Краткая справка по формату DDS

DDS (DirectDraw Surface) - это формат файлов текстур, разработанный Microsoft для использования в DirectX. Это **нативный формат текстур** для графических API DirectX.

Основные особенности:

1. Сжатие текстур

- Поддерживает форматы сжатия **BC1-BC7** (Block Compression)
- **BC1 (DXT1)** - 4 бита/пиксель, без альфа-канала
- **BC2 (DXT3)** - 8 бит/пиксель, с резкой альфой
- **BC3 (DXT5)** - 8 бит/пиксель, с плавной альфой
- **BC4/BC5** - для нормальных карт
- **BC6H/BC7** - для HDR и высококачественных текстур

Блочное сжатие - это аппаратное сжатие текстур, где изображение разбивается на блоки 4×4 пикселя, которые сжимаются независимо.

2. Mip-уровни

cpp
```
desc.mipmapsCount = textureDesc.mipmapsCount; // Уровни детализации
```

- Содержит предварительно рассчитанные уменьшенные копии текстуры
- Ускоряет рендеринг на разных расстояниях

3. Типы текстур

- **2D текстуры** (обычные)
- **Cube maps** (6 граней для skybox)
- **Volume textures** (3D)
- **Array textures**

В вашем коде:

Загрузка DDS:

cpp
```
TextureDesc textureDesc;
bool success = LoadDDS(L"../Common/Kitty.dds", textureDesc);
```

Структура DDS файла:

text
```
[Сигнатура "DDS "] → [Основной заголовок] → [DXT10 заголовок*] → [Данные текстуры]
```

Преимущества в вашем проекте:

- **Быстрая загрузка** - данные готовы для видеопамяти
- **Аппаратное сжатие** - поддерживается видеокартами
- **Mip-уровни** - улучшают качество фильтрации

- **Cube maps** - для skybox

Пример использования:

cpp
```
// В Renderer.cpp загружается текстура кота
const std::wstring TextureName = L"../Common/Kitty.dds";
TextureDesc textureDesc;
LoadDDS(TextureName.c_str(), textureDesc);

// И cubemap для неба
const std::wstring TextureNames[6] = {
    L"../Common/posx.dds", L"../Common/negx.dds", // и т.д.
};
```

DDS - это оптимальный формат для игровых текстур, так как минимизирует обработку при загрузке и максимально использует возможности графического hardware.

Таблица функций и методов в порядке их появления в коде

№	Функция/Метод	Назначение и принцип работы	Ссылка на лекцию
1	`WinMain`	Точка входа в приложение. Инициализирует окно, DirectX, ресурсы и запускает главный цикл.	Лекция 2, стр. 2–4
2	`InitWindow`	Создаёт и регистрирует окно с заданными размерами и стилем.	Лекция 2, стр. 5–6
3	`InitDirectX`	Инициализирует DirectX: создаёт устройство, контекст, своп-цепь, растеризатор и буфер глубины.	Лекция 2, стр. 7–15
4	`D3D11CreateDeviceAndSwapChain`	Создаёт логическое устройство GPU (ID3D11Device) и своп-цепь для вывода в окно.	Лекция 2, стр. 13–15
5	`SetupBackBuffer`	Создаёт render target view для back buffer’а своп-цепи и буфер глубины.	Лекция 2, стр. 28–29
6	`InitCube`	Создаёт геометрию куба (вершинный и индексный буферы), компилирует шейдеры, создаёт input layout.	Лекция 3, стр. 5–12; Лекция 6, стр. 11–13
7	`D3DCompile`	Компилирует шейдер из исходного кода в байт-код DXBC.	Лекция 3, стр. 17–22
8	`CreateInputLayout`	Описывает структуру вершинного буфера для передачи в вершинный шейдер.	Лекция 3, стр. 26–28
9	`InitBuffers`	Создаёт константные буферы для матриц модели (`m`) и вида.	Лекция 5, стр. 6–8
10	`LoadTexture`	Загружает DDS-текстуру, создаёт ресурс текстуры, shader resource view и сэмплер.	Лекция 6, стр. 14–26
11	`LoadDDS`	Читает DDS-файл, извлекает заголовок, формат, данные и мип-уровни.	Лекция 6, стр. 20–24
12	`InitSkybox`	Создаёт сферу для skybox, загружает cubemap-текстуру, компилирует шейдеры.	Лекция 6, стр. 34–42
13	`CreateSphere`	Генерирует вершины и индексы для сферы методом параметризации.	Лекция 6, стр. 34
14	`WndProc`	Обрабатывает сообщения окна: изменение размера, вращение камеры, колесо мыши.	Лекция 2, стр. 2–4
15	`ResizeSwapChain`	Изменяет размер back buffer’а и буфера глубины при изменении окна.	Лекция 2, стр. 28
16	`UpdateCamera`	Обновляет матрицу вида и проекции на основе параметров камеры, записывает в буфер.	Лекция 5, стр. 21–24
17	`Render`	Основной цикл рендеринга: очистка, настройка конвейера, отрисовка skybox и куба.	Лекция 3, стр. 52–54; Лекция 6, стр. 28
18	`IASetVertexBuffers` / `IASetIndexBuffer`	Настраивают входной ассемблер: привязывают вершинный и индексный	Лекция 3, стр. 32

№	Функция/Метод	Назначение и принцип работы	Ссылка на лекцию
		буферы.	
19	`VSSetShader` / `PSSetShader`	Устанавливают вершинный и пиксельный шейдеры в конвейер.	Лекция 3, стр. 23–24
20	`PSSetShaderResources` / `PSSetSamplers`	Передают текстуры и сэмплеры в пиксельный шейдер.	Лекция 6, стр. 27–28
21	`VSSetConstantBuffers`	Привязывают константные буферы к вершинному шейдеру.	Лекция 5, стр. 6
22	`DrawIndexed`	Запускает отрисовку геометрии по индексам.	Лекция 3, стр. 54
23	`UpdateSubresource`	Копирует данные из CPU в GPU-буфер (например, матрицу модели).	Лекция 5, стр. 9–10
24	`Map` / `Unmap`	Отображают GPU-буфер в CPU-память для записи (динамический буфер сцены).	Лекция 5, стр. 43–45
25	`Cleanup`	Освобождает все созданные DirectX-ресурсы.	–
26	`CreateSphere` (вспом.)	Генерация геометрии сферы для skybox.	Лекция 6, стр. 34
27	`GetBytesPerBlock` / `DivUp`	Вспомогательные функции для работы с DXT-сжатыми текстурами.	Лекция 6, стр. 17–19

Ключевые моменты для защиты:

1. **Инициализация DirectX и окна** — основана на лекции 2. Важно понимать роль `ID3D11Device`, `ID3D11DeviceContext` и `IDXGISwapChain`.
2. **Геометрия и шейдеры** — описаны в лекции 3. Вершинный буфер, индексный буфер, input layout и компиляция шейдеров.
3. **Матрицы и камера** — лекция 5. Константные буферы, матрицы вида и проекции, управление камерой.
4. **Текстурирование** — лекция 6. Загрузка DDS, создание текстур, сэмплеров, работа с cubemap.
5. **Рендеринг** — порядок вызовов: очистка, настройка конвейера, привязка ресурсов, отрисовка.
6. **Обработка сообщений** — реагирование на изменение размера окна и ввод мыши.

Таблица построена в порядке следования кода — можно идти по ней параллельно с чтением исходника.

Сравнение кодов: что нового добавилось в task_4

Вот подробное сравнение нового кода (`task_4`) с предыдущим (`task_3`):

1. Текстурирование кубика

Компонент	`task_3` (старый)	`task_4` (новый)	Что изменилось
Структура вершины	`struct Vertex` с позицией и цветом (RGBA)	`struct TextureVertex` с позицией и UV-координатами	Вместо цвета в вершине теперь хранятся текстурные координаты
Шейдеры куба	Цвет берется из вершины (`vertex.color`)	Цвет берется из текстуры (`colorTexture.Sample()`)	Добавлено текстурирование вместо вершинных цветов
Текстуры	Нет текстур	Добавлены: - `ID3D11Texture2D* m_pTexture` - `ID3D11ShaderResourceView* m_pTextureView` - `ID3D11SamplerState* m_pSampler`	Полная система текстурирования
Загрузка DDS	Нет	Добавлена функция `LoadTexture()` и вспомогательные функции `LoadDDS()`, `GetBytesPerBlock()`, `DivUp()`	Поддержка формата DDS (DirectDraw Surface)

2. Skybox (небесная сфера)

Компонент	`task_3` (старый)	`task_4` (новый)	Что изменилось
Геометрия skybox	Нет	Добавлена сфера: - `m_pSphereVertexBuffer` - `m_pSphereIndexBuffer` - `m_sphereIndexCount`	Создание сферической геометрии для окружения
Шейдеры skybox	Нет	Добавлены: - `m_pSphereVertexShader` - `m_pSpherePixelShader` - `m_pSphereInputLayout`	Отдельные шейдеры для skybox
Cubemap текстура	Нет	Добавлены: - `m_pCubemapTexture` - `m_pCubemapView`	Кубическая текстура из 6 граней

Компонент	task_3 (старый)	task_4 (новый)	Что изменилось
Константный буфер skybox	Нет	`m_pSphereGeomBuffer` с `SphereGeomBuffer` структурой (матрица + размер)	Отдельный буфер для параметров skybox

3. Состояния глубины

Компонент	task_3 (старый)	task_4 (новый)	Что изменилось
Состояния глубины	Одно состояние: `m_pDepthStencilState`	Два состояния: - `m_pSkyboxDepthState` (только чтение) - `m_pNormalDepthState` (чтение и запись)	Разделение для правильного рендеринга skybox

4. Изменения в функциях

Инициализация:

Функция	task_3	task_4	Изменения
WinMain	Инициализирует только куб	Дополнительно загружает текстуру и инициализирует skybox	Расширенная инициализация
InitDirectX	Создает одно состояние глубины	Создает два состояния глубины (skybox и обычное)	Поддержка разных режимов глубины
InitCube	Создает цветной куб (8 вершин)	Создает текстурированный куб (24 вершины с UV)	Полная переработка геометрии под текстурирование
Новая функция	-	`InitSkybox()`	Инициализация небесной сферы
Новая функция	-	`LoadTexture()`	Загрузка DDS текстур

Рендеринг:

Функция	task_3	task_4	Изменения
Render	Рендерит только куб	Рендерит skybox, затем куб	Многослойный рендеринг
UpdateCamera	Вычисляет только `vp` матрицу	Вычисляет `vp` и позицию камеры (`cameraPos`)	Позиция камеры нужна для skybox

Функция	`task_3`	`task_4`	Изменения
ResizeSwapChain	Только изменение размера	Дополнительно обновляет радиус skybox	Skybox зависит от aspect ratio

5. Новые структуры данных

Структура	Назначение в `task_4`	Отличие от `task_3`
`TextureVertex`	Вершина с UV-координатами	Заменяет `Vertex` с цветом
`TextureDesc`	Описание загруженной DDS текстуры	Новая структура для работы с DDS
`SphereGeomBuffer`	Константный буфер для skybox	Дополнительный буфер с размером сферы
`Camera`	Та же структура, но в `UpdateCamera` теперь вычисляется `cameraPos`	Добавлено сохранение позиции камеры

6. Новые шейдеры

Куб:

- `task_3`: Вершинный шейдер принимает `COLOR`, пиксельный возвращает `vertex.color`
- `task_4`: Вершинный шейдер принимает `TEXCOORD`, пиксельный читает текстуру

Skybox (новое в `task_4`):

- Вершинный шейдер: преобразует сферу, учитывает позицию камеры
- Пиксельный шейдер: читает cubemap по 3D-направлению (`TextureCube`)

7. Ключевые концептуальные добавления

1. **Текстурирование вместо вершинных цветов** - более реалистичная визуализация
2. **Skybox** - создание бесконечного окружения
3. **Cubemap** - специальный тип текстуры для окружения (6 граней)
4. **Раздельные состояния глубины** - skybox не должен влиять на Z-буфер
5. **DDS формат** - стандарт для текстур в DirectX, поддержка сжатия (DXT)
6. **Многослойный рендеринг** - правильный порядок: skybox → объекты

8. Порядок рендеринга в `task_4`:

1. Очистка буферов
2. Установка состояния глубины для skybox (только чтение)
3. Рендеринг skybox
4. Установка обычного состояния глубины (чтение и запись)
5. Рендеринг кубика
6. Present

9. Зависимости от лекций:

- **Лекция 6 "Текстурирование"** - объясняет все новые концепции:
  - Текстурные координаты (стр. 4-7)
  - Создание текстур и семплеров (стр. 14-26)
  - DXT сжатие (стр. 17-19)
  - Skybox и cubemap (стр. 31-42)

Итог: `task_4` добавляет полную систему текстурирования и окружения (skybox), что значительно улучшает визуальное качество по сравнению с простым цветным кубом в `task_3`.

Детальный конспект

1. Инициализация приложения

Область кода	Что делает	Как работает	Теория
WinMain()	Точка входа Windows приложения	1. Инициализирует окно 2. Инициализирует DirectX 3. Загружает ресурсы (куб, текстуры, skybox) 4. Запускает главный цикл сообщений 5. При завершении освобождает ресурсы	Лекция 2, стр. 2-4
InitWindow()	Создает и регистрирует окно	1. Регистрирует класс окна (WNDCLASSEX) 2. Вычисляет размеры с учетом рамок (AdjustWindowRect) 3. Создает окно (CreateWindow) 4. Показывает окно (ShowWindow)	Лекция 2, стр. 5-6
WndProc()	Обрабатывает сообщения окна	Обрабатывает: - WM_SIZE: изменение размера окна → ResizeSwapChain() - WM_RBUTTONDOWN/UP: вращение камеры мышью - WM_MOUSEWHEEL: зум камеры - WM_DESTROY: завершение приложения	Лекция 2, стр. 2-4

2. Инициализация DirectX 11

Область кода	Что делает	Как работает	Теория
InitDirectX()	Инициализирует графический конвейер	1. Создает устройство и своп-цепь (D3D11CreateDeviceAndSwapChain) 2. Создает растеризатор (CreateRasterizerState) 3. Создает состояния глубины для skybox и обычных объектов 4. Настраивает back buffer и буфер глубины (SetupBackBuffer)	Лекция 2, стр. 7-30
D3D11CreateDeviceAndSwapChain()	Создает основные объекты DirectX	Создает: - ID3D11Device: логическое представление GPU - ID3D11DeviceContext: контекст для команд - IDXGISwapChain: цепочка буферов для вывода в окно	Лекция 2, стр. 13-15, 26-27
SetupBackBuffer()	Настраивает цели рендеринга	1. Получает back buffer из своп-цепи (GetBuffer) 2. Создает Render Target View (CreateRenderTargetView) 3. Создает текстуру буфера глубины (CreateTexture2D) 4. Создает Depth Stencil View (CreateDepthStencilView)	Лекция 2, стр. 28-30

3. Геометрия и шейдеры куба

Область кода	Что делает	Как работает	Теория
InitCube()	Создает 3D куб с текстурами	1. Определяет 24 вершины с UV-координатами 2. Определяет 36 индексов для 12 треугольников 3. Создает вершинный и индексный буферы 4. Компилирует шейдеры из строкового кода 5. Создает Input Layout	Лекция 3, стр. 5-12; Лекция 6, стр. 11-13
Структура TextureVertex	Описывает формат вершины	Содержит: - float x, y, z: позиция в пространстве - float u, v: текстурные координаты (0.0-1.0)	Лекция 6, стр. 4-7
Вершинный шейдер куба	Преобразует вершины	1. Умножает вершину на матрицу модели (m) 2. Умножает на матрицу вида-проекции (vp) 3. Передает UV-координаты в пиксельный шейдер	Лекция 5, стр. 4-5, 21-24
Пиксельный шейдер куба	Применяет текстуру	1. Берет текстуру (Texture2D) и сэмплер 2. Читает цвет по UV-координатам (Sample) 3. Возвращает цвет пикселя	Лекция 6, стр. 27
D3DCompile()	Компилирует HLSL шейдеры	Преобразует текстовый код шейдера в бинарный DXBC, который понимает GPU. В Debug-режиме добавляет отладочную информацию	Лекция 3, стр. 17-22
CreateInputLayout()	Связывает буфер вершин с шейдером	Описывает соответствие между полями в вершинном буфере и семантиками в шейдере (POSITION, TEXCOORD)	Лекция 3, стр. 25-28

4. Матрицы и система камеры

Область кода	Что делает	Как работает	Теория
InitBuffers()	Создает константные буферы	1. Создает буфер для матрицы модели (GeomBuffer) 2. Создает буфер для матрицы вида-проекции и позиции камеры (SceneBuffer)	Лекция 5, стр. 6-8
UpdateCamera()	Обновляет матрицы камеры	1. Из сферических координат (r, theta, phi) вычисляет позицию камеры 2. Создает матрицу вида (XMMatrixLookAtLH) 3. Создает матрицу проекции (XMMatrixPerspectiveFovLH) 4. Перемножает и транспонирует матрицы 5. Записывает в буфер через Map/Unmap	Лекция 5, стр. 21-24, 33-37
XMMatrixLookAtLH()	Создает матрицу вида	Принимает позицию камеры (eye), точку наблюдения (at) и вектор "вверх" (up).	Лекция 5, стр.

Область кода	Что делает	Как работает	Теория
		Возвращает матрицу, переводящую мировые координаты в систему координат камеры	21
XMMatrixPerspectiveFovLH()	Создает матрицу проекции	Принимает угол обзора (fov), соотношение сторон, ближнюю и дальнюю плоскости. Возвращает матрицу перспективной проекции	Лекция 5, стр. 33-37
Map()/Unmap()	Обновление динамического буфера	1. Map с флагом D3D11_MAP_WRITE_DISCARD получает указатель на память GPU 2. CPU записывает данные 3. Unmap завершает доступ	Лекция 5, стр. 43-45
UpdateSubresource()	Копирует данные в GPU-буфер	Копирует данные из CPU-памяти в буфер на GPU. Используется для буферов, которые меняются нечасто (например, матрица модели)	Лекция 5, стр. 9-10

5. Текстурирование и DDS формат

Область кода	Что делает	Как работает	Теория
LoadTexture()	Загружает 2D текстуру	1. Загружает DDS файл (LoadDDS) 2. Проверяет поддержку формата (CheckFormatSupport) 3. Создает текстуру (CreateTexture2D) 4. Создает Shader Resource View (CreateShaderResourceView) 5. Создает сэмплер (CreateSamplerState)	Лекция 6, стр. 14-26
LoadDDS()	Читает DDS файл	1. Проверяет сигнатуру файла (0x20534444 = "DDS ") 2. Читает заголовок (DDSHeader) 3. Определяет формат по FourCC коду (DXT1, DXT3, DXT5) 4. Читает данные текстуры с мип-уровнями	Лекция 6, стр. 17-19, 20-24
CreateSamplerState()	Настраивает чтение текстуры	Создает сэмплер с параметрами: - Filter: ANISOTROPIC (анизотропная фильтрация) - AddressU/V/W: WRAP (повторение текстуры) - MaxAnisotropy: 16 - BorderColor: белый	Лекция 6, стр. 8
DXT сжатие	Сжатие текстур	Форматы BC1/DXT1, BC2/DXT3, BC3/DXT5. Блоки 4×4 пикселя. DXT1: 8 байт на блок (сжатие 6:1). Поддерживается аппаратно на GPU	Лекция 6, стр. 17-19

6. Skybox (небесная сфера)

Область кода	Что делает	Как работает	Теория
InitSkybox()	Инициализирует небесную сферу	1. Генерирует геометрию сферы (CreateSphere) 2. Создает вершинный и индексный буферы 3. Компилирует шейдеры для skybox	Лекция 6, стр. 31-42

Область кода	Что делает	Как работает	Теория
		4. Загружает 6 текстур для cubemap 5. Создает cubemap текстуру и SRV	
CreateSphere()	Генерирует сферу из треугольников	Параметризация по широте/долготе. Для каждого узла сетки вычисляет сферические координаты, затем преобразует в декартовы (x, y, z)	Лекция 6, стр. 34
Cubemap текстура	6 текстур для окружения	6 граней: +X, -X, +Y, -Y, +Z, -Z. В коде загружаются как отдельные DDS файлы, затем объединяются в один ресурс с флагом D3D11_RESOURCE_MISC_TEXTURECUBE	Лекция 6, стр. 33, 39-42
Шейдер skybox	Отрисовывает небесную сферу	Вершинный шейдер: масштабирует сферу, добавляет позицию камеры, передает локальные координаты Пиксельный шейдер: читает cubemap по 3D-направлению (TextureCube.Sample)	Лекция 6, стр. 36-38
Состояние глубины для skybox	Skybox рисуется "на бесконечности"	DepthWriteMask = ZERO (не записывает глубину), DepthFunc = LESS_EQUAL. Это позволяет skybox рисоваться позади всего, но не влиять на буфер глубины	Лекция 6, стр. 34

7. Рендеринг (главный цикл)

Область кода	Что делает	Как работает	Теория
Render()	Рендерит один кадр	1. Очищает back buffer и буфер глубины 2. Устанавливает viewport 3. Обновляет камеру 4. Рендерит skybox 5. Рендерит куб 6. Отображает кадр (Present)	Лекция 3, стр. 52-54
Очистка буферов	Подготовка к новому кадру	ClearRenderTargetView: заливает back buffer цветом ClearDepthStencilView: устанавливает глубину в 1.0 (максимум)	Лекция 2, стр. 30
Настройка конвейера	Устанавливает состояние рендеринга	IASet: вершинный буфер, индексный буфер, input layout, топология VSSet/PSSet: шейдеры PSSetShaderResources/PSSetSamplers: текстуры и сэмплеры VSSetConstantBuffers: константные буферы	Лекция 3, стр. 32, 52-54
DrawIndexed()	Запускает отрисовку	Принимает количество индексов. Запускает весь графический конвейер для указанной геометрии	Лекция 3, стр. 54
Present()	Отображает кадр	Выводит back buffer на экран. Параметр (1, 0) включает VSync (ожидание вертикальной синхронизации)	Лекция 2, стр. 30

8. Обработка изменения размера окна

Область кода	Что делает	Как работает	Теория
ResizeSwapChain()	Изменяет размер буферов	1. Сбрасывает текущие RTV и DSV 2. Изменяет размер своп-цепи (ResizeBuffers) 3. Пересоздает back buffer и буфер глубины 4. Обновляет радиус skybox (зависит от aspect	Лекция 2, стр. 28

Область кода	Что делает	Как работает	Теория
		ratio)	
WM_SIZE сообщение	Обработка изменения размера окна	При изменении размера окна Windows отправляет WM_SIZE. WndProc вызывает ResizeSwapChain для пересоздания буферов с новым размером	Лекция 2, стр. 2-4

9. Управление камерой

Область кода	Что делает	Как работает	Теория
Структура Camera	Хранит параметры камеры	Сферические координаты: - poi: точка интереса (центр вращения) - r: расстояние от камеры до точки - theta: угол наклона (вертикаль) - phi: угол поворота (горизонталь)	Лекция 5 (неявно)
Обработка мыши	Вращение и зум камеры	WM_RBUTTONDOWN/UP: захват/освобождение мыши WM_MOUSEMOVE: изменение углов phi и theta WM_MOUSEWHEEL: изменение расстояния r	Реализовано для интерактивности

10. Вспомогательные функции

Область кода	Что делает	Как работает	Теория
Cleanup()	Освобождает ресурсы	Вызывает Release() для всех COM-объектов DirectX в порядке, обратном созданию	Принципы COM
SAFE_RELEASE макрос	Безопасное освобождение	Проверяет указатель на nullptr перед вызовом Release(), затем обнуляет указатель	Лекция 2, стр. 9
DivUp()	Деление с округлением вверх	(a + b - 1) / b. Используется для расчета количества блоков в сжатых текстурах (размер должен быть кратен 4)	Лекция 6, стр. 22
GetBytesPerBlock()	Размер блока DXT сжатия	BC1/DXT1: 8 байт на блок 4×4 BC2/DXT3, BC3/DXT5, BC4-7: 16 байт на блок 4×4	Лекция 6, стр. 17-19

Ключевые концепции для защиты:

1. **Графический конвейер DirectX 11**: IA → VS → RS → PS → OM. Код точно следует этой последовательности в функции Render().
2. **Текстурирование**: Загрузка DDS, создание SRV, настройка сэмплера (фильтрация, адресация). DXT сжатие экономит память и пропускную способность.
3. **Матричные преобразования**: Локальные → Мировые → Вид → Проекция → Экранные. Реализовано через константные буферы и умножение матриц в шейдере.
4. **Skybox**: Cubemap вместо 2D текстуры на сфере для минимальных искажений. Особое состояние глубины (не записывает глубину).
5. **Управление ресурсами**: Правильное создание и освобождение COM-объектов, обработка изменения размера окна.
6. **Шейдеры**: HLSL компилируется в runtime. Input Layout связывает структуры вершин с семантиками шейдера.