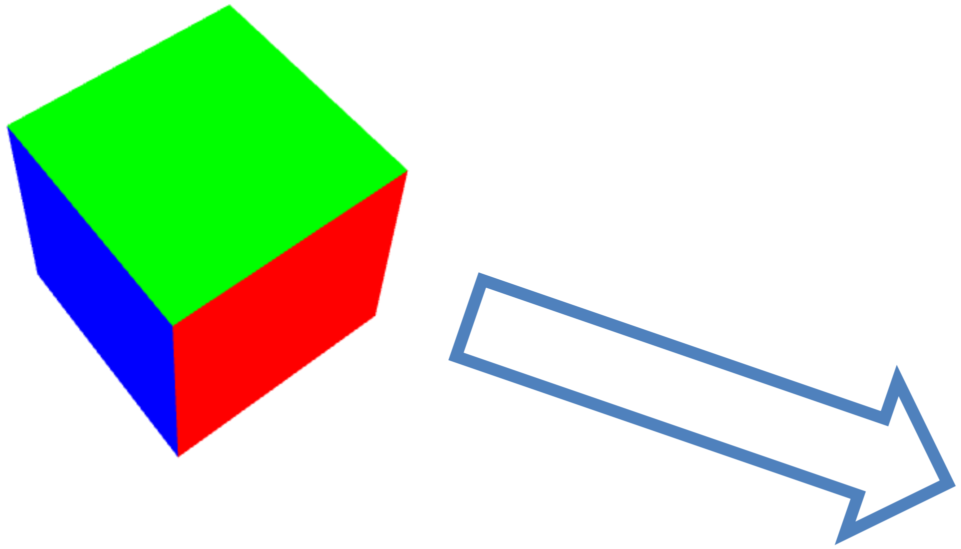
POLITECNICO
MILANO 1863

Scene Graphs

Computer Graphics 2021

Erica Stella (erica.stella@polimi.it)

Drawing Multiple Objects



Drawing Multiple Objects

Simplest case

- Render the **same shape** in different locations
- Use the **same vertex/fragment shaders**

For each instance of the object:

- Set the world (model) matrix to place the object in the right position
- Call `gl.drawElements/gl.drawArrays` (with the right number of indices or vertices)

Drawing Multiple Objects

<u>A Bit More Complex</u>

- **Different shapes** in different locations
- Use the **same vertex/fragment shaders**

For each object:

- Initialise and populate a <u>Vertex Array Object</u> with the right attributes
  - 1 VAO for each object!!!
- Set the world (model) matrix to place the object in the right position
- Call `gl.drawElements`/`gl.drawArrays` (with the right number of indices or vertices)

Drawing Multiple Objects

General Case

- **Different shapes** in different locations
- **Different vertex/fragment shaders**

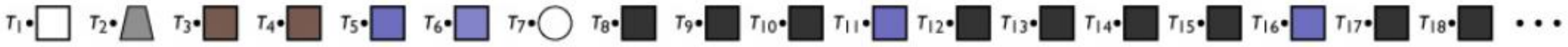
Create one GLSL program for each vertex+fragment shader we want to use
Store uniform and attributes locations for each program

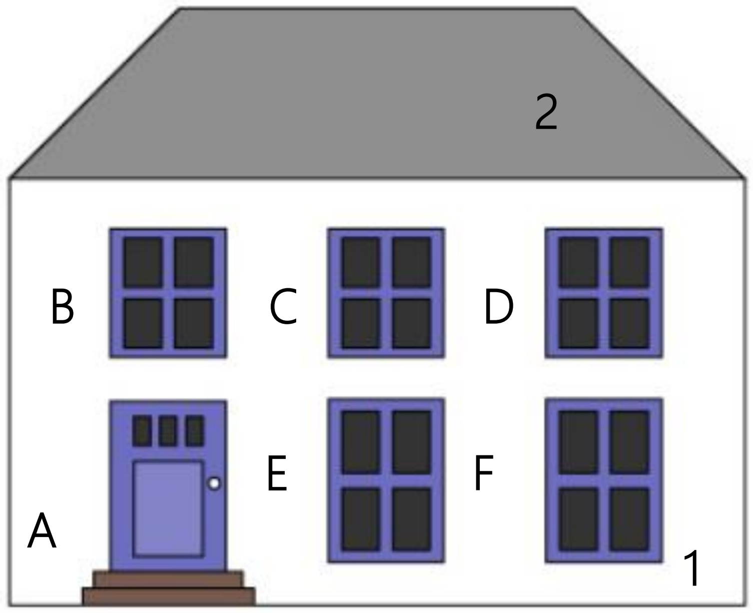
For each object:

- Initialise and populate a Vertex Array Object with the right attributes
- Set the world (model) matrix to place the object in the right position
- Update the uniforms needed to draw that thing with the given shader.
- Call `gl.drawElements`/`gl.drawArrays` (right number of indices/vertices!)

A big issue when Drawing Multiple Objects

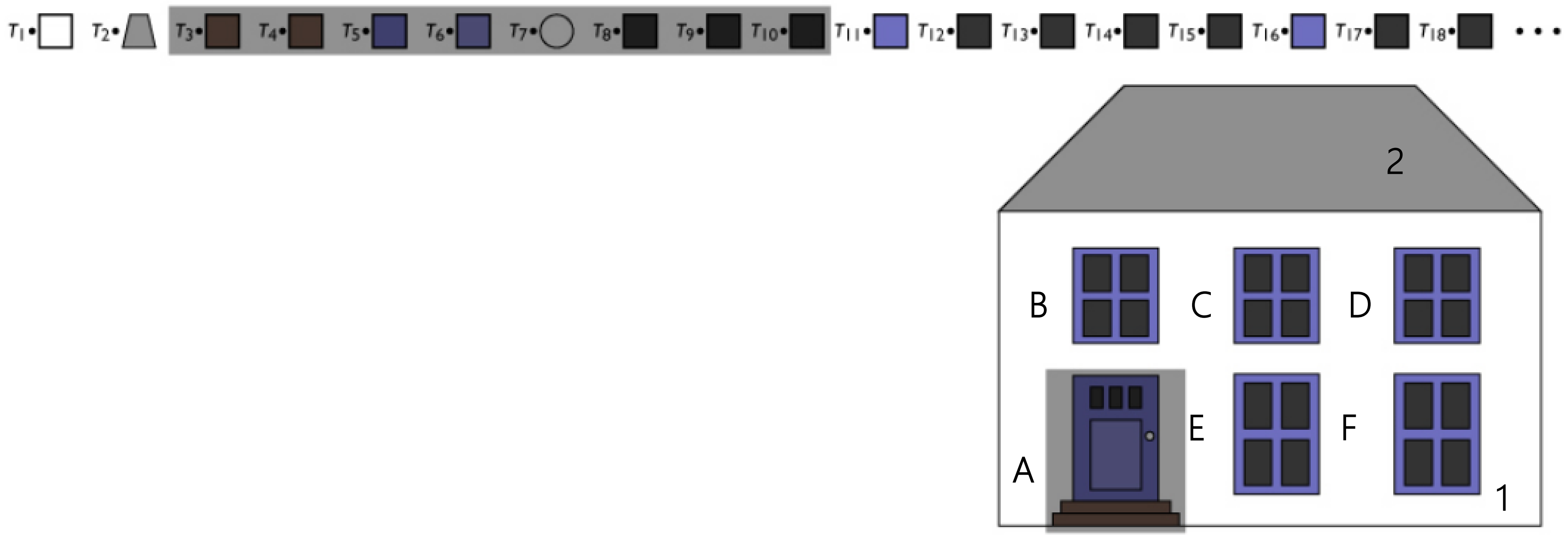
- If you need to draw multiple objects, you also need to update the world (model) matrix for each object
- What if the position of object B depends on the position of object A?





A big issue when Drawing Multiple Objects

- If you need to draw multiple objects, you also need to update the world (model) matrix for each object
- What if the position of object B depends on the position of object A?

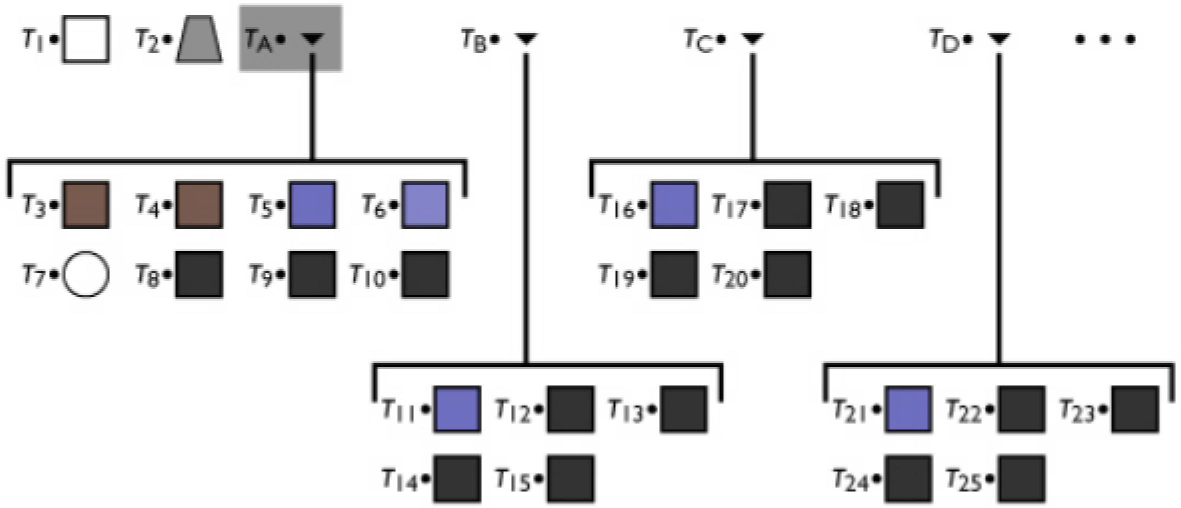


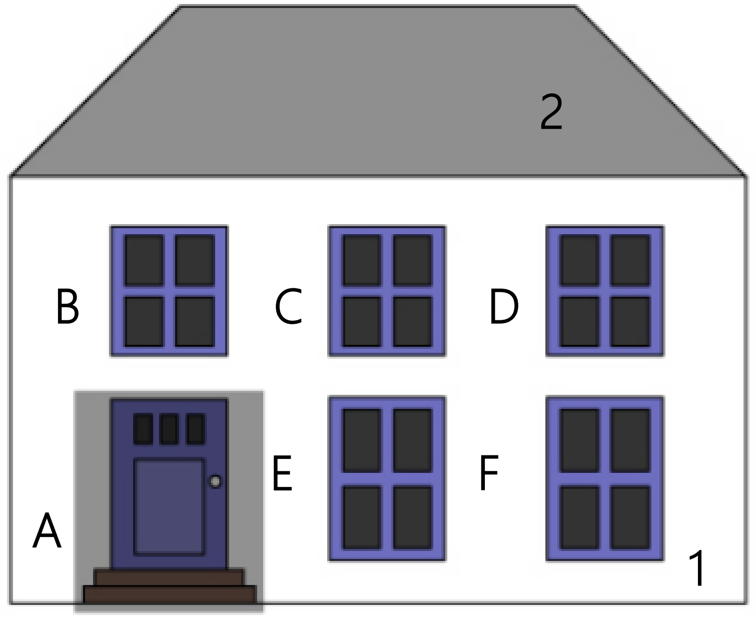
Groups of objects

- Treat a set of objects as one
- Introduce new structure type: “group” or “node”
  - Contains references to other nodes or objects (meshes)
- This makes the scene into a tree
  - Interior nodes = groups or nodes (entrance)
  - Leaf nodes = objects (stairs + handle + door)
  - Edges = membership of objects/nodes in groups

Groups of objects

- Add group as a new object type
  - let the data structure reflect the drawing structure
  - enables high-level editing by changing just one node



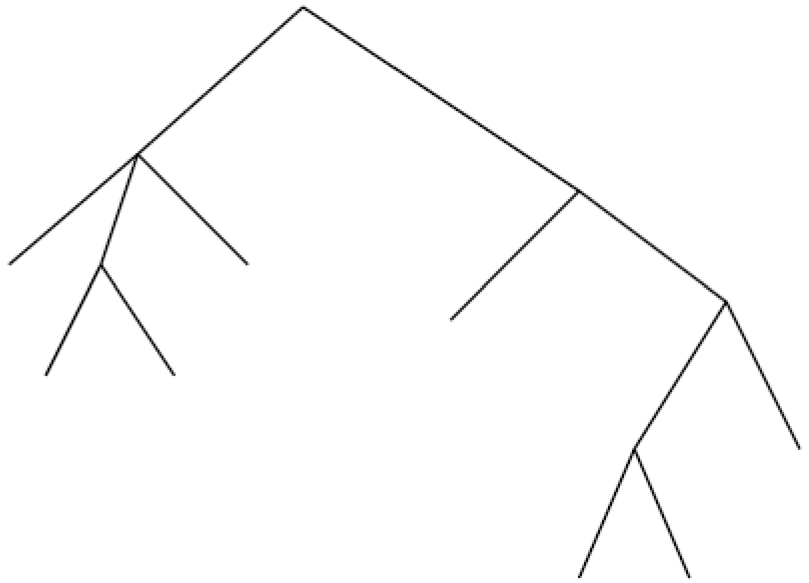


Scene graph: Simplest form

Tree:

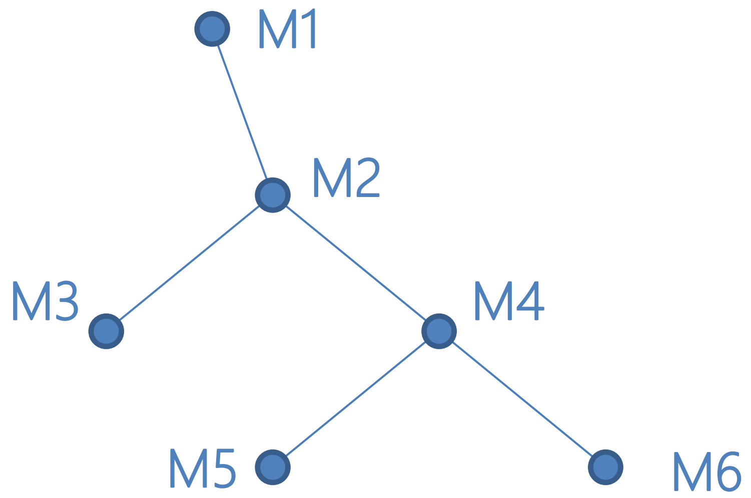
– Every node has one parent

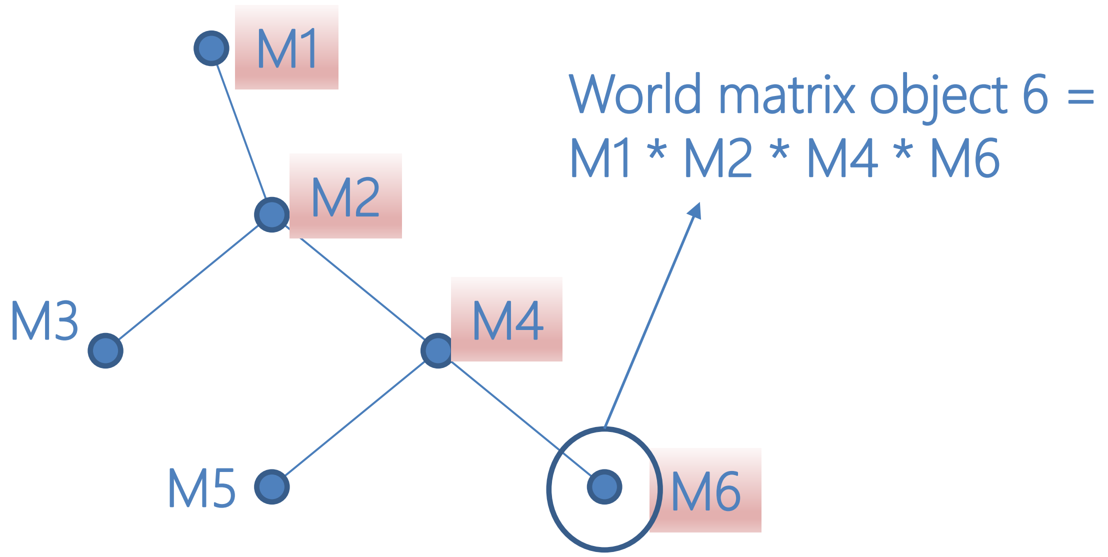
– Leaf nodes are identified with objects in the scene



Scene Graphs and transformations

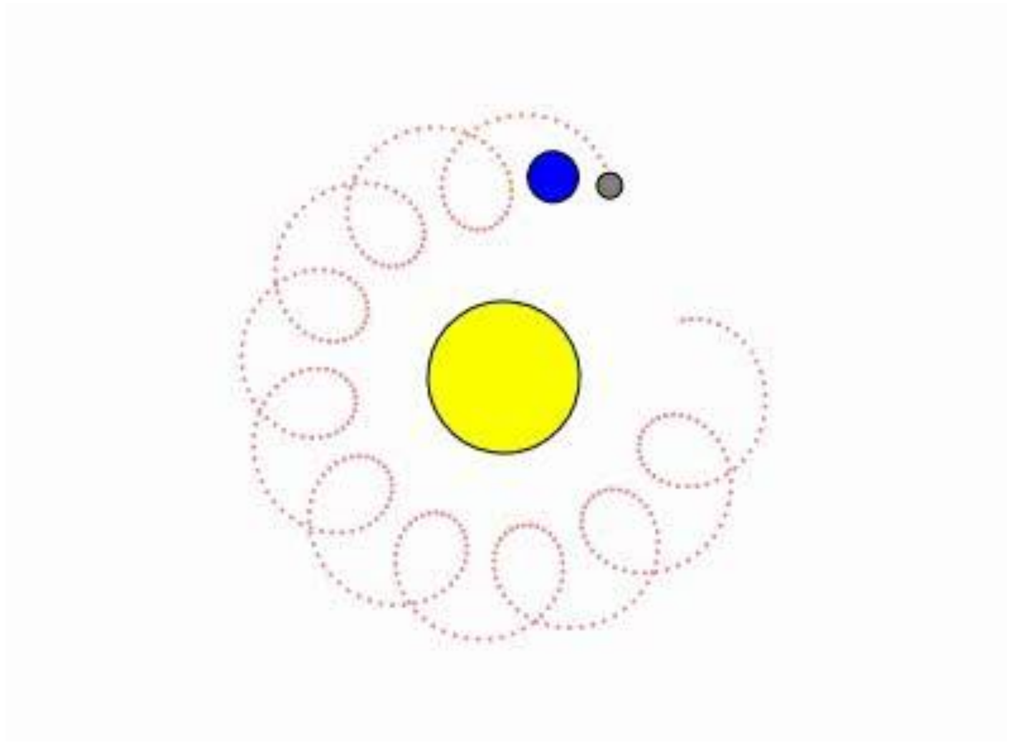
- Transformation matrices are associated with nodes or edges
  - Each transformation applies to all geometry below the node
- The Object transformation is the product of all matrices along path from root:
  - Each object transform describes relationship between its local coordinates and its group coordinates





Exercise SEM: The Sun, The Earth and The Moon

- Render three spheres:
  - A big yellow sphere at the center of the screen spinning around itself
  - A small blue sphere that spins around the yellow one and spins around itself
  - A small gray sphere that spins around the blue one and spins around itself



Let’s Create the Scene Graph

We define a data structure for each Node to store:

- The pointer to the children
- localMatrix: the matrix that transforms the node and the children
- worldMatrix: the matrix that transforms the node and the children form local space to world space

```
var Node = function() {
  this.children = [];
  this.localMatrix = utils.identityMatrix();
  this.worldMatrix = utils.identityMatrix();
};
```

Let’s Create the Scene Graph

SetParent function to define the hierarchy

```
Node.prototype.setParent = function(parent) {
  // remove us from our parent
  if (this.parent) {
    var ndx = this.parent.children.indexOf(this);
    if (ndx >= 0) {
  // remove elem ndx(current node) from our parent
      this.parent.children.splice(ndx, 1);
    }
  }

  // Add us to our new parent
  if (parent) {
    parent.children.push(this);
  }
  this.parent = parent;
};
```

Compute the world matrix for each leaf

Recursive function to compute world matrices from local matrices based on their parent-child relationships

```
Node.prototype.updateWorldMatrix = function(matrix) {
  if (matrix) {
    this.worldMatrix = utils.multiplyMatrices(matrix, this.localMatrix);
  } else {
    // no matrix was passed in so just copy localMatrix->worldMatrix.
    utils.copy(this.localMatrix, this.worldMatrix);
  }

  // now process all the children
  var worldMatrix = this.worldMatrix;
  this.children.forEach(function(child) {
    child.updateWorldMatrix(worldMatrix);
  });
};
```

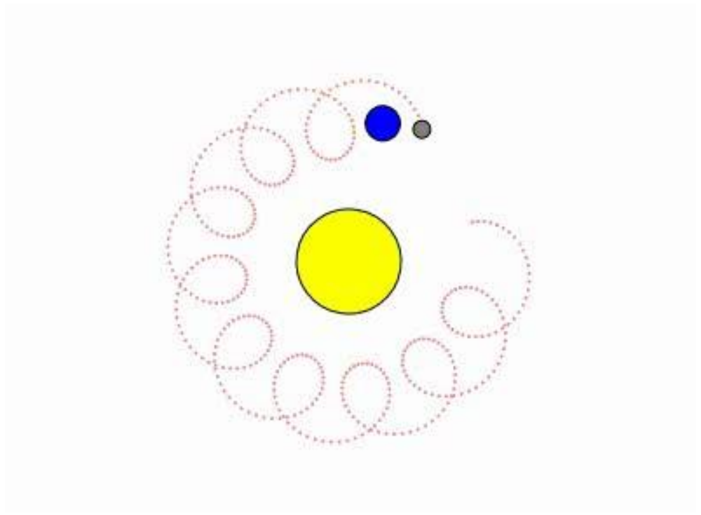
Solution 1

A first attemp is to draw each sphere individually.

Issues:

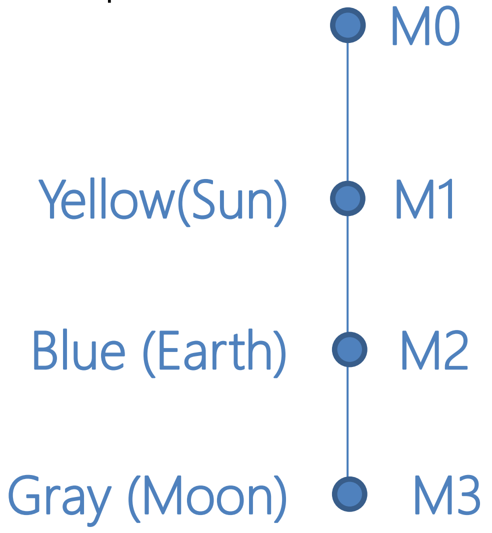
- Need to define the world matrices for each object such that they move synchronously
- What if I want to change the direction of the spinning of just one sphere

Conclusion: Not such a great idea!



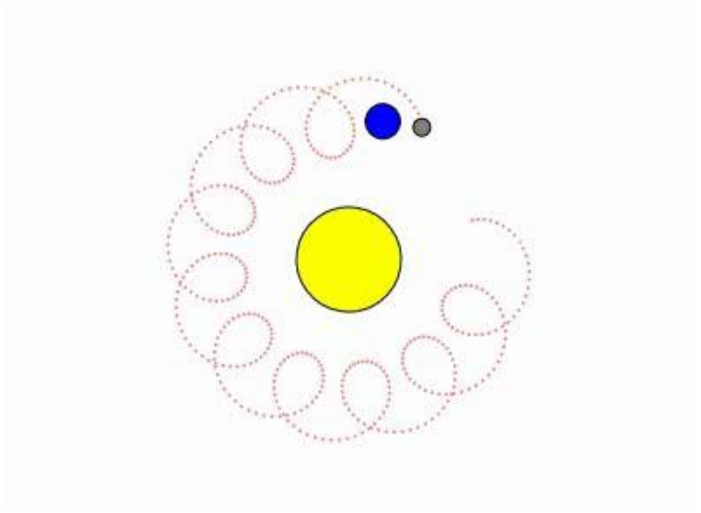
Solution 2

Create a simple Scene Graph

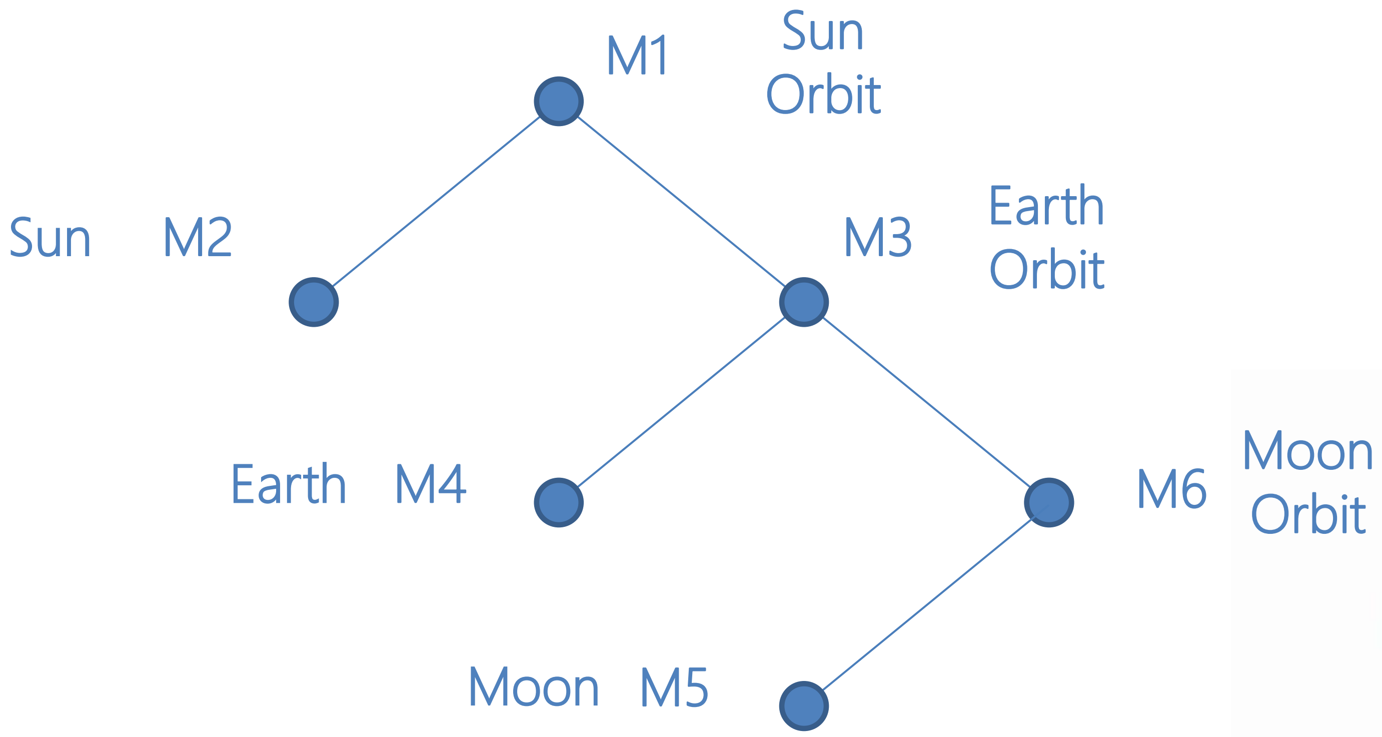


Better than before but… what if:

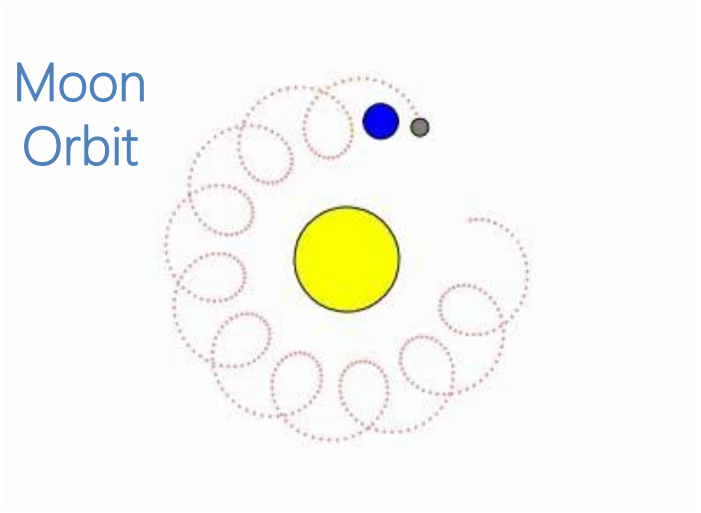
- I want to use a single sphere model
  and apply a scaling matrix to each sphere?
- I want different spinning velocities



A better Way



M1
Sun Orbit
Sun
M2
M3
Earth Orbit
Earth
M4
M6
Moon Orbit
Moon
M5



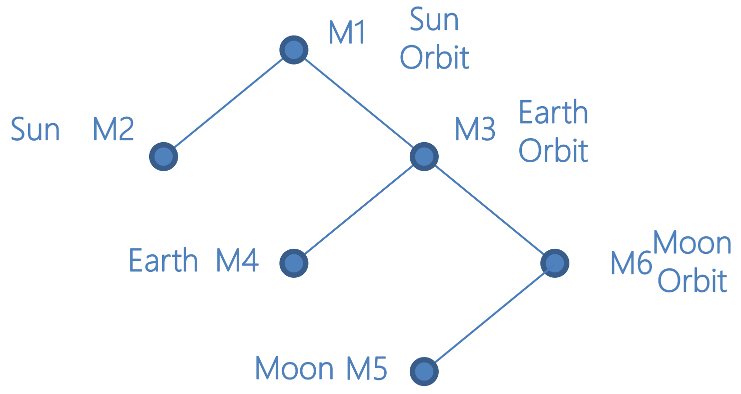
Scene Graph Definition

M1 → Identity

M3 → translation by 100 units on x

M6 → translation by 30 units on x

```
var sunOrbitNode = new Node();


var earthOrbitNode = new Node();
earthOrbitNode.localMatrix = utils.MakeTranslateMatrix(100, 0, 0);


var moonOrbitNode = new Node();
moonOrbitNode.localMatrix = utils.MakeTranslateMatrix(30, 0, 0);
```

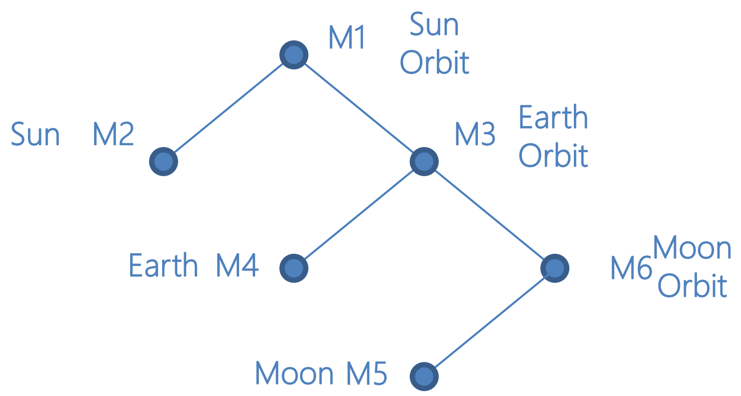


Scene Graph Definition

M2 → Scaling 5x

M4 → Scaling 2x

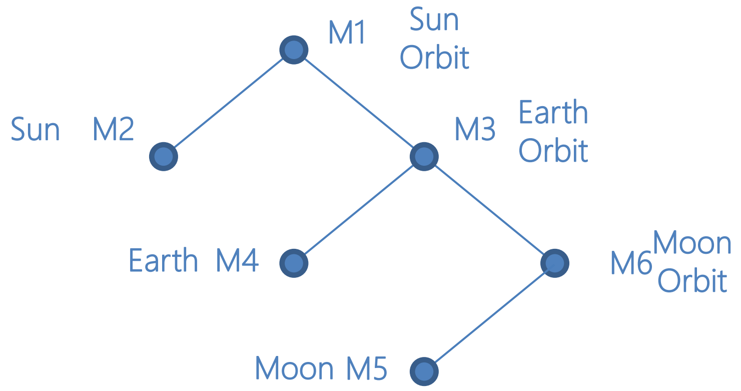
```
var sunNode = new Node();
sunNode.localMatrix = utils.MakeScaleMatrix(5, 5, 5);
sunNode.drawInfo = {
  materialColor: [0.6, 0.6, 0.0],
  programInfo: program,
  bufferLength: indexData.length,
  vertexArray: vao,
};

var earthNode = new Node();
earthNode.localMatrix = utils.MakeScaleMatrix(2, 2, 2);
earthNode.drawInfo = {
  materialColor: [0.2, 0.5, 0.8],
  programInfo: program,
  bufferLength: indexData.length,
  vertexArray: vao,
};
```



Scene Graph Definition

M5 → Scaling 0.7x

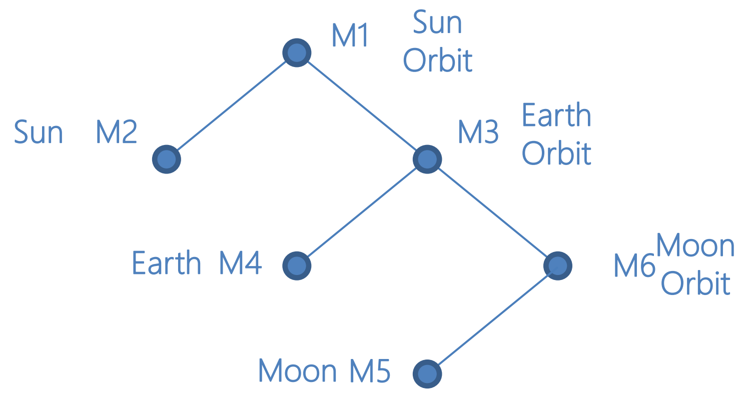
```
var moonNode = new Node();
moonNode.localMatrix = utils.MakeScaleMatrix(0.7, 0.7, 0.7);
moonNode.drawInfo = {
  materialColor: [0.6, 0.6, 0.6],
  programInfo: program,
  bufferLength: indexData.length,
  vertexArray: vao,
};
```



Scene Graph Definition

Set the parenthood relationships

```
  sunNode.setParent(sunOrbitNode);

  earthOrbitNode.setParent(sunOrbitNode);

  earthNode.setParent(earthOrbitNode);

  moonOrbitNode.setParent(earthOrbitNode);

  moonNode.setParent(moonOrbitNode);

// define an array of objects to be rendered
var objects = [
    sunNode,
    earthNode,
    moonNode,
  ];
```



Scene Graph Rendering

Update the local matrices independently

```
earthOrbitNode.localMatrix =
     utils.multiplyMatrices(utils.MakeRotateYMatrix(0.1), earthOrbitNode.localMatrix);

moonOrbitNode.localMatrix =
     utils.multiplyMatrices(utils.MakeRotateYMatrix(0.1), moonOrbitNode.localMatrix);

sunNode.localMatrix =
     utils.multiplyMatrices(utils.MakeRotateYMatrix(0.05), sunNode.localMatrix);

earthNode.localMatrix =
     utils.multiplyMatrices(utils.MakeRotateYMatrix(0.5), earthNode.localMatrix);

moonNode.localMatrix =
     utils.multiplyMatrices(utils.MakeRotateYMatrix(-0.1), moonNode.localMatrix);
```

Update all the `worldMatrix` in the scene graph recursively from the root

```
sunOrbitNode.updateWorldMatrix();
```

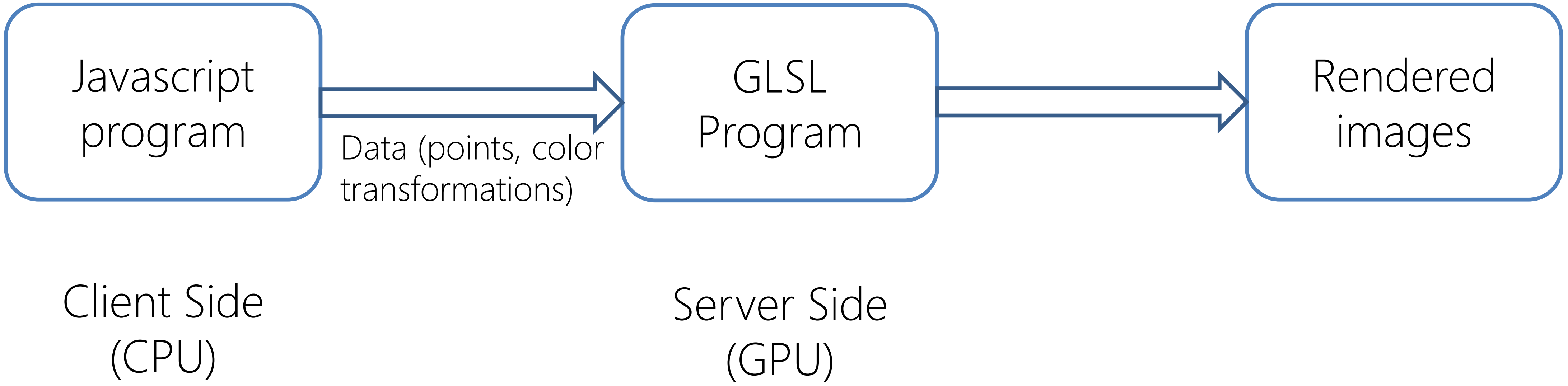
Scene Graph Rendering

Render each object with its own shader

```
objects.forEach(function(object) {
   gl.useProgram(object.drawInfo.programInfo);

   var projectionMatrix = utils.multiplyMatrices(viewProjectionMatrix, object.worldMatrix);
   var normalMatrix = utils.invertMatrix(utils.transposeMatrix(object.worldMatrix));

   gl.uniformMatrix4fv(matrixLocation, gl.FALSE, utils.transposeMatrix(projectionMatrix));
   gl.uniformMatrix4fv(normalMatrixHandle, gl.FALSE, utils.transposeMatrix(normalMatrix));
   gl.uniform3fv(materialDiffColorHandle, object.drawInfo.materialColor);
   gl.uniform3fv(lightColorHandle,  directionalLightColor);
   gl.uniform3fv(lightDirectionHandle,  directionalLight);

   gl.bindVertexArray(object.drawInfo.vertexArray);
   gl.drawElements(gl.TRIANGLES, object.drawInfo.bufferLength, gl.UNSIGNED_SHORT, 0 );
  });
```

Wrap up

WebGL pipeline



WebGL Program (javascript File)

Initialization:

- Load models
- Create **shaders** and programs and **uniform/attributes locations**
- Create **buffers** and upload vertex data
- Create a **vertex array object** (VAO) for each thing you want to draw:
  - for each attribute call **`gl.bindBuffer`**, **`gl.vertexAttribPointer`**, **`gl.enableVertexAttribArray`**
  - bind any indices to **`gl.ELEMENT_ARRAY_BUFFER`**
- Create **textures** and upload texture data

WebGL Program (javascript File)

<u>Rendering:</u>

- Clear/set other global states (viewport color, color buffer, depth testing, culling, …)
- For each thing you want to draw
  - Call **`gl.useProgram`** for the GLSL program needed.
  - Bind the vertex array for that thing with **`gl.bindVertexArray`**
  - Update uniforms:
    - With **`gl.uniformXXX`** for each uniform
    - With **`gl.activeTexture`** and **`gl.bindTexture`** for each texture
  - Call **`gl.drawArrays`** or **`gl.drawElements`**

Shader Program (GLSL File)

Vertex Shader:

- Take all the per-vertex attributes as input (position, normals, uv)
- Define the position of the vertices in the clip space with
  - MVP matrix (Model-View-Perspective or Projection matrix) * $[x,y,z,1]^T$
- (Optionally, for Gouraud Shading) define the vertex lighting
- Outputs the positions, normals (transformed in the right space), uv

Fragment Shader:

- Take the attributes output by the Vertex Shader
- Compute the color of each fragment (lighting + texturing with uv maps)
- Output the fragment colour