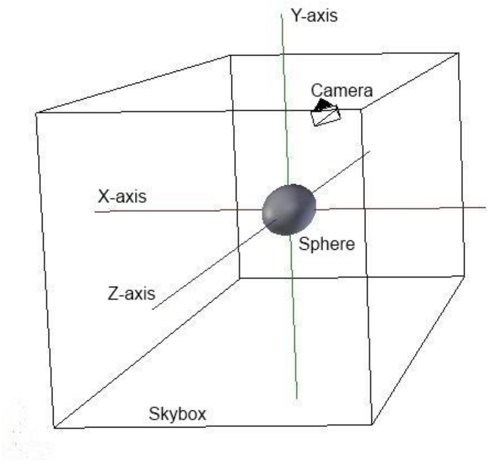
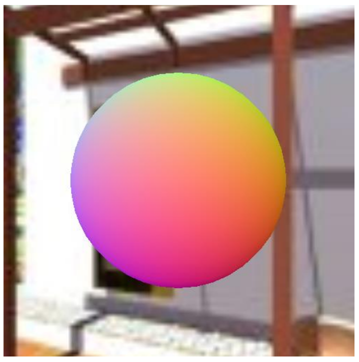
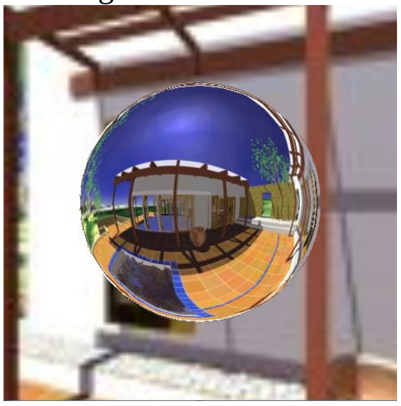
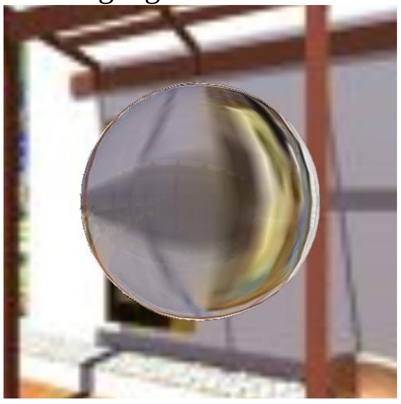
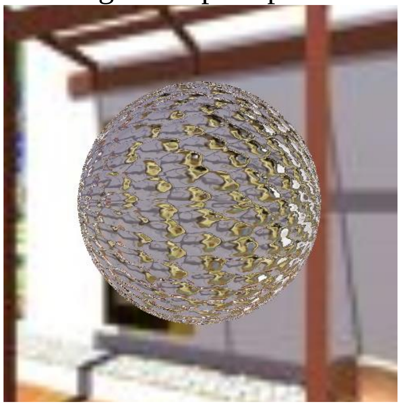
Exercise 10: Environment Mapping and Bump Mapping

Reading	Angel: Angel: chap.7.10
Purpose	The purpose of this exercise is to become familiar with the concepts behind environment mapping and bump mapping. We will use environment mapping to simulate mirrors and glass and in the process learn to use cube maps, reflection/refraction functions and Schlick's approximation. Finally we will apply bump mapping to a sphere to add small scale details.
Setup	 The exercise will use the following setup. A camera rotates around a sphere while looking at the sphere. Around the camera and sphere is a large cube, which will act as a skybox. Mouse drag will rotate the camera. Different shaders are available (mirror, glass and bumpmap) and can be changed by pressing 's'.
Part 1 Loading cube map and create skybox shader. 	We need the skybox shader to show the environment around a sphere object. • Implement the function `initCubemapTexture()` to load the six sides of the cubemap using the function `loadBMPRaw()`. **Hints:** Use chapter 7.9 to see how to build a cubemap texture. Look at the function `initNormalmapTexture()` to see how to use the function `loadBMPRaw()` to load a BMP file. • Modify the skybox shader to lookup in the cubemap – you must use the glsl-function `texture(cubemap, vec3)` to perform the cube-map lookup, where `vec3` is a vector of unit length. Use the object space coordinates as the texture coordinates. You should now see skybox behind the sphere when running the program (see the image to the left).

Exercise 10: Environment Mapping and Bump Mapping

Part 2 Creating a mirror shader 	We will now create a mirror shader for the sphere, where the environment (from the cube map) is reflected. The shader currently shows the normal to the sphere (in world space) computed in the fragment shader. The normal is currently normalized to values between 0.0 and 1.0 for visualization purpose only. • Compute the incident vector in the fragment shader in world space (based on the camera position and the fragment position). Note that the there is no model-transformation used in this exercise. • Compute the reflected vector based on the normal and the incident vector. Use the reflected vector to lookup the color in the cube map. **Hint:** You can use the GLSL function `reflect(vec3 incident, vec3 normal).` You should now see a mirror-like sphere (see image to the left).
Part 3 Creating a glass shader 	We will here create a glass shader that combines reflection with refraction to achieve a glass-like effect. • Modify the glass shader to work just as the mirror shader as a starting point. • Compute the refraction vector and lookup the refraction color in the cube map. **Hint:** You can use the GLSL function `refract(vec3 incident, vec3 normal, float eta)`, where eta is the ratio of indices of refraction. • Use Schlick’s Fresnel approximation to blend between refraction color and the reflection color. You should now see a glass-like sphere when running the program (and changing to glass shader by pressing 's' once). The sphere should be most transparent in the middle.

Exercise 10: Environment Mapping and Bump Mapping

Part 4 Creating a bump-map 	We will here use a bump-map to perturb the normal to give the impression that the surface is 'bumpy'. • Modify the bumpmap shader to work as the glass shader as a starting point. • In the fragment shader compute the texture coordinates used to lookup in the normal map. These texture coordinates can be computed using the spherical coordinates (θ, φ) of the object space position. • Lookup the normal map color in the normal map using (θ, φ) scaled to the [0; 1] range. If you output this color you should see the normal map wrapped around the sphere. • The color found in the normal map is currently defined in the [0,1] range, but we need to transform the color into the [-1,1] range to get the actual normal. • Compute the tangent vectors. The tangent vectors and the geometric normal form an orthonormal basis in which the shading normal is defined. Compute the tangent vectors in the fragment shader by taking the partial derivative of the object space position with respect to each of the spherical coordinates. Organize these basis vectors in a 3×3 matrix. • Finally, compute the bump mapped normal as the product of this matrix and the shading normal. This normal should replace the normal used in the glass shader. You should now see glass-like sphere with a bumpy surface.
Part 5 Optional Chromatic dispersion	Extend the program to add chromatic dispersion. Dispersion is the optical effect responsible for rainbows. It is caused by a wavelength dependent index of refraction. To simulate this, do three lookups in the cube map, one for red, green, and blue, each with different index of refraction, when computing the refraction color.