Visualisierung und Interaktion für die Medizin (VIM)

Texture Mapping

Christopher Brumann Prof. Dr. Markus Kukuk

SS 2018

FH-Dortmund

Introduction to textures

Textures are neccessary for good computer graphics

- Texture mapping determines the look of pixels
- Textures are important for a realistic look

”If it looks like computer graphics, it is not good computer graphics.” – Jeremy Birn

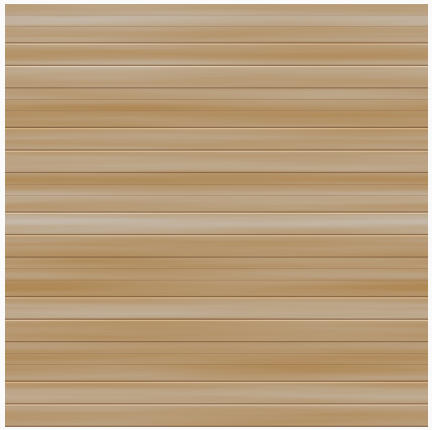
Why texture mapping is important

- Rendered scenes can look artificial, due to missing details
- It is important to add more surface detail
- Increasing the geometrical complexity increases costs
- Use texture mapping to create the illusion of details

http://www.cgarchitect.com/2017/11/painterly-still

How texture mapping operates in general

- A texture map provides surface detail
- Texture maps are used in the fragment shader for
  - Combination of the fragment with lookup value(s)
  - Computation of the fragment with based on lookup value(s)

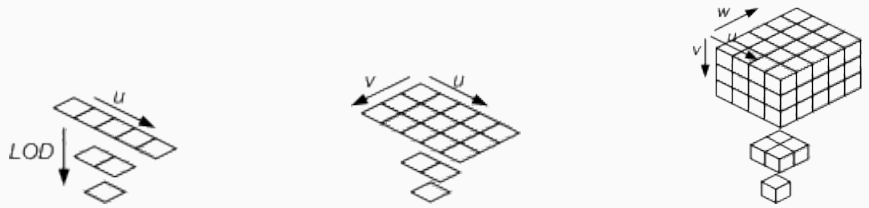


- Advantage: Image complexity has no influence to computational complexity

A texture is just data

A texture map is an array of values stored in texture memory

- A texture map can be 1D, 2D or 3D



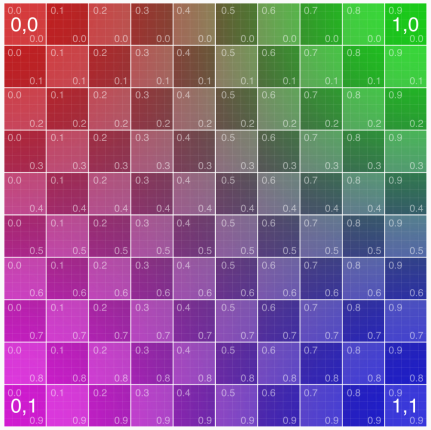
- In pictures the fundamental element is called **pixel**
- In textures the fundamental element is called **texel**

- Texel can contain scalar values
  - In image texturing: RGB(A) values

- Texel can contain vectors
  - Bump mapping: surface normals
  - Environment mapping: Reflection vectors

Image texturing – Texel contain scalar values

- A 2D-Texture map is defined as `texture[w(s)][h(t)]` of type RGB(A)
- Important: Texture coordinates are located in $[0, 1]$



- Can be mapped and scaled to any arbitrary surface

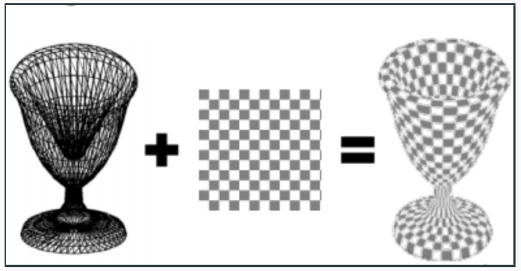
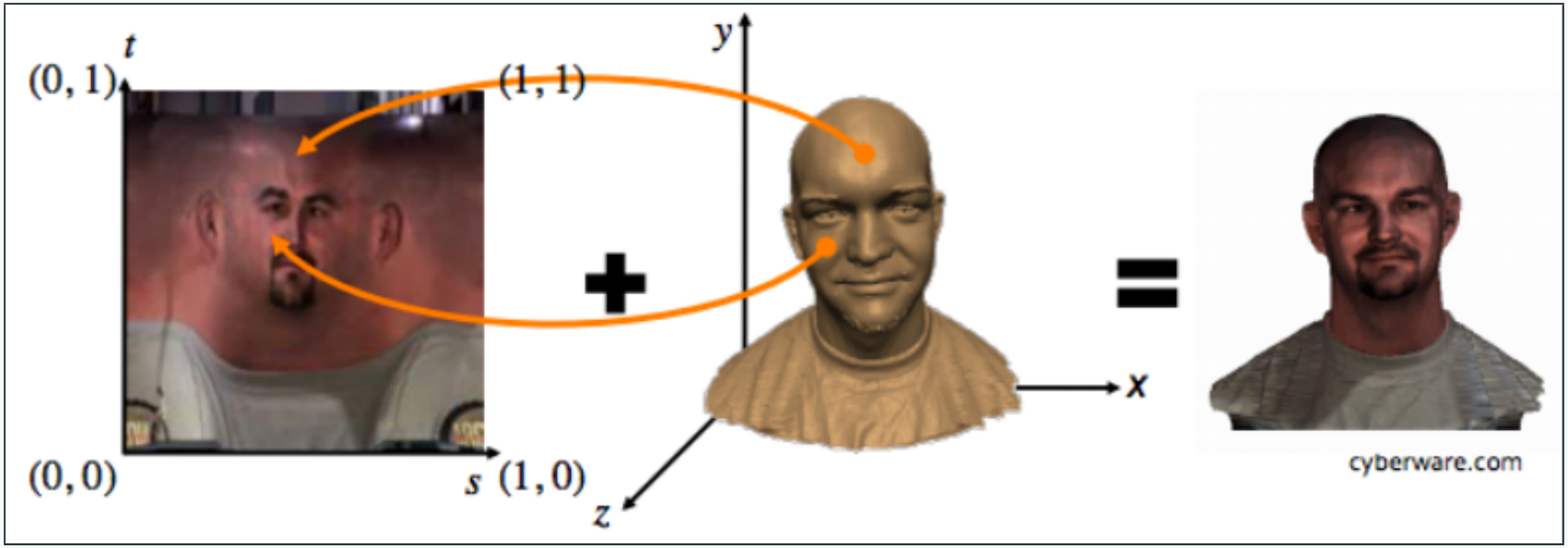


Image texturing in general

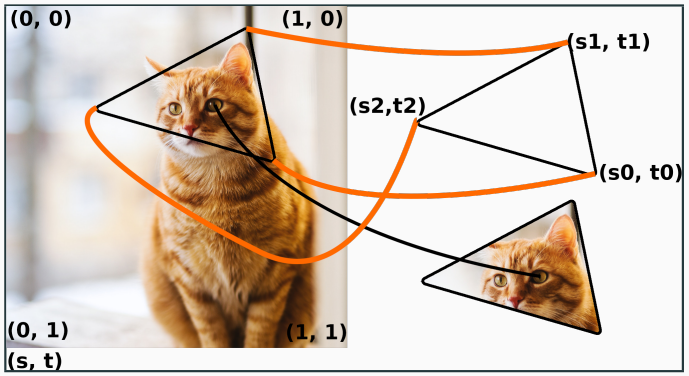
Steps for mapping an image texture on a 3D object:

1. Pick a surface point in (x, y, z)
2. Lookup the corresponding texel (texture coordinates)
3. Compute the final fragment color



Texture coordinates s and $t \in [0, 1]$

- For each vertex a texture coordinate is assigned
- The coordinate's dimension matches the texture's dimension



How to provide texture coordinates:

- Manually specified by the programmer
- Interpolated during rasterization
- Texturing during fragment processing

Image texturing in WebGL

Image texturing in WebGL

- To better understand image texturing
- Let’s remove the shading and go back to a static color

```
var vertexShaderSource = `#version 300 es

// an attribute is an input (in) to a vertex shader.
// It will receive data from a buffer
in vec4 a_position;
in vec3 a_normal;

// A matrix to transform the positions by
uniform mat4 u_modelViewProjection;
uniform mat4 u_model;

uniform vec3 u_lightWorldPosition;
uniform vec3 u_viewWorldPosition;

// a varying the color to the fragment shader
out vec3 v_normal;
out vec3 v_surfaceToLight;
out vec3 v_surfaceToView;

// all shaders have a main function
void main() {

  // gl_Position is a special variable a vertex shader
  // is responsible for setting

  gl_Position = u_modelViewProjection * a_position;

  // Pass the normal to the fragment shader.
  v_normal = mat3(u_model) * a_normal;


  // compute the world position of the surfoace
  vec3 surfaceWorldPosition = (u_model * a_position).xyz;

  // compute the vector of the surface to the light
  // and pass it to the fragment shader
  v_surfaceToLight = u_lightWorldPosition - surfaceWorldPosition;


  // compute the vector of the surface to the view/camera
  // and pass it to the fragment shader
  v_surfaceToView = u_viewWorldPosition - surfaceWorldPosition;

}
`;
```

```
var fragmentShaderSource = `#version 300 es

// fragment shaders don't have a default precision so we need
// to pick one. mediump is a good default. It means "medium precision"
precision mediump float;

// the varied normal passed from the vertex shader

in vec3 v_normal;
in vec3 v_surfaceToLight;
in vec3 v_surfaceToView;

uniform vec4  u_color;
uniform float u_shininess;

// we need to declare an output for the fragment shader
out vec4 outColor;

void main() {
   // because v_normal is a varying it's interpolated
  // so it will not be a uint vector. Normalizing it
  // will make it a unit vector again

  vec3 normal                = normalize(v_normal);
  vec3 surfaceToLightDirection = normalize(v_surfaceToLight);

  vec3 surfaceToViewDirection  = normalize(v_surfaceToView);
  vec3 halfVector              = normalize(surfaceToLightDirection + surfaceToViewDirection);

  // compute the light by taking the dot product
  // of the normal to the light's reverse direction

  float light    = dot(normal, surfaceToLightDirection);

  float specular = 0.0;
  if (light > 0.0) {
    specular = pow(dot(normal, halfVector), u_shininess);
  }

  //float specular = dot(normal, halfVector);

  outColor = u_color;

  // Lets multiply just the color portion (not the alpha)
  // by the light

  //outColor.rgb = u_color.rgb * light + u_color.rgb * specular;

  outColor.rgb *= light;

  // Just add in the specular
  outColor.rgb += specular;
}
`;
```

Image texturing in WebGL

- To better understand image texturing
- Let’s remove the shading and go back to a static color

```
40  var vertexShaderSource = `#version 300 es
41
42  // an attribute is an input (in) to a vertex shader.
43  // It will receive data from a buffer
44  in vec4 a_position;
45
46  // A matrix to transform the positions by
47  uniform mat4 u_modelViewProjection;
48
49  // all shaders have a main function
50  void main() {
51    gl_Position = u_modelViewProjection * a_position;
52  }
53  `;
```

```
56  var fragmentShaderSource = `#version 300 es
57
58  // fragment shaders don't have a default precision so we need
59  // to pick one. mediump is a good default. It means "medium precision"
60  precision mediump float;
61
62  // we need to declare an output for the fragment shader
63  out vec4 outColor;
64
65  void main() {
66    outColor = vec4(0.2, 0., 0.2, 1.0);
67  }
68  `;
```

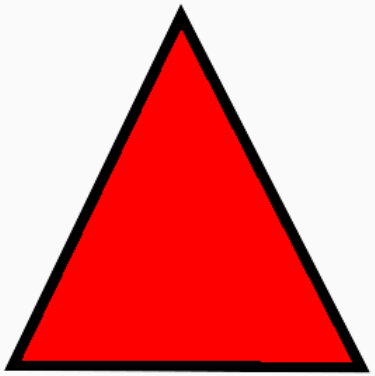
Image texturing in WebGL

1. Define texture coordinates
2. Setup texture coordinates
3. Load an image texture
4. Vertex Shader: Pass through texture coordinates
5. Fragment Shader: Lookup the interpolated texel value

Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424               0,   60,   65,
425             -80,  -60,   65,
426              80,  -60,   65,
427
428              80,  -60,   65,
429             -80,  -60,   65,
430               0,    0,  -65,
431
432               0,   60,   65,
433              80,  -60,   65,
434               0,    0,  -65,
435
436               0,   60,   65,
437               0,    0,  -65,
438             -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550         ];
```

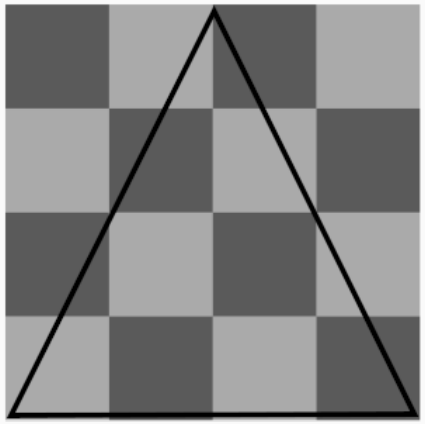
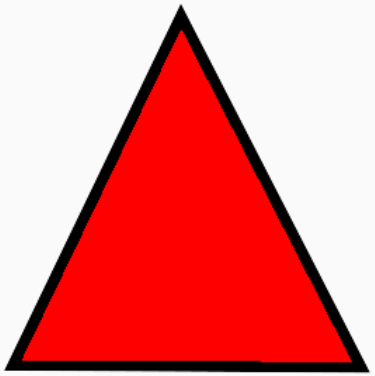


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432             0,   60,   65,
433            80,  -60,   65,
434             0,    0,  -65,
435
436             0,   60,   65,
437             0,    0,  -65,
438           -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539
540
541
542
543
544
545
546
547
548
549
550         ];
```

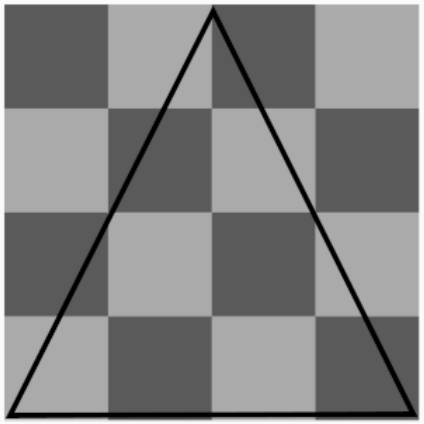
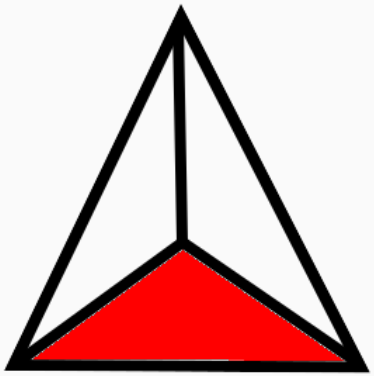


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432             0,   60,   65,
433            80,  -60,   65,
434             0,    0,  -65,
435
436             0,   60,   65,
437             0,    0,  -65,
438           -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539
540
541
542
543
544
545
546
547
548
549
550         ];
```

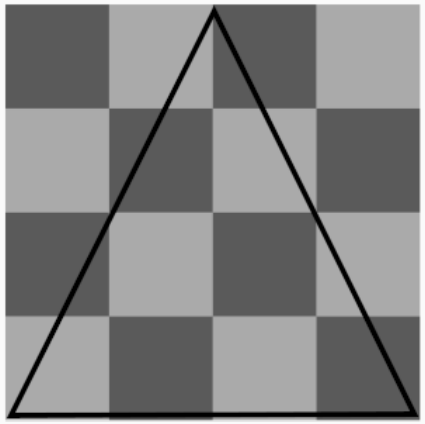
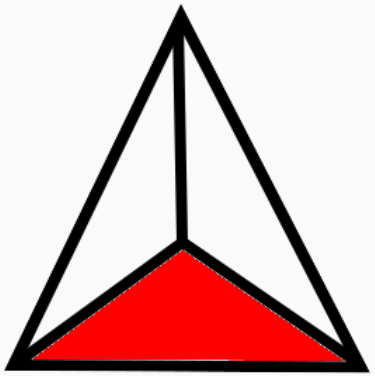


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432             0,   60,   65,
433            80,  -60,   65,
434             0,    0,  -65,
435
436             0,   60,   65,
437             0,    0,  -65,
438           -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539             0.0, 1.0,
540             1.0, 1.0,
541             0.5, 0.0,
542
543
544
545
546
547
548
549
550         ];
```

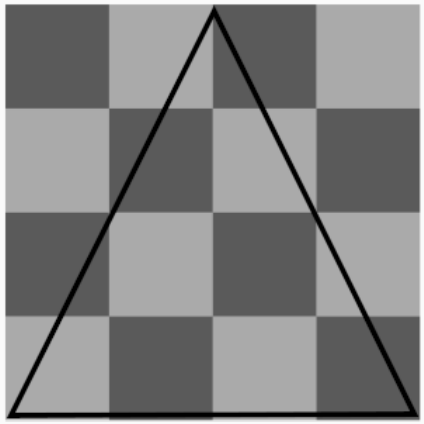
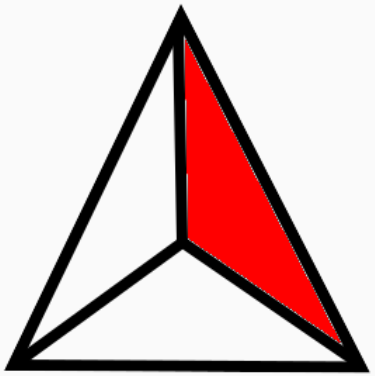


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432             0,   60,   65,
433            80,  -60,   65,
434             0,    0,  -65,
435
436             0,   60,   65,
437             0,    0,  -65,
438           -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539             0.0, 1.0,
540             1.0, 1.0,
541             0.5, 0.0,
542
543
544
545
546
547
548
549
550         ];
```

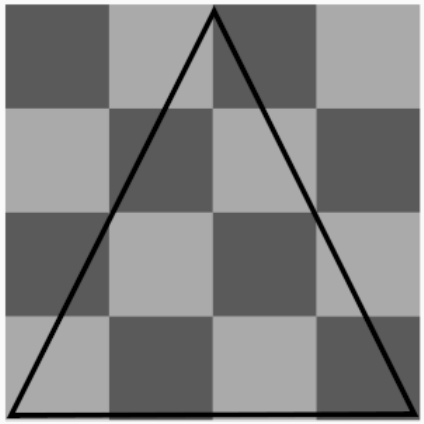
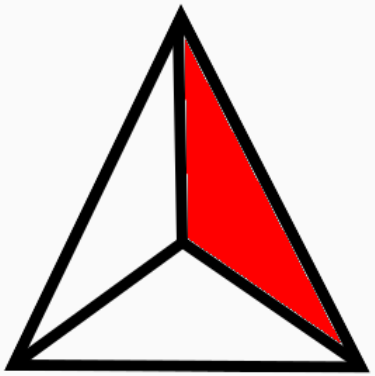


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424              0,   60,   65,
425            -80,  -60,   65,
426             80,  -60,   65,
427
428             80,  -60,   65,
429            -80,  -60,   65,
430              0,    0,  -65,
431
432              0,   60,   65,
433             80,  -60,   65,
434              0,    0,  -65,
435
436              0,   60,   65,
437              0,    0,  -65,
438            -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539             0.0, 1.0,
540             1.0, 1.0,
541             0.5, 0.0,
542
543             0.0, 1.0,
544             1.0, 1.0,
545             0.5, 0.0,
546
547
548
549
550         ];
```

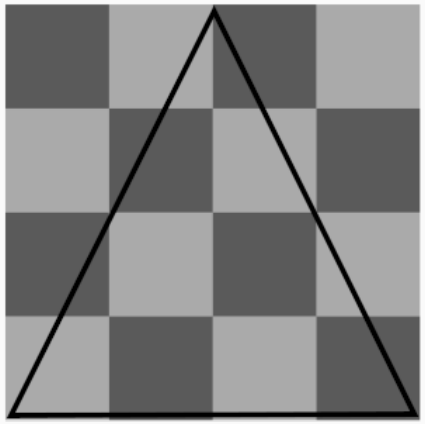
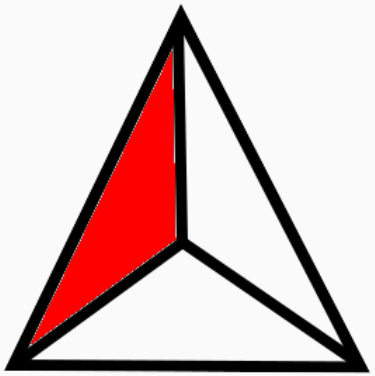


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432            0,   60,   65,
433           80,  -60,   65,
434            0,    0,  -65,
435
436            0,   60,   65,
437            0,    0,  -65,
438          -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539             0.0, 1.0,
540             1.0, 1.0,
541             0.5, 0.0,
542
543             0.0, 1.0,
544             1.0, 1.0,
545             0.5, 0.0,
546
547
548
549
550         ];
```

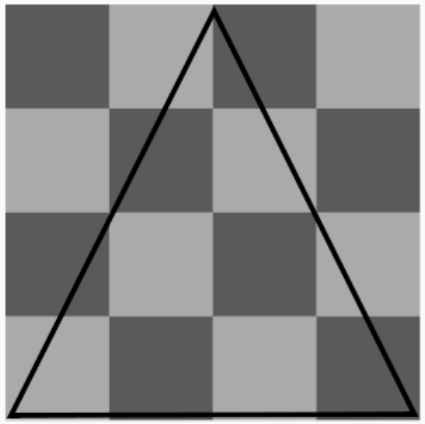
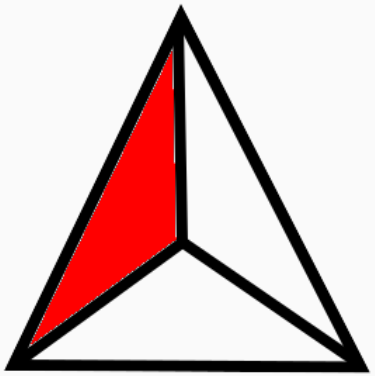


Image texturing in WebGL

1. Define texture coordinates

```
423         var positions = [
424             0,   60,   65,
425           -80,  -60,   65,
426            80,  -60,   65,
427
428            80,  -60,   65,
429           -80,  -60,   65,
430             0,    0,  -65,
431
432            0,   60,   65,
433           80,  -60,   65,
434            0,    0,  -65,
435
436            0,   60,   65,
437            0,    0,  -65,
438          -80,  -60,   65,
439
440         ];
```



```
534         var texcoords =[
535             0.5, 0.0,
536             0.0, 1.0,
537             1.0, 1.0,
538
539             0.0, 1.0,
540             1.0, 1.0,
541             0.5, 0.0,
542
543             0.0, 1.0,
544             1.0, 1.0,
545             0.5, 0.0,
546
547             1.0, 1.0,
548             0.5, 0.0,
549             0.0, 1.0,
550         ];
```

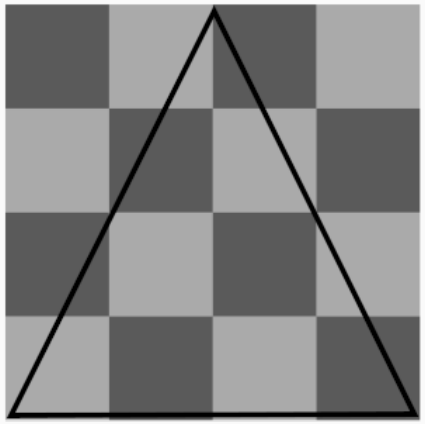


Image texturing in WebGL

2. Setup texture coordinates

- Attribute location
- Texture coordinate buffer

```
404     var texcoordAttributeLocation = gl.getAttribLocation(program, "a_texcoord");
```

```
552 var texcoordBuffer = gl.createBuffer();
553 gl.bindBuffer(gl.ARRAY_BUFFER, texcoordBuffer);
554
555 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(texcoords), gl.STATIC_DRAW);
556
557 gl.enableVertexAttribArray(texcoordAttributeLocation);
558
559 var size = 2;
560 var type = gl.FLOAT;
561 var normalize = true;
562 var stride = 0;
563 var offset = 0;
564
565 gl.vertexAttribPointer(texcoordAttributeLocation, size, type, normalize, stride, offset);
```

Image texturing in WebGL

3. Load an image texture

   - And specify the foobar texture image

```
570         var texture = gl.createTexture();
571         gl.bindTexture(gl.TEXTURE_2D, texture);
572
573         var mip = 0;
574         var internalFormat = gl.RGBA;
575         var width = 1, height = 1, border = 0;
576         var format = gl.RGBA;
577         var type = gl.UNSIGNED_BYTE;
578
579         gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, width, height, border, format, type,
580             new Uint8Array([0, 0, 255, 255]));
581
582         var image = new Image();
583         image.src = "./resources/checkerboard.png";
584         image.addEventListener('load', function() {
585             gl.bindTexture(gl.TEXTURE_2D, texture);
586             gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, format, type, image);
587             gl.generateMipmap(gl.TEXTURE_2D);
588             drawScene();
589         });
```

4. Vertex Shader: Pass through texture coordinates

```
40  var vertexShaderSource = `#version 300 es
41
42  // an attribute is an input (in) to a vertex shader.
43  // It will receive data from a buffer
44  in vec4 a_position;
45
46  in vec2 a_texcoord;
47
48  // A matrix to transform the positions by
49  uniform mat4 u_modelViewProjection;
50
51  // A varying to pass the texture coordinate to the fragment shader
52  out vec2 v_texcoord;
53
54  void main() {
55    gl_Position = u_modelViewProjection * a_position;
56
57    // Pass the texcoord to the fragment shader
58    v_texcoord = a_texcoord;
59  }
60  `;
```

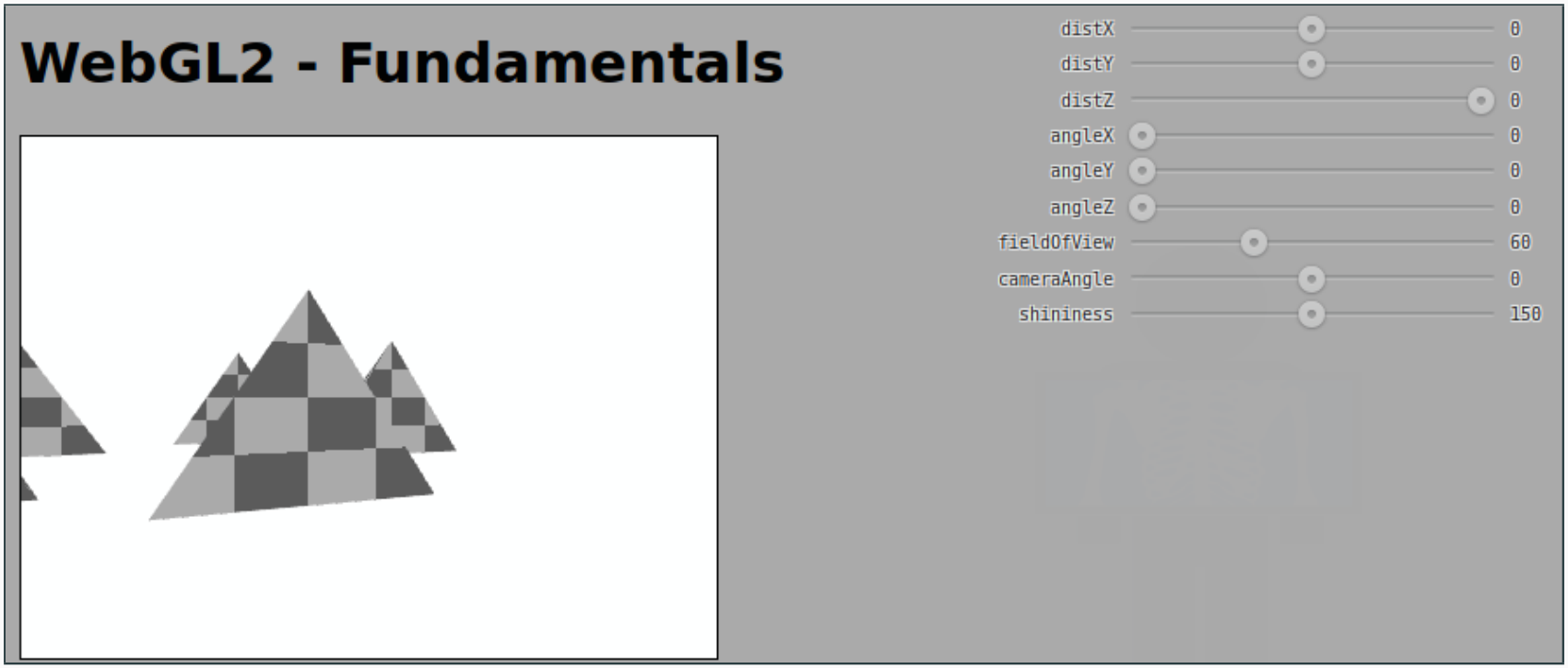
Image texturing in WebGL

5. Fragment Shader: Lookup the interpolated texel value

```
63  var fragmentShaderSource = `#version 300 es
64
65  // fragment shaders don't have a default precision so we need
66  // to pick one. mediump is a good default. It means "medium precision"
67  precision mediump float;
68
69  in vec2 v_texcoord;
70
71  uniform sampler2D u_texture;
72
73  // we need to declare an output for the fragment shader
74  out vec4 outColor;
75
76  void main() {
77    outColor = texture(u_texture, v_texcoord);
78  }
79  `;
```

Image texturing in WebGL

See File: `01-example-3D-Add_Texture.html`



Texture parameters and MIP Maps

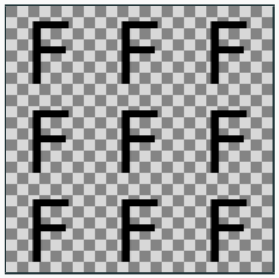
Parameters can change texel lookup

- What if we specify texture coordinates $(s, t) < 0$ or > 1?
- What happens if we render the textured object smaller/bigger than the texture's size?
- In WebGL texture parameters can be changed by using:

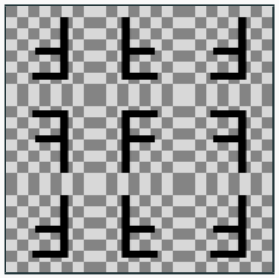
  `gl.texParameter[fi](target, pname, param)`

Texture Wrapping

We can set these parameters for handling (s, t) values:



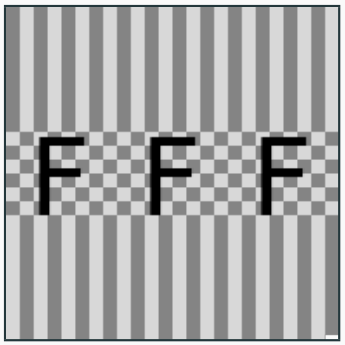
gl.REPEAT



gl.MIRRORED_REPEAT

gl.CLAMP_TO_EDGE

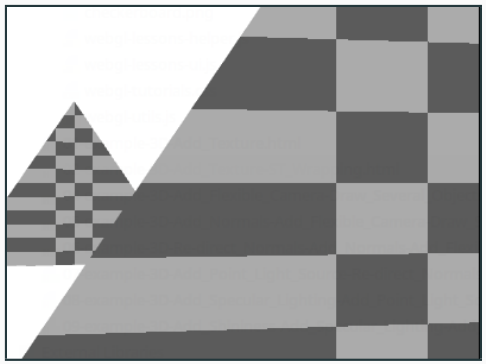
for `TEXTURE_WRAP_S` and `TEXTURE_WRAP_T` separately:



Minification and Magnification filter

Minification:
The texture must be reduced if the object is very far away from the viewer (extreme case: the object is only 1 pixel in size)

Magnification:
The texture must be enlarged to cover the object when it is close to the viewer.



- In WebGL the parameters are
  - `TEXTURE_MIN_FILTER`
  - `TEXTURE_MAG_FILTER`
- Possible values are
  - `NEAREST`
  - `LINEAR`

Texture parameters in WebGL

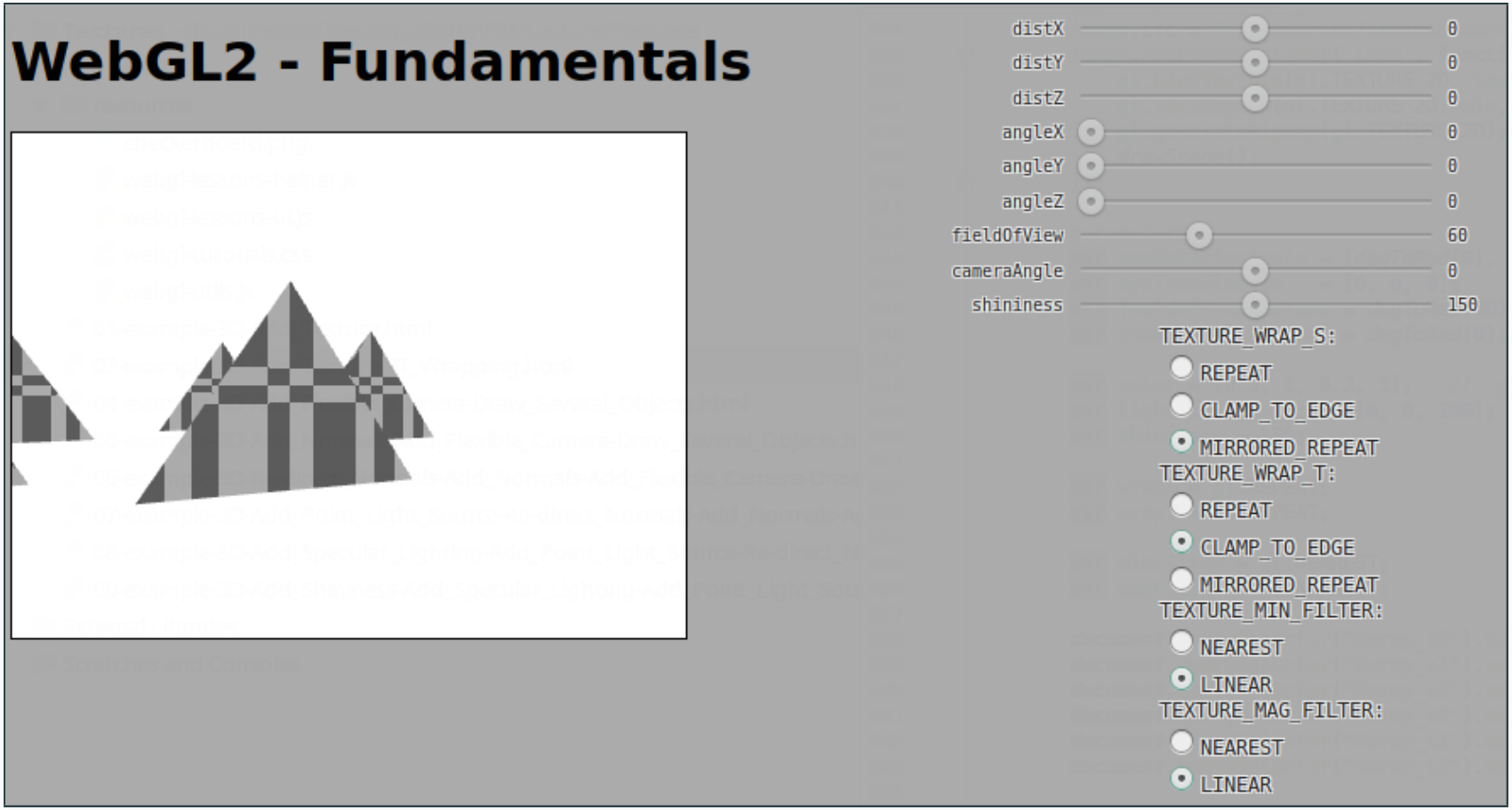
1. Select your texture parameter values
2. Bind the corresponding texture
3. Set / update the values for the parameter

```
552         var wrapS = gl.REPEAT;
553         var wrapT = gl.REPEAT;
554
555         var minFilter = gl.NEAREST;
556         var magFilter = gl.NEAREST;
```

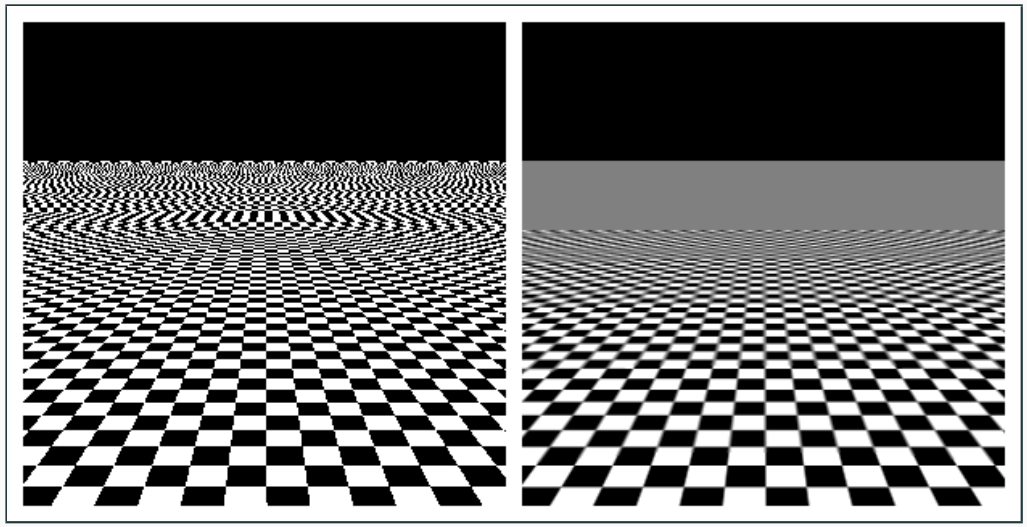
```
708         var modelViewMatrix = m4.multiply(viewMatrix, modelMatrix);
709         var modelViewProjecitonMatrix = m4.multiply(perspectiveMatrix, modelViewMatrix);
710
711         gl.bindTexture(gl.TEXTURE_2D, texture);
712         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, minFilter);
713         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, magFilter);
714         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, wrapS);
715         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, wrapT);
716
717         // Set the matrix.
718         gl.uniformMatrix4fv(matrixLocationMVP, true, modelViewProjecitonMatrix);
```

Texture parameters in WebGL

See File: `02-example-3D-Add_Texture-ST_Wrapping_MinMag_Filter.html`



MIP Mapping



https://commons.wikimedia.org/wiki/File:Mipmapping_example.png

- MIP Mapping is an anti-aliasing technique
- Uses different texture sizes of the same texture for rendering
- Fragments near the camera will use a ”high resolution texture”
- Fragments far away will use a ”low resolution texture”

MIP Mapping

You have two options to use MIP Mapping:

The easiest way is to call `gl.generateMipmap`:

```
var texture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture);

var mip = 0;
var internalFormat = gl.RGBA;
var width = 1, height = 1, border = 0;
var format = gl.RGBA;
var type = gl.UNSIGNED_BYTE;

gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, width, height, border, format, type,
    new Uint8Array([0, 0, 255, 255]));

var image = new Image();
image.src = "./resources/checkerboard.png";
image.addEventListener('load', function() {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, format, type, image);
    gl.generateMipmap(gl.TEXTURE_2D);
    drawScene();
});
```

MIP Mapping

You have two options to use MIP Mapping:

Otherwise you can provide your own mips, but you need to provide all of them (down to scale 1×1):

```
178     mipTexture = gl.createTexture();
179     gl.bindTexture(gl.TEXTURE_2D, mipTexture);
180
181     let c = document.createElement("canvas");
182     let ctx = c.getContext("2d");
183     let mips = [
184         {size: 512, color: "rgb(255,255,255)",},
185         {size: 256, color: "rgb(0,255,0)",},
186         {size: 128, color: "rgb(0,0,255)",},
                        [...]
193         {size: 1, color:   "rgb(255,255,255)",}
194     ];
195
196     mips.forEach(function (s, level) {
197         let size = s.size;
198         c.width = size;
199         c.height = size;
200         ctx.fillStyle = s.color;
201         ctx.fillRect(0, 0, size, size);
202         ctx.fillStyle = "rgb(255, 0, 0)";
203         ctx.font = (size/4) + "px Arial";
204         ctx.fillText(size + "px", 0, size/2);
205         gl.texImage2D(gl.TEXTURE_2D, level, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, c);
206     });
```

Extended Minification and Magnification filter

More parameters for `TEXTURE_MIN_FILTER`:

	NEAREST*	LINEAR*
*	largest mip, 1px	largest mip, 4px
*_MIPMAP_NEAREST	1 mip, 1px	1 mip, 4px
*_MIPMAP_LINEAR	2 mips, 1px	2 mips, 4px

```
gl.texParameteri(
    gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
    gl.NEAREST_MIPMAP_LINEAR
)
```

Extended Minification and Magnification filter

More parameters for `TEXTURE_MIN_FILTER`:

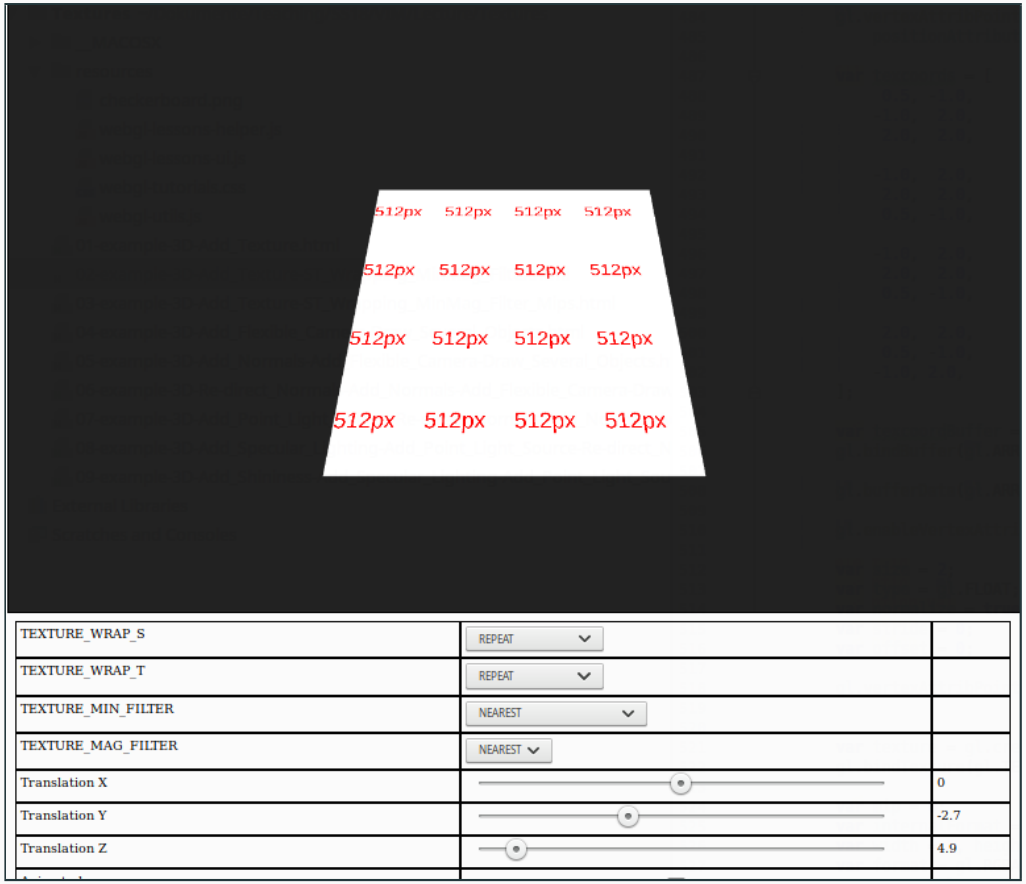
	NEAREST*	LINEAR*
*	largest mip, 1px	largest mip, 4px
*_MIPMAP_NEAREST	1 mip, 1px	1 mip, 4px
*_MIPMAP_LINEAR	2 mips, 1px	2 mips, 4px

```
gl.texParameteri(
    gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
    gl.NEAREST_MIPMAP_LINEAR
)
```

Only applicable for `TEXTURE_MIN_FILTER`

MIP Mapping in WebGL

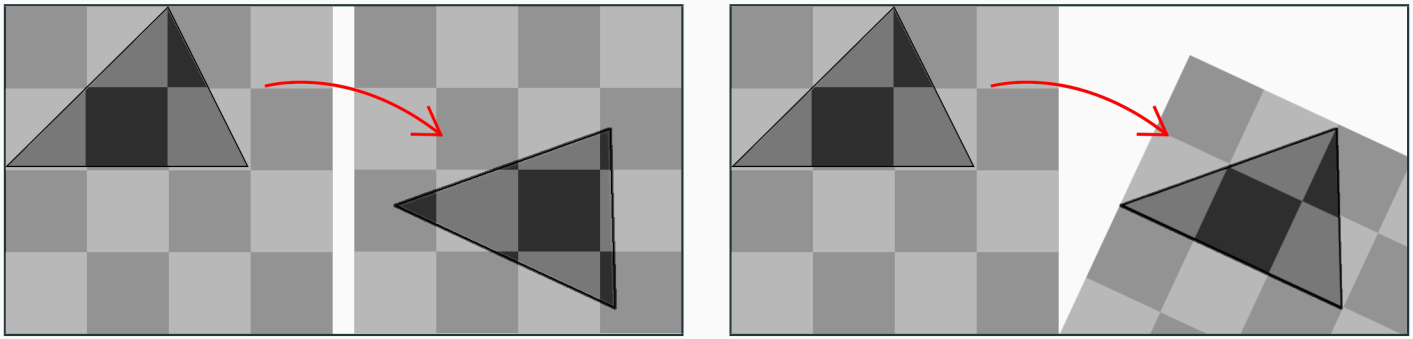
See File: `03-example-3D-Add_Texture-ST_Wrapping_MinMag_Filter_Mips.html`



Texture mapping strategies

Texture mapping techniques

- How do we paste our 2D image texture onto a 3D object?



(a) Non-parametrically **(b)** Parametrically

(a) Non-parametrically:

- Texture is fixed to world coordinates
- Gives a projector effect

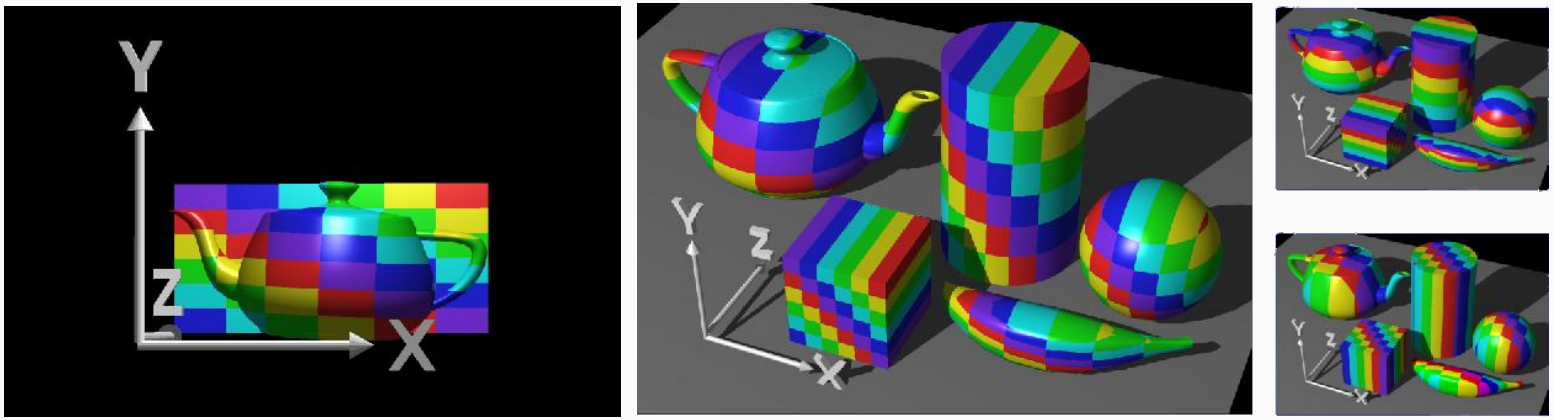
(b) Parametrically:

- Texture is tied to object
- Map object coordinates to texture coordinates

Parametric Mapping

Planar / orthographic map:

- Remove a single object coordinate (x, y, z) and map the texture onto that coordinate plane
- The texture is constant in this direction

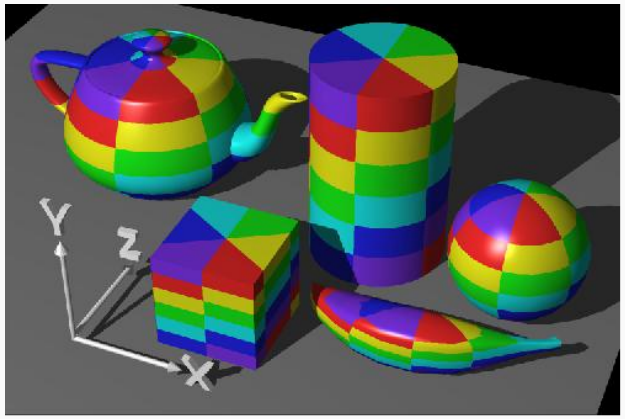


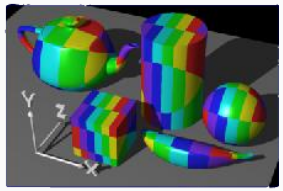
https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

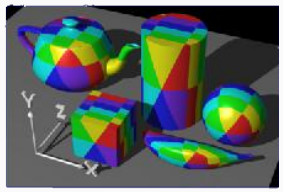
Parametric Mapping

Cylindrical map:

- Convert object coordinates (x, y, z) to (r, θ, h) and use θ for s and h for t
- This wraps the texture around the object
- At minimum and maximum, the texture gets pinched together



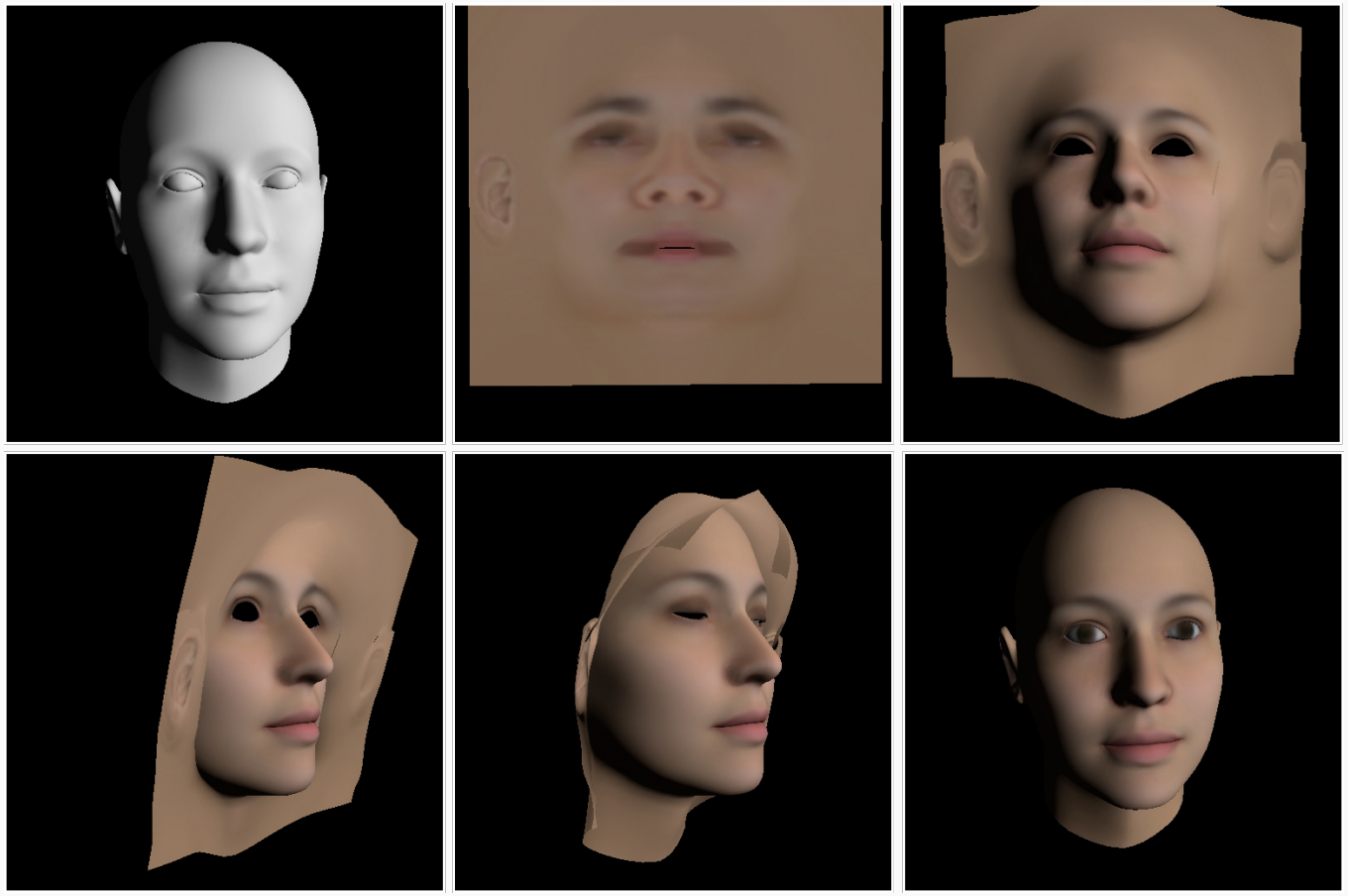




https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

Parametric Mapping

Cylindrical mapping is useful for faces:

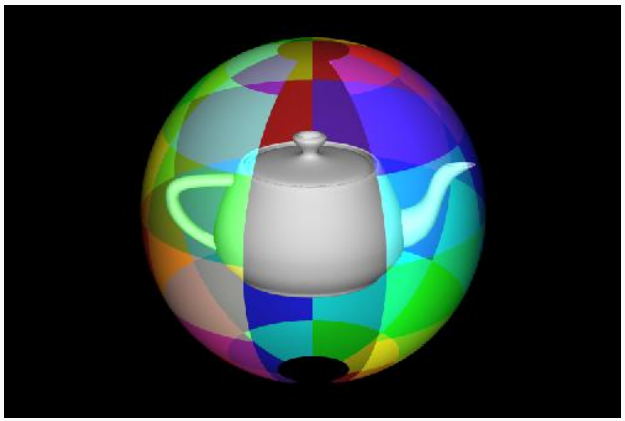


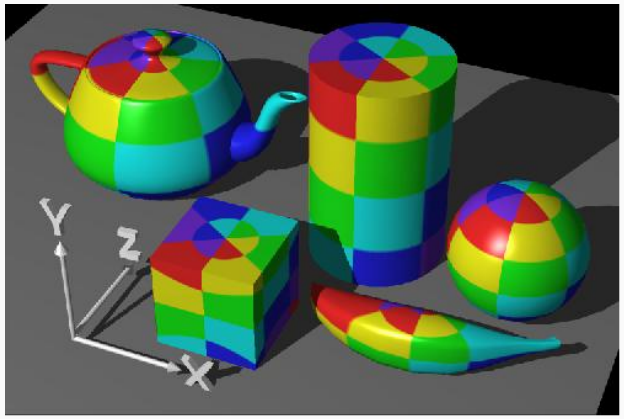
https://www.evl.uic.edu/sjames/cs525/project1.html

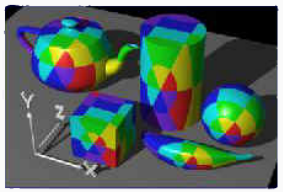
Parametric Mapping

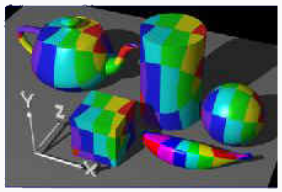
Spherical map:

- Convert object coordinates (x, y, z) to spherical (θ, φ) coordinates
- Use longitude φ for s and latitude θ for t
- Pinches texture at the poles and stretches squares along the equator







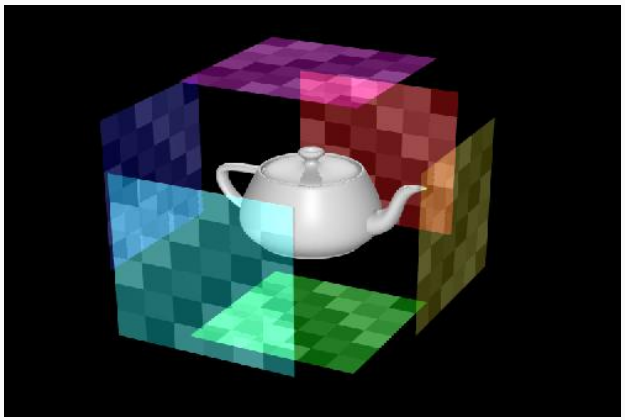


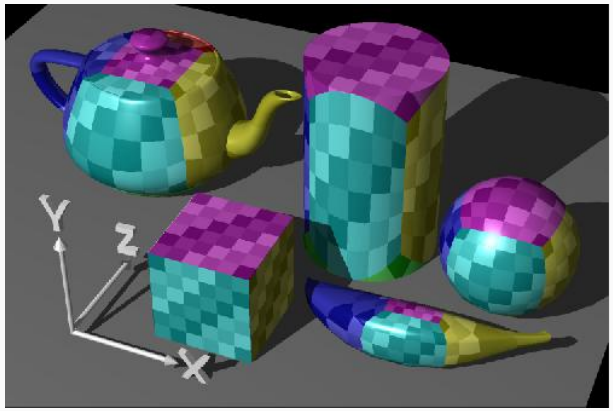
https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm
https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_3.htm

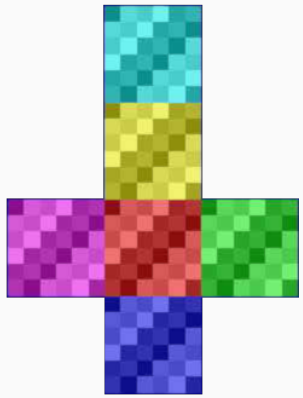
Parametric Mapping

Cube / Box map:

- Use six planar maps, one for each face of the cube



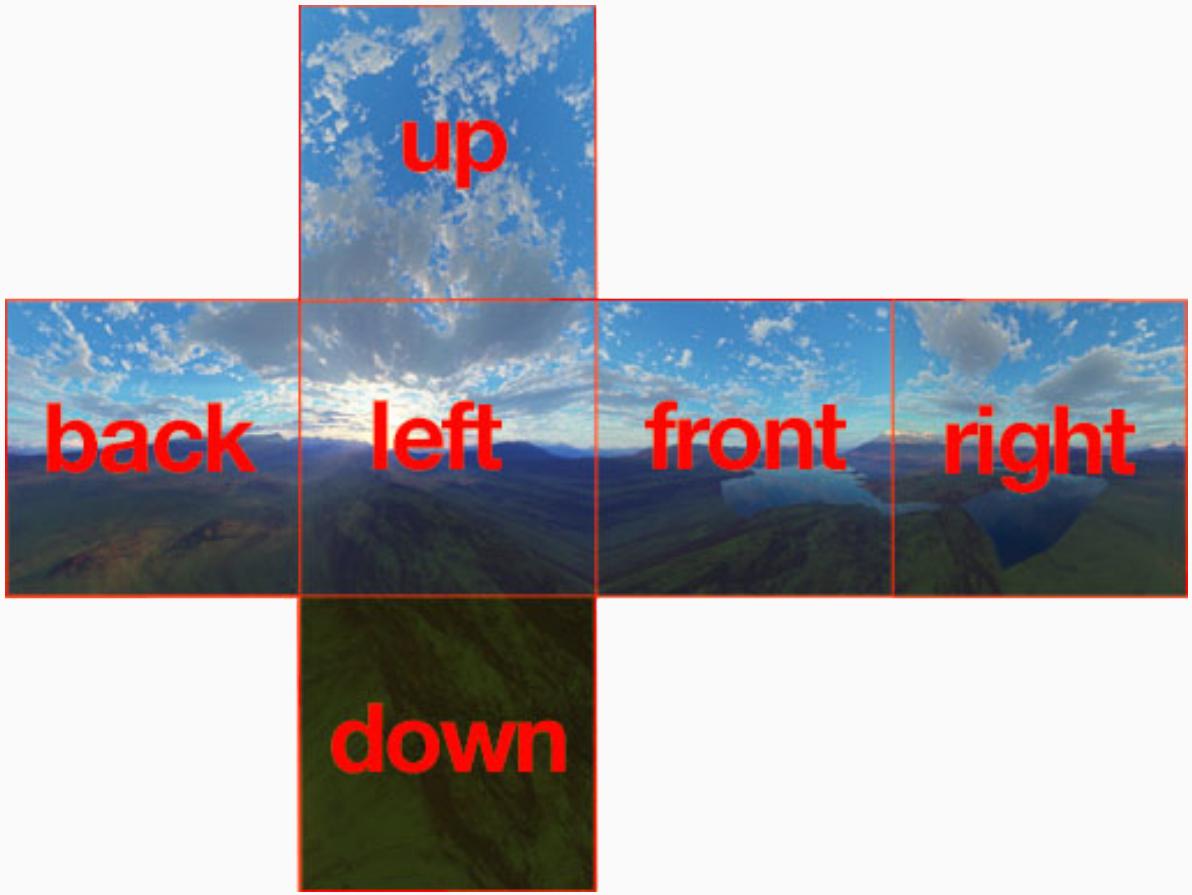




https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_3.htm

Parametric Mapping

Cube / Box mapping is useful for skyboxes:



https://commons.wikimedia.org/wiki/File:Skybox_example.png

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.
2. Render 4 triangles separately, each with a different texture

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.
2. Render 4 triangles separately, each with a different texture

Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.
2. Render 4 triangles separately, each with a different texture
3. Method similar to cube mapping, using a texture atlas

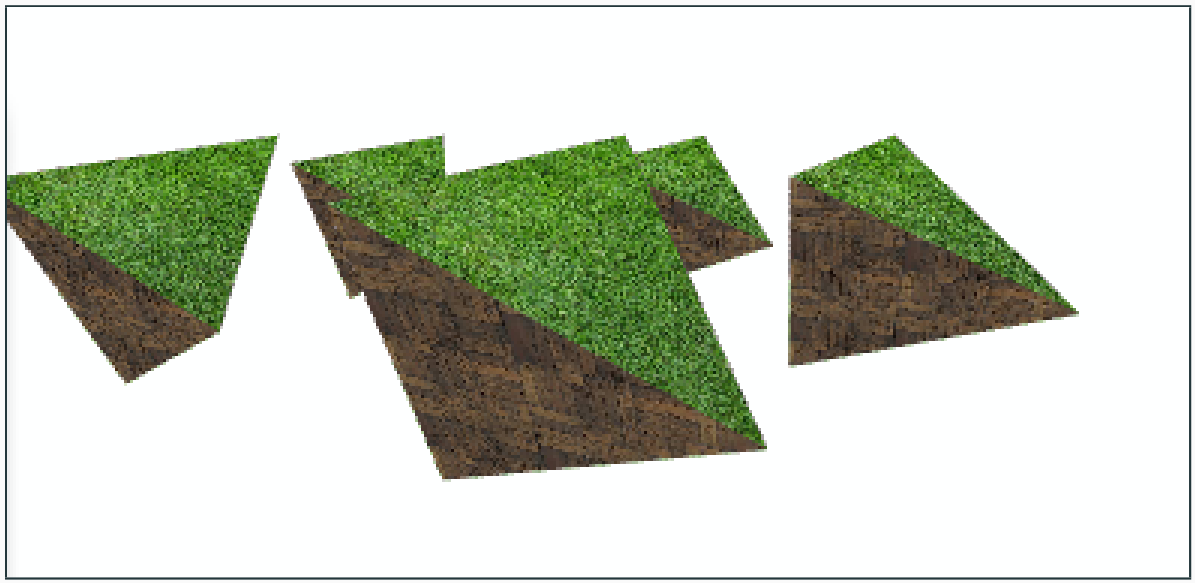
Parametric Mapping

How can we texture our tetrahedron with separate image textures per side?

1. Implementing a complicated shader using 4 different textures. Provide extra information per vertex to decide which texture to use.
2. Render 4 triangles separately, each with a different texture
3. Method similar to cube mapping, using a texture atlas

Per side tetrahedron image texturing

1. Create the texture atlas
2. Change texture coordinates
3. Set the correct texture wrapping parameter



Per side tetrahedron image texturing

1. Create the texture atlas

   - Merge all image textures in a single file
   - Use this file as image texture

```
554         gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, width, height, border, format, type,
555             new Uint8Array([0, 0, 255, 255]));
556
557         var image = new Image();
558         image.src = "./resources/textureatlas.png";
559         image.addEventListener('load', function() {
560             gl.bindTexture(gl.TEXTURE_2D, texture);
561             gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, format, type, image);
562             gl.generateMipmap(gl.TEXTURE_2D);
563             drawScene();
564         });
```

Per side tetrahedron image texturing

2. Change texture coordinates

```
487  var texcoords = [
488      0.25, 0.00,
489      0.00, 1.00,
490      0.50, 1.00,
491
492      0.75, 0.00,
493      0.25, 0.00,
494      0.50, 1.00,
495
496      0.50, 1.00,
497      1.00, 1.00,
498      0.75, 0.00,
499
500      0.75, 0.00,
501      1.00, 1.00,
502      1.25, 0.00,
503  ];
```

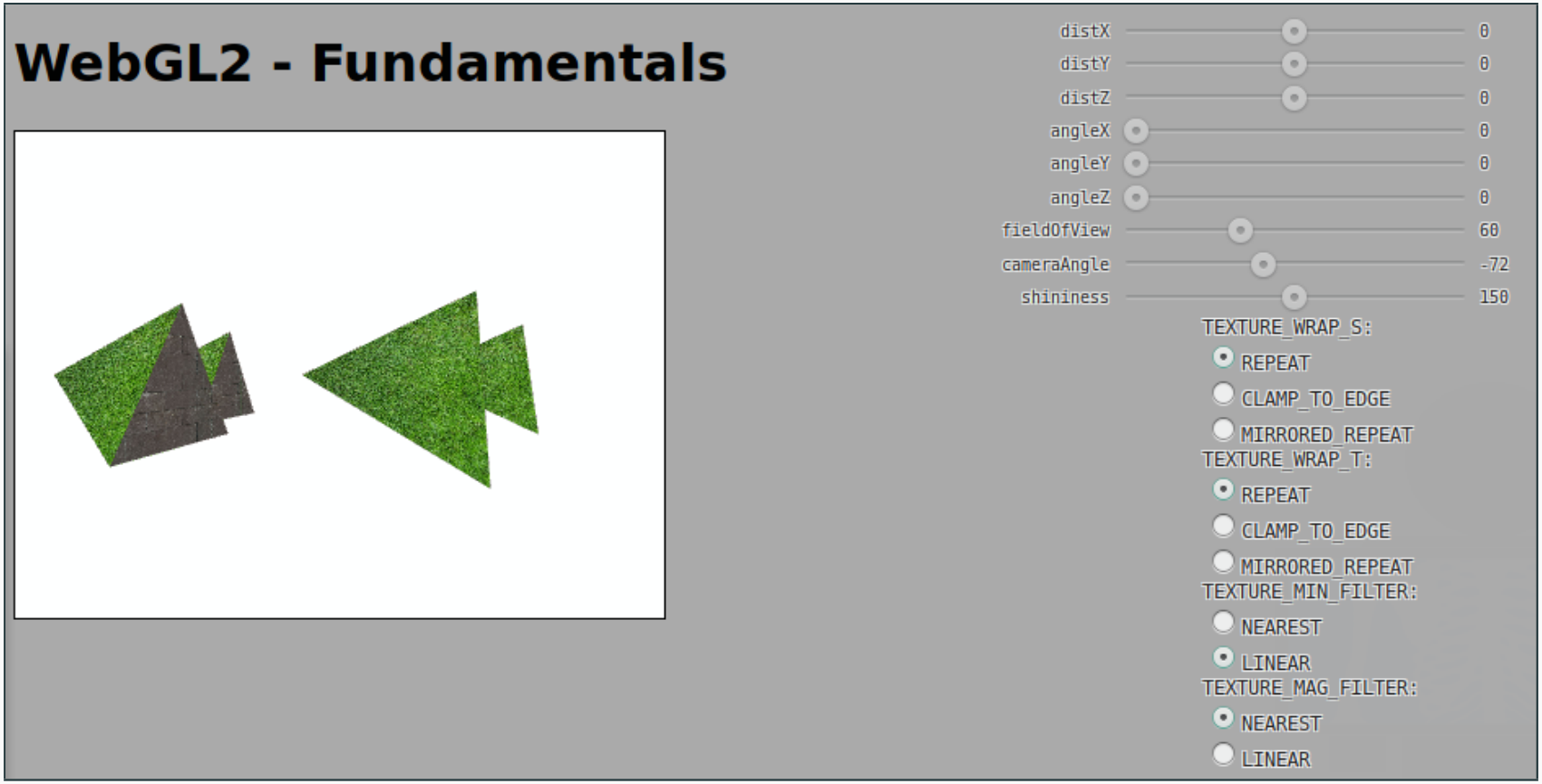
```
505  var textureWidth = 1024;
506  var textureHeight = 512;
507  var w =  textureWidth-1;
508  var h = textureHeight-1;
509  var texcoordsAbs = [
510       256/w,   0/h,
511         0/w, 512/h,
512       512/w, 512/h,
513
514       768/w,   0/h,
515       256/w,   0/h,
516       512/w, 512/h,
517
518       512/w, 512/h,
519      1024/w, 512/h,
520       768/w,   0/h,
521
522       768/w,   0/h,
523      1024/w, 512/h,
524      1280/w,   0/h,
525  ];
```

3. Set the correct texture wrapping parameter

Set `gl.TEXTURE_WRAP_S` to `gl.REPEAT`

Per side tetrahedron image texturing

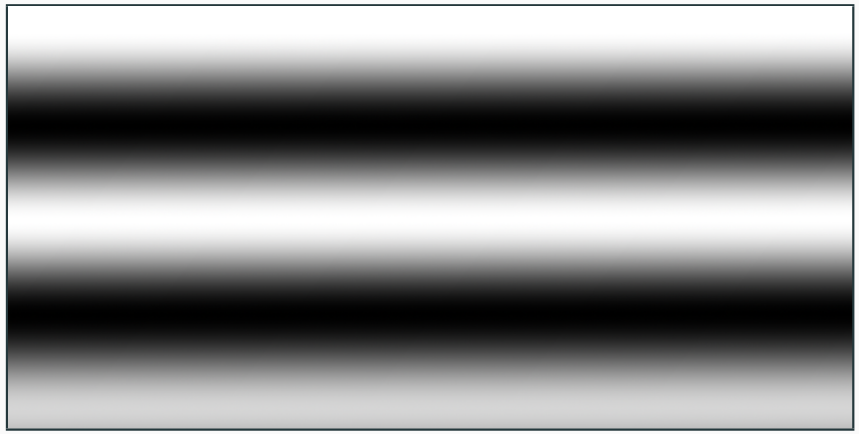
See File: `04-example-3D-Add_Texture-ST_Wrapping_MinMag_Filter_Mips_TextureAtlas.html`



Multiple textures for a single object

Texturing each side with two textures

1. Load two textures
2. Create two WebGL texture objects
3. Change the shader to use two textures
4. Lookup shader locations and set texture units
5. Bind textures to texture units



Texturing each side with two textures

1. Load two textures 2. Create two WebGL texture objects

```
527     var texture0 = gl.createTexture();
528     gl.bindTexture(gl.TEXTURE_2D, texture0);
529     var mip = 0;
530     var internalFormat = gl.RGBA;
531     var width = 1, height = 1, border = 0;
532     var format = gl.RGBA;
533     var type = gl.UNSIGNED_BYTE;
534
535     gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, width, height, border, format, type,
536         new Uint8Array([0, 255, 0, 255]));
537
538     var image0 = new Image();
539     image0.src = "./resources/textureatlas.png";
540     image0.addEventListener('load', function() {
541         gl.bindTexture(gl.TEXTURE_2D, texture0);
542         gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, format, type, image0);
543         gl.generateMipmap(gl.TEXTURE_2D);
544         drawScene();
545     });
546
547     var texture1 = gl.createTexture();
548     gl.bindTexture(gl.TEXTURE_2D, texture1);
549     gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, width, height, border, format, type,
550         new Uint8Array([255, 0, 0, 255]));
551
552     var image1 = new Image();
553     image1.src = "./resources/textureatlasOverlay.png";
554     image1.addEventListener('load', function() {
555         gl.activeTexture(gl.TEXTURE1);
556         gl.bindTexture(gl.TEXTURE_2D, texture1);
557         gl.texImage2D(gl.TEXTURE_2D, mip, internalFormat, format, type, image1);
558         gl.generateMipmap(gl.TEXTURE_2D);
559         drawScene();
560     });
```

Texturing each side with two textures

3. Change the shader to use two textures

```
87  var fragmentShaderSource = `#version 300 es
88
89  // fragment shaders don't have a default precision so we need
90  // to pick one. mediump is a good default. It means "medium precision"
91  precision mediump float;
92
93  in vec2 v_texcoord;
94
95  uniform sampler2D u_image0;
96  uniform sampler2D u_image1;
97
98  // we need to declare an output for the fragment shader
99  out vec4 outColor;
100
101 void main() {
102   vec4 color0 = texture(u_image0, v_texcoord);
103   vec4 color1 = texture(u_image1, v_texcoord);
104
105   outColor = color1 * color0;
106 }
107 `;
```

Texturing each side with two textures

4. Lookup shader locations and set texture units

- A texture unit is a reference to a texture

```
430         // look up uniform locations
431         var positionAttributeLocation     = gl.getAttribLocation(program, "a_position");
432         var texcoordAttributeLocation = gl.getAttribLocation(program, "a_texcoord");
433
434         var u_image0Location = gl.getUniformLocation(program, "u_image0");
435         var u_image1Location = gl.getUniformLocation(program, "u_image1");
```

```
661
662             // Tell it to use our program (pair of shaders)
663             gl.useProgram(program);
664
665             gl.uniform1i(u_image0Location, 0);  // texture unit 0
666             gl.uniform1i(u_image1Location, 1);  // texture unit 1
667
668             // Bind the attribute/buffer set we want.
669             gl.bindVertexArray(vao);
```

Texturing each side with two textures

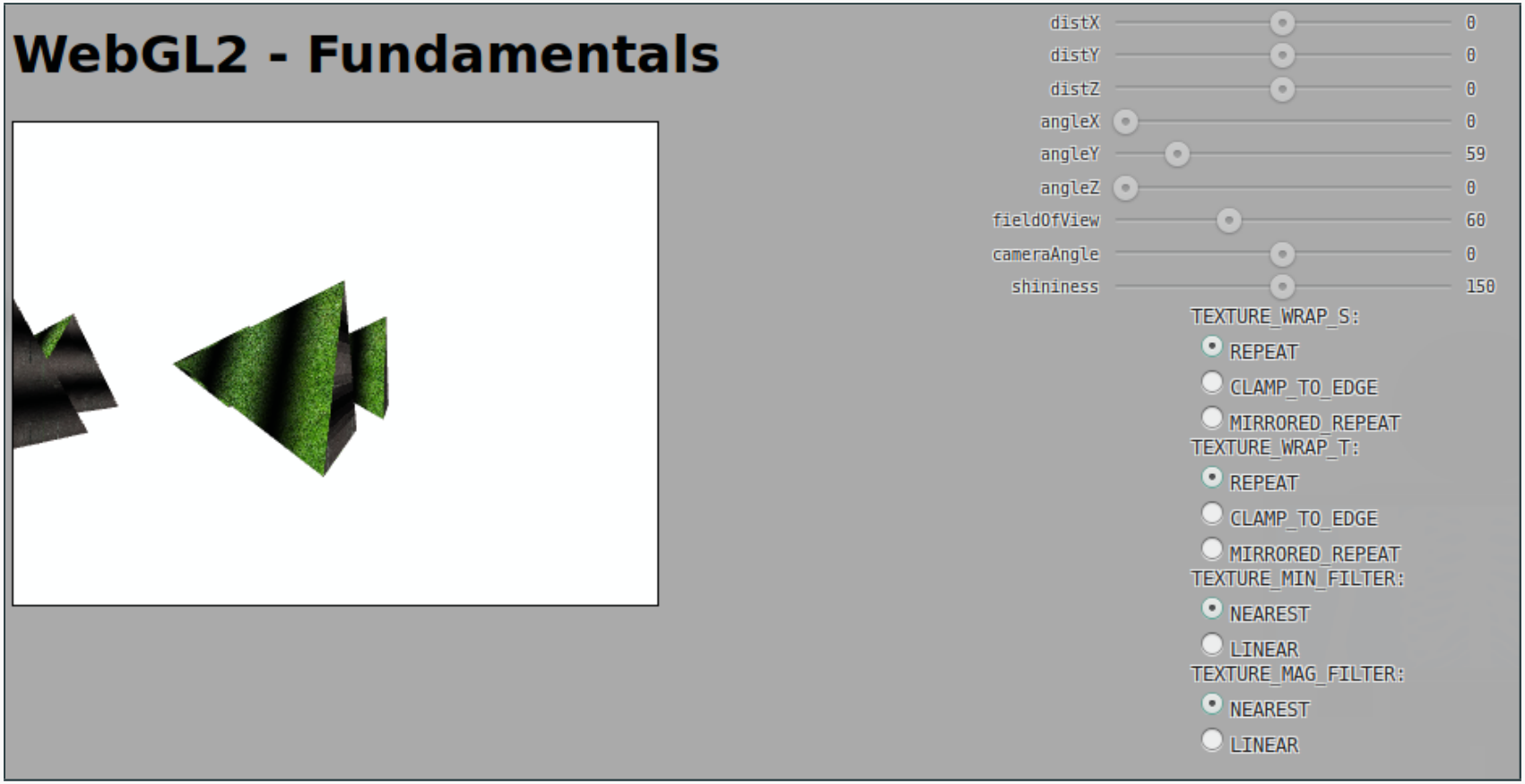
5. Bind textures to texture units

   - You tell the shader which texture unit to use for each sampler

```
756                 gl.activeTexture(gl.TEXTURE0);
757                 gl.bindTexture(gl.TEXTURE_2D, texture0);
758                 gl.activeTexture(gl.TEXTURE1);
759                 gl.bindTexture(gl.TEXTURE_2D, texture1);
760
761                 // draw
762                 var primitiveType = gl.TRIANGLES;
763                 var offset = 0;
764                 var count = 12;
765                 gl.drawArrays(primitiveType, offset, count);
```

Multiple textures per side

See File: `05-example-3D-Add_Texture-ST_Wrapping_MinMag_Filter_Mips_TextureAtlas_MultiTexture.html`



Final result with light enabled

See File: `06-example-3D-Add_Texture-ST_Wrapping_MinMag_Filter_Mips_TextureAtlas_MultiTexture_LightEnabled.html`