1. **Нормализованные координаты**

Согласно методичке, нормализованные координаты - это координаты, заданные в промежуточной системе координат и нормированные относительно диапазона от -1 до +1. В вашем коде это используется в нескольких местах:

1. **Создание ортогональной матрицы**

Ортогональная матрица создается в нескольких местах:

// В MatrExtension.cs

public static Matrix4x4 CreateLookAtMatrix(in Vector3 camPos, in Vector3 camTar, in Vector3 camUpVec)

{

*// Создание ортогональной системы координат камеры*

    var aZ = Vector3.Normalize(-(camTar - camPos));  *// Первый базисный вектор*

    var aX = Vector3.Normalize(Vector3.Cross(camUpVec, aZ));  *// Второй базисный вектор*

    var aY = Vector3.Cross(aZ, aX);  *// Третий базисный вектор*

*// Формирование ортогональной матрицы*

    return new Matrix4x4(

        aX.X, aY.X, aZ.X, 0,

        aX.Y, aY.Y, aZ.Y, 0,

        aX.Z, aY.Z, aZ.Z, 0,

        Vector3.Dot(aX, negativeCameraPosition),

        Vector3.Dot(aY, negativeCameraPosition),

        Vector3.Dot(aZ, negativeCameraPosition),

        1

    );

}

5. Базис и его задание в коде

Базис - это система координатных векторов, которая определяет систему координат. В вашем коде базис задается в нескольких местах:

При создании системы координат камеры (в CreateLookAtMatrix):

var aZ = Vector3.Normalize(-(camTar - camPos));  *// Направление взгляда*

var aX = Vector3.Normalize(Vector3.Cross(camUpVec, aZ));  *// Правый вектор*

var aY = Vector3.Cross(aZ, aX);  *// Верхний вектор*

При создании мировой матрицы (в CrWorldMatr):

public static Matrix4x4 CrWorldMatr(in Vector3 posit, in Vector3 forward, in Vector3 up)

{

    var aZ = Vector3.Normalize(-forward);  *// Нормализованный вектор*

    var aX = Vector3.Normalize(Vector3.Cross(up, aZ));  *// Нормализованный перпендикулярный вектор*

    var aY = Vector3.Cross(aZ, aX);  *// Перпендикулярный вектор*

    var result = new Matrix4x4();

    for (var i = 0; i < 3; i++)

    {

        result[0, i] = aX[i];  *// Первая строка - вектор X*

        result[1, i] = aY[i];  *// Вторая строка - вектор Y*

        result[2, i] = aZ[i];  *// Третья строка - вектор Z*

        result[3, i] = posit[i];  *// Четвертая строка - позиция*

    }

    result[3, 3] = 1;

    return result;

}

В обоих случаях базис формируется как три взаимно перпендикулярных единичных вектора (ортонормированный базис), что обеспечивается через:

* Нормализацию векторов (Vector3.Normalize)
* Векторное произведение (Vector3.Cross) для получения перпендикулярных векторов
* Последовательное построение всех трех векторов базиса

Этот базис используется для преобразования координат между различными системами координат (мировой, камеры, экранной).

**Является ли мировая матрица ортогональной**   
Да, мировая матрица является ортогональной в своей верхней левой части 3×3, потому что:

1. Векторы aX, aY и aZ образуют ортонормированный базис:

* Они взаимно перпендикулярны (благодаря Vector3.Cross)
* Они нормализованы (благодаря Vector3.Normalize)
* Они образуют правую систему координат

1. Свойства ортогональной матрицы:

* Все столбцы (и строки) взаимно перпендикулярны
* Все столбцы (и строки) имеют единичную длину
* Транспонированная матрица равна обратной

Однако, полная 4×4 матрица не является строго ортогональной из-за компонента переноса (posit) и масштабирующего элемента (1 в позиции [3,3]). Это делает её аффинной матрицей преобразования, где:

* Верхняя левая часть 3×3 - ортогональная матрица поворота
* Правый столбец - вектор переноса
* Нижняя строка - [0, 0, 0, 1] для сохранения однородных координат

Поэтому более точно будет сказать, что мировая матрица - это аффинная матрица преобразования, содержащая ортогональную подматрицу 3×3 для поворотов.

1. **private const float X0 = 0.0f;** и **private const float Y0 = 0.0f;**

* Начальные координаты в плоскости экрана
* Определяют центр проекции на экране

1. **private const float MinDepth = -100;** и **private const float MaxDepth = 100;**

* Определяют диапазон глубины для z-буфера
* MinDepth = -100 - минимальное значение глубины
* MaxDepth = 100 - максимальное значение глубины
* Используются для правильного определения, какие объекты находятся спереди, а какие сзади

1. **NearPlaneDistance и FarPlaneDistance:**

Эти параметры используются при создании матрицы проекции в методе CreatePerspectivViewport:

Это реализовано через однородные координаты (x, y, z, w), где:

Ближняя плоскость (NearPlane) = 0.9f определяет минимальное расстояние от камеры до видимых объектов

Дальняя плоскость (FarPlane) = 1.0f определяет максимальное расстояние

Объекты между этими плоскостями будут видимы и правильно спроецированы

Объекты вне этого диапазона будут отсечены (не будут отображаться)

1. **Начальные координаты X0, Y0:**

Эти координаты используются для:

1. Определения точки, через которую проходит проекция
2. Смещения начала координат на экране

При изменении Y0 кажется, что меняется положение камеры, потому что:

* Y0 влияет на преобразование из 3D в 2D координаты
* Изменение Y0 смещает точку проекции по вертикали
* Это создает эффект, похожий на перемещение камеры, хотя на самом деле смещается только проекционная плоскость

Когда вы меняете Y0:

Положительное Y0 смещает проекцию вниз

Отрицательное Y0 смещает проекцию вверх

Это влияет на все точки сцены одновременно

Важно понимать разницу:

Перемещение камеры меняет точку обзора сцены

Изменение Y0 меняет только то, как 3D координаты проецируются на 2D экран

Хотя визуальный эффект может быть похож, механизм совершенно разный

В вашем проекте эти параметры работают вместе для создания правильной проекции 3D сцены на 2D экран, учитывая:

Область видимости камеры (через near/far planes)

Положение проекции на экране (через X0/Y0)

Перспективное искажение

Отсечение невидимых объектов

**В твоём проекте 3D-объекты проецируются на экран по оси Z.**  
Камера "смотрит" вдоль **отрицательной оси Z**, а плоскость экрана расположена в XY.

Давайте посмотрим каждый пункт в вашем коде. Сначала прочитаю основные файлы:

Read file: MyKursProj/Logic/Rendering.cs

Read file: MyKursProj/GeometryAlgorithms/MatrExtension.cs

Теперь разберем каждый пункт на основе вашего кода:

Согласно заданию из методички

1. \*\*Загрузка .obj файла\*\*:

```csharp

// В Program.cs

const string path = "15.obj";

var models = await ObjModel.LoadObjAsync(path);

```

2. \*\*Область видимости в нормализованных координатах\*\*:

```csharp

// В Rendering.cs

private const float FieldOfView = 1f;

private const float NearPlaneDistance = 0.9f;

private const float FarPlaneDistance = 1.0f;

```

3. \*\*Матрицы видового преобразования\*\*:

```csharp

// В MatrExtension.cs

// Масштабирование

public static Matrix4x4 CrScaleMatrix(in float size)

// Мировая матрица (включает поворот и сдвиг)

public static Matrix4x4 CrWorldMatr(in Vector3 posit, in Vector3 forward, in Vector3 up)

```

4. \*\*Положение наблюдателя\*\*:

```csharp

// В Rendering.cs

private Vector3 camPos;

private readonly Vector3 camTarget = new(0.0f, 0.0f, 0.0f);

private readonly Vector3 cameraUpVector = new(0.0f, 1.0f, 0.0f);

```

5. \*\*Матрица LookAt\*\*:

```csharp

// В MatrExtension.cs

public static Matrix4x4 CreateLookAtMatrix(in Vector3 camPos, in Vector3 camTar, in Vector3 camUpVec)

```

6. \*\*Матрица Viewport\*\*:

```csharp

// В MatrExtension.cs

public static Matrix4x4 CreateViewport(in float x, in float y, in float width, in float height,

in float minDepth, in float maxDepth)

```

7. \*\*Суперпозиция преобразований\*\*:

```csharp

// В Rendering.cs

final = world \* scale \* view \* projection \* viewport;

```

8. \*\*Движение наблюдателя\*\*:

```csharp

// В Rendering.cs

private void CameraTurn(in Vector3 cameraTurn)

{

if (cameraTurn == new Vector3(-1f, -1f, -1f))

CameraSpherePosition = radiusPhiTheta;

else

CameraSpherePosition +=

cameraTurn \* new Vector3(1f, SpectatorStep, SpectatorStep);

}

```

Все эти преобразования применяются в методе `DataUpdate`:

```csharp

private void DataUpdate(in Size \_size, in Vector3 cameraTurn)

{

size = \_size;

zBuffer = [];

CameraTurn(cameraTurn);

scale = MatrExtension.CrScaleMatrix(ScaleF);

view = MatrExtension.CreateLookAtMatrix(camPos, camTarget, cameraUpVector);

viewport = MatrExtension.CreateViewport(X0, Y0, size.Width, size.Height, MinDepth, MaxDepth);

projection = MatrExtension.CreatePerspectivViewport(FieldOfView,

(float)size.Width / size.Height, NearPlaneDistance, FarPlaneDistance);

}

```