Link para o repositório do GitHub com o código fonte:
https://github.com/VictorC98/abcg/tree/main/AtividadesPraticas/Atividade3
Link para o Github Pages com a aplicação:
https://victorc98.github.io/abcg/ShootingRange2.0/

Técnicas utilizadas

A aplicação consiste em um campo de tiro com duas fileiras de alvos e um alvo pequeno e acima dos outros, todos usam um modelo obtido gratuitamente no site free 3d.com. incluindo o arquivo .mtl e .png usados na texturização. No espaço da aplicação também estão presentes um grande retângulo para o chão e um para a parede atrás dos alvos.

O usuário tem controle sobre a câmera e sobre cada grupo de alvos, as instruções de controle estão no canto inferior esquerdo da tela, usando a fonte Raleway. Foi utilizado a estrutura SDL_Event da SDL, divididos em pressionamento de teclas (SDL_KEYDOWN) e liberação de teclas (SDL_KEYUP), para controlar os alvos, decidindo quais alvos devem ficar orientados para cima, com rotação 0 e quais devem ficar “deitados”, com rotação 90°. O alvo pequeno pode ser movimentado para a esquerda e para a direita (eixo x), desaparecendo e reaparecendo do lado oposto caso o alvo se desloque demais para um lado.

Os controles estão no canto inferior esquerdo da tela. O usuário pode se movimentar com wasd ou setas, mudar a posição dos alvos com 1,2,3 e 4, e mudar a posição do alvo pequeno com q,e. Com o mouse, o usuário pode mover a câmera e usar zoom com o botão direito.

Foi utilizado o SDL_MOUSEMOTION para capturar os movimentos relativos do mouse, junto com o SDL_SetRelativeMouseMode(SDL_TRUE) para prender o mouse no centro da tela e escondê-lo, assim permitindo o controle da câmera através do mouse. Como isso dificulta fechar a aplicação, foi adicionado o botão Escape para liberar o mouse. Também foi usado o SDL_MOUSEBUTTONDOWN para aplicar um zoom na câmera e o SDL_MOUSEBUTTONUP para cancelar o efeito. O efeito de zoom foi feito alterando o Field of View da câmera.

A câmera usada é uma câmera LookAt que utiliza as seguintes funções para movimentação:

- dolly: faz a câmera ir para a frente e para trás ao longo da direção de visão .

- truck: faz a câmera deslizar para os lados.
- pan: faz a câmera girar em torno de seu eixo y, permitindo olhar para os lados.
- rotatex: faz a câmera girar em torno de seu eixo x, permitindo olhar para cima e para baixo.

A função dolly computa o vetor “para frente” subtraindo a posição da câmera do ponto LookAt. Depois muda a posição da câmera e do ponto LookAt multiplicando o vetor “para frente” por uma velocidade predeterminada, baseada em deltaTime.

A função truck computa o vetor para a esquerda fazendo o produto vetorial dos vetores para frente e para cima, depois muda a posição da câmera e do ponto LookAt da mesma maneira que a função dolly.

A função pan faz o movimento de girar a câmera em torno de seu eixo y. Isso é feito alterando apenas o ponto LookAt por meio de uma série de transformações de matrizes.

A função rotatex faz o movimento de girar a câmera em torno de seu eixo x. Ela funciona da mesma forma que a função pan, mas ela usa o vetor de direção para direita (m_right) na transformação de rotação, enquanto a pan usa o vetor para cima. Isso faz com que, ao olhar para trás, a câmera inverta os controles e mexer o mouse para cima faz a câmera olhar para baixo, e vice-versa. Para consertar esse problema, a câmera inverte a velocidade de rotação se o usuário olhar para trás.

Depois de cada função da câmera, a matriz de visão é recalculada através da função computeViewMatrix.

Os alvos utilizam o identificador GL_TRIANGLES na função de renderização glDrawArrays para criar o conjunto de triângulos que compõem cada modelo, enquanto o chão e a parede utilizam o identificador GL_TRIANGLE_STRIP para formar uma polilinha. Todos os modelos usam geometria indexada, ou seja, usam um arranjo ordenado de posições de vértices armazenado no VBO, que são atrelados a um VAO, e um arranjo de números inteiros que representam os índices para esses vértices armazenados no EBO.

A aplicação usa uma textura para modificar as propriedades de reflexão difusa do modelo de reflexão de Blinn–Phong. O arquivo lookat.vert é o vertex shader, que calcula os vetores v, n e l e os envia ao fragment shader. O vetor v é o vetor superfície-câmera, o vetor l é o vetor superfície-fonte de luz e o vetor n é o

vetor normal unitário correspondente à superfície. O arquivo lookat.frag é o fragment shader, que utiliza o amostrador de textura 2D “sampler2D” para acessar os texture elements da textura difusa. Esse fragment shader suporta múltiplos modos de mapeamento, mas é apenas utilizado o mapeamento a partir das coordenadas de textura contidas no vetor fragTexCoord. Se o fragmento pertence a um triângulo visto de frente, a cor de saída é a própria cor de entrada. Se não, a cor de saída é o tom de cinza usado no chão e na parede, que são vistos de trás.

Os modelos são texturizados a partir de uma imagem PNG, usando um arquivo .mtl que contém a descrição das propriedades dos materiais do modelo. A renderização é feita por meio da função “render”, que ativa a primeira “unidade de textura” (GL_TEXTURE0) e liga o identificar de textura à unidade recém ativada. Por fim, a função glTexParameteri é chamada para configurar os filtros de textura e modos de empacotamento. Os valores utilizados são os valores padrão do OpenGL.

Referências

Modelo usado: https://free3d.com/3d-model/shooting-range-targhet-22266.html