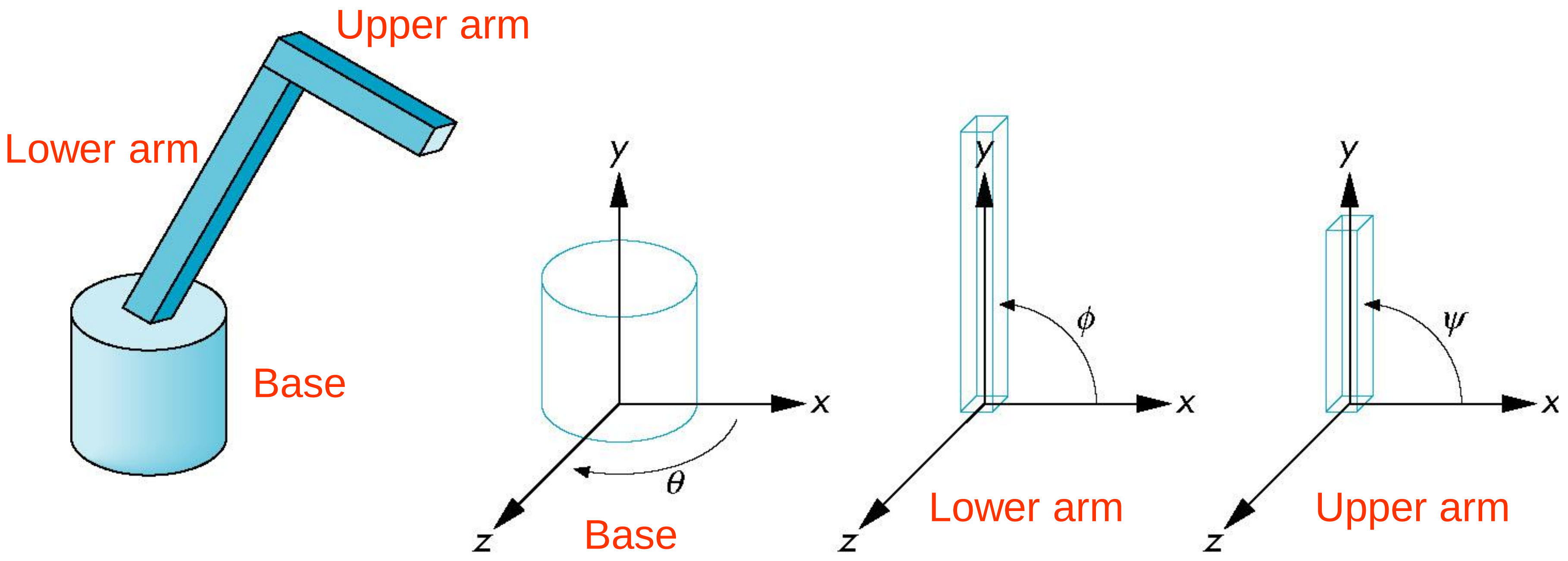
- If we use the fact that all the wheels are identical, we get a *directed acyclic graph*
  - Not much different than dealing with a tree



Modeling with Trees

- Must decide what information to place in nodes and what to put in edges
- Nodes
  - What to draw
  - Pointers to children
- Edges
  - May have information on incremental changes to transformation matrices (can also store in nodes)

Robot Arm

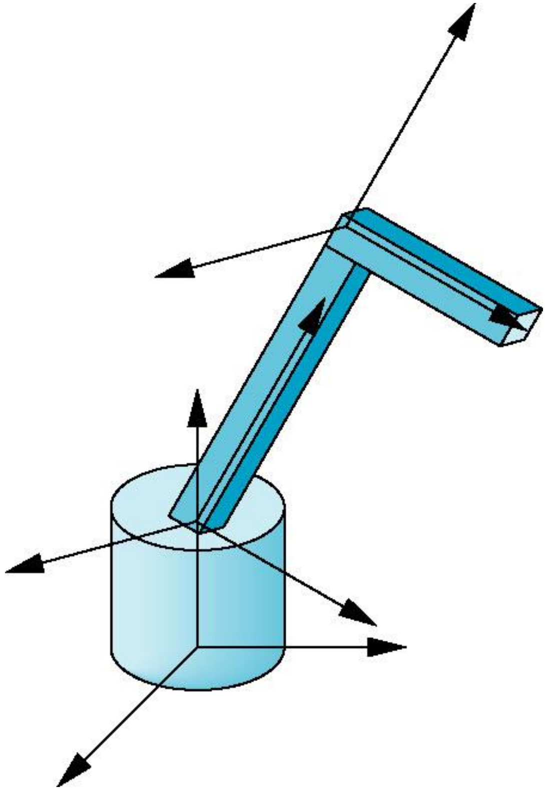


robot arm

parts in their own coodinate systems

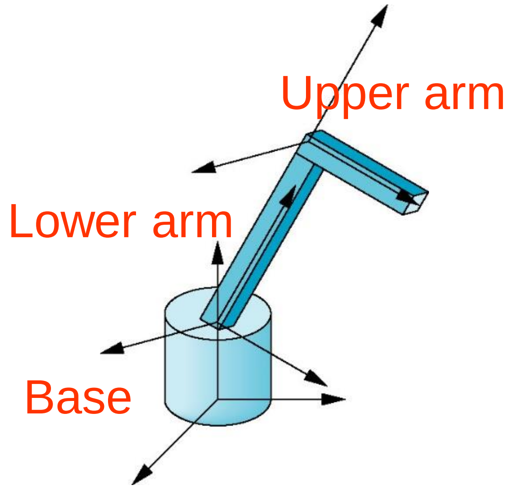
Articulated Models

- Robot arm is an example of an *articulated model*
  - Parts connected at joints
  - Can specify state of model by giving all joint angles

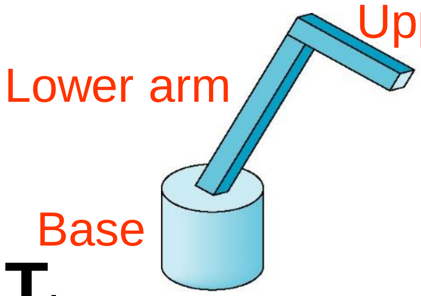


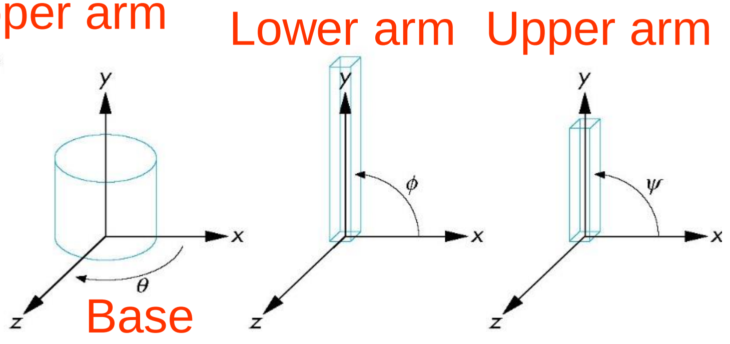
Relationships in Robot Arm

- Base rotates independently
  - Single angle determines position
- Lower arm attached to base
  - Its position depends on rotation of base
  - Must also translate relative to base and rotate about connecting joint
- Upper arm attached to lower arm
  - Its position depends on both base and lower arm
  - Must translate relative to lower arm and rotate about joint connecting to lower arm



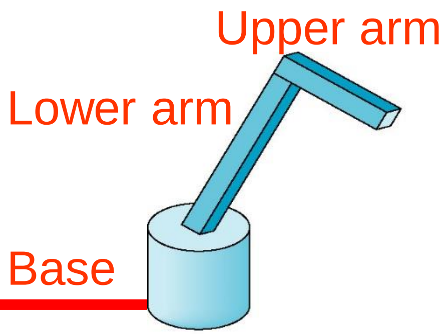
Required Matrices

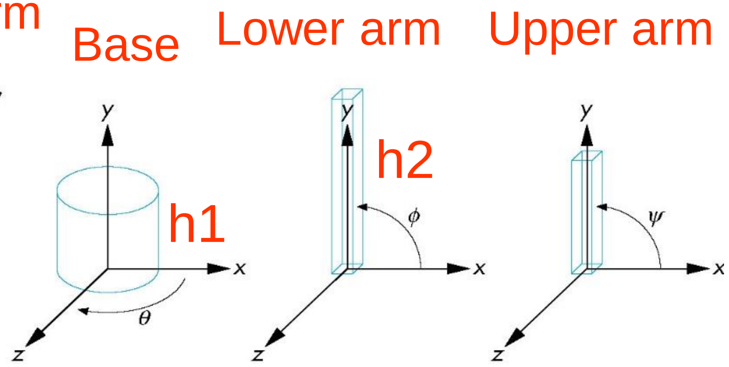




- Rotation of base: $\mathbf{R}_b$
  - Apply $\mathbf{M} = \mathbf{R}_b$ to base
- Translate lower arm <u>relative</u> to base: $\mathbf{T}_{lu}$
- Rotate lower arm around joint: $\mathbf{R}_{lu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu}$ to lower arm
- Translate upper arm <u>relative</u> to lower arm: $\mathbf{T}_{uu}$
- Rotate upper arm around joint: $\mathbf{R}_{uu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu} \mathbf{T}_{uu} \mathbf{R}_{uu}$ to upper arm

WebGL Code for Robot



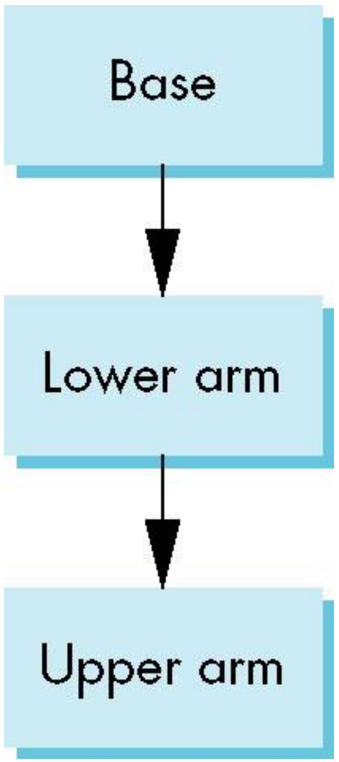



```
var render = function() {
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );
    modelViewMatrix = rotate(theta[Base], 0, 1, 0 );
    base();
    modelViewMatrix = mult(modelViewMatrix,
            translate(0.0, BASE_HEIGHT, 0.0));
    modelViewMatrix = mult(modelViewMatrix,
             rotate(theta[LowerArm], 0, 0, 1 ));
    lowerArm();
    modelViewMatrix  = mult(modelViewMatrix,
            translate(0.0, LOWER_ARM_HEIGHT, 0.0));
    modelViewMatrix  = mult(modelViewMatrix,
            rotate(theta[UpperArm], 0, 0, 1) );
    upperArm();
    requestAnimFrame(render);
}
```

$\boxed{\mathbf{M} = \mathbf{R}_b}$

$\boxed{\mathbf{M} = \mathbf{R}_b\ \mathbf{T}_{lu}\ \mathbf{R}_{lu}}$

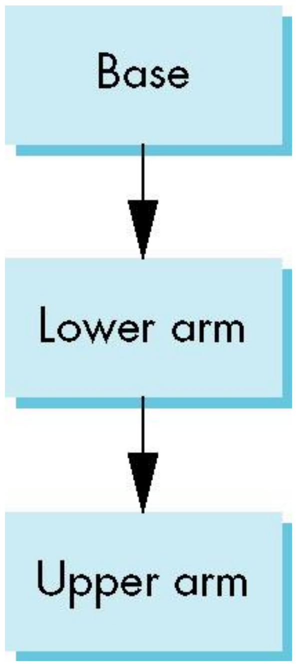
$\boxed{\mathbf{M} = \mathbf{R}_b\ \mathbf{T}_{lu}\ \mathbf{R}_{lu}\ \mathbf{T}_{uu}\ \mathbf{R}_{uu}}$



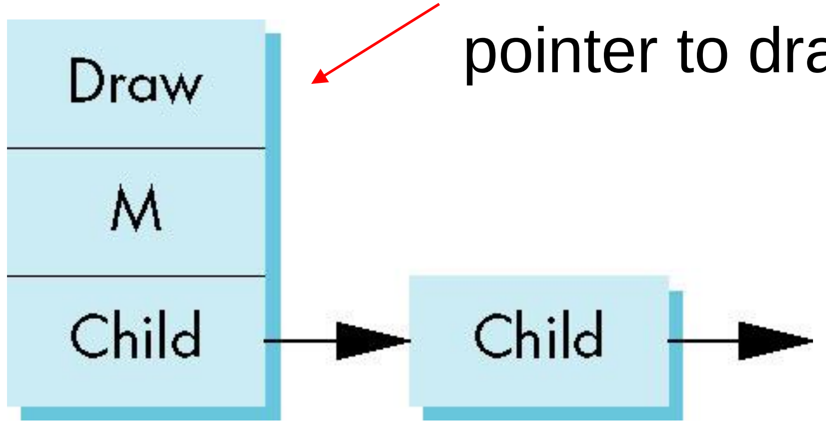
Tree Model of Robot

- Note code shows relationships between parts of model
  - Can change “look” of parts easily without altering relationships
- Simple example of tree model
- Want a general node structure

for nodes



Possible Node Structure



Code for drawing part or pointer to drawing function

linked list of pointers to children

matrix relating node to parent

Generalizations

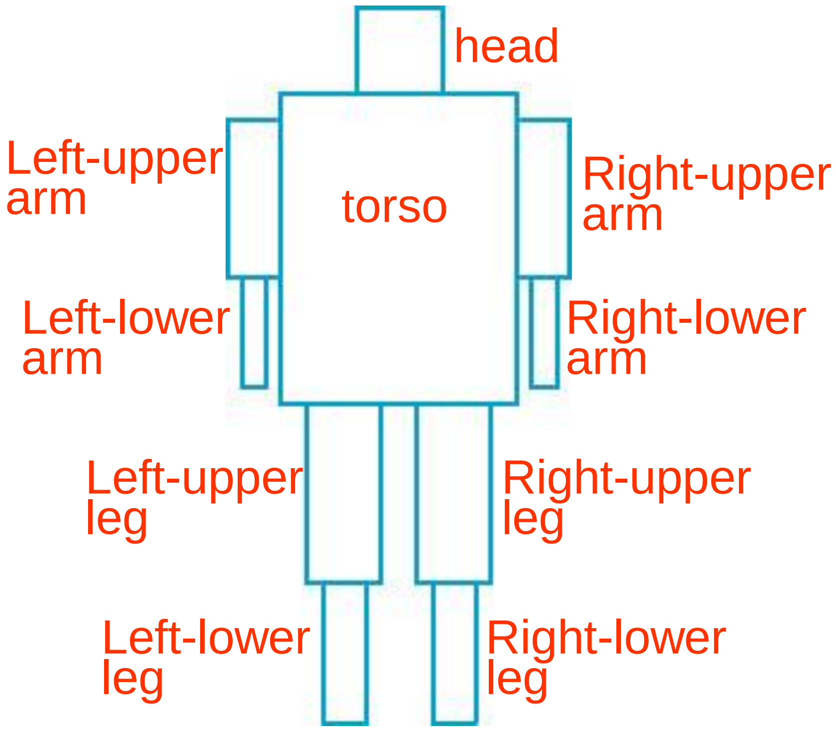
- Need to deal with multiple children
  - How do we represent a more general tree?
  - How do we traverse such a data structure?
- Animation
  - How to use dynamically?
  - Can we create and delete nodes during execution?

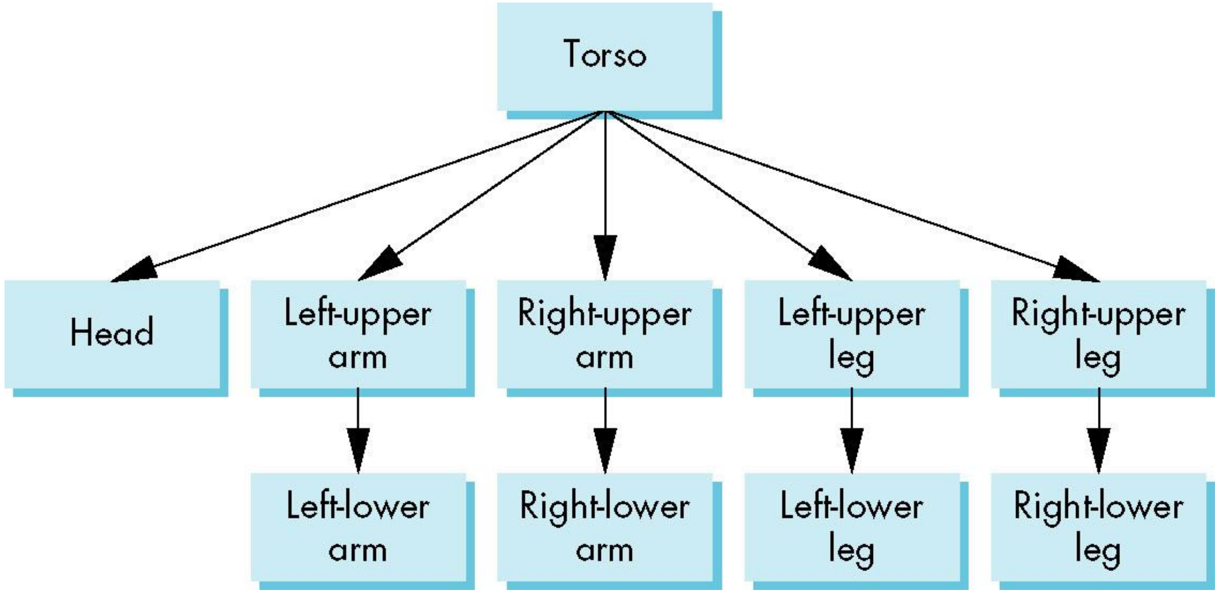
Hierarchical Modeling II

Objectives

- Build a tree-structured model of a humanoid figure
- Examine various traversal strategies
- Build a generalized tree-model structure that is independent of the particular model

Humanoid Figure

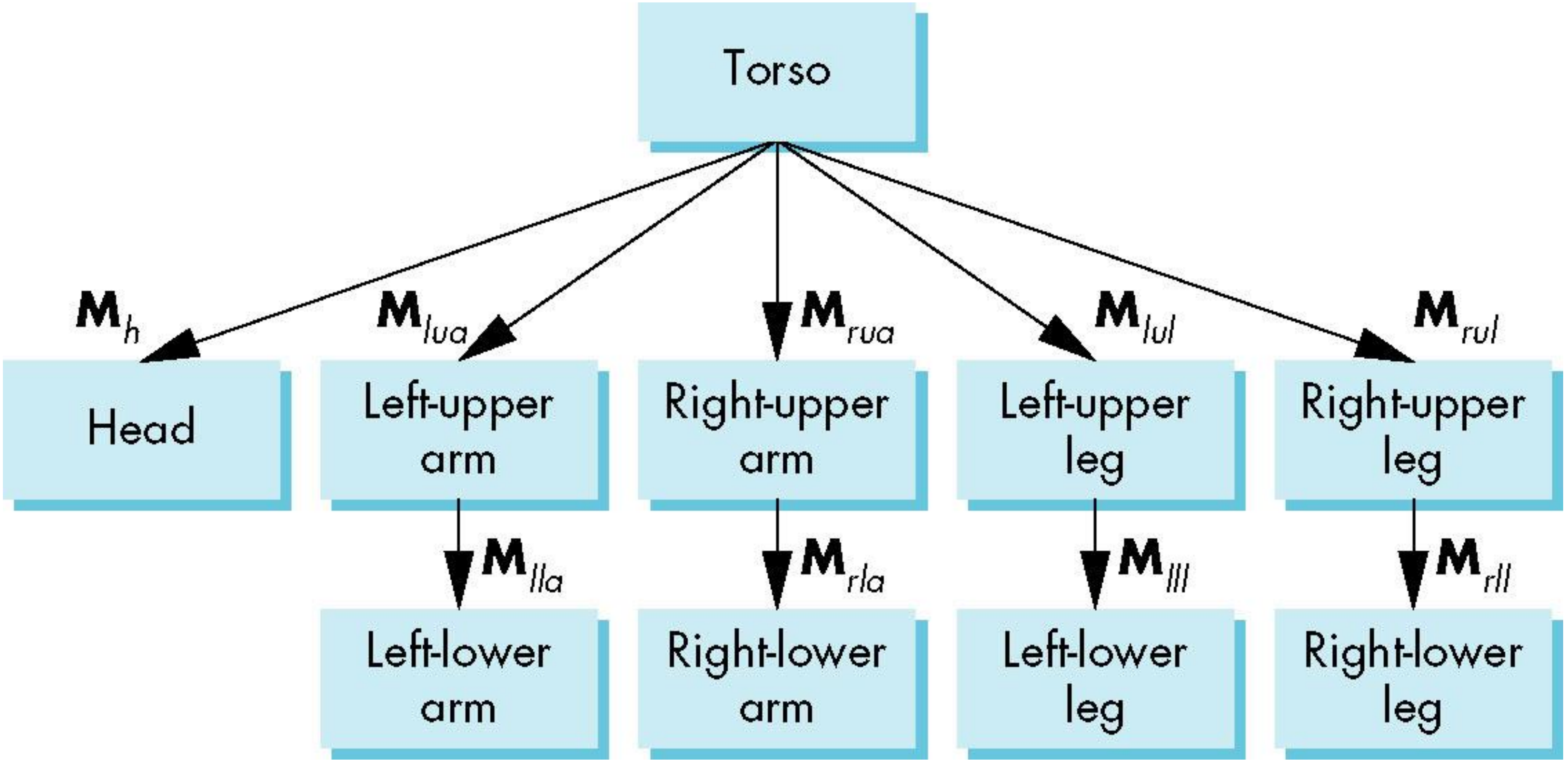




Building the Model

- Can build a simple implementation using quadrics: ellipsoids and cylinders
- Access parts through functions
  - `torso()`
  - `leftUpperArm()`
- Matrices describe position of node with respect to its parent
  - $\mathbf{M}_{lla}$ positions left lower leg with respect to left upper arm

Tree with Matrices

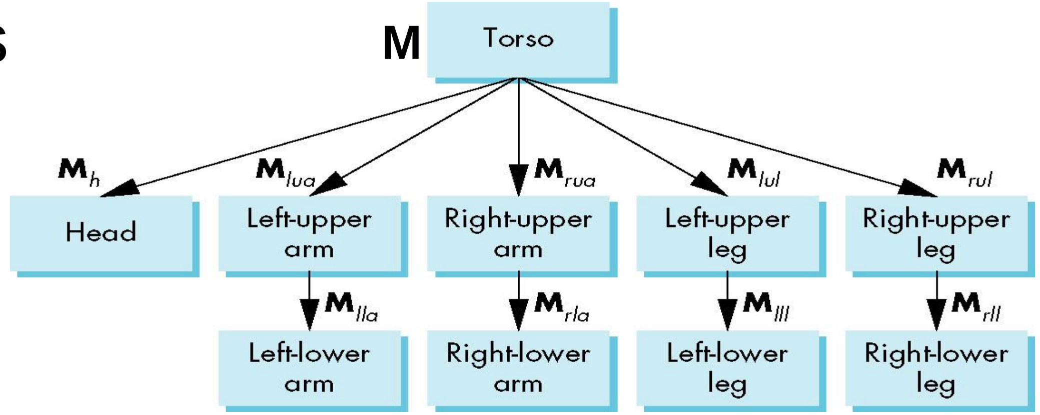


Display and Traversal

- The position of the figure is determined by *11 joint angles* (*two for the head and one for each other part*)
- Display of the tree requires a *graph traversal*
  - Visit each node once
  - Display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation

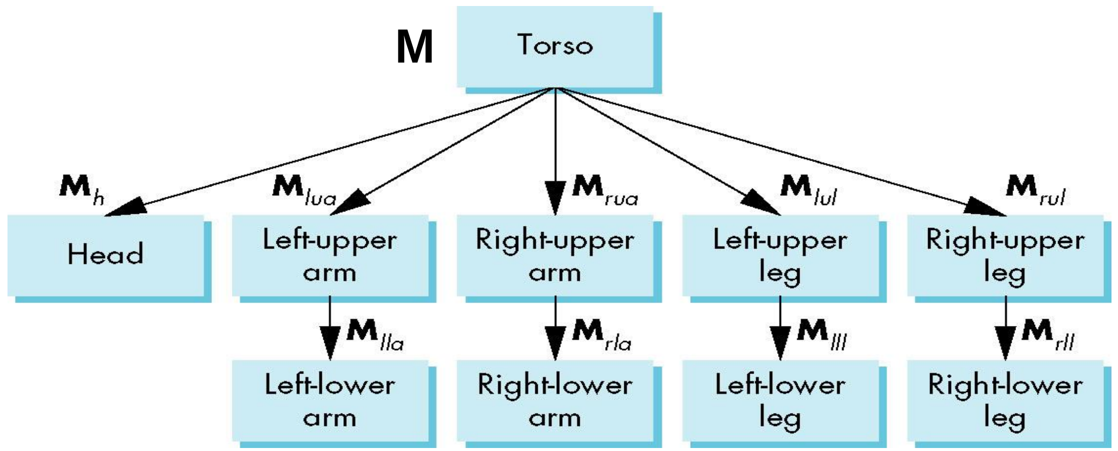
Transformation Matrices

- There are 10 relevant matrices
  - **M** positions and orients entire figure through the torso which is the root node
  - $\mathbf{M}_h$ positions head with respect to torso
  - $\mathbf{M}_{lua}$, $\mathbf{M}_{rua}$, $\mathbf{M}_{lul}$, $\mathbf{M}_{rul}$ position arms and legs with respect to torso
  - $\mathbf{M}_{lla}$, $\mathbf{M}_{rla}$, $\mathbf{M}_{lll}$, $\mathbf{M}_{rll}$ position lower parts of limbs with respect to corresponding upper limbs



Stack-based Traversal

- Set model-view matrix to **M** and draw torso
- Set model-view matrix to $\mathbf{MM}_h$ and draw head
- For left-upper arm need $\mathbf{MM}_{lua}$ and so on
- Rather than recomputing $\mathbf{MM}_{lua}$ from scratch or using an inverse matrix, we can use the ***matrix stack*** to store **M** and other matrices as we traverse the tree



Traversal Code

```
figure() {
    PushMatrix()
    torso();
    Rotate (…);
    head();
    PopMatrix();
    PushMatrix();
    Translate(…);
    Rotate(…);
    left_upper_arm();
    PopMatrix();
    PushMatrix();
```

save present model-view matrix

update model-view matrix for head

recover original model-view matrix

save it again

update model-view matrix for left upper arm

recover and save original model-view matrix again

rest of code

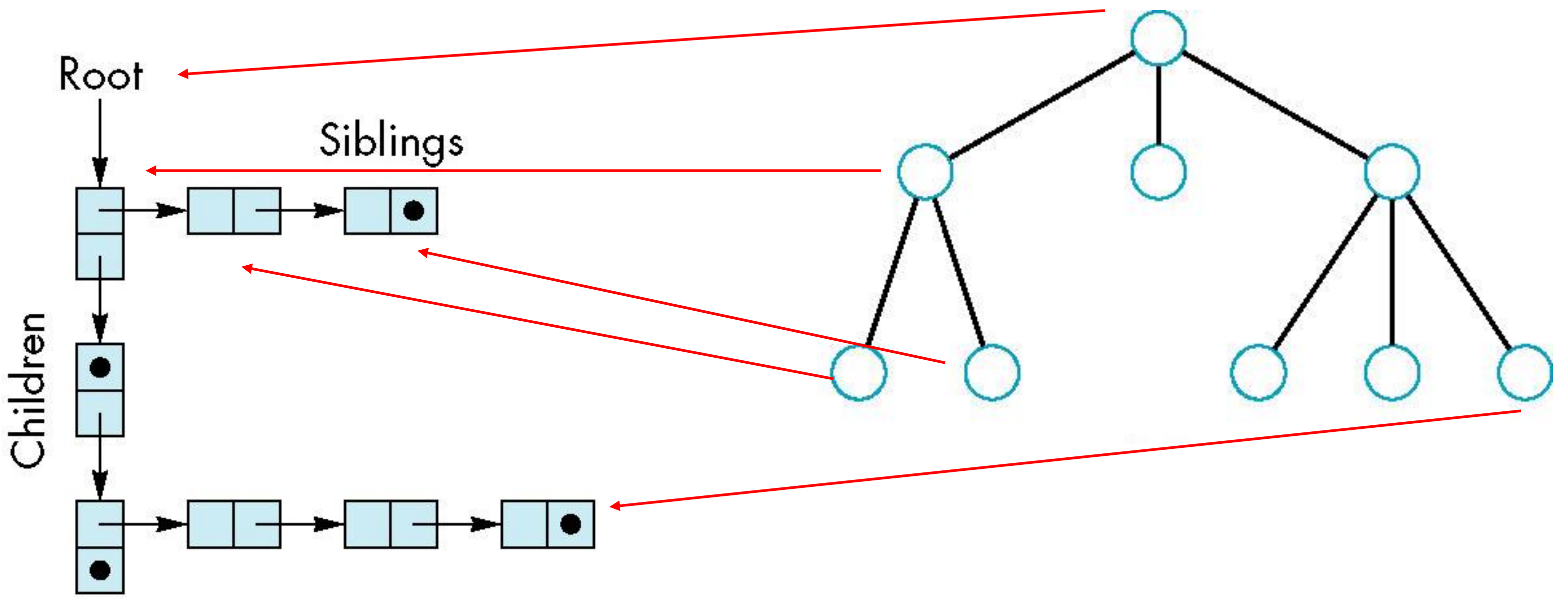
Analysis

- The code describes a particular tree and a particular traversal strategy
  - Can we develop a more general approach?
- Note that the sample code does not include state changes, such as changes to colors
  - May also want to push and pop other attributes to protect against unexpected state changes affecting later parts of the code

General Tree Data Structure

- Need a data structure to represent tree and an algorithm to traverse the tree
- We will use a *left-child right sibling* structure
  - Uses linked lists
  - Each node in data structure is two pointers
  - Left: next node
  - Right: linked list of children

Left-Child Right-Sibling Tree



Tree node Structure

- At each node we need to store
  - Pointer to sibling
  - Pointer to child
  - Pointer to a function that draws the object represented by the node
  - Homogeneous coordinate matrix to multiply on the right of the current model-view matrix
    - Represents changes going from parent to node
    - In WebGL this matrix is a 1D array storing matrix by columns

Creating a treenode

```
function createNode(transform,
            render, sibling, child) {
    var node = {
    transform: transform,
    render: render,
    sibling: sibling,
    child: child,
    }
    return node;
};
```

Initializing Nodes

```
function initNodes(Id) {
    var m = mat4();
        switch(Id) {
        case torsoId:
           m = rotate(theta[torsoId], 0, 1, 0 );
           figure[torsoId] = createNode( m, torso, null, headId );
           break;
       case head1Id:
       case head2Id:
           m = translate(0.0, torsoHeight+0.5*headHeight, 0.0);
           m = mult(m, rotate(theta[head1Id], 1, 0, 0))m = mult(m,
                        rotate(theta[head2Id], 0, 1, 0));
           m = mult(m, translate(0.0, -0.5*headHeight, 0.0));
           figure[headId] = createNode( m, head, leftUpperArmId, null);
           break;
```

Notes

- The position of figure is determined by 11 joint angles stored in `theta[11]`
- Animate by changing the angles and redisplaying
- We form the required matrices using `rotate` and `translate`
- Because the matrix is formed using the model-view matrix, we may want to first push original model-view matrix on matrix stack

Preorder Traversal

```
function traverse(Id) {
   if(Id == null) return;
   stack.push(modelViewMatrix);
   modelViewMatrix = mult(modelViewMatrix, figure[Id].transform);
   figure[Id].render();
   if(figure[Id].child != null) traverse(figure[Id].child);
   modelViewMatrix = stack.pop();
   if(figure[Id].sibling != null) traverse(figure[Id].sibling);
 }
var render = function() {
        gl.clear( gl.COLOR_BUFFER_BIT );
        traverse(torsoId);
        requestAnimFrame(render);
}
```

Notes

- We must save model-view matrix before multiplying it by node matrix
  - Updated matrix applies to children of node but not to siblings which contain their own matrices
- The traversal program applies to any left-child right-sibling tree
  - The particular tree is encoded in the definition of the individual nodes
- The order of traversal matters because of possible state changes in the functions

Dynamic Trees

- Because we are using JS, the nodes and the node structure can be changed during execution
- Definition of nodes and traversal are essentially the same as before but we can add and delete nodes during execution
- In desktop OpenGL, if we use pointers, the structure can be dynamic

Graphical Objects and Scene Graphs 1

Objectives

- Introduce graphical objects
- Generalize the notion of objects to include lights, cameras, attributes
- Introduce scene graphs

Limitations of Immediate Mode Graphics

- When we define a geometric object in an application, upon execution of the code the object is passed through the pipeline
- It then disappeared from the graphical system
- To redraw the object, either changed or the same, we had to reexecute the code
- Display lists provided only a partial solution to this problem

Retained Mode Graphics

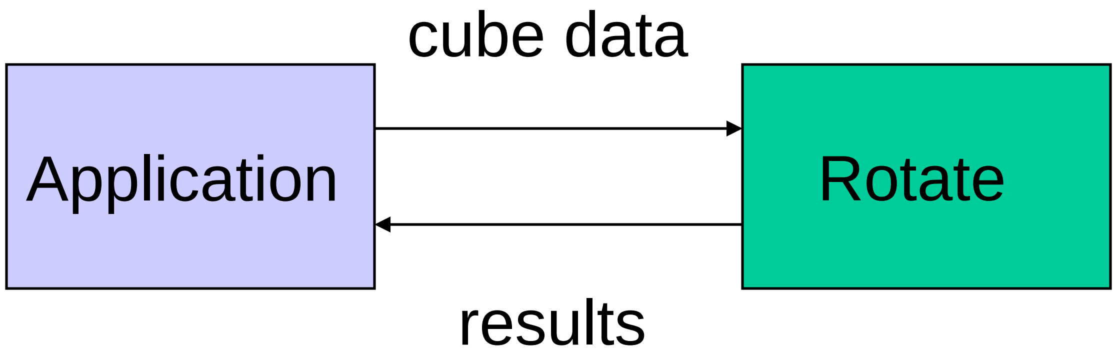
- Display lists were server side
- GPUs allowed data to be stored on GPU
- Essentially all immediate mode functions have been deprecated
- Nevertheless, OpenGL is a low level API

OpenGL and Objects

- OpenGL lacks an object orientation
- Consider, for example, a green sphere
  - We can model the sphere with polygons
  - Its color is determined by the OpenGL state and is not a property of the object
  - Loose linkage with vertex attributes
- Defies our notion of a physical object
- We can try to build better objects in code using object-oriented languages/techniques

Imperative Programming Model

- Example: rotate a cube



- The rotation function must know how the cube is represented
  - Vertex list
  - Edge list

Object-Oriented Programming Model

- In this model, the representation is stored with the object

- The application sends a *message* to the object
- The object contains functions (*methods*) which allow it to transform itself

C/C++/Java/JS

- Can try to use C structs to build objects
- C++/Java/JS provide better support
  - Use class construct
  - With C++ we can hide implementation using public, private, and protected members i
  - JS provides multiple methods for object

Cube Object

- Suppose that we want to create a simple cube object that we can scale, orient, position and set its color directly through code such as

```
var mycube = new Cube();
mycube.color[0]=1.0;
mycube.color[1]= mycube.color[2]=0.0;
mycube.matrix[0][0]=………
```

Cube Object Functions

- We would also like to have functions that act on the cube such as
  - **`mycube.translate(1.0, 0.0,0.0);`**
  - **`mycube.rotate(theta, 1.0, 0.0, 0.0);`**
  - **`setcolor(mycube, 1.0, 0.0, 0.0);`**
- We also need a way of displaying the cube
  - **`mycube.render();`**

Building the Cube Object

```
var cube {
        var color[3];
        var matrix[4][4];

}
```

The Implementation

- Can use any implementation in the private part such as a vertex list
- The private part has access to public members and the implementation of class methods can use any implementation without making it visible
- Render method is tricky but it will invoke the standard OpenGL drawing functions

Other Objects

- Other objects have geometric aspects
  - Cameras
  - Light sources
- But we should be able to have nongeometric objects too
  - Materials
  - Colors
  - Transformations (matrices)

JS Objects

```
cube mycube;

material plastic;
mycube.setMaterial(plastic);

camera frontView;
frontView.position(x ,y, z);
```

JS Objects

- Can create much like Java or C++ objects
  - constructors
  - prototypes
  - methods
  - private methods and variables

```
var myCube = new Cube();
myCube.color = [1.0, 0.0, 0.0];
myCube.instance = …….
```

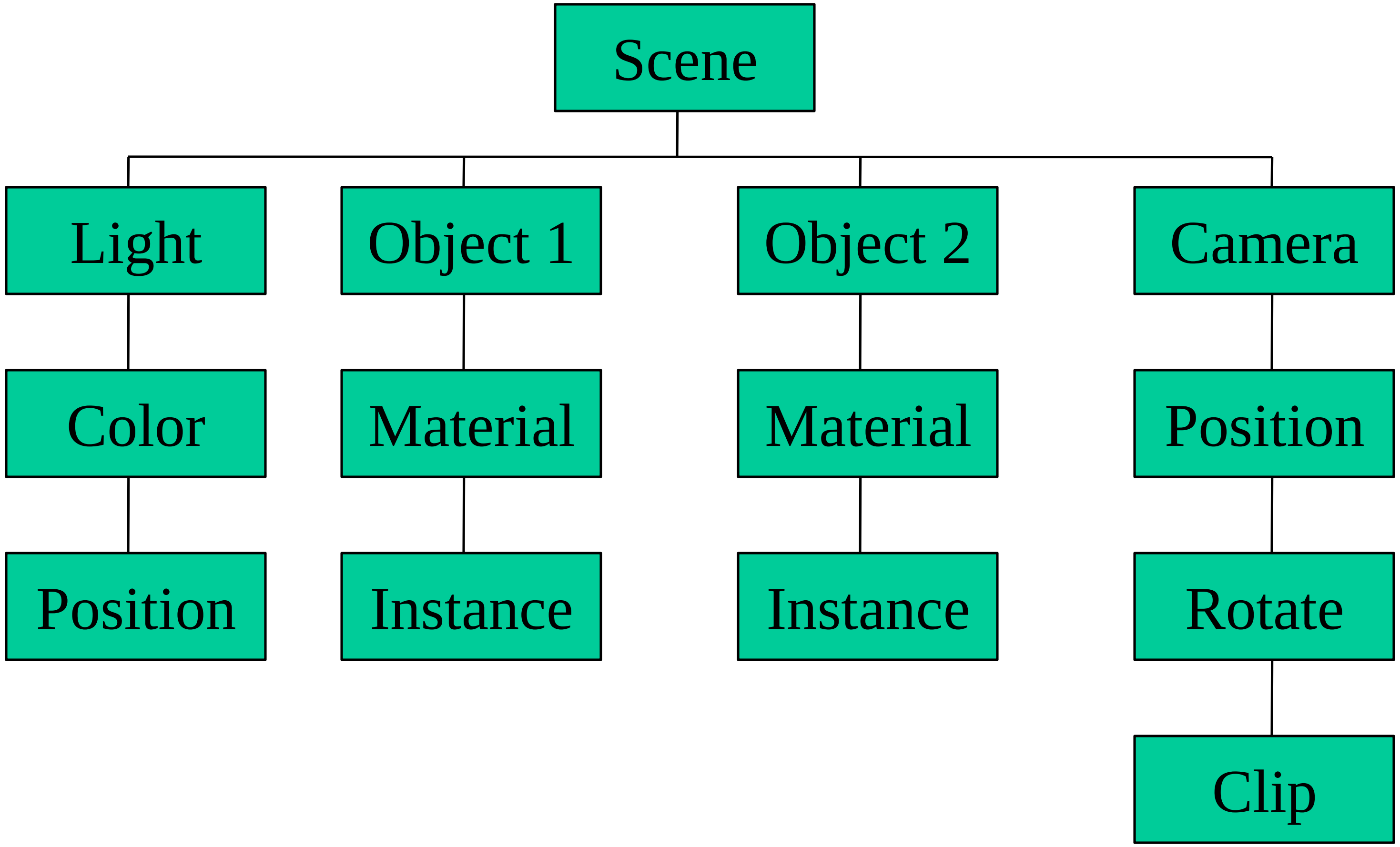
Light Object

```
var myLight = new Light();

// match Phong model

    myLight.type = 0; //directional
    myLight.position = ......;
    myLight.orientation = ......;
    myLight.specular = ......;
    myLight.diffuse = ......;
    myLight.ambient = ......;
}
```

Scene Descriptions

- If we recall figure model, we saw that
  - We could describe model either by tree or by equivalent code
  - We could write a generic traversal to display
- If we can represent all the elements of a scene (cameras, lights,materials, geometry) as JS objects, we should be able to show them in a tree
  - Render scene by traversing this tree

Scene Graph


Scene
Light
Object 1
Object 2
Camera
Color
Material
Material
Position
Position
Instance
Instance
Rotate
Clip

Traversal

```
myScene = new Scene();
myLight = new Light();
myLight.Color = ……;
…
myscene.Add(myLight);
object1 = new Object();
object1.color = …
myscene.add(object1);
…
…
myscene.render();
```

Graphical Objects and Scene Graphs 2

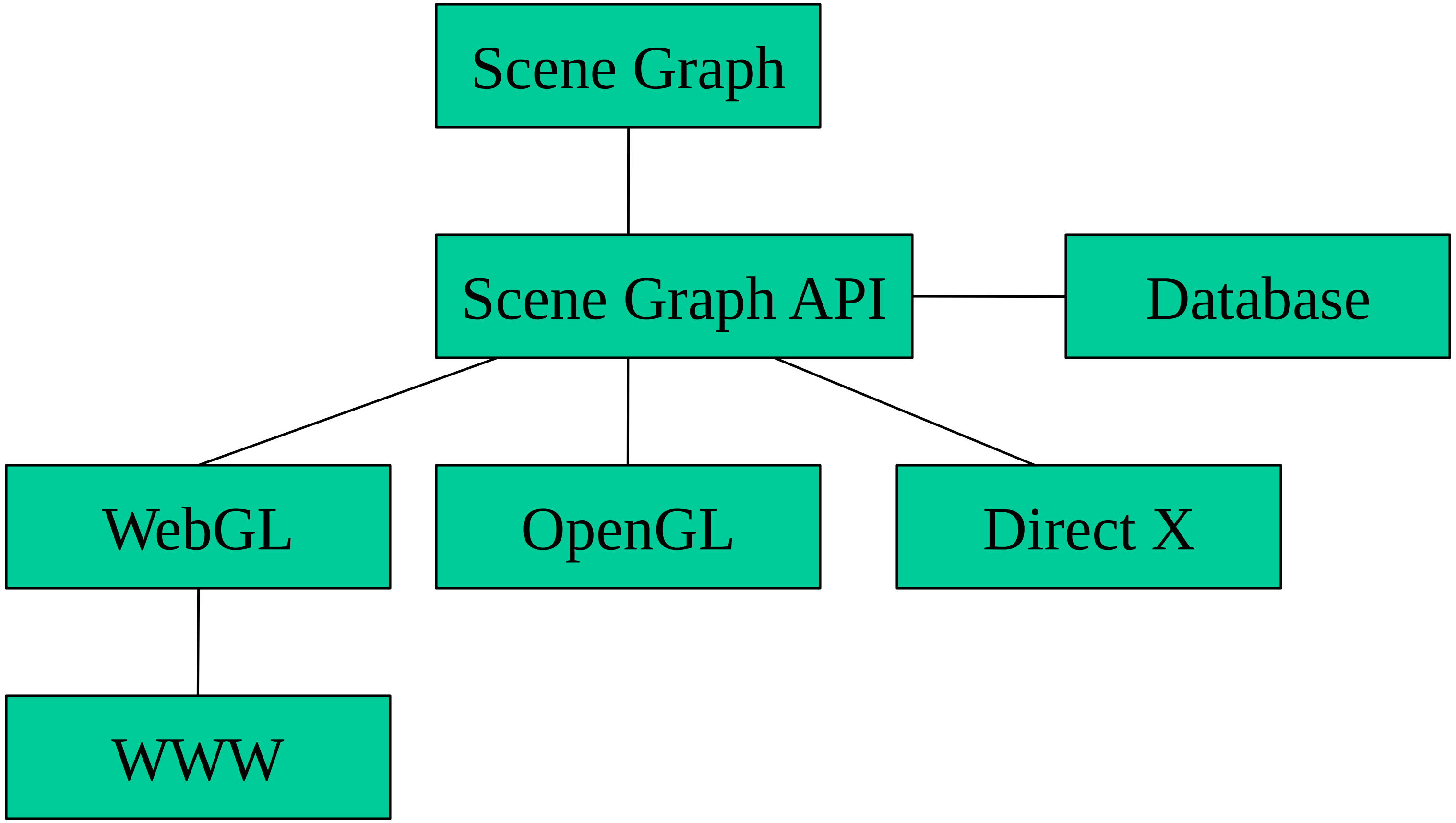
Objectives

- Look at some real scene graphs
- three.js (threejs.org)
- Scene graph rendering

Scene Graph History

- OpenGL development based largely on people who wanted to exploit hardware
  - real time graphics
  - animation and simulation
  - stand-alone applications
- CAD community needed to be able to share databases
  - real time not and photorealism not issues
  - need cross-platform capability
  - first attempt: PHIGS

Scene Graph Organization



Inventor and Java3D

- Inventor and Java3D provide a scene graph API
- Scene graphs can also be described by a file (text or binary)
  - Implementation independent way of transporting scenes
  - Supported by scene graph APIs
- However, primitives supported should match capabilities of graphics systems
  - Hence most scene graph APIs are built on top of OpenGL, WebGL or DirectX (for PCs)

VRML

- Want to have a scene graph that can *be used over the World Wide Web*
- Need links to other sites to support distributed data bases
- Virtual Reality Markup Language
  - Based on Inventor data base
  - Implemented with OpenGL

Open Scene Graph

- Supports very complex geometries by adding occulusion culling in first pass
- Supports translucently through a second pass that sorts the geometry
- First two passes yield a geometry list that is rendered by the pipeline in a third pass

three.js

- Popular scene graph built on top of WebGL
  - also supports other renderers
- See threejs.org
  - easy to download
  - many examples
- Also Eric Haines' Udacity course
- Major differences in approaches to computer graphics

three.js scene

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, window.innerWidth/
window.innerHeight, 0.1, 1000);

 var renderer = new THREE.WebGLRenderer();
 renderer.setSize(window.innerWidth, window.innerHeight);
 document.body.appendChild(renderer.domElement);

 var geometry = new THREE.CubeGeometry(1,1,1);
 var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
 var cube = new THREE.Mesh(geometry, material);
 scene.add(cube);
 camera.position.z = 5;
```

three.js render loop

```
var render = function () {
 requestAnimationFrame(render);
 cube.rotation.x += 0.1;
 cube.rotation.y += 0.1;
 renderer.render(scene, camera);
 };
 render();
```

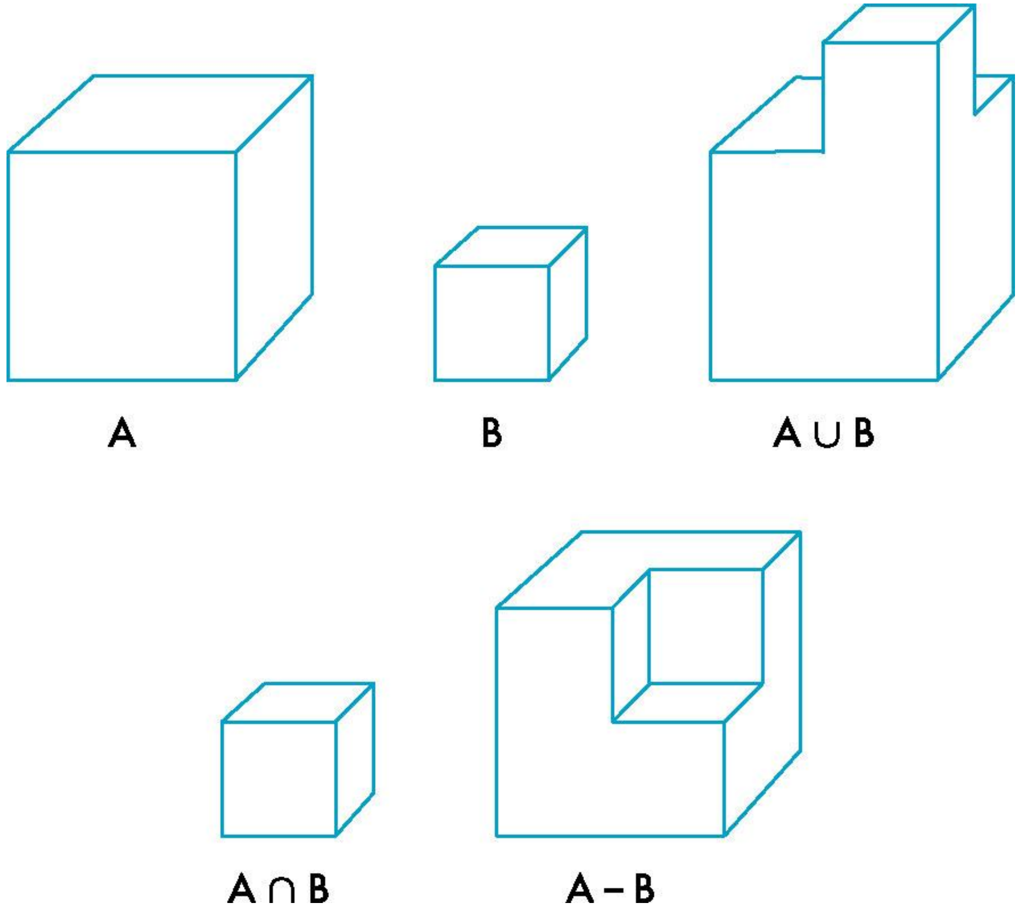
Other Tree Structures

Other Tree Structures

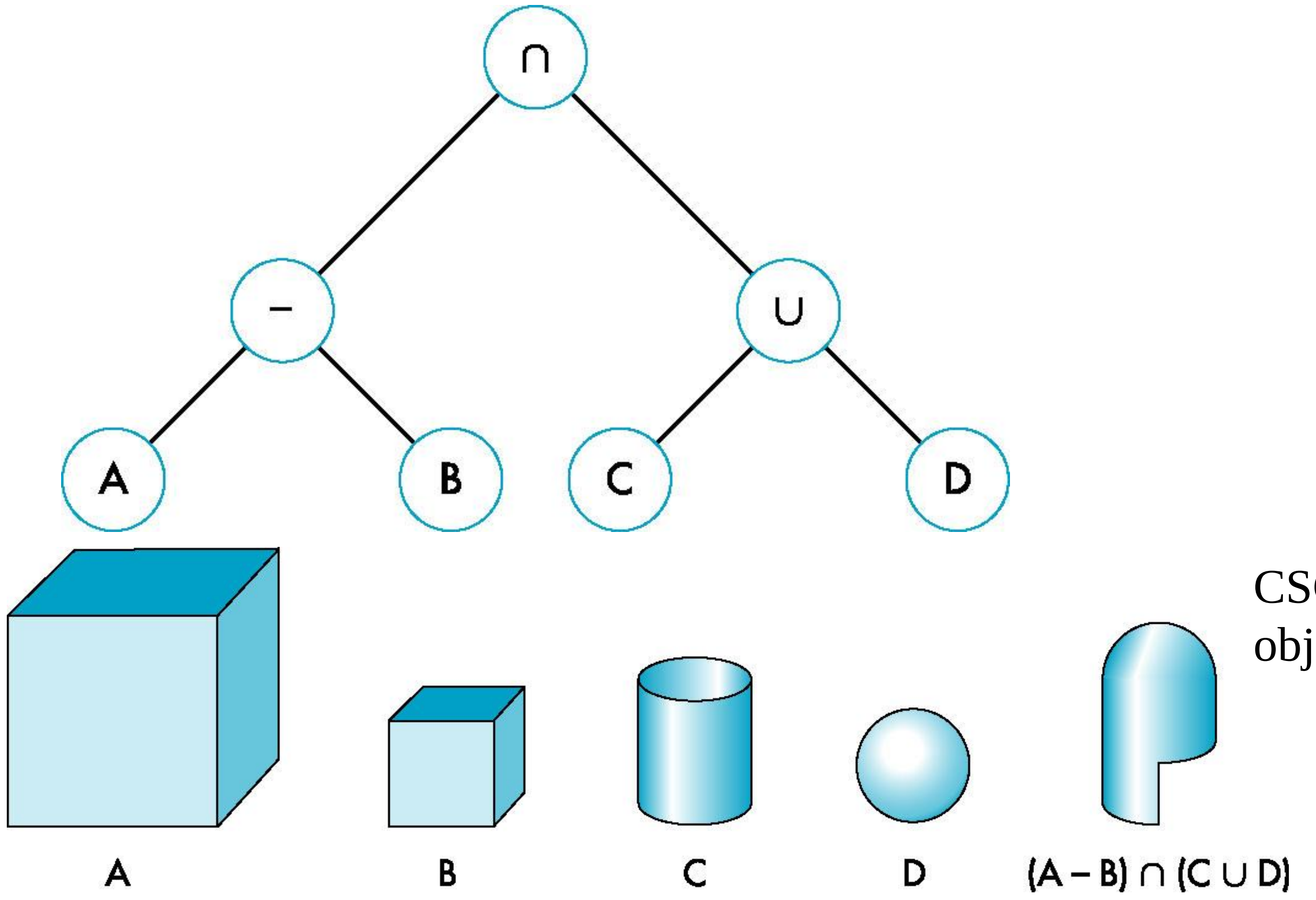
- Constructive Solid Geometry (CSG) Trees
- Binary Spatial-Partition (BSP) Trees
- Quadtrees and Octrees

CSG Trees

- Set operators: $\cap$, $\cup$, $-$

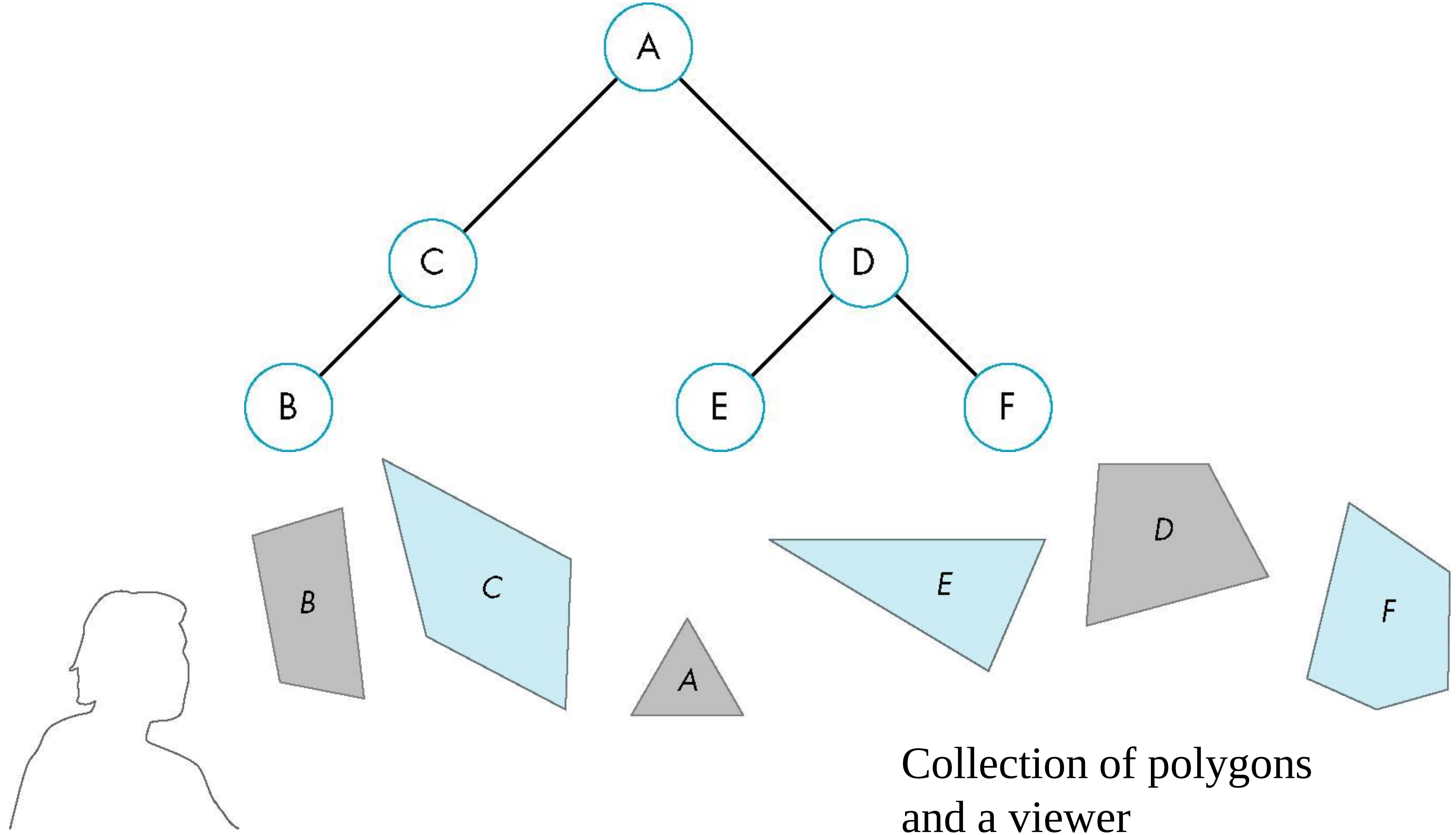


CSG Trees



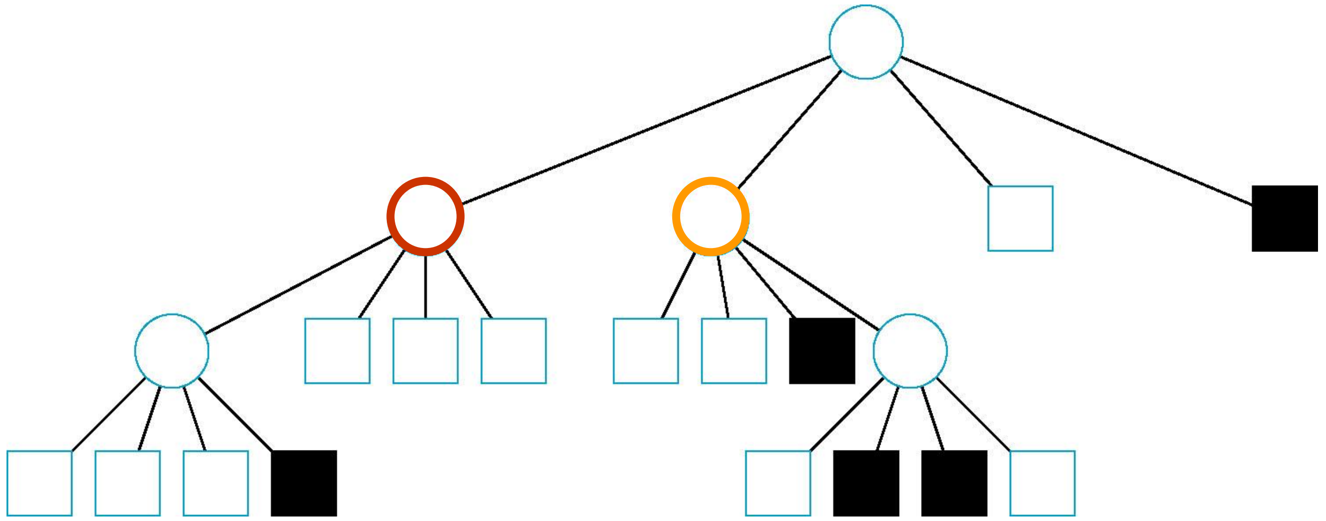
CSG object

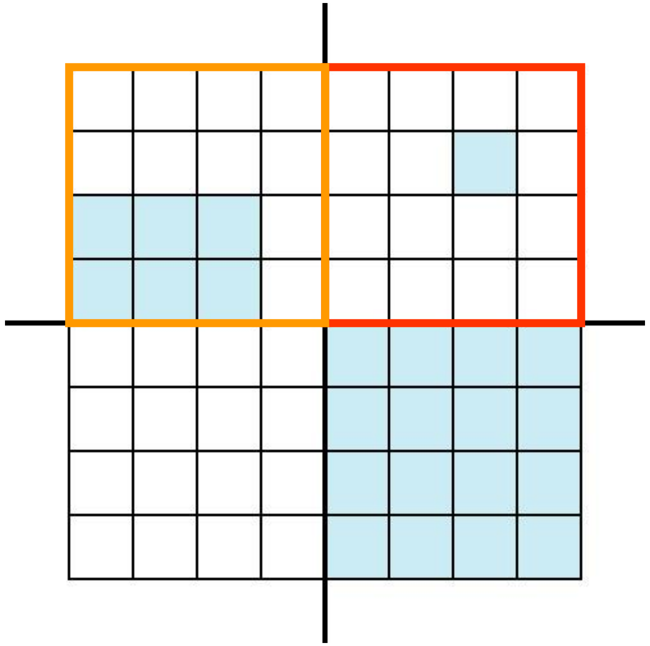
BSP Trees



Collection of polygons
and a viewer

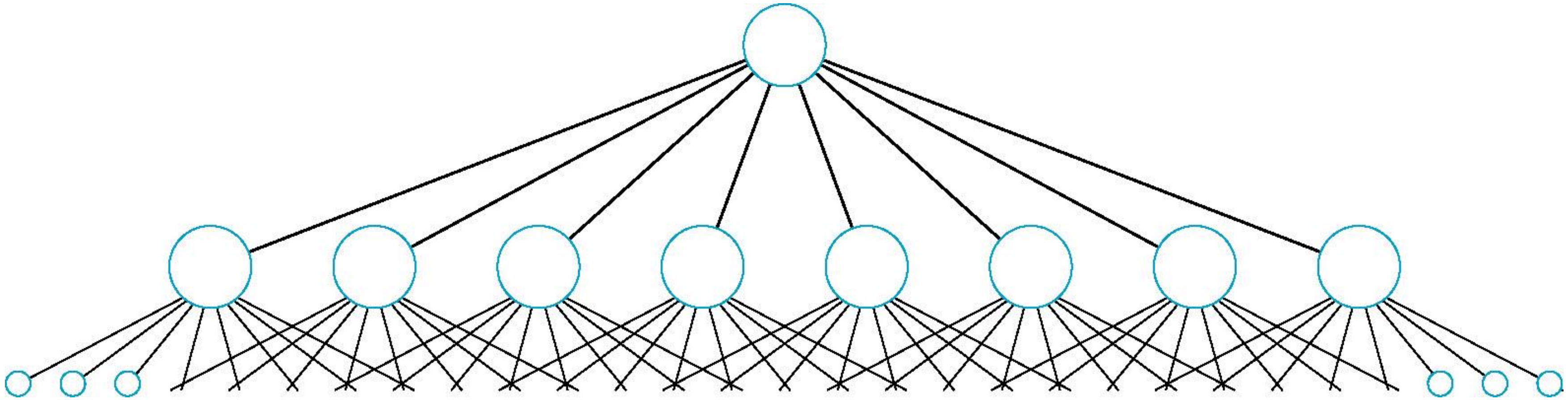
Quadtrees



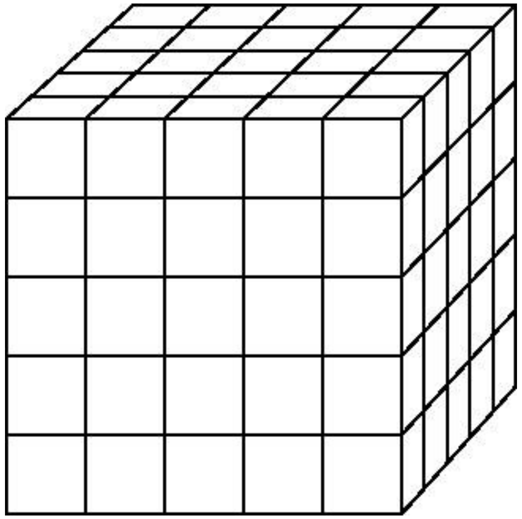


First subdivision of space

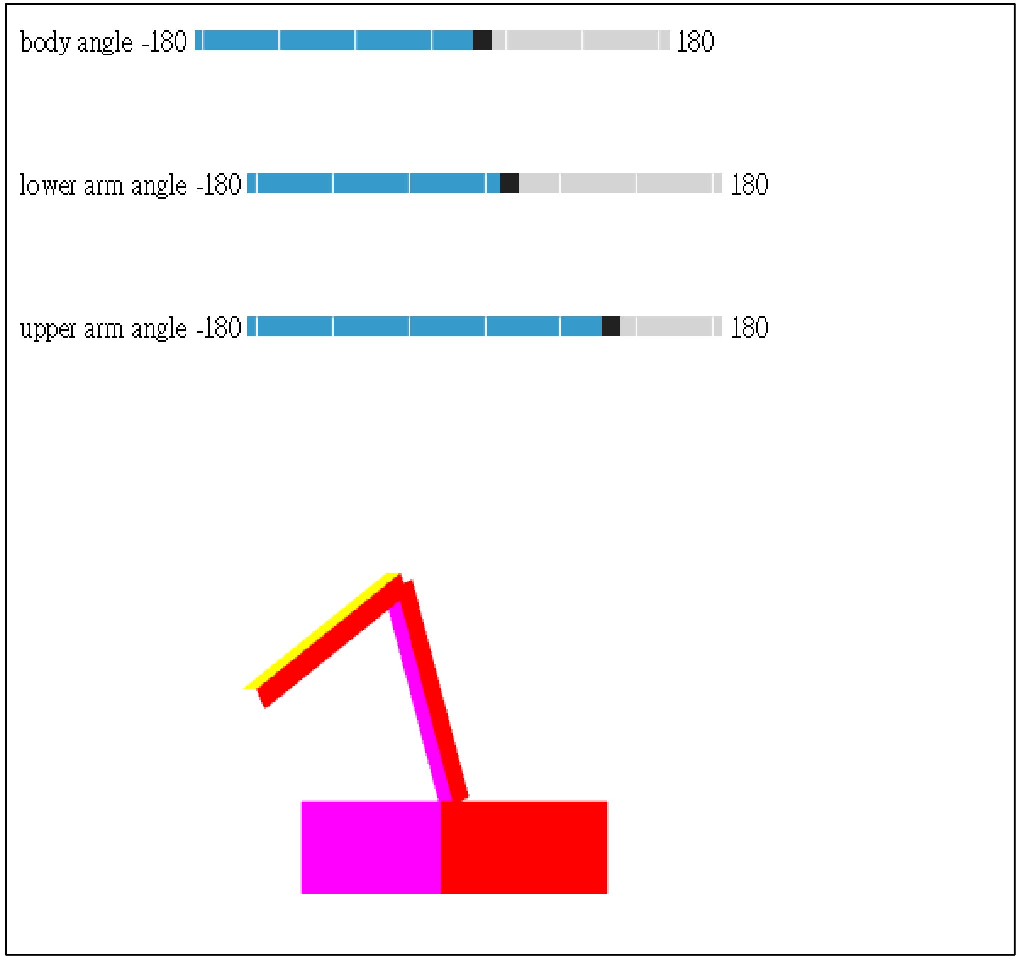
Octrees



Volume data set



Sample Programs: robotArm.html, robotArm.js



robotArm.html (1/5)

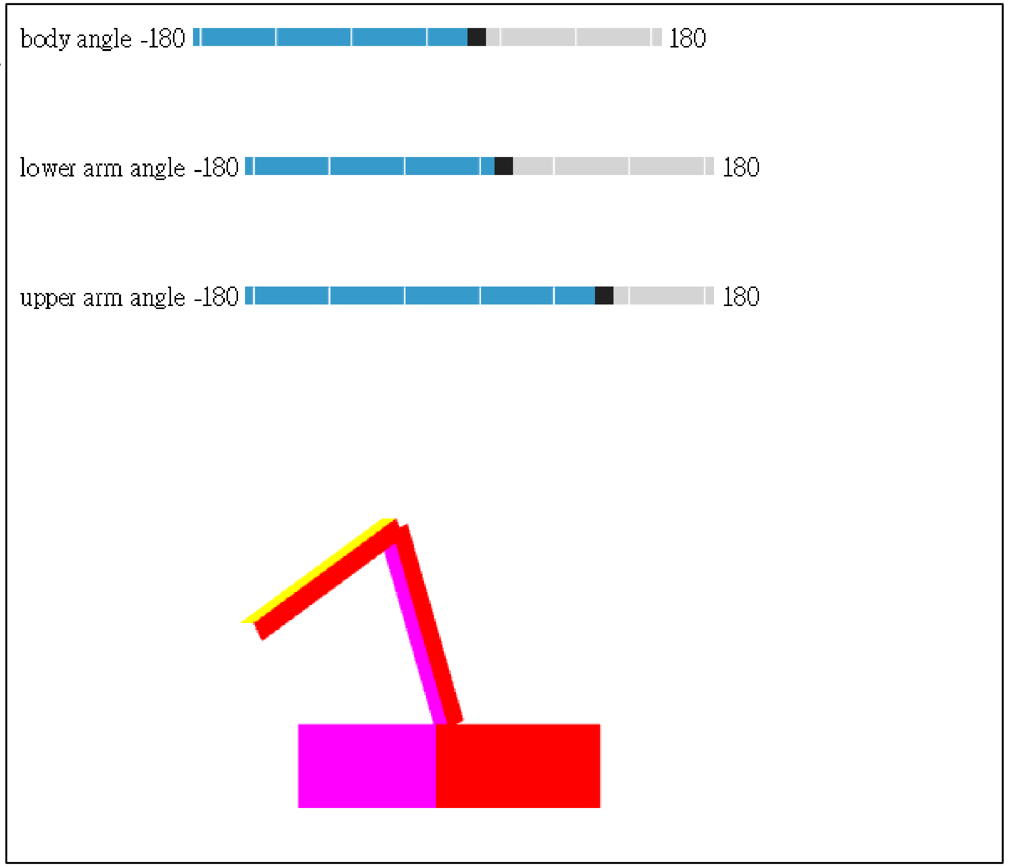
```
<html>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute  vec4 vPosition;
attribute  vec4 vColor;
varying vec4 fColor;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main()
{
    fColor = vColor;
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
}
</script>
```

robotArm.html (2/5)

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying  vec4 fColor;

void main()
{
    gl_FragColor = fColor;
}
</script>
```

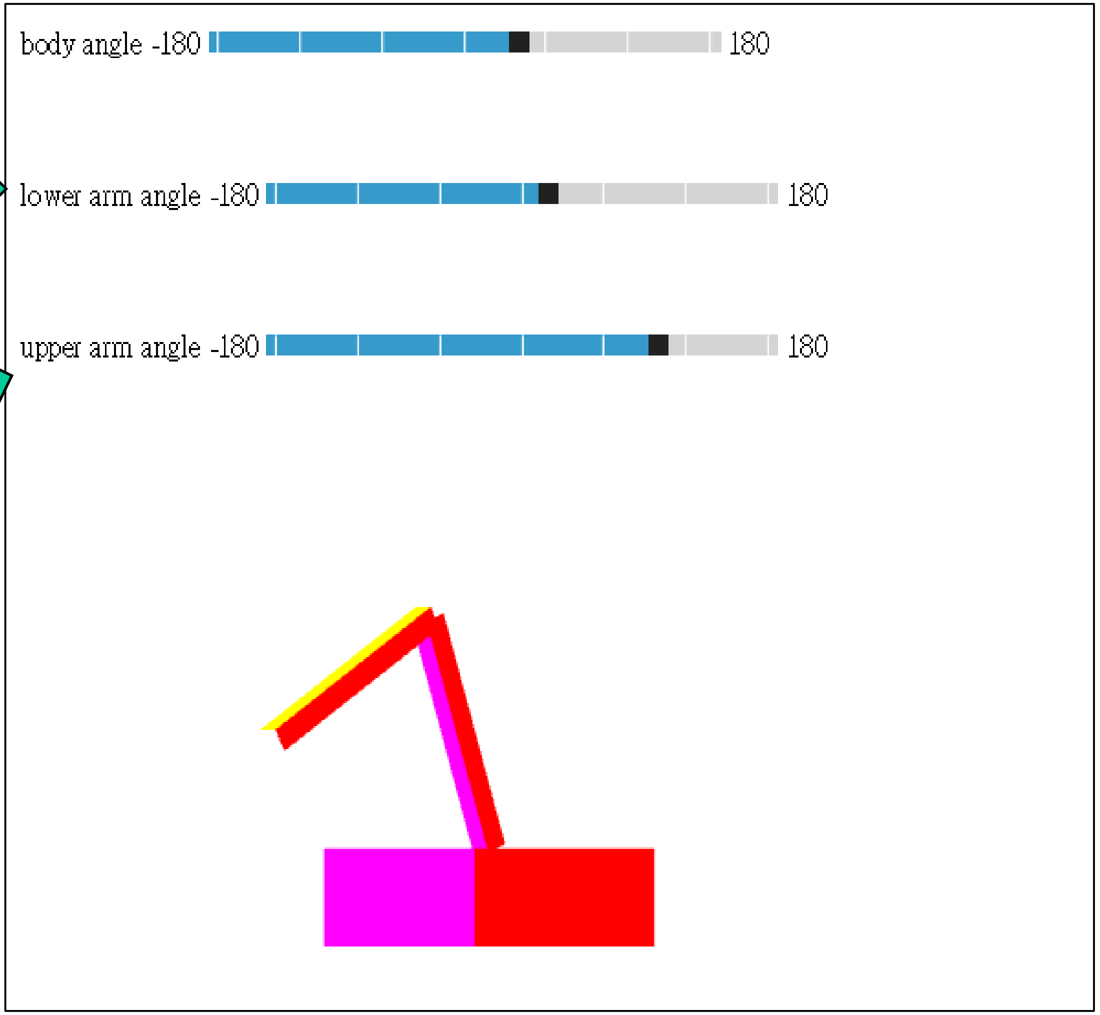
robotArm.html (3/5)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="robotArm.js"></script>

<div id="slider1">
body angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```



robotArm.html (4/5)

```
<div id="slider2">
lower arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>

<div id="slider3">
upper arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```



robotArm.html (5/5)

```
<body>
<canvas id="gl-canvas" width="512"" height="512"
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

robotArm.js (1/11)

```
var NumVertices = 36; //(6 faces)(2 triangles/face)(3 vertices/triangle)

var points = [];
var colors = [];

var vertices = [
    vec4( -0.5, -0.5,  0.5, 1.0 ),
    vec4( -0.5,  0.5,  0.5, 1.0 ),
    vec4(  0.5,  0.5,  0.5, 1.0 ),
    vec4(  0.5, -0.5,  0.5, 1.0 ),
    vec4( -0.5, -0.5, -0.5, 1.0 ),
    vec4( -0.5,  0.5, -0.5, 1.0 ),
    vec4(  0.5,  0.5, -0.5, 1.0 ),
    vec4(  0.5, -0.5, -0.5, 1.0 )
];
```

robotArm.js (2/11)

```
// RGBA colors
var vertexColors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 1.0, 1.0, 1.0, 1.0 ),  // white
    vec4( 0.0, 1.0, 1.0, 1.0 )   // cyan
];
// Parameters controlling the size of the Robot's arm
var BASE_HEIGHT      = 2.0;
var BASE_WIDTH       = 5.0;
var LOWER_ARM_HEIGHT = 5.0;
var LOWER_ARM_WIDTH  = 0.5;
var UPPER_ARM_HEIGHT = 5.0;
var UPPER_ARM_WIDTH  = 0.5;
```

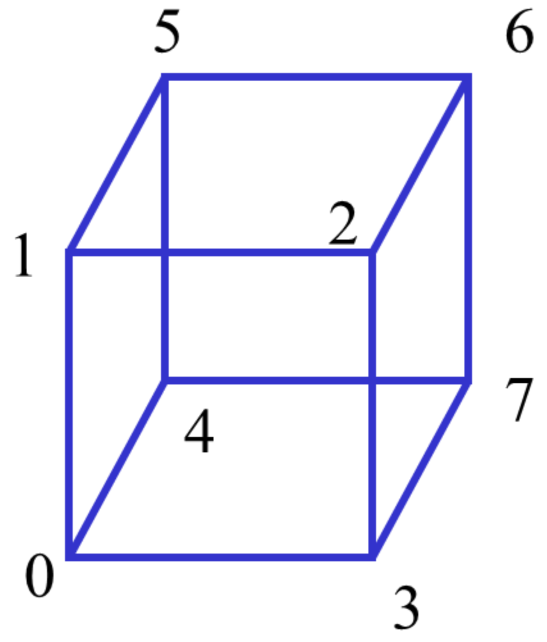
robotArm.js (3/11)

```
// Shader transformation matrices
var modelViewMatrix, projectionMatrix;

// Array of rotation angles (in degrees) for each
rotation axis

var Base = 0;
var LowerArm = 1;
var UpperArm = 2;

var theta= [ 0, 0, 0];

var angle = 0;

var modelViewMatrixLoc;

var vBuffer, cBuffer;
```

robotArm.js (4/11)

```
function quad(  a,  b,  c,  d ) {
    colors.push(vertexColors[a]);
    points.push(vertices[a]);
    colors.push(vertexColors[a]);
    points.push(vertices[b]);
    colors.push(vertexColors[a]);
    points.push(vertices[c]);
    colors.push(vertexColors[a]);
    points.push(vertices[a]);
    colors.push(vertexColors[a]);
    points.push(vertices[c]);
    colors.push(vertexColors[a]);
    points.push(vertices[d]);
}
```

```
function colorCube() {
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```




```
// Remmove when scale in MV.js supports scale matrices

function scale4(a, b, c) {
   var result = mat4();
   result[0][0] = a;
   result[1][1] = b;
   result[2][2] = c;
   return result;
}
```

robotArm.js (5/11)

```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );

    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
    gl.enable( gl.DEPTH_TEST );

    //
    //  Load shaders and initialize attribute buffers
    //
    program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
    colorCube();
```

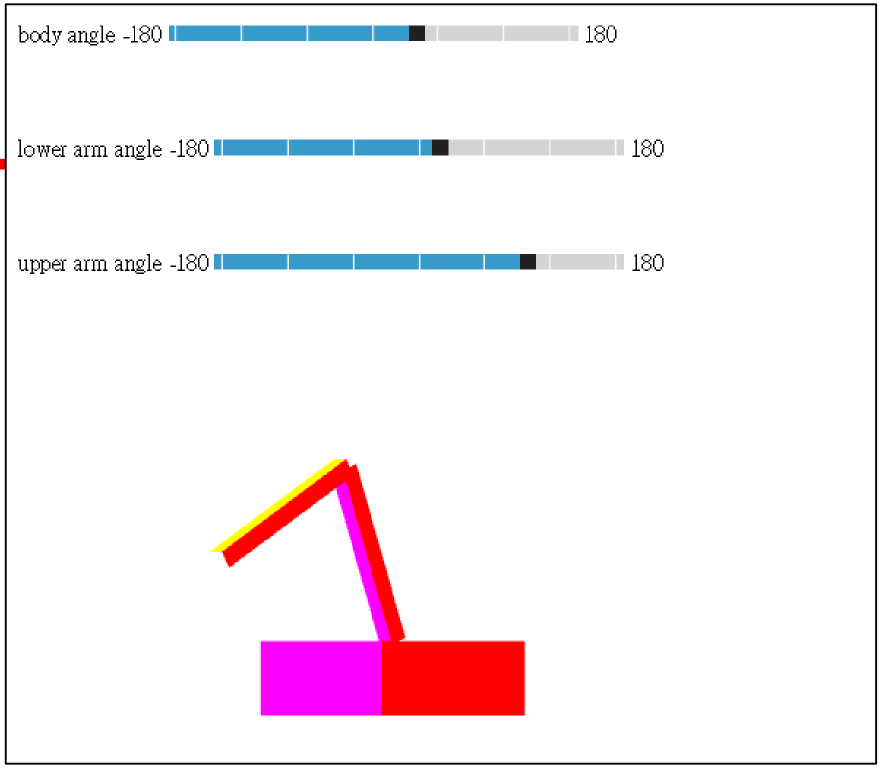
robotArm.js (6/11)

```
// Load shaders and use the resulting shader program

program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );
```

robotArm.js (7/11)

```
// Create and initialize  buffer objects

vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
```

robotArm.js (8/11)




```
    document.getElementById("slider1").onchange = function() {
        theta[0] = event.srcElement.value;
    };
    document.getElementById("slider2").onchange = function() {
         theta[1] = event.srcElement.value;
    };
    document.getElementById("slider3").onchange = function() {
         theta[2] =  event.srcElement.value;
    };

    modelViewMatrixLoc = gl.getUniformLocation(program, "modelViewMatrix");

    projectionMatrix = ortho(-10, 10, -10, 10, -10, 10);
    gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"),  false, flatten(projectionMatrix) );

    render();
}
```

robotArm.js (9/11)

```
function base() {
    var s = scale4(BASE_WIDTH, BASE_HEIGHT, BASE_WIDTH);
    var instanceMatrix = mult( translate( 0.0, 0.5 * BASE_HEIGHT, 0.0 ), s);
    var t = mult(modelViewMatrix, instanceMatrix);
    gl.uniformMatrix4fv(modelViewMatrixLoc,  false, flatten(t) );
    gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
}

function upperArm() {
    var s = scale4(UPPER_ARM_WIDTH, UPPER_ARM_HEIGHT, UPPER_ARM_WIDTH);
    var instanceMatrix = mult(translate( 0.0, 0.5 * UPPER_ARM_HEIGHT, 0.0 ),s);
    var t = mult(modelViewMatrix, instanceMatrix);
    gl.uniformMatrix4fv( modelViewMatrixLoc,  false, flatten(t) );
    gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
}
```

robotArm.js (10/11)

```
function lowerArm()
{
    var s = scale4(LOWER_ARM_WIDTH, LOWER_ARM_HEIGHT, LOWER_ARM_WIDTH);
    var instanceMatrix = mult( translate( 0.0, 0.5 * LOWER_ARM_HEIGHT, 0.0 ), s);
    var t = mult(modelViewMatrix, instanceMatrix);
    gl.uniformMatrix4fv( modelViewMatrixLoc,  false, flatten(t) );
    gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
}
```

robotArm.js (11/11)

```
var render = function() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );

    modelViewMatrix = rotate(theta[Base], 0, 1, 0 );
    base();

    modelViewMatrix = mult(modelViewMatrix, translate(0.0, BASE_HEIGHT, 0.0));
    modelViewMatrix = mult(modelViewMatrix, rotate(theta[LowerArm], 0, 0, 1 ));
    lowerArm();

    modelViewMatrix  = mult(modelViewMatrix, translate(0.0, LOWER_ARM_HEIGHT, 0.0));
    modelViewMatrix  = mult(modelViewMatrix, rotate(theta[UpperArm], 0, 0, 1) );
    upperArm();

    requestAnimFrame(render);
}
```

Sample Programs: figure.html, figure.js

figure.html (1/6)

```html
<html>

<script id="vertex-shader" type="x-shader/x-vertex">
attribute  vec4 vPosition;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main()
{  gl_Position = projectionMatrix * modelViewMatrix * vPosition; }

</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

void main()
{  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0); }

</script>
```

figure.html (2/6)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/InitShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="figure.js"></script>

<div id="slider0">
torso angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
<div id="slider10">
head2 angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```

figure.html (3/6)

```
<div id="slider1">
head1 angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>

<div id="slider2">
left upper arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```

figure.html (4/6)

```
<div id="slider3">
left lower arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
<div id="slider4">
right upper arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
<div id="slider5">
right lower arm angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```

figure.html (5/6)

```
<div id="slider6">
left upper leg angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
<div id="slider7">
left lower leg angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
<div id="slider8">
right upper leg angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>
```

figure.html (6/6)

```
<div id="slider9">
right lower leg angle -180 <input id="slide" type="range"
 min="-180" max="180" step="10" value="0"
  />
 180
</div><br/>


<body>
<canvas id="gl-canvas" width="512"" height="512"
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```

figure.js (1/25)

```
var canvas;
var gl;
var program;
var projectionMatrix;
var modelViewMatrix;
var instanceMatrix;
var modelViewMatrixLoc;

var vertices = [
    vec4( -0.5, -0.5,  0.5, 1.0 ),
    vec4( -0.5,  0.5,  0.5, 1.0 ),
    vec4( 0.5,  0.5,  0.5, 1.0 ),
    vec4( 0.5, -0.5,  0.5, 1.0 ),
    vec4( -0.5, -0.5, -0.5, 1.0 ),
    vec4( -0.5,  0.5, -0.5, 1.0 ),
    vec4( 0.5,  0.5, -0.5, 1.0 ),
    vec4( 0.5, -0.5, -0.5, 1.0 )
];
```

figure.js (2/25)

```
var torsoId = 0;
var headId  = 1;
var head1Id = 1;
var head2Id = 10;
var leftUpperArmId = 2;
var leftLowerArmId = 3;
var rightUpperArmId = 4;
var rightLowerArmId = 5;
var leftUpperLegId = 6;
var leftLowerLegId = 7;
var rightUpperLegId = 8;
var rightLowerLegId = 9;
```

figure.js (3/25)

```
var torsoHeight = 5.0;
var torsoWidth = 1.0;
var upperArmHeight = 3.0;
var lowerArmHeight = 2.0;
var upperArmWidth  = 0.5;
var lowerArmWidth  = 0.5;
var upperLegWidth  = 0.5;
var lowerLegWidth  = 0.5;
var lowerLegHeight = 2.0;
var upperLegHeight = 3.0;
var headHeight = 1.5;
var headWidth = 1.0;
```

figure.js (4/25)

```
var numNodes = 10;
var numAngles = 11;
var angle = 0;

var theta = [0, 0, 0, 0, 0, 0, 180, 0, 180, 0, 0];

var numVertices = 24;

var stack = [];

var figure = [];

for( var i=0; i<numNodes; i++) figure[i] = createNode(null, null, null, null);

var vBuffer;
var modelViewLoc;

var pointsArray = [];
```

figure.js (5/25)

```
function scale4(a, b, c) {
   var result = mat4();
   result[0][0] = a;
   result[1][1] = b;
   result[2][2] = c;
   return result;
}

function createNode(transform, render, sibling, child) {
    var node = {
    transform: transform,
    render: render,
    sibling: sibling,
    child: child,
    }
    return node;
}
```

figure.js (6/25)

```
function initNodes(Id) {

    var m = mat4();

    switch(Id) {

    case torsoId:

        m = rotate(theta[torsoId], 0, 1, 0 );
        figure[torsoId] = createNode( m, torso, null, headId );
        break;
```

figure.js (7/25)

```
case headId:
case head1Id:
case head2Id:

    m = translate(0.0, torsoHeight+0.5*headHeight, 0.0);
    m = mult(m, rotate(theta[head1Id], 1, 0, 0))
    m = mult(m, rotate(theta[head2Id], 0, 1, 0));
    m = mult(m, translate(0.0, -0.5*headHeight, 0.0));
    figure[headId] = createNode( m, head, leftUpperArmId, null);
    break;

case leftUpperArmId:

    m = translate(-(torsoWidth+upperArmWidth), 0.9*torsoHeight, 0.0);
    m = mult(m, rotate(theta[leftUpperArmId], 1, 0, 0));
    figure[leftUpperArmId] = createNode( m, leftUpperArm, rightUpperArmId, leftLowerArmId );
    break;
```

figure.js (8/25)

```
case rightUpperArmId:

    m = translate(torsoWidth+upperArmWidth, 0.9*torsoHeight, 0.0);
    m = mult(m, rotate(theta[rightUpperArmId], 1, 0, 0));
    figure[rightUpperArmId] = createNode( m, rightUpperArm, leftUpperLegId, rightLowerArmId );
    break;

case leftUpperLegId:

    m = translate(-(torsoWidth+upperLegWidth), 0.1*upperLegHeight, 0.0);
    m = mult(m , rotate(theta[leftUpperLegId], 1, 0, 0));
    figure[leftUpperLegId] = createNode( m, leftUpperLeg, rightUpperLegId, leftLowerLegId );
    break;
```

figure.js (9/25)

```
case rightUpperLegId:

    m = translate(torsoWidth+upperLegWidth, 0.1*upperLegHeight, 0.0);
    m = mult(m, rotate(theta[rightUpperLegId], 1, 0, 0));
    figure[rightUpperLegId] = createNode( m, rightUpperLeg, null, rightLowerLegId );
    break;

case leftLowerArmId:

    m = translate(0.0, upperArmHeight, 0.0);
    m = mult(m, rotate(theta[leftLowerArmId], 1, 0, 0));
    figure[leftLowerArmId] = createNode( m, leftLowerArm, null, null );
    break;
```

figure.js (10/25)

```
case rightLowerArmId:

    m = translate(0.0, upperArmHeight, 0.0);
    m = mult(m, rotate(theta[rightLowerArmId], 1, 0, 0));
    figure[rightLowerArmId] = createNode( m, rightLowerArm, null, null );
    break;

case leftLowerLegId:

    m = translate(0.0, upperLegHeight, 0.0);
    m = mult(m, rotate(theta[leftLowerLegId], 1, 0, 0));
    figure[leftLowerLegId] = createNode( m, leftLowerLeg, null, null );
    break;
```

figure.js (11/25)

```
case rightLowerLegId:

    m = translate(0.0, upperLegHeight, 0.0);
    m = mult(m, rotate(theta[rightLowerLegId], 1, 0, 0));
    figure[rightLowerLegId] = createNode( m, rightLowerLeg, null, null );
    break;

    }

}     // end of function initNodes(Id)
```

figure.js (12/25)

```
function traverse(Id) {

   if(Id == null) return;
   stack.push(modelViewMatrix);
   modelViewMatrix = mult(modelViewMatrix, figure[Id].transform);
   figure[Id].render();
   if(figure[Id].child != null) traverse(figure[Id].child);
   modelViewMatrix = stack.pop();
   if(figure[Id].sibling != null) traverse(figure[Id].sibling);
}

function torso() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5*torsoHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4( torsoWidth, torsoHeight, torsoWidth));
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}
```

figure.js (13/25)

```
function head() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * headHeight, 0.0 ));
    instanceMatrix = mult(instanceMatrix, scale4(headWidth, headHeight, headWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}

function leftUpperArm() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * upperArmHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(upperArmWidth, upperArmHeight, upperArmWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}
```

figure.js (14/25)

```
function leftLowerArm() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * lowerArmHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(lowerArmWidth, lowerArmHeight, lowerArmWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}

function rightUpperArm() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * upperArmHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(upperArmWidth, upperArmHeight, upperArmWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}
```

figure.js (15/25)

```
function rightLowerArm() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * lowerArmHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(lowerArmWidth, lowerArmHeight, lowerArmWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}

function  leftUpperLeg() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * upperLegHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(upperLegWidth, upperLegHeight, upperLegWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}
```

figure.js (16/25)

```
function leftLowerLeg() {

    instanceMatrix = mult(modelViewMatrix, translate( 0.0, 0.5 * lowerLegHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(lowerLegWidth, lowerLegHeight, lowerLegWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}

function rightUpperLeg() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * upperLegHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(upperLegWidth, upperLegHeight, upperLegWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}
```

figure.js (17/25)

```
function rightLowerLeg() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * lowerLegHeight, 0.0) );
    instanceMatrix = mult(instanceMatrix, scale4(lowerLegWidth, lowerLegHeight, lowerLegWidth) )
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4*i, 4);
}

function quad(a, b, c, d) {
     pointsArray.push(vertices[a]);
     pointsArray.push(vertices[b]);
     pointsArray.push(vertices[c]);
     pointsArray.push(vertices[d]);
}
```

figure.js (18/25)

```
function cube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```

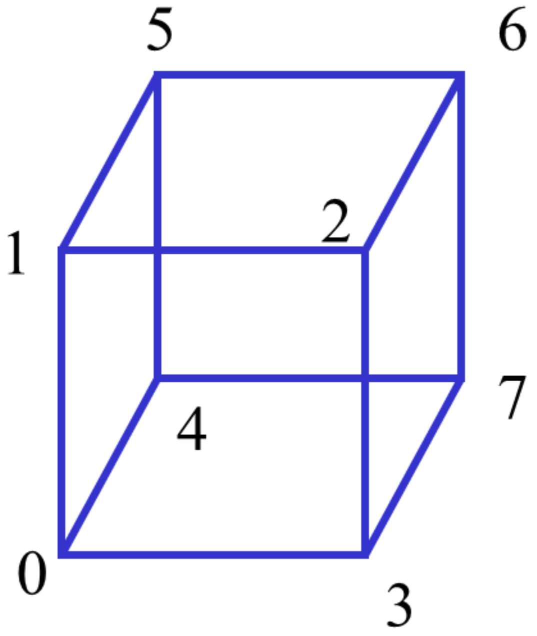


figure.js (19/25)

```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```

figure.js (20/25)

```
//  Load shaders and initialize attribute buffers

program = initShaders( gl, "vertex-shader", "fragment-shader");

gl.useProgram( program);

instanceMatrix = mat4();

projectionMatrix = ortho(-10.0,10.0,-10.0, 10.0,-10.0,10.0);
modelViewMatrix = mat4();


gl.uniformMatrix4fv(gl.getUniformLocation( program, "modelViewMatrix"), false, flatten(modelViewMatrix) );
gl.uniformMatrix4fv( gl.getUniformLocation( program, "projectionMatrix"), false, flatten(projectionMatrix) );

modelViewMatrixLoc = gl.getUniformLocation(program, "modelViewMatrix")

cube();
```

figure.js (21/25)

```
vBuffer = gl.createBuffer();

gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData(gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

figure.js (22/25)

```
document.getElementById("slider0").onchange = function() {
        theta[torsoId ] = event.srcElement.value;
        initNodes(torsoId);
    };
document.getElementById("slider1").onchange = function() {
        theta[head1Id] = event.srcElement.value;
        initNodes(head1Id);
    };
document.getElementById("slider2").onchange = function() {
         theta[leftUpperArmId] = event.srcElement.value;
         initNodes(leftUpperArmId);
    };
document.getElementById("slider3").onchange = function() {
         theta[leftLowerArmId] =  event.srcElement.value;
         initNodes(leftLowerArmId);
    };
```

figure.js (23/25)

```
document.getElementById("slider4").onchange = function() {
        theta[rightUpperArmId] = event.srcElement.value;
        initNodes(rightUpperArmId);
    };
document.getElementById("slider5").onchange = function() {
         theta[rightLowerArmId] =  event.srcElement.value;
         initNodes(rightLowerArmId);
    };
document.getElementById("slider6").onchange = function() {
        theta[leftUpperLegId] = event.srcElement.value;
        initNodes(leftUpperLegId);
    };
document.getElementById("slider7").onchange = function() {
         theta[leftLowerLegId] = event.srcElement.value;
         initNodes(leftLowerLegId);
    };
```

figure.js (24/25)

```
document.getElementById("slider8").onchange = function() {
        theta[rightUpperLegId] =  event.srcElement.value;
        initNodes(rightUpperLegId);
    };
document.getElementById("slider9").onchange = function() {
        theta[rightLowerLegId] = event.srcElement.value;
        initNodes(rightLowerLegId);
    };
document.getElementById("slider10").onchange = function() {
        theta[head2Id] = event.srcElement.value;
        initNodes(head2Id);
    };

for(i=0; i<numNodes; i++) initNodes(i);

render();

}  // end of window.onload
```

figure.js (25/25)

```
var render = function() {

        gl.clear( gl.COLOR_BUFFER_BIT );
        traverse(torsoId);
        requestAnimFrame(render);
}
```