

Desenho de Objetos com WebGL

2019-2020
Fernando Birra

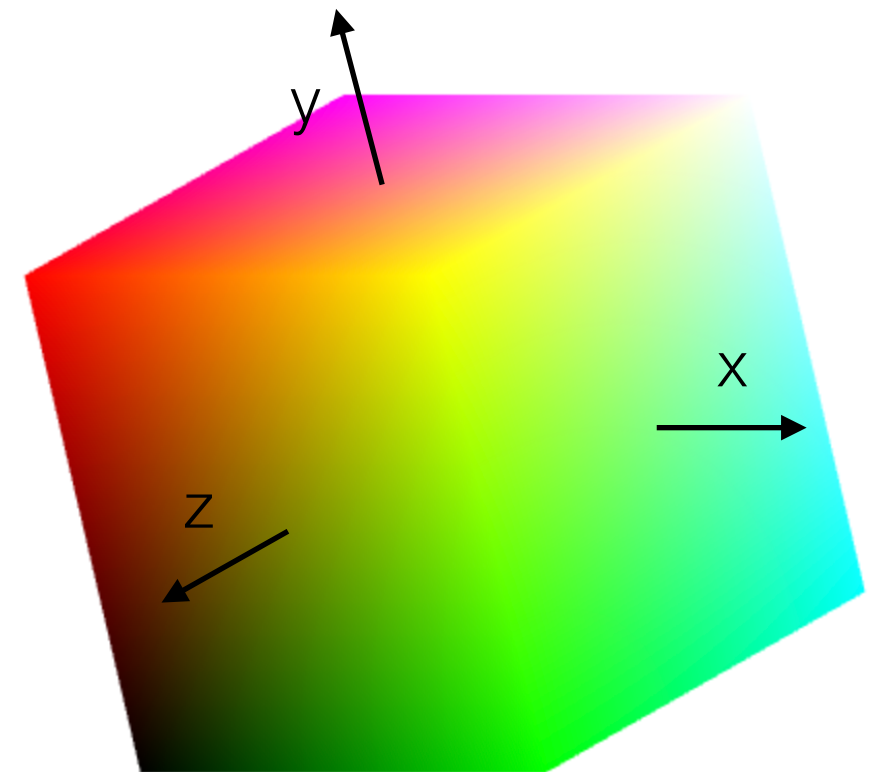
Objetivos

- Ilustrar a construção dum objeto modelado como um B-Rep, usando WebGL.
- Analisar duas formas de indicação da geometria:
 - Usando arrays
 - Usando elementos (índices)

Modelação dum cubo

- Um array com os vértices:

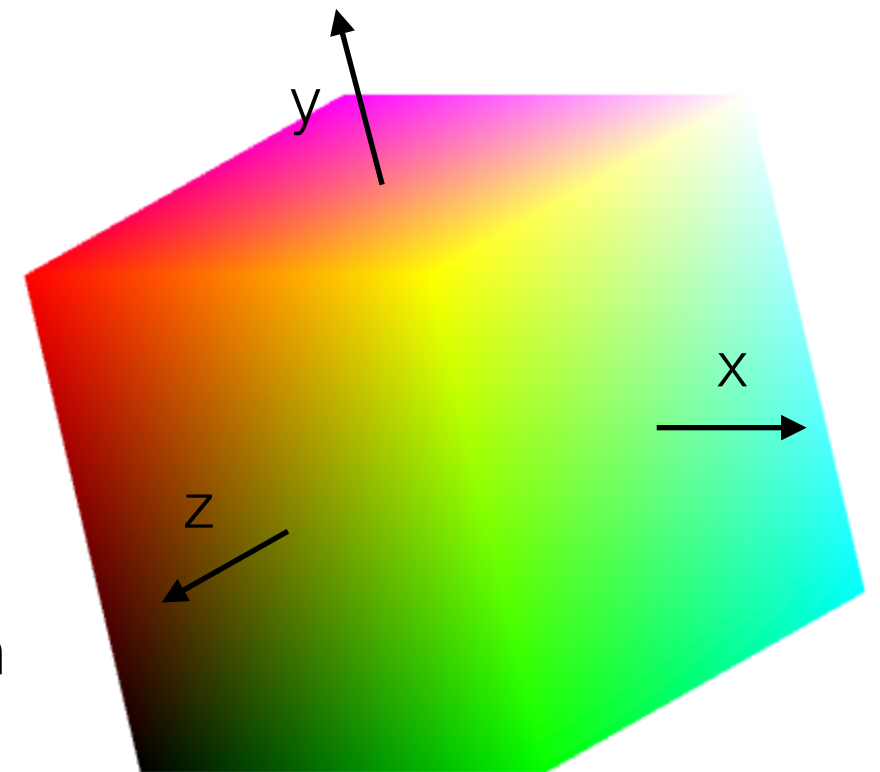
```
var vertices = [  
  vec4( -0.5, -0.5,  0.5, 1.0 ),  
  vec4( -0.5,  0.5,  0.5, 1.0 ),  
  vec4(  0.5,  0.5,  0.5, 1.0 ),  
  vec4(  0.5, -0.5,  0.5, 1.0 ),  
  vec4( -0.5, -0.5, -0.5, 1.0 ),  
  vec4( -0.5,  0.5, -0.5, 1.0 ),  
  vec4(  0.5,  0.5, -0.5, 1.0 ),  
  vec4(  0.5, -0.5, -0.5, 1.0 )  
];
```



Modelação dum cubo

- Um array com as cores:

```
var vertexColors = [  
  [ 0.0, 0.0, 0.0, 1.0 ], // black  
  [ 1.0, 0.0, 0.0, 1.0 ], // red  
  [ 1.0, 1.0, 0.0, 1.0 ], // yellow  
  [ 0.0, 1.0, 0.0, 1.0 ], // green  
  [ 0.0, 0.0, 1.0, 1.0 ], // blue  
  [ 1.0, 0.0, 1.0, 1.0 ], // magenta  
  [ 1.0, 1.0, 1.0, 1.0 ], // white  
  [ 0.0, 1.0, 1.0, 1.0 ] // cyan  
];
```

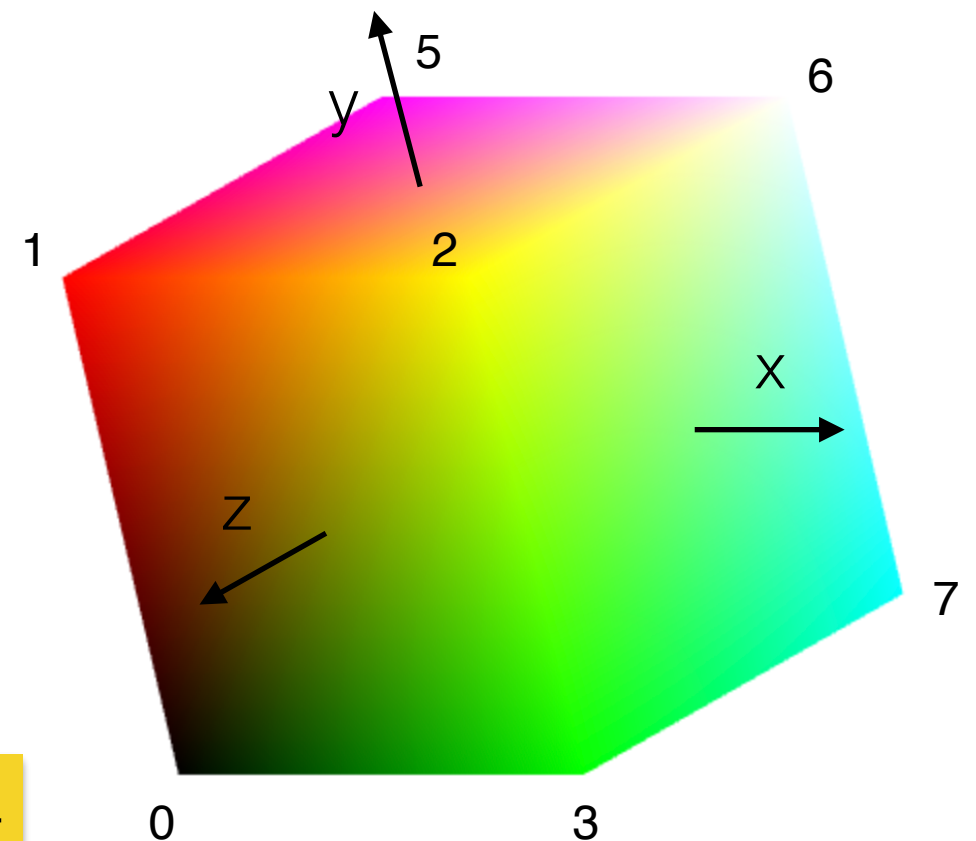


Modelação dum cubo

- Construção das faces:

```
function colorCube()  
{  
    quad( 1, 0, 3, 2 );  
    quad( 2, 3, 7, 6 );  
    quad( 3, 0, 4, 7 );  
    quad( 6, 5, 1, 2 );  
    quad( 4, 5, 6, 7 );  
    quad( 5, 4, 0, 1 );  
}
```

Os vértices estão ordenados por forma a obter normais que apontam para fora do objeto.
Cada chamada de `quad( )` gera 2 triângulos.



Modelação dum cubo

- Inicialização:

```
var canvas, gl;  
var numVertices = 36;  
var points = [];  
var colors = [];
```

```
window.onload = function init(){  
    canvas = document.getElementById( "gl-canvas" );  
    gl = WebGLUtils.setupWebGL( canvas );  
  
    colorCube();  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
    gl.enable(gl.DEPTH_TEST);
```

→ Necessário para ativação do Z-Buffer
(Hidden Surface Removal)

Modelação dum cubo

- Preencher os arrays `points` e `colors` com a os dados em `vertices` e `vertexColors` para os 2 triângulos do quad:

```
function quad(a, b, c, d)
{
    var indices = [ a, b, c, c, d, a ];
    for ( var i = 0; i < indices.length; ++i ) {

        points.push( vertices[indices[i]]);
        colors.push( vertexColors[indices[i]] );

        // for solid colored faces use
        //colors.push(vertexColors[a]);
    }
}
```

Por cada chamada `quad()` é escrita a informação de 6 vértices nos respetivos buffers

Modelação dum cubo

- Buffer com o atributo cor:

O array “inflacionado” com as cores produzidas por quad()

```
var cBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );
```

```
var vColor = gl.getAttribLocation( program, "vColor" );  
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vColor );
```

Modelação dum cubo

- Buffer com o atributo posição:

O array “inflacionado” com as posições produzidas por quad()

```
var vBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );
```

```
var vPosition = gl.getAttribLocation( program, "vPosition" );  
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vPosition );
```

Modelação dum cubo

- Função de desenho:

```
function render(){  
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );  
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );  
    requestAnimationFrame( render );  
}
```

Limpa o conteúdo do Z-Buffer
(Necessário para Hidden Surface Removal)

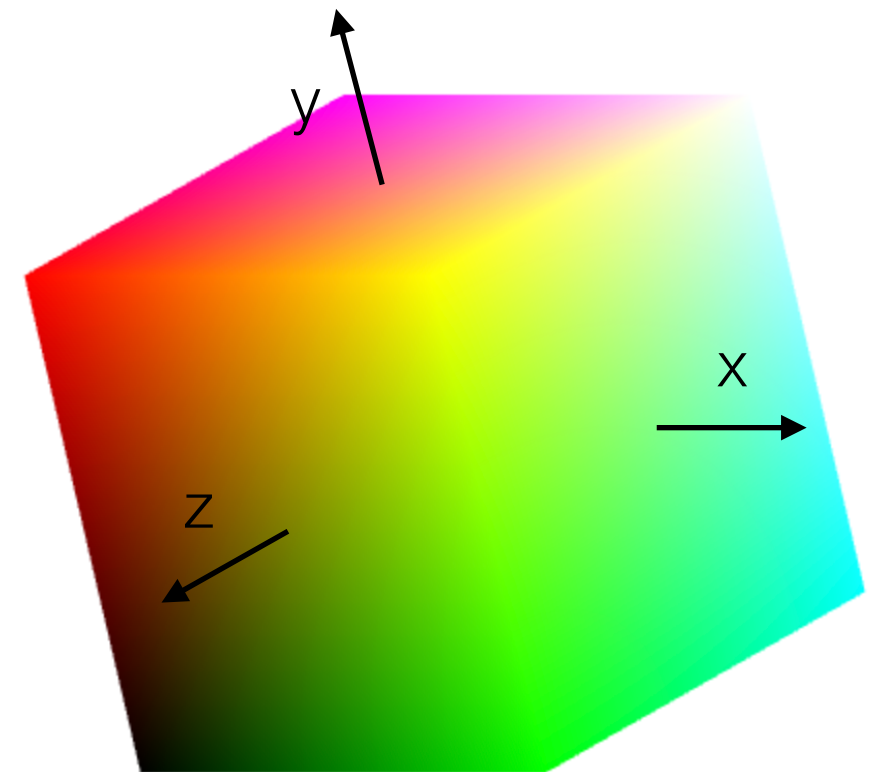


Uma alternativa melhor...

Modelação dum cubo (II)

- Um array com os vértices:

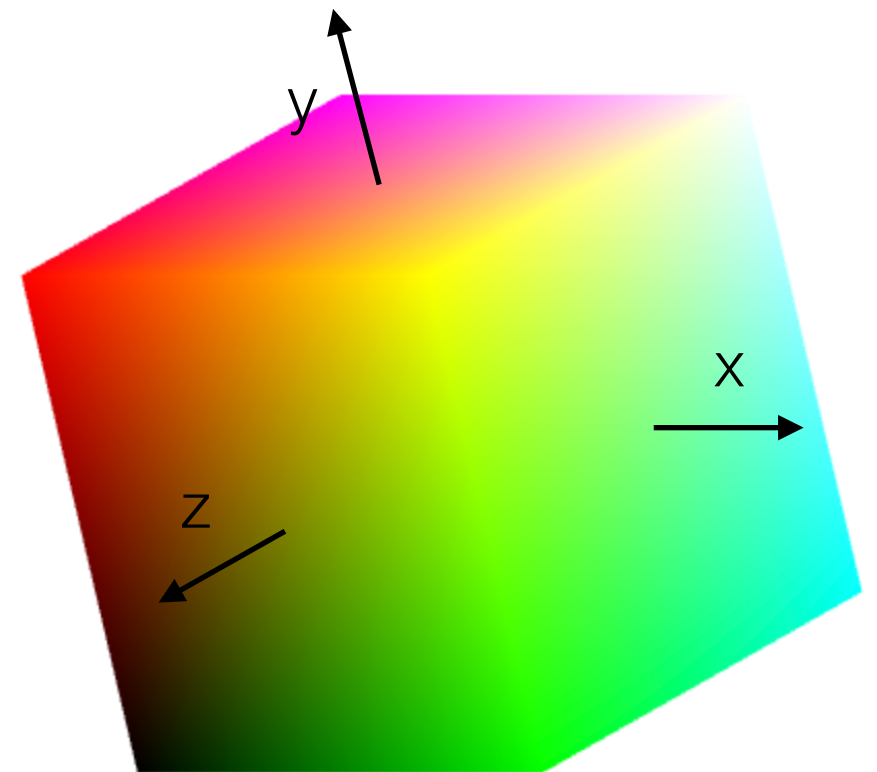
```
var vertices = [  
  vec4( -0.5, -0.5,  0.5, 1.0 ),  
  vec4( -0.5,  0.5,  0.5, 1.0 ),  
  vec4(  0.5,  0.5,  0.5, 1.0 ),  
  vec4(  0.5, -0.5,  0.5, 1.0 ),  
  vec4( -0.5, -0.5, -0.5, 1.0 ),  
  vec4( -0.5,  0.5, -0.5, 1.0 ),  
  vec4(  0.5,  0.5, -0.5, 1.0 ),  
  vec4(  0.5, -0.5, -0.5, 1.0 )  
];
```



Modelação dum cubo (II)

- Um array com as cores:

```
var vertexColors = [  
  [ 0.0, 0.0, 0.0, 1.0 ], // black  
  [ 1.0, 0.0, 0.0, 1.0 ], // red  
  [ 1.0, 1.0, 0.0, 1.0 ], // yellow  
  [ 0.0, 1.0, 0.0, 1.0 ], // green  
  [ 0.0, 0.0, 1.0, 1.0 ], // blue  
  [ 1.0, 0.0, 1.0, 1.0 ], // magenta  
  [ 1.0, 1.0, 1.0, 1.0 ], // white  
  [ 0.0, 1.0, 1.0, 1.0 ] // cyan  
];
```



Modelação dum cubo (II)

- Usar índices para os arrays `vertices/vertexColors` para descrever as faces:

```
var indices = [  
  1, 0, 3,  
  3, 2, 1,  
  2, 3, 7,  
  7, 6, 2,  
  3, 0, 4,  
  4, 7, 3,  
  6, 5, 1,  
  1, 2, 6,  
  4, 5, 6,  
  6, 7, 4,  
  5, 4, 0,  
  0, 1, 5  
];
```

← As 12 faces triangulares que irão formar o cubo são guardadas como índices numa lista de vértices.

Os índices precisam ser enviados para o GPU e a chamada de desenho necessita ser ajustada para operar sobre índices em vez de arrays de vértices.

Modelação dum cubo (II)

- Desenho com `drawElements()`:

Enviar os índices
para o GPU

```
var iBuffer = gl.createBuffer();  
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, iBuffer);  
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,  
              new Uint8Array(indices),  
              gl.STATIC_DRAW);
```

Desenhar com índices

```
gl.drawElements( gl.TRIANGLES, numVertices, gl.UNSIGNED_BYTE, 0 );
```

- Ainda poderá ser mais eficiente se se usarem *triangle strips* ou *triangle fans*.

Modelação dum cubo (II)

- Inicialização (cores):

O array original com as 8 cores

```
var cBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertexColors), gl.STATIC_DRAW );
```

```
var vColor = gl.getAttribLocation( program, "vColor" );  
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vColor );
```

Modelação dum cubo (II)

- Inicialização (posição)

O array original com as 8 posições

```
var vBuffer = gl.createBuffer();  
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );  
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );
```

```
var vPosition = gl.getAttribLocation( program, "vPosition" );  
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0 );  
gl.enableVertexAttribArray( vPosition );
```

Modelação dum cubo (II)

- Função de desenho:

```
function render(){  
    gl.clear( gl.COLOR_BUFFER_BIT |gl.DEPTH_BUFFER_BIT);  
    theta[axis] += 2.0;  
    gl.uniform3fv(thetaLoc, theta);  
    gl.drawElements( gl.TRIANGLES, numVertices, gl.UNSIGNED_BYTE, 0 );  
    requestAnimationFrame( render );  
}
```

O cubo apenas tinha 8 vértices, por isso um byte chega para os indexar (256 índices distintos). Para modelos maiores, usa-se `gl.UNSIGNED_SHORT`