Programowania w OpenGL z użyciem shader'ów, Transformacje, Kamera

1. Cel ćwiczenia:

Zapoznanie z programowaniem grafiki przy użyciu shader'ów, użycie transformacji układu współrzędnych, obsługa kamery.

2. Wstęp:

OpenGL sam nie obsługuje pojęcia kamery, ale można ją zasymulować, przesuwając wszystkie obiekty na scenie w przeciwnym kierunku. W tym celu należy skonfigurować kamerę np. typu FPS, która pozwala swobodnie poruszać się na scenie 3D. Konieczne jest również, wprowadzenie klawiatury.

3. Widok 3D

W celu uzyskania ostatecznego położenia punktów konieczne jest przekształcenie ich współrzędnych z uwzględnieniem macierzy modelu, widoku i projekcji, co w shaderze może wyglądać tak:

Deklaracja zmiennych:

uniform mat4 model;

uniform mat4 view;

uniform mat4 proj;

Określenie położenia punktów:

*gl_Position = proj * view * model * vec4(position, 1.0);*

4. Macierz modelu

Dostępna jest biblioteka pomocnicza GLM udostępniająca funkcje ułatwiające pracę z macierzami. Biblioteka jest wygodna w użyciu, wystarczy dołączyć do projektu folder z plikami nagłówkowymi. Najnowszą wersję można pobrać ze strony ***https://glm.g-truc.net*** (https://glm.g-truc.net/0.9.9/index.html)

Po rozpakowaniu biblioteki do katalogu głównego projektu, we właściwościach projektu *C/C++ -> Dodatkowe katalogi plików nagłówkowych ->edytuj dodać ścieżkę względną:* ***..\glm***.

(podobnie jak na 1 zajęciach !! – podobnie a nie tak samo !!)

W projekcie dodać kod:

#include <glm/glm.hpp>

#include <glm/gtc/matrix_transform.hpp>

#include <glm/gtc/type_ptr.hpp>

Aby rozpocząć rysowanie w 3D, należy stworzyć macierz modelu. Macierz modelu może składać się z przekształcenia translacji, skalowania i / lub rotacji, które mają zostać wykonane w celu przekształcenia wszystkich wierzchołków obiektu w globalną przestrzeń świata. Macierz taka może wyglądać następująco:

glm::mat4 model = glm::mat4(1.0f);

model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));

Wysłanie do shadera:

GLint uniTrans = glGetUniformLocation(shaderProgram, "model");

glUniformMatrix4fv(uniTrans, 1, GL_FALSE, glm::value_ptr(model));

5. Kamera / Widok (Macierz widoku)

Kiedy mówimy o przestrzeni kamery/widoku, mówimy o wszystkich współrzędnych wierzchołków, widzianych z perspektywy kamery jako punktu początkowego sceny. Macierz widoku przekształca wszystkie współrzędne świata na współrzędne widoku, które są zależne od położenia kamery i kierunku. Aby zdefiniować kamerę, trzeba znać jej pozycję w przestrzeni świata, kierunek w którym patrzy i wektor skierowany w górę.

GLM umożliwia tworzenie np.; macierzy **LookAt**, którą można wykorzystać jako macierz widoku.

glm::mat4 view;

view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f),

glm::vec3(0.0f, 0.0f, 0.0f),

glm::vec3(0.0f, 1.0f, 0.0f));

Aby kamera mogła się poruszać trzeba zdefiniować zmienne ją opisujące:

glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);

glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

I funkcję tworzącą macierz widoku można zdefiniować tak:

view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

Dla przyciśniętego klawisza w górę zmienną opisującą położenie kamery można aktualizować tak:

*cameraPos += cameraSpeed * cameraFront;*

A dla klawisza obrotu, tak:

obrot -= cameraSpeed;

cameraFront.x = sin(obrot);

cameraFront.z = -cos(obrot);

Ostatecznie należy wysłać macierz do karty graficznej aby uwzględnić ją w shaderach.

GLint uniView = glGetUniformLocation(shaderProgram, "view");

glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));

6. Macierz projekcji

Macierz projekcji można zdefiniować następująco:

glm::mat4 proj = glm::perspective(glm::radians(45.0f), 800.0f / 800.0f, 0.06f, 100.0f);

GLint uniProj = glGetUniformLocation(shaderProgram, "proj");

glUniformMatrix4fv(uniProj, 1, GL_FALSE, glm::value_ptr(proj));

7. Ćwiczenie:

Należy przygotować projekt uwzględniający macierze projekcji, widoku i modelu oraz umożliwiający poruszanie się po scenie w widoku FPS. Na scenie proszę umieścić sześcian oraz włączyć z-bufor. Przykładowa tablica wierzchołków dla sześcianu:

float vertices[] = {

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f

};