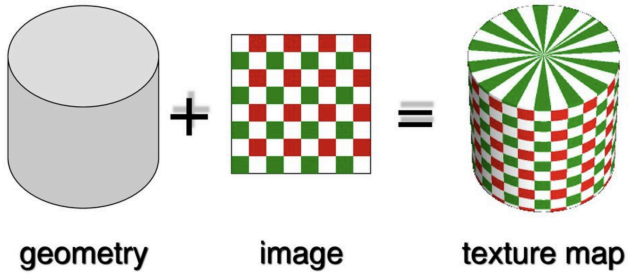
November 6 class lecture notes

Video of interactive explanation in blended reality

https://vimeo.com/232230096

Texture mapping



Security problem:

From a browser, we can't access local files.

So how do we load texture images?

We need to run a server:

```
python3 -m http.server &
```

We need to run the web page as localhost:

```
http://localhost:8000
```

We need to define a u,v value at every pixel. We do this as follows.

Change vertex length from 6 to 8:

Each vertex = x,y,z, nx,ny,ny, u,v

```
let vertexSize = 8;
```

Append u,v to every vertex:

```
let createMesh = (nu, nv, p) => {
   let V = (u,v) => p(u,v).concat([u, v]);
   let mesh = [];
```

```
    for (let j = nv-1 ; j >= 0 ; j--) {
       for (let i = 0 ; i <= nu ; i++)
          mesh.push(V(i/nu,(j+1)/nv),  V(i/nu,j/nv));
       mesh.push(V(1,j/nv), V(0,j/nv));
    }
    return new Float32Array(mesh.flat());
}
```

Other changes to the code go in several places:

Define a new vertex attribute aUV

```
vertexAttribute('aPos', 3, 0);
vertexAttribute('aNor', 3, 3);
vertexAttribute('aUV' , 2, 6);
```

Copy aUV to vUV in vertex shader

```
attribute vec2 aUV;
…
vUV = aUV;
```

Define sampler in fragment shader

```
uniform sampler2D uSampler;
…
vec4 texture = texture2D(uSampler, vUV);
…
vec3 color = sqrt(uColor * c) * texture.rgb;
```

Declare uSampler in Javascript

```
gl.uniform1i(uSampler, 0);
```

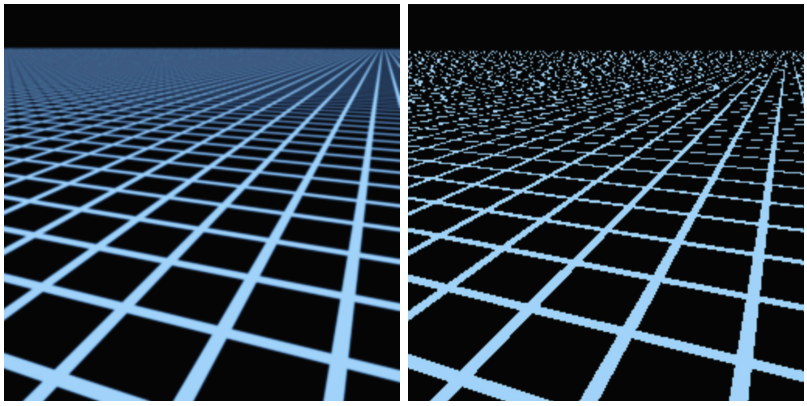
Loading a texture image:

```
let texture = (index, fileName) => {
   let image = new Image();
   image.onload = () => {
      gl.activeTexture (gl.TEXTURE0 + index);
      gl.bindTexture   (gl.TEXTURE_2D, gl.createTexture());
      gl.texImage2D    (gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      gl.texParameteri (gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
      gl.texParameteri (gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
      gl.generateMipmap(gl.TEXTURE_2D);
   }
```

```
        image.src = fileName;
    }
```

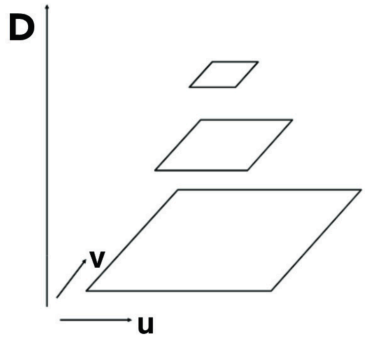
Mip-mapping

We want to avoid undersampling / jaggies



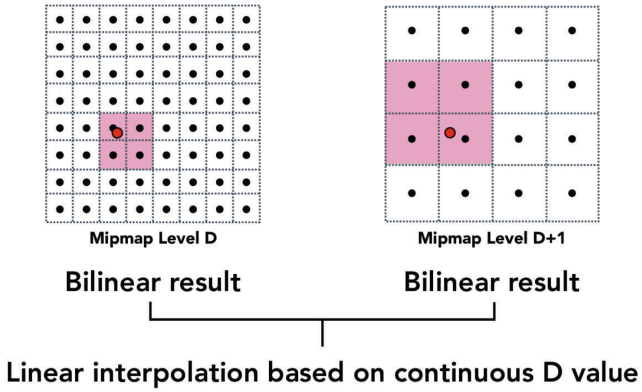
Great solution: Lance Williams (1983) – Mip-mapping

(1) Make a power-of-two image pyramid
(2) Do trilinear interpolation over u, v, and level of Detail



"Mip hierarchy"
level = D

Trilinear Filtering



What if we want multiple textures?

Declare `uSampler` in Javascript as an array

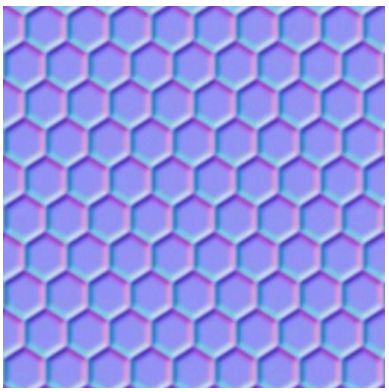
```
gl.uniform1iv(uSampler, [0,1,2,....,15]);
```

Find the right sampler in fragment shader using a loop

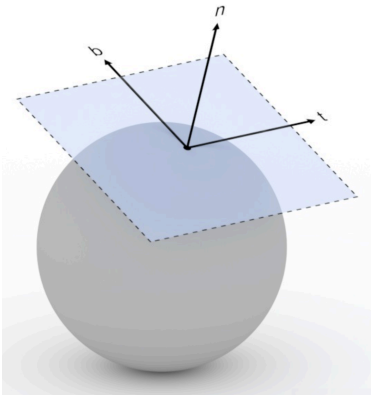
```
uniform sampler2D uSampler[16];
varying vec2 vUV;
…
vec4 texture;
for (int i = 0 ; i < 16 ; i++)
   if (uTexture == i)
      texture = texture2D(uSampler[i], vUV);
…
vec3 color = sqrt(uColor * c) * texture.rgb;
gl_FragColor = vec4(color, uOpacity * texture.a);
```

Bump mapping

In an r,g,b image, encode variation in normal as a function of u,v:



Red = vary **t**angent
Green = vary **b**inormal
Blue = vary **n**ormal



```
varying vec3 vPos, vNor, vTan;
…
vec3 nor = normalize(vNor);
for (int i = 0 ; i < 16 ; i++)
    if (uBumpTexture == i) {
       vec3 b = 2. * texture2D(uSampler[i], vUV).rgb - 1.;
       vec3 tan = normalize(vTan);
       vec3 bin = cross(nor, tan);
       nor = normalize(b.x * tan + b.y * bin + b.z * nor);
    }
```

We need to add a tangent attribute to every vertex:

Each vertex = x,y,z, nx,ny,nz, u,v, tx,ty,tz

```
vertexSize = 11;

let createMesh = (nu, nv, p) => {
   let V = (u,v) => {
      let P = p(u,v);
      let Q = p(u+.001,v);
      let x = Q[0]-P[0], y = Q[1]-P[1], z = Q[2]-P[2],
          s = Math.sqrt(x*x + y*y + z*z);
      return P.concat([u, v, x/s, y/s, z/s]);
   }
   let mesh = [];
   for (let j = nv-1 ; j >= 0 ; j--) {
      for (let i = 0 ; i <= nu ; i++)
         mesh.push(V(i/nu,(j+1)/nv),  V(i/nu,j/nv));
      mesh.push(V(1,j/nv), V(0,j/nv));
```

```
    }
    return new Float32Array(mesh.flat());
}
```

The vertex shader now looks like this:

```
    attribute vec3 aPos, aNor, aTan;
    attribute vec2 aUV;
    uniform mat4 uMatrix, uInvMatrix;
    varying vec3 vPos, vNor, vTan;
    varying vec2 vUV;
    void main() {
        vec4 pos = uMatrix * vec4(aPos, 1.0);
        vec4 nor = vec4(aNor, 0.0) * uInvMatrix;
        vec4 tan = vec4(aTan, 0.0) * uInvMatrix;
        vPos = pos.xyz;
        vNor = nor.xyz;
        vTan = tan.xyz;
        vUV  = aUV;
        gl_Position = pos * vec4(1.,1.,-.1,1.);
    }
```

Using a Web canvas as a texture source

If you want to create textures that animate over time, one way to do that is to use an HTML canvas as your texture source. In folder `texture3` I show how to do that. As I show in that example, you can set the content of the canvas at every animation frame by writing to the canvas via a 2D drawing context.

When using a canvas as your texture source, the most important implementation difference is that we need to send the texture to the GPU at every animation frame, so that the rendered texture will change over time. This is obviously more expensive because of the extra data traffic between the CPU and the GPU, but it gives you a powerful way to create animated textures.