Подробные пояснения к программе "Вращающийся куб с управлением камерой"

1. Новые структуры и переменные

`GeomBuffer` и `SceneBuffer` - это структуры для константных буферов (слайды 4, 7, 44). Константные буферы - это специальные области памяти, которые передают данные из CPU в шейдеры на GPU. Они называются "константными", потому что их содержимое не меняется в течение одного вызова шейдера.

`m_pGeomBuffer` и `m_pSceneBuffer` - это указатели на буферы в памяти видеокарты:

- `GeomBuffer` хранит матрицу модели (m) - она отвечает за положение, вращение и масштаб нашего куба
- `SceneBuffer` хранит матрицу вида-проекции (vp) - она объединяет настройки камеры и перспективы

Переменные управления камерой отслеживают вращение камеры с помощью мыши (слайд 39):

- `m_camera` - структура с точкой интереса, расстоянием и углами в сферической системе координат
- `m_rbPressed` - отслеживает нажатие правой кнопки мыши
- `m_prevMouseX`, `m_prevMouseY` - запоминают предыдущую позицию мыши для плавного управления

2. Инициализация буферов (InitBuffers)

Создаем два константных буфера с разными настройками (слайды 7, 8, 44):

`GeomBuffer` - для матрицы модели:

- Использует `D3D11_USAGE_DEFAULT` - буфер обновляется через `UpdateSubresource` (слайд 13)
- `CPUAccessFlags = 0` - процессор не имеет прямого доступа к этому буферу
- Подходит для данных, которые меняются не очень часто

`SceneBuffer` - для матрицы вида-проекции:

- Использует `D3D11_USAGE_DYNAMIC` - буфер будет часто обновляться
- `D3D11_CPU_ACCESS_WRITE` - процессор может записывать в этот буфер через `Map/Unmap` (слайд 45)
- Подходит для данных, которые меняются каждый кадр (как положение камеры)

3. Геометрия кубика (InitCube)

8 вершин - это все углы куба. Каждая вершина имеет:

- Позицию в 3D пространстве (x, y, z)
- Цвет (нижняя грань красная, верхняя - зеленая)

36 индексов для 12 треугольников:

- Каждая грань куба состоит из 2 треугольников
- 6 граней × 2 треугольника = 12 треугольников
- Каждый треугольник имеет 3 вершины, поэтому 12 × 3 = 36 индексов

Почему используем индексы? (слайды 11-12)

- Без индексов нам пришлось бы передавать 36 вершин (много повторяющихся данных)
- С индексами передаем только 8 уникальных вершин + 36 указателей на них
- Экономит память и увеличивает производительность

4. Обновленные шейдеры

Вершинный шейдер теперь использует два константных буфера (слайды 4, 5):

- `GeomBuffer` в регистре b0 - матрица модели
- `SceneBuffer` в регистре b1 - матрица вида-проекции

Последовательность преобразований в шейдере:

1. `worldPos = mul(float4(vertex.pos, 1.0), m)` - применяем матрицу модели (переход из локальных в мировые координаты)
2. `result.pos = mul(worldPos, vp)` - применяем матрицу вида-проекции (переход из мировых в однородные координаты отсечения)

Это соответствует математике из лекции: сначала преобразуем объект в мировые координаты, затем в координаты камеры с перспективой.

5. Управление камерой (UpdateCamera)

Матрица вида (view matrix) создается с помощью `XMMatrixLookAtLH` (слайд 39):

- `eye` - позиция камеры в мировых координатах (вычисляется из сферических координат)
- `at` - точка, куда смотрит камера (центр сцены, `m_camera.poi`)
- `up` - вектор "вверх" для ориентации камеры (обычно (0,1,0))

Матрица проекции (projection matrix) создается с помощью `XMMatrixPerspectiveFovLH` (слайды 36, 37):

- `fov` - угол обзора (field of view) в радианах (60 градусов = π/3)
- `aspectRatio` - соотношение сторон окна (ширина/высота)
- `nearZ`, `farZ` - ближняя и дальняя плоскости отсечения

Сферические координаты для камеры:

- Камера вращается вокруг точки интереса (poi) на расстоянии `m_camera.r`
- Углы `m_camera.phi` (горизонтальный) и `m_camera.theta` (вертикальный) вычисляются из движения мыши

6. Вращение кубика

В данном коде кубик не вращается (матрица модели - единичная). Однако в лекции рассматривалось вращение (слайды 12, 13, 24). Если бы мы хотели вращать кубик, мы бы обновляли матрицу модели каждый кадр, например, с помощью вращения вокруг осей.

Обновление буфера через `UpdateSubresource` (для GeomBuffer) и через `Map/Unmap` (для SceneBuffer) (слайды 13, 45):

- Для GeomBuffer: `UpdateSubresource` копирует данные из CPU памяти в GPU буфер (используется для буферов с `D3D11_USAGE_DEFAULT`)
- Для SceneBuffer: используем `Map/Unmap`, потому что буфер динамический (`D3D11_USAGE_DYNAMIC`)

7. Обработка ввода

Обработка сообщений мыши в функции `WndProc`:

- `WM_RBUTTONDOWN` - начинаем отслеживать движение мыши для камеры (правая кнопка)
- `WM_RBUTTONUP` - прекращаем отслеживание
- `WM_MOUSEMOVE` - обрабатываем движение мыши при зажатой кнопке
- `WM_MOUSEWHEEL` - обрабатываем прокрутку для изменения расстояния камеры

Вычисление дельты движения:

- `dx`, `dy` - разница по X и Y, нормализованная относительно размеров окна и умноженная на скорость вращения

Ограничение вертикального угла предотвращает переворот камеры:

- `m_camera.theta` ограничен между $-\pi/2+0.001$ и $\pi/2-0.001$ (чуть меньше 90 градусов в каждую сторону)

8. Отрисовка (Render)

Настройка конвейера:

- `OMSetRenderTargets()` - устанавливаем back buffer и depth stencil view как цель рендеринга
- `ClearRenderTargetView()` - очищаем экран темно-синим цветом
- `ClearDepthStencilView()` - очищаем буфер глубины
- `RSSetViewports()` - устанавливаем область вывода

Установка шейдеров и буферов:

- `VSSetShader()` и `PSSetShader()` - устанавливаем вершинный и пиксельный шейдеры
- `VSSetConstantBuffers(0, 2, constantBuffers)` - передаем оба константных буфера в вершинный шейдер (регистры b0 и b1)

Отрисовка куба:

- `DrawIndexed(36, 0, 0)` - рисуем 36 индексов (вместо 36 вершин без индексов)

- GPU автоматически интерполирует цвета между вершинами

Использованные концепции из лекции

Константные буферы (слайды 4, 5, 7, 8, 44):

- Передача данных из CPU в шейдеры
- Разделение на статические и динамические буферы (GeomBuffer и SceneBuffer)
- Использование разных регистров (b0, b1)

Матричные преобразования (слайды 12, 13, 24):

- Матрицы вращения (хотя в коде куб не вращается, но матрица модели есть)
- Комбинирование преобразований через умножение матриц (в шейдере: сначала модель, затем вид-проекция)
- Обновление буферов через UpdateSubresource и Map/Unmap

Перспективная проекция (слайды 36, 37):

- Матрица перспективы с углом обзора (XMMatrixPerspectiveFovLH)
- Ближняя и дальняя плоскости отсечения
- Соотношение сторон экрана

Разделение матриц (слайды 44, 45):

- Отдельные буферы для матрицы модели и вида-проекции
- Разная частота обновления для оптимизации (модель статична, вид-проекция обновляется каждый кадр)

Управление камерой (слайд 39):

- Матрица вида через LookAt (XMMatrixLookAtLH)
- Сферические координаты для орбитальной камеры
- Обработка ввода с мыши

Шейдеры (слайды 4, 5):

- Вершинный шейдер выполняет преобразования координат
- Пиксельный шейдер определяет цвет каждого пикселя (в данном случае просто цвет вершины)
- Использование константных буферов в шейдерах

Геометрия (слайды 11, 12):

- Использование буферов вершин и индексов
- Экономия памяти за счет индексации

Буфер глубины (не рассматривался в лекции, но важен для правильного отображения 3D):

- Позволяет определять, какие пиксели ближе и должны быть отрисованы поверх дальних

!!!

Кубик вращается, если зажать правую кнопку мыши.

Можно также приближать и отдалять кубик.

Код на данный момент

Краткая таблица соответствия кода лекциям.

Строки кода / Функция	Объяснение	Соответствующие страницы PDF
`InitWindow()` с `AdjustWindowRect`	Создание окна с расчетом клиентской области	2.Инициализация.pdf: стр. 5-6
Цикл с `PeekMessage`	Графический цикл приложения	2.Инициализация.pdf: стр. 4
`InitDirectX()` с выбором адаптера	Инициализация DXGI и выбор GPU	2.Инициализация.pdf: стр. 10-15
Создание `ID3D11Debug`	Отладочный слой DirectX	2.Инициализация.pdf: стр. 16-20
`DXGI_SWAP_EFFECT_FLIP_DISCARD`	Создание swap chain с современным алгоритмом смены буферов	2.Инициализация.pdf: стр. 24-27
Структура `Vertex`	Описание формата вершин	3.Треугольник.pdf: стр. 7
Создание вершинного и индексного буферов	Геометрия куба в видеопамяти	3.Треугольник.pdf: стр. 8-12
Компиляция шейдеров через `D3DCompile`	Компиляция HLSL в байткод	3.Треугольник.pdf: стр. 18-22
`CreateInputLayout`	Разметка вершинного буфера	3.Треугольник.pdf: стр. 28
Настройка `IASetVertexBuffers` и др.	Конфигурация Input Assembler	3.Треугольник.pdf: стр. 32
Вершинный шейдер с трансформациями	Преобразование координат через матрицы	5.Перспективная проекция.pdf: стр. 4-5
Структуры `GeomBuffer` и `SceneBuffer`	Константные буферы для матриц	5.Перспективная проекция.pdf: стр. 7-8
`CreateBuffer` для константных буферов	Создание буферов преобразований	5.Перспективная проекция.pdf: стр. 7-8
`UpdateSubresource` для `m_pGeomBuffer`	Обновление матрицы модели на GPU	5.Перспективная проекция.pdf: стр. 12-13
`Map`/`Unmap` для `m_pSceneBuffer`	Динамическое обновление матрицы вида-проекции	5.Перспективная проекция.pdf: стр. 45
`XMMatrixLookAtLH` и `XMMatrixPerspectiveFovLH`	Математика камеры и проекции через DirectXMath	5.Перспективная проекция.pdf: стр. 39
Управление камерой мышью	Вращение камеры	5.Перспективная

Строки кода / Функция	Объяснение	Соответствующие страницы PDF
	вокруг объекта	проекция.pdf: стр. 47 (задание)
Буфер глубины (`m_pDepthBuffer`, `m_pDepthStencilView`)	Тестирование глубины для 3D	-
`OMSetDepthStencilState`	Настройка теста глубины	-
`ClearDepthStencilView`	Очистка буфера глубины	-
Разделение матриц m и vp в разные буферы	Оптимизация для множества объектов	5.Перспективная проекция.pdf: стр. 42
`VSSetConstantBuffers` с двумя буферами	Установка константных буферов в шейдере	5.Перспективная проекция.pdf: стр. 42

Примечания:

- Буфер глубины не был явно рассмотрен в предоставленных лекциях, но необходим для 3D-рендеринга
- Управление камерой мышью было заданием, но конкретная реализация через сферические координаты не показана в лекциях
- Использование `DirectXMath` соответствует рекомендациям из лекций (стр. 39)
- Разделение матриц на m и vp реализовано согласно рекомендациям (стр. 42)
- Все основные концепции из лекций были реализованы в коде

Код на данный момент не похож на представленный на гитхабе готовый вариант (https://github.com/anthark/DX11Tutorial2).