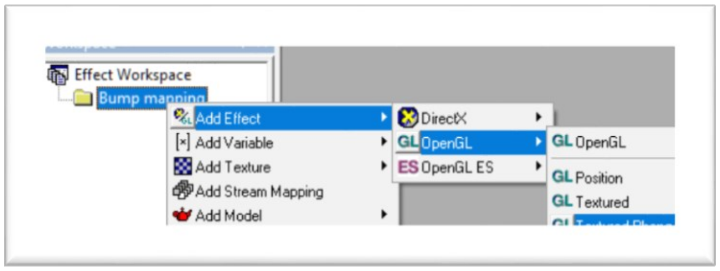
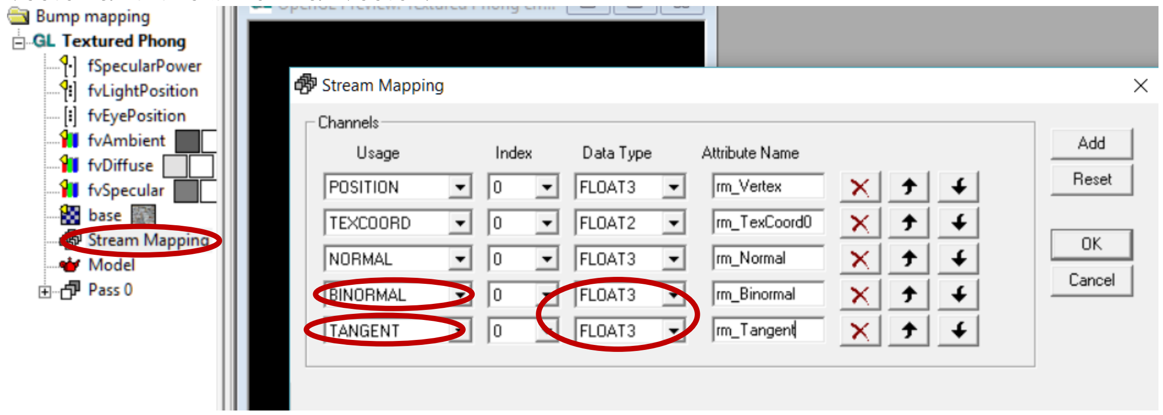
The aim of the lab is to write a set of bump-mapping effects using GLSL. All the exercises described below are based on the per-fragment-lighting rendering technique. If you do not have a good understanding about the differences between per-vertex lighting and per-fragment lighting, please review Week 25's lecture notes on lighting. You can also find a brief introduction online about how to implement per-fragment lighting using GLSL from Clockworkcoders Tutorials at https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php

## Exercise 1. Bump mapping using a normal map

1) Add the default OpenGL per-fragment lighting effect "Textured Phong" to the effect group you have just created.



   a) Open Stream Mapping node and add two more channels to the stream mapping to input to the vertex shader two more vectors associated with each vertex of your triangle mesh model: the tangent vector and the binormal vector:



| Usage | Index | Data Type | Attribute Name |
|---|---|---|---|
| POSITION | 0 | FLOAT3 | rm_Vertex |
| TEXCOORD | 0 | FLOAT2 | rm_TexCoord0 |
| NORMAL | 0 | FLOAT3 | rm_Normal |
| BINORMAL | 0 | FLOAT3 | rm_Binormal |
| TANGENT | 0 | FLOAT3 | rm_Tangent |

   You need these vectors to build the matrix required to transform light direction and view direction from view space into tangent space.

2) Open vertex program
   a) Declare the following built-in RenderMonkey variables:

```
attribute vec3 rm_Binormal;
attribute vec3 rm_Tangent;
```

b) Transform the tangent and binormal vectors into view space, this is because the default per-fragment lighting is implemented based on view space.

```
vec3 vBinormal  = gl_NormalMatrix * rm_Binormal;
vec3 vTangent  = gl_NormalMatrix * rm_Tangent;
```

c) Normalize normal, binormal and tangent vectors into unit vectors:

```
Normal       = normalize(Normal);
vBinormal    = normalize(vBinormal);
vTangent     = normalize(vTangent);
```

d) Construct the matrix to transform view direction and light direction into tangent space :

```
mat3 View2Tangent = mat3(vTangent , vBinormal , Normal );
```

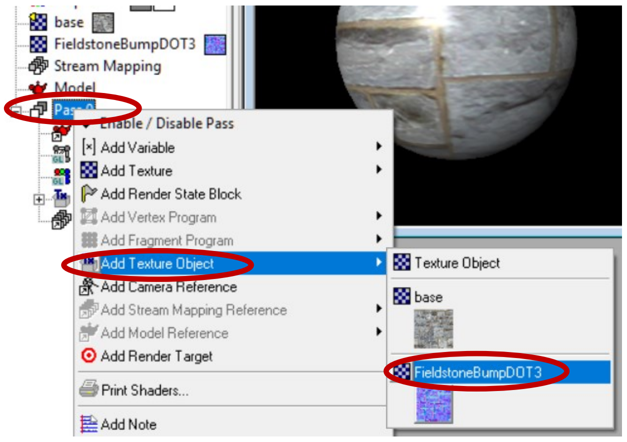
(e) Transform light direction and view direction into tangent space using the matrix constructed at step (d) :

```
ViewDirection       *=  View2Tangent;
LightDirection      *=  View2Tangent;
```

3) Open fragment program

a) Load a premade **normal map** texture, say, the texture "FieldstoneBumpDOT3.tga" provided in RenderMonkey.

b) Create a texture object corresponding to the texture FieldstoneBumpDOT3.tga. A texture object is an OpenGL object to which a shader program can get access. To create the texture object in a pass, right-click the pass and then select "Add Texture Object" from the drop-down manu to add an texture object by linking it with a texture you have created.



c) Rename the the texture object, say, as "bumpMap"

d) In the top of fragment shader, add a uniform variable corresponding to the texture object:

```
uniform sampler2D bumpMap;
```

e) In the body of main( ), decompress the normal map:
   vec3 N = 2.0*texture2D( bumpMap, Texcoord ).xyz - 1.0 ;

f) Replace **Normal** vector in the following line of code with the vector N defined above:
   vec3 fvNormal = normalize( Normal );

4) Recompile your effect, you should now see a bumpy graphics object. You may need to change the default texture with a texture that matches the bump texture you used to enhance the visibility of a bumpy surface.
5) Introduce variables to specify the bump density and bump height.

## Exercise 2. Bump mapping using a height map

Height mapping is similar to normal mapping. The only difference between normal mapping and height mapping is that the normal vectors at different fragments need to be calculated from a height map. If you have done Exercise 1 successfully, simply make a copy of the effect you implemented in Exercise 1 and rename it as, say, Height mapping. In exercise 1, the normal vector is directly read from a normal map. In this exercise, you need to calculate the normal vector fvNormal from a height map.

1) Load a texture representing a height map and create a texture object from it. In fragment shader, create a uniform variable corresponding to the texture object:
   uniform sampler2D heightMap;

2) Get the height at a fragment from the height map, say, from the colour channel red:
   float Height = texture2D( HeightMap, texCoord).r;

3) Find the normal vector:
   ```
   float  dFx= dFdx(Height);
   float  dFy= dFdy(Height);
   vec3 N= normalize(  vec3(−dFx, −dFy, 1.0) );
   ```

   or define the normal vector using a uniform variable bumpH to enhance the significance of bumps:
   ```
   vec3 N= normalize(  vec3( − bumpH*dFx,  −bumpH*dFy, 1.0) );
   ```

4) Apply the basic lighting model using the normal vector **N**.
5) Compile you shaders. You should now see a bump-mapped effect. You may notice some obvious artefact of the bump effect. You can improve the visual quality of bump-mapped object by using the idea introduced in the lecture.

## Exercise 3. Procedural Bump mapping

Create some procedural normal maps for generating bumpy effects without using a normal map or height map.

## Exercise 4. Parallax bump mapping

Based on what you have achieved in Exercises 1 and 2, implement parallax bump mapping using the three provided textures, namely, rockwall, rockwall_normal, rockwall_height.

## Exercise 5 (optional). Parallax bump mapping without using a normal map

Based on what you have achieved in Exercise 4 to implement parallax bump mapping effect using only a height map and a colour map.

## Exercise 6 (optional). Ray-marching-based parallax bump mapping

Improve what you have achieved in Exercise 4 following the idea of ray marching to implement a more accurate parallax bump mapping effect by calculating a more accurate approximation to the texture coordinates at which eye ray intersects the high map surface.

==========================================================

Written by Dr Qingde Li(q.li@hull.ac.uk).
Mar. 15, 2019.