Project Objectives:
In this project you will learn about 2D texture mapping.

Project Description:
This is your final project of the quarter and now that you have learned about transformations, clipping, lighting, shading, culling, graphics primitives and animation, you will further bring your 3D scenes to life. You will do this by refining your models with high quality images. As before, your application will have to read complex models provided in the Fundamentals of Computer Graphics (FCG) file format and manipulate the provided data. You are expected to implement the following components for this assignment.

Part 1:
In your previous projects you have implemented a couple of basic 3D primitives, which you are now asked to texture map:
- Implement box mapping
- Implement cylinder mapping
- Implement cone mapping

Part 2:
You are asked to provide a user interface that allows showing the different features of your implementation, including:
- Support clamp mapping and repeat mapping
- Support modulate and replace texture functions (lighting is needed)

Part 3:
- Create a texture mapped highway underneath the Porsche and allow it to move as the tires spin
- Create a sky-box that encapsulates your scene and allows you to create an animated sky. I.e. the viewer will be located inside the box as a sky texture is applied to it and moved over time to simulate moving clouds.

Extra Credit (10 points):
- Implement sphere mapping
- Create a sky-sphere that encapsulates your scene and allows you to create an animated sky. I.e. the viewer will be located inside the sphere (or hemisphere) and a sky texture is applied to it and moved over time to simulate moving clouds.

Some of the requirements from previous projects carry over into this one since texture mapping either requires it or because they greatly aid in exploring the rendered models:
- Support scene interaction with the mouse (translate, rotate, scale)
- Support multiple lights (material properties and normals have to work)

In order to map two-dimensional images (textures) onto polygons you will have to define texture coordinates on a per vertex basis. These texture coordinates are then used by OpenGL to properly align your textures with the polygonal model. OpenGL provides the glTexCoord2f(…) function for this purpose. Texture coordinates are given in their own coordinate space, and are not modified by vertex transformations.
During the mapping process, color information from the texture is used to modify the color of the rendered primitive on a per-pixel basis. In OpenGL, textures can be specified with the glTexImage2D function. The glTexImage2D call provides a variety of internal and external texture storage formats. For this project, you will only have to support the GL_RGB, GL_UNSIGNED_BYTE external modes. You are free to choose the internal storage mode that best suits your needs.

Texture coordinates \((s, t)\) are normalized such that \((0, 0)\) always maps to the lower-left corner of the current image and \((1, 1)\) to the upper-right corner. Texture sizes are generally expected to be a power of two, both in width and in height.

If the texture coordinates that you specify are outside the \([0, 1]\) interval, two different strategies are supported by OpenGL to map them back into the interval:

- **Texture coordinate clamping**: all coordinates smaller than zero are clamped to zero, and all coordinates larger than one are clamped to one.
- **Texture coordinate repeating**: coordinate \(x\) is wrapped to the coordinate \(x - \text{floor}(x)\). This will cause a "tiling" of the texture across the primitive (i.e. you will see the same texture multiple times).

The wrapping modes can be set independently for both coordinate axes.

After the interpolated texture coordinates have been wrapped to the interval \([0, 1]\) using one of the above methods, they are scaled to the sizes of the current texture image, and the texture value is computed. There are two different methods to compute a texture value:

- **Nearest-neighbor sampling**: The value of the texture map entry ("texel") closest to the computed texture coordinate is returned.
- **Bi-linear interpolation**: The texture value is computed from the four texels surrounding the computed texture coordinate, and averaging their values using bi-linear interpolation.

After a texture value has been computed for a pixel using the above methods, the color is used to modulate the interpolated pixel color. OpenGL supports several different modulation formulas but the ones your implementation has to support are GL_MODULATE and GL_REPLACE. The GL_MODULATE formula computes the final pixel color by multiplying the pixel's interpolated color and texture color.

Chapter 9 of the OpenGL Programming guide provides a fairly detailed discussion of the texturing capabilities at your disposal.

**HAVE FUN !!!**