

CS 770/870

Assignment 4: Hierarchical Composite Object Creation Using a Scene Graph

October 15, 2008

Due: Monday, 10/27 by 23:59:59

Last allowed submission: 10/31 23:59:59

The purpose of this assignment is to give you practice combining all that you've learned so far in the context of an object-oriented scene graph. For this assignment, you must implement a scene-editing program that has the following features:

1. **Transformable 3D shapes.** It supports at least 3 classes representing simple shapes, possibly the same as those implemented for Assignment 3. Each instance of each shape must have its own unique name, and must support the following commands (each command is relative to the *current* coordinate frame: either that of the object's parent, or if the object has no parent, then it is relative to world coordinates).

- **Rotate to** *deg* degrees from the original orientation, along the axis (*ax*, *ay*, *az*) in the current coordinate frame. Rotations should default to causing the object to spin about its center. (Syntax: **rotate deg ax ay az**)
- **Translate to** (*tx*, *ty*, *tz*) in the current coordinate frame. Translations should default to specifying the new location of the object's center. (Syntax: **translate tx ty tz**)
- **Scale to** a scale factor of either *s* or (*sx*, *sy*, *sz*) from the original size in the current coordinate frame. A scale should apply in the reference frame of the object—horizontal, vertical, etc. with respect to the canonical orientation of the object, as opposed to the orientation it has after rotation. (Syntax: **scale {s | sx sy sz}**)
- **Create a copy of itself** called *new_shape_name*. The new shape should have all the same values as the original, including rotation, translation, and scale properties. (Syntax: **copy new_shape_name**)

The results of the commands should be visible on the next refresh cycle.

2. **Composite 3D shapes.** It supports the creation of composite objects by creating hierarchies of simple and composite shapes. Your implementation should include a new shape class for composite objects that supports all the commands above, as well as the following commands.

- **Add shape** *shape_name* as a child to this shape. The child shape should now render itself using this shape's coordinate frame as the current one. (Syntax: **add shape_name**)
- **Create a deep copy of itself** called *new_shape_name*. The copy includes not only the top composite object, but all children, grandchildren, etc. should be copied with their current names as children, grandchildren, etc. of *new_shape_name*. (Syntax: **copy new_shape_name**)

3. **Control window(s).** You must have at least one way of issuing commands to shapes. You can use the `CommandPanel` class and the syntax specified with each command, or a

ControlPanel with a selection box and sliders (as in Assignment 2), or both. In addition to the commands given to specific objects, you must also support the following global commands.

- **Tell shape** *shape_name* to perform a supported *command* (such as those specified above).
(Syntax: **tell** *shape_name* *command*..)
- **Add a new shape** *shape_type* to the scene with the name *shape_name*. The *shape_type* can be either a basic shape or a composite shape. While you can name your shapes and composites anything you like, you must also support shape-type names “s1”, “s2”, “s3”, and “composite” so that we can create files that will load into anyone’s program. For example, if you have shapes named “carpet”, “house”, and “pacman”, then we should be able to create a carpet using a shape-type of “s1”, a house using “s2”, a pacman using “s3”, and a composite object using “composite”.
(Syntax: **new** *shape_type* *shape_name*)
- **Save the scene to** *file_name*. Saves all shapes to the file *file_name* and closes the file. (Syntax: **save** *file_name*)
- **Save one or more shapes to** *file_name*, listed by the shapes’ names and closes the file.
(Syntax: **save** *file_name* *sname_1* [*sname_2*] ... [*sname_n*])
- **Load** *file_name*. Loads all shapes from the file *file_name* and closes the file.
(Syntax: **load** *file_name*)

4. **Viewing.** You must use a reasonable perspective view, and you must provide GLUI controls for scaling the scene and rotating it about the vertical axis. You can (and should) limit the amount the user can scale the scene, as long as it is still possible to zoom out to see a fairly large scene.

5. **File Input.** You should be able to read a file containing commands from our simple command language for defining a scene of objects. The file name should be specified on the command line. All of the global commands are required except for `save`. Comments can be included and should be used in files you create. They start with ‘#’ and continue to the end of the line. Blank lines should be allowed and used in files you create, as well.

6. **Predefined scene.** If no input file is specified, or if the specified file name does not exist, the program should invoke code that displays a predefined scene that contains at least 2 different-looking instances of one composite shape and one instance each of two other composite shapes. You must use at least one `load` command to pull in a composite shape definition from another file. It is ok to add new syntax to make your files briefer (such as defining a new command to represent a series of commands that apply to the same object) as long as you still support the required syntax.

Point allocation

25 Predefined scene visible in a perspective view with the ability to rotate and scale the scene from the gui. Scene contains 2 instances of a composite shape and one instance each of two other composite shapes.

- 15 The 2 instances of one composite in the default scene are different in some substantial way: a part of it (child object) is rotated, translated, and/or scaled.
- 10 There is a way to rotate, translate, and scale shapes from the gui or command line, and they move relative to the proper reference frame.
- 5 There is a way to add new shapes to the scene from the gui or command line.
- 15 There is a way to add basic shapes and copies of shapes to composite shapes from the gui or command line.
- 10 New composite shapes can be saved (using `save` with a filename and an individual shape's name), the program can be quit and restarted, and the saved shape reloaded successfully (using `load`).
- 10 New scenes containing composite shapes can be saved (using `save` and a filenames), the program quit and restarted, and all the shapes in the scene reloaded successfully (using `load`).

Features that might qualify for the last 10 points include but are not limited to the following.

- Implement a way to delete shapes and move them around in a composite shape's hierarchy.
- Implement a way to specify an origin for shape rotation, translation, and scale other than the shape's center, but have it default to being at the shape's center.
- Implement commands to change "properties" of a shape. For example, the command "`color 1.0 0.0 0.0`" might specify that the main color for a shape should be read, and perhaps the same for all its children.
- Implement a command to add properties to composite shapes. For example, the command "`addProperty mainColor color myChild1 myChild2`" might specify that a composite shape should have a new property called "mainColor", and that any changes to mainColor should propagate to the color property for the child shapes myChild1 and myChild2, but not to any other child shapes.

Note:

Points are earned by correctly implementing features robustly. Points are deducted for bugs, incorrect implementation, poor style, poor design decisions, etc.

Command syntax summary

```

new shape_type shape_name
save file_name
save file_name sname_1 [sname_2] ... [sname_n]
load file_name
tell shape_name command ...
  where command may be one of the following for any shape:
  rotate deg ax ay az
  translate tx ty tz
  scale {s | sx sy sz}
  copy new_shape_name
  and in addition command for a composite may also be:
  add shape_name

```