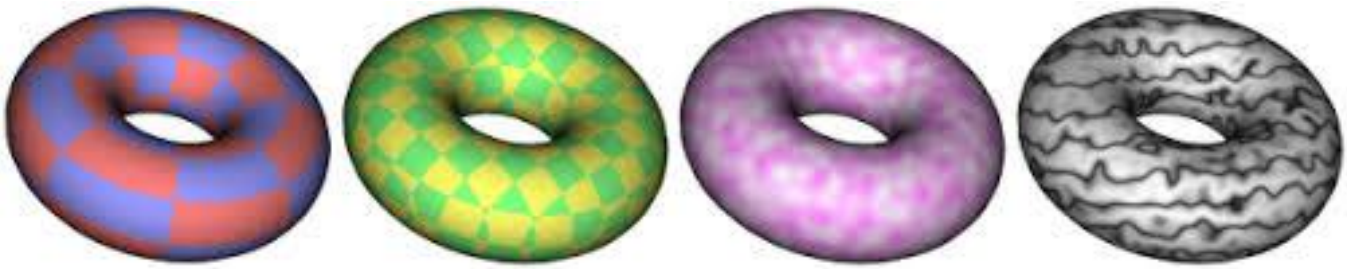
Programming Assignment #4

In this assignment you will texture your torus. See examples below:



Instructions:

1) Download **PA4_support_code.zip**, which contains two pairs of shaders (*texpersp* and *texpersplight*) you can use to render with a texture. Place both into the shaders folder inside your project. You will have to write code to create and initialize them in the same way as the existing shaders.

2) In this PA, you will have to load images into your program. You can do this in any way you prefer, including writing your own code to do it.

If you do not already know of a way, you can use the CImg Library (uploaded to CatCourses in **CImg.zip**). It is a simple image processing library contained in a single header file. You can include it in your project and load an image with just a few lines, for example:

```
#include <CImg.h>
using namespace cimg_library;

CImg<unsigned char> texture;
texture.load( "../donut.bmp" );
```

To test if your file loaded correctly, you can do `texture.display()`, which will open a new window and display it. If you have issues loading a file, it might be a particular format CImg does not support without dependencies; in that case, just convert the image to a bmp and it should work.

3) When you are ready, implement the requirements below.

Requirements:

Requirement 1 (60%): Compute correct texture coordinates for your torus

You will reuse your code for creating a smooth torus and add code for creating and using an image file as a texture for it.

The support code provides two pairs of shaders, *texpersp* and *texpersplight*:

- **texpersp.vert, texpersp.frag:** render with perspective projection and apply a texture using vec2 texture coordinates. Required arrays: vec4 coordinates, vec4 colors, vec2 texture coordinates.
- **texpersplight.vert, texpersplight.frag:** render with perspective projection and apply a texture using vec2 texture coordinates, and also apply Phong illumination. Required arrays: vec4 coordinates, vec4 colors, vec4 normals, vec2 texture coordinates.

You could start with *texpersp*, but since you should already have normals from last PA you can also go straight to *texpersplight*.

You will need to build a new array, this time of vec2's since texture coordinates are two-dimensional. (Remember to adjust the relevant OpenGL calls, such as `glBufferData` and `glVertexAttribPointer`, to take into account this difference!)

Texture coordinates are in the [0,1] range. **You will need to divide this range over your whole object.** If you simply give 0 or 1 to the texture coordinates of the vertices of each triangle, you will be repeating the whole texture on each face, which is not what we want. Your texture should completely wrap around the torus in a nice way (see examples above). If you get a strange result most likely your texture coordinates are wrong.

After you have loaded your image using CImg or some other method, you must generate a texture in the GPU and send the image data to it, for example:

```
	GLuint textureId;
	glGenTextures( 1, &textureId );
	glBindTexture( GL_TEXTURE_2D, textureId );
	glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, texture.width(), texture.height(), 0,
GL_RGB, GL_UNSIGNED_BYTE, texture.data() );
	glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
	glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
	glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
	glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
```

After this, once you have generated correct texture coordinates, you simply send the id of the texture you want to use to the shader, and activate the texture unit:

```
	PerspectiveTextureLightShader.SetUniform( "texId", textureId );

	glActiveTexture( GL_TEXTURE0 );
	glBindTexture( GL_TEXTURE_2D, textureId );
	// draw calls...
```

Requirement 2 (30%): Controls

In order to demonstrate that your object is correct and that your texture coordinates can be generated for any torus resolution, include the usual controls for changing the resolution of your torus (15%):

- '**q**' : increment the number of faces
- '**a**' : decrement the number of faces
- '**w**' : increment the *r* radius (by a small value)
- '**s**' : decrement the *r* radius (by a small value)
- '**e**' : increment the *R* radius (by a small value)
- **'d'** : decrement the *R* radius (by a small value)

Every time one of the keys above is pressed your application should instantly re-generate a new torus with updated texture coordinates. The result should always display the texture completely around the object for any chosen resolution.

Then, add the following control (15%):

- '**space bar**' : pressing the space bar should have your application cycle among at least 3 different texture images!

To change textures, you may simply load 3 images and resend the data to the GPU when using a different one. Alternatively, you can generate and set up 3 textures with `glGenTextures()` and the other calls, and send a different texture id to the shader in the display function. You may also try other solutions.

Requirement 3 (10%): Overall quality.

Everything counts here and, once again, there is no need to do anything complex. Just make sure your project works and looks good and you will get full points.

Submission:

Please follow the instructions in parules.txt (uploaded to CatCourses). In particular: please do not include any third-party support code and do not forget to Clean Solution before preparing your project for submission! Also, check for hidden folders (such as .vs) which can sometimes balloon to hundreds of megabytes!