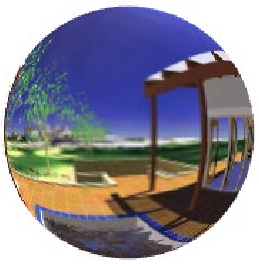
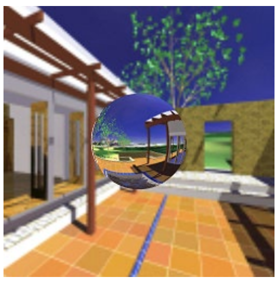
02561 COMPUTER GRAPHICS **DTU COMPUTE**

Worksheet 7: Environment mapping and normal mapping

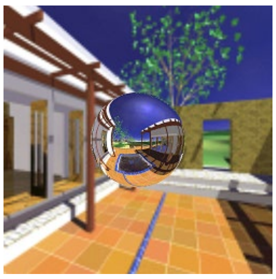
Reading	RTR: Sections 10.4 and 6.7
Purpose	The purpose of this set of exercises is to become familiar with the concepts behind environment mapping and normal mapping. We will use environment mapping to render a curved reflector and, in the process, learn how to use cube maps and the reflection function. We will also use normal mapping to add scale-like surface details.

Part 1
Cube map

	Start from a textured sphere (Part 3 of Worksheet 6). Instead of the ordinary 2D texture, we will now use a cube map to texture the sphere.
	
	Modify your texture initialization such that it loads a cube map from six image files, one file for each face. The files are in `textures.zip` (on DTU Learn). The file names and their orientation in the cube map are:
	`var cubemap = ['textures/cm_left.png',    // POSITIVE_X`
	`               'textures/cm_right.png',   // NEGATIVE_X`
	`               'textures/cm_top.png',     // POSITIVE_Y`
	`               'textures/cm_bottom.png',  // NEGATIVE_Y`
	`               'textures/cm_back.png',    // POSITIVE_Z`
	`               'textures/cm_front.png'];  // NEGATIVE_Z`
	
	If you load the image files in a loop that stores them into an array (`imgs`), you can create a 2d texture with six layers
	`size: [imgs[0].width, imgs[0].height, 6],`
	and use a second loop to copy each image to the right layer of the texture.
	
	Once the cube map is initialized, no inverse map is needed to compute texture coordinates. Simply use the world space normal as texture coordinates when looking up the texture color in the fragment shader. No shading is needed, simply return the texture color.

Part 2 Environment 	The next step is to also draw the environment in the background. To do this, we draw a screen-filling quad very close to the far plane of the view frustum and texture it using the cube map. • A screen-filling quad close to the far plane is most easily drawn using clip coordinates, where the diagonal goes from $(-1,-1,0.999,1)$ to $(1,1,0.999,1)$. Insert this background quad into your scene. • Draw the background quad using the same shaders as in Part 1 but introduce a uniform matrix $\boldsymbol{M}_{\text{tex}}$ in the vertex shader that transforms the vertex position to texture coordinates. • For the sphere, $\boldsymbol{M}_{\text{tex}}$ is an identity matrix. The vertices of the background quad are however in clip space, so its model-view-projection matrix is an identity matrix, but its $\boldsymbol{M}_{\text{tex}}$ should transform from clip space positions to world space directions. Create $\boldsymbol{M}_{\text{tex}}$ for the background quad using (a) the inverse of the projection matrix to go from clip coordinates to camera coordinates and (b) the inverse of the rotational part of the view matrix (no translation) to get direction vectors in world coordinates. Explain the transformation.

Worksheet 7: Environment mapping and normal mapping

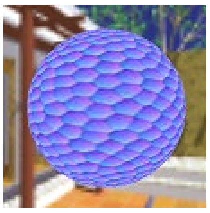
Part 2
Reflection

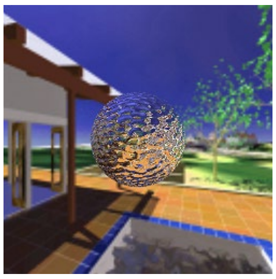


The sphere is not really like a mirror ball. Instead of looking up the environment in the normal direction, we should look up the environment in the direction of reflection.

- Create a uniform variable (`reflective`) to distinguish reflective objects (the mirror ball) from other objects (the background quad).
- Upload the eye position as a uniform variable and compute the direction of incidence (the view vector, $\boldsymbol{v}$) in world space coordinates in the fragment shader.
- Use the `select` function (or an `if`-statement) to choose the direction of reflection as texture coordinates for reflective objects.
  Hint: You can use the built-in WGSL function
  `fn reflect(incident: vec3f, normal: vec3f) -> vec3f`

Part 4
Bump mapping





Finally, we will perturb the normal of the mirror ball using a normal map to give the impression that the ball surface is 'bumpy'.

- Load the normal map texture from the file `textures/normalmap.png`. Map it onto the sphere using the same technique as in Part 3 of Worksheet 6. The first image for this part is the expected result.
- Bind the normal map to another texture so that it can be used together with the cube map. The color found in the normal map is in $[0,1]^3$. Transform it to be in $[-1,1]^3$ to get the actual normal.
- The normal retrieved from the normal map is in tangent space. We need to transform it to world space to use it in place of the sphere normal when calculating the direction of reflection. Use the following function as an efficient way to perform this change of basis transformation:[1]

```
fn rotate_to_normal(n: vec3f, v: vec3f) -> vec3f
{
    let sgn_nz = sign(n.z + 1.0e-16);
    let a = -1.0/(1.0 + abs(n.z));
    let b = n.x*n.y*a;
    return vec3f(1.0 + n.x*n.x*a, b, -sgn_nz*n.x)*v.x
           + vec3f(sgn_nz*b, sgn_nz*(1.0 + n.y*n.y*a), -n.y)*v.y
           + n*v.z;
}
```

  The first argument is the surface normal `n` in world coordinates, the second argument is the tangent space vector to be transformed to world space. In our case, the tangent space vector is the normal retrieved from the normal map. The transformation returns the bump mapped normal to be used in place of the sphere normal when rendering reflective objects.

[1] Frisvad, J. R. Building an orthonormal basis from a 3D unit vector without normalization. *Journal of Graphics Tools 16*(3), pp. 151-159, August 2012. https://people.compute.dtu.dk/jerf/code/hairy/
Duff, T., Burgess, J., Christensen, P., Hery, C., Kensler, A., Liani, M., and Villemin, R. Building an orthonormal basis, revisited. *Journal of Computer Graphics Techniques 6*(1), pp. 1-8. March 2017.