Universidade Federal do Mato Grosso do Sul
Computação Gráfica - 2023 - 2
Prof. Jonathan de Andrade Silva
Pedro Paulo Araújo de Paula e Silva

Relatório Trabalho

O seguinte relatório abordará a implementação do trabalho respectivo, cujo objetivo foi aplicar conceitos da computação gráfica utilizando da biblioteca *OpenGL*. Dentro deste objetivo principal, 3 objetivos específicos foram alcançados, sendo eles: 1. Criação de um objeto ou sólido geométrico no espaço vetorial $\mathbb{R}^3$, 2. Realizar operações de transformação e 3. Realizar a projeção ortogonal a um observador. Para a abordagem do relatório corrente, nos baseamos nestes três objetivos.

1. Sólidos geométricos:

No atual trabalho, foram criados três sólidos geométricos, o Cubo, Pirâmide Regular e Octógono. Para o desenvolvimento dos três foram criadas três classes com os respectivos nomes que herdam da classe *Primitives*. Para cada sólido geométrico foi criado um buffer model *(modelVBO)* que contém três Buffers, o *positionVBO* (vértices), *ColorVBO* (cores) e *EBO* (índices), e portanto com apenas um buffer eu consigo lidar com aquele objeto respectivo e aplicar as transformações necessárias.

Todas as formas foram escritas a partir da construção de triângulos, primeiramente eu declaro quais são os vértices no vetor de vértices que será carregado para dentro da GPU no buffer de vértices, posteriormente, declaramos as cores de cada vértice, sendo assim o OpenGL faz a interpolação vértice a vértice para cada face, que é especificada no vetor de de índices e através do stride na função *glVertexAttribPointer.*

Os shaders Main.vert e Main.frag são utilizados para a coloração e posicionamento de cada vértice, sendo assim o Main.vert recebe coordenadas do vértice e um vetor de cores que é passado para o .frag que completa colorindo os vértices, sendo assim com apenas um .vert posso definir o padrão que todos os vértices irão seguir no posicionamento e como as cores se comportam, posteriormente isso fará mais sentido quando forem realizadas a aplicação de texturas e efeitos de iluminação. Para o projeção ortogonal o shader Minor.vert foi utilizado.

2. Transformações

Neste caso as transformações aplicadas foram a de translação (model -> world) utilizando um valor aleatório para a posição, e rotação tanto no eixo x como no eixo y. Para dar o efeito de animação foi utilizada uma variável dt para variar a matriz de transformação ao longo do tempo.

Primeiramente uma matriz de translação é inicializada e com os valores randômicos, posteriormente uma matriz de rotação é inicializada, depois a multiplicação dessas matrizes é efetuada, e por fim essa matriz de transformação é carregada para dentro da GPU, onde as multiplicações são feitas ao longo do tempo.

3. Projeção Ortogonal

Para a projeção ortogonal foram utilizadas duas classes, a Câmera e a Terreno, a classe terreno serve apenas para dar um senso de direção ao espectador , que a cada unidade acrescenta um ponto e realiza a criação de um polígono, entretanto somente as arestas são impressas, isso é configurado no Minor.vert .

Já a classe Câmera utiliza o shader Minor.vert, e uns recursos de visualização de openGL, como a matriz de projeção `perspective` que recebe como parâmetro o ângulo de abertura da câmera, o aspecto da janela (width/height), o tamanho do plano de corte próximo e o plano de corte distante.

Essa matriz de projeção é alterada através de multiplicações com entradas do mouse e teclado (W,A,S,D) e multiplicado pelo mundo dentro shader Minor.vert.

Todos os códigos estão aqui.

Agradecimentos:

Visando a transparência, gostaria de agradecer ao canal do Sérgio Silva pela playlist de 31 vídeos que explicam detalhadamente os conceitos, como shaders, bind buffers, câmeras, etc. Boa parte do código deste trabalho foi baseado no material disponibilizado pelo Sérgio em seu canal no YouTube.