UFPB

Centro de Informática – João Pessoa
Disciplina: Computação Gráfica
Klismann de Oliveira Barros - 20200085284
Thaís Gabrielly Marques - 20180135293

ATIVIDADE PRÁTICA 2 - IMPLEMENTAÇÃO DO PIPELINE GRÁFICO

Descrição:

Na atividade foi pedido que implementasse-mo-nos os estágios geométricos decorrentes de um pipeline gráfico, teríamos que desenvolver as transformações geométricas, matrizes, lidar com o espaço homogêneo e por fim desenhar na tela utilizando o algoritmo da atividade anterior.

Estratégias adotadas:

Seguindo como determinado, não poderíamos utilizar bibliotecas externas e nem estender o uso da biblioteca Three.js, as matrizes exigidas deveriam ser implementadas de forma manual. Então, antes de concentrar no código, revimos as aulas e analisamos o pipeline gráfico completo.

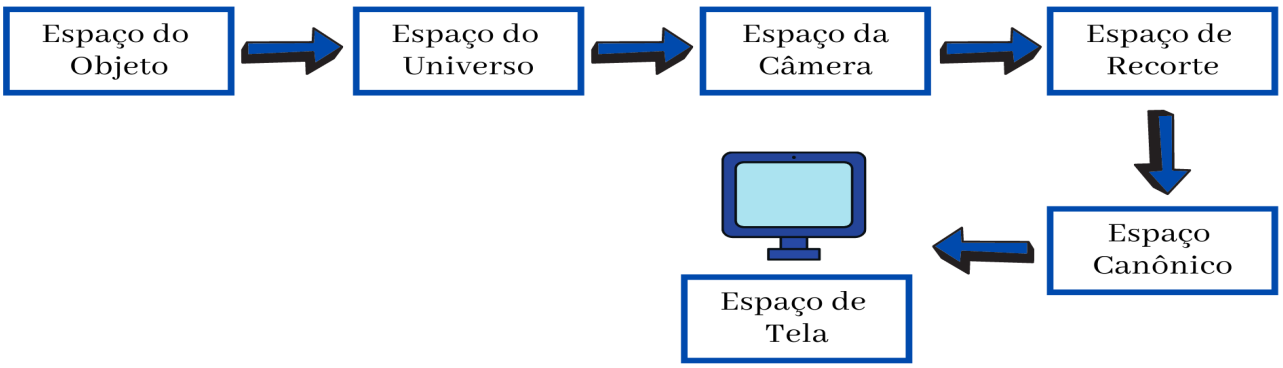

Imagem 01 - Percurso de um frame

Dividindo o pipeline em etapas:

1. Para ir do espaço do objeto para o espaço do universo é preciso implementar uma matriz de modelagem, nela é contido o número considerável de transformações geométricas: translações e rotações por exemplo

2. Sair do espaço universo para o espaço da câmera é preciso implementar uma matriz de visualização, é nela que passamos as informações de direção, posição e up.

3. Passando do espaço de câmera para o espaço de recorte é preciso da implementação da matriz de projeção, ela recebe as vértices vindas do espaço de câmera. Recebe parâmetros de distância e é com isso que se determina a distorção.

4. Não é possível ir direto do espaço de recorte para o espaço canônico, então é realizada uma homogeneização, que por ela transforma os pontos do espaço de recorte para o espaço seguinte. Essa etapa não precisa do uso de matriz, pois durante o processo todas as coordenadas do de um vetor do espaço de recorte são divididas por uma coordenada homogênea.

5. Na última etapa é preciso implementar a matriz Viewport, pois é por ela que podemos exibir no espaço de tela..

E além de implementar manualmente cada etapa, também necessita que modifique algumas passagens do algoritmo da atividade anterior para que se adequasse a essa atividade e pudesse rasterizar o cubo, no caso estamos chamando apenas a função **MidPointLineAlgorithm(),** dentro da parte de rasterização.

Resultados gerados:

Do Espaço Objeto para o Espaço do Universo:

```
let m_model = new THREE.Matrix4();

m_model.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);
```

Imagem 02 - Matriz Model (modelagem)

Do Espaço do Universo para o Espaço da Câmera:

```
const d_vector = new THREE.Vector3();
let Zcam = new THREE.Vector3();
let Xcam = new THREE.Vector3();
let Ycam = new THREE.Vector3();

d_vector.subVectors(cam_look_at, cam_pos);
Zcam = d_vector.normalize().clone().negate(); // - d/|d|
Xcam = Xcam.crossVectors(cam_up, Zcam).normalize();
Ycam = Ycam.crossVectors(Zcam, Xcam).normalize();
  // Construir 'm_bt', a inversa da matriz de base da câmera.

  let m_bt = new THREE.Matrix4();

  m_bt.set(Xcam.x, Xcam.y, Xcam.z, 0.0,
           Ycam.x, Ycam.y, Ycam.z, 0.0,
           Zcam.x, Zcam.y, Zcam.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);
```

Imagem 03 - Matriz View (visualização)

Do Espaço da Câmera para o Espaço de Recorte:

```
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, -1.0/D, 1.0);

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

Imagem 04 - Matriz de Projeção

Do Espaço de Recorte ou Projetivo para o Espaço Canônico:

Homogeneização:

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

Imagem 05 - Divisão por W

Do Espaço Canônico para o Espaço de Tela:

```
let m_viewport = new THREE.Matrix4();
let matrix_translation = new THREE.Matrix4();
let matrix_scale = new THREE.Matrix4();

matrix_scale.set( 64, 0.0, 0.0, 64,
                  0.0, 64, 0.0, 64,
                  0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 1.0);

matrix_translation.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.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 = matrix_scale.clone().multiply(matrix_translation);

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

Imagem 06 - Matriz Viewport

Resultado final (Rasterização):

A princípio não precisa implementar todo o código da atividade passada, basta anexar link da atividade 1 na aba js e chamar a função que é responsável pela rasterização, porém é preciso alterar algumas coisas na função MidPointLineAlgorithm para utilizar corretamente nessa atividade.

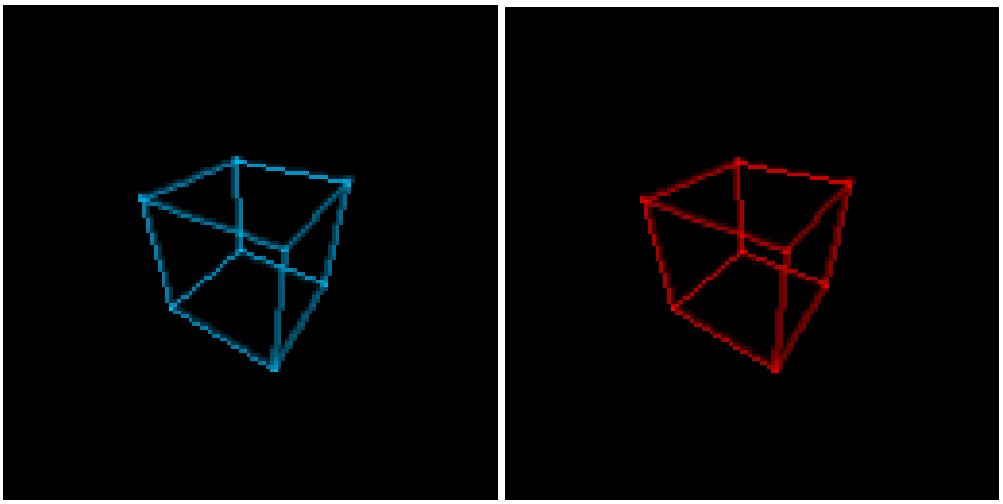

Imagem 07 - Resultado final

Dificuldades e possíveis melhorias:

Nossa maior dificuldade foi implementar as matrizes, fazer a comunicação das operações durante o código, entender a lógica e saber quais funções da Three.js poderíamos utilizar nas partes relacionadas a álgebra linear.

Além de ter dificultado bastante na parte da pesquisa por conta que muitas fontes utilizam recursos prontos da própria three.js ou de bibliotecas externas, o que seria dados não relevantes à nossa pesquisa, atrasando consideravelmente o desenvolvimento.

Melhorias futuras: testar novas abordagens, pesquisar em outras linguagens e tentar observar semelhanças e adaptar para o javascript, filtrar melhor os erros, fazer comparações com o que a própria Three.js oferece, além de tratar e desenhar outras figuras.

Referências bibliográficas:

- Notas e aulas do professor.
- https://sig-arq.ufpb.br/arquivos/20211290042888296533649223259a6ff/cga_ieee_vxx_ix_jim_blinns_corner_homogeneous_perspective_transform_xxxx.pdf
- https://sig-arq.ufpb.br/arquivos/20210081608f2e296533093aee967f72d/Chapter_7_Shirley.pdf

- https://www.canva.com/design/DAEdg8hcw8s/OCT_yxFOQPHg0tfQx1bQmQ/view?utm_content=DAEdg8hcw8s&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink (imagem, criação própria).

- https://celke.com.br/artigo/tabela-de-cores-html-nome-hexadecimal-rgb

- https://codepen.io/gabriellymarques02/pen/poevvpG (algoritmo da atividade anterior, com as pequenas modificações para suprir as necessidades dessa atividade).

Link repositório:

➢ https://codepen.io/Klismann_Barros/pen/LYWEVZz (cubo azul).

➢ https://codepen.io/gabriellymarques02/pen/QWdoGVP (cubo vermelho).