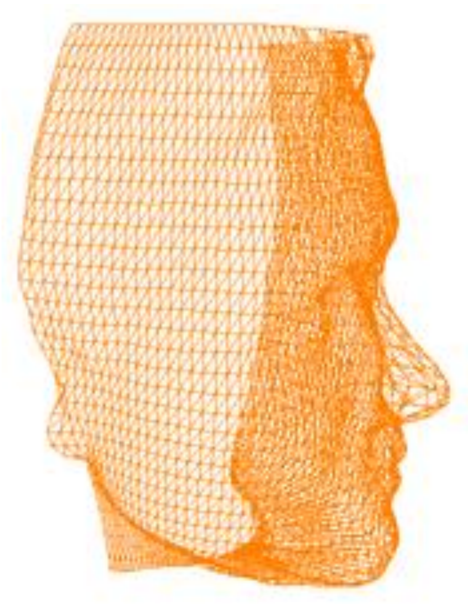
COSC422 Advanced Computer Graphics



9 Scene Graphs

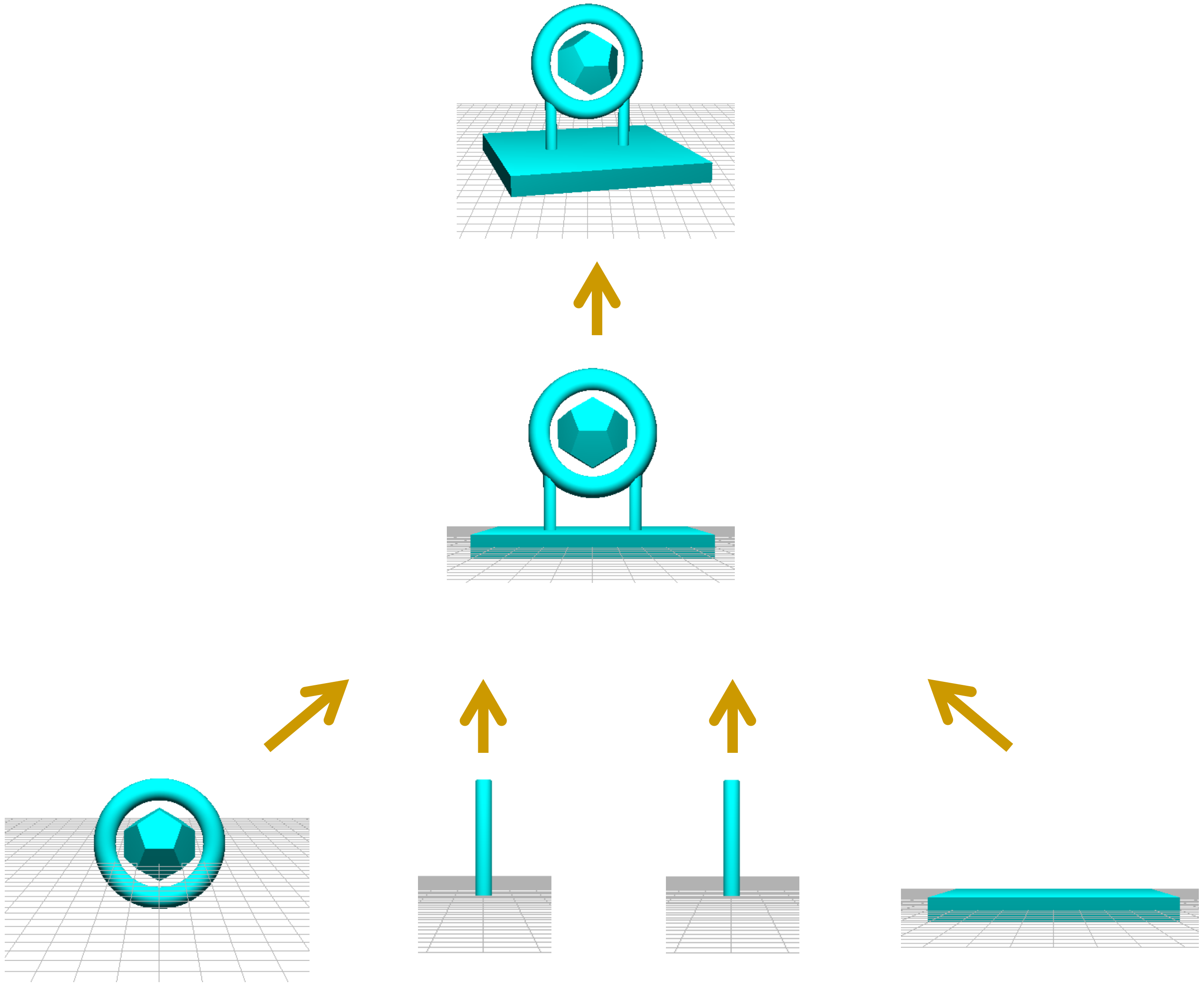
Semester 2
2021

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Model Construction

- A 3D mesh model or scene generally consists of combinations of several meshes that are combined together in the workspace of a content creation tool.
- A model may contain multiple copies of the same mesh, but with different transformations (eg. wheels of a car). A scene file need store only one copy of the mesh (wheel), and its transformations relative to another mesh (car's body)
- We can represent the construction of a composite mesh model, and in general a scene, using a hierarchy of transformations.

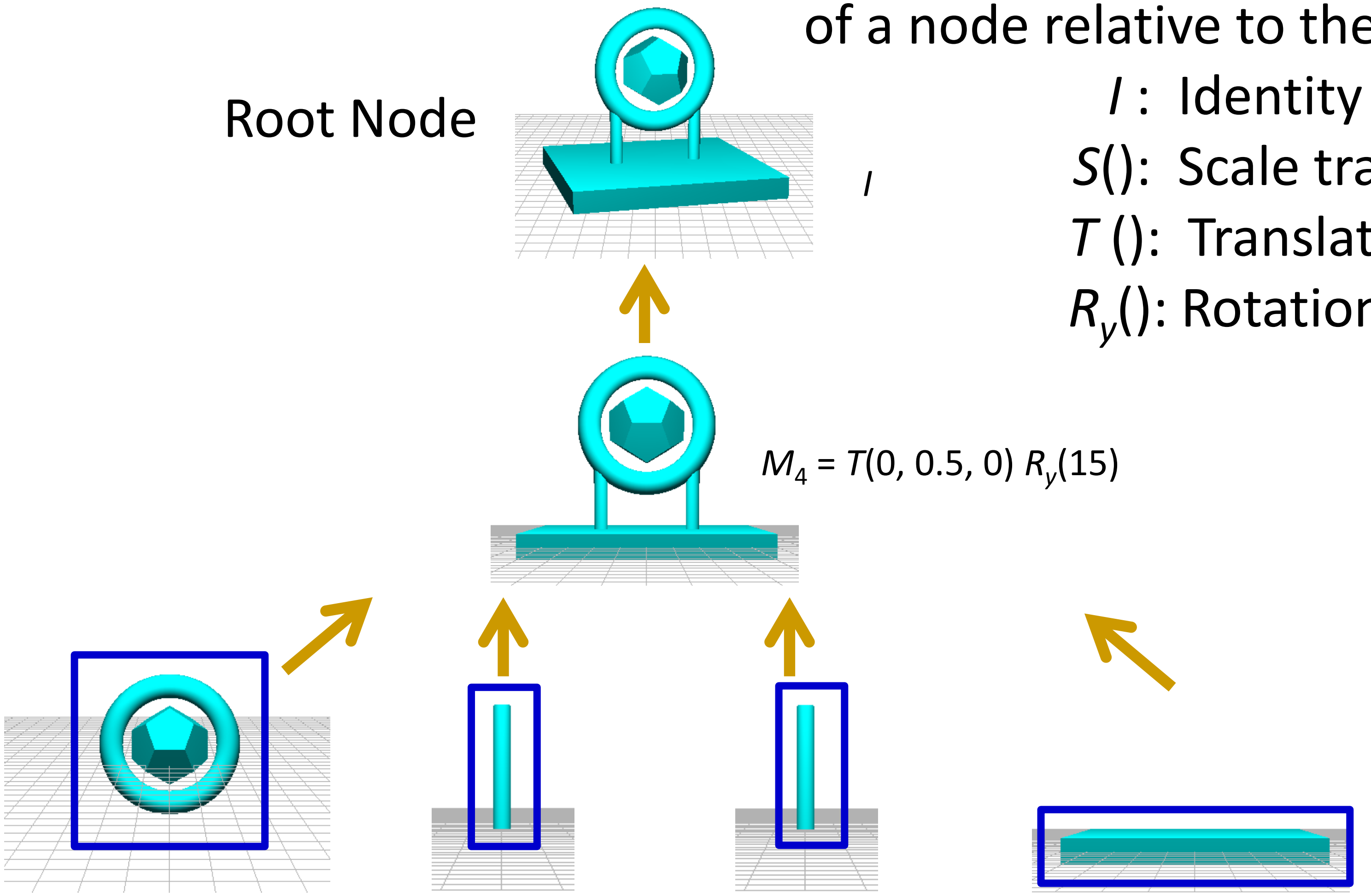
Model Construction Example



Transformation Hierarchy

A matrix represents a transformation of a node relative to the parent node.

I :	Identity matrix
$S()$:	Scale transformation
$T()$:	Translation
$R_y()$:	Rotation about y-axis



Root Node

I

$M_4 = T(0, 0.5, 0)\ R_y(15)$

$M_0 = T(0, 5, 0)$

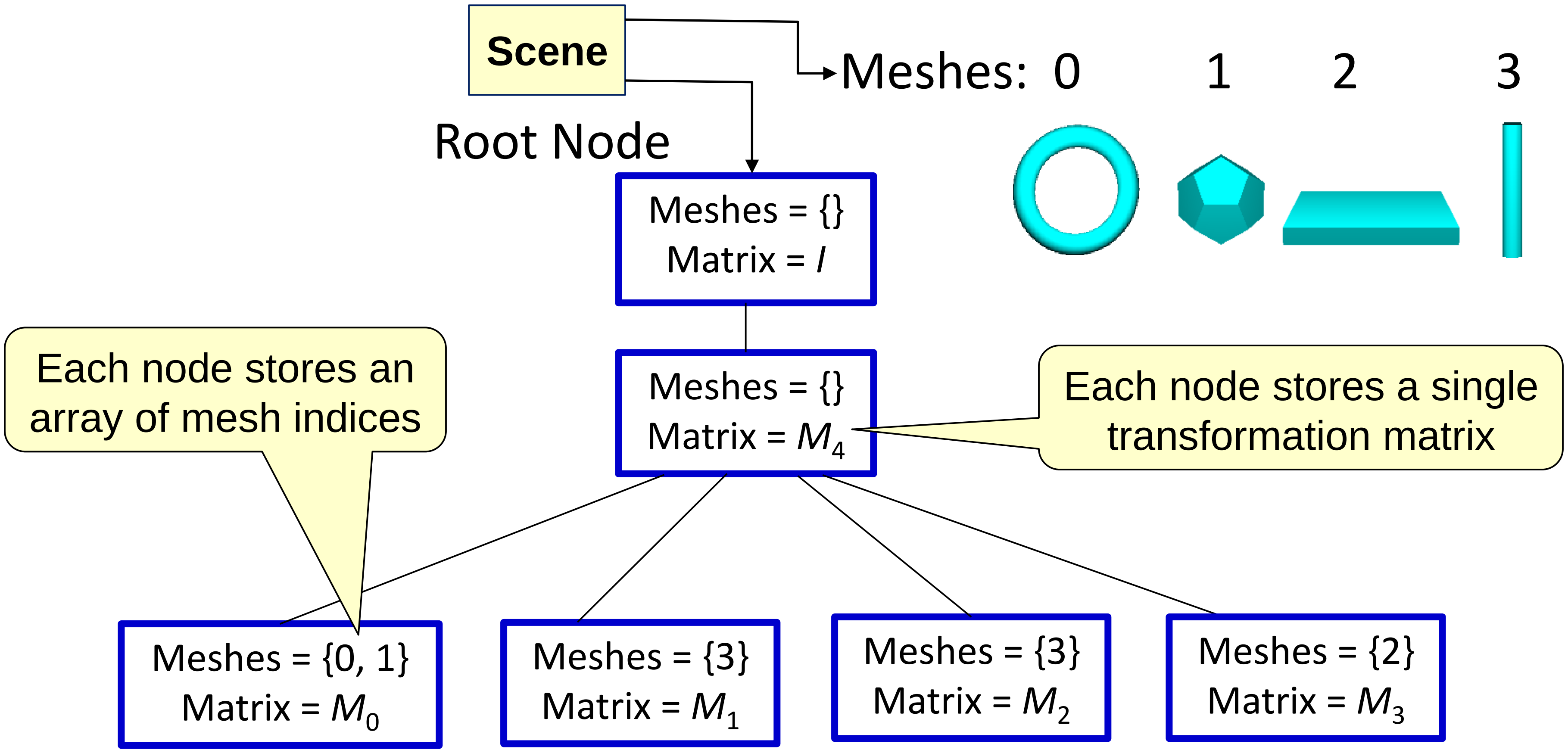
$M_1 = T(-2,0,0)S(1,0.8,1)$

$M_2 = T(2, 0, 0)\ S(1,0.8,1)$

$M_3 = I$

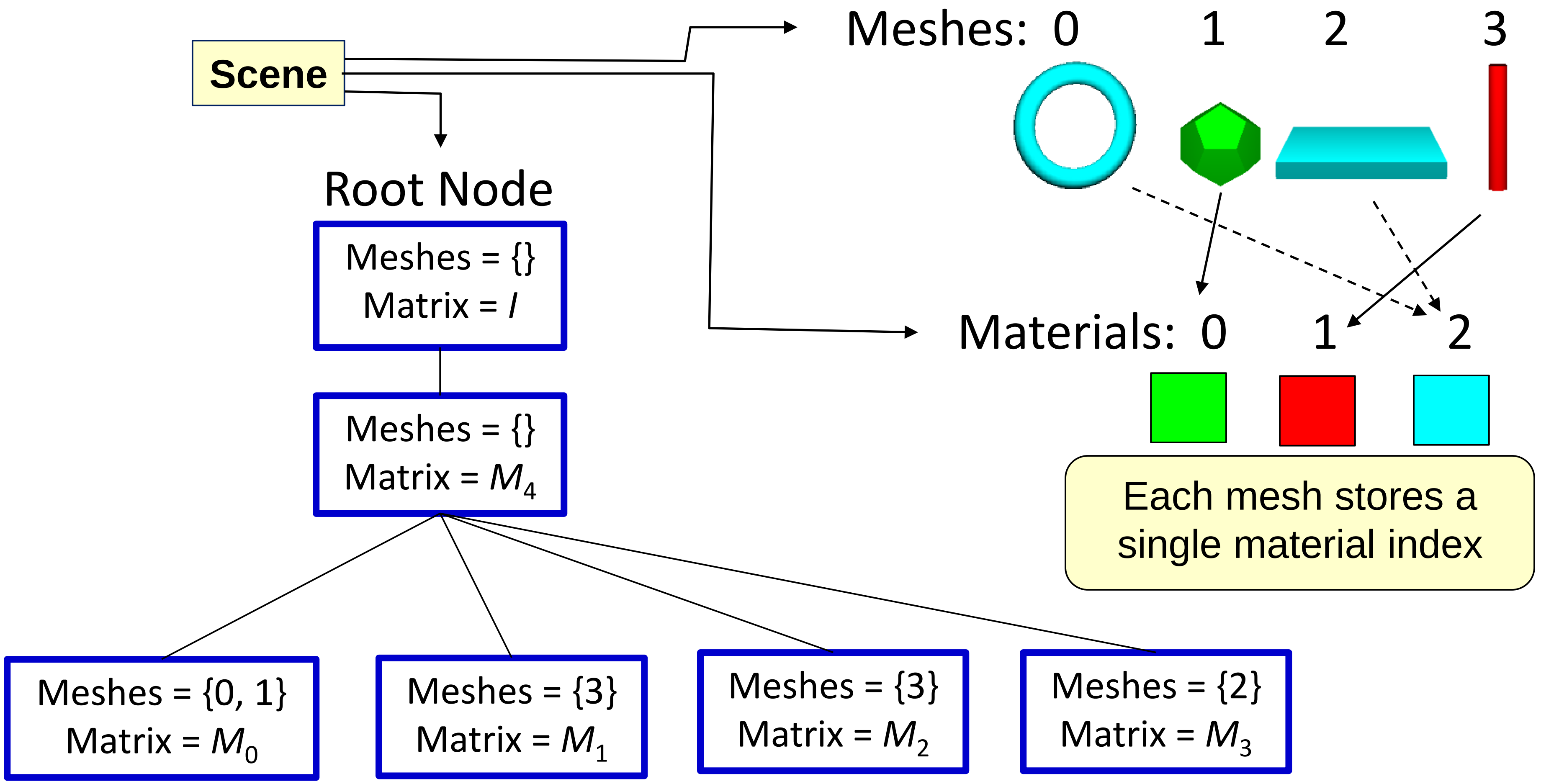
Model Representation

Meshes are stored in a common array as part of the **scene definition**. The mesh array is a scene attribute.



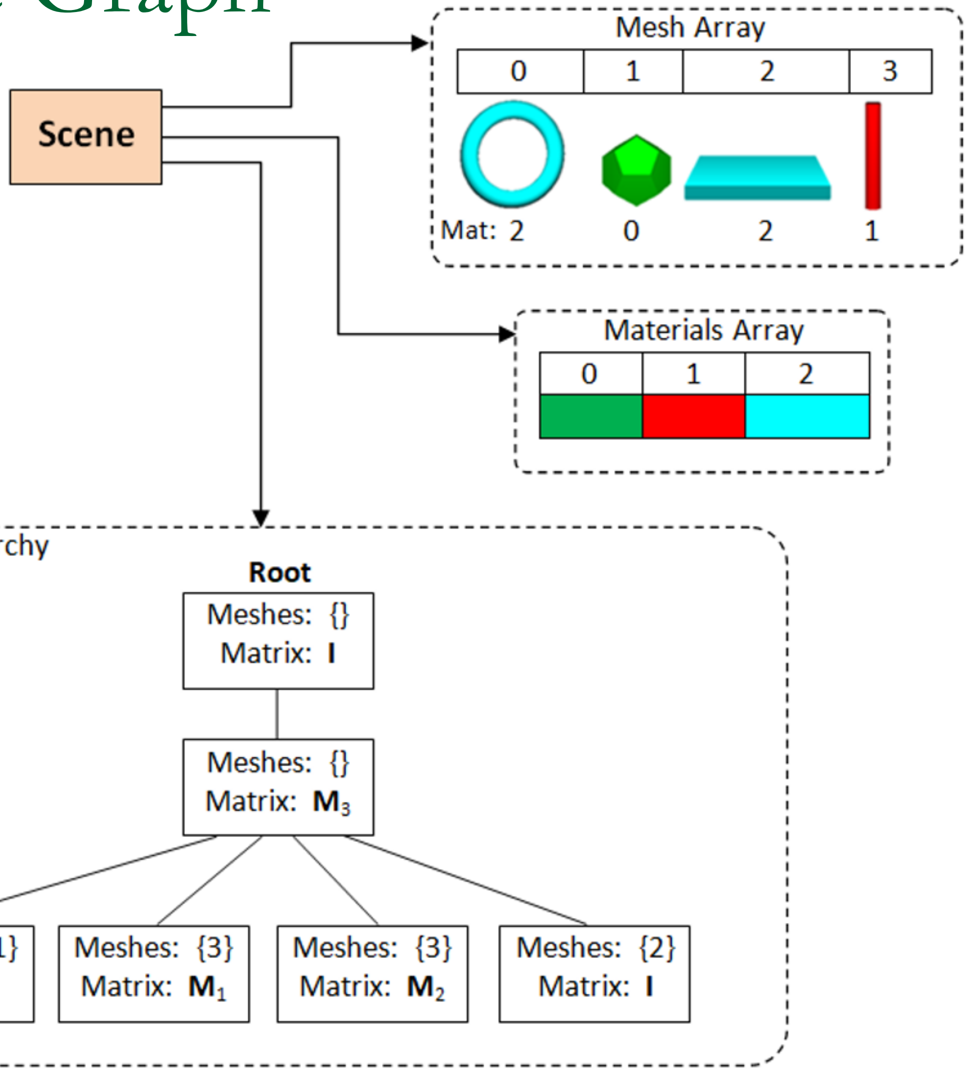
Scene Graph

The scene also stores a list of materials. Each mesh has an associated material index.



Each mesh stores a single material index

Scene Graph



$$\mathbf{M}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}_3 = \begin{bmatrix} 0.866 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Open Asset Import Library (Assimp)

http://www.assimp.org

Latest version - source: 5.0.1 (Jan 2020)

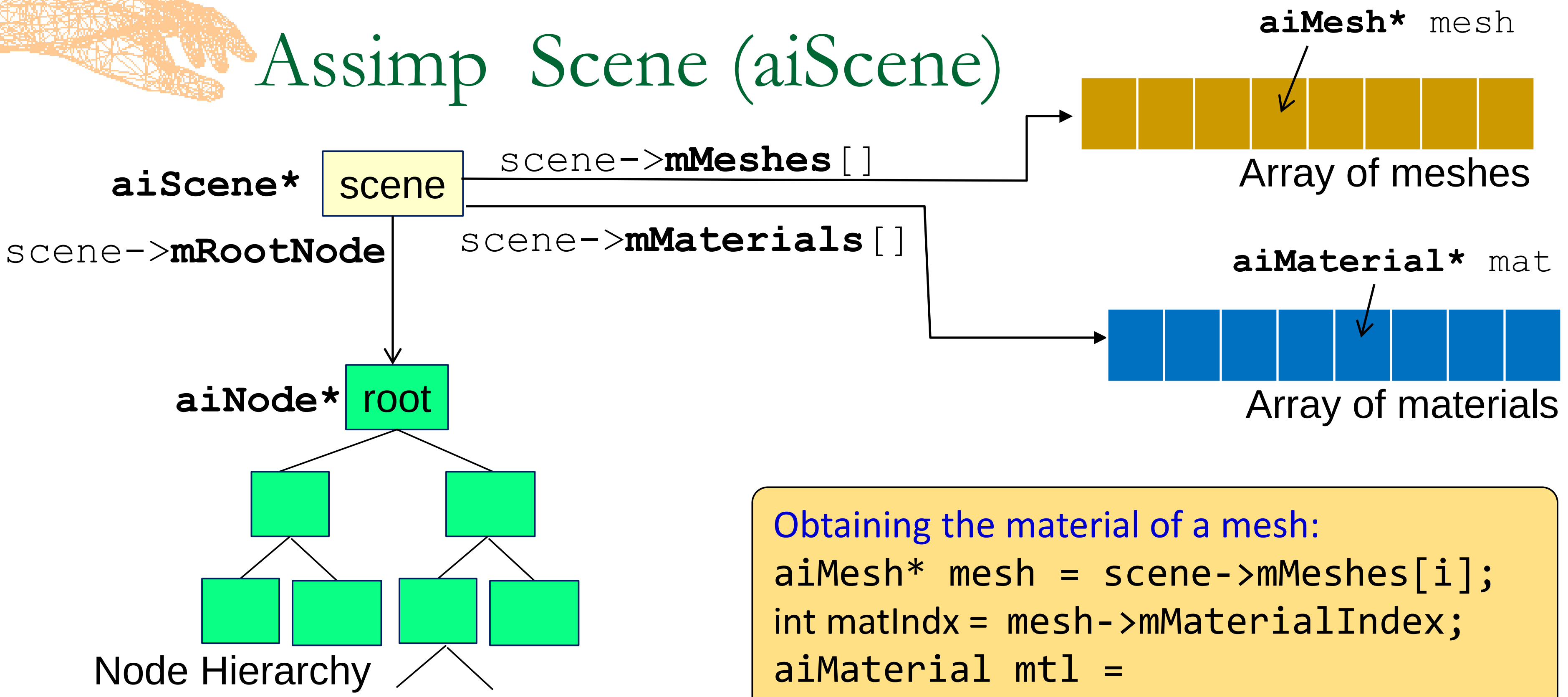
- A C++ library for reading and converting mesh files
- Generates the node hierarchy of a scene graph
- Also includes support for rigged models and complex skeletal animations

Open Asset Import Library (Assimp)

- Supports around 40 model file formats:

  OFF
  OBJ
  PLY
  STL
  DAE (Collada)
  BLEND (Blender)
  BVH (Biovision Hierarchy)
  3DS
  FBX
  X (Direct X)
  DXF (Autodesk)
  MDL (Game Studio)
  …

Assimp Scene (aiScene)



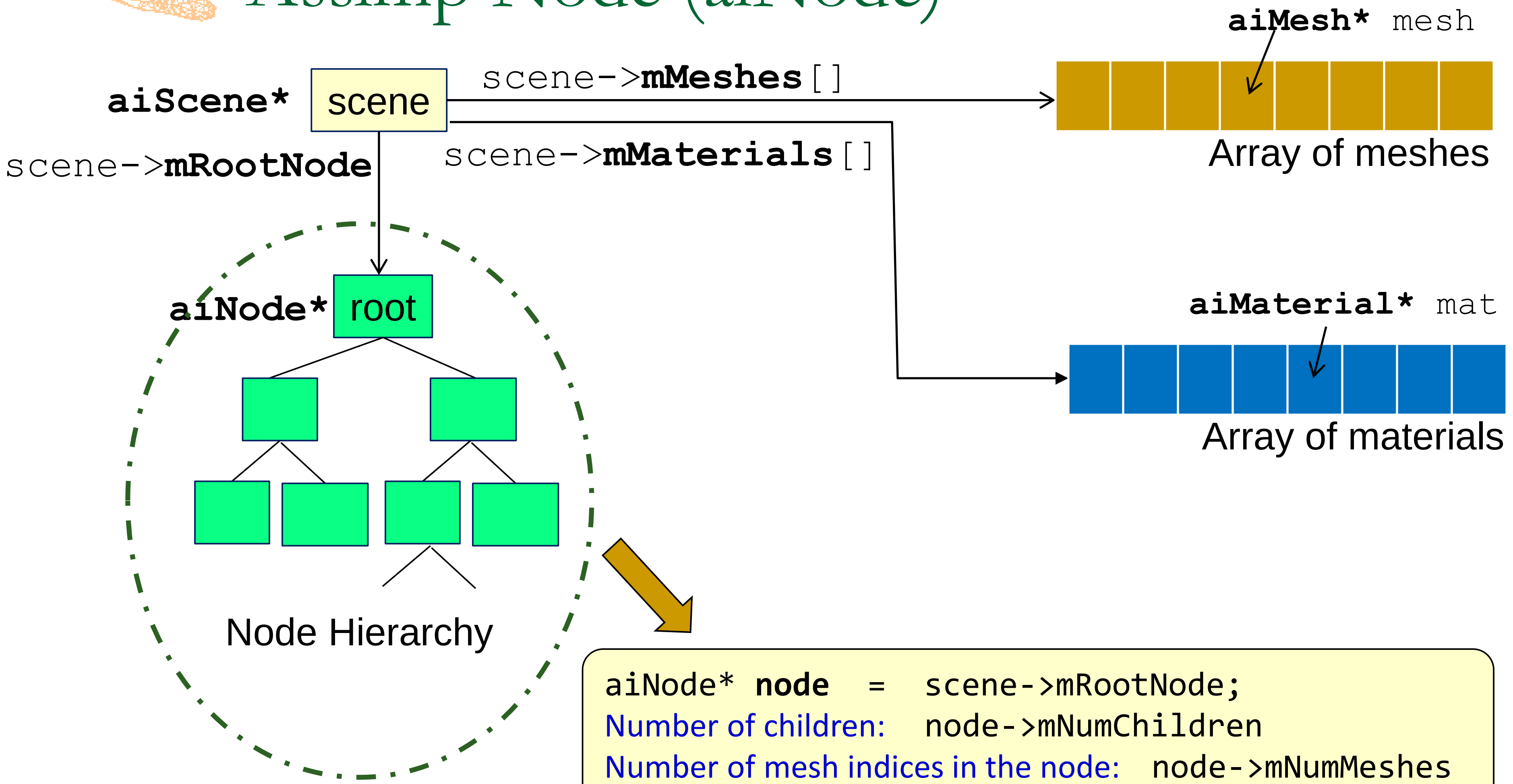
Obtaining the material of a mesh:

```
aiMesh* mesh = scene->mMeshes[i];
int matIndx = mesh->mMaterialIndex;
aiMaterial mtl =
        scene->mMaterials[matIndx];
```

```
aiScene* scene  = aiImportFile(file, aiProcessPreset_...);
```

Number of materials:	`scene->mNumMaterials`
Number of meshes:	`scene->mNumMeshes`
Number of lights:	`scene->mNumLights`

Assimp Node (aiNode)



```
aiNode* node   =  scene->mRootNode;
```

Number of children: `node->mNumChildren`

Number of mesh indices in the node: `node->mNumMeshes`

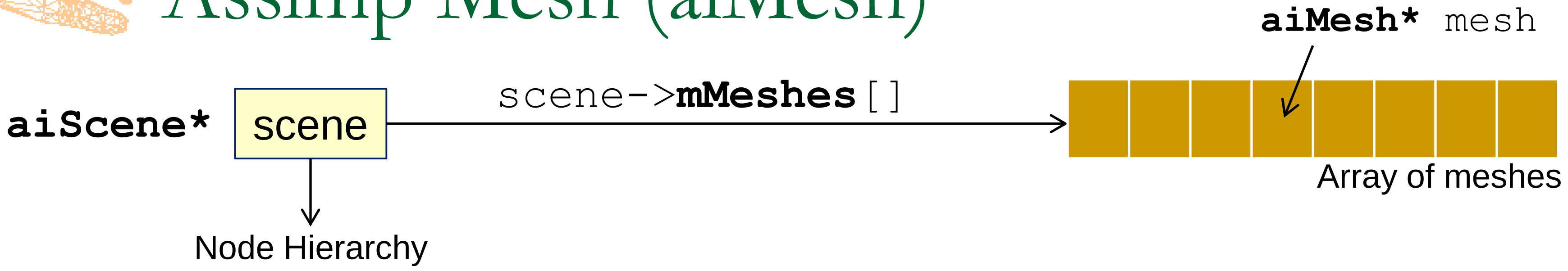
A mesh index in the node: `node->mMeshes[i]`

Node’s transformation matrix: `node->mTransformation`

Parent node: `node = node->mParent;`

Recursive descent: `node = node->mChildren[i];`

Assimp Mesh (aiMesh)



Mesh Attributes

```
aiMesh* mesh  =  scene->mMeshes[i];
```

Number of faces: `mesh->mNumFaces`

Number of vertices: `mesh->mNumVertices`

Name of the mesh: `mesh->mName`

Vertices of the mesh: `aiVector3D v = mesh->mVertices[i];`
i = 0,…,mesh->mNumVertices-1

Vertex normals: `aiVector3D n = mesh->mNormals[i];`
i = 0,…,mesh->mNumVertices-1

Vertex coordinates: `glVertex3f(v.x, v.y, v.z);`

Faces of the mesh: `mesh->mFaces[i]` i = 0,…,mesh->mNumFaces-1

Number of vertices of a face: `mesh->mFaces[i].mNumIndices`

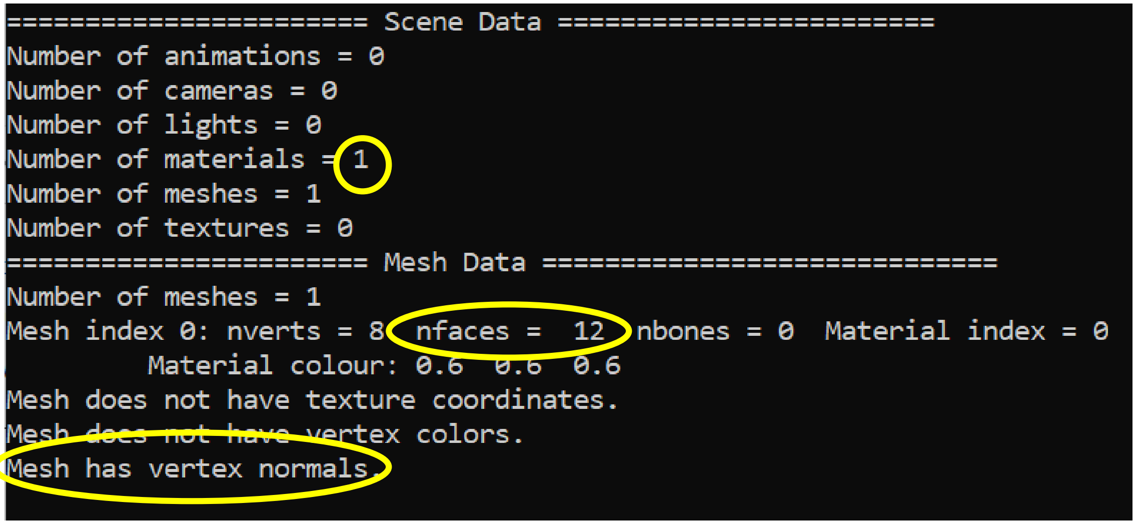
Loading a Mesh Object

```
#include <assimp/cimport.h>
#include <assimp/types.h>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
#include "assimp_extras.h"

bool loadModel(const char* fileName)
{
   scene = aiImportFile(fileName, aiProcessPreset_TargetRealtime_MaxQuality);
   if(scene == NULL) exit(1);
   printSceneInfo(scene);
   printMeshInfo(scene);
   get_bounding_box(scene, &scene_min, &scene_max);
   return true;
}
```

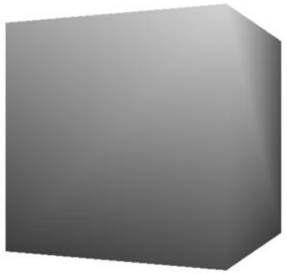
Import Postprocessing in Assimp

Cube.off

```
OFF
8 6 0
0 1 0
0 0 0
1 0 0
1 1 0
0 1 1
0 0 1
1 0 1
1 1 1
4 0 3 2 1
4 4 5 6 7
4 0 1 5 4
4 7 6 2 3
4 0 4 7 3
4 5 1 2 6
```



```
======================= Scene Data ========================
Number of animations = 0
Number of cameras = 0
Number of lights = 0
Number of materials = 1
Number of meshes = 1
Number of textures = 0
======================= Mesh Data =============================
Number of meshes = 1
Mesh index 0: nverts = 8 nfaces =  12 nbones = 0  Material index = 0
         Material colour: 0.6  0.6  0.6
Mesh does not have texture coordinates.
Mesh does not have vertex colors.
Mesh has vertex normals.
```



Mesh Object

Attributes of a ‘mesh’ object

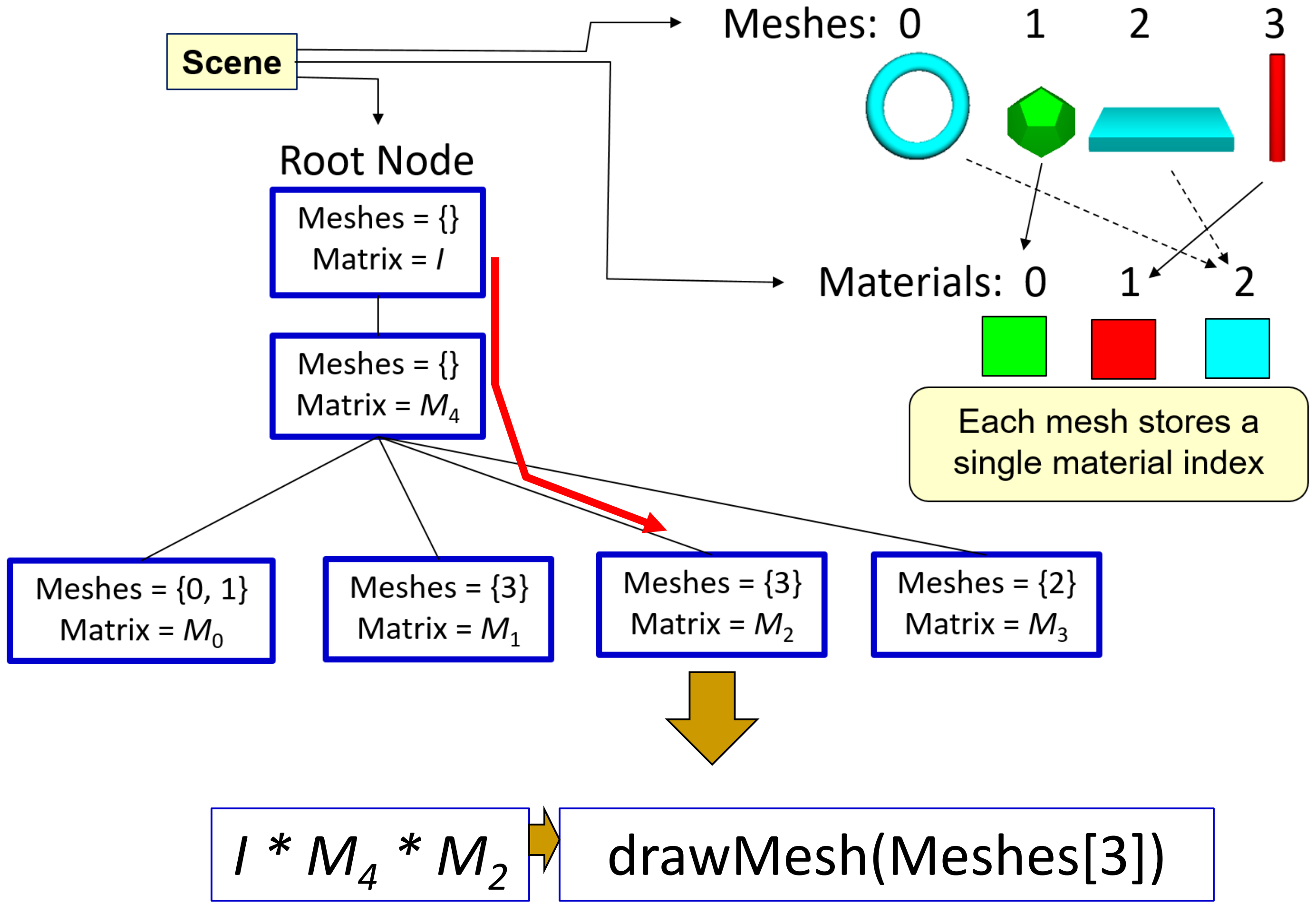
```
int nfaces = mesh->mNumFaces;            //Number of faces
int nverts = mesh->mNumVertices;         //Number of vertices
int matindx = mesh->mMaterialIndex;      //Get index of material

aiVector3D vertex = mesh->mVertices[index];  //a vertex
aiVector3D normal = mesh->mNormals[index];   //vertex normal
aiFace* face = &mesh->mFaces[0];              //a face
```

Drawing a ‘mesh’ object

```
meshDraw(aiMesh* mesh)
```

```
for (int k = 0; k < mesh->mNumFaces; k++) {
   face = &mesh->mFaces[k];
   glBegin(GL_TRIANGLES);
       for(int i = 0; i < face->mNumIndices; i++) {
           int index = face->mIndices[i];
           glNormal3fv(&mesh->mNormals[index].x);
           glVertex3fv(&mesh->mVertices[index].x);
       }
    glEnd();
}
```

Rendering a Mesh Object



$I * M_4 * M_2$ drawMesh(Meshes[3])

Transformations Mesh Vertices

Rendering Using a Scene Graph

```
void display() {
   ...
   render(scene,  scene->mRootNode);
}
```

```
void render(aiScene *sc, aiNode* nd) {
    aiMatrix4x4 m = nd->mTransformation;
    aiTransposeMatrix4(&m);
    glPushMatrix();
    glMultMatrixf((float*)&m);
    for (int n = 0; n < nd->mNumMeshes; n++) {
        meshIndex = nd->mMeshes[n];
        mesh = scene->mMeshes[meshIndex];
        drawMesh(mesh); }

        for (int n = 0; n < nd->mNumChildren; n++)
               render(sc, nd->mChildren[n]);
    glPopMatrix();
}
```