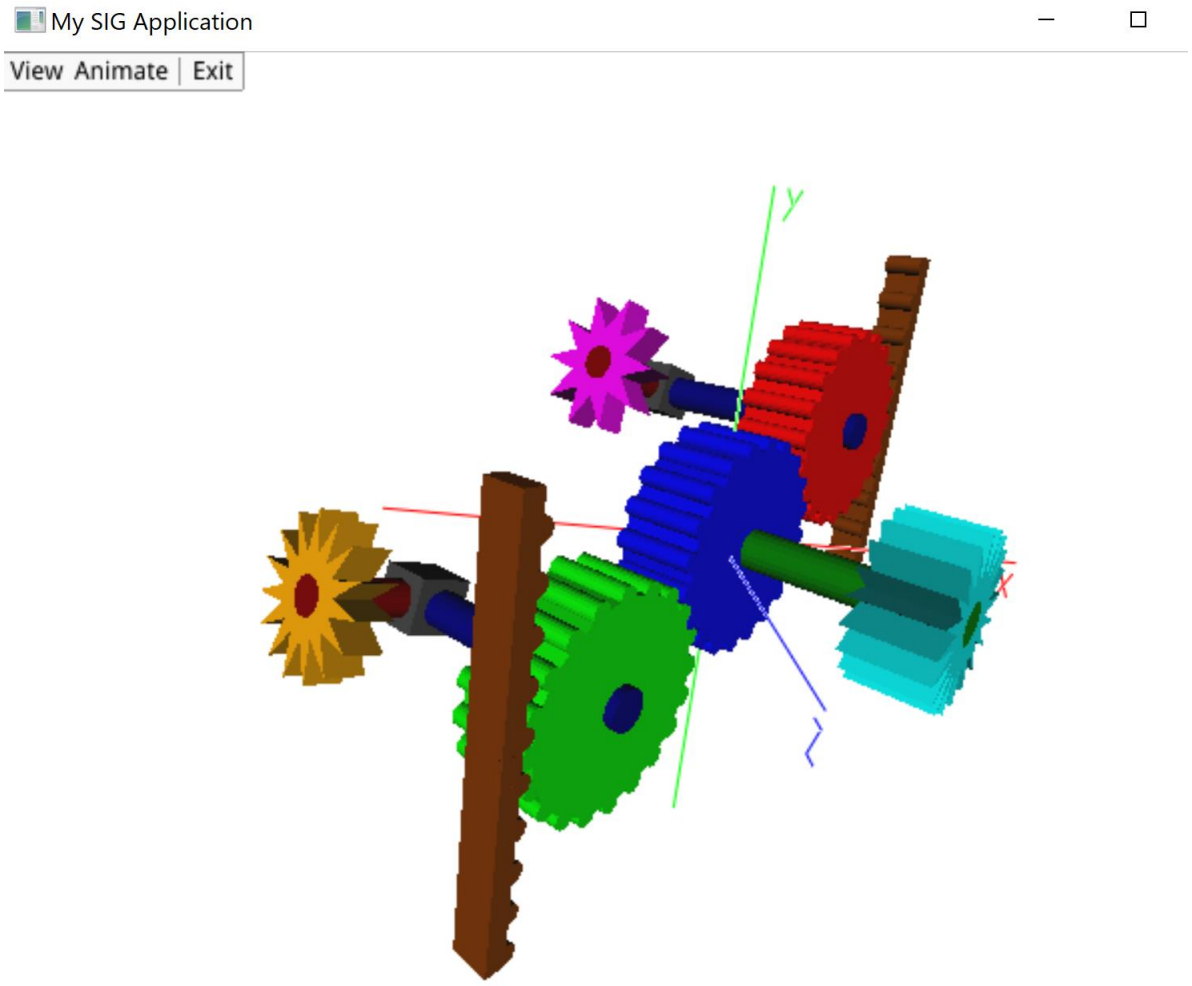
EECS 267 Lab 01 – Primitives and Meshes



Main Points

Customized 3D Spiked Gear

The code can be found under the function:

```
void make_custom_gear(GsModel* m, float r1, float d, int nfaces);
```

In this Lab, I decided to first make a 3D gear that had spiked pyramids as edges and a hole in the middle. This actually was the longest part of the whole lab. The basic idea was to create two circles using trig and geometry (an inner circle and an outer circle, to form a hole in the middle). Then connect each of the circle's outer points to each other and form a series of triangles. Then I went ahead and did this a second time but *d* distance under the first set of

circles. Now that I have created two circles with holes in the middle, I was able to connect them by finding the cross-sections between the two circles with triangles. (This is what formed the 3D shape of the base of the gear). I found the outer surface and the inner surface of these two circles. (This way there's a wall on the inside of the gear as well as the outside of the gear). After creating the base of the gear. I went ahead and drew triangles on the edges of the top and bottom circles. Then I connected the triangles to form the pyramid edges.

3D Line Gear

The code can be found under the function:

```
void make_line_gear(SnGroup *g, float length, float d, GsColor color);
```

This gear was created by making 3 separate Scene Groups. Group one consisted of a box, group two consisted of a second box right next to the first one. Group three consisted of a cylinder, on top of the second box. After making these 3 groups, I through them all under another Scene Group, "tooth". I would run through this process again and again, but each time I would displace "tooth" by 2*length (using translations) and insert it underneath the main scene group, "teeth". By the end of this process I would have a linear set of gear teeth. I made use of the make_box and make_cylinder functions.

3D Normal Gear

The code can be found under the function:

```
void make_normal_gear_base(GsModel* m, float r, float d, int nfaces);
void make_normal_gear_edges(SnGroup* g, float r, float d, GsColor color, int
nfaces);
```

To create a normal 3D gear, I made use of the make_cylinder function. To make the base of the gear, I just made a cylinder. To create the teeth of the gear, I found the outer points of the circular base, and called make_cylinder on each of those points.

Connecting Everything Together

To connect every gear together, I made sure I had each object inside its own scene group. The scene group data structure made it very convenient to keep everything organized. Especially

when I had to performing rotations on objects that depended on the position/current angle of another object.

To make everything animated, I used a frames per second system. The current angle of rotation would change over time. I would retrieve the transformation matrix of each gear object and perform a rotation using the current angle of rotation.

Some gears rotated in place, so it was a simple function call of rotz(angle) inside the run_animation function. Other gears, such as my spinning gear rotating around a path created by a rotating cylinder on a rotating box (see the orange and magenta gears in the image above), required some more thinking.

To make the orange and magenta spiked gears spin and rotate with the cylinder and box at the same time, I had to use 3 scene groups.

The first scene group was used to rotate the gear downwards, so it's flat on the X-Z plane. The second scene group was used to translate the gear up by 0.65. The reason for this was because in order to rotate the gear around the cylinder and box, I would have to rotate around the distance of the cylinder+box, which is 0.65. If I rotated the gear around the origin, the gear wouldn't have followed the rotation path of the cylinder. This second scene group would be in charge of the rotation around the y-axis (the spinning). The third scene group was used to translate the gear to its correct location (on top of the cylinder). Also, this scene group was in control of the rotation around the Z axis.

As for the line gear motion, it was simply updating the translation vector in small increments of its y-value (to have it move up or down). When it hit a limit/boundary, I would just reset the translation vector to its initial value.

Evaluation Information

```
void MyViewer::run_evaluation_test()
```

Use this function by calling it in the MyViewer constructor to test the performance of different inputs and different gear types. (The tests are commented out inside the function. Just uncomment one of the nine tests to see it run). See the table below for evaluation information. All tests were performed on an Intel i7 processor, and an Intel® HD Graphics 520 card.

Gear Type	# of faces	Creation Time (Seconds)	Memory Usage (During Animation)
Custom Spiked Gear	100,000	0.250604	135.5 MB
Custom Spiked Gear	300,000	0.769952	377.8 MB
Custom Spiked Gear	500,000	1.21075	622.4 MB
Normal Gear	10	0.000320435	14.7 MB
Normal Gear	50	0.00264621	17.7 MB
Normal Gear	300	0.531373	49.4 MB
Line Gear	1000	0.0680447	230.8 MB
Line Gear	5000	0.255429	1,093 MB
Line Gear	7500	0.399366	1,632 MB
Entire Scene	Around 10-20	0.00331736	23.8 MB