Animação de objetos 3D

Alexandre Diniz de Souza, Giovane H. Iwamoto, Leandro José G. Pereira

Universidade Federal de Mato Grosso do Sul (UFMS) - FACOM

1. Objetos

Os objetos que escolhemos foram um **cubo**, uma **pirâmide** e um **prisma hexagonal** devido a facilidade de compor esses poliedros por triângulos. No nosso programa definimos e calculamos manualmente onde posicionar cada vértice e a partir disso utilizamos um vetor de índices para determinar quais vértices compõem cada triângulo para formar os objetos desejados.

Para cada um dos objetos criamos um *VBO*, *VAO* e *EBO* exclusivo para ele para melhor manipulação dos objetos.

2. Animação

Nossa animação foi realizada utilizando a rotação dos objetos criados em torno do eixo Y conforme solicitado. Para realizar tal rotação utilizamos o GLM para transladar o objeto para a origem, depois rotacionar e por último retornar o objeto para o local que ele se encontrava. Através do GLM será gerado uma matriz a qual é repassada para o shader de vértices para que seja realizada a rotação.

3. Projeção

A projeção que escolhemos foi a projeção por perspectiva na qual os objetos que estão mais distantes apresentam aspecto menor quando executamos o programa e os objetos mais próximos da câmera são posicionados mais à frente e têm dimensões maiores.

4. Código

Utilizamos o template disponibilizado pelo *Replit* recomendado pelo professor. Seguindo essa estrutura básica, nossos shaders - tanto de vértices quanto de fragmento - estão localizados na pasta *res / shaders.*

Todo o código principal que desenvolvemos está no arquivo *main.c* localizado na pasta *src*. Neste arquivo importamos as principais bibliotecas do openGL e as estruturas básicas fornecidas pelo *Replit*.

Dentro da nossa função *main* começamos inicializando uma instância de shader para cada objeto da nossa cena. Depois, descrevemos os vértices de cada face dos objetos que escolhemos desenhar na cena (conforme indicado/comentado no código-fonte).

Esses vértices são guardados nos vetores *verticesCubo*, *verticesPiramide* e *verticesPrisma*. Após descrever cada vértice dos objetos, utilizamos os vetores de

índices - *indicesCubo*, *indicesPiramide* e *indicesPrisma* para formar os triângulos que compõem cada face dos objetos.

Dado que descrevemos os vértices e os triângulos formados, criamos os *VBOs*, *VAOs* e *EBOs* de cada objeto da cena e indicamos que os 3 primeiros valores dos vetores de coordenadas referenciam as coordenadas dos vértices dos objetos e que os 3 valores restantes referenciam os valores das cores em RGB de cada vértice.

Após essa etapa, indicamos o uso dos shaders de cada objeto através da função *useShader()* disponibilizada pelo template do *Replit*.

Tendo definido os vértices que compõe cada objeto, as cores de cada vértice/face, como formar cada face através de triângulos e o uso dos shaders, partimos para o loop o qual tem a função de ficar desenhando na tela os objetos que desejamos até que a janela seja fechada.

Dentro do loop, para cada objeto da cena definimos a projeção de perspectiva por meio das funções *glm_translate* e *glm_perspective*, depois criamos a matriz de rotação que será responsável pela animação em torno do eixo Y do objeto - primeiramente transladando o objeto para a origem, depois rotacionando o objeto no eixo Y e por fim, transladando o objeto novamente para sua posição na cena - e por fim passamos essas matrizes responsáveis pela projeção perspectiva e animação por rotação para os seus respectivos shaders através da função *setShaderMat4()*.

Para demonstrar que nossos objetos são independentes, atribuímos diferentes velocidades de rotação para cada objeto da cena. Sendo assim, o cubo rotaciona a ¼ da velocidade do Prisma e a pirâmide rotaciona a ¾ da velocidade de rotação do Prisma.

Dado que precisávamos passar matrizes de rotações distintas para cada objeto devido a velocidade de rotação diferentes, foi necessário a criação de instâncias de shaders distintos para cada objeto. Ou seja, embora os shaders tenham o mesmo código eles recebem matrizes distintas para cada objeto.

Por fim, após o encerramento do loop, temos a liberação de memória alocada para cada um dos buffers (*VBO*, *VAO*, *EBO*) de cada objeto visto que cada um tem seus próprios buffers.

5. Execução

O código fonte enviado contém um Makefile que pode ser utilizado para gerar o executável do nosso trabalho.

1. Na pasta raiz execute o comando *make*
2. O arquivo executável está localizado na pasta *bin*, portanto para executar o programa basta digitar *bin/main* ainda na pasta raiz do projeto.

OBS.: Caso queira compilar e executar o programa em um única comando use *make && bin/main*