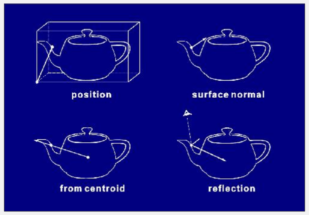
Environment Mapping

Using cubemaps for environment mapping –

- render the skybox (with cubemap applied)
- render the objects, using the skybox as a map shape
  - compute texture coordinates as the reflection vector from the object point



http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Rendering the Skybox

- vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
attribute vec3 coords;
varying vec3 v_objCoords;
void main() {
    vec4 eyeCoords = modelview * vec4(coords,1.0);
    gl_Position = projection * eyeCoords;
    v_objCoords = coords;
}
```

- fragment shader

```
precision mediump float;
varying vec3 v_objCoords;
uniform samplerCube skybox;
void main() {
      gl_FragColor = textureCube(skybox, v_objCoords);
}
```

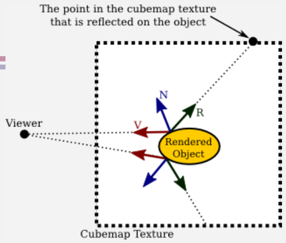
cubemap textures are sampled using 3D vector from the origin to the OC point on the cube
(the vector does not need to be normalized)

- `draw`

draw a large cube, centered at the origin, enclosing the entire scene (including the camera)

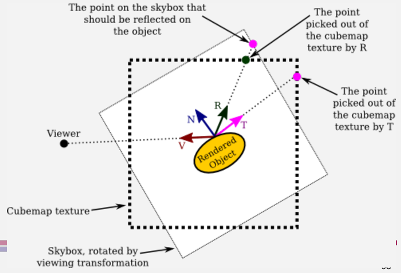
Rendering the Objects

reflection vector R points to the skybox point to use – it is what to use to sample the cubemap


The point in the cubemap texture that is reflected on the object
Viewer
N
R
V
Rendered Object
Cubemap Texture

if the view is rotated, the skybox must be rotated to compensate – the object will reflect a different point

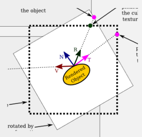
apply the inverse of the viewing transform to R in order to find the correct spot on the cubemap T


The point on the skybox that should be reflected on the object
The point picked out of the cubemap texture by R
The point picked out of the cubemap texture by T
Viewer
N
R
T
V
Rendered Object
Cubemap texture
Skybox, rotated by viewing transformation

Rendering the Objects

- vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
attribute vec3 coords;
attribute vec3 normal;
varying vec3 v_eyeCoords;
varying vec3 v_normal;
void main() {
    vec4 eyeCoords = modelview * vec4(coords,1.0);
    gl_Position = projection * eyeCoords;
    v_eyeCoords = eyeCoords.xyz;
    v_normal = normalize(normal);
}
```



- fragment shader

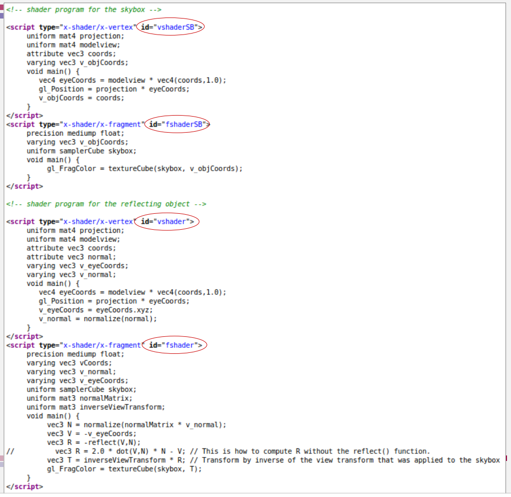
```
precision mediump float;
varying vec3 vCoords;
varying vec3 v_normal;
varying vec3 v_eyeCoords;
uniform samplerCube skybox;
uniform mat3 normalMatrix;
uniform mat3 inverseViewTransform;
void main() {
      vec3 N = normalize(normalMatrix * v_normal);
      vec3 V = -v_eyeCoords;
      vec3 R = -reflect(V,N);
      vec3 T = inverseViewTransform * R;
      gl_FragColor = textureCube(skybox, T);
}
```

javascript computation of inverse viewing transform
modelview is just the viewing transform at this point →

```
mat3.fromMat4(inverseViewTransform, modelview);
mat3.invert(inverseViewTransform,inverseViewTransform);
```

Using Different Shader Programs

- give separate IDs



```
<!-- shader program for the skybox -->

<script type="x-shader/x-vertex" id="vshaderSB">
     uniform mat4 projection;
     uniform mat4 modelview;
     attribute vec3 coords;
     varying vec3 v_objCoords;
     void main() {
        vec4 eyeCoords = modelview * vec4(coords,1.0);
        gl_Position = projection * eyeCoords;
        v_objCoords = coords;
     }
</script>
<script type="x-shader/x-fragment" id="fshaderSB">
     precision mediump float;
     varying vec3 v_objCoords;
     uniform samplerCube skybox;
     void main() {
          gl_FragColor = textureCube(skybox, v_objCoords);
     }
</script>

<!-- shader program for the reflecting object -->

<script type="x-shader/x-vertex" id="vshader">
     uniform mat4 projection;
     uniform mat4 modelview;
     attribute vec3 coords;
     attribute vec3 normal;
     varying vec3 v_eyeCoords;
     varying vec3 v_normal;
     void main() {
        vec4 eyeCoords = modelview * vec4(coords,1.0);
        gl_Position = projection * eyeCoords;
        v_eyeCoords = eyeCoords.xyz;
        v_normal = normalize(normal);
     }
</script>
<script type="x-shader/x-fragment" id="fshader">
     precision mediump float;
     varying vec3 vCoords;
     varying vec3 v_normal;
     varying vec3 v_eyeCoords;
     uniform samplerCube skybox;
     uniform mat3 normalMatrix;
     uniform mat3 inverseViewTransform;
     void main() {
          vec3 N = normalize(normalMatrix * v_normal);
          vec3 V = -v_eyeCoords;
          vec3 R = -reflect(V,N);
//          vec3 R = 2.0 * dot(V,N) * N - V; // This is how to compute R without the reflect() function.
          vec3 T = inverseViewTransform * R; // Transform by inverse of the view transform that was applied to the skybox
          gl_FragColor = textureCube(skybox, T);
     }
</script>
```

Using Different Shader Programs

- create programs for both in `initGL`

```
prog_SB = createProgram(gl, "vshaderSB", "fshaderSB");


prog = createProgram(gl, "vshader", "fshader");
```

- switch programs as needed in `draw`

```
gl.useProgram(prog_SB);
```

Limitations

- objects only reflect the skybox, not other objects rendered in the scene
  - need to generate the skybox texture by rendering the scene with the background skybox and objects