CS 370 - Assignment 4

1. This is the subject of *key framing* in animation where the initial and final orientations of a model are given and the rendering algorithm must create intermediate frames to produce a smooth transition. It also occurs in robotics as inverse kinematics, where the end of the arm must go from one point to another to accomplish a task. If we let the various components have sizes and orientations given by:

Base - height h_1, rotation θ
Lower arm - height h_2, width w_2, rotation ϕ
Upper arm - height h_3, width w_3, rotation ψ

Therefore the final transformation for points on the upper arm is given by:

$$\mathbf{M} = \mathbf{R}_y(\theta)\mathbf{T}(0,h_1,0)\mathbf{R}_z(\varphi)\mathbf{T}(\pm(w_2+w_3)/2,h_2,0)\mathbf{R}_z(\psi)$$

One solution to the problem is to find joint angles corresponding to the two desired points (note however that these sets of angles are often *not* unique.) Then the angles can be linearly parameterized by

$$\theta = (1-\alpha)\theta_i + \alpha\theta_f$$
$$\phi = (1-\alpha)\phi_i + \alpha\phi_f$$
$$\psi = (1-\alpha)\psi_i + \alpha\psi_f$$

where $\{\theta_i,\phi_i,\psi_i\}$ are the initial angles and $\{\theta_f,\phi_f,\psi_f\}$ are the final angles. As the parameter varies $0 \le \alpha \le 1$, the joint angles will cause the tip to go from the initial point to the final point.

Unfortunately, this simple approach can often result in problems such as gimble lock (joint angles that produce an impossible physical configuration) or large rates of change in angles which produces strange animations. A common way to avoid such problems is to specify a path the end must follow (rather than simply the endpoints.) Then using a more mathematically advanced technique involving quaternions (an extension of complex numbers) for the angular changes can produce much more acceptable results.

2. One way to produce a scene graph that is independent of traversal algorithm, i.e. does not depend on transformation concatenations from node to node, is to store the complete transformation for each piece within the node structure. This way the node structure is simply a linked list without a child/sibling representation. As each node is traversed, its entire transformation is loaded into the ModelView matrix rather than being based on any prior transformations from previously rendered nodes. Clearly this does not take advantage of the relationships between nodes for efficient rendering.

3. Back-to-front renderings (i.e. painter's algorithm) ensure that the object appearing in the scene is the one (or part of one) closest to the viewer. However it involves rendering *all* the objects in the scene which may be inefficient if a majority of them are occluded by the closest object. Front-to-back rendering, particularly for ray tracing schemes, would be more efficient as rendering can stop as soon as the first opaque object is encountered (since this is the termination point of the ray.) It also allows for proper color blending for scenes consisting of primarily translucent objects (using $\alpha_s, 1-\alpha_s$ blending factors.)