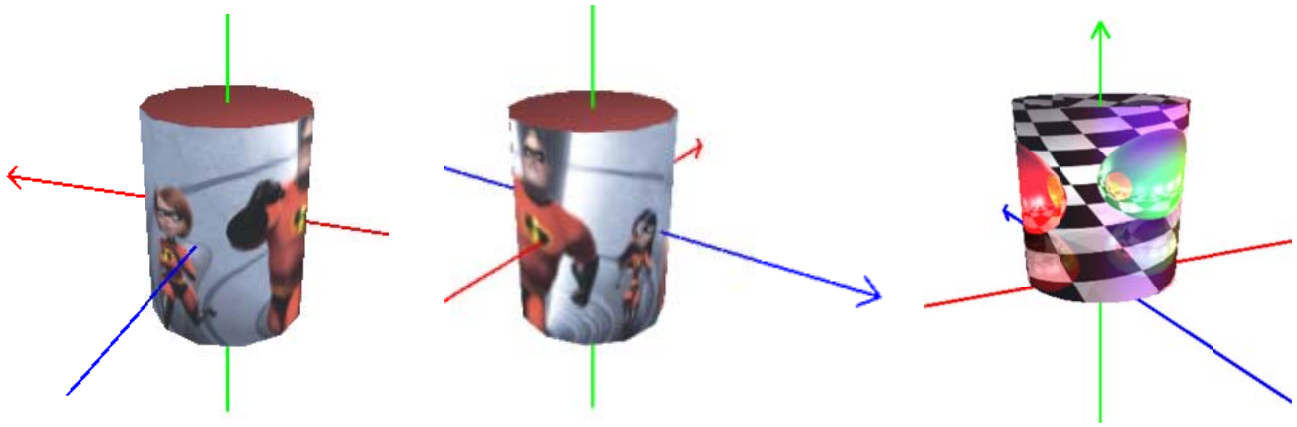
Programing Assignment #6

In this assignment you will texture a tube model, as shown in the pictures below.



Instructions:

Requirement 1 (50%) – Compute correct texture coordinates and apply it to your tube

Follow the steps below:

1) First add to your project the files inside gsimage.7z, which is given as support code. These files contain a simple GsImage class which will allow you load an image in .png format. ***These files have to be placed inside the gsim directory of the support code.***

2) Create a SoTexturedTube class in your project and use it to contain the textured tube that you will implement. You will need to work with (at least) 3 arrays/buffers: one for vertex coordinates, one for normals, and one for texture coordinates. For example, you should have the following three arrays as members of your SoTexturedTube class:

```
std::vector<GsVec>   P; // coordinates
std::vector<GsVec>   N; // normals
std::vector<GsVec2>  T; // texture coords
```

3) In your init() function you will therefore need to declare 3 OpenGL buffers. Keep all the uniform variables you used for your smooth object PA, and add one more:

```
uniform_location ( n, "Tex1" );
```

The uniform variable "Tex1" is used by the shader to access your “texture unit”. In this PA you will only need 1 texture per object, so we can set that uniform variable to contain index 0 at initialization time, and it will always be 0. (It will be 0 even later when you will add a 2nd texture.)

The code below contains several initialization calls that you will need to do in order to load and initialize your texture:

```
GsImage I;
gsuint id;
if ( !I.load("../image.png") ) { std::cout<<"COULD NOT LOAD IMAGE!\n"; exit(1); }

glGenTextures ( 1, &_texid ); // generated ids start at 1
glBindTexture ( GL_TEXTURE_2D, _texid );
glTexImage2D ( GL_TEXTURE_2D, 0, 4, I.w(), I.h(), 0, GL_RGBA, GL_UNSIGNED_BYTE, I.data() );

glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
glGenerateMipmap ( GL_TEXTURE_2D );

glBindTexture ( GL_TEXTURE_2D, 0 );
glBindVertexArray ( 0 );

I.init(0,0); // free image from CPU
```

4) Once the image is loaded and initialized as a texture, you can then turn your attention to generating the texture coordinates. When you create the coordinates of your tube, for each vertex added to your P array, you must as well compute its corresponding texture coordinates and add them to the T array. For every vertex P[i] its corresponding texture coordinates will be T[i]. Remember that texture coordinates are 2D and should be between 0 and 1. Your build() method will need to bind the texture buffer with commands like the ones below:

```
glBindBuffer ( GL_ARRAY_BUFFER, buf[2] );
glBufferData ( GL_ARRAY_BUFFER, T.size()*2*sizeof(float), &T[0], GL_STATIC_DRAW );
glVertexAttribPointer ( 2, 2, GL_FLOAT, GL_FALSE, 0, 0 ); // false means no normalization
```

5) Finally, if everything above is correctly addressed, you can then use the new shaders provided at the end of this document to check your results. You should draw your object as triangles, in the same way as in the “smooth object” PA.

Choose any image to load as a texture. Your textured tube must be perfectly wrapped by the texture, without any gaps or overlaps. There must not be any strange rendering artifacts due texture coordinates not correctly computed.

Requirement 2 (10%) – Keys to control your tube generation

Maintain the following key controls used to parameterize the capsule of PA3:

- '**q/a**' : increment/ decrement the number of faces (by one)
- '**w/s**' : increment/ decrement the top face radius (by a small value)
- '**e/d**' : increment/ decrement the bottom face radius (by a small value)

Your texture mapping should always work correctly, completely wrapping around the tube for any valid combination of the parameters above. Every time one of the keys is pressed your application should instantly re-generate a new tube with always correct texture coordinates.

Requirement 3 (20%) – Add a switch to a second texture

For this requirement you will load 2 images and switch between them by pressing a key. You will basically call 2 times the code that loads and declares a texture to OpenGL; but you should of course save in your code the 2 different texture ids, so that at rendering time you can switch to the texture that you would like to use. Key to use: **'z'**; which should be the key to switch between two different image textures. Your draw method should look like this:

```
glUseProgram ( prog ); // set program to use

glBindVertexArray ( va[0] );
glBindTexture ( GL_TEXTURE_2D, CurrentTexID ); // select here the texture to use

// update all needed uniforms
...

// draw:
glDrawArrays ( GL_TRIANGLES, 0, _numpoints );
glBindVertexArray ( 0 );
```

Requirement 4 (20%) – Control of the shading influence

As a final requirement, you will edit the provided fragment shader so that it uses a controlled variable for specifying how much of the object's shading color is added to the texture color. In the provided fragment shader a simple fixed constant (of 0.75) is used. Replace that by a uniform and update its value according to the following key controls:

- '**x**' : increases influence of the shading color
- '**c**' : decreases the influence of the shading color

This requirement is extremely simple to be implemented, you just need to understand how to create another uniform variable to pass to your shader. Feel free to explore other combinations between the color computed by the shader and the color extracted from the texture. The images above show variations obtained with a red material and with constants 0, 0.25, 0.5, and 0.75. Re-use all your shading parameters from PA4.

Enjoy your texturing !

Example vertex shader for textured Gouraud shading (texgouraud.vert):

```
# version 400

layout (location = 0) in vec3 vPos;
layout (location = 1) in vec3 vNorm;
layout (location = 2) in vec2 vTexCoord;

uniform mat4 vTransf;
uniform mat4 vProj;
uniform vec3 lPos;
uniform vec4 la;
uniform vec4 ld;
uniform vec4 ls;
uniform vec4 ka;
uniform vec4 kd;
uniform vec4 ks;
uniform float sh;

out vec4 Color;
out vec2 TexCoord;

vec4 shade ( vec4 p )
 {
   vec3 n = normalize ( vNorm*mat3(vTransf) ); // vertex normal
   vec3 l = normalize ( lPos-p.xyz );          // light direction
   vec3 r = reflect ( -l, n );                 // reflected ray
   vec3 v = vec3 ( 0, 0, 1.0 );                // view point
   vec4 amb = la*ka;
   vec4 dif = ld*kd*max(dot(l,n),0.0);
   vec4 spe = ls*ks*pow(max(r.z,0.0),sh);      // r.z==dot(v,r)
   if ( dot(l,n)<0 ) spe=vec4(0.0,0.0,0.0,1.0);
   return amb + dif + spe;
 }

void main ()
 {
   vec4 p = vec4(vPos,1.0) * vTransf; // vertex pos in eye coords
   Color = shade ( p );
   TexCoord = vTexCoord;
   gl_Position = p * vProj;
 }
```

Example fragment shader for textured Gouraud shading (texgouraud.frag):

```
# version 400

in  vec4 Color;
in  vec2 TexCoord;
out vec4 fColor;

uniform sampler2D Tex1;

void main()
 {
   fColor = 0.75*Color + texture2D ( Tex1, TexCoord );
 }
```