CS175 Final Project Writeup
Christian Scarlett
HUID: 41397128

A Simple 3D Modeling Program

Project Description

This project is an implementation of a simple program that allows the user to create compound 3D models out of primitive shapes. The program is inspired by Tinkercad, an online CAD program. The program allows the user to spawn in simple primitive shapes and "group" them together to create more complex models. The motivation is to enable creation of complex models by conglomerating simpler objects, and to introduce a robust interface for altering a model and its component object hierarchy.

Note: "nodes" and "objects" are interchangeable terms in this writeup. "Selecting" and "picking" are different concepts - selecting means an object is added to a queue to be grouped or ungrouped, and picking means an object is chosen for manipulation using the arcball interface.

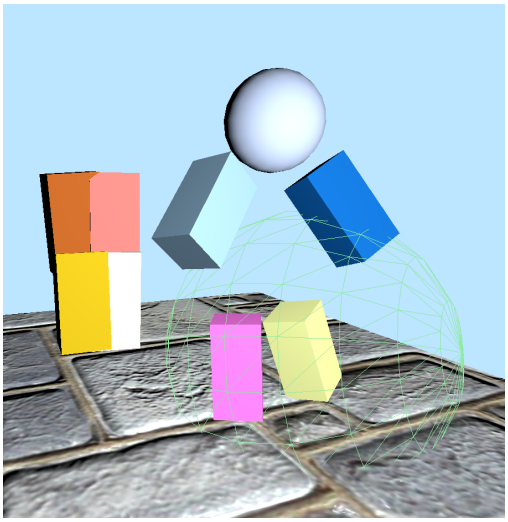
Using the Program

The program is controlled using the mouse and keyboard. Here are descriptions of each keyboard key's function:

- E, R, T, Y - Spawn in a new primitive shape (cube, sphere, rectangle, oval, respectively)
- O - Toggle Parent Picking mode. With Parent Picking deactivated, picking works as normal. With Parent Picking activated, picking an object will pick the group with the highest hierarchy that it's a part of. More on what this means later.
- P - Picking as implemented in previous homeworks. Pressing P toggles picking mode. While in picking mode, left clicking on an object in the scene will select that object, or if in Parent Picking mode, its highest group.
- J - Enter Select mode. If not picking, clicking on an object adds it to a group of "selected" objects.
- G - While in Select mode, pressing G will group the objects that are currently selected. Now, when those objects are picked, they will move together as one, new object. Select mode will also deactivate.
- F - While in Select mode, pressing F will ungroup all objects that are currently selected. Each ungrouped object will be decomposed into its component objects. Select mode will also deactivate.

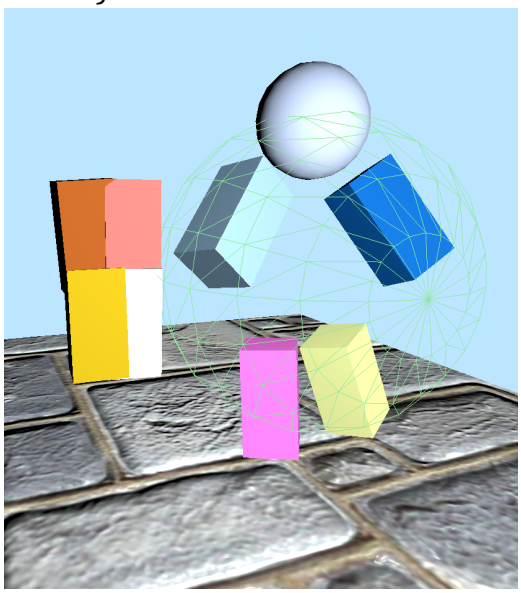
Here is a walkthrough of how this program might be used:

1. Program is opened. The user presses T to spawn some rectangles and R to spawn a sphere. By pressing P to pick each object, the user is able to arrange the spawned objects like so:



As seen in the photo, the user is trying to create a person. They've arranged some rectangles to shape the torso, and some other shapes to make the arms and legs. The yellow leg is picked right now because no objects have been grouped yet.

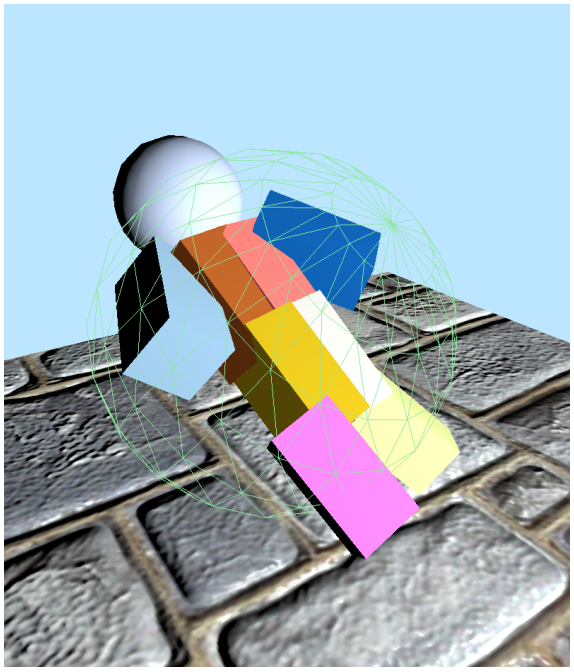
2. The user wants to move the limbs and head of the person onto the body, but they don't want to move each individual piece of the body or the limbs. So, they press J to enter select mode, click on each body rectangle, and press G to group them. They do the same to the limbs and head.



Shown in the picture, the head and limbs are grouped and picked. Because Parent Picking is enabled, picking any component object of a group will pick the entire group. Note that the group object's center is the average of the position of the component objects', and thus is where the arcball is drawn.

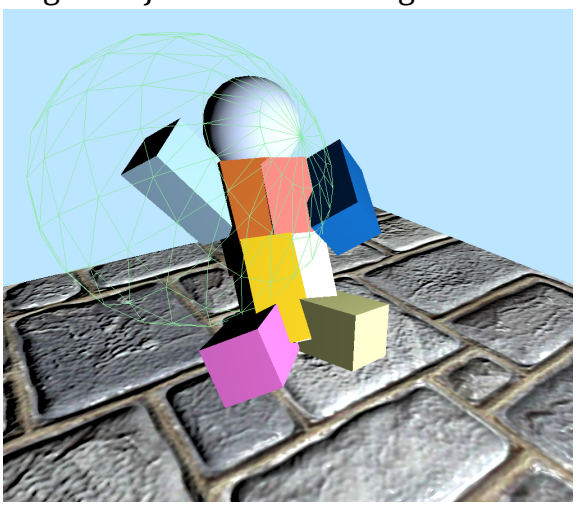
3. Then, the user drags the limbs onto the torso. Again, they press J to enter Select mode, click on a limb to select the limb group, and a body rectangle

to select the body group, and press G to group them. The person is now complete! The user rotates the entire model:



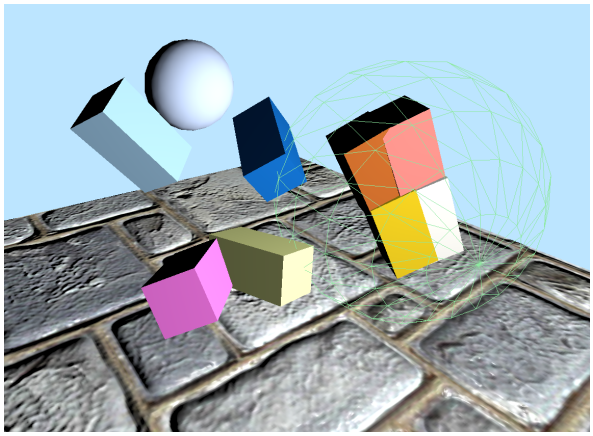
Shown is the entire grouped model, picked and rotated.

4. The user likes their person model, but they want the model to sit and wave to the user. So, the user presses O to deactivate Parent Picking mode and picks each arm and leg to adjust to their liking.



Parent picking mode is disabled, so the user can select each primitive object (the light blue arm primitive is selected in the photo).

5. The user is not satisfied with their torso object and wants to pull it aside to make some changes. So, they press O to toggle parent picking back on, press J to select the entire person model, and press F to ungroup the person. The person is now decomposed into its component objects, the torso and the limbs. The user then presses P to pick the body and moves it aside.



Design: Builder Class

The design for implementing this program is based around a Builder class. This class exposes methods for spawning, grouping, and ungrouping objects in the context of the program. The most important methods are listed here:

- addObject(ShapeGeo shape) - create a new object with shape `shape` and add it to the scene graph. Note this class also defines a few valid "shapes" within the context of the builder. These shapes include cubes, spheres, rectangles, and ovals.
- addToSelected(shared_ptr<SgRbtNode> node) - add the object (node) to the queue of selected objects.
- group() - add all objects in the selected queue to a group to create a new conglomerate object whose component parts are the selected objects.
- ungroup() - for all objects that are in the selected queue, decompose them into their component objects.

Grouping

Conceptually, objects are a hierarchy of other objects. All "parent" objects live as children of the world node in the scene graph. Parent objects have the highest hierarchy. Objects can have as children other objects/groups. A child has a lower hierarchy than its parent.

Grouping is a procedure that is as follows:

1. Create a new parent node positioned at the average position of each selected node.
2. For each selected node:
    a. Remove node from the world root node.
    b. Recontextualize its position and rotation in the context of the parent node's position. (node.RBT <- parent.RBT.inverse * child.RBT)
    c. Add node as parent node's child.
3. Add new parent as a child to the world root node.
4. Clear the queue of selected nodes (cleanup).

Ungrouping

Ungrouping is the inverse of grouping, which breaks a conglomerate object into its component parts. This is used if the user wants to correct an error or alter each individual component part of a larger hierarchical object.

Ungrouping is described like this:

1. For each selected group node, do the following steps:
2. For each component object of the group node:
    a. Remove the child from the group node.
    b. Recontextualize the child's position and rotation in the world root node environment. (child.RBT <- parent.RBT * child.RBT)
    c. Add each child as a child node of the world root node.
3. Remove the selected group node from the world.
4. Lastly, clear the queue of selected nodes (cleanup).

Parent Picking

A group of object's parent node has no bone, so it cannot be picked or selected using the traditional picking technique. Thus, we need a way of getting access to the parent node of the highest hierarchy given a child node. The Visitor interface is well suited for this.

Thus we define a new Visitor named ParentPicker who takes as input the child node that it is looking for (the child node is picked using the traditional picking technique). We run the ParentPicker's DFS from the world root node, maintaining a stack of node pointers. If, on a path, the DFS runs into the child node, it will set an internal state variable `parent` to the second entry in the stack and end the DFS. Then, to access the parent node that we wanted, we can call a helper method `getParent()` that returns that state variable, which is a pointer to the parent we wanted. This is correct because the invariant is maintained that all parent nodes of the highest hierarchy are children of the world root node.

Child Picking

In order to get the children of a selected group node, another Visitor is implemented. While this isn't an ideal implementation - as the children are all simply one hierarchy down, making an entire DFS traversal unnecessary - typing constraints on iterating over a SgNode's children make it difficult to verify that a node represents something that is in the context of the builder. By using a Visitor, we are able to ensure that we do not consider SgShapeNodes, which are not relevant to the grouping hierarchy.

Similarly, this Visitor, named ChildPicker, maintains a stack of nodes. On a `visit` of a SgTransformNode, if the node stack's size is 1, we know this is a child object node. The visitor appends this to a list of child nodes in its internal state. This internal state is accessed by the program after the DFS is run by calling a helper method `getChildren()` which returns the list of child nodes.

Takeaways and Key Learnings

The main takeaway from this project is that there are a lot of ways that a program like this can be implemented, and each method can come with its own constraints. For example, one method I considered was, when grouping objects, simply creating a new node in the scene graph and making its “bone” a sum of all the geometries of the grouped objects. This would make picking and selecting quite easy. However, this would make it difficult to ungroup and keep track of the hierarchical structure in general. It seemed more straightforward how to deduce the “parent” nodes for each group than it would be to keep the hierarchy and geometry separated. So, I decided to encode the hierarchy in the scene graph where each node was responsible for its own geometry, and write the necessary helpers to determine parent nodes.

Initially, I wanted to make a more feature-rich modeling software that included direct manipulation of vertices and binary operations. When experimenting early in this project, I realized that I may have to unify the type of geometry across all objects in the builder or create a separate geometry representation so I would have the necessary information to do these things. This would require a complete overhaul of a geometry framework, not to mention the implementation of the complex procedures of Constructive Solid Geometry, so I settled on what I could reasonably implement in the 2 week timeframe.

Resources Used

Previous homeworks, the C++ documentation, and various StackOverflow posts were consulted in the making of this program and are cited where necessary. The project is built on top of my implementation of Homework #9 for this class.