POLITECNICO
MILANO 1863

Textures

Computer Graphics 2021

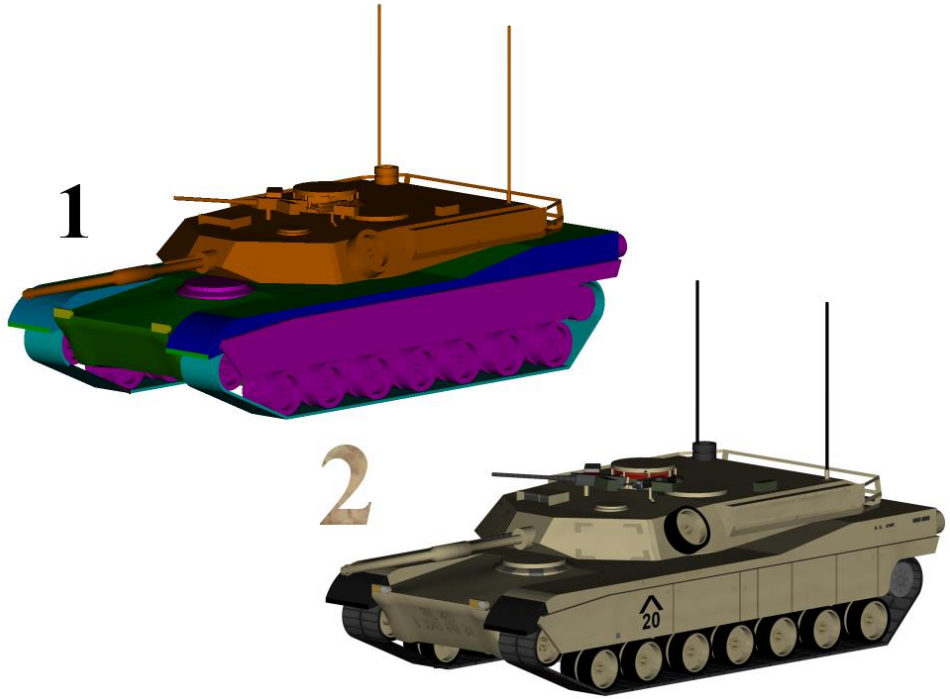
Erica Stella (erica.stella@polimi.it)

How to make the scene more realistic?

- We saw that lighting improves the appearance of the rendering but…

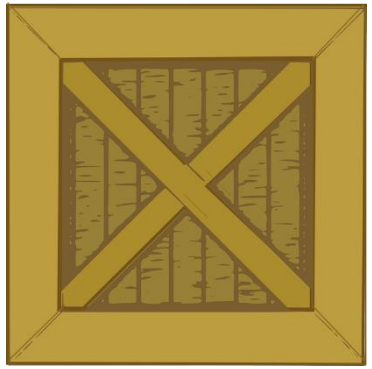
- Real objects shows complex color patterns that cannot be represented by a simple color for each vertex.

- **Textures** (or **texture map**) are images applied to the 3D model to make its appearance more realistic

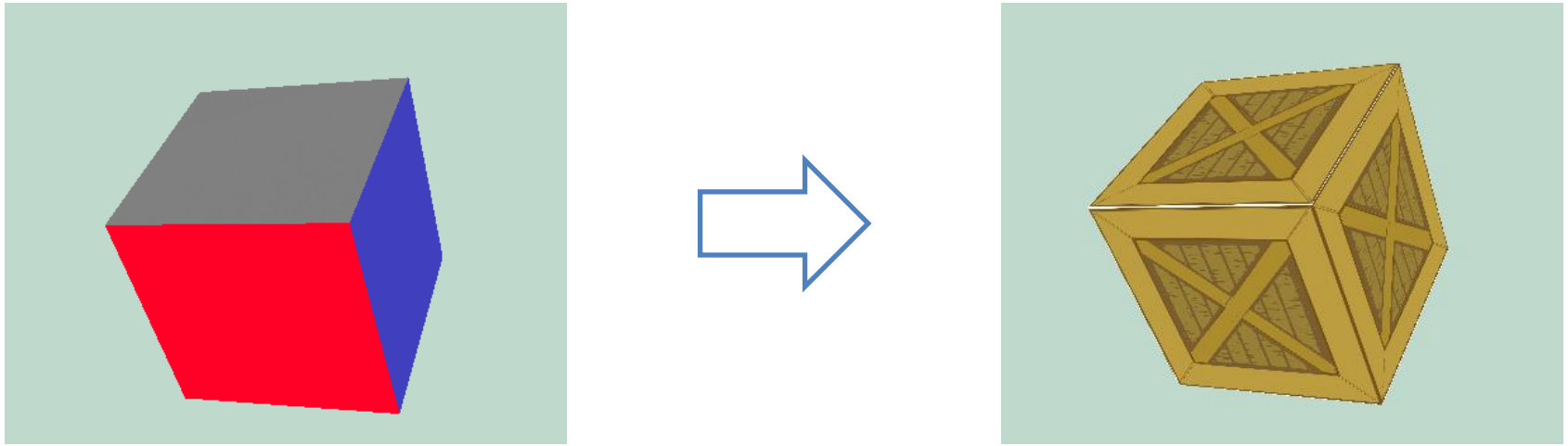


Example: Cube with Texture

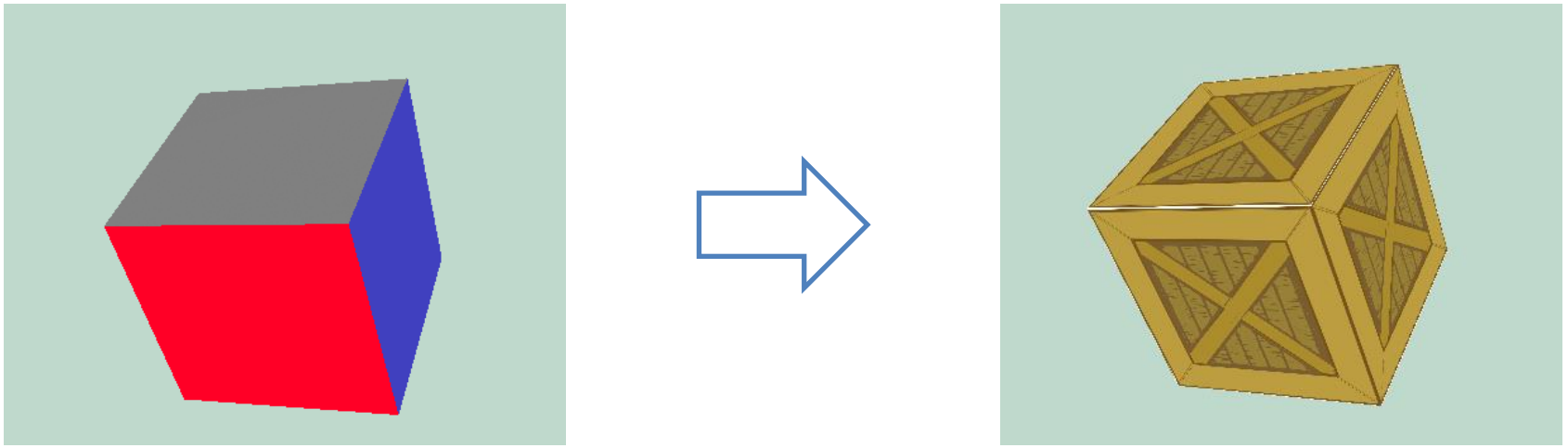
Given the cube we used in the previous lectures, we want to apply this texture



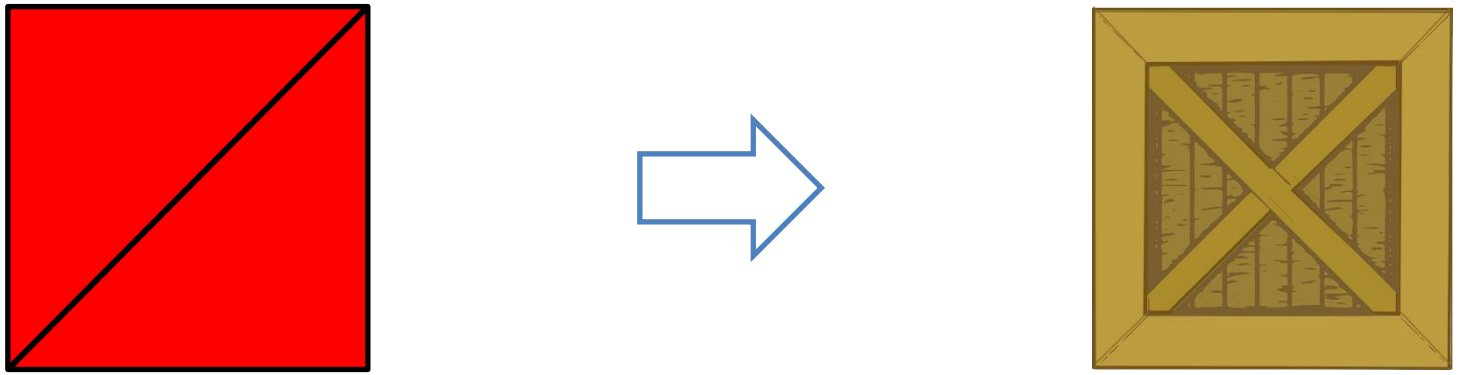
to its faces, to obtain something like this:



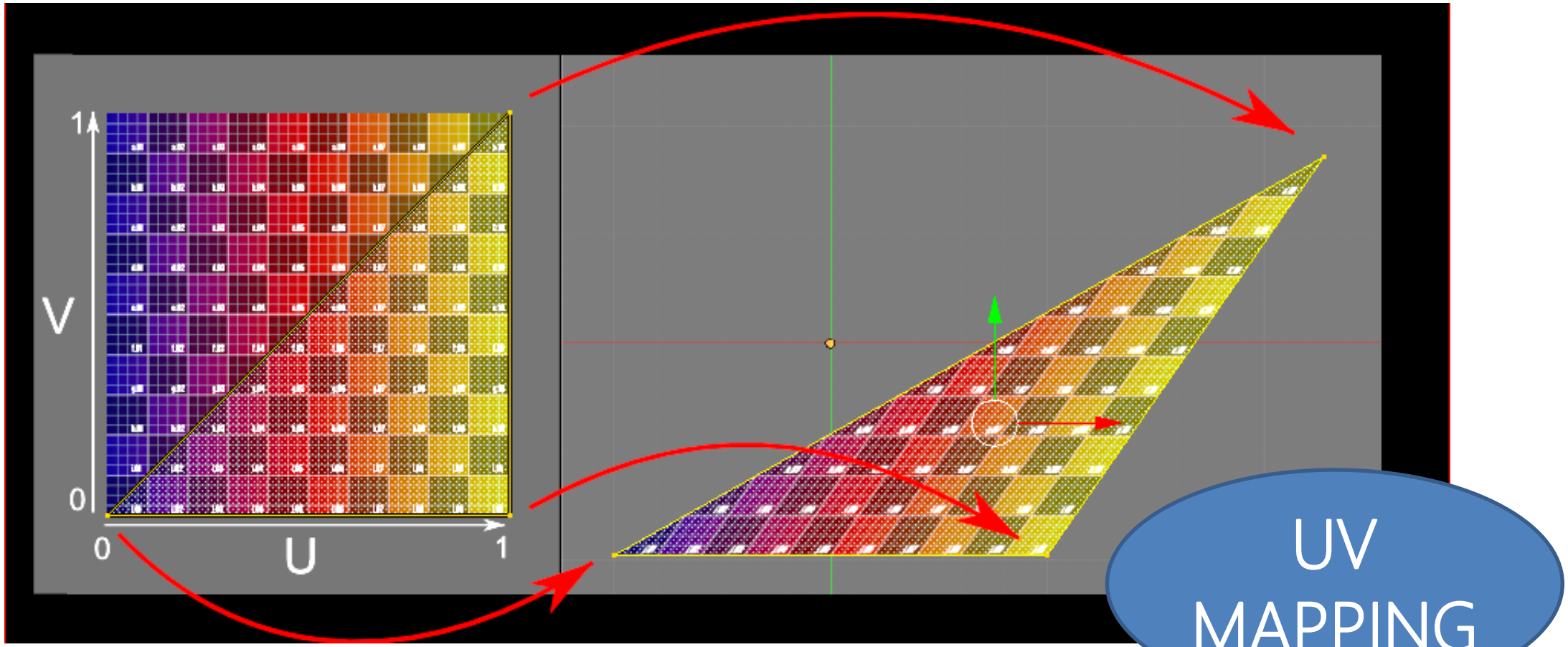
How Vertices Get Mapped into Textures



For each fragment of each facet we need to tell WebGL where to sample the color from the image



How Vertices Get Mapped into Textures



Example: Cube with Texture

The steps are:

1. Load the image into the javascript code (client)
2. Pack the image into a dedicated WebGL object
3. Send the texture image to the GPU
4. Send to the shaders the mapping from the coordinates of the image to the vertices (uv coordinates)
5. Render the cube

Load image

- To load image files in WebGL we can use the HTML **Image()** object.

```
var image=new Image();
```

- We will use two properties of the **`Image`** object:

```
image.src = image_URL;
image.onload = function(e){};
```

Url of the image

Function called once the image is loaded

Pack and Send the Image into a WebGL Texture Object

- Once the data are loaded from the file:
  - Create the texture object
  - And set it as the current active
  - Bind it to the active slot

```
var texture=gl.createTexture();
//In WebGL there are (at least) 8 texture slots, all subsequent
//function modifying the state will happen on the active slot
//Slots are numbered gl.TEXTUREi, e.g., gl.TEXTURE1, gl.TEXTURE2
//Starting from 0
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture); //Bound to slot 0
```

Pack and Send the Image into a WebGL Texture Object

- Then, we can pack the data with the **`texImage2D`** function and set some parameters

```
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true); //WebGL has inverted uv coordinates
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.generateMipmap(gl.TEXTURE_2D);
```

- Flip image in the y direction to match the coordinate system of WebGL and **`Image`**()

```
 gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
```

Pack and Send the Image into a WebGL Texture Object

```
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
```

- **`texImage2D`** function loads the image data in the texture object (in the GPU):
  - `gl.TEXTURE_2D` target texture type (or `gl.TEXTURE_CUBE_MAP`)
  - `0` level of the texture (mipmap) (always 0 at the creation)
  - `gl.RGBA` format of the input image used
  - `gl.RGBA` format of the resulting texture (also **`gl.RGB`**)
  - `gl.UNSIGNED_BYTE` type of data defining the image
  - `image` source of the data to create the texture with

Pack and Send the Image into a WebGL Texture Object

- Define how textures are interpolated whenever their size needs to be incremented or diminished

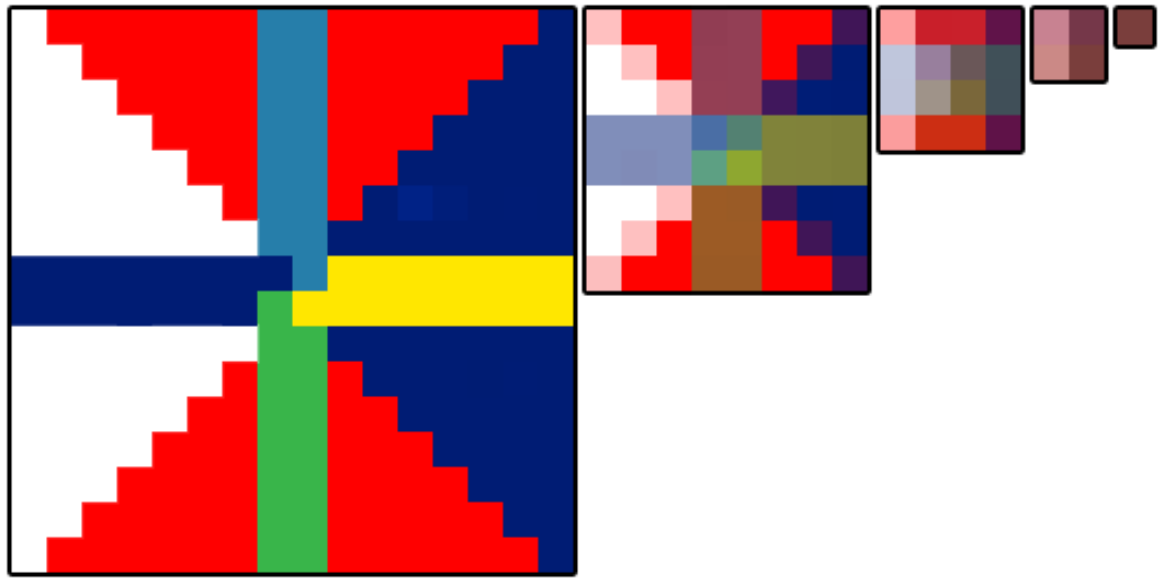
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

  - `gl.TEXTURE_2D,` target texture type
  - `gl.TEXTURE_MAG_FILTER,` set properties for the magnification filter.
  - `gl.TEXTURE_MIN_FILTER,` set properties for the minification filter
    - `gl.LINEAR` enable bi-linear interpolation
    - With `MAG_FILTER` other valid values are: `gl.NEAREST;`
    - With `MIN_FILTER` other valid value: `gl.NEAREST, gl.LINEAR_MIPMAP_LINEAR, gl.NEAREST_MIPMAP_NEAREST, gl.NEAREST_MIPMAP_LINEAR, gl.LINEARMIPMAP_NEAREST;`

Pack and Send the Image into a WebGL Texture Object

- Enable the generation of mipmap, which are smaller copies of your texture sized down and filtered in advance

```
gl.generateMipmap(gl.TEXTURE_2D);
```



Send the UV mapping

UV coordinates associated to each vertex can be sent with a buffer

```
var vertices = [
 [...]
];

var indices = [
 [...]
];

var uv = [
0,1
0,0
1,1
0,0
[. . .]
];
```

Send the UV mapping

UV coordinates associated to each vertex can be sent with a buffer

```
[...]
  var uvLocation = gl.getAttribLocation(program, "a_uv");

  var uvBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER, uvBuffer);

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(uv), gl.STATIC_DRAW);

  gl.vertexAttribPointer(uvLocation, 2, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(uvLocation);

[...]
```

How do Shaders deal with Textures?

Textures data can be accessed in a shader using a *particular lookup function*

```
vec4 texture(sampler2D sampler, vec2 uvCoord);
```

`sampler2d` is the identifier of the texture object

`uvCoord` are the UV coordinates for the lookup

It returns a **`vec4`**, with the color it fetches

How do Shaders deal with Textures?

- **sampler2D** sends the identifier (handle) of the *texture data* from the client code to the shader.

```
[...]

in vec2 uvCoord;
out vec4 outColor;
uniform sampler2D sampler;

void main() {

  outColor = texture(sampler, uvCoord);
}
```

- Then **`texture()`** extracts the value of the texel (texture element), that can be used to compute the final color of the fragment (here **`outColor`**)

How do Shaders deal with Textures?

```
#version 300 es

in vec4 a_position;
in vec2 a_uv;

out vec2 uvCoord;
uniform mat4 matrix;

void main() {
  uvCoord = a_uv;
  gl_Position = matrix * a_position;
}
```

```
#version 300 es

precision mediump float;

in vec2 uvCoord;
out vec4 outColor;

uniform sampler2D sampler;

void main() {

  outColor = texture(sampler, uvCoord);
}
```

Render the Cube

During the rendering cycle:

```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(textureFileHandle, 0);
```

- The active texture unit is set to 0. Subsequent texture state calls will affect this unit only.
- Depending on the specific implementation, a different number of texture units can be defined at one time and assigned to different levels.
- **Note**: more than one level is required only when multiple textures are used at once on the same face, otherwise the same level can be used for different textures.

Render the Cube

During the rendering cycle:

```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(textureFileHandle, 0);
```

- The texture must be selected again with the glBindTexture() command.
- The function uses the id of the texture object.

Render the Cube

During the rendering cycle:

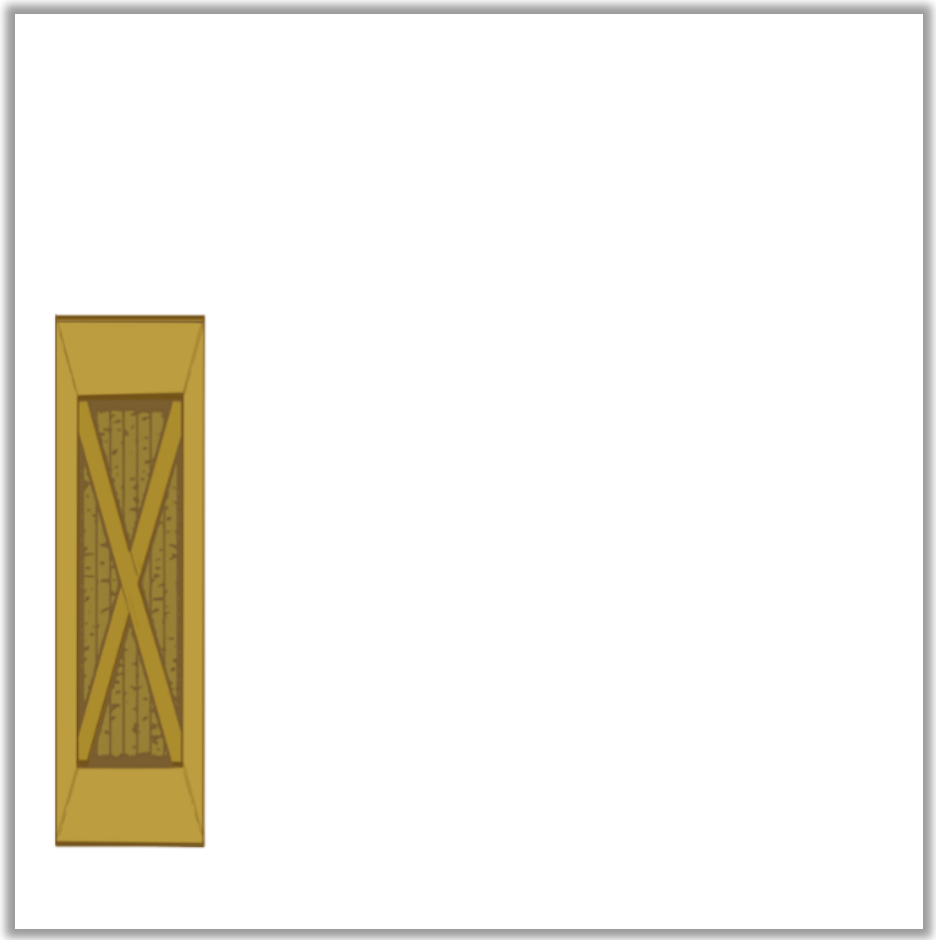
```
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(textureHandle, 0);
```

- Tell the shader our texture `uniform sampler2D sampler;` is in unit 0

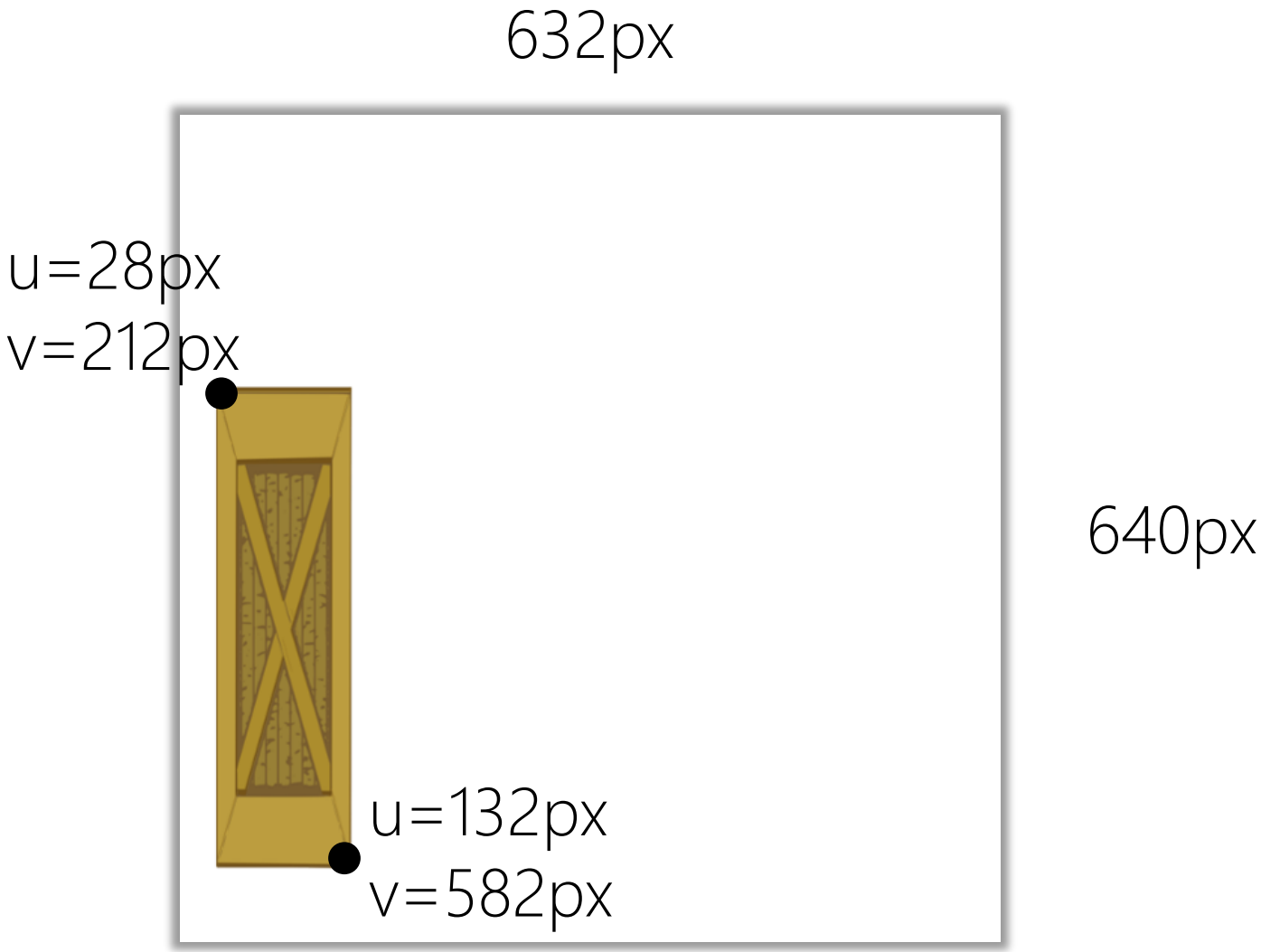
UV Mapping

Example with the Cube

What if we have this (partial) texture and we want to obtain the textured cube before?



Example with the Cube



Cube Maps

- Cube maps can be created with special WebGL commands, in particular selecting TEXTURE_CUBE_MAP when calling the bind command.

```
var texture=gl.createTexture();
gl.activeTexture(gl.TEXTURE0 + 3);
gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
```

Cube Maps

- Each face of the cube, is characterized by a literal name, and by the file in which it is contained.

```
target: gl.TEXTURE_CUBE_MAP_NEGATIVE_X,
url: 'negx.jpg',

target: gl.TEXTURE_CUBE_MAP_POSITIVE_Y,
url: 'posy.jpg',

target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Y,
url: 'negy.jpg',

target: gl.TEXTURE_CUBE_MAP_POSITIVE_Z,
url: 'posz.jpg',

target: gl.TEXTURE_CUBE_MAP_NEGATIVE_Z,
url: 'negz.jpg',
```

Cube Maps

- Each image must be separately loaded with a procedure similar to the one used for standard 2D textures.

```
const level = 0;
const internalFormat = gl.RGBA;
const width = 512;
const height = 512;
const format = gl.RGBA;
const type = gl.UNSIGNED_BYTE;

const image = new Image();
image.src = 'negx.jpg’,;
image.addEventListener('load', function() {
  gl.activeTexture(gl.TEXTURE0 + 3);
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, texture);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_X, level,
                internalFormat, format, type, image);
  gl.generateMipmap(gl.TEXTURE_CUBE_MAP);
}
```

Cube Maps

- Finally, the filtering and the MipMap for the complete cube texture should be generated.

```
gl.generateMipmap(gl.TEXTURE_CUBE_MAP);
gl.texParameteri(gl.TEXTURE_CUBE_MAP,
                 gl.TEXTURE_MIN_FILTER,
                 gl.LINEAR_MIPMAP_LINEAR);
```

Cube Maps

- In a shader, cube map uniforms are indexed by specific samplers.

```
uniform samplerCube u_tex_Env;
```

- As outlined earlier, texel can be fetched being indexed by a direction vector, instead of a UV coordinate.

```
vec3 refDir = -reflect(v,n);
vec4 specFactFromEnvMap = texture(u_tex_Env, refDir);
```