Software Applications: Digital Imaging

Wind Turbine Assignment – Matthew Dyson

The original brief of this assignment stipulated the creation of a Java3D model of a wind turbine, including moving parts and appropriate event listeners to call animations. This document details the process that I went through in creating my final turbine design, the problems and changes made during development, and technical information on the structure of the object tree.

Creating the universe

To begin, I created a blank universe with two lighting objects, one to shine on the turbine in order to illuminate it, and the other to illuminate the entire scene. This gives the impression of daylight over the whole scene. I chose not to make the turbine cast shadows, as the code required to make a shadow move in time with the turbine was too complex and wasteful of resources, which would have slowed down the animation effect. To begin with, I worked with a completely blank scene in order to give the best environment for designing the turbine model.

Creating the turbine model

Initially, I opted for a simple object making up the turbine itself, which would be made up of the standard Java3D objects (cylinders, cubes etc) which could then be easily adapted in future. With the future animation in mind, I separated the turbine into 3 different overlapping TransformGroups (see figure 1). The basic turbine mast is contained within one group, containing another group for the entire head (which can then have the vertical rotation applied to it), which in turn contains a final group with the blades and cone (which will perform the horizontal rotation).

In creating a Turbine object, the user stipulates the number of blades that the turbine will contain, allowing for a further level of customization. Once created, a private BranchGroup is initialized, to which is added all the separate components (in their respective TransformGroups) to make up the complete model. This BranchGroup can then be added into a universe as needed, allowing for multiple turbines in the same environment – a virtual wind farm!

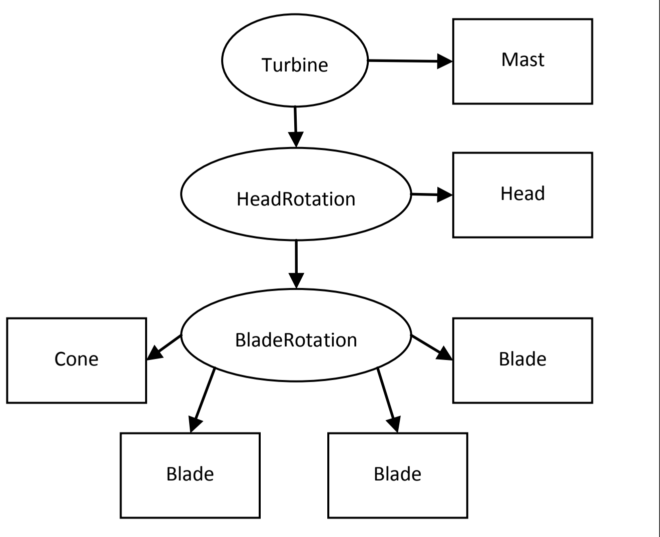


Figure 1: Turbine Object Grouping

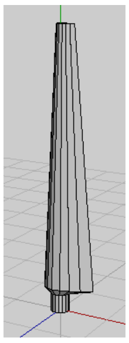
Each separate part of the Turbine BranchGroup is assigned a copy of the same Appearance object (containing a Material), which gives a white colour with a grey diffuse and specula.

Animations

First, I worked on the horizontal rotation of the blades and cone, aiming to get an infinite loop of the blades moving 360 degrees around the central axis. I accomplished this by use of a RotationInterpolator with an Alpha object running the timings, which consisted of a full loop every 5 seconds. This RotationInterpolator is then appended to the BladeRotation TransformGroup on creation, to start the blades turning once the object has been created.

I then applied this same process to the HeadRotation TransformGroup, changing the axis of rotation to make the entire head rotate round the vertical axis as soon as the object is rotated.

Adapting the turbine model



Once the basic turbine had been created (original code left within Turbine.class as comment for reference), I adapted the blades within the Turbine class to import an object file of a more realistic looking turbine blade (shown to the right). My earlier choice to use a single object for multiple blades ensured that at this point, I only had to design a single turbine blade, meaning that all the blades a user adds will look exactly the same, and be in exact proportion to each other.

In order to get a realistic looking turbine model, I also performed some rotations and translations on the object when importing, so that the blades are slightly angled into the wind, as in real life.

Event listeners

To allow rotation of the turbine head to occur on a button click command, I created a new method within the Turbine class (signature – doRotation(double moveByAngle)) which adapts the RotationInterpolator affecting the HeadRotation, as well as the Alpha object. The maximum angle of rotation is set to the current angle (relative to the vertical axis) plus the desired angle to move by (in radians), and the start time of the Alpha is set to the current time. This causes the rotation to begin, and the new angle will be reached 2.5 seconds after movement began (to give a smooth transition).

I then created a MouseBehaviour class that calls this function, passing a random number between 0.5 and 1.5, or the negative equivalent (to allow rotation both ways to be demonstrated). This behaviour is then added to the mast of the turbine. I chose to use the turbine mast as the button to avoid cluttering the screen, keeping the scene as realistic as possible and also allowing for use on multiple Turbine objects within the same scene.

Final universe

Finally, I added a background texture containing a sky, and a flattened Cube to form the floor, textured to look like grass.

Potential Expansions

In order to expand this project further, I would consider the following options. Dynamically changing the blade speed through an internal function of the Turbine class would allow for more realistic modelling, especially if linked to other objects reacting in time. This idea could also be linked to the rotation of the turbine head. The entire turbine could be made to react to external variable changes (a separate Wind object containing direction and speed) that could be modified at runtime, allowing multiple turbines to react in the same way to wind changes, as they would in the real world.

The addition of shadows to the universe would also improve the realism of the scene, however this is at the cost of vastly increased processing times, and could potentially create problems with the camera and lighting. Being able to slew and direct the camera around the scene would create a more immersive simulation, but again at the expense of processing cycles.

Obviously, the actual turbine model could be improved by realistic modelling within more advanced 3D Computer Aided Design software, and also by the addition of realistic textures to the model itself.