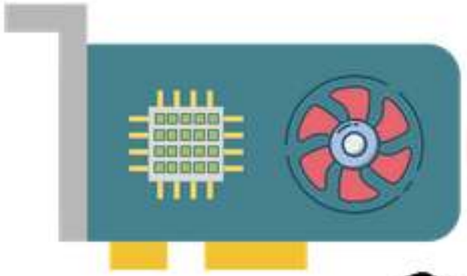
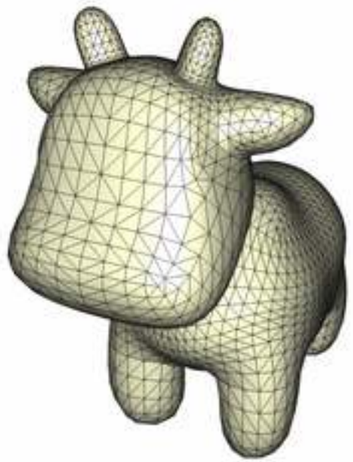
Things we know how to do right now.



WebGL™



varyings

gl_Position gl_FragColor

VS → rasterizer → FS

draw

buffer

mesh

vertices

triangles

attributes (a_Position)

varyings (v_Position)

uniforms (u_ModelMatrix)

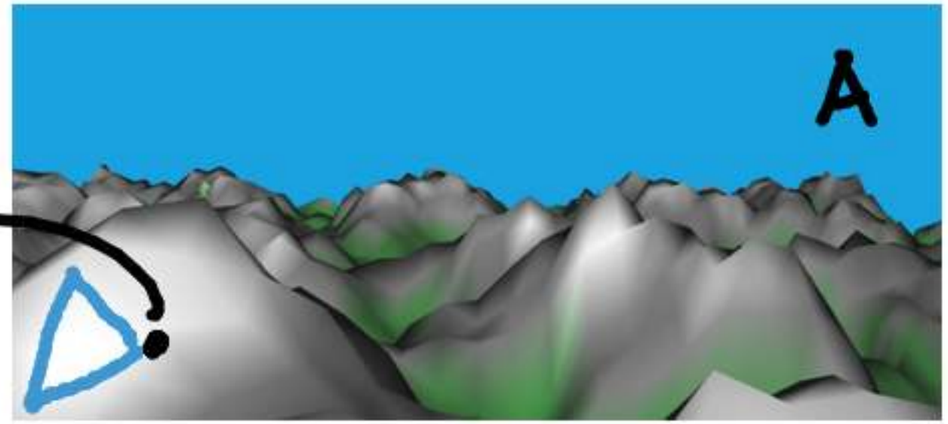
Recap on how we have been painting our models.

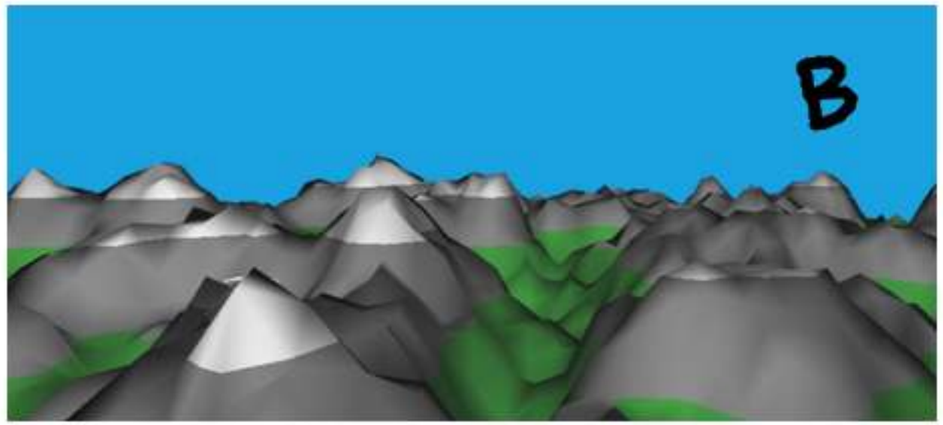
JS

if (h > 0.8)
  show
else if (h > 0)
  rock
else
  lava

color
(a-color)

$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$



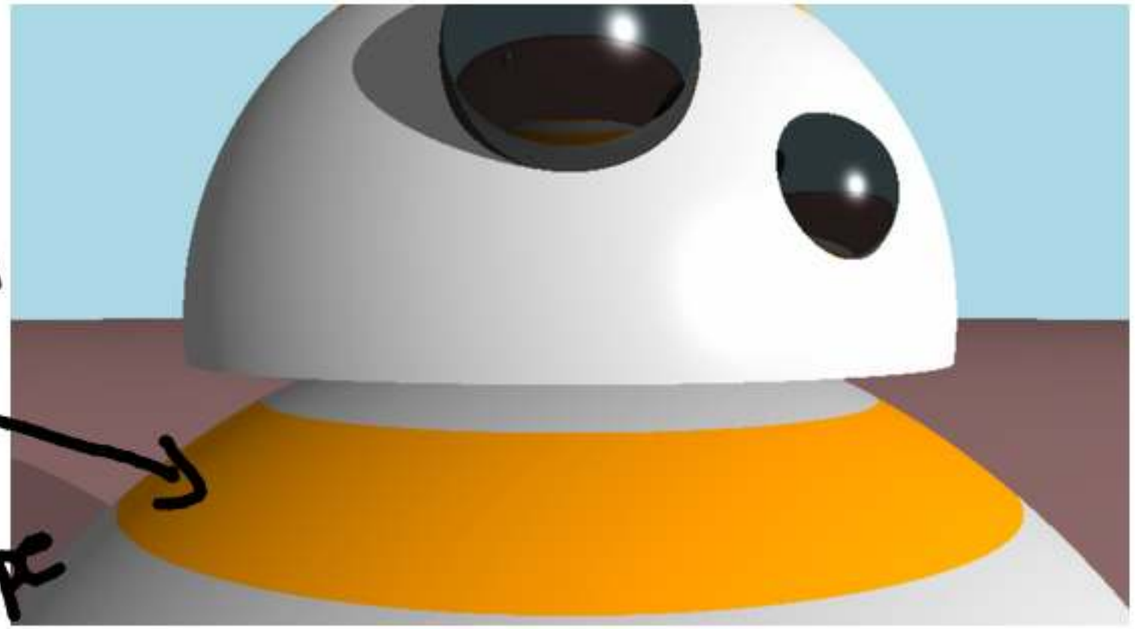


pass h to FS

if-else directly in FS
↓
Km

Other places we have seen this? Think about week 3.

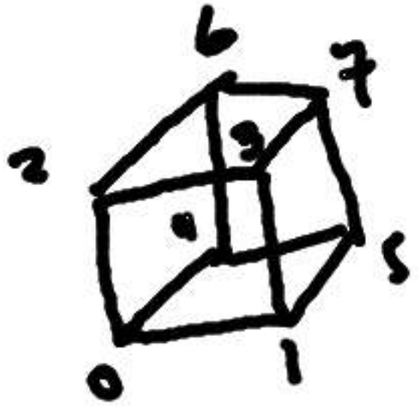
$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$



"procedural" texturing

Warmup: can we use a `Uint8Array` instead? (event # 9000136)

```
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(mesh.triangles), gl.STATIC_DRAW);
```



For our cube (8 vertices) from last class, can we buffer triangles indices (using gl.bufferData) using a Uint8Array? 23

Yes, but something else is missing. 48%

No, 8 bits is not enough for this number of vertices. 22%

Yes, we can just change this to Uint8Array. 22%

I'm not sure. 9%

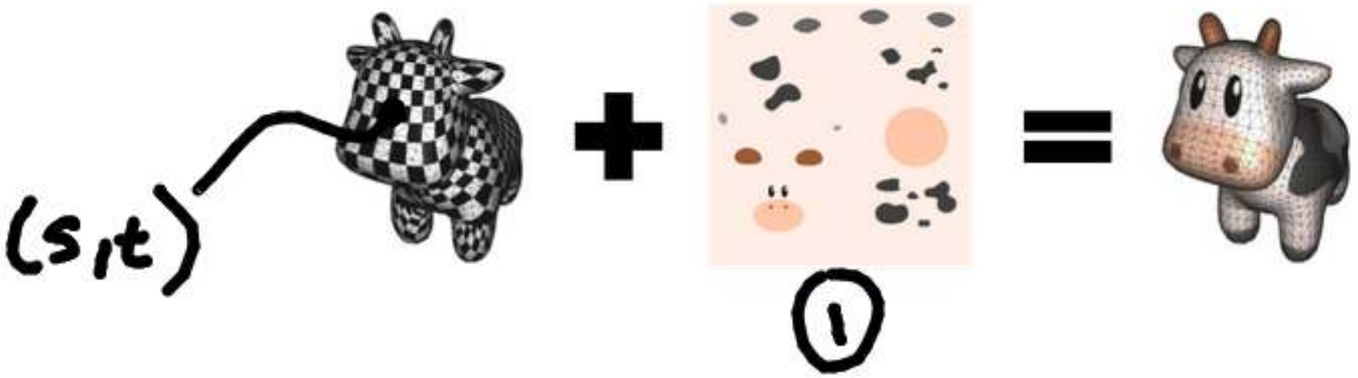
Edit response

gl.UNSIGNED_BYTE

```
gl.drawElements(gl.TRIANGLES, mesh.triangles.length, gl.UNSIGNED_SHORT, 0);
```

16 bits

The main idea of texturing: sample an image to determine surface properties.



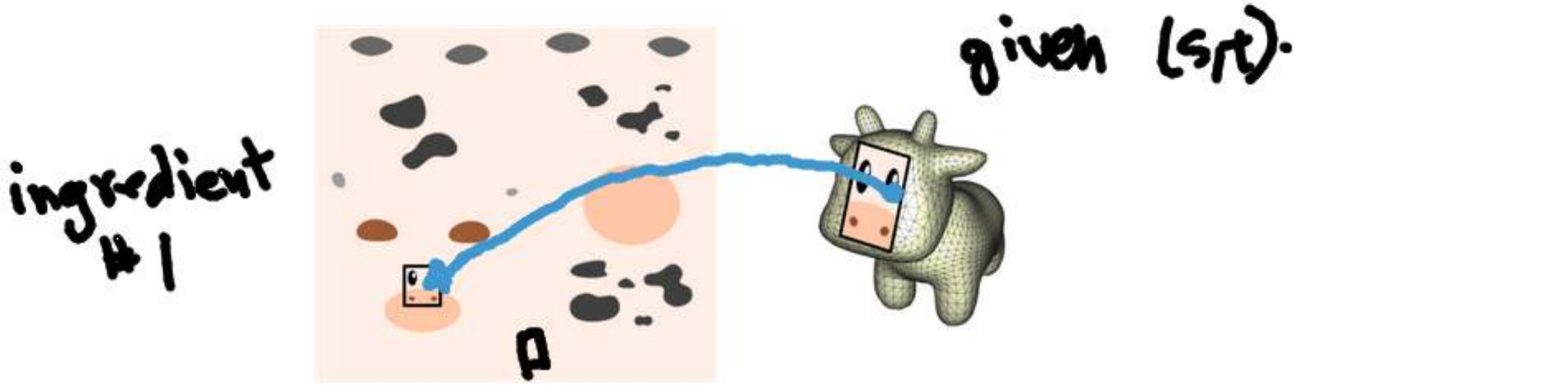
three ingredients:

① image

② texture coordinates (s,t)

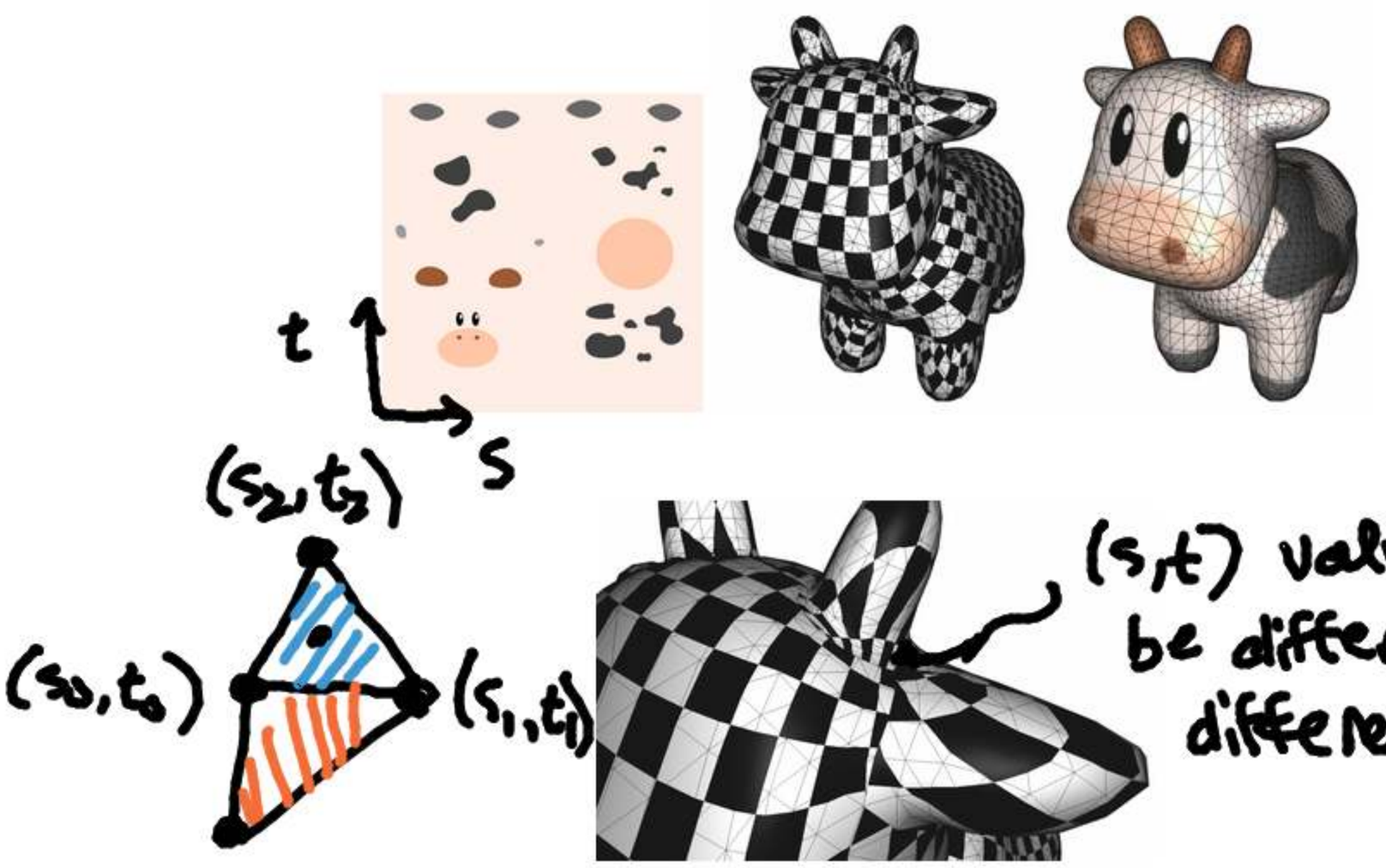
③ how we look up "texel"

Ingredients #1 and #3: which *texel* to sample?



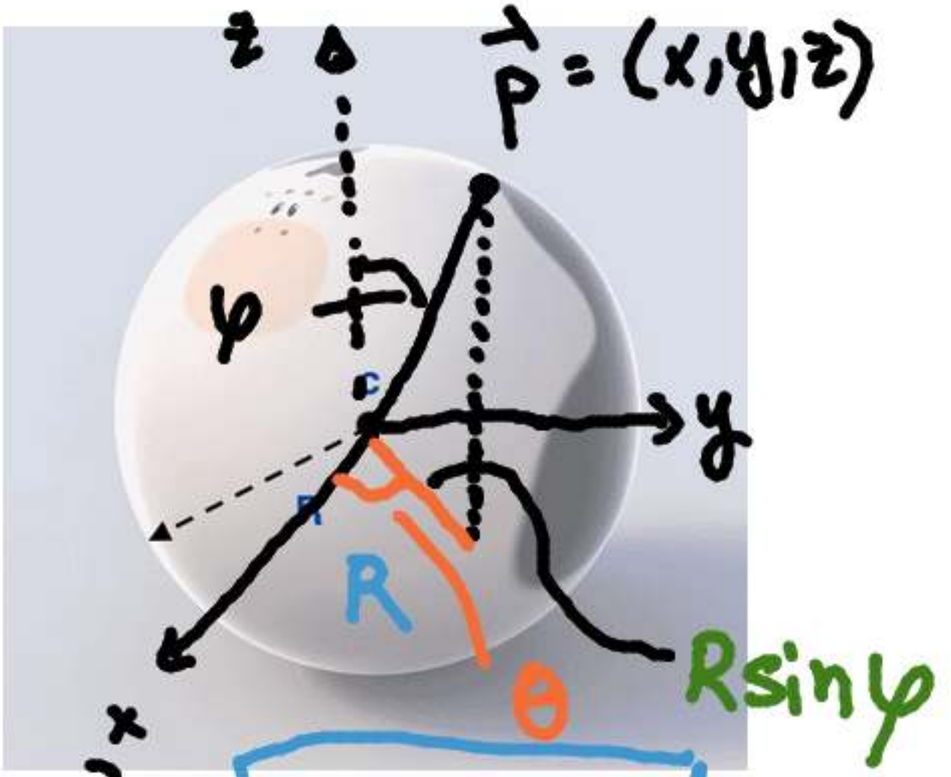
for now, assume we use nearest texel (center)

Ingredient #2: we need *texture coordinates*.



A special case: texture coordinates on a sphere.

Why? approximate lots of things like spheres!



compute (s,t) using (θ,φ)

$z = R\cos\varphi$

$x = R\sin\varphi\cos\theta$

$y = R\sin\varphi\sin\theta$

$\varphi = \arccos(z/R)$ in range $[0,\pi]$

$\theta = \operatorname{atan}(y,x)$ in range $[-\pi,\pi]$

$(s,t) \in [0,1]^2$

$$s = \frac{\theta + \pi}{2\pi}$$

$$t = \frac{\varphi}{\pi}$$

Doing it with WebGL.

In JavaScript:

```
1  // retrieve the image
2  let image = document.getElementById("spot-texture");
3
4  // create the texture and activate it
5  let texture = gl.createTexture();
6  gl.activeTexture(gl.TEXTURE0); // <-- important! make a note of N in gl.TEXTUREN
7  gl.bindTexture(gl.TEXTURE_2D, texture);
8
9  // define the texture to be that of the requested image
10 gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, gl.RGB, gl.UNSIGNED_BYTE, image);
11
12 // set filter parameters
13 gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
14 gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
15
16 // tell webgl which texture index to use for the uniform sampler2D in the shader
17 gl.uniform1i(gl.getUniformLocation(program, 'tex_Image'), 0);
```

In shader:

```
1  precision highp float;
2
3  uniform sampler2D tex_Image;
4
5  void main() {
6    float s = ...;
7    float t = ...;
8    vec3 km = (texture2D(tex_Image, vec2(s, t))).rgb;
9    gl_FragColor = vec4(km, 1.0);
10 }
```

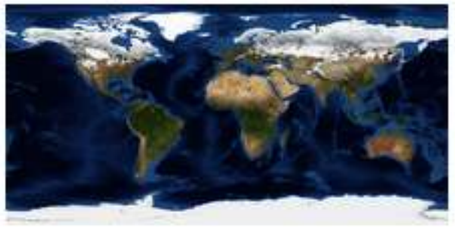
Other properties to modify with textures?

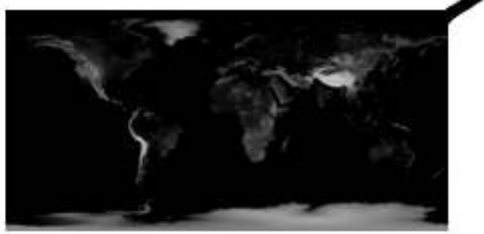
$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$

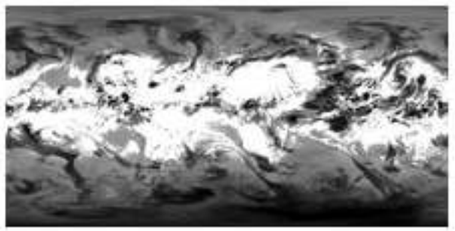
Km

displace

clouds

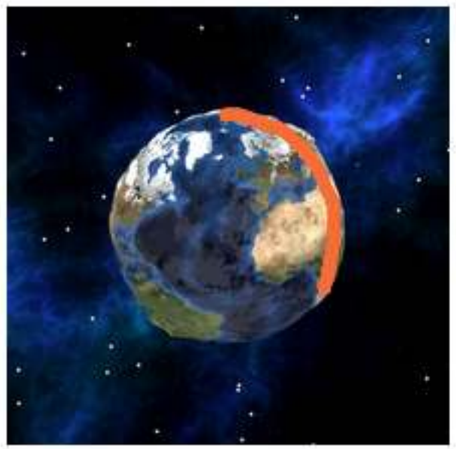






h

a-Position → $\vec{p}$



multiply by MVP matrix to get gl_Pos...

$\vec{p}' = \vec{p} + h\vec{n}$

$= \vec{p}(1+h)$

$\vec{n} = \vec{p} - \vec{c}$

$(0,0,0)$