# Scene Graphs

## 1 What are they?

Scene Graphs are a neat way to represent complex objects which have to obey certain constraints (if you have objects consisting of rigid pieces with joints or a complex object whose dynamics satisfies certain constraints then you will probably want to think about it in terms of scene graphs). The representation is by a way of an acyclic graph. Recall that an acyclic graph is a directed graph (thus, having vertices and edges, arrows joining certain pairs of vertices) which has no closed cycles. Trees are a very common form of an acyclic graph.

The vertices of a scene graph contain pieces of an object to be drawn (think of them as a bunch of triangles). The piece can possibly be empty (no triangles at a node). Each edge has a transformation associated with it. There is one special node called the root, which will be the one we start drawing things from.

To draw a scene graph, we will traverse it starting from the root node. First, the piece stored at the root will be drawn. Then, for each edge out of the root node, we will do the following:

**1.** Save the current modelview matrix by pushing it onto the matrix stack.

**2.** Multiply the modelview matrix on the right by the transformation associated with the edge.

**3.** Call the drawing procedure recursively, pretending that the endpoint of the edge is the root.

**4.** Restore the original modelviev matrix, by popping it from the matrix stack.

## 2 Illuminating Examples

We'll explain by giving two 2D examples. It should be clear how to carry things over to 3D, so we'll stick to the 2D case here.

### 2.1 Clock

Our clock will have a dial and two hands, which we would like to place so that they start at the center of the dial (this is the only constraint – we won't care that the long one usually moves some 12 times faster than the short one). Assume that we have a procedure which draws the dial (so that it is centered at the origin) and the two hands so that their starting point is the origin (Figure 1). Now, to draw the hands so that they always begin at the center of the dial we need to first apply a rotation transformation to each of them and then apply the transformation which needs to be applied to the whole clock. If $T$ is that transformation then $T \circ R_\alpha$ and $T \circ R_\beta$ are the transformations we need to apply to the hands. Before drawing the dial, $T$ has to be applied.

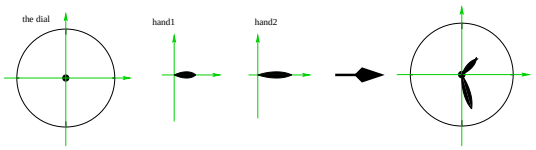
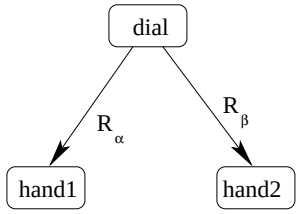



Figure 1: Components of the clock and all of them put together

There are several possible ways to do that: the trivial one would be to compute the matrix to be applied and use it as the modelview matrix before each of the three pieces is drawn. However, this would burden the programmer with the complicated task of keeping track of all the matrices. Because of this, OpenGL and various rendering and modeling software packages often contain some support for scene graphs. In OpenGL, we have the matrix stack and various operations allowing to modify the current modelview (or projection, but this is not done too often) matrix, most of them allowing to multiply it by some simple transformation on the right. Together with the matrix stack, this makes it very easy to convert a scene graph into OpenGL code. An example scene graph and its corresponding OpenGL code is shown in Figure 2.




```
draw_the_dial();
glPushMatrix();
    glRotate(by α  around the origin);
    draw_hand1();
glPopMatrix();
glPushMatrix();
   glRotate(by β  around the origin);
   draw_hand2();
glPopMatrix();
```

Figure 2: A scene graph for the clock and the corresponding OpenGL pseudocode.

## 2.2 Robot arm

It's going to consist of two hands, one forearm and one upper arm. Clearly, the whole arm always needs to be drawn in a consistent way (the pieces have to fit together - see Figure 3).

The corresponding scene graph and OpenGL code are shown in Figure 4. The angles $\alpha$, $\beta$ and $\gamma$ are parameters describing the relative location/orientation of the pieces (try to state their precise meaning...).

An alternative scene graph representing the same robot arm is shown in Figure 5. Notice that it leads to exactly the same OpenGL pseudocode as the one in Figure 4. I put it here to show you that a scene graph doesn't need to be a tree in order to make sense. If a lot of pieces repeat, it may be easier and more economical to represent them with just one node.

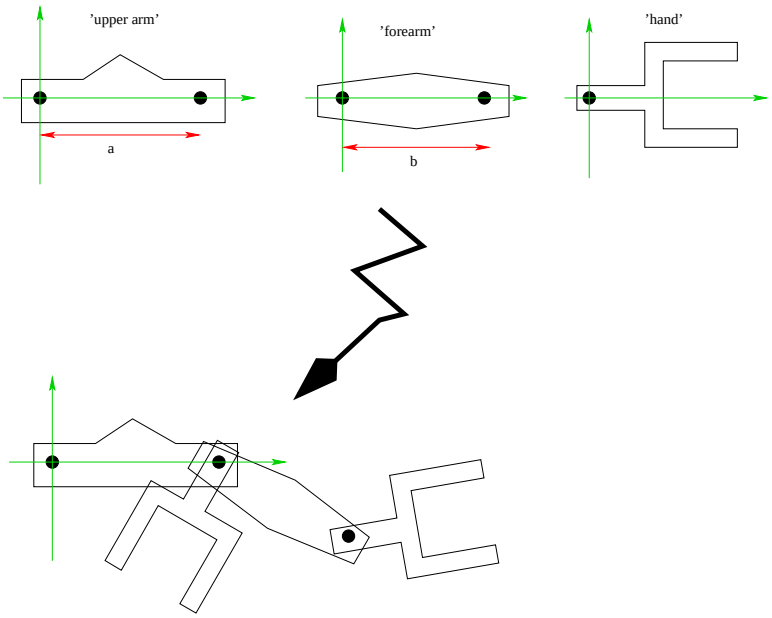
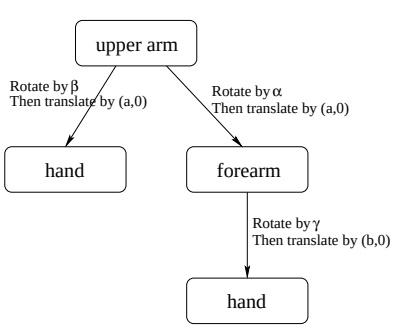



Figure 3: Components of the arm and all of them put together




```
draw_upper_arm();
glPushMatrix();
    glTranslate(a,0);
    glRotate(by α  around the origin);
    draw_hand();
glPopMatrix();
glPushMatrix();
    glTranslate(a,0);
    glRotate(by β  around the origin);
    draw_forearm();
    glPushMatrix();
        glTranslate(b,0);
        glRotate(by   γ around the origin);
        draw_hand();
   glPopMatrix();
glPopMatrix();
```

Figure 4: The scene graph for the robot arm and an OpenGL pseudocode drawing it

# 3 Merging scene graphs

Another nice thing about scene graphs is that they can be merged together to merge complex objects. For example, if we need to put a clock into one of the hands of the, the scene graph in Figure 6 does the job..

Notice that the scene graph above results in the clock drawn so that it is gripped at noon and 6 ticks. To make it possible for it to be drawn in another

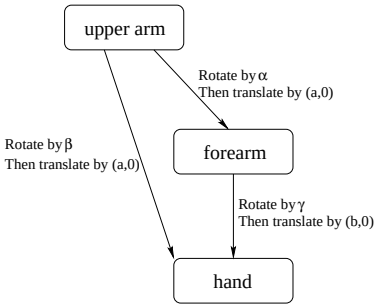



Figure 5: An alternative for 4

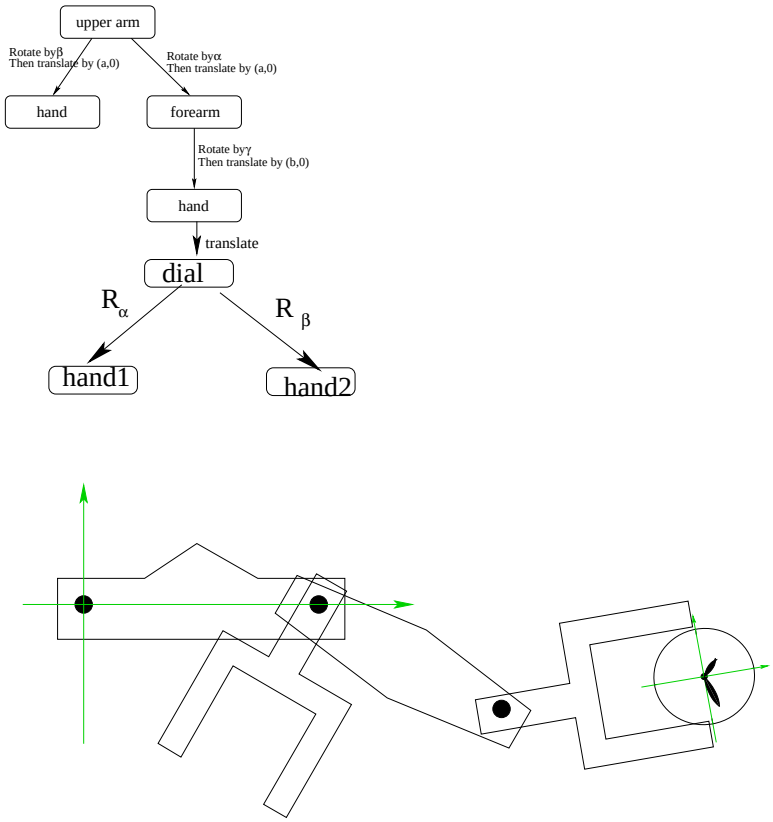



Figure 6: Components of the arm and all of them put together

orientation, we can associate a composition of a rotation and translation around the origin with the arrow joining the hand and the dial rather than a translation..