Universidade Federal da Paraíba
Ciências da Computação
Introdução a Computação Gráfica

ATIVIDADE II

Alunos:
Luciano Pereira - 20190018530
Abraão Homualdo - 20200095558

Resumo

Como objetivo desta atividade, foi desenvolvido cálculos por embasamento através de matrizes com o intuito e finalidade de gerar uma renderização de um cubo formado por arestas e um objeto 3D escolhido pelo grupo

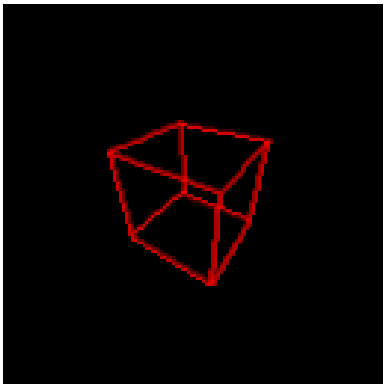


Figura 1: Resultado do Rendering da Atividade

Em princípio, foram alterados todos os valores das matrizes (Model, View, Viewport, Projeção) e com a participação da Homogeneização, como também as matrizes de transformação

No projeto em si, foi utilizado o framework Three.js, para criação das matrizes 4x4 como modelo descrito, e operações de álgebra linear. Abaixo seguem algumas das matrizes de transformação essenciais, presente no escopo da matriz de modelagem

```
function Matriz_escala(sx, sy, sz) { //T

    let m_escala = new THREE.Matrix4();

    m_escala.set(sx , 0.0, 0.0, 0.0,
                 0.0, sy , 0.0, 0.0,
                 0.0, 0.0, sz , 0.0,
                 0.0, 0.0, 0.0, 1.0);

    return m_escala;
}
```

Figura 2: Matriz da Transformação da escala do Cubo

```
function Matriz_shear(shx, shy, shz) { //Sh

    let m_shear = new THREE.Matrix4(); // M

    m_shear.set(1.0, shy, shz, 0.0,
                shx, 1.0, shz, 0.0,
                shx, shy, 1.0, 0.0,
                0.0, 0.0, 0.0, 1.0);

    return m_shear;
}
```

Figura 3: Matriz Shear

```
function Matriz_translacao(tx, ty, tz) { //T

    let m_translacao = new THREE.Matrix4();

    m_translacao.set(1.0, 0.0, 0.0,  tx,
                     0.0, 1.0, 0.0,  ty,
                     0.0, 0.0, 1.0,  tz,
                     0.0, 0.0, 0.0, 1.0);

    return m_translacao;
}
```

Figura 4: Matriz de Translação

```
function Matriz_rotacao(eixo, angulo) {

    let radiano = angulo * Math.PI / 180.0;

    let cos = Math.cos(radiano);
    let sen = Math.sin(radiano);

    let m_rotacao = new THREE.Matrix4();

    if (eixo == 'X' || eixo == 'x') {
        m_rotacao.set(1.0, 0.0, 0.0, 0.0,
                      0.0, cos,-sen, 0.0,
                      0.0, sen, cos, 0.0,
                      0.0, 0.0, 0.0, 1.0);

    } else if (eixo == 'Y' || eixo == 'y') {
        m_rotacao.set(cos, 0.0, sen, 0.0,
                      0.0, 1.0, 0.0, 0.0,
                     -sen, 0.0, cos, 0.0,
                      0.0, 0.0, 0.0, 1.0);

    } else if (eixo == 'Z' || eixo == 'z') {
        m_rotacao.set(cos,-sen, 0.0, 0.0,
                      sen, cos, 0.0, 0.0,
                      0.0, 0.0, 1.0, 0.0,
                      0.0, 0.0, 0.0, 1.0);
    }

    return m_rotacao;

}
```

Figura 5: Matriz de Rotação (com as devidas condições de estruturação)

.

```
/**************************************************************************
 * Parâmetros da camera sintética.
 **************************************************************************/
let cam_pos = new THREE.Vector3(1.3,1.7,2.0);     // posição da câmera no esp. do Universo.
let cam_look_at = new THREE.Vector3(0.0,0.0,0.0); // ponto para o qual a câmera aponta.
let cam_up = new THREE.Vector3(0.0,1.0,0.0);      // vetor Up da câmera.


/**************************************************************************
 * Matriz View (visualização): Esp. Universo --> Esp. Câmera
 * OBS: A matriz está carregada inicialmente com a identidade.
 **************************************************************************/

// Derivar os vetores da base da câmera a partir dos parâmetros informados acima.

let direcao = new THREE.Vector3();
let x_cam = new THREE.Vector3();
let y_cam = new THREE.Vector3();
let z_cam = new THREE.Vector3();

direcao.subVectors(cam_look_at, cam_pos);

//ZCAM = - D / |D|
z_cam = direcao.normalize();
z_cam = z_cam.negate();


// XCAM = (U X ZCAM) / |U X ZCAM|
x_cam = x_cam.crossVectors(cam_up, z_cam).normalize();
// YCAM = (ZCAM X XCAM) / |ZCAM X XCAM|
y_cam = y_cam.crossVectors(z_cam, x_cam).normalize();
```

Sobre o escopo da Matriz View, relacionada a câmera, foram criadas as variáveis relacionadas a direção, XCAM, YCAM e ZCAM, com suas formulas e explicações adquiridas das aulas apresentadas a turma, e em seguida montada a matriz que é a inversa da base da câmera, com os parâmetros relacionados as coordenadas de XCAM, YCAM e ZCAM e depois uma matriz de translação para coincidir as origens do espaço da câmera com o do universo, ambas as duas são multiplicadas e assim formando a Matriz View

```
// Construir 'm_bt', a inversa da matriz de base da câmera.

let m_bt = new THREE.Matrix4();

m_bt.set(x_cam.x, x_cam.y, x_cam.z, 0.0,
         y_cam.x, y_cam.y, y_cam.z, 0.0,
         z_cam.x, z_cam.y, z_cam.z, 0.0,
         0.0, 0.0, 0.0, 1.0);

// Construir a matriz 'm_t' de translação para tratar os casos em que as
// origens do espaço do universo e da câmera não coincidem.

let m_t = new THREE.Matrix4();

m_t.set(1.0, 0.0, 0.0, -cam_pos.x,
        0.0, 1.0, 0.0, -cam_pos.y,
        0.0, 0.0, 1.0, -cam_pos.z,
        0.0, 0.0, 0.0, 1.0);

// Constrói a matriz de visualização 'm_view' como o produto
//  de 'm_bt' e 'm_t'.
let m_view = m_bt.clone().multiply(m_t);

for (let i = 0; i < 8; ++i)
    vertices[i].applyMatrix4(m_view);
```

Em seguida vem a matriz de projeção e homogeneização. Que envolve a direção da câmera, o resultado da aplicação da matriz de projeção com a matriz resultante da página passada gera um valor diferente de 1 para a coordenada W, que precisa ser resolvida, sendo a solução a homogeneização cujo promove uma divisão de todos os parâmetros pelo valor de W

```
/*****************************************************************************
 * Matriz de Projecao: Esp. Câmera --> Esp. Recorte
 * OBS: A matriz está carregada inicialmente com a identidade.
 *****************************************************************************/


let m_projection = new THREE.Matrix4();
let d = 1.0

m_projection.set(1.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, d,
                 0.0, 0.0, -1/d, 1.0);

for (let i = 0; i < 8; ++i)
    vertices[i].applyMatrix4(m_projection);

/*****************************************************************************
 * Homogeneizacao (divisao por W): Esp. Recorte --> Esp. Canônico
 *****************************************************************************/

for (let i = 0; i < 8; ++i) {
    vertices[i].divideScalar(vertices[i].w);
}
```

Finalmente, vem a última sequência de matrizes, o escopo da matriz Viewport, que envolve os parâmetros da altura e largura, que na matriz de escala são divididos por 2, e uma matriz de translação que move X e Y em uma unidade, tais matrizes são multiplicadas e assim encerrando o pipeline gráfico, o que vem em seguida é a rasterização

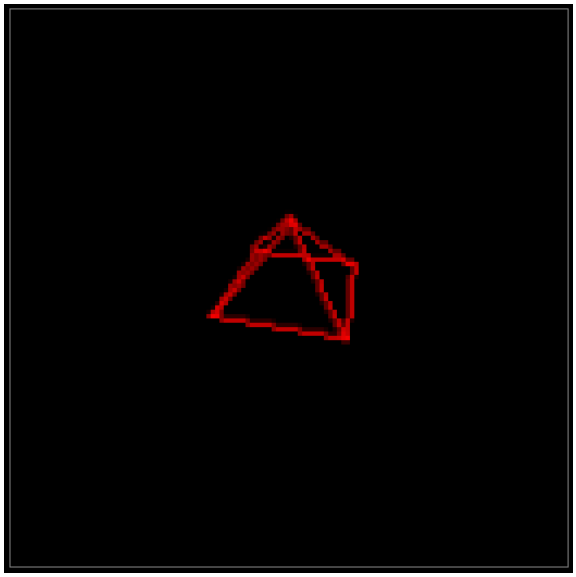
```
/*****************************************************************************
 * Matriz Viewport: Esp. Canônico --> Esp. Tela
 * OBS: A matriz está carregada inicialmente com a identidade.
 *****************************************************************************/

let w = 128; // largura
let h = 128; // altura

let escala = new THREE.Matrix4()
escala = Matriz_escala(w/ 2, h/ 2, 1.0); //escala a matriz proporcionalmente com a resolucao
let translacao = new THREE.Matrix4();
translacao = Matriz_translacao(1.0, 1.0, 0.0);

let m_viewport = new THREE.Matrix4();

m_viewport.set( 1.0, 0.0, 0.0, 0.0,
                0.0, 1.0, 0.0, 0.0,
                0.0, 0.0, 1.0, 0.0,
                0.0, 0.0, 0.0, 1.0);

// M.viewport = M.escala * M.translação
m_viewport = m_viewport.multiply(escala);
m_viewport = m_viewport.multiply(translacao);

for (let i = 0; i < 8; ++i)
    vertices[i].applyMatrix4(m_viewport);
```

A rasteirização usa o MidPointLineAlgorithm, com o codigo produzido na Atividade I

```
color = [255, 0, 0, 0]; // vermelho
for (let i = 0; i < 12; ++i) {
    MidPointLineAlgorithm(vertices[edges[i][1]].x, vertices[edges[i][1]].y, vertices[edges[i][0]].x, vertices[edges[i][0]].y, color, color);
}
```

Objeto feito pelo grupo

Foi feito uma pirâmide, que contem 5 vértices e 8 arestas



```
/**************************************************************************
 * Piramide
 **************************************************************************/
//                                 X     Y     Z    W (coord. homogênea)
    let vertices = [new THREE.Vector4( 1,  0,  0, 1.0),
                    new THREE.Vector4( 0,  0, -1, 1.0),
                    new THREE.Vector4(-1,  0,  0, 1.0),
                    new THREE.Vector4( 0,  0,  1, 1.0),
                    new THREE.Vector4( 0, 0.85, 0, 1.0),
                    new THREE.Vector4( 0,  0,  0, 1.0),
                    new THREE.Vector4( 0,  0,  0, 1.0),
                    new THREE.Vector4( 0,  0,  0, 1.0),

                ];

/**************************************************************************
 * As arestas da Piramide
 **************************************************************************/
    let edges = [[0,1],
                [2,1],
                [3,2],
                [0,3],

                [3,4],
                [0,4],
                [1,4],
                [2,4],
                ];
```

Dificuldades

A dificuldade que foi encontrada no projeto foi que, a elaboração das matrizes exigia uma certa noção de geometria condicional, onde a alteração dos valores em cadeia fazia com que o cubo ele não tomasse a forma desejada, pôr fim a renderização sofreu grandes mutações até chegar no seu devido resultado, pois as matrizes devem estar em conjunto numérico e complementar.

Bibliografia

Framework Three.Js, https://github.com/mrdoob/three.js/

Repositório da Atividade, https://github.com/luci18530/Atividade2-ICG