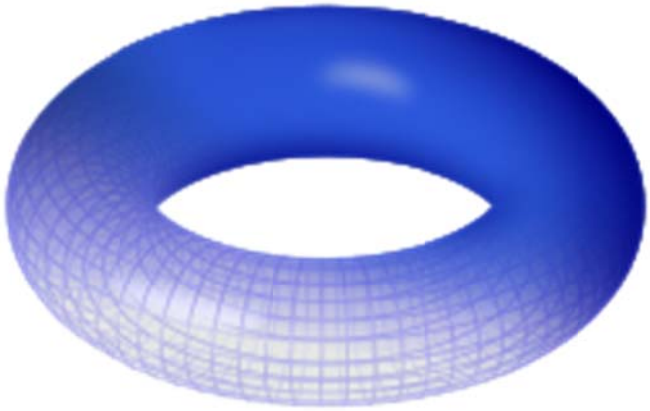
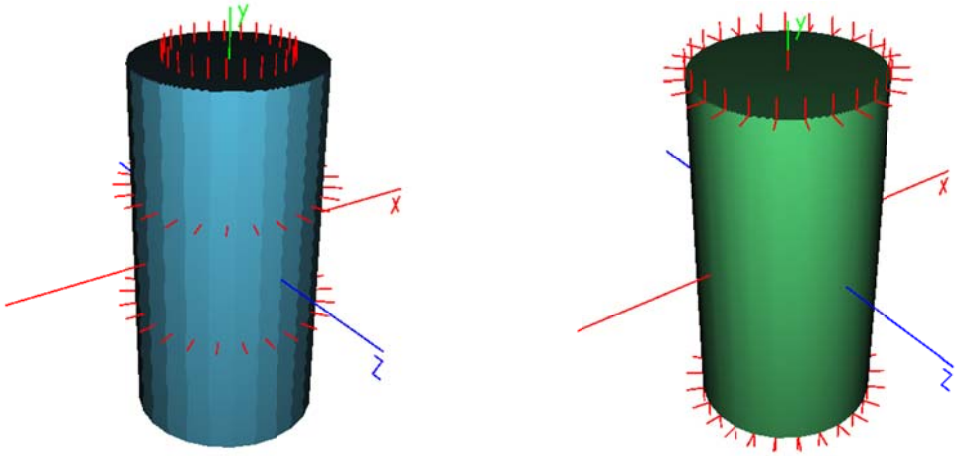
Programing Assignment #4

In this assignment you will generate faces and normal vectors for your torus primitive object in order to render it in smooth shading:



You will display your torus in two modes: smooth and flat shading, and will display the normal vectors as well. As an example, see below how to display the normal vectors for a cylinder primitive (but do not forget you will be working on a torus primitive instead):



Requirement 1 (30%) - Implement and display your shaded torus object

You will re-use your code from the assignment where you created a torus primitive but you will now display the torus in both smooth and flat shading. For simplicity, the recommended approach is to place the triangles and normal vectors that you generate in GsModel, and let SnModel's renderer do the OpenGL interface for you.

You are welcome to instead extend your previous torus scene node in order to accomplish the asked requirements; however, in this case you will need to update your calls to OpenGL and send additional arrays and use new shaders. While this can be quite difficult if you are new to OpenGL, if you choose to go this way you may use glr_model.cpp as a guide.

But the recommended approach is to re-use your code that generates triangles for a torus primitive, and instead of sending your data to OpenGL from your own scene node type, you will instead simply put the triangles in the V and F arrays of GsModel:
- the V array contains all used vertices,
- and the F array has indices to the entries in V:

```
struct Face
{      int a, b, c;
       ...
};
GsArray<GsPnt> V;  //!< List of vertex coordinates
GsArray<Face>  F;  //!< Triangular faces indices to V
```

These arrays are declared inside GsModel, so be sure to read the many comments in gs_model.h to understand the format.

After the V and F arrays are filled, you will then fill the N array with normal vectors:

```
GsArray<GsVec> N;  //!< List of normals
```

Smooth Shading:

For smooth shading the N array should have the same size as the V array. This means each vertex v is associated with 1 normal vector, which is the normal vector to the torus surface at point v. It is straightforward to compute the normal vector from the parameters of the torus: it will simply be the normalized vector from the circular axis of the torus to v. For each vertex computed and stored in V, you should then add the corresponding normal vector to N. After all elements are set, call:

```
mymodel->set_mode ( GsModel::Smooth, GsModel::NoMtl );
```

⇨ Note that your code has to compute the normal vectors - do not call the smooth() function in GsModel !

Flat Shading:

For flat shading the N array should have the same size as the F array, such that each triangular face is associated with 1 normal vector, which is the normal vector to the triangular face. After all normals are set, call:

```
mymodel->set_mode ( GsModel::Flat, GsModel::NoMtl );
```

Requirement 2 (30%) – Visualization of Normals

You will also need to draw line segments showing each normal vector you are using in GsModel. You should draw the normal vectors in the following way:

- Smooth shading: segments will originate from each vertex and point outwards,
- Flat shading: segments will originate from the center of each face and point outwards. (See figures at the beginning of this document for examples.)

For flat shading, the reason for the normal vectors to originate from the center of each triangle is to indicate that all vertices of a triangle use the same normal vector; while for smooth shading each vertex uses a unique normal vector.

To display the segments representing the normal vectors you just need to use node SnLines in your scene graph. Every pair of points pushed to SnLines will form a line segment. After the normals are created and stored in GsModel, you can then read them and create segments in SnLines. For ex., you may write a function to compute the segments by doing something like this:

```
void MyViewer::compute_segments ( bool flat )
{
	SnLines* l = my_sn_lines_node;
	l->init();
	l->color ( ... );
	if ( flat )
	{	GsModel& m = *my_sn_model_node->model();
		for ( int i=0; i<m.F.size(); i++ )
		{
			const GsVec& a = m.V[m.F[i].a];
			const GsVec& b = ...
			const GsVec& c = ...
			GsVec fcenter = (a+b+c)/3.0f;
			l->push ( fcenter, fcenter+(m.N[i]*my_scale_factor) );
		}
	}
	else
	{	// Treat smooth case here
		...
	}
}
```

Visualizing the normals is a good way to be sure you are achieving correct renderings. If the object does not look correctly shaded then your normals may not be correct.

Requirement 3 (30%) – Controls

In order to demonstrate that your object is correct, include the usual controls for changing the resolution of your torus:

- '**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)

Then, add the following controls:

- '**z**' – to enable Flat Shading
- '**x**' – to enable Smooth Shading
- '**c**' – show normal vectors
- '**v**' – do not show normal vectors

Additional Notes:

- you may just call my_sn_lines_node->visible(bool b) to show/hide nodes.
- As before, you may start this project from the sigapp.7z project.
- Do not forget that you may post questions to our forum:
https://sigdev.boardhost.com/
(questions should be about how to use SIG, and not about finding a bug in your PA!)

Requirement 4 (10%): Overall quality.

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

Submission:

Please present and submit your PA as usual according to parules.txt. (Do not forget to upload your project before the deadline!)