02561 Computer Graphics

Environment mapping and normal mapping

Jeppe Revall Frisvad

October 2023

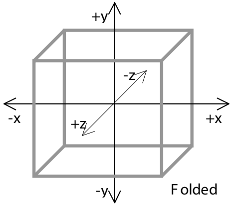
Exercise: Loading a cube map (W07P1)

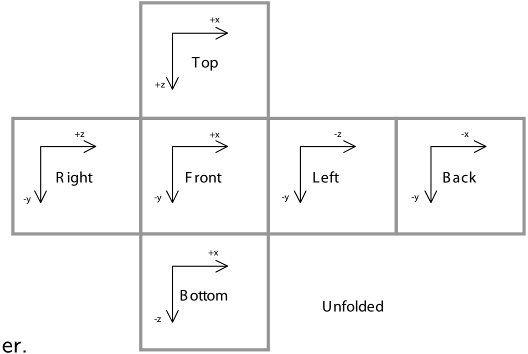
```
var g_tex_ready = 0;

function initTexture(gl)
{
  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

  gl.activeTexture(gl.TEXTURE0);
  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

  for(var i = 0; i < 6; ++i) {
    var image = document.createElement('img');
    image.crossorigin = 'anonymous';
    image.textarget = gl.TEXTURE_CUBE_MAP_POSITIVE_X + i;
    image.onload = function(event)
    {
      var image = event.target;
      gl.activeTexture(gl.TEXTURE0);
      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.texImage2D(image.textarget, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
      ++g_tex_ready;
    };
    image.src = cubemap[i];
  }
  gl.uniform1i(gl.getUniformLocation(gl.program, "texMap"), 0);
}
```





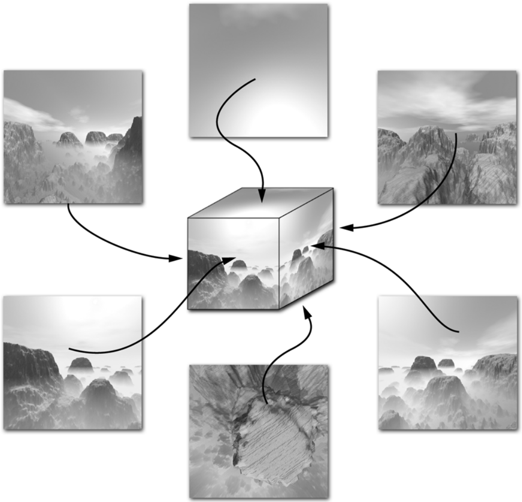
- Do not render if `g_tex_ready < 6`.
- Use a `samplerCube` uniform variable and `textureCube` for look-up in the fragment shader.
- Use the world space surface normal as texture coordinates to get started.

Environment mapping

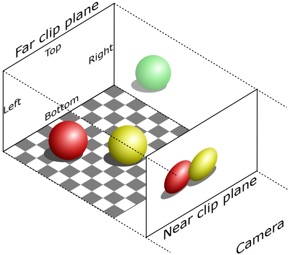
- Environment mapping:

  Map an omnidirectional image onto everything surrounding the scene.
- Cube mapping:

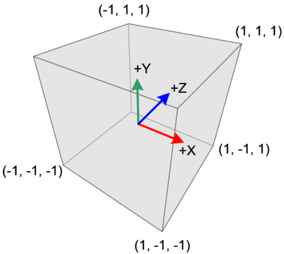
  Use a direction to perform look-ups into an omnidirectional image consisting of six texture images (square resolution, 90° field of view).
- Look-ups return the light $L_{\text{env}}(\vec{\omega})$ received from the environment when looking in the direction $\vec{\omega}$.
- Look-up directions $\vec{\omega}$ should be in world space (but normalization is not required).



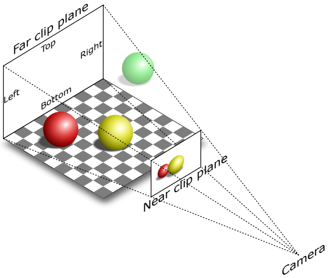
Drawing a frame-filling quad at the far plane



orthographic projection



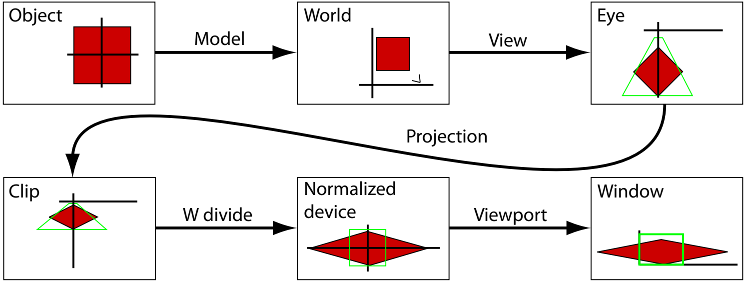
NDC cube



perspective projection

- In normalized device coordinates (NDC), a quad with vertices $(-1,-1,0.999,1),(1,-1,0.999,1),(-1,1,0.999,1),(1,1,0.999,1)$ always fills out the frame and is always hindmost.
- Set model, view, and projection matrices to identity matrices (`mat4()`) to draw using NDC coordinates.
- We can draw a frame-filling quad at the far back to activate the fragment shader for all background pixels.

Exercise: Filling the background (W07P2)

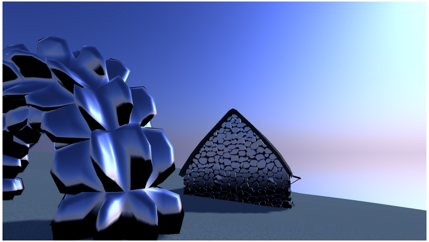


- Given the vertex position in normalized device coordinates $\boldsymbol{p}_n$, we find the direction of a ray going through the pixel in world space using

$$\vec{i}_w = \begin{bmatrix} & & & 0 \\ & (\boldsymbol{V}^{-1})^{3\times 3} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{P}^{-1}\,\boldsymbol{p}_n\,, \qquad (\boldsymbol{V}^{-1})^{3\times 3} \text{ is the upper left } 3\times 3 \text{ part of the inverse view matrix.}$$

- Since $\vec{i}_w$ is the texture coordinates that we need for the environment map, we find this by applying a texture matrix $\boldsymbol{M}_{\text{tex}}$ to the vertex position: $\vec{i}_w = \boldsymbol{M}_{\text{tex}}\boldsymbol{p}_n$.

Exercise: Reflection (W07P3)

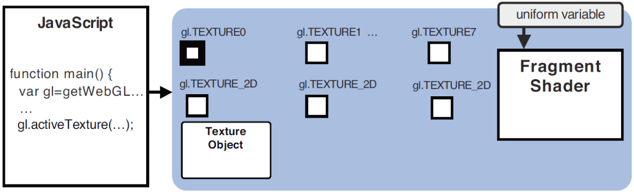


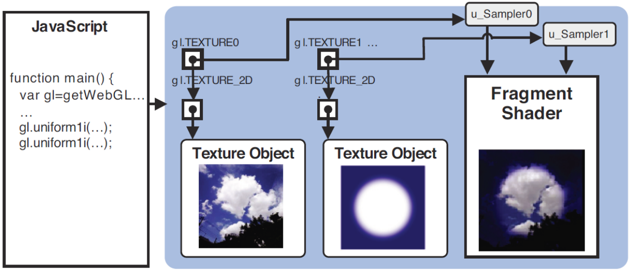
- When rendering an object, we have
  $\boldsymbol{M}_{\text{tex}} = \boldsymbol{I}$ (identity matrix)
  and model, view, and projection as usual.

- Reflective environment mapping:
  $\vec{i}_w = \frac{\boldsymbol{p}_w - \boldsymbol{e}}{\|\boldsymbol{p}_w - \boldsymbol{e}\|}\,, \quad \vec{r}_w = \vec{i}_w - 2\left(\vec{n}_w \cdot \vec{i}_w\right)\vec{n}_w\,.$
- where $\boldsymbol{e}$ is eye position, $\boldsymbol{p}_w$ and $\vec{n}_w$ are world space position and normal of the fragment, and $\vec{r}_w$ is the direction of the reflected ray.
- Use `reflect(`$\vec{i}_w$`, `$\vec{n}_w$`)` in the shader to find $\vec{r}_w$
  (the normalization of $\vec{i}_w$ is not required).
- We now need a uniform variable to indicate whether the shader should use reflection or not (as the background quad should not reflect).

Multitexturing

- Texture units: `gl.TEXTURE0, ..., gl.TEXTURE7`
- Texture targets: `gl.TEXTURE_2D`, `gl.TEXTURE_CUBE_MAP`
- Texture objects: `gl.createTexture()`
- Texture samplers (example): `uniform sampler2D tex;`
- Select a texture unit using `gl.activeTexture`.
- Bind a texture object to a target for the currently selected texture unit using `gl.bindTexture`.
- Set the unit to be used by a texture sampler in a shader using `gl.uniform1i(loc,` *i*`);`

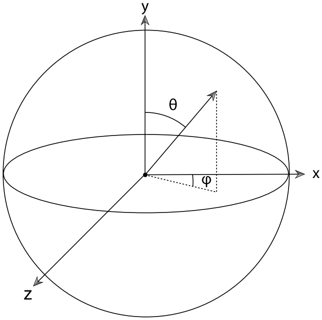




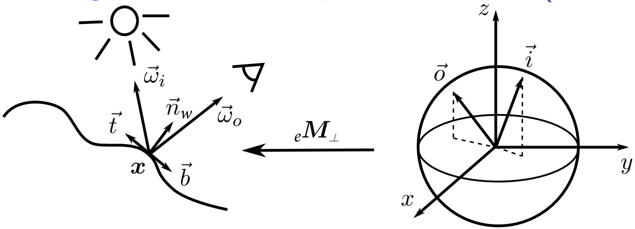
(the number *i* links to TEXTURE*i*)

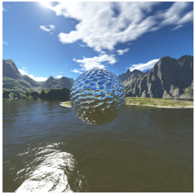
Spherical inverse mapping

- Spherical coordinates provide a uv-mapping of the unit sphere: $(u, v) = \left(1 - \frac{\varphi}{2\pi}, \frac{\theta}{\pi}\right)$.
- The corresponding Euclidean space coordinates are: $(x, y, z) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$.
- Alternatively, if we want y to be the up direction, as in eye space (and in the figure to the right), then $(x, y, z) = (\sin\theta\cos\varphi, \cos\theta, \sin\theta\sin\varphi)$.
- Inserting u and v, we have the uv-mapping
$$(x, y, z) = (\sin(\pi v)\cos(2\pi(1-u)), \cos(\pi v), \sin(\pi v)\sin(2\pi(1-u)))\,.$$
- The inverse mapping provides texture coordinates given a position on the unit sphere (such as a surface normal).
- We have $y = \cos(\pi v)$ and $\frac{z}{x} = \tan(2\pi(1-u))$, then $u = 1 - \frac{\tan^{-1}\frac{z}{x}}{2\pi} = 1 - \frac{\text{atan}(z,x)}{2\pi}$ and $v = \frac{\cos^{-1} y}{\pi} = \frac{\text{acos}(y)}{\pi}$.



Exercise: Tangent to world space rotation (W07P4)





- Suppose we look up a texture value $\boldsymbol{c}$ from a normal map.
- The normals from the normal map are in tangent space: $\vec{n}_\perp = 2\boldsymbol{c} - 1$.
- We need to transform the vectors to world space $\vec{n}_{\text{bump}} = {}_e\boldsymbol{M}_\perp\,\vec{n}_\perp$
- The change of basis matrix is ${}_e\boldsymbol{M}_\perp = \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix}$
- A helper function finding $\vec{n}_{\text{bump}}$ given $\vec{n}_\perp$ and the normal in world coordinates $\vec{n}_w$:

```
vec3 rotate_to_normal(vec3 n, vec3 v) {
  float sgn_nz = sign(n.z + 1.0e-12);
  float a = -1.0/(1.0 + abs(n.z));
  float b = n.x*n.y*a;
  return vec3(1.0 + n.x*n.x*a, b, -sgn_nz*n.x)*v.x
       + vec3(sgn_nz*b, sgn_nz*(1.0 + n.y*n.y*a), -n.y)*v.y
       + n*v.z;
}
```

$$\vec{n}_{\text{bump}} = \texttt{rotate\_to\_normal}(\vec{n}_w, \vec{n}_\perp)$$
$$\boldsymbol{r}_w = \texttt{reflect}(\boldsymbol{p}_w - \boldsymbol{e}, \vec{n}_{\text{bump}})\,.$$

WebGPU environment mapping using cube map textures

- Excellent resources on WebGPU texturing:
  - https://webgpufundamentals.org/webgpu/lessons/webgpu-textures.html
  - https://webgpufundamentals.org/webgpu/lessons/webgpu-importing-textures.html
  - https://webgpufundamentals.org/webgpu/lessons/webgpu-cube-maps.html

- Kenwright has a chapter on WebGPU environment mapping using cube mapping:
  - Benjamin Kenwright. Environment Mapping (Cube Mapping). In *Web Programming Using the WebGPU API: A Practical Guide with Examples*. ACM SIGGRAPH 2023 Courses, Chapter 12, pp. 105-118. August 2023.

- Another code sample is available here:
  - https://webgpu.github.io/webgpu-samples/samples/cubemap