Ciências ULisboa

Faculdade de Ciências da Universidade de Lisboa

Relatório 2º Trabalho de Computação Gráfica 2020/2021

WebGL

Grupo 18 - Computação Gráfica:

António Fróis nº 51050

José Alves nº 44898

Miguel Cruz nº 43266

Notas

Para este trabalho foi utilizado como base o ficheiro webgl-demo-2obj.js, colocado pelo professor no fórum e o ‘Sample7’. Devido a confusões ao começar o trabalho usámos também o Sample5 então nos exercícios em que era necessário computar luz foi usado o Sample7. Quando nos apercebemos desta falha já não havia tempo para mudar, daí os exercícios irem por pastas e poderem ter mais que 1 ficheiro js.

Exercício 1

Exercício 1.1

Foi aproveitado o 1º cubo já existente, mudando apenas a cor do mesmo para verde, conforme se encontra na figura seguinte.

```
const faceColors = [
  [0.0,  1.0,  0.0,  1.0],    // Front face: green
  [0.0,  1.0,  0.0,  1.0],    // Back face: green
  [0.0,  1.0,  0.0,  1.0],    // Top face: green
  [0.0,  1.0,  0.0,  1.0],    // Bottom face: green
  [0.0,  1.0,  0.0,  1.0],    // Right face: green
  [0.0,  1.0,  0.0,  1.0],    // Left face: green
];
```

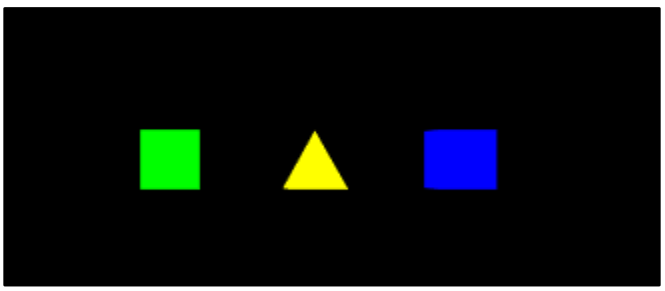
Para a pirâmide foi alterado o código das posições já existente para o cubo anterior, eliminando a face superior, relativamente às faces laterais, os pontos da base permaneceram iguais, foi eliminado um ponto do topo e colocado o ponto superior de cada face lateral na mesma coordenada a fim de as faces laterias se tornarem um triangulo. A face 'Bottom' permaneceu inalterada a fim de ser a base da pirâmide (conforme o código na figura ao lado).

```
const positions = [
  // Lateral face
   0.0,  1.0,  0.0,
  -1.0, -1.0,  1.0,
   1.0, -1.0,  1.0,

  // Lateral face
  0.0,  1.0,  0.0,
  1.0, -1.0, -1.0,
 -1.0, -1.0, -1.0,

  // Lateral face
   0.0,  1.0,  0.0,
   1.0, -1.0,  1.0,
   1.0, -1.0, -1.0,

  // Lateral face
   0.0,  1.0,  0.0,
   1.0, -1.0, -1.0,
  -1.0, -1.0,  1.0,

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

Foi também alterado as cores das faces da pirâmide para amarelo, conforme a figura em baixo.

```
const faceColors = [
  [1.0,  1.0,  0.0,  1.0],    // Lateral face: yellow
  [1.0,  1.0,  0.0,  1.0],    // Lateral face: yellow
  [1.0,  1.0,  0.0,  1.0],    // Lateral face: yellow
  [1.0,  1.0,  0.0,  1.0],    // Lateral face: yellow
  [1.0,  1.0,  0.0,  1.0],    // Bottom face: yellow
];
```

De salientar que não foi realizada a esfera, no entanto a fim de realizar os exercícios seguintes foi colocado um cubo azul no seu lugar.

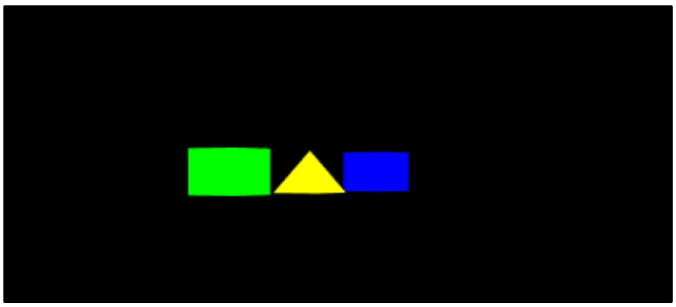


Exercicio 1.3

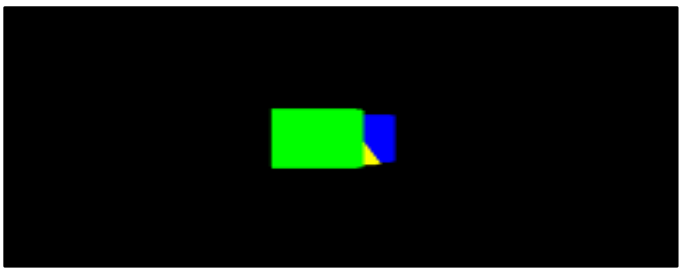
A fim de modificar a posição da câmara de modo a apresentar uma visão lateral foi utilizada a função ‘rotateY’ sobre a viewMatrix, que roda a matriz em torno do eixo do Y.

```
  mat4.translate(viewMatrix,     // destination matrix
    viewMatrix,     // matrix to translate
    [-5.0, 0.0, -40.0]);  // amount to translate
mat4.rotateY(viewMatrix,     // destination matrix
    viewMatrix,     // matrix to ROTATE
    1.57);  // amount to ROTATE
```

A fim de mostrar a rotação foi primeiro realizada uma rotação de 60º e posteriormente de 90º conforme as seguintes imagens.



Rotação de 60º



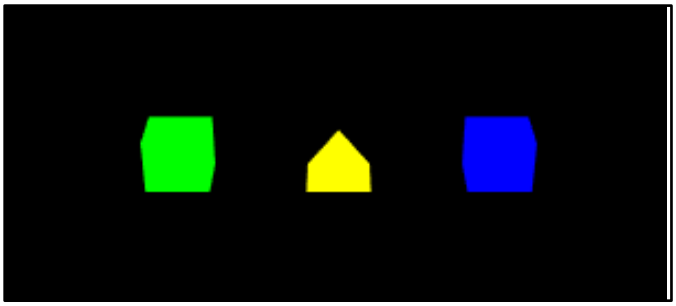
Rotação de 90º

Exercício 1.4

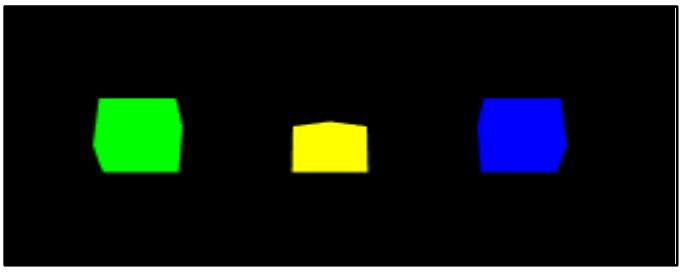
A fim de modificar a posição da câmara de modo a apresentar uma visão de topo foi utilizada a função ‘rotateX’ sobre a viewMatrix, que roda a matriz em torno do eixo do X.

```
  const viewMatrix = mat4.create();
  mat4.translate(viewMatrix,     // destination matrix
    viewMatrix,     // matrix to translate
    [-5.0, 0.0, -40.0]);  // amount to translate
mat4.rotateX(viewMatrix,     // destination matrix
    viewMatrix,     // matrix to ROTATE
    1.57);  // amount to ROTATE
```

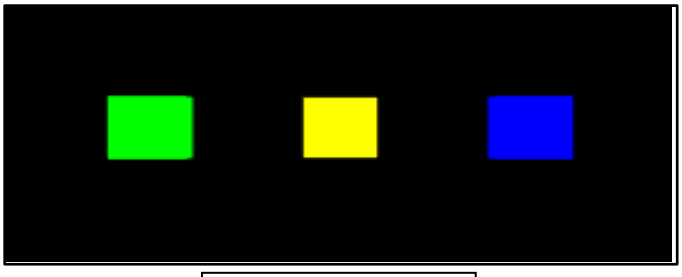
A fim de mostrar a rotação foi primeiro realizada uma rotação de 45º e posteriormente de 60º e 90º conforme as seguintes imagens.



Rotação de 45º



Rotação de 60º



Rotação de 90º

Exercício 2

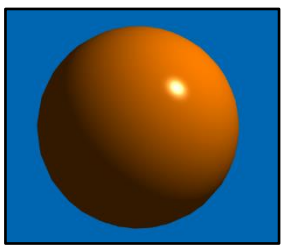
Houve grande dificuldade na realização do exercício 2 no que dizia respeito à iluminação. Tentou-se aplicar o método Phong Shading através do seguinte código:

```
vec4 vertPos4 = uPMatrix * vec4(aVertexNormal, 1.0);
vertPos = vec3(vertPos4) / vertPos4.w;
normalInterp = vec3(uNMatrix * vec4(aVertexNormal, 0.0));
gl_Position = uMVMatrix * vertPos4;
```

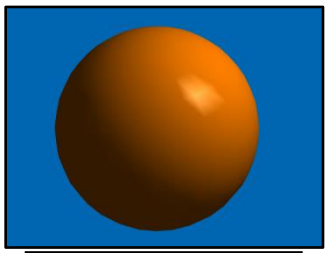
```
vec3 N = normalize(normalInterp);
vec3 L = normalize(lightPos - vertPos);

// Lambert's cosine law
float lambertian = max(dot(N, L), 0.0);
float specular = 0.0;
if(lambertian > 0.0) {
  vec3 R = reflect(-L, N);      // Reflected light vector
  vec3 V = normalize(-vertPos); // Vector to viewer
  // Compute the specular term
  float specAngle = max(dot(R, V), 0.0);
  specular = pow(specAngle, shininessVal);
}
gl_FragColor = vec4(Ka * ambientColor +
                    Kd * lambertian * diffuseColor +
                    Ks * specular * specularColor, 1.0);

// only ambient
if(mode == 2) gl_FragColor = vec4(Ka * ambientColor, 1.0);
// only diffuse
if(mode == 3) gl_FragColor = vec4(Kd * lambertian * diffuseColor, 1.0);
// only specular
if(mode == 4) gl_FragColor = vec4(Ks * specular * specularColor, 1.0);
```

A prática pode não ter corrido como esperado, mas, de uma perspetiva mais teórica, se compararmos o método Shading de Phong com o método Shading de Gouraud podemos facilmente perceber que em termos de resultado visual o primeiro é mais eficaz.

Enquanto que o Shading de Gouraud interpola as cores a partir das cores dos vértices, o Shading Phong utiliza as normais, calcula explicitamente a cor em cada ponto da imagem. Isto faz com que seja mais rigoroso e apresente zonas de highlight. Podemos ver essa diferença no ponto iluminado das esferas da imagem.



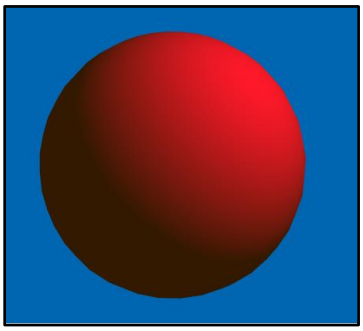
Phong Shading



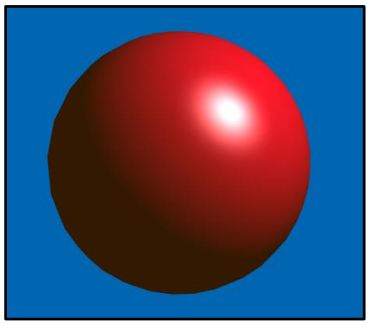
Gouraud Shading

Se tentarmos analisar numa perspetiva mais temporal ou de recursos, podemos concluir que o Shading de Gouraud é melhor. Isto porque maior qualidade de imagem implica maior capacidade de computação e de tempo.

A aplicação de uma componente especular num objeto faz com que este fique com um efeito mais espelhado. É através deste que ele passa a refletir a luz que lhe chega e a não ser apenas iluminado pela luz ambiente e difusa. Podemos verificar a diferença nas imagens abaixo.

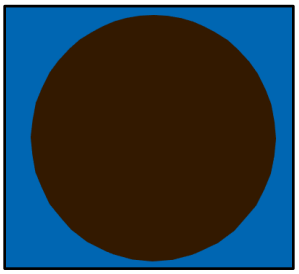


Sem Iluminação Especular

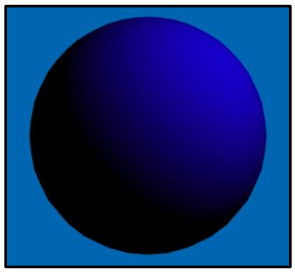


Com Iluminação Especular

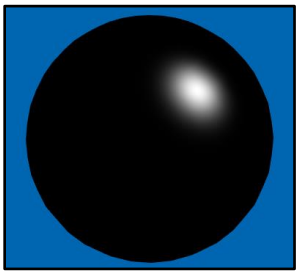
Analisando as componentes somente com um tipo de iluminação, podemos observar as diferenças na imagem abaixo.



Iluminação Ambiente



Iluminação Difusa



Iluminação Especular

Exercício 3

Exercício 3.1 (usado sample7)

Para rodar a luz em torno do cubo foi aplicada à matriz das normais onde se computa a luz uma rotação no eixo dos y. Esta é feita na matriz do cubo mas guardada na da luz de modo a que seja a luz a rodar.

```
//ROTACAO LUZ

mat4.rotate(normalMatrix,modelViewMatrix,cubeRotation, [0,1,0]);
```

Exercício 3.2 (usado sample5)

De modo a se fazer rodar a câmara em torno da cena, após esta ter sido criada foram-lhe aplicadas duas operações: uma translação de modo a afastar a câmara da origem e de seguida uma rotação em torno do eixo dos X de modo à câmara começar a andar à volta da cena. O ângulo vai sendo aumentando a cada iteração da função render e então o ângulo em que está a matriz da câmara também vai sendo alterado.

```
// Set the drawing position to the "identity" point, which is
// the center of the scene.
const viewMatrix = mat4.create();
mat4.translate(viewMatrix,     // destination matrix
  viewMatrix,     // matrix to translate
  [-0.0, 0.0, -30.0]);  // amount to translate
//Rotacao matriz da camara
mat4.rotate(viewMatrix, viewMatrix, cubeRotation, [0.0, 1.0, 0.0]);
```

Link para a animação: https://youtu.be/tIUosbducNA

Exercício 3.3 (usado sample5)

Para esta alínea é usado o mesmo método que na alínea anterior, mas com ligeiras alterações. A forma de atualizar o ângulo de rotação mantém-se, já a forma de fazer a rotação é que é alterada. De modo a se fazer com o cubo verde gire à volta da pirâmide é-lhe aplicada uma rotação seguida de uma translação, mas para conseguirmos que o cubo gire realmente à volta da pirâmide temos que lhe aplicar a rotação através da função na figura abaixo:

```
//ng: e agora, depois de ter desenhado o obj1, vamos repetir alguns
const modelMatrix2 = mat4.create();
mat4.translate(modelMatrix2, modelMatrix2, [5,0,0]);

// Rotacao da matriz do cubo em relacao a da piramide no eixo dos z
// ------------------------EX3.3
mat4.rotate(modelMatrix2,modelMatrix,cubeRotation, [0,1,0]);
mat4.translate(modelMatrix2, modelMatrix2, [7,0,0]);
//-----------------------------
```

Nesta função de rotação, a transformação é aplicada à matriz da pirâmide, mas guardada na matriz do cubo, ou seja, estamos a rodar o cubo em torno da pirâmide. De seguida é aplicada uma translação ao cubo. Esta duas operações criam a rotação do cubo em torno da pirâmide.

O mesmo processo é efetuado de modo ao cubo azul girar em torno do cubo verde como se verifica no código abaixo:

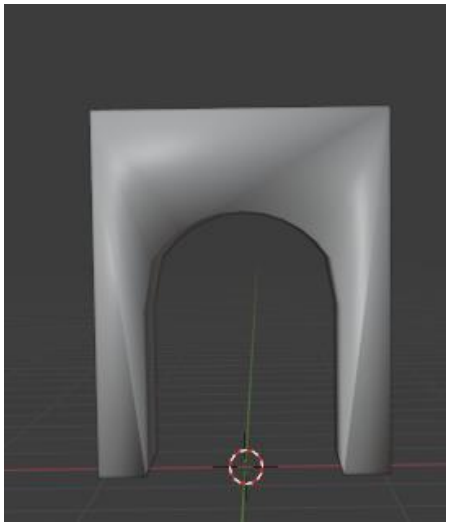
```
const modelMatrix3 = mat4.create();

  mat4.translate(modelMatrix3, modelMatrix3, [10,0,0]);

  // Rotacao da matriz do cubo em relacao a do cubo1 no eixo dos z
  // ------------------------EX3.3
  mat4.rotate(modelMatrix3,modelMatrix2,cubeRotation, [0,1,0]);
  mat4.translate(modelMatrix3, modelMatrix3, [3,0,0]);
  // -----------------------------
```

Como se verifica, é aplicada a rotação ao cubo verde (modelMatrix2) e guardada na matriz do cubo azul e de seguida aplicada uma translação, o que faz com que o cubo azul gire em torno do cubo verde.

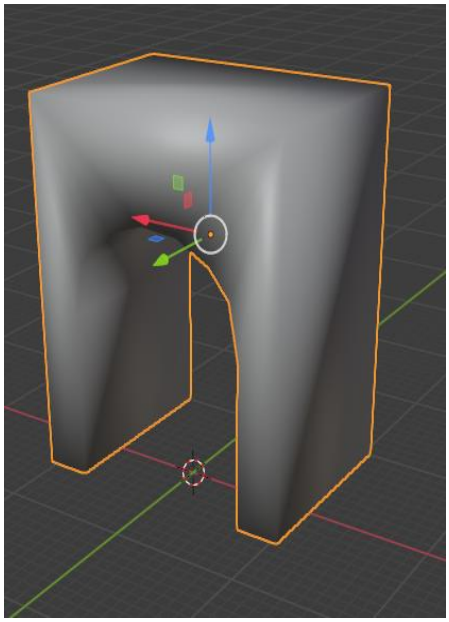
Link para a animação: https://youtu.be/J95-T8mjJRc

Exercício 4

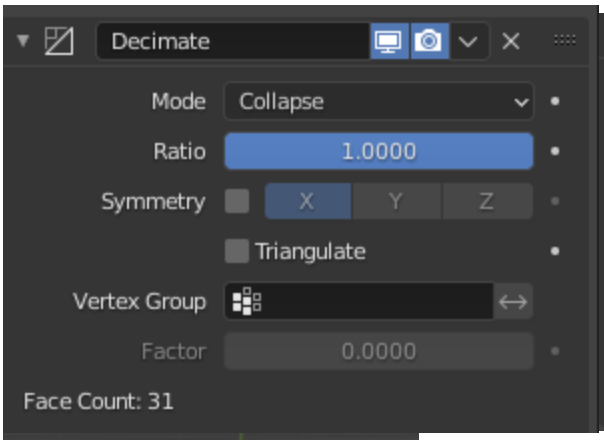
Exercício 4.1

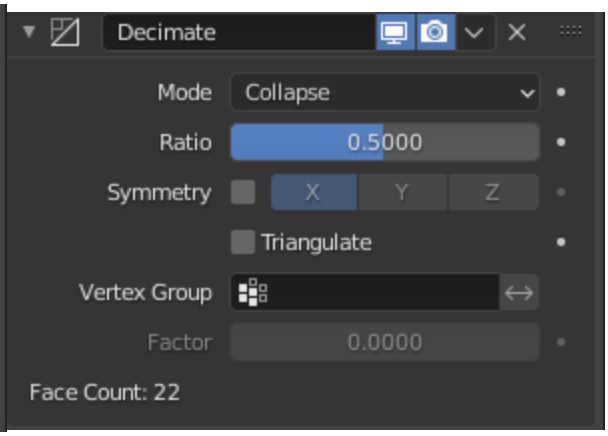


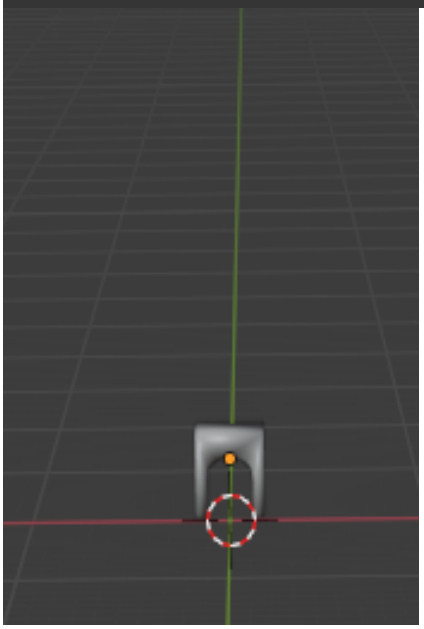
Antes das modificações serem iniciadas fomos buscar o modelo do arco que foi utilizado no nosso projeto do Blender.



De seguida, reduziu-se o número de faces do arco através do modifier Decimate, usando a opção Collapse as faces foram reduzidas para 22. Não se diminuiu mais o ratio de modo a não deformar o arco.







Por fim reduziu-se apenas o tamanho do arco.

No final o modelo foi convertido para o formato .Json, usando as opções corretas no Blender para gerar os ficheiros .obj e .mtl e o script obj_parser.py para gerar o .json pretendido, e o modelo foi colocado nos ficheiros do trabalho.

Exercício 4.2

Para este exercício foi seguido o guião que o professor publicou mas de qualquer modo vamos pôr aqui os passos feitos para carregar o arco no WebGL.

```
<!--Adicao scene e utils-->
<script src='Scene.js'></script>
<script src='Utils.js'></script>
```

Primeiramente foram incluídos os ficheiros Scene,js e Utils.js.

De seguida, criamos o gl fora do main, fazemos load da scene, um sleep de 5s para o vetor ficar preenchido e começamos a main.

```
var cubeRotation = 0.0;
const canvas = document.querySelector('#glcanvas');
const gl = canvas.getContext('webgl') || canvas.getContext('experimental-webgl');


var Scene1 = init_building(gl);
//setTimeout(console.log(Scene1.objects), 5000);

function sleep (time) {
  return new Promise((resolve) => setTimeout(resolve, time));
}
sleep(5000).then(() => {
  main();
});

//
// Start here
//
function main() {
```

Por fim usámos a função draw_json para carregar o arco, a qual é chamada na ao desenhar a cena(função drawScene()).

```
function init_building(gl){
  Scene.loadObjectByParts('./model/part','Arco',2);
    return Scene;
}
```

```
function draw_json(gl, programInfo) {

  try{

    for (var i = 0; i <  Scene1.objects.length; i++){
        var object = Scene1.objects[i];
        console.log(object);

        // Now set up the colors for the faces. We'll use solid colors
   // for each face.

   const faceColors = [
     [1.0,  1.0,  0.0,  1.0],
     [1.0,  1.0,  0.0,  1.0],
     [1.0,  1.0,  0.0,  1.0],
     [1.0,  1.0,  0.0,  1.0],
     [1.0,  1.0,  0.0,  1.0],
     [1.0,  1.0,  0.0,  1.0],
   ];

   // Convert the array of colors into a table for all the vertices.

   var colors = [];

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

     // Repeat each color four times for the four vertices of the face
     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);
```

```
        //Setting uniforms
        //gl.uniform4fv(prg.uMaterialDiffuse, object.diffuse);
        //gl.uniform1i(prg.uWireframe,object.wireframe);
        //gl.uniform1i(prg.uPerVertexColor, object.perVertexColor);

        //Setting attributes
        gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);
        gl.disableVertexAttribArray(programInfo.attribLocations.vertexColor);

        gl.bindBuffer(gl.ARRAY_BUFFER, object.vbo);
        gl.vertexAttribPointer(programInfo.attribLocations.vertexPosition, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);


         if (object.perVertexColor){
        gl.bindBuffer(gl.ARRAY_BUFFER, object.cbo);
        gl.vertexAttribPointer(programInfo.attribLocations.vertexColor,4,gl.FLOAT, false, 0,0);
        gl.enableVertexAttribArray(programInfo.attribLocations.vertexColor);
         }

        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, object.ibo);

        var modelMatrix = mat4.create();
        gl.uniformMatrix4fv(programInfo.uniformLocations.modelMatrix, false, modelMatrix);

        gl.drawElements(gl.TRIANGLES, object.indices.length, gl.UNSIGNED_SHORT,0);

        gl.bindBuffer(gl.ARRAY_BUFFER, null);
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);
        console.log("drawing");
    }
}
catch(err){
    alert(err);
    console.error(err.description);
}
console.log("done");
```

Como desde o início não conseguimos, nem percebemos que nos faltava computar luz nos objetos também não o fizemos neste exercício por não o sabermos como fazer mas para demonstrar o resultado o final mudou-se a cor do fundo do canvas para ser poder ver o arco.