Foundations of Computer Graphics

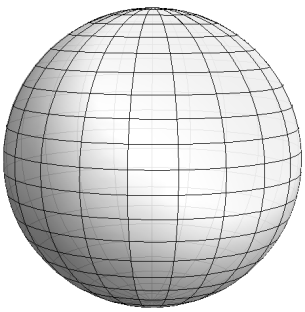
Assignment 5

Instructions:

- Assignments can be submitted in groups of at most three. The purpose of groups is to learn from each other, not to divide work. Each member should participate in solving the problems and have a complete understanding of the solutions submitted.
- Submit your assignments as a zip file (one per group) which includes the Common directory and a separate directory for each of the assignments so that we can run your code by just extracting the zip file and double-clicking the html files.

Problem 1 (10 points)**.**
Make the following modifications to the code you wrote for Problem 1 of the previous assignment.

1. Modify the `Sphere` function so that the mesh is created by splitting each quad in a grid formed by circles of fixed latitude or longitude (shown below). Furthermore, the object returned by the function should have three properties `positions, normals` and `triangles` where `positions[i]` and `normals[i]` contain the position and normal vector at the i^{th} vertex and `triangles[j]` contains the three indices of the vertices forming the j^{th} triangle in the mesh.

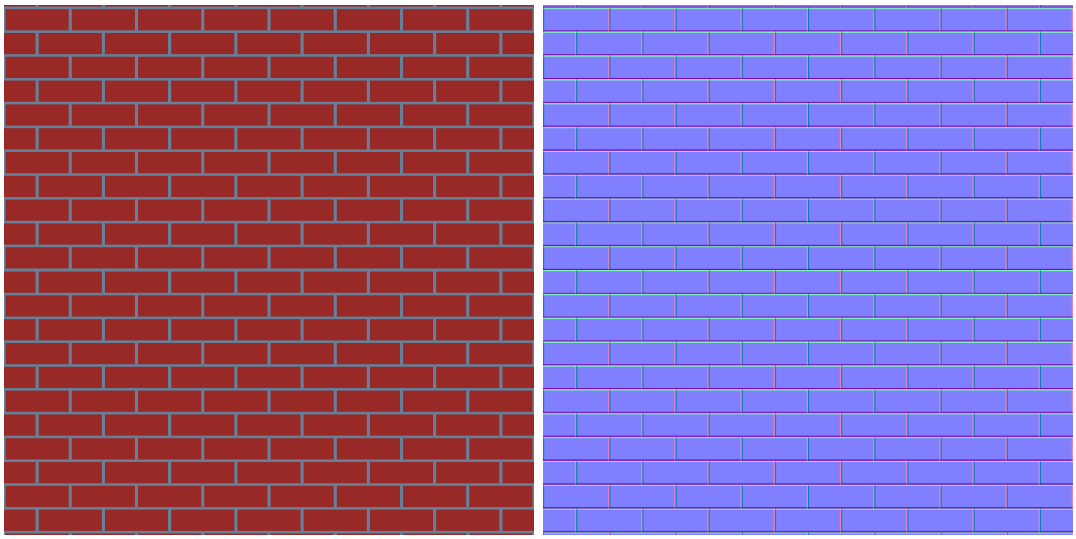


2. Update the `objInit(Obj)` function so that when the argument `Obj` has the `triangles` property, it uses index buffers and `gl.drawElements` instead of `gl.drawArrays` in the `Obj.draw()` function. The modified function should still work with the previous version of the `Sphere` function i.e., when the argument `Obj` does not have the `triangles` property.

Problem 2 (10 points)**.**
Display a unit square with the texture `wood-diffuse.jpg` and using `wood-normal.jpg` as the normal map. Use phong lighting and set the shininess value so the square looks like a polished wood surface.

Provide a virtual trackball so that the square can be moved around. You should check that as the square is moved, the texture and lighting remain consistent.

Problem 3 (10 points)**.**
The goal of this exercise is to procedurally generate the diffuse and normal map for a brick wall shown below.



Given texture coordinates (s, t), we need need an algorithm that computes the corresponding diffuse color and the normal vector.

The diffuse map consists of two colors, the brick color (RGB color (0.6, 0.16, 0.15)) and the mortar color (RGB color (0.4, 0.5, 0.6)). Each brick has a width 0.24 and height 0.08. The diffuse color corresponding to the texture coordinate (s, t) can be computed from this information.

Each brick has an elevation of $h = 0.01$. The elevation goes smoothly from 0.0 at the center of a mortar strip to h on either edge of the strip. Choose and appropriate smooth function and compute the normal vector from the resulting height function.

Write code to create the diffuse and normal maps and display them on the screen. Recall that in the normal map we obtain the RGB color by linearly mapping the coordinates of the corresponding unit normal vector to the range $[0, 1]$.

Test the created diffuse and normal maps with your code for Problem 2.

Problem 4 (10 points)**.**
The goal of this exercise is to produce drawings of Mandelbrot and Julia sets.

The Mandelbrot set is defined as follows. For each complex number $x + iy$, we construct a sequence of complex numbers $z_0, z_1, \cdots$ where $z_0 = 0$ and $z_{n+1} = z_n^2 + c$, where $c = x + iy$ and $i = \sqrt{-1}$. The complex number $x + iy$ belongs to the Mandelbrot set if and only if all the complex numbers in the sequence have magnitude at most 2. The magnitude of a complex number $a + ib$ is $\sqrt{a^2 + b^2}$.

For drawing such a set, we associate with each point (x, y) in the plane, the complex number $x + iy$ and we assign a color black or white to it depending on whether $x + iy$ belongs to the Mandelbrot set. However, a more beautiful drawing is obtained if we color each point depending on how quickly in the sequence corresponding to it we get a complex number of magnitude more than 2.

For the purpose of programming, we can compute the first (at most) N complex numbers in the sequence for each point and find the smallest k such that z_k has magnitude more than 2. We can then give the point a color based on k. If none of the complex numbers numbers among the first N in the sequence have magnitude more than 2, then we can give the point a default color (say black). N can be set to around 300.

Julia sets are define in a very similar way. This time, the sequence for the point (x, y) starts with $z_0 = x + iy$ and c is a fixed complex number that belongs to the Mandelbrot set. Every choice of c gives a different drawing.

We would also like Pan and Zoom functionalities in our program i.e., we should be able to drag the picture using the mouse and zoom into or out of the picture using the mouse wheel. Please write the appropriate event handlers for the same.