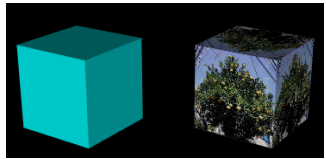
02561 Computer Graphics

Shadow mapping and off-screen buffers

Jeppe Revall Frisvad

November 2019

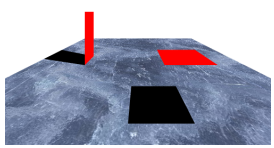
Drawing objects with different shaders



- Implement different shaders
  (use different html IDs or glsl files).
- Create different programs (using `initShaders`).
- When rendering, do the following for each object:
  1. Specify the program to be used (using `gl.useProgram`).
  2. Bind vertex buffer layers and enable attributes (using `initAttributeVariable`).
  3. Set uniform variables (view matrix and likewise, if they differ from the initial state).
  4. Draw the geometry (using `gl.drawArrays` or `gl.drawElements`, for example).
- `initAttributeVariable` requires two extra data fields in each buffer object:
  - `buffer.num` is the number of coordinates (usually 2, 3, or 4).
  - `buffer.type` is the type of the coordinates (usually `gl.FLOAT`).

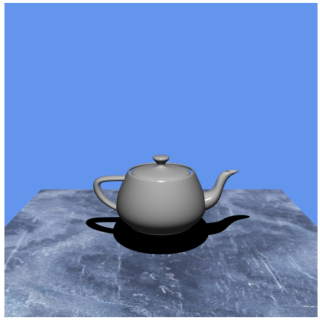
```
function initAttributeVariable(gl, attribute, buffer)
{
  gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
  gl.vertexAttribPointer(attribute, buffer.num, buffer.type, false, 0, 0);
  gl.enableVertexAttribArray(attribute);
}
```

Exercise: Jumping teapot on marble table top (W09P1)



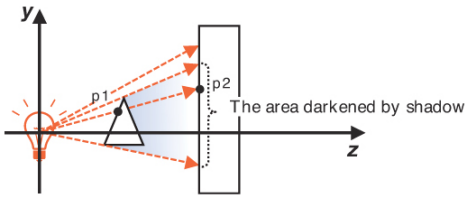
+

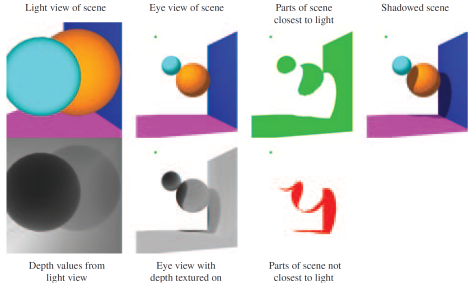
→



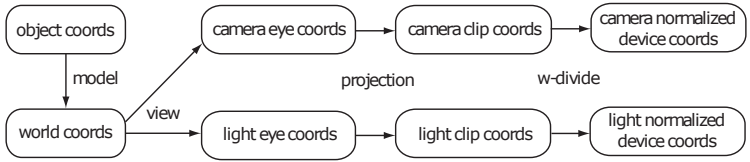
- Starting from the result of Worksheet 8, replace the two shadow casting quads by the Newell teapot (loaded from an obj file as in Worksheet 5).
- The ground plane and the teapot need different shaders.
- Create buttons for switching light circulation and vertical teapot movement on/off.

Shadow mapping



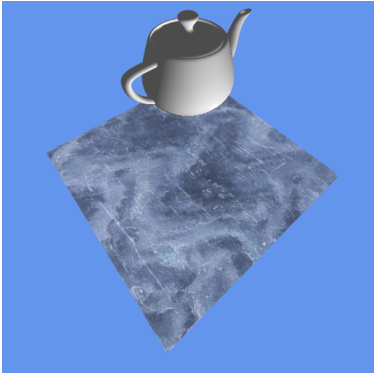


- Render the scene from the point of view of the light source.
- Store position/depth $\boldsymbol{p}_1$ seen from the light source in a shadow map (a texture).
- For each observed fragment, find the corresponding fragment in the shadow map.
- Check if the observed fragment position $\boldsymbol{p}_2$ is significantly different from $\boldsymbol{p}_1$.

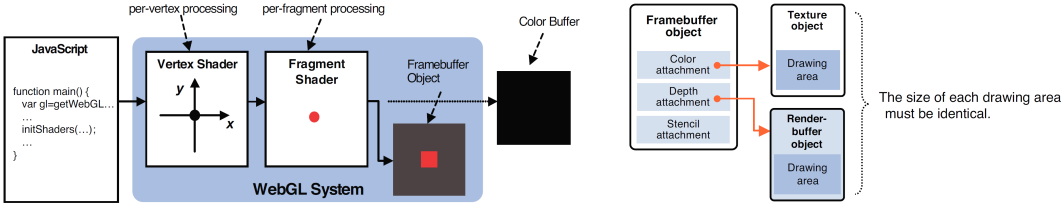


Exercise: Rendering from the point of view of the light source (W09P2a)

- The light source needs its own projection and view matrices: $\boldsymbol{P}_\ell$ and $\boldsymbol{V}_\ell$.
  - Directional light: orthographic projection matrix.
  - Point light: perspective projection matrix.
- Use $\boldsymbol{P}_\ell$ and $\boldsymbol{V}_\ell$ to make the view volume a tight bound around the scene.



Frame buffer object (render-to-texture)




```
function initFramebufferObject(gl, width, height)
{
  var framebuffer = gl.createFramebuffer(); gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
  var renderbuffer = gl.createRenderbuffer(); gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
  gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16, width, height);

  var shadowMap = gl.createTexture(); gl.activeTexture(gl.TEXTURE0); gl.bindTexture(gl.TEXTURE_2D, shadowMap);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, width, height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  framebuffer.texture = shadowMap;

  gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, shadowMap, 0);
  gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);
  var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
  if (status !== gl.FRAMEBUFFER_COMPLETE) { console.log('Framebuffer object is incomplete: ' + status.toString()); }
  gl.bindFramebuffer(gl.FRAMEBUFFER, null); gl.bindRenderbuffer(gl.RENDERBUFFER, null);
  framebuffer.width = width; framebuffer.height = height;
  return framebuffer;
}
```

Exercise: Drawing to an off-screen buffer (W09P2b)

- Initialize the frame buffer object:

```
var fbo = initFramebufferObject(gl, texwidth, texheight);
```

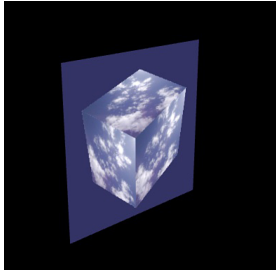
- When rendering:

```
gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
gl.viewport(0, 0, fbo.width, fbo.height);

//... code inserted here renders to the texture in the fbo ...

gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

- Render the scene to a texture and
  apply the rendered texture to the ground quad.



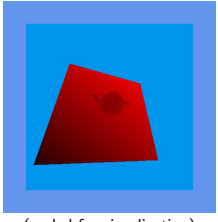
Drawing depth and considering numerical precision

- In the fragment shader, the depth value is always available in `gl_FragCoord.z` .
- The depth value z_{frag} is the z-coordinate in NDC space scaled to $[0, 1]$:

$$z_{\text{frag}} = \frac{1}{2}\frac{p_{\text{clip},z}}{p_{\text{clip},w}} + \frac{1}{2} \quad , \qquad \boldsymbol{p}_{\text{clip}} = \boldsymbol{PV}\,\boldsymbol{p}_{\text{world}} \, .$$

- We can write a simple fragment shader outputting the depth as red color:

```
precision mediump float;
void main()
{
  gl_FragColor = vec4(gl_FragCoord.z, 0.0, 0.0, 0.0);
}
```



(scaled for visualization)

- Unfortunately, a color has only 8-bit precision.
- We can use floating point bit shifting to better this problem:

```
precision mediump float;
void main()
{
    const vec4 bitShift = vec4(1.0, 256.0, 256.0*256.0, 256.0*256.0*256.0);
    const vec4 bitMask = vec4(1.0/256.0, 1.0/256.0, 1.0/256.0, 0.0);
    vec4 rgbaDepth = fract(gl_FragCoord.z*bitShift);
    rgbaDepth -= rgbaDepth.gbaa*bitMask;
    gl_FragColor = rgbaDepth;
}
```

Exercise: Shadow mapping in the fragment shader (W09P2c)

- What are the texture coordinates to be used for the shadow map?
- Again, we need the NDC fragment position scaled to $[0, 1]$:

$$(u_\ell, v_\ell, z_\ell, 1) = \frac{1}{2}\frac{\boldsymbol{p}_\ell}{p_{\ell,w}} + \frac{1}{2} \quad , \qquad \boldsymbol{p}_\ell = \boldsymbol{P}_\ell \boldsymbol{V}_\ell \, \boldsymbol{p}_{\text{world}} \, .$$

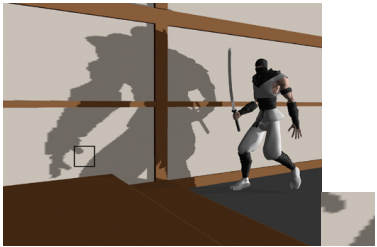
- The texture coordinates are then (u_ℓ, v_ℓ) and the depth from the light is `vec4 rgbaDepth = texture2D(shadowMap, shadowCoord.xy);`
- For the 8-bit precision shadow map, we have: `float depth = rgbaDepth.r;` and the fragment is in shadow if $z_\ell >$ `depth + 0.005`.
- The reason for `0.005` is the 8 bits: $2^{-8} + \epsilon = 0.005$.
- For the version with bit shift, we have: `float depth = unpackDepth(rgbaDetph);`

```
float unpackDepth(const in vec4 rgbaDepth) {
  const vec4 bitShift = vec4(1.0, 1.0/256.0, 1.0/(256.0*256.0), 1.0/(256.0*256.0*256.0));
  return dot(rgbaDepth, bitShift);
}
```

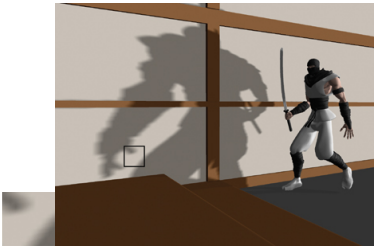
- One should think this would give 32-bit precision, but since `gl_FragCoord` is defined `mediump` we only have 10-bit precision. New threshold: $2^{-10} + \epsilon = 0.0015$.

Percentage-closer filtering

- Aliasing (staircase/pixelation/jaggies) is a significant issue in shadow mapping.
- Sampling an area of 4-by-4 texels is a significant improvement.



0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1



- We need a look-up with an offset in texture space:

```
vec4 offset_lookup(sampler2D map, vec3 shadowCoord, vec2 offset) {
  return texture2D(map, shadowCoord.xy + offset*texmapscale);
}
```

where `texmapscale` is a uniform `vec2` with `1/fbo.width` and `1/fbo.height`.

- Michael Bunnell and Fabio Pellacini. Shadow map antialiasing. In *GPU Gems*. Chapter 11. Addison-Wesley, 2004. https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch11.html