ЛАБОРАТОРНАЯ РАБОТА №5

Ориентация в пространстве и матрицы. Заготовка сцены.

Цель работы: сделать заготовку для сцены и настройки камеры.

Задание: написать код для создания шахматной площадки, осей координат и перемещения в пространстве над сценной.

Ориентация в пространстве и матрицы.

Мировая, видовая и проекционная матрицы - это удобный инструмент для разделения трансформаций.

Мировая матрица

Стандартный треугольник в базовом примере задается множеством вершин, координаты которых заданы относительно центра объекта, т. е. вершина с координатами (0, 0, 0) будет находиться в центре объекта.

Далее для перемещения модели, все, что мы делаем - это применяем масштабирование, поворот или перенос. Эти действия выполняются для каждой вершины, в каждом кадре и тем самым наша модель перемещается на экране.

Теперь наши вершины в мировом пространстве. *Мы перешли из пространства объекта (все вершины заданы относительно центра объекта) к мировому пространству (все вершины заданы относительно центра мира).*

Схематично это можно показать так:

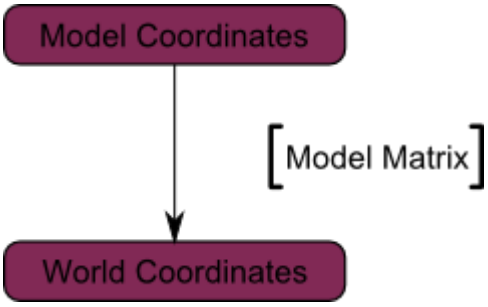

Рис. 1 Схема пересчета координат

Видовая матрица

«Движок не перемещает корабль. Корабль остается на том же месте, а движок перемещает вселенную вокруг него.»

Попробуйте представить это применительно к камере. Например, если вы хотите сфотографировать гору, то вы не перемещаете камеру, а перемещаете гору. Это не возможно в реальной жизни, но это невероятно просто в компьютерной графике.

Итак, изначально ваша камера находится в центре мировой системы координат. Чтобы переместить мир вам необходимо ввести еще одну матрицу. Допустим, что вы хотите переместить камеру на 3 юнита ВПРАВО

(+X), что будет эквивалентом перемещения всего мира на 3 юнита ВЛЕВО (-X).

А вот и продолжение схемы:

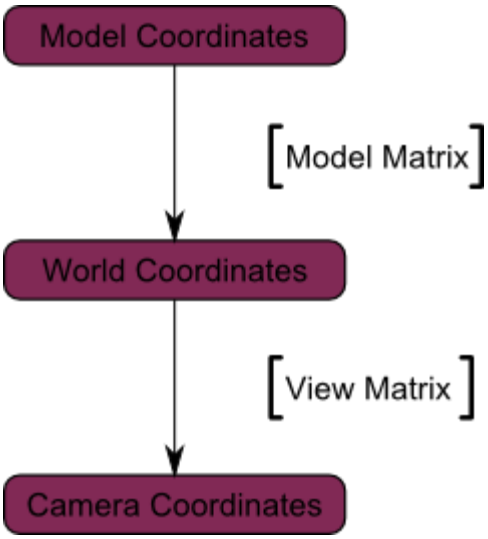

Рис. 2 Схема пересчета координат

Проекционная матрица

Итак, теперь мы находимся в пространстве камеры. Это означает, что вершина, которая получит координаты x == 0 и y == 0 будет отображаться по центру экрана. Однако, при отображении объекта огромную роль играет также дистанция до камеры (z). Для двух вершин, с одинаковыми x и y, вершина имеющая большее значение по z будет отображаться ближе, чем другая.

Это называется перспективной проекцией:

Мы перешли из Пространства Камеры (все вершины заданы относительно камеры) в Однородное пространство (все вершины находятся в небольшом кубе. Все, что находится внутри куба - выводится на экран).

Схема:

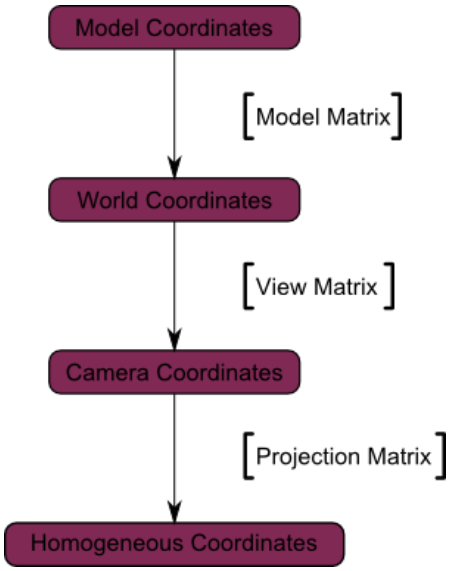

Рис. 3 Схема пересчета координат

Сейчас есть glFrustum — умножающая текущую матрицу на перспективную матрицу. В ней последовательно указываются координаты для левой и правой вертикальных плоскостей отсечения, для нижней и верхней горизонтальных плоскостей отсечения и в конце расстояния до плоскостей отсечения ближней и дальней глубины. Для плоскостей отсечения расстояния должны быть положительными.

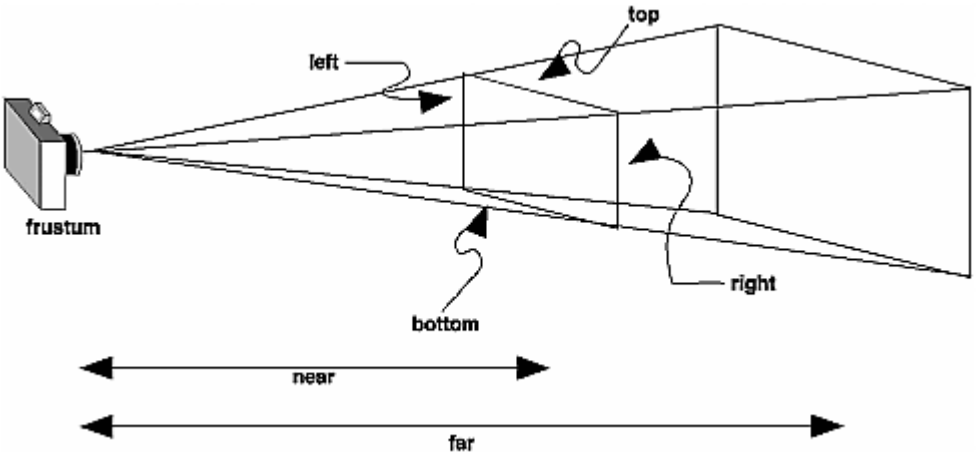

Рис. 4 Работа перспективной проекции glFrustum

Обратите внимание, что плоскость ближнего отсечения – это то что будет выводится на экран. Никаких обрезок объектов на расстояние near, производится не будет.

Использовать glFrustrum нужно будет один раз перед основным циклом:

```
void WndResize(int x, int y){
glViewport(0,0,x,y); //перестраивает размеры окна
float k=x/(float)y; //соотношение сторон
float sz = 0.1; //единица размера
glLoadIdentity(); //загрузка единичной матрицы
glFrustum(-k*sz, k*sz, -sz, sz, sz*2, 100); //установка перспективной проэкции
}
```

И перед основным циклом:

```
RECT rct;
   GetClientRect(hwnd,&rct);
   WndResize(rct.right,rct.bottom);
```

Для ориентации небольшая подключаемая библиотека camera.h. В ней есть структура для координат камеры:

```
struct SCamera {
   float x,y,z;
   float Xrot,Zrot;
};
```

и функции для их изменения.

Для удобства можно прописать в одну функцию и вызывать ее в основном цикле:

```
void MoveCamera(){
Camera_MoveDirectional(
        GetKeyState('W')< 0 ? 1 : GetKeyState('S')< 0 ? -1 : 0,
        GetKeyState('D')< 0 ? 1 : GetKeyState('A')< 0 ? -1 : 0,
        0. 1);
Camer_AutoMoveByMouse(400,400,0.1);
}
```

И в основном цикле условие для работы этой функции только если окно в фокусе:

```
if (GetForegroundWindow()==hwnd) MoveCamera();
```

Глубина.

Для правильного отображения перекрывающихся объектов есть буфер глубины, разрешение для работы с ним:

```
glEnable(GL_DEPTH_TEST);
```

также только один раз перед основным циклом.

И дополнительно нужно включить очистку буфера глубины в каждом кадре в первых строках главного цикла:

```
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Задания

1. Сделать оси системы координат

   Сделайте 3 статичных, разных по цвету линии в направлении по каждой из осей (X, Y, Z), при это не убирайте начальный треугольник.

2. Создать возможность перемещаться по сцене

   Библиотеку подключаем как обычно через camera.cpp и camera.h

   Что бы сразу после подключения и настройки перемещения видеть начальный треугольник перепишите 6 строку в camera.cpp:

```
…
camera={0,0,2.5, 285,0};
…
```

Если является ошибка … undefined reference to …, измените расширение основного файла с «.c» на «.cpp»

3. Сделать шахматную площадку n*n сегментов от центра во все стороны. Число n задаётся как один из аргументов функции.