Introdução à Computação Gráfica

Emmanuella Faustino Albuquerque
20170002239

Atividade Prática 2: Implementação do Pipeline Gráfico

18 de outubro de 2021

VISÃO GERAL

Nesta atividade foram implementados todos os estágios do Pipeline Gráfico, na função ***executeGPipeline()***, por meio das matrizes de transformação: Matriz de Modelagem(Model), Matriz de Visualização(View), Matriz de Projeção(Projection), Matriz Viewport e a Homogeneização.

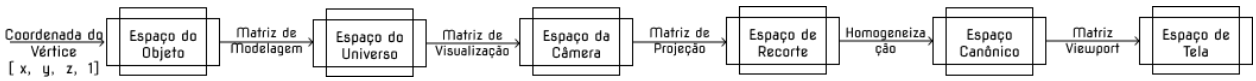
Além disso, foram adicionadas outras formas geométricas e uma função(i.e ***StartRotation()***), que rotaciona os modelos adicionados em torno do eixo y, com o intuito de melhorar a visualização das formas.

ESTRATÉGIAS

Primeiramente foi preciso derivar os vetores da base da câmera, Xcam, Ycam, e o Zcam, a partir do vetor de posição(**P**) da câmera, do **Look At** e do **U**p. Com eles, foi possível encontrar o vetor direção(**D**) e montar a matriz de visualização a partir das equações abaixo.

$$\vec{D} = \vec{LookAt} - \vec{P} \quad Z\vec{cam} = -\frac{\vec{D}}{|\vec{D}|} \quad X\vec{cam} = \frac{\vec{U} \times Z\vec{cam}}{|\vec{U} \times Z\vec{cam}|} \quad Y\vec{cam} = \frac{Z\vec{cam} \times X\vec{cam}}{|Z\vec{cam} \times X\vec{cam}|}$$

Estrutura do Pipeline



Matrizes utilizadas e Homogeneização

Para realizar as transformações geométricas do Pipeline Gráfico, foram utilizadas as matrizes listadas abaixo, além da matriz de modelagem carregada com a identidade. A homogeneização foi realizada pela divisão do vetor dos vértices no espaço de recorte pelo escalar W.

$$\vec{t} = \vec{P} - \vec{o}, \; onde$$
$$o = origem \; esp. \; universo$$

$$M_{projection} = M_p \times M_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & \frac{-1}{d} & 0 \end{bmatrix}$$

$$M_{view} = B^T \times T = \begin{bmatrix} Xcam(x) & Xcam(y) & Xcam(z) & 0 \\ Ycam(x) & Ycam(y) & Ycam(z) & 0 \\ Zcam(x) & Zcam(y) & Zcam(z) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{viewport} = S \times T = \begin{bmatrix} width/2 & 0 & 0 & 0 \\ 0 & height/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rasterização

Após construir as matrizes acima, e aplicá-las a cada um dos vértices do objeto, foi realizada a rasterização, na função ***Render()***, utilizando o algoritmo do ponto médio desenvolvido na atividade anterior. Com o propósito de organizar melhor o código, a função ***drawLine***(i.e o algoritmo do ponto médio) foi incluída dentro da classe Canvas disponibilizada pelo professor.

Para realizar a rasterização de cada linha, cada aresta do objeto indicado foi analisada, ligando cada vértice. Ao final, foi criada uma função ***renderWrapper()***, para facilitar a execução das transformações dos vértices em conjunto com a rasterização.

RESULTADOS

Cena Final: Cubo

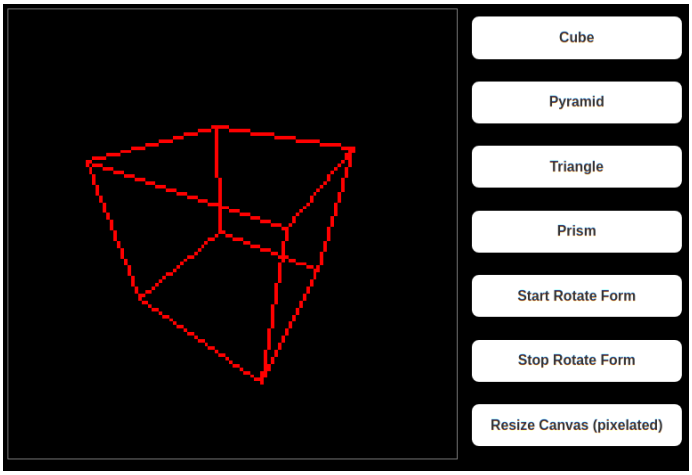


Imagem 1: Cubo Rasterizado

Cena Final: Outras Formas Geométricas

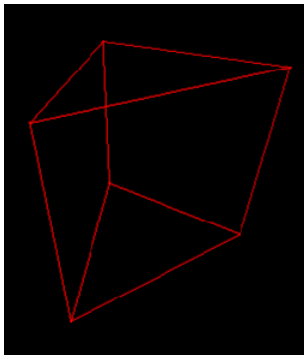


Imagem 2: Prisma com canvas redimensionado 512x512

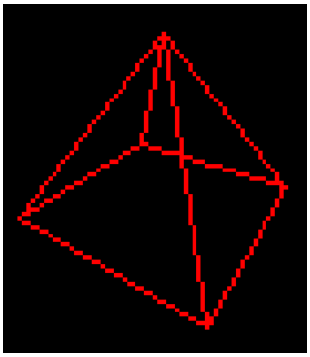


Imagem 3: Pirâmide com canvas original em 128x128

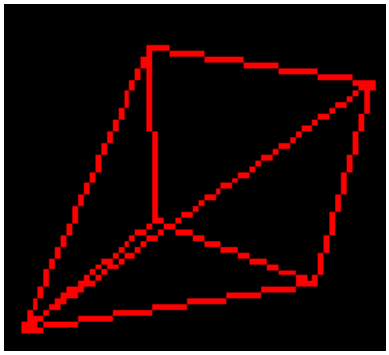


Imagem 4: Triângulo com canvas original em 128x128

Obs: a mudança na visualização da Imagem 2, ocorre pelo fato que o canvas estava com as mesmas dimensões do elemento canvas HTML exibido/criado no browser. [ref 5]

Dificuldades

A maior dificuldade foi conseguir encontrar formas geométricas descritas com vértices e arestas, já que normalmente os modelos .obj são descritos com vértices e faces.

Possíveis melhorias

Seria interessante reconstruir a matriz de projeção com o view frustum igual ao utilizado pelo opengl, no qual no espaço canônico as coordenadas vão de -1 a 1.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] OpenGL Transformation. Song Ho Ahn. Disponível em: http://www.songho.ca/opengl/gl_transform.html. Acesso em: 15 de outubro de 2021.

[2] Vector4. Threejs. Disponível em: https://threejs.org/docs/#api/en/math/Vector4. Acesso em: 15 de outubro de 2021.

[3] Prism Reference. HWS. Disponível em: https://math.hws.edu/eck/cs424/f15/lab7/prism.png. Acesso em: 15 de outubro de 2021.

[4] Timeouts e intervalos. MDN Web Docs. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Timeouts_and_intervals. Acesso em: 15 de outubro de 2021.

[5] Canvas width and height in HTML5. Stackoverflow. Disponível em: https://stackoverflow.com/questions/4938346/canvas-width-and-height-in-html5. Acesso em: 15 de outubro de 2021.