Hierarchical Modeling

We implemented hierarchical modeling by creating a scene graph that can organize affine transformations (translation, rotation, and scaling) and color changes between objects and groups of objects using a tree structure. We created this tree structure by designating certain surfaces as root nodes, allowing each node to have a list of children, and keeping track of the stack of transformations with a list of matrices. The new parameters we added to the Scene struct in scene.h are:

- vector<mat4f> xformlist // stack of matrices in order of root to leaf node
- vector<int> rootlist // list of root surface indices in scene->surfaces

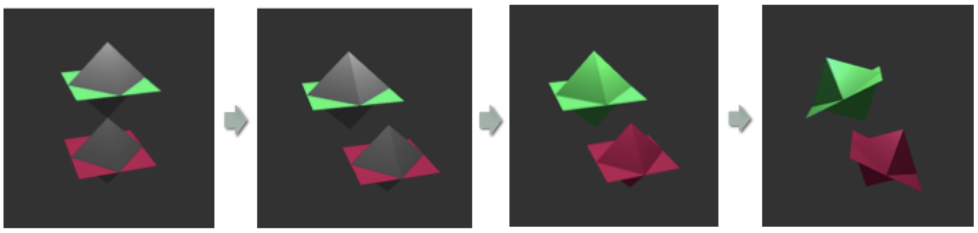
There are several new parameters we added to the Surface struct in scene.h in order to implement the scene graph tree structure. These are:

- vec3f tform // vec3f of translation in (x,y,z) direction
- vec3f rform // vec3f of rotation in (x,y,z) direction
- vec3f sform // vec3f of scaling in (x,y,z) direction
- bool isRoot = false // whether surface is a root
- bool applyColor = false // whether to apply color to children
- bool hasChildren = true // whether surface has any children
- vector<int> children // list of childrens' indices in scene->surfaces
- frame3f parentFrame // keeps track of parent surface's frame, if any

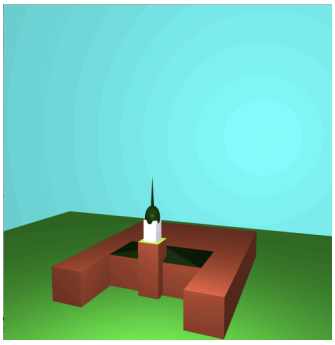
We created a recursive function traverse(Surface* surface) that parses the tree structure defined in the json file; in the code, it is located in between the subdivide_surface(Surface* surface) and subdivide(Scene* scene) methods in 02_model.cpp from assignment02. The overall structure of traverse comes from the pseudo-code provided in Shirley 12.2, reproduced below:

```
function traverse(node)
    push(M_local)
    apply composite matrix from stack
    traverse(left child)
    traverse(right child)
    pop()
```

Our actual implementation was a little different from the pseudo-code. Instead of following a binary tree structure where each node can only have two children, our implementation allows virtually unlimited numbers of children (only constrained by memory capabilities of the machine). In addition, the order in which we applied transformations and called traverse on children had to be changed for the transformations to be applied correctly; in particular, we had to reset the current surface's transform matrix before concatenating parent transforms onto it.



In the image above, there are two root nodes (green quad and red quad) and two child nodes (un-subdivided spheres). Translations of the root nodes also translate the child nodes, as seen in the second image. In the third and fourth image, we see that color and rotation transformations of parents also apply to children.



By using hierarchical modeling, we're able to create some relatively complex scenes with an extremely limited toolset of shapes. In these models of Baker Bell Tower, we used only quads and spheres to create the various structures.

For the model on the left, the green roof of the center building was created with an un-subdivided sphere, as was the spike at the top of the bell tower. The green dome was created with a scaled, rotated, translated, and subdivided sphere at subdivision_level=3.

In the model on the right, part of the bell tower is scaled to two times its original height, rotated about the y and z axis, and translated to demonstrate how affine transformations of parent nodes apply to children. The other parts of the model are not affected because they are not children of the bell tower node.

The transverse method is called inside subdivide(Surface* surface).

Hierarchical modeling run command example : once in hellokitty/scenes, run:

../bin/mk/02_model tester.json	for test demonstration
../bin/mk/02_model belltower.json	for bell tower
../bin/mk/02_model belltower_wonky.json	for wonky bell tower