Engenharia - Computação Gráfica
Prof: Luciano Soares <lpsoares@insper.edu.br>

WebGL : parte 3

Vamos agora mudar para algo realmente 3D.

1. Renderizando um Cubo

Vamos evoluir nosso código que desenha um quadrado para a terceira dimensões adicionando mais cinco faces para criar assim um cubo. Para fazer isso com eficiência, vamos mudar a rotina de desenho que está usando o método gl.drawArrays() para usar a matriz de vértices diretamente com a o método gl.drawElements(), que trará mais flexibilidade para o sistema.

Como queremos que cada face tenha uma cor diferente neste primeiro exemplo, teremos de passar o mesmo vértice 3 vezes com variações nos valores de sua cor (embora a posição seja repetida).

Primeiro, vamos construir um buffer das posições dos vértices do cubo, para isso atualize o código em initBuffers(). Perceba que é praticamente o mesmo que no desenho do quadrado, mas um pouco mais longo, pois há 24 vértices (4 de cada lado):

```
const positions = [
    // face frontal
    -1.0, -1.0,  1.0,
    1.0, -1.0,  1.0,
    1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,

    // face traseira
    -1.0, -1.0, -1.0,
    -1.0,  1.0, -1.0,
    1.0,  1.0, -1.0,
    1.0, -1.0, -1.0,

    // face superior
    -1.0,  1.0, -1.0,
    -1.0,  1.0,  1.0,
    1.0,  1.0,  1.0,
    1.0,  1.0, -1.0,

    // face inferior
    -1.0, -1.0, -1.0,
    1.0, -1.0, -1.0,
    1.0, -1.0,  1.0,
    -1.0, -1.0,  1.0,
```

```
    // face direita
    1.0, -1.0, -1.0,
    1.0,  1.0, -1.0,
    1.0,  1.0,  1.0,
    1.0, -1.0,  1.0,

    // face esquerda
    -1.0, -1.0, -1.0,
    -1.0, -1.0,  1.0,
    -1.0,  1.0,  1.0,
    -1.0,  1.0, -1.0,
];
```

Como adicionamos o componente z aos vértices desta vez, precisamos atualizar o numComponents de atributo vertexPosition para 3.

```
{
    const numComponents = 3;  // passa 3 valores por vez
    const type = gl.FLOAT;    // os dados no buffer são floats de 32bit
```

Também precisamos construir um vetor de cores para os vértices. Este código começa definindo uma cor para cada face, então usa um loop para montar um vetor das cores para cada um dos vértices.

```
        // Passe a lista de posições para o WebGL
        gl.bufferData(gl.ARRAY_BUFFER,
                      new Float32Array(positions),
                      gl.STATIC_DRAW);

        // Definindo uma cor para cada face do cubo
        const faceColors = [
            [1.0,  1.0,  1.0,  1.0],    // Face frontal: branca
            [1.0,  0.0,  0.0,  1.0],    // Face traseira: vermelha
            [0.0,  1.0,  0.0,  1.0],    // Face superior: verde
            [0.0,  0.0,  1.0,  1.0],    // Face inferior: azul
            [1.0,  1.0,  0.0,  1.0],    // Face direita: amarela
            [1.0,  0.0,  1.0,  1.0],    // Face esquerda: violeta
        ];

        // Convertendo o vetor de cores para uma matriz para todos os 24 vértices
        var colors = [];

        for (var j = 0; j < faceColors.length; ++j) {
            const c = faceColors[j];

            // Repita cada cor 4 vezes para os 4 vértices das faces do cubo
            colors = colors.concat(c, c, c, c);
```

```
        }

        const colorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
```

Uma vez que os vetores de vértices foram gerados, precisamos construir o vetor de elementos.

```
        const colorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);

        // Contruindo vetor de elementos que especifica os indices dos vértices
        // para cada face do cubo
        const indexBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);

        // Este vetor define cada face como dois triângulos, usando os
        // indices no vetor de vértices para especificar cada triângulo
        const indices = [
            0,  1,  2,      0,  2,  3,    // frente
            4,  5,  6,      4,  6,  7,    // trás
            8,  9,  10,     8,  10, 11,   // superio
            12, 13, 14,     12, 14, 15,   // inferior
            16, 17, 18,     16, 18, 19,   // direita
            20, 21, 22,     20, 22, 23,   // esquerda
        ];

        // Enviando os elemntos para o vetor GL
        gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
            new Uint16Array(indices), gl.STATIC_DRAW);

        return {
            position: positionBuffer,
            color: colorBuffer,
            indices: indexBuffer,
        };
    }
```

A matriz de índices define cada face como um par de triângulos, especificando os vértices de cada triângulo como um índice do vetor dos vértices do cubo. Assim, o cubo é descrito como uma coleção de 12 triângulos.

Em seguida, precisamos adicionar código para à nossa função drawScene()desenhar usando o buffer dos índices do cubo. Para isso devemos adicionar as chamadas gl.bindBuffer() e gl.drawElements():

```
        // Diga ao WebGL quais indices usar para conectar os vértices
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffers.indices);

        // Diga ao WebGL para usar nosso programa ao desenhar
        gl.useProgram(programInfo.program);

        // Defina os uniforms dos shaders
        gl.uniformMatrix4fv(
            programInfo.uniformLocations.projectionMatrix,
            false,
            projectionMatrix);
        gl.uniformMatrix4fv(
            programInfo.uniformLocations.modelViewMatrix,
            false,
            modelViewMatrix);

        {
            const offset = 0;
            const vertexCount = 36;
            const type = gl.UNSIGNED_SHORT;
            gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
        }

    }
```

Como cada face do cubo é composta por dois triângulos, são usados 6 vértices por lado, ou 36 vértices totais no cubo, embora muitos deles sejam os mesmos.

Finalmente, vamos adicionar uma segunda rotação em torno do eixo y para ficar mais interessante o movimento, vamos fazer essa segunda rotação um pouco mais lenta multiplicando-a por 0.7:

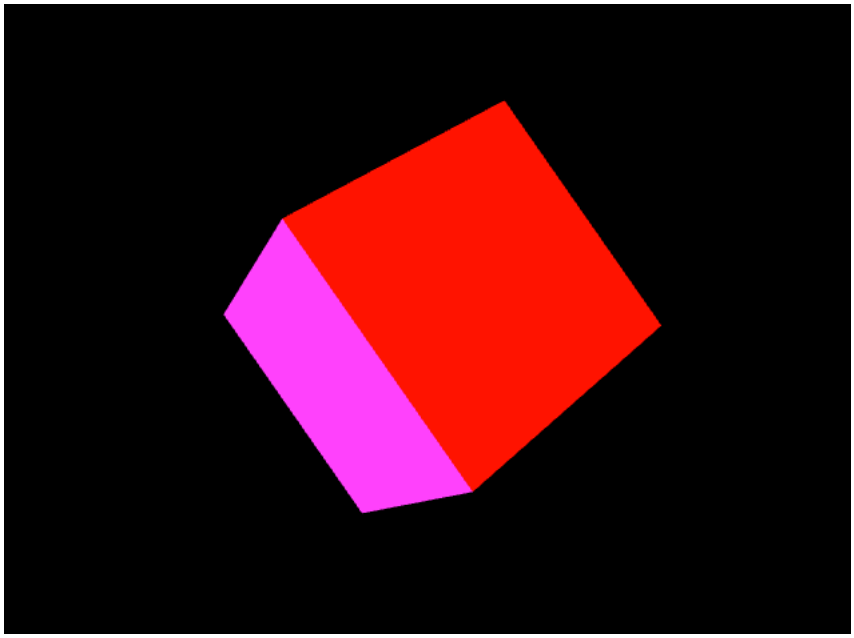
```
        mat4.rotate(modelViewMatrix,  // matriz de destino
                    modelViewMatrix,  // matriz para rotacionar
                    squareRotation,   // quantidade de radianos a se rotacionar
                    [0, 0, 1]);       // eixo para ser girar em volta

        mat4.rotate(modelViewMatrix,  // matriz de destino
                    modelViewMatrix,  // matriz para rotacionar
                    squareRotation*.7,// quantidade de radianos a se rotacionar *0.7
                    [0, 1, 0]);       // eixo para ser girar em volta
```

No final você deve conseguir ver um cubo rotacionando, se quiser modifique as rotações para ter uma melhor visualização de todas as faces.

Engenharia - Computação Gráfica

Prof: Luciano Soares <lpsoares@insper.edu.br>



Referências:

Esse documento foi baseado em:

- WebGL tutorial : https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial