UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
SCC0250 COMPUTAÇÃO GRÁFICA - PROFª MARIA CRISTINA F. DE OLIVEIRA

TRABALHO 1

TRANSFORMAÇÕES 2D EM OPENGL

Relatório

CÉSAR AUGUSTO LIMA……..9771525
EDUARDO ZABOTO MIROLLI……....9778501
GABRIEL ROMUALDO SILVEIRA PUPO……..9896250

SÃO CARLOS
2019

Introdução

O primeiro trabalho da disciplina consiste em apresentar uma cena com um catavento, que sofre duas possíveis transformações: rotação e translação. Ambas as operações são controladas pelo usuário, através de algumas teclas do teclado. Vale citar que o código para tal objetivo foi implementado em C++ e teve como base o código utilizado durante as aulas práticas, disponível em: https://github.com/rnakanishi/icmc-cg.git. Por fim, junto ao código fonte do trabalho encontra-se este relatório e um arquivo README descrevendo passo-a-passo como compilá-lo e executá-lo em Linux.

Desenvolvimento

Implementação do problema descrito

Para a instanciação do catavento na cena, criou-se um vetor de vértices que descrevem todos os pontos de cada triângulo que compõe o catavento. Com este centrado no ponto (0, 0), as coordenadas de cada triângulo são:

- Triângulo inferior: (0, -0.5, 0); (0.3, -0.5, 0); (0, 0, 0)
- Triângulo direito: (0.5, 0, 0); (0.5, 0.3, 0); (0, 0, 0)
- Triângulo superior: (-0.3, 0.5, 0); (0, 0.5, 0); (0, 0, 0)
- Triângulo esquerdo: (-0.5, 0, 0); (-0.5, -0.3, 0); (0, 0, 0)

A fim de processar a transformação do catavento na placa gráfica, extraiu-se a variável uniforme `transform` contida no *vertex shader* para preenchê-la depois com a matriz de transformação, que será calculada no decorrer do programa. Essa matriz tem dimensões 4x4 pois a cena está no espaço R^3, incluindo, então, as três coordenadas *x*, *y*, *z* mais a homogênea *h*.

Em seguida, o programa entra no laço principal de execução, em que cada iteração é processada a cada *frame*. Primeiro, captura-se as entradas do teclado do usuário para saber qual operação deve ser feita no *frame* atual. Para transladar o catavento, seguindo uma taxa de translação pré-definida no arquivo de cabeçalho, os comandos são:

- W: mover para cima (soma a taxa à coordenada *y*);
- A: mover para esquerda (subtrai a taxa da coordenada *x*);
- S: mover para baixo (subtrai a taxa da coordenada *y*);
- D: mover para direita (soma a taxa à coordenada *x*).

Para rotacioná-lo, seguindo uma taxa de rotação pré-definida no mesmo arquivo de cabeçalho, os comandos são:

- Q: diminui a velocidade de rotação (subtrai a taxa da velocidade atual);
- E: aumenta a velocidade de rotação (soma a taxa à velocidade atual).

Para parar completamente a rotação do catavento, o usuário deve pressionar a tecla Espaço. Com o catavento parado, ele não consegue alterar a velocidade da rotação. Para fechar a janela, deve pressionar Esc.

Cria-se a matriz de translação do *frame* substituindo as coordenadas *x* e *y*:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A matriz de rotação do *frame*, por sua vez, é criada da seguinte forma:

$$\begin{bmatrix} \cos(v) & \text{sen}(v) & 0 & 0 \\ -\text{sen}(v) & \cos(v) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

em que *v* é a velocidade de rotação no *frame* atual. A matriz acima descreve um movimento de rotação em sentido horário quando $v > 0$, e um em sentido anti-horário quando $v < 0$.

Para aplicar as transformações no catavento, atualiza-se na função `glUniformMatrix4fv` a variável uniforme com a transposta do produto da matriz de rotação com a de translação, resultando em:

$$\begin{bmatrix} \cos(v) & -\text{sen}(v) & 0 & x \\ \text{sen}(v) & \cos(v) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A partir daí, o *vertex shader* aplica as transformações multiplicando a uniforme com as coordenadas do catavento, P' = (T * R) * P.

Por fim, os triângulos do catavento são renderizados na cena, com as modificações impostas pelo usuário, com a função `glDrawArrays`.

Descrição técnica do programa implementado

Para renderizar o catavento na cena, foram usadas algumas funções da OpenGL para criar um `shaderProgram`, uma parte do pipeline de renderização responsável por conectar as entradas e as saídas do *vertex shader* e do *fragment shader*, ambos descritos em linguagem GLSL nas strings `vertexShaderSource` e `fragmentShaderSource`. Esse processo é inteiramente realizado na função `createRenderingPipeline()`.

Após a criação desse programa, os vértices de cada triângulo que formava o catavento são passados ao vetor `vertices[]` anteriormente descrito. Usando esse vetor, as informações do objeto são passadas ao *Vertex Buffer Object* (VBO) e ao *Vertex Array Object* (VAO), ambos essenciais para sua renderização. Ao final, usa-se a função `glDrawArrays()` no laço principal de execução para desenhar o catavento de fato. Vale lembrar que esse somente procedimento ainda não aplica as transformações.

O procedimento da transformação em si, inicia-se na implementação em GLSL do vertexShader, onde se cria a variável uniforme `transform`, utilizada para o cálculo da variável interna `gl_position`. Em seguida, na função `run()` quando ocorre a criação do pipeline de renderização, feita pela função `createRenderingPipeline()` o objeto fica apto a sofrer as transformações propriamente ditas e por isso é necessário armazenar a localização da variável uniforme em uma variável `shaderTransform`, por meio da função `glGetUniformLocation()`, para usá-la logo depois no loop principal de execução.

Finalmente as transformações são calculadas através da multiplicação de `rotationMatrix` por `translationMatrix` e os resultados são passados para a variável uniforme `shaderTransform`, através da função `glUniformMatrix4fv()`, mencionada no tópico anterior. Após fazer isso a transformação estará finalizada e o objeto pode ser desenhado na tela usando `glDrawArrays()`. Depois, a iteração recomeça realizando o procedimento acima novamente, o que faz com que o catavento fique girando na cena e se mova para os lados de acordo com as necessidades do usuário, até que ele pressione a tecla ESC para sair do loop e encerrar a execução do programa.

Contribuições

- César: Implementação das transformações
- Eduardo: Relatório, desenho do catavento
- Gabriel: Relatório, revisão de código