GLSL shaders

Due date: Monday, May 14

In this assignment you will implement a vertex and a fragment shader. A basic framework for loading and rendering a mesh using shaders is provided on the class web page. You need to install the `GLEW` library to compile this code if you are using Windows.

1 What your program should do

Vertex shader. Write a shader that (a) computes all necessary information for the bump-mapping fragment shader (similar to the brick shader in GLSL examples); (b) creates a periodic blend between the original surface and the surface displaced in the normal direction by an amount depending on the surface point position, $h(x, y, z) = c \sin(x) \sin(y) \sin(z)$, where c is a scale you choose based on the mesh size (1/100-1/20 of the mesh bounding box size is reasonable).

You need to interpolate both the position and the normal for each vertex. To approximate the normal n_d of the displaced surface, use the following formula:

$$n_d = n - t_u \frac{\partial h}{\partial u} - t_v \frac{\partial h}{\partial v},$$

n is unit the normal of the original surface, t_u and t_v are unit tangent vectors, and $\frac{\partial h}{\partial u}$ and $\frac{\partial h}{\partial v}$ are derivatives of $h(x, y, z)$ along these tangent vectors. The result needs to be normalized so that its length is 1.

To apply this formula in a vertex shader, you need to pick two tangent vectors t_v and t_u perpendicular to the normal passed to the shader from C code (e.g. take the cross product of the normal with a coord. direction to get t_u and then compute $t_v = n \times t_u$ (both vectors need to be normalized to have length 1).

To compute derivatives of h along tangents at a vertex, take the gradient $[\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{\partial h}{\partial z}]$ of $h(x, y, z)$ with respect to (x, y, z), and compute dot products of the gradient with t_u and t_v.

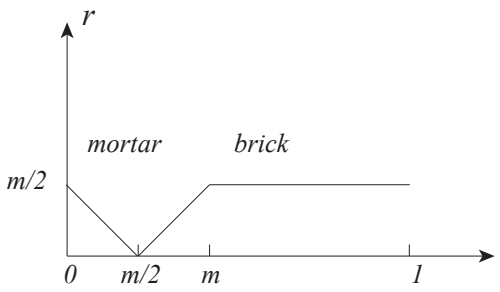
Fragment shader. Implement a procedural bump map. The bump map will modify the normal (computed in the vertex shader and interpolated to each pixel).

Generate a pattern similar to GLSL brick shader example (feel free to use another pattern of your choice that can be computed procedurally). Your normals should coincide with the surface normals passed from the vertex shader, on the part of the surface corresponding to brick, and create an appearance of a V-shaped groove on the suraface in mortar areas.

The brick texture is defined in `brick.frag` example as follows. If brick dimensions are (w, h, d), then each brick is corresponds to 3 integer indices (m, n, k). Its location is defined by inequalities $m < x/w < m + 1$, $n < y/h < n + 1$, $k < z/d < k + 1$ for even n, and the first inequality is replaced by $m - 0.5 < x/w < m + 0.5$ for odd n (this creates the brick pattern shift) along y axis).

Let (x', y', z') be equal to $(\{x/w\}, \{y/h\}, \{z/d\})$ for even n, and $(\{x/w + 0.5\}, \{y/h\}, \{z/d\})$ for odd n, where $\{\}$ means taking the fractional part. Then the condition for brick (as opposed to mortar) color to be used at the point (x, y, z) is given by $t < x', y', z' < 1$, where t is mortar thickness relative to the total brick size.

The distance r to the center of the mortar area can be determined as $r = \min(|x' - t/2|, |y' - t/2|, |z' - (t/2)|)$, and truncate it at $t/2$: $r = min(t/2, r)$ The figure shows r along one of the coordinate directions away from corners (i.e. in the case when the same coordinate is always picked in the minimum).



If i is the coordinate selected by minimum, then take the normal change vector dn to be e_i, i.e. the vector along the direction towards the nearest brick boundary. Adjust this vector to be perpendicular to the normal to get a vector dn', and normalize it to have unit length.

Finally, define the new normal as $n' = n$ if $r = t/2$, and $n' = n \pm g\, dn'$, where g is a constant determining how deep the grove appears (e.g. for the same depth as width, use $t/2$), and the sign is chosen based on whether $x' - t/2$ is positive or negative, if x' is the coordinate closest to $t/2$ (it can also be y' and z').

2 What to turn in

The source code (C and shaders) and a working executable of your program.