Computer Graphics
Environment Background

László Szécsi szecsi@iit.bme.hu

AIT

Compute ray direction

- ray from eye through pixel in world space, which is, normalized, the
- vector from eye to pixel, which is, interpolated, the
- vector from eye to vertex of full-viewport quad, which can be computed using a matrix

Ray direction from normalized device coords

$$\boldsymbol{x}_{\mathrm{w}}\boldsymbol{VP} = \boldsymbol{x}_{\mathrm{ndc}}$$

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{w}} - \boldsymbol{e}$$

$$\hat{\boldsymbol{d}} = \frac{\boldsymbol{d}}{|\boldsymbol{d}|}$$

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{VP})^{-1} - \boldsymbol{e}$$

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{VP})^{-1}\boldsymbol{E}^{-1}$$

$$\boldsymbol{d} = \boldsymbol{x}_{\mathrm{ndc}}(\boldsymbol{EVP})^{-1}$$

(**EVP**)$^{-1}$ is henceforth called rayDirMatrix

Display environment as a background

- draw full viewport quad (hurray!)
- new VS: computes ray direction
    - must take matrix that computes world-space-cords-minus-eye-position from ndc (a.k.a. `camera.rayDirMatrix`)
        - camera must compute this matrix
    - does no transformation (being a full viewport quad)
    - z=0.99999, behind everything
- FS gets ray direction from VS
    - addresses cube texture
    - returns color from texture

Cube texture

- need a uniform in FS

```
// need a sample uniform
uniform struct { samplerCube envTexture; } material;
```

```
// read from ray direction
fragmentColor = texture ( material.envTexture, rayDir.xyz);
```

- in `Scene` create `TextureCube`, pass it to the FS through its `Material`, with a `GameObject` using the material (geometry is textured quad)

```
this.envTexture = new TextureCube(gl, [
    "media/posx512.jpg",
    "media/negx512.jpg",
    "media/posy512.jpg",
    "media/negy512.jpg",
    "media/posz512.jpg",
    "media/negz512.jpg",]
    );
```

```
this.backgroundMaterial.envTexture.set(this.envTexture);
```

TextureCube.js – loading images

```
"use strict";
/* exports TextureCube */
class TextureCube {
  constructor(gl, mediaFileUrls) {
    gl.pendingResources[mediaFileUrls[0]] = ++gl.pendingResources[mediaFileUrls[0]] || 1;
    this.mediaFileUrls = mediaFileUrls;
    this.glTexture = gl.createTexture();
    this.loadedCount = 0;
    this.images = [];
    for(let i=0; i<6; i++){
      this.images[i] = new Image();
      this.images[i].onload = () => {
                             this.loaded(gl); }
      this.images[i].src = mediaFileUrls[i];
    }
  }
```

TextureCube.js – resource creation

```
 loaded(gl){
  this.loadedCount++;
  if(this.loadedCount < 6) {
   return;
  }
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, this.glTexture);
  for(let i=0; i<6; i++){
   gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X+i, 0,
            gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, this.images[i]);
  }
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);
  gl.generateMipmap(gl.TEXTURE_CUBE_MAP);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, null);
  if( --gl.pendingResources[this.mediaFileUrls[0]] === 0 ) {
   delete gl.pendingResources[this.mediaFileUrls[0]];
  }
 }
}
```

Do not forget to

- include new shaders in index.HTML
- create required `Shader`, `TexturedProgram`, `Material` objects
- set the cube texture to the backgound material
- create a `TexturedQuadGeometry`
- create `Mesh` using the above material and geometry
- create a `GameObject` using the above mesh