Algorithm 5 Sceneview

Introduction to Computer Graphics, Fall 2015

Solution Key

This assignment is worth 6% of your Sceneview grade.

1. Consider the following excerpt from a scenefile:

```xml
<transblock>
  <scale x=".05" y="1.0" z=".05"/>
  <translate x="0" y=".5" z="0"/>
  <rotate x="1" y="0" z="0" angle="45"/>
  <object type="primitive" name="cylinder">
    <diffuse r="1" g="1" b="1"/>
  </object>
</transblock>
```

[1 point] To transform the cylinder $C$ into the desired cylinder $C'$, in which order would you multiply the three transformations: translate ($T$), rotate ($R$), and scale ($S$) for your Sceneview project? You should consult the Sceneview project handout and the scene file reference on the course website.

Since matrix multiplication goes right-to-left, and the Sceneview specs and scene file reference tell us to multiply the transforms in the order they are given, we get

$$C' = S \times T \times R \times C$$

[1 point] Describe each step in the multiplication you gave above. Is the order you described above how you would want to organize transforms in a scenefile? Propose a better (more logical) order for how you would generally order transformations in a trans block.

First, the cylinder is rotated 45 degrees along the x-axis, rotating around its center. Then the cylinder is translated 0.5 units along the y-axis of the world (so the rotated cylinder is shifted up in the world rather than following along the y-axis of the cylinder itself). Finally, the object is scaled to 5% its original size in the x and z directions. However, since the cylinder was translated away from the origin, the cylinder will not scale uniformly and will move in space and end up skewed after scaling.

This order makes it hard to visualize the scaling ahead of time and leads to confusion. A more logical order would be to scale first, then rotate, and finally translate the object like so:

$$C' = T \times R \times S \times C$$

This way, everything is consistently transformed and the axes are stay aligned. Remember though, your project should compose transforms in the order given. It is up to the scenefile creator to use a logical ordering.

2. When coding Sceneview, you will have to compose transformations whenever there is an object tree block contained within a trans block. Consider the following contrived excerpt from a scenefile:

```xml
<transblock>
  <rotate x="0" y="1" z="0" angle="60"/>
  <scale x=".5" y=".5" z=".5"/>
  <object type="tree">
    <transblock>
      <translate x="0" y="2" z="0"/>
      <scale x="1" y=".5" z="1"/>
      <object type="primitive" name="sphere">
        <diffuse r="1" g="1" b="0"/>
      </object>
    </transblock>
  </object>
</transblock>
```

[1 point] Suppose you composed the two transformations in the outer trans block (the rotate and scale), calling the resultant matrix $CTM_1$, and then composed the transformations in the inner trans block (the translate and scale), calling the result $CTM_2$. In what order must you multiply these matrices to obtain a single composite matrix with the desired effect on the primitive sphere object?

```
COMPOSITE = CTM1 * CTM2
```
3. Being sure of the order in which matrices must be multiplied puts you well on your way to completing Sceneview. The other major hurdle is deciding how you will efficiently traverse the objects from the parse tree provided by CS123ISceneParser.

a. [1 point] When drawing a static scene, is there a more efficient way to draw objects than traversing scenegraph every time? What feature(s) of a scenegraph would make it slow to traverse multiple times for drawing?

Flattening the parse tree makes it quicker and easier to traverse when drawing the scene. Using the original parse tree every time the scene is drawn would be inefficient because all nodes would have to be traversed, including the ones that contain no primitives, and we would have to compose matrices every traversal even though they don't change.

b. [1 point] What type of data structure will you use for drawing that can store your more efficient version of the tree?

You want to use a list structure of some sort. Good answers include data structures that are easy to iterate through. Array, linked list, std::list, std::vector, QList, etc. are all good choices. Poor choices include maps, sets, hash maps, and trees. These are more costly to build and more costly to iterate through.

c. [1 point] What information will you store at each of the nodes in your data structure? Please give valid types and descriptions of any types you are defining yourself.

The necessary information: the composite transformation matrix, the color, and the type of object to be drawn. You should not store a full instance of the object primitive for each leaf node – see the flyweight pattern.