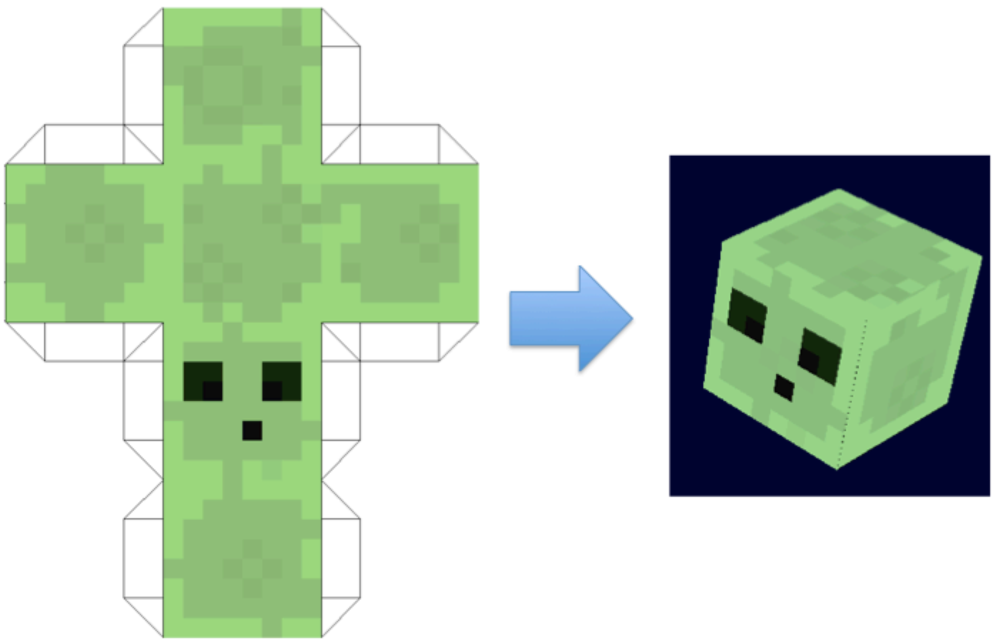
CSE471 Introduction to Computer Graphics (Fall 2019)
Instructor: Prof. Won-Ki Jeong
Due date: Nov 24, 2019, 11:59 pm.

Assignment 3: Texture Mapping and Off-screen Rendering

In this assignment, you will implement environmental mapping to render mirror-like surfaces using OpenGL. The goal of this assignment is to learn how to use texture mapping and off-screen rendering techniques. The following sections describe the functions you need to implement.

1. 2D texture mapping (20 points)

Using the provided images of minecraft characters, write a code to render 3D models as shown below (this is the simplest example, and you can create more complicated models using the images provided):



In order to do this, you need to load the image, create a 2D texture, and assign per-vertex texture coordinate. You also need to design the 3D model's geometry manually (mostly collection of cubes).

Use the I/O functions in bmploader.cpp to load the BMP images.

Grading:
- Render accurate texture-mapped minecraft character (10 pts)
- Virtual trackball to rotate the object (10 pts)

2. Environment mapping using static cube mapping (30 points)

You need to implement environment mapping using a static cube map texture (texture is not changing). A cube map texture consists of six 2D textures, and each 2D texture can be created as follows:

```
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGBA,
             imageSize, imageSize, 0, GL_RGBA, GL_UNSIGNED_BYTE,
             tex1);
```

`tex1` is the array storing raw texture values for positive X face. You need to do this six times for `GL_TEXTURE_CUBE_MAP_POSITIVE/NEGATIVE_X/Y/Z` to generate all six faces. You can use a synthetic checkerboard texture map as follows (this is 4x4 2D texture per each side):

```
#define imageSize 4
static GLubyte tex1[imageSize][imageSize][4];
static GLubyte tex2[imageSize][imageSize][4];
static GLubyte tex3[imageSize][imageSize][4];
static GLubyte tex4[imageSize][imageSize][4];
static GLubyte tex5[imageSize][imageSize][4];
static GLubyte tex6[imageSize][imageSize][4];

for (i = 0; i < imageSize; i++) {
      for (j = 0; j < imageSize; j++) {
            c = ((((i&0x1)==0)^((j&0x1))==0))*255;
            tex1[i][j][0] = (GLubyte) c;
            tex1[i][j][1] = (GLubyte) c;
            tex1[i][j][2] = (GLubyte) c;
            tex1[i][j][3] = (GLubyte) 255;

            tex2[i][j][0] = (GLubyte) c;
            tex2[i][j][1] = (GLubyte) c;
            tex2[i][j][2] = (GLubyte) 0;
            tex2[i][j][3] = (GLubyte) 255;

            tex3[i][j][0] = (GLubyte) c;
            tex3[i][j][1] = (GLubyte) 0;
            tex3[i][j][2] = (GLubyte) c;
            tex3[i][j][3] = (GLubyte) 255;

            tex4[i][j][0] = (GLubyte) 0;
            tex4[i][j][1] = (GLubyte) c;
            tex4[i][j][2] = (GLubyte) c;
            tex4[i][j][3] = (GLubyte) 255;

            tex5[i][j][0] = (GLubyte) 255;
            tex5[i][j][1] = (GLubyte) c;
            tex5[i][j][2] = (GLubyte) c;
            tex5[i][j][3] = (GLubyte) 255;

            tex6[i][j][0] = (GLubyte) c;
            tex6[i][j][1] = (GLubyte) c;
            tex6[i][j][2] = (GLubyte) 255;
            tex6[i][j][3] = (GLubyte) 255;
        }
}
```

When you create your texture map, use texture wrap mode and min/mag filter mode appropriately. For example, below is `repeat` for wrapping and `nearest neighbor` for min/mag filtering.

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_REPEAT);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Once you create a cube map, you can enable the cube map texture as follows:

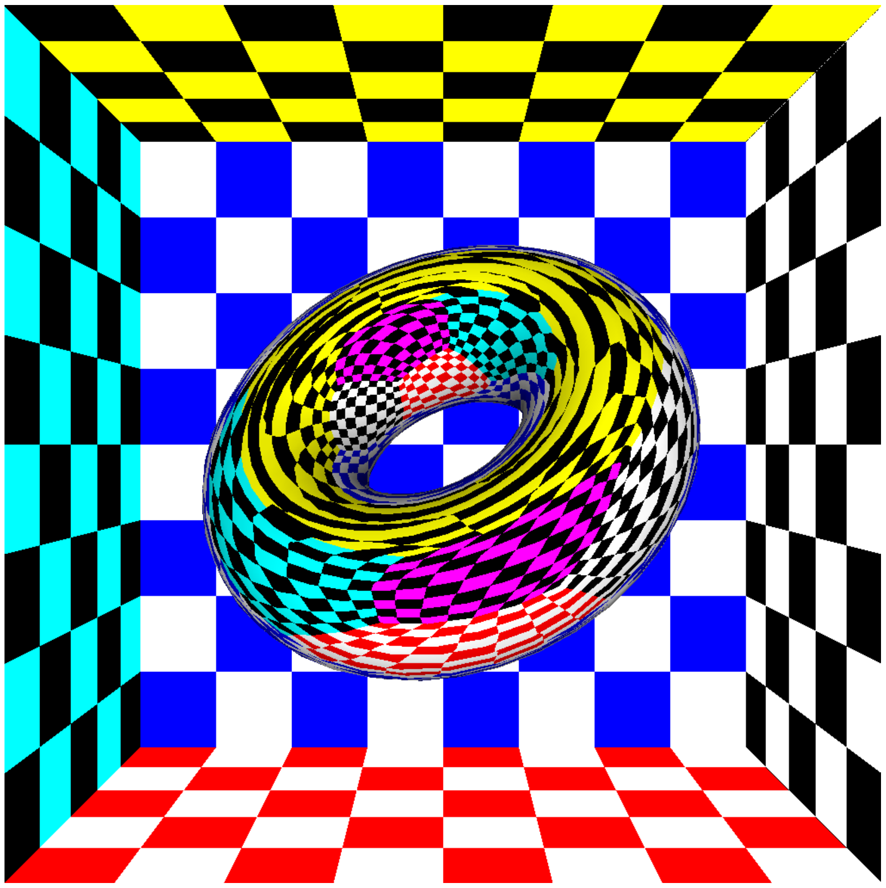
```
glEnable(GL_TEXTURE_CUBE_MAP);
```

Similar to other textures, you need to assign per-vertex texture coordinate. For a cube map, texture coordinate is a (s,t,r) 3D vector and actual texture sampling is done by shooting a ray from the center of the cube to the given vector direction. Even though we can assign texture coordinate manually, OpenGL also provides an automatic texture coordinate generation method as follows:

```
glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );
glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );
glTexGeni( GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP );
glEnable( GL_TEXTURE_GEN_S );
glEnable( GL_TEXTURE_GEN_T );
glEnable( GL_TEXTURE_GEN_R );
```

In order to test the correctness of the cube map, you should draw a box whose faces are texture-mapped checker board textures generated above, and render a 3D object (sphere, torus, or teapot) inside the box with a cube map texture.

The below is the screenshot of environment mapping on a torus using a given synthetic checkerboard cube texture map.



Note that you can get a better mirror-like effect if you turn on the lighting and render the surface in white color (use the material diffuse color (1,1,1)).

You may find more information in the following tutorial:
https://www.nvidia.com/object/cube_map_ogl_tutorial

Grading:
- Render a box and an object as shown above figure (10 pts)
- Correctness of reflection (10 pts)
- Virtual trackball to rotate the object only (not the outer box) (10 pts)

Extra points:
- Find interesting cube map textures on the Internet and use them instead of checkerboard textures. You can google "cube map image" (20 pts)

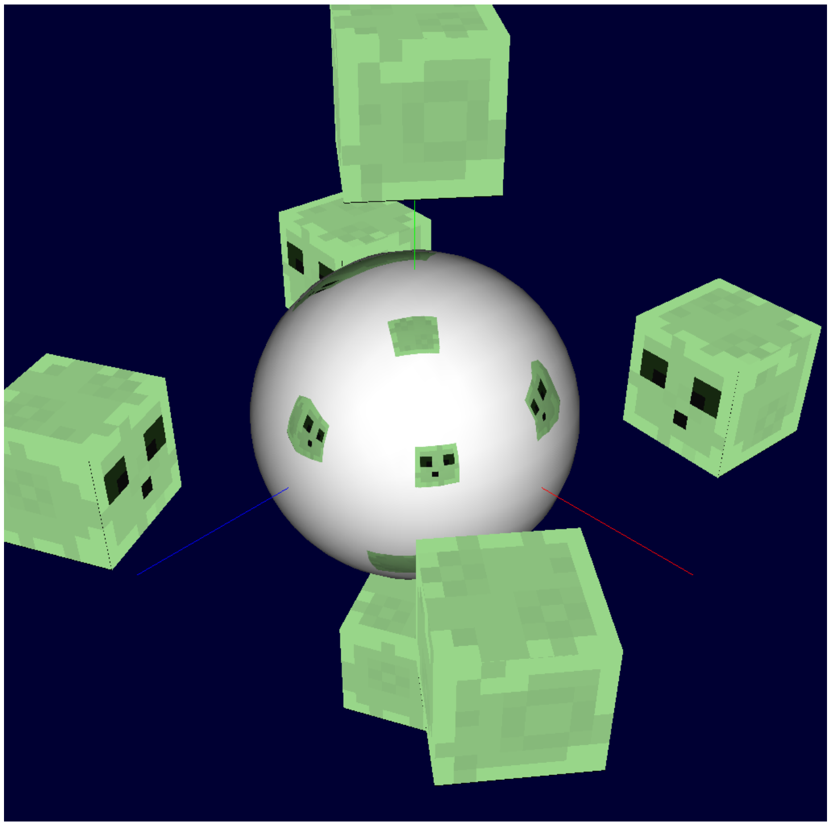
3. Environment mapping using dynamic cube mapping (50 points)

The goal of this problem is to improve the cube map implemented above to render mirror-like object that reflects its surroundings including moving objects. The cube map we used above is created only once and does not change later, so the reflected environment scene does not change. Because we want to render moving objects, we need to update a cube map per every frame to simulate environment reflection correctly.

What you need to do is as follows:

- Place objects and move their position (use idle function in glut to continuously change the location of each object)
- Take six pictures at the origin, along each axis using a FBO to update the cube map texture. Make sure you use 90 degree field of view along each axis so that you can capture a water-tight 360 degree view (use gluPerspective(90, 1, near, far)).
- Render the mirror-like object using cube mapping, together with the moving objects with regular texture mapping

This is a screen-shot of the expected result (You can make your own scene. This is just an example):



You can find more info about FBO in the following page:
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/

Grading:
- Dynamic cube mapping using FBO (25 pts)
- Correctness of reflection (15 pts)
- Animation of objects (10 pts)

Extra points:
- Create at least three complete textured characters (complete means a character having head, body, arms, legs) and use them as moving objects. Use interesting papercraft images found on the Internet (30 pts)

4. Report (10 pts)

Submit a report describing your code, implementation details, and results.

Good luck!!!