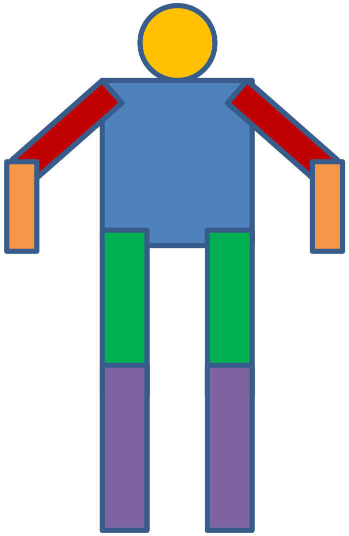
Scene Graph

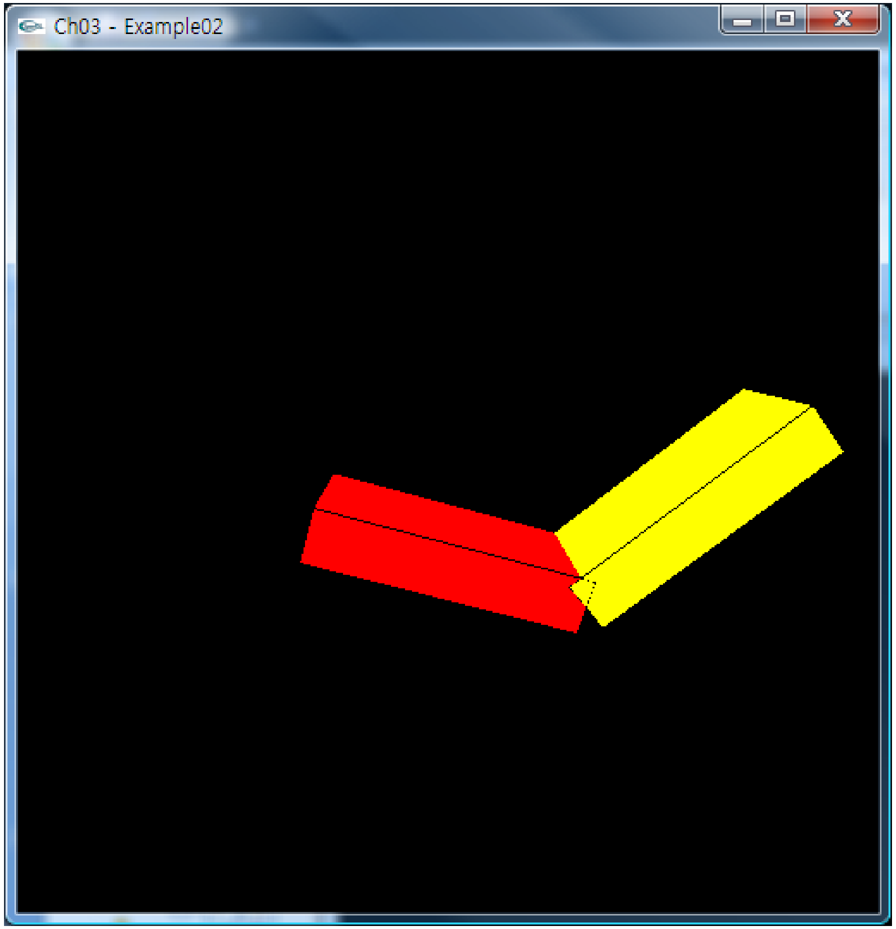
KOREA UNIVERSITY

KUCG

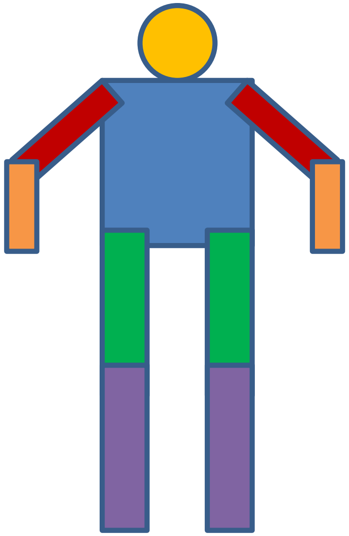
The scenes we want to draw…

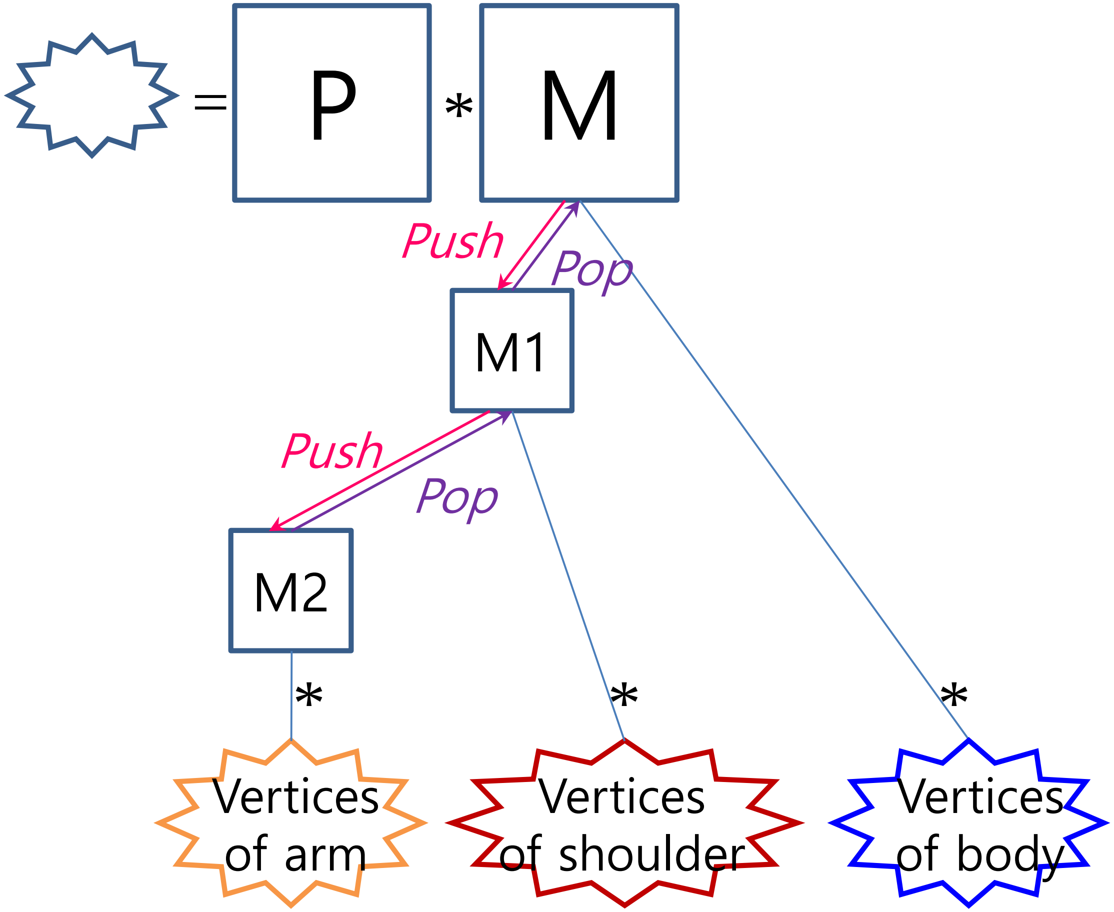
‘Robot Arm Demo’ Revisited



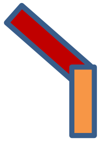


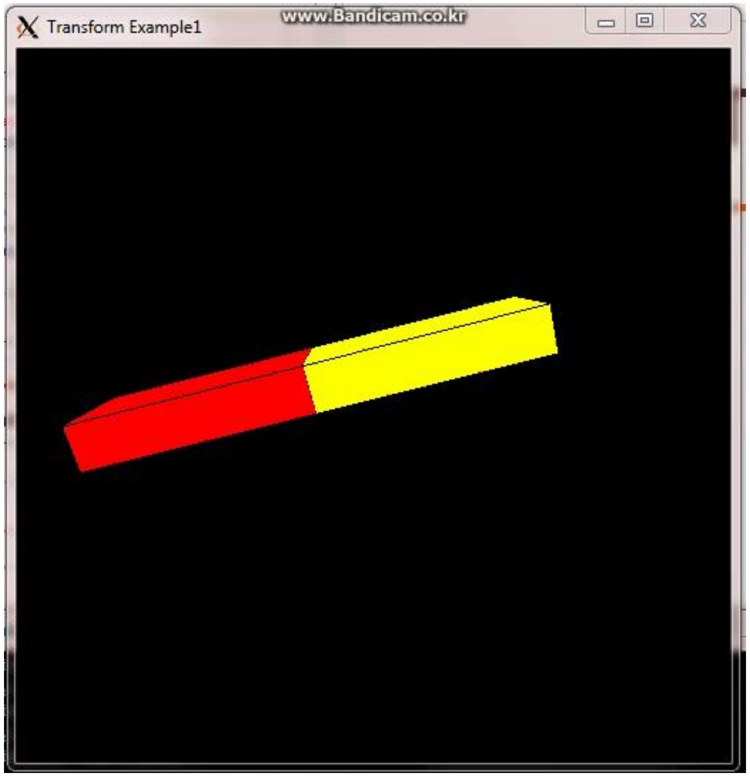
Object Hierarch





Procedural Representation





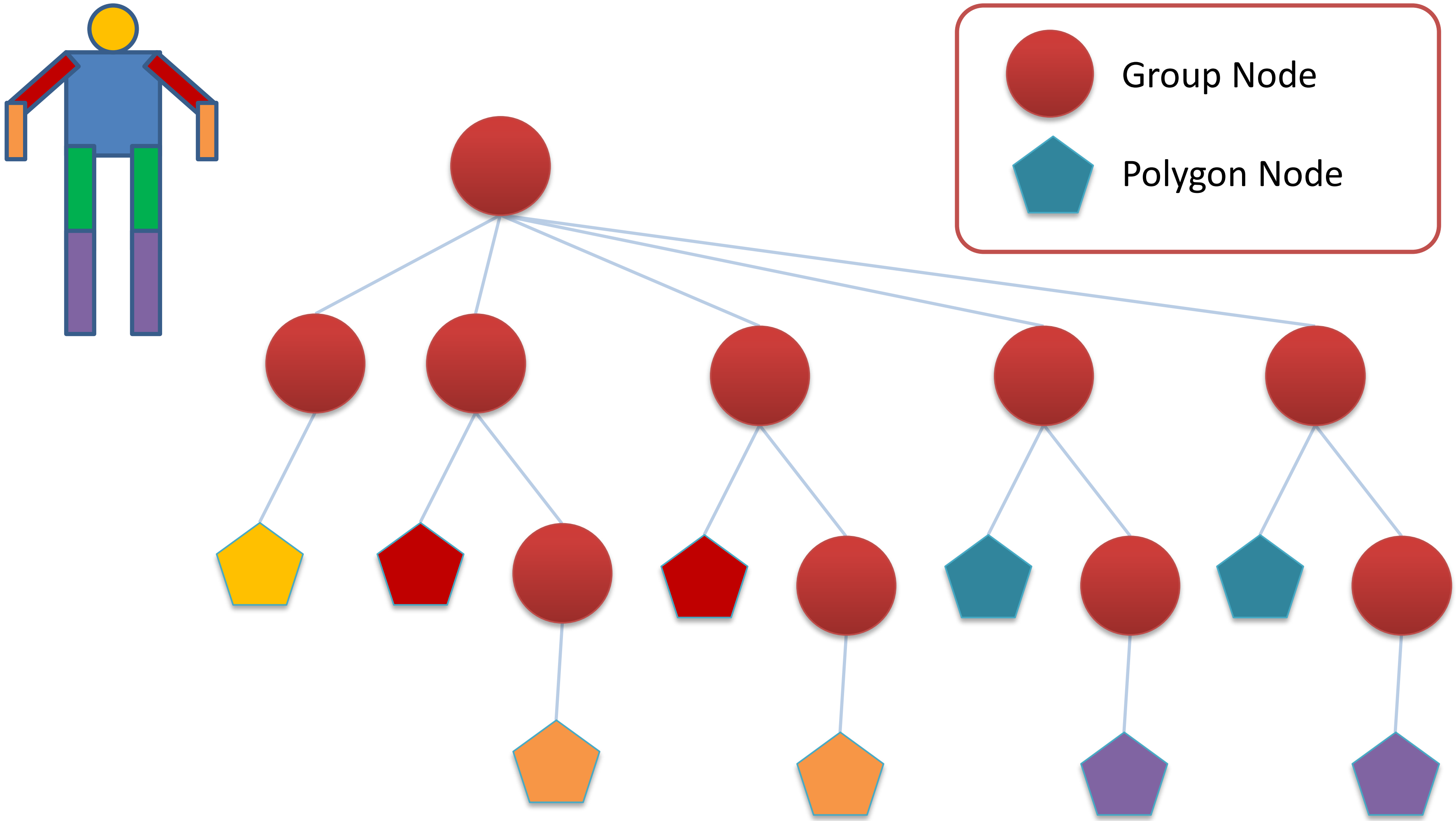
```
int shoulder = 0, elbow = 0;
void display() {
        glPushMatrix();
                glRotatef(20, 1, 0, 1);
                glPushMatrix();
                        glTranslatef(-1.0, 0.0, 0.0);
                        glRotatef(shoulder, 0.0, 0.0, 1.0);
                        glTranslatef(1.0, 0.0, 0.0);

                        glPushMatrix();
                                glScalef(2.0, 0.4, 1.0);
                                glColor3f(1,0,0);
                                glutSolidCube(1.0);
                        glPopMatrix();

                        glTranslatef(1.0, 0.0, 0.0);
                        glRotatef(elbow, 0.0, 0.0, 1.0);
                        glTranslatef(1.0, 0.0, 0.0);

                        glPushMatrix();
                                glScalef(2.0, 0.4, 1.0);
                                glColor3f(1,1,0);
                                glutSolidCube(1.0);
                        glPopMatrix();
                glPopMatrix();
        glPopMatrix();
        glXSwapBuffers(dpy, win);
}
```

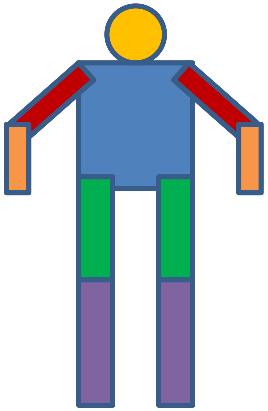
Graph Representation



● Group Node

⬟ Polygon Node

Procedural vs. Graph Representation

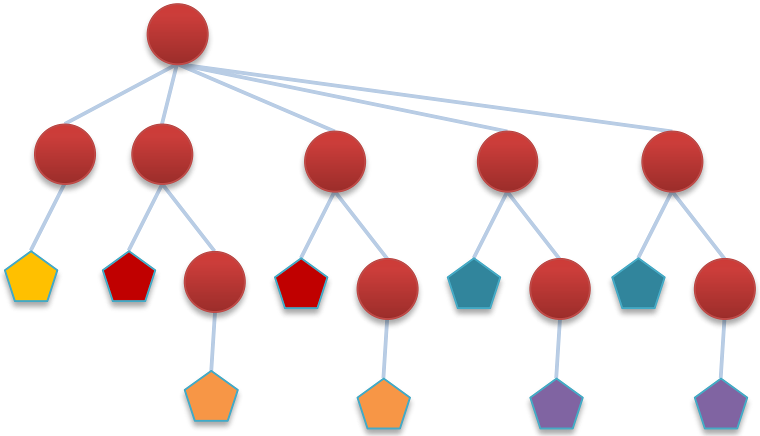


```
int shoulder = 0, elbow = 0;
void display() {
    glPushMatrix();
        glRotatef(20, 1, 0, 1);
        glPushMatrix();
            glTranslatef(-1.0, 0.0, 0.0);
            glRotatef(shoulder, 0.0, 0.0, 1.0);
            glTranslatef(1.0, 0.0, 0.0);

            glPushMatrix();
                glScalef(2.0, 0.4, 1.0);
                glColor3f(1,0,0);
                glutSolidCube(1.0);
            glPopMatrix();

            glTranslatef(1.0, 0.0, 0.0);
            glRotatef(elbow, 0.0, 0.0, 1.0);
            glTranslatef(1.0, 0.0, 0.0);

            glPushMatrix();
                glScalef(2.0, 0.4, 1.0);
                glColor3f(1,1,0);
                glutSolidCube(1.0);
            glPopMatrix();
        glPopMatrix();
    glPopMatrix();
    glXSwapBuffers(dpy, win);
}
```

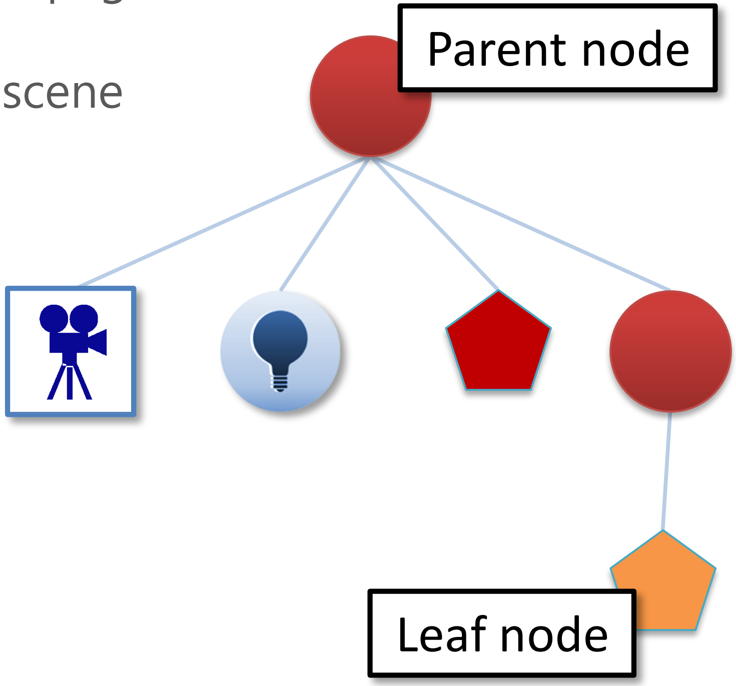


Whenever the scene needs to be redrawn, “visiting the graph in a certain order” ends up drawing the objects in various states.

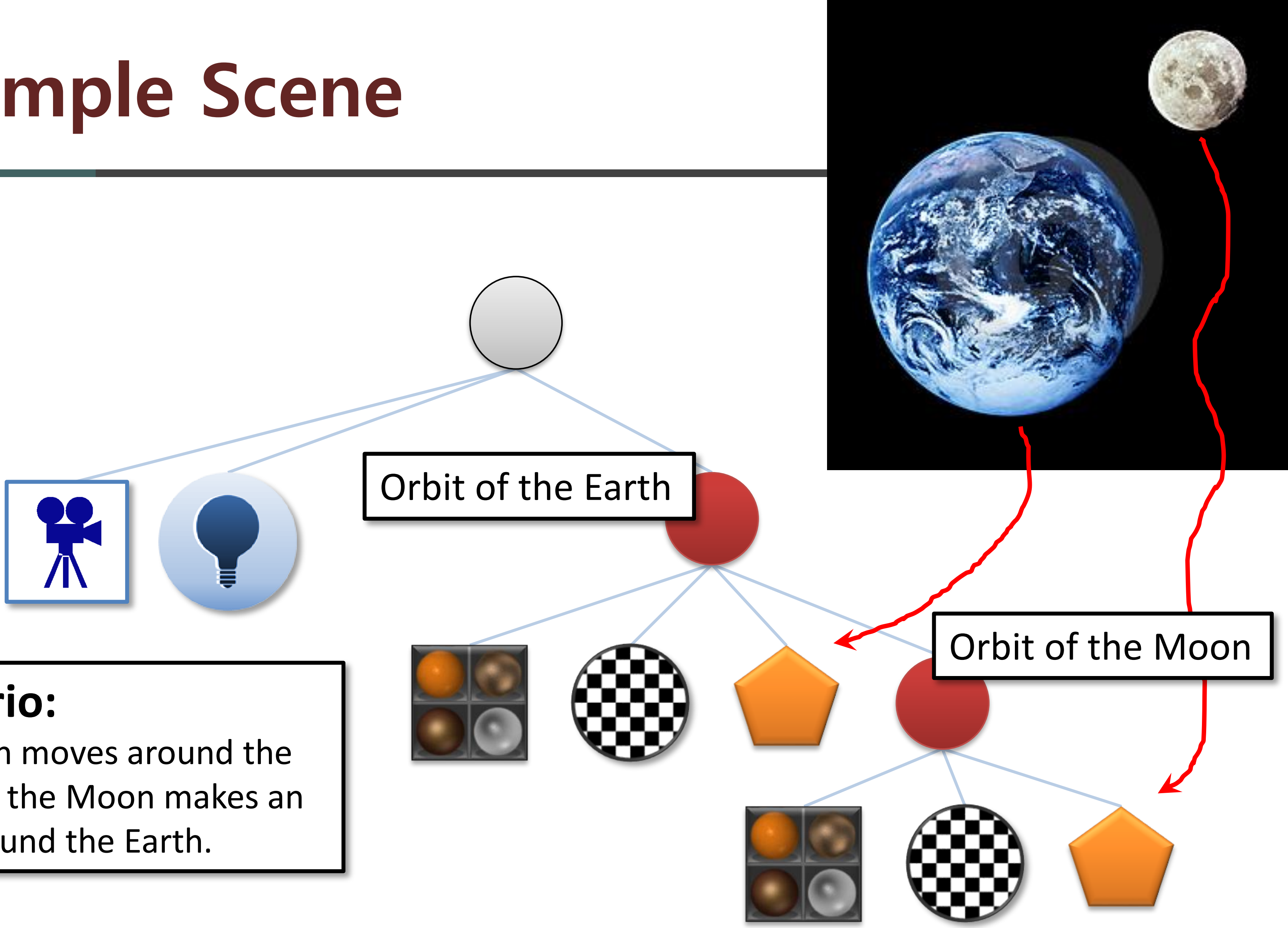
Implementing undo/redo is easier if a graph representation is internally used.

Scene Graph

- The scene graph is a structure that arranges the logical representation of a graphical scene.
  - Nodes are organized in a graph or tree structure.
  - A 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.
  - It allows the user to manage complex scene logically, and efficiently.
  - **Basic nodes:**
    - Group (or transform)
    - Polygon mesh
    - Shader
    - Texture
    - Camera
    - Light source
    - And more…

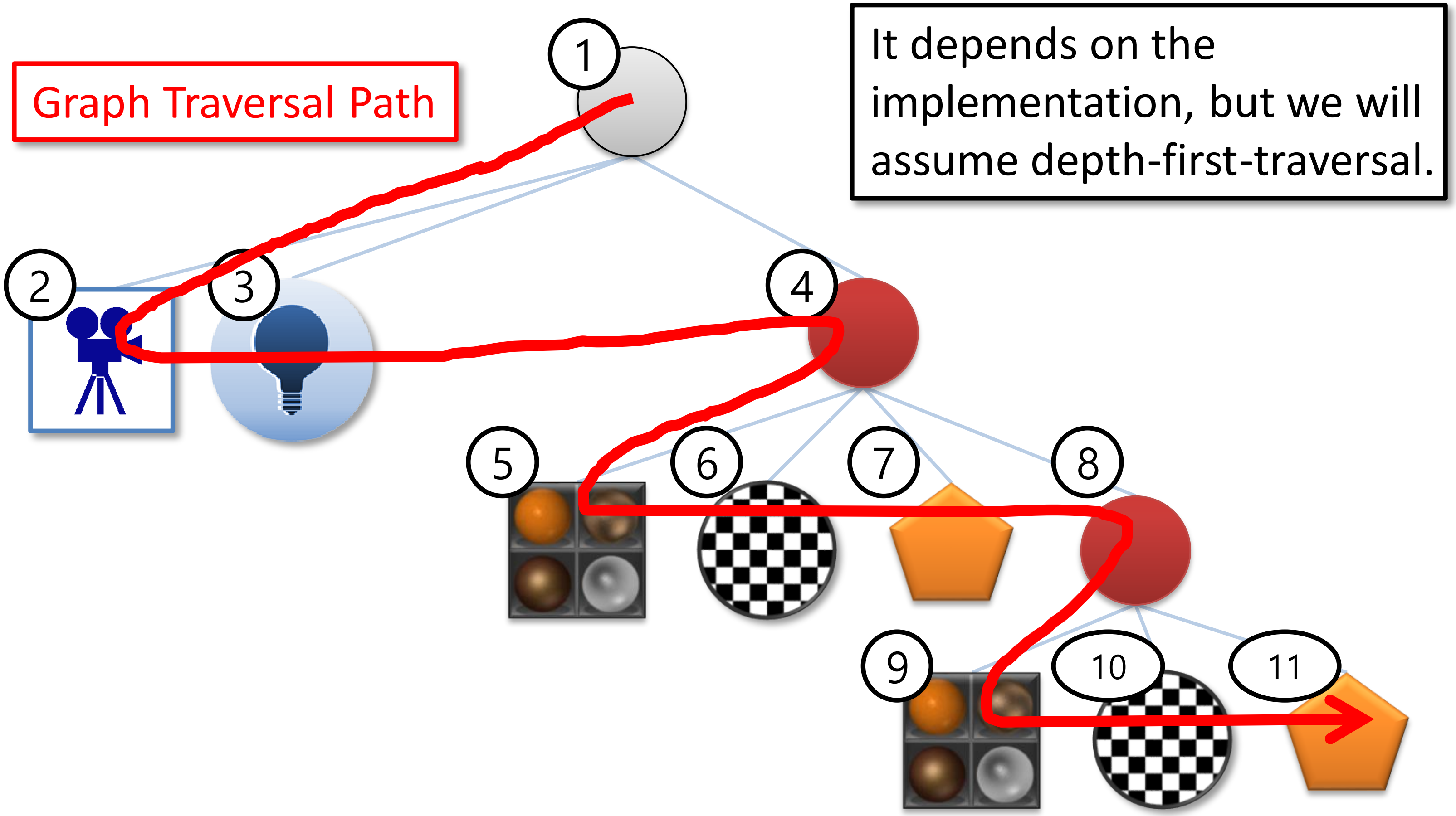


Example Scene



Scenario:
The Earth moves around the Sun, and the Moon makes an orbit around the Earth.

Traversing Scene Graph

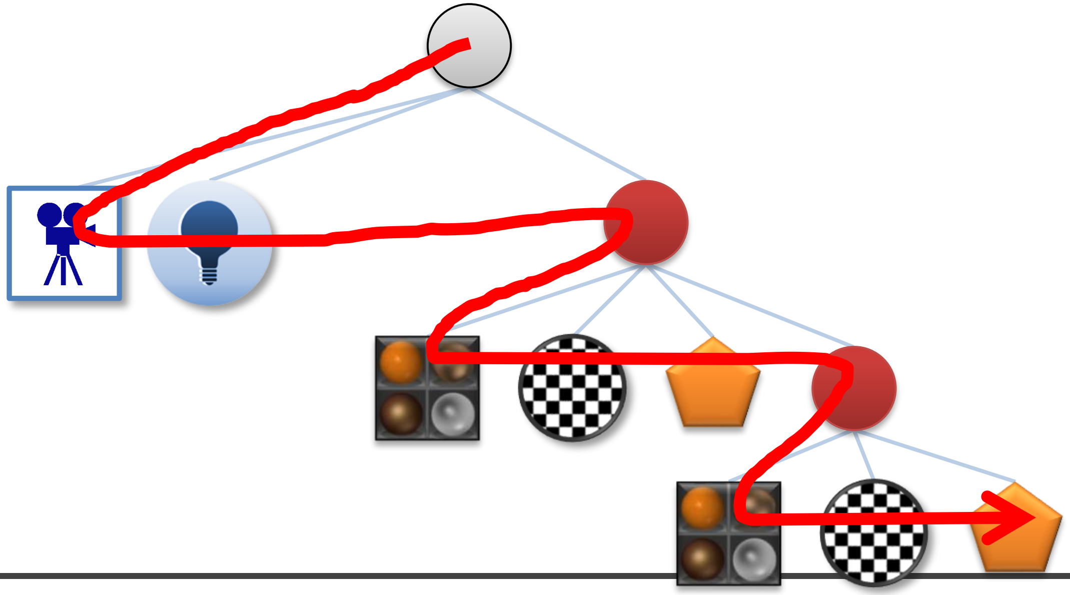


It depends on the implementation, but we will assume depth-first-traversal.

Options for State Inheritance

- Option 1: A leaf node's state is defined by the nodes in a direct path between the scene graph's root and the leaf.

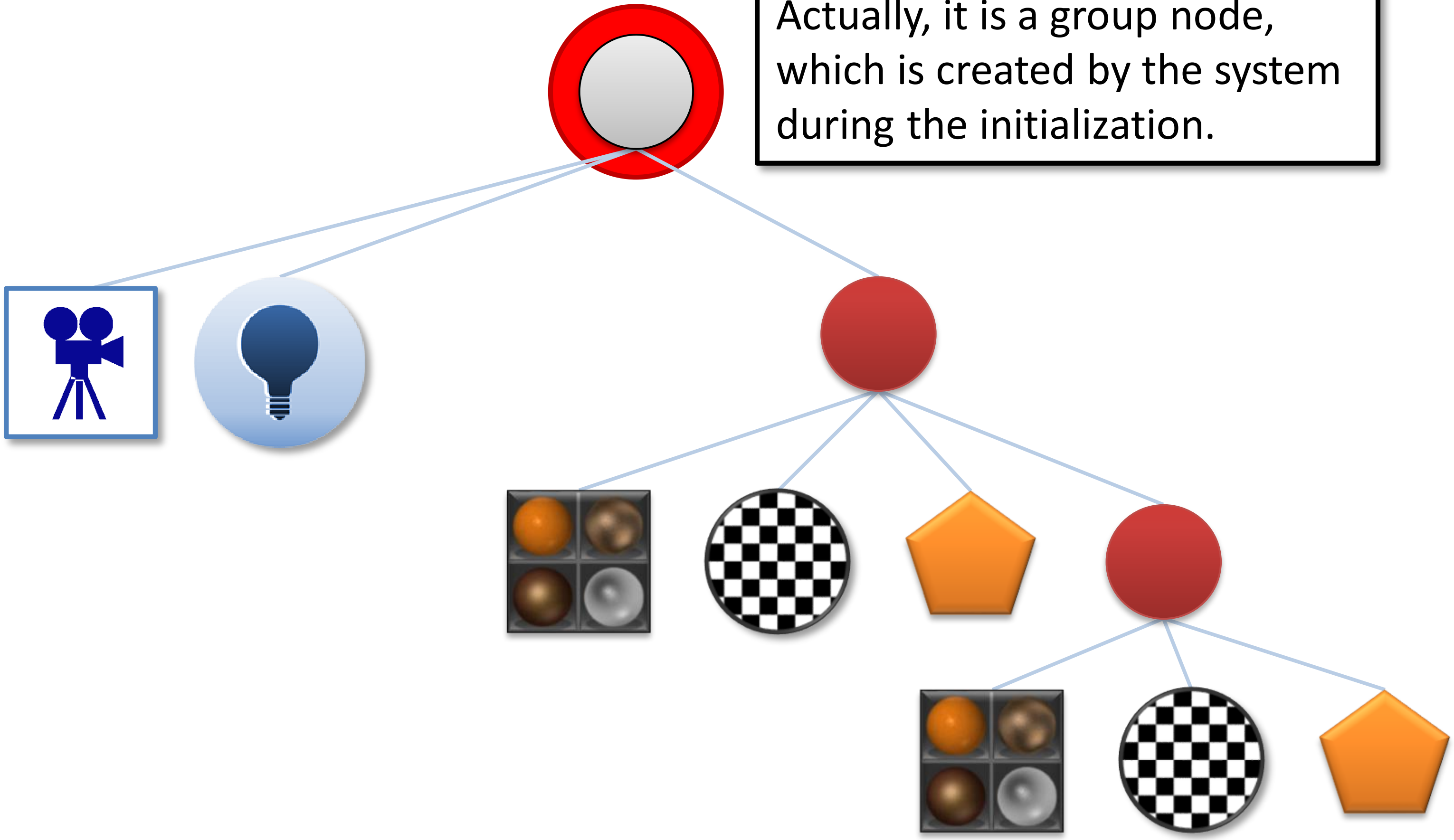
- **Option 2: When a node above or to the left of a node changes the graphics state, the change affects the graphics state of all nodes below it or to its right.**
  - This class will take Option 2.



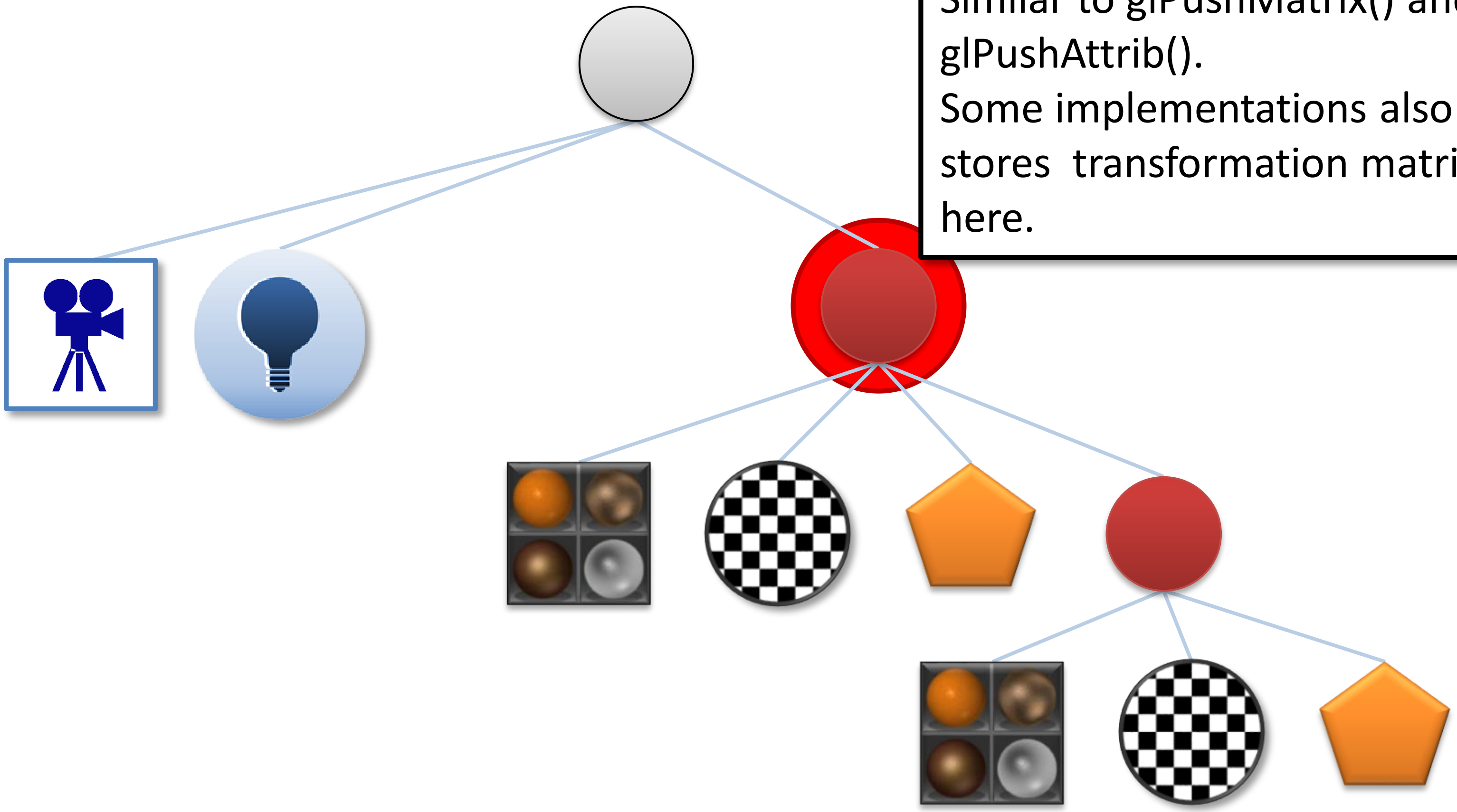
Root of the Scene

Root Node :

Unique root node of the scene.
Actually, it is a group node,
which is created by the system
during the initialization.



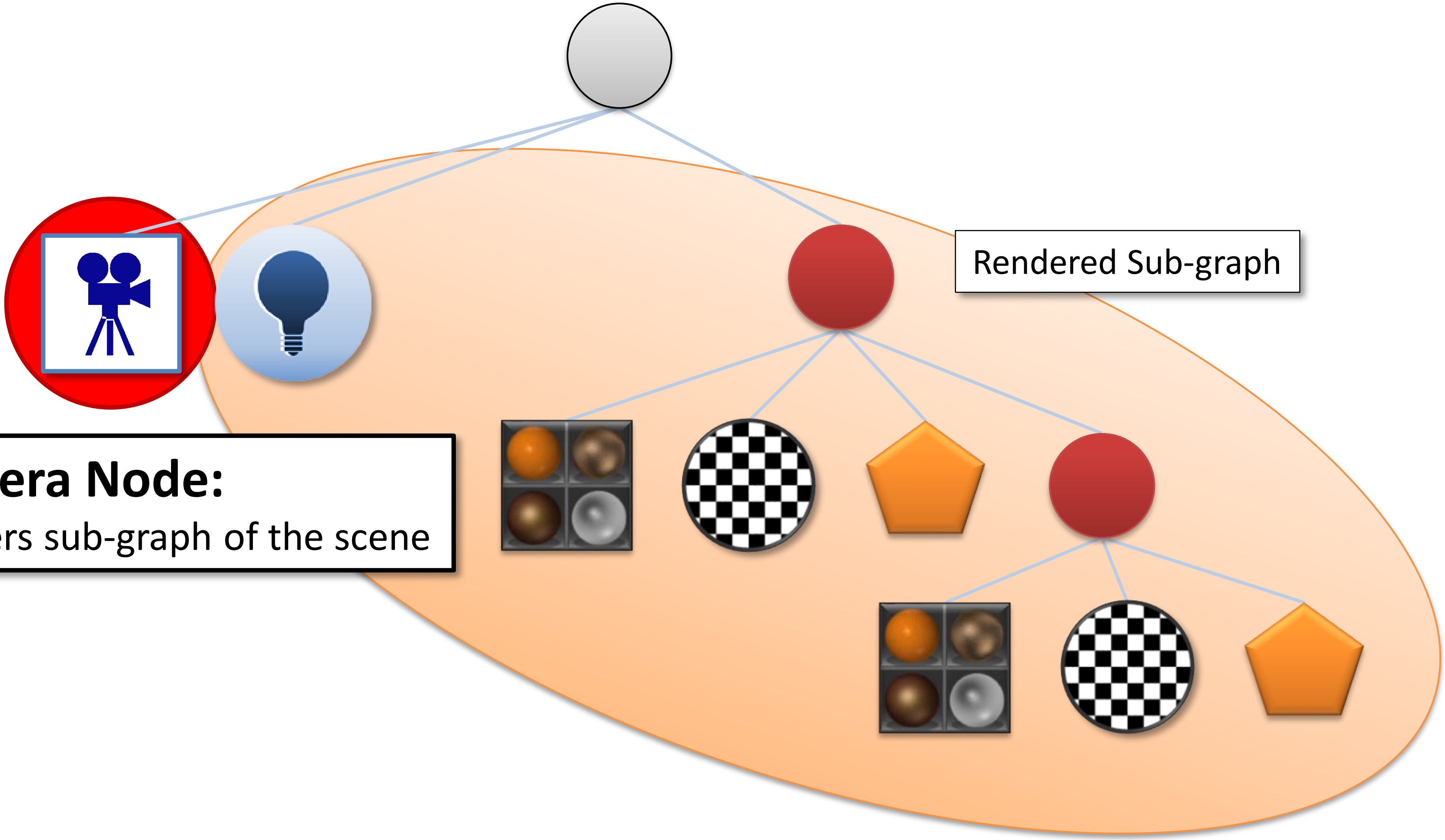
Group Node



Group Node:

Separates graphical state of its children from other nodes. Similar to glPushMatrix() and glPushAttrib(). Some implementations also stores transformation matrix here.

Camera

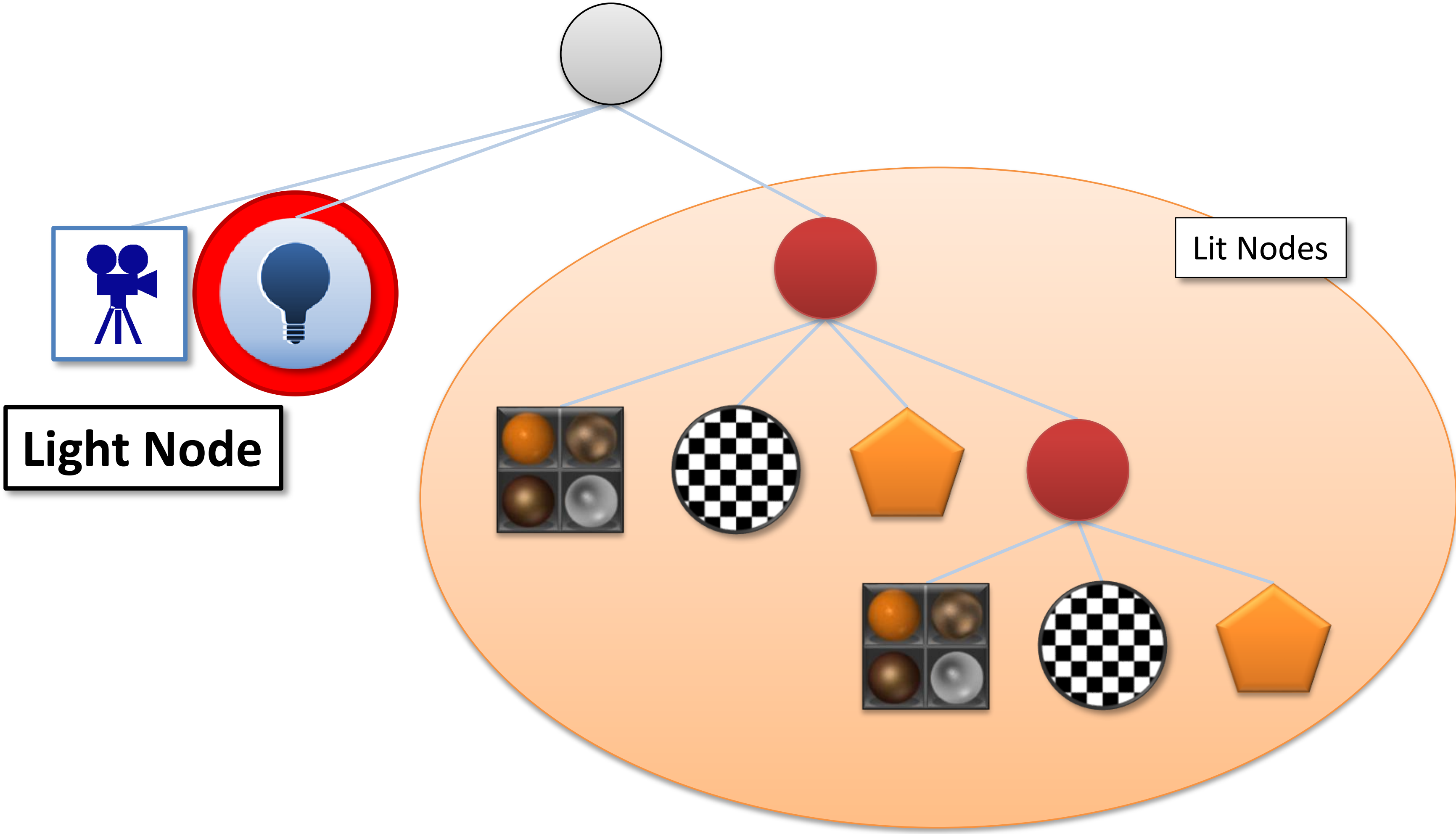


Rendered Sub-graph

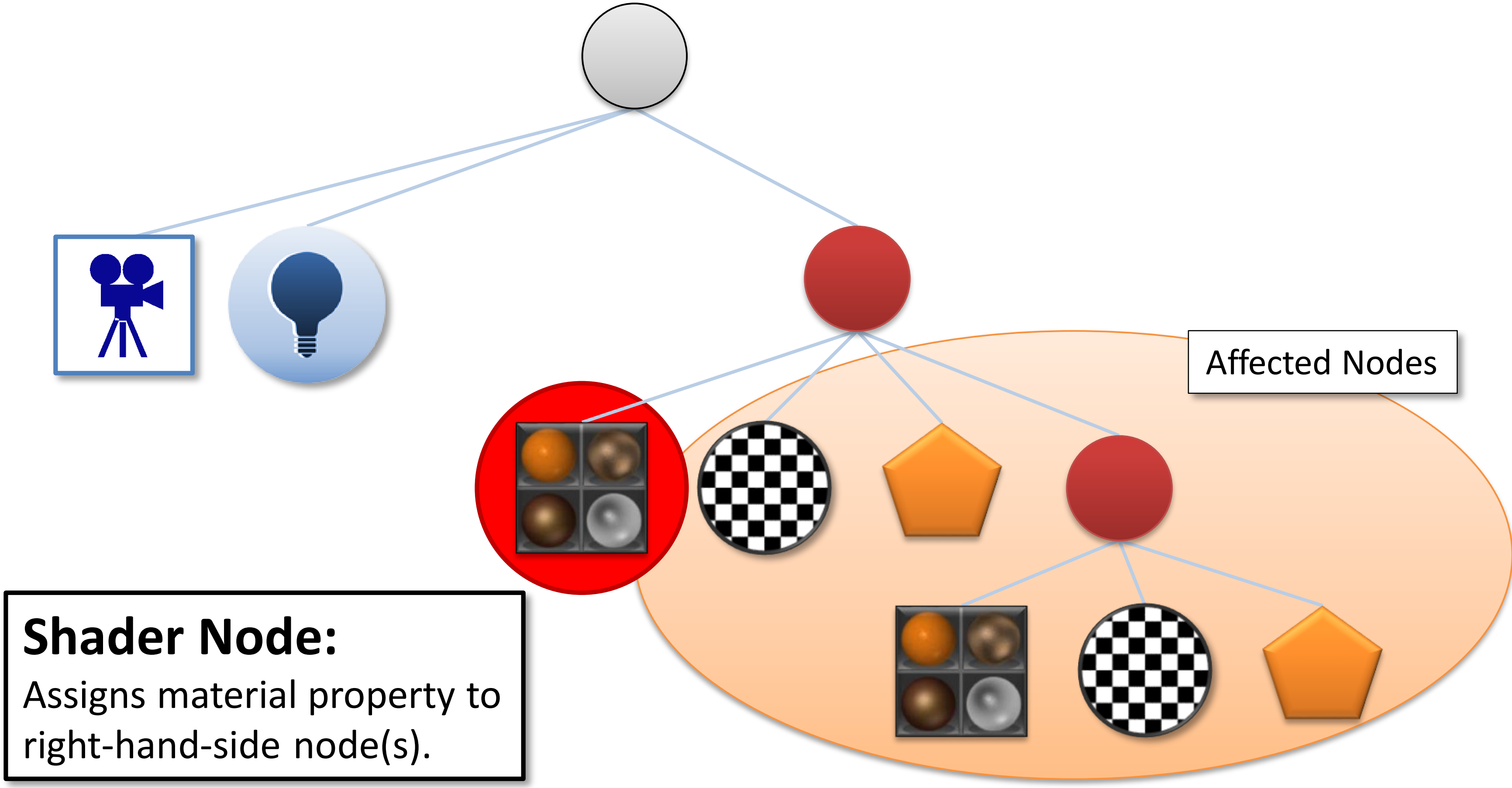
Camera Node:

Renders sub-graph of the scene

Light



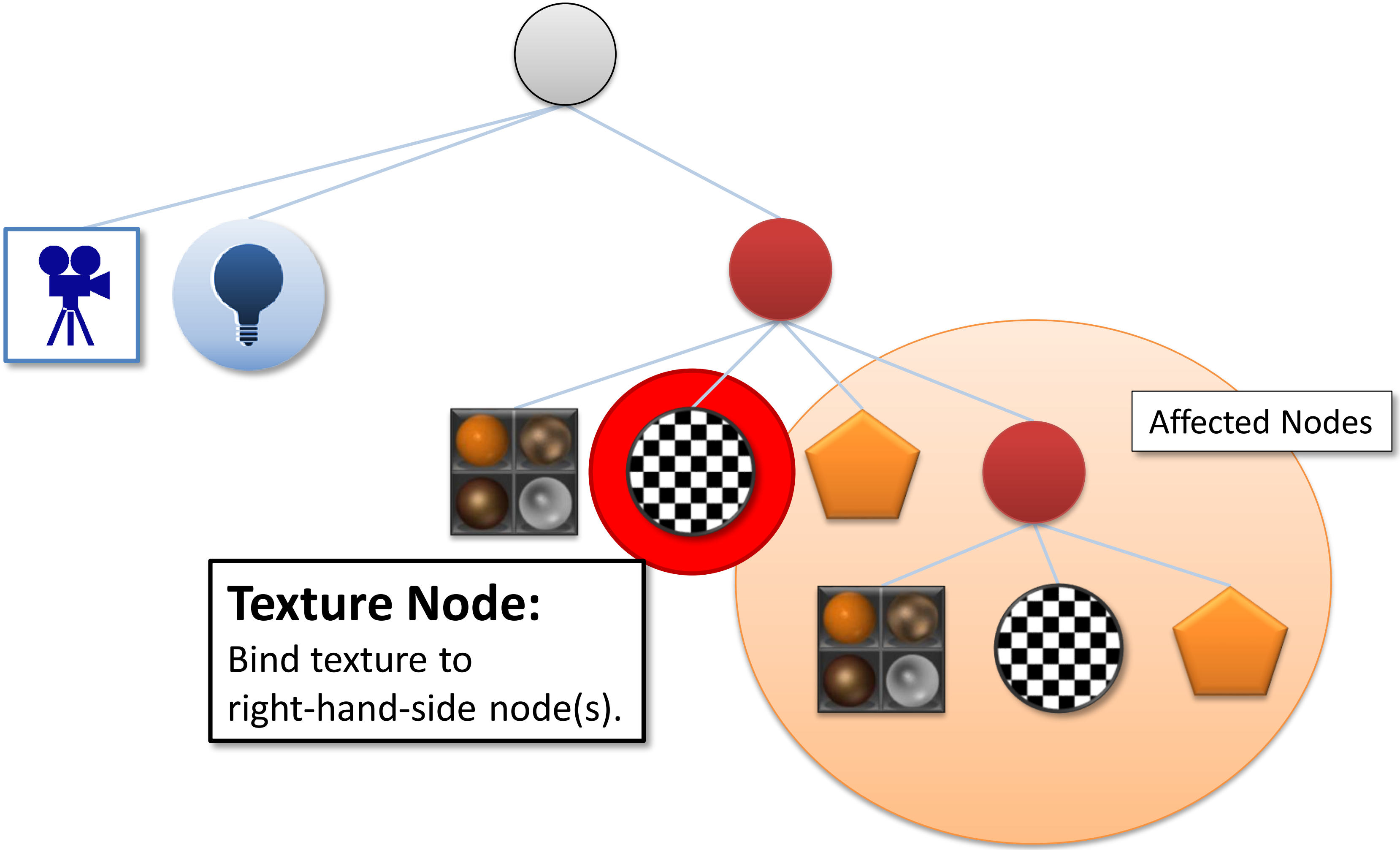
Shader



Shader Node:

Assigns material property to right-hand-side node(s).

Texture

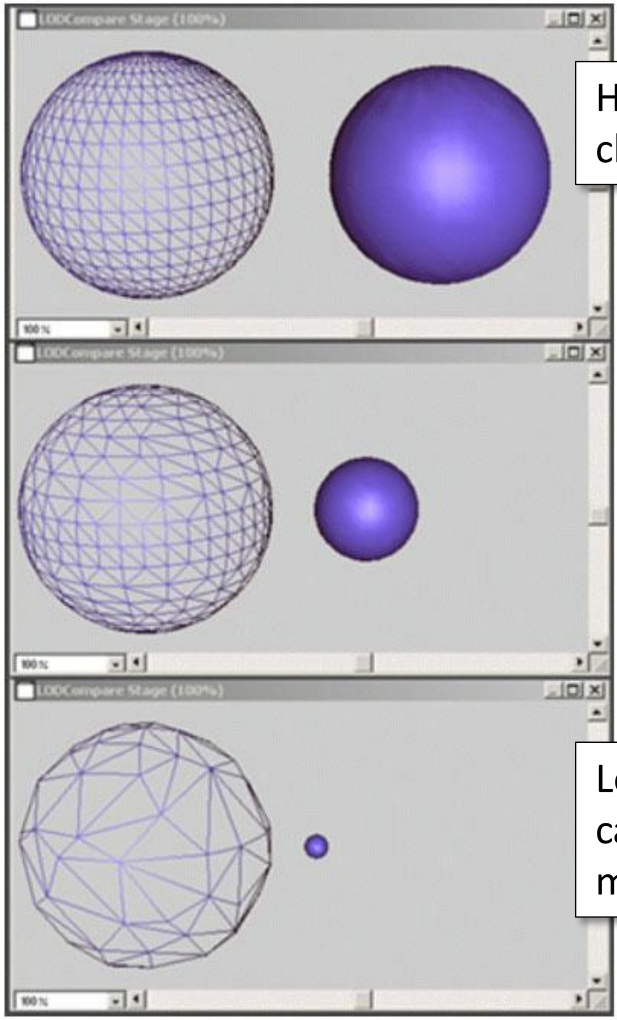


Affected Nodes

Texture Node:

Bind texture to right-hand-side node(s).

Switch (Selector) Node

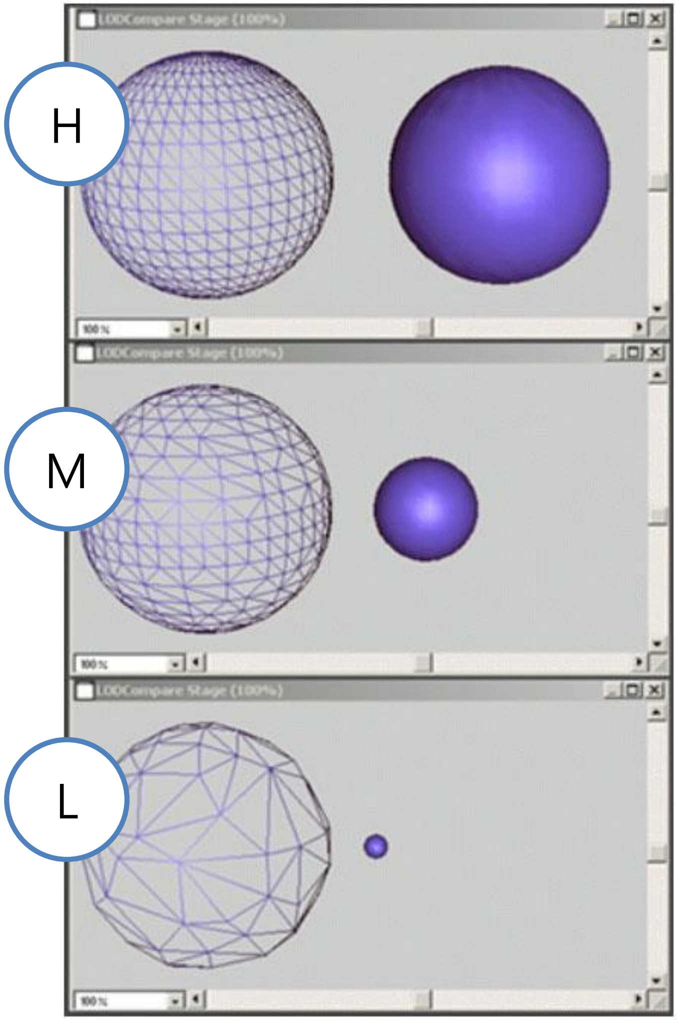


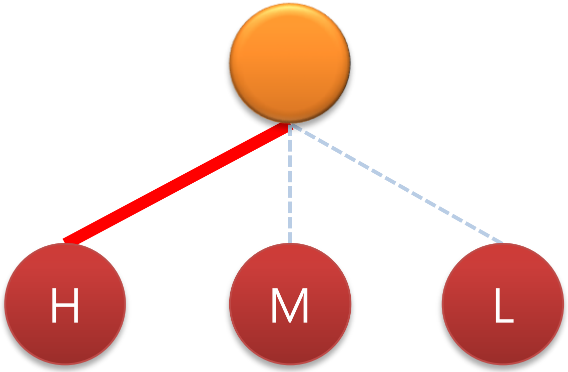
High-resolution mesh for close-up rendering

This concept is called **Level of Detail (LoD)**, and LoD techniques increases the efficiency of rendering by decreasing the workload on graphics pipeline stages, usually vertex transformations.

Low-resolution mesh when camera moves away from the mesh

Switch (Selector) Node





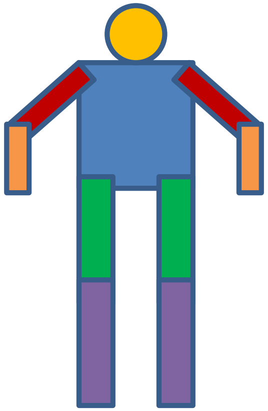
Switch Node:

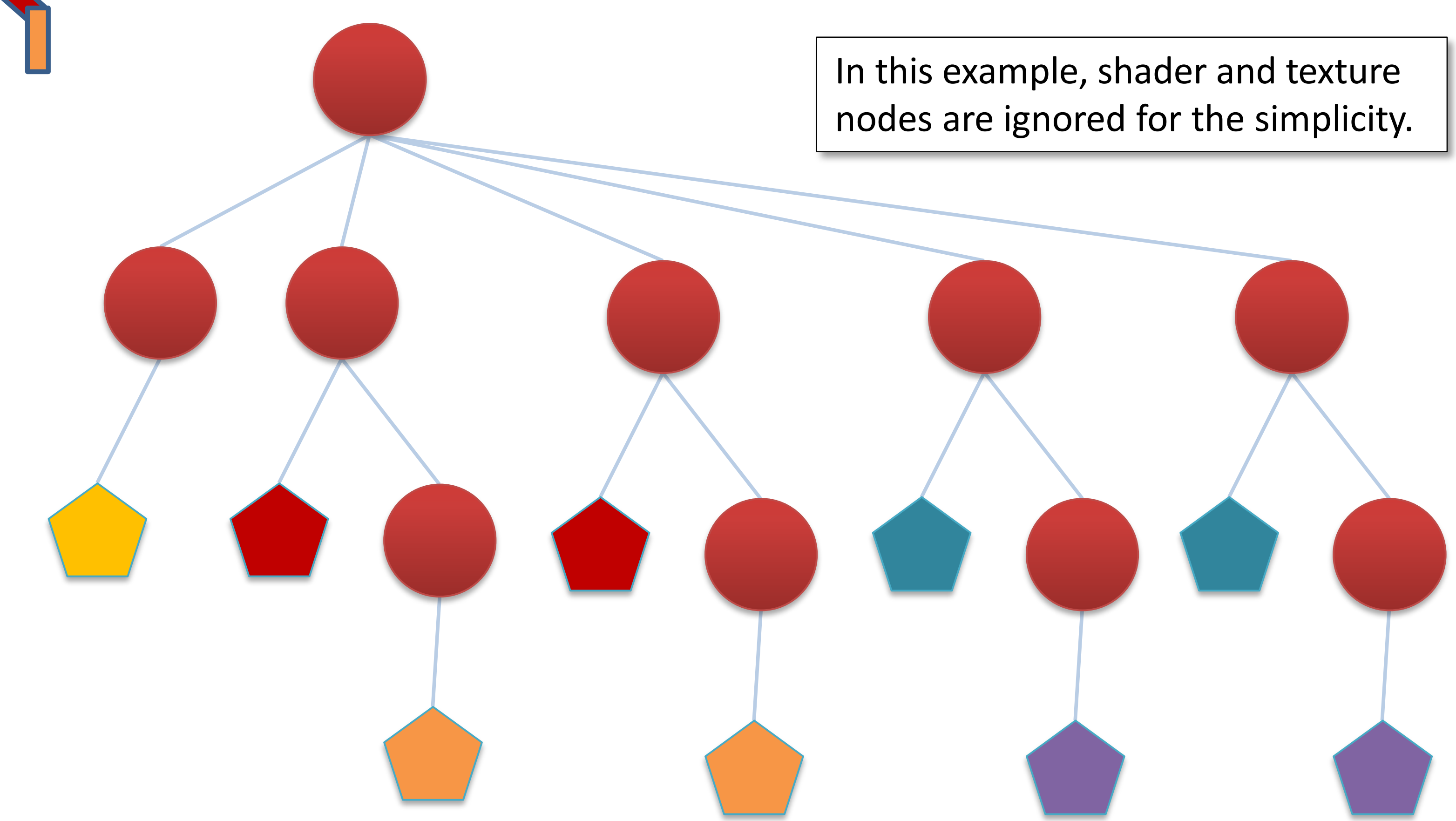
Select rendering path according to the current scene configuration, and hide other children (or sub-graph). Used for level-of-detail rendering.

Instancing – Data Reuse

- **A node (or sub-graph) can have multiple parent nodes, which enables instancing of the node.**
  - Using a single mesh data with multiple transform (group) nodes, complex scene can be easily generated.

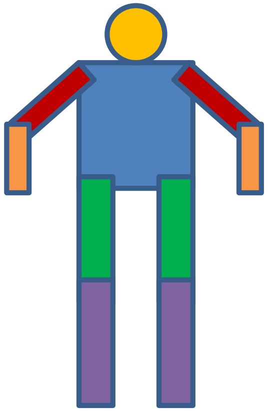
Instancing Example

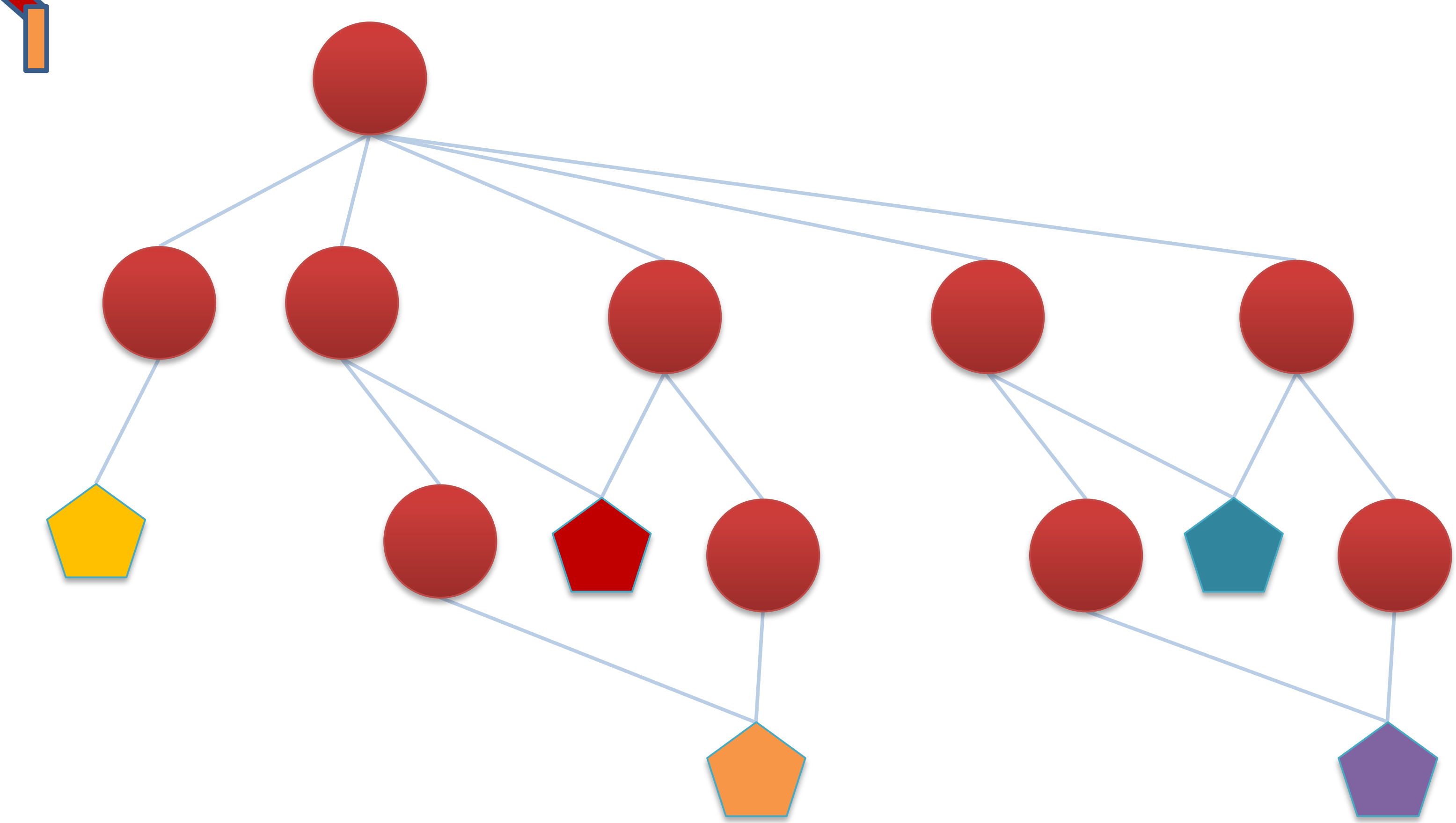




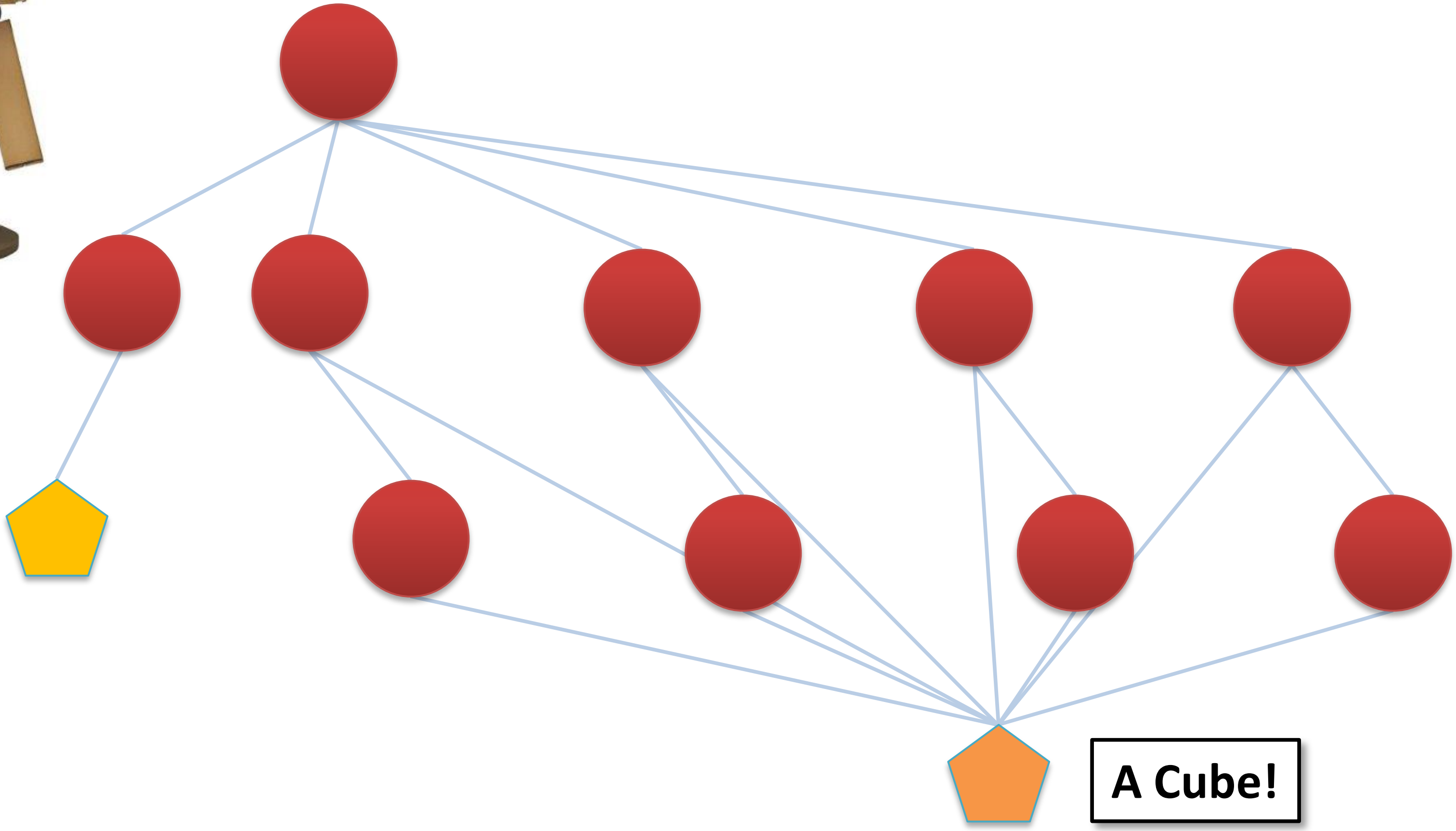
In this example, shader and texture nodes are ignored for the simplicity.

If our robot is symmetric…





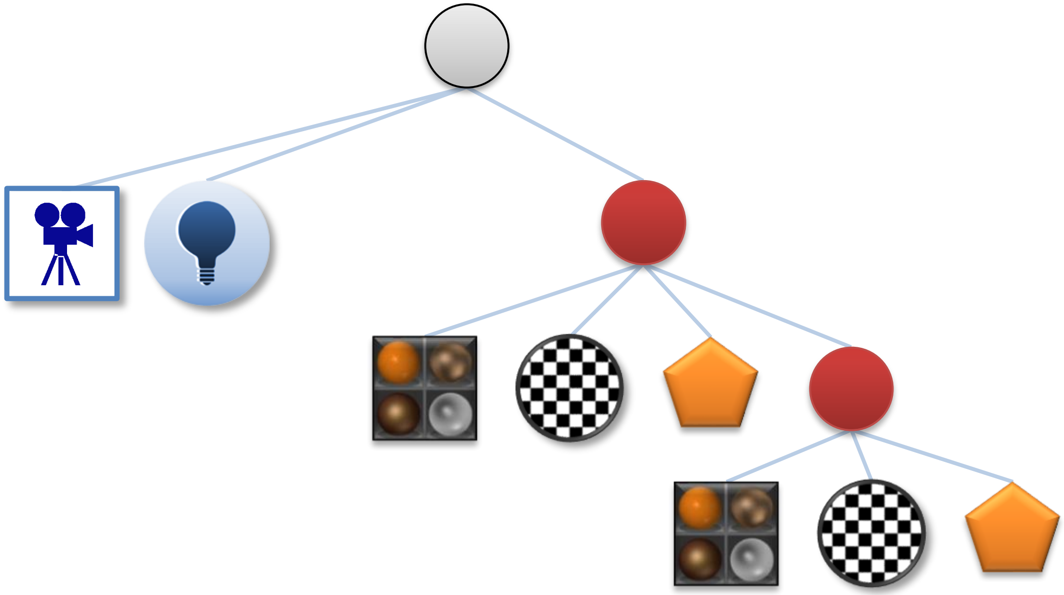
If our robot is made out of box...



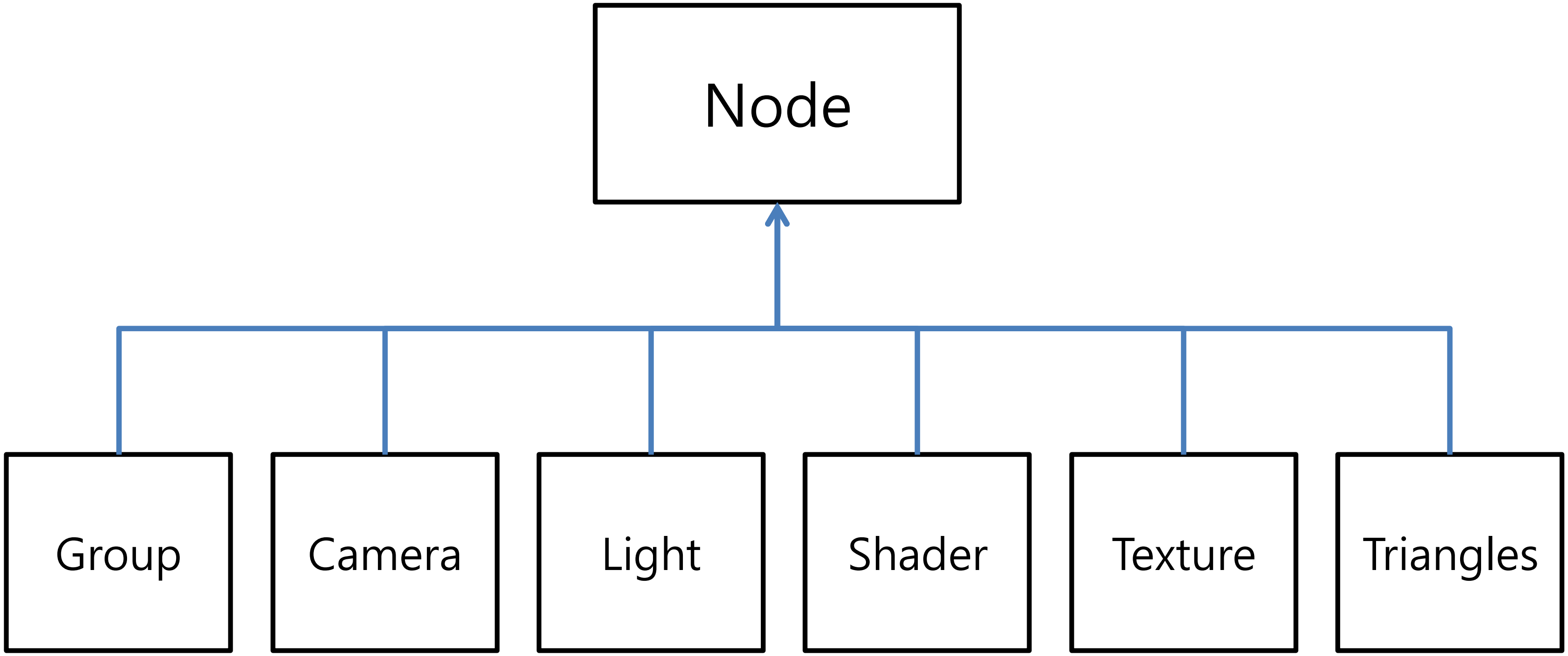
A Cube!

Implementing the Scene Graph

- **Just same as implementing tree or graph structure**
  - Scene graph can be easily implemented in the OOP fashion, such that the graph traversal can be done very straightforwardly.
  - When graph traversal policy is defined, simply change the OpenGL state under traversal order.



Simple Class Hierarchy



How to Define 'Node'

```
class Node {
  virtual void glRender() = 0;
}
```

glRender() is a function, which will be invoked when we visit this node during the graph traversal.
glRender() defines what should be done when that node is visited.
Everything related to the OpenGL goes here.

How to Define ‘Group’

```
class Group : public Node {
  void glRender();
  void addChild(Node* n);
  vector<Node*> children;

  float trans[3];
  float angle, axis[3];
  float scale[3];
}
```

```
void Group::glRender() {
    glPushMatrix();
    glPushAttrib(…);
    glTranslate(trans);
    glRotate(angle, axis);
    glScale(scale);

    for each child i
        children[i]->glRender();

    glPopAttrib(…);
    glPopMatrix();
}
```

How to Define 'Camera'

```
class Camera : public Node {
  void glRender();

  float position[3];
  float lookat[3], up[3];
  float viewAngle;
  float zNear, zFar;
}
```

How do we know width and height of current viewport outside of the resize function?

```
void Camera::glRender() {
    int w = getViewPortWidth();
    int h = getViewPortHeight();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(…);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(…);
}
```

Retrieving Current Viewport State

```
GLint viewport[4];
glGetIntergerv(GL_VIEWPORT, viewport);
viewport[0]; // x
viewport[1]; // y
viewport[2]; // width
viewport[3]; // height

float aspectRatio = viewport[2]/float(viewport[3]);
```

Modified Resize Callback Function

```
void resize(GLint w, GLint h) {
   glViewport(0,0,w,h);
   // no matrix manipulation in here
}
```

How to Define ‘Light’

```
class Light: public Node {
  void glRender();

  float ambient[4];
  float diffuse[4];
  float specular[4];
  float position[4];
}
```

```
void Light::glRender() {
    glLight(GL_LIGHT0, GL_AMBIENT, ambient);
    glLight(GL_LIGHT0, GL_DIFFUSE, ambient);
    glLight(GL_LIGHT0, GL_SPECULAR, ambient);
    glLight(GL_LIGHT0, GL_POSITION, position);
}
```

How to Define ‘Shader’

```
class Shader: public Node {
  void glRender();

  float ambient[4];
  float diffuse[4];
  float specular[4];
  float shininess;
}
```

How to Define ‘Texture’

```
class Texture: public Node {
  void glRender();

  GLuint texID;
}
```

```
void Texture::glRender() {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texID);
}
```

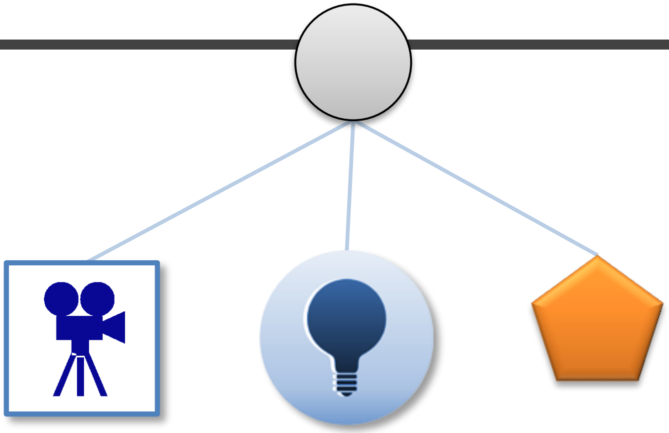
How to Define ‘Triangles’

```
class Triangles : public Node {
  void glRender();

  size_t nTriangles;
  float* vertices;
  float* normals;
  float* texCoords;
}
```

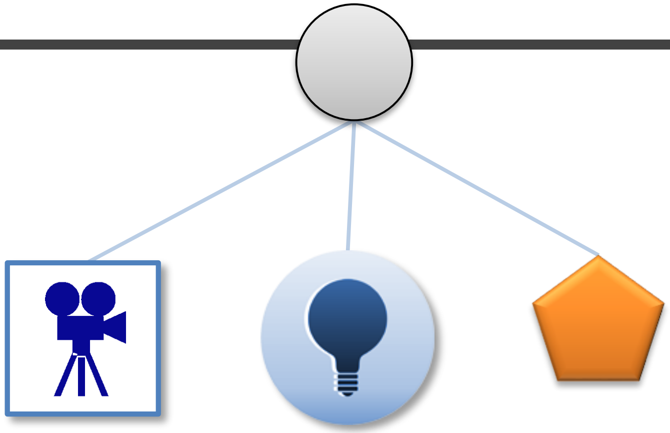
```
void Triangles::glRender() {
  for each triangle i
    glTexCoords(texCoords[2*i], texCoords[2*i+1]);
    glNormal(vertices[3*i], vertices[3*i+1],vertices[3*i+2]);
    glVertex(vertices[3*i], vertices[3*i+1],vertices[3*i+2]);
}
```

Usage of Scene Graph - init



```
Group root;
int init(…) {
   Camera* camNode = new Camera(…);
   Light* lightNode = new Light(…);
   Triangles* triNode = new Triangles(…);
   root.addChild(camNode);
   root.addChild(lightNode);
   root.addChild(triNode);
}
```

Usage of Scene Graph - display



```
void display() {
   glClear(…);
   root.glRender();
   glFlush();
   glXSwapBuffers();
}
```