ТЕХНОАТОМ

Лекция 9. Введение в визуализацию при помощи OpenGL

МИФИ, 2016

Роман Кузнецов

ТЕХНОАТОМ

Просьба отметиться на портале!

План лекции

1. Графический конвейер.
2. Обзор OpenGL.
3. Создание простейшего OpenGL-приложения.

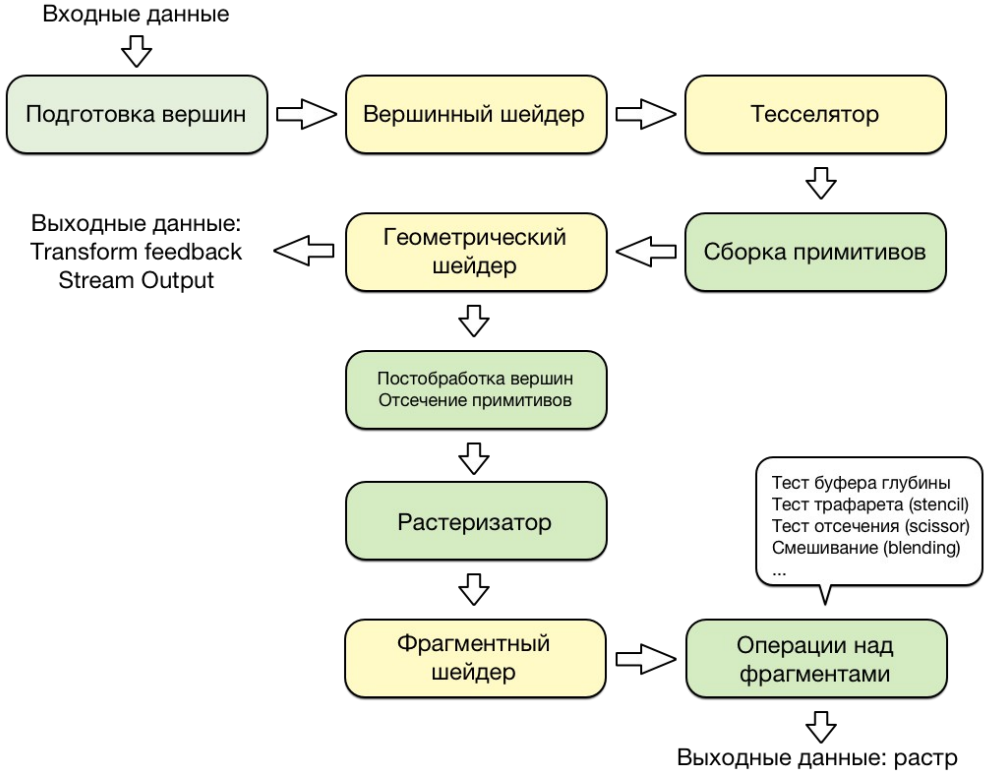
Графический конвейер

- аппаратно-программный комплекс, задачей которого является отображение на экране графики.

Процесс отображения графики на экрана называется **рендерингом**.

Программа, работающая на том или ином участке конвейера, называется **шейдером**.

Графический конвейер



Примитивы

1) Точка

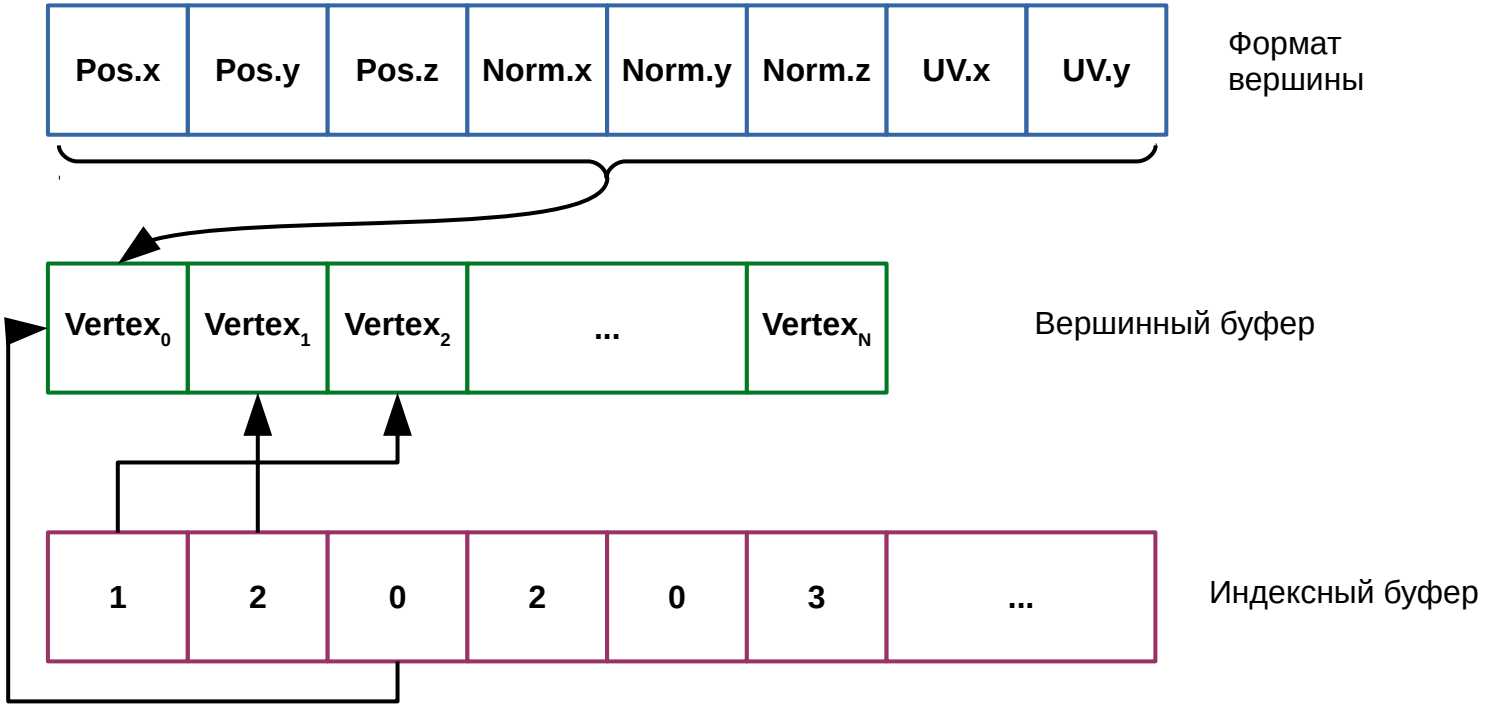
2) Линия

3) Треугольник

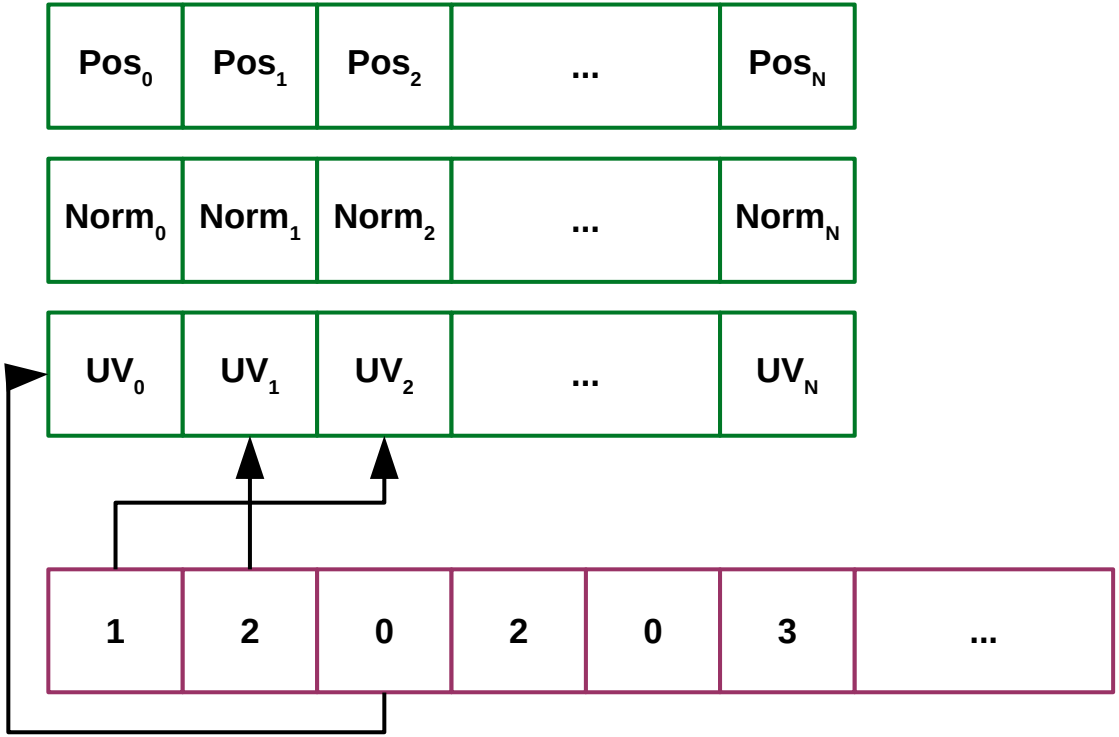
Все состоят из определенного количества вершин.

Вершина — сущность, содержащая некоторые атрибуты, необходимые для определения графических свойств примитива (позиция на экране, цвет и т.д.)

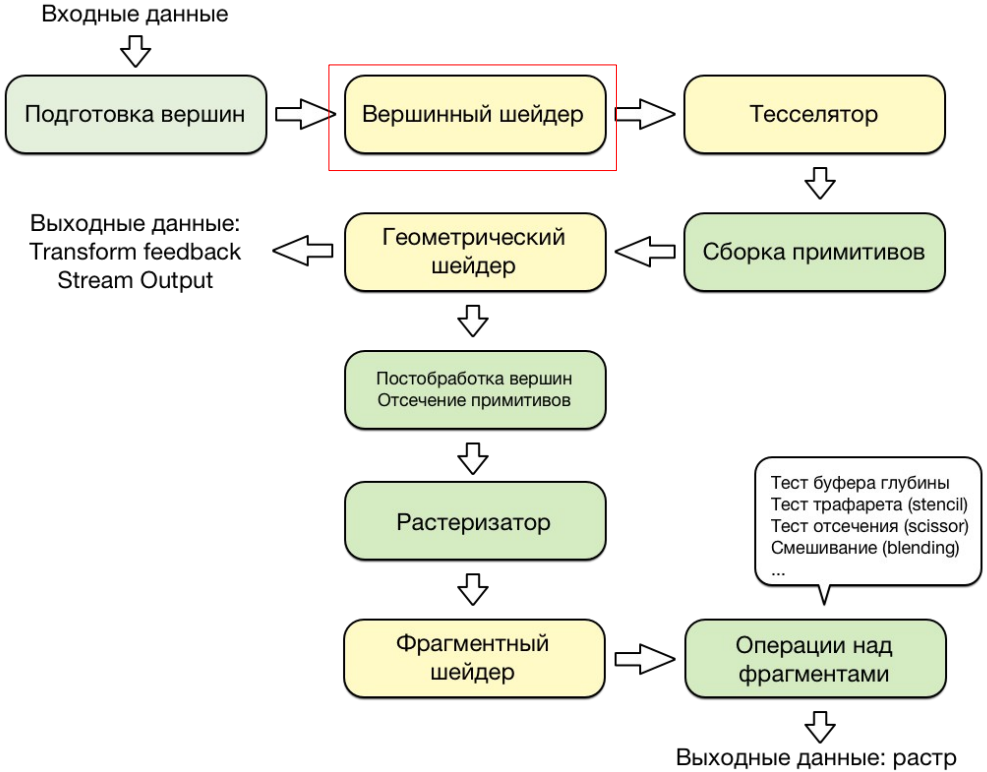
Подготовка вершин



Подготовка вершин



Графический конвейер



Вершинный шейдер

```
attribute highp vec3 a_position;
attribute highp vec2 a_texCoord;
```

Входные данные (вершина)

```
uniform mediump mat4 u_modelViewProjection;
```

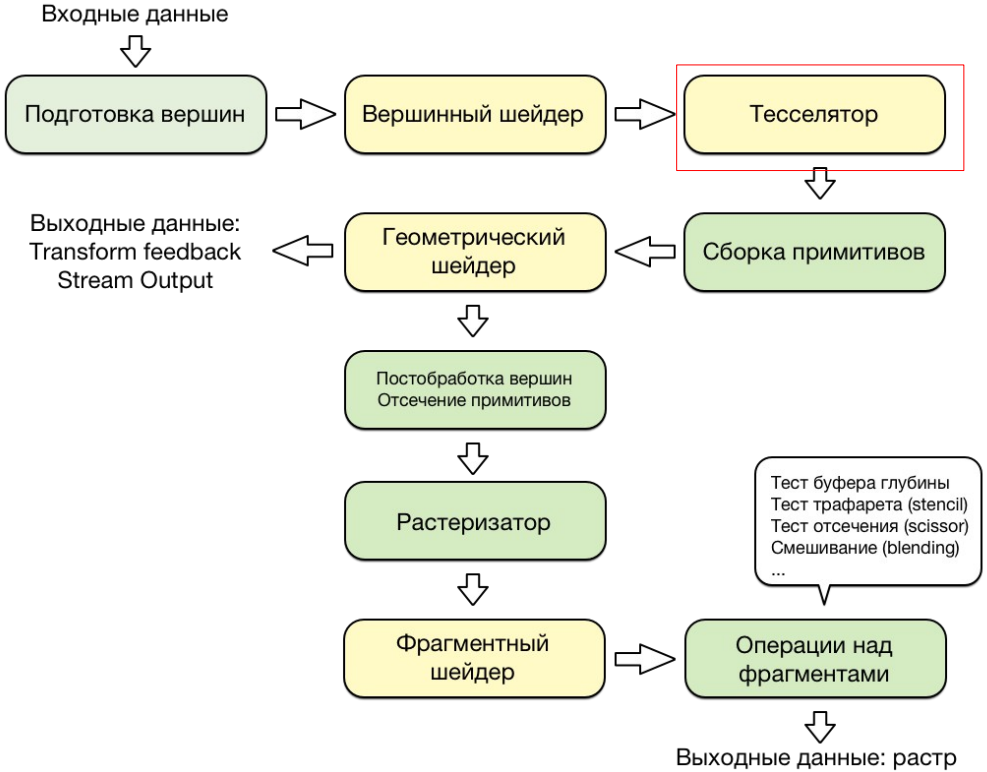
Параметры

```
varying highp vec2 v_texCoord;
```

Выходные данные

```
void main(void)
{
  gl_Position = u_modelViewProjection * vec4(a_position, 1.0);
  v_texCoord = a_texCoord;
};
```

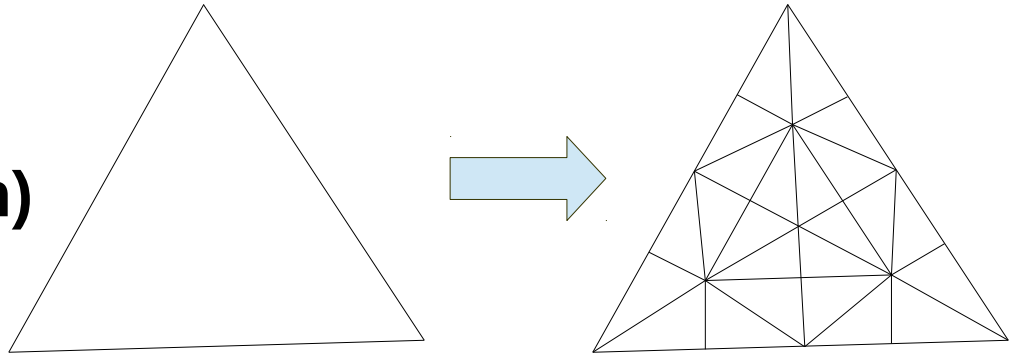
Графический конвейер



Тесселятор

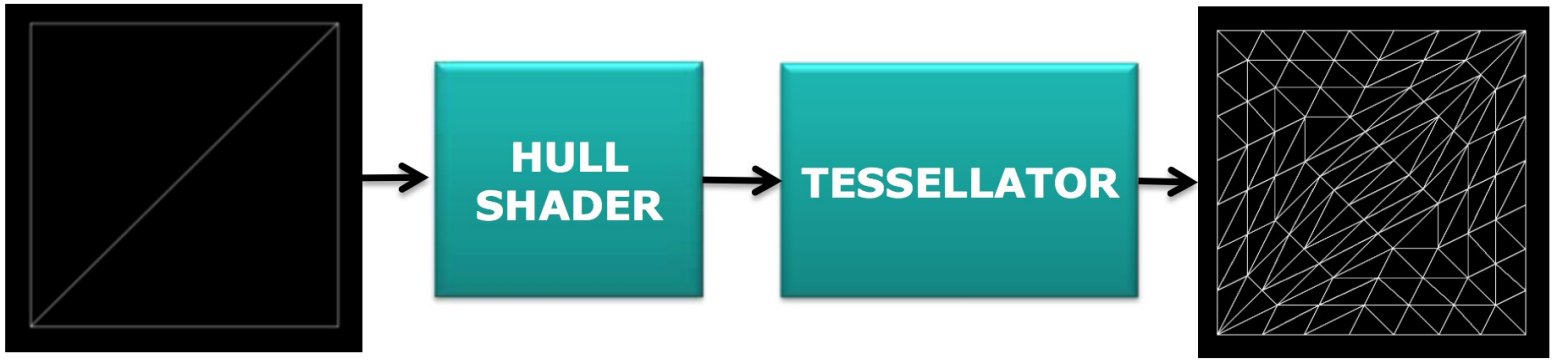
Генерирует дополнительные вершины внутри примитива.

Патч (patch)



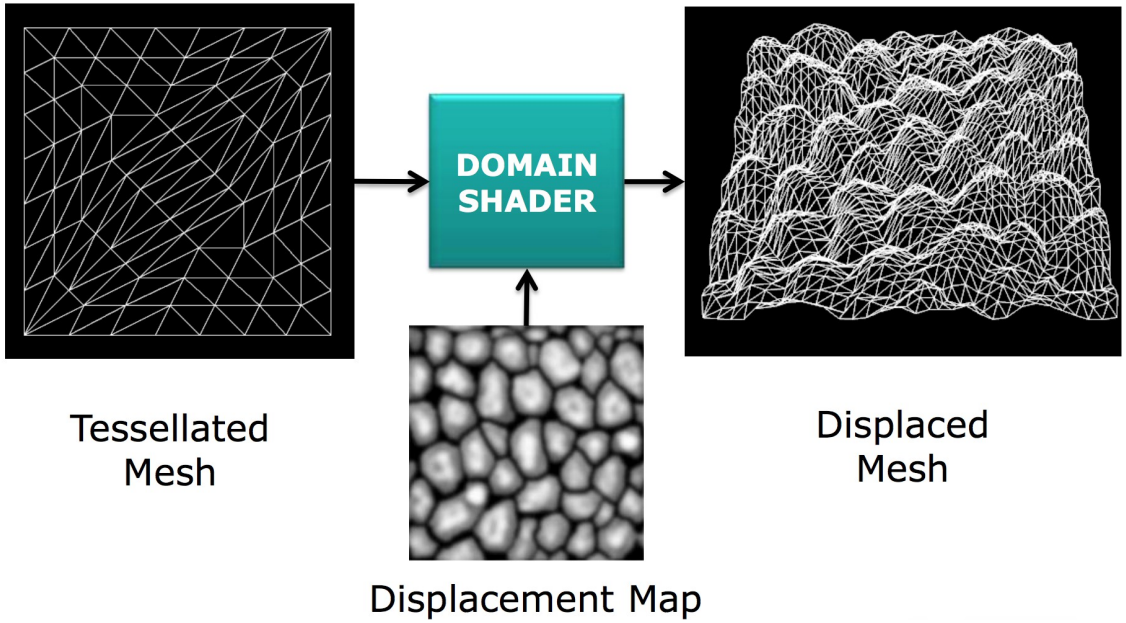
1. Tesselation Control Shader (Hull shader);
2. Аппаратный тесселятор;
3. Tesselation Evaluation Shader (Domain Shader).

Тесселятор: пример использования



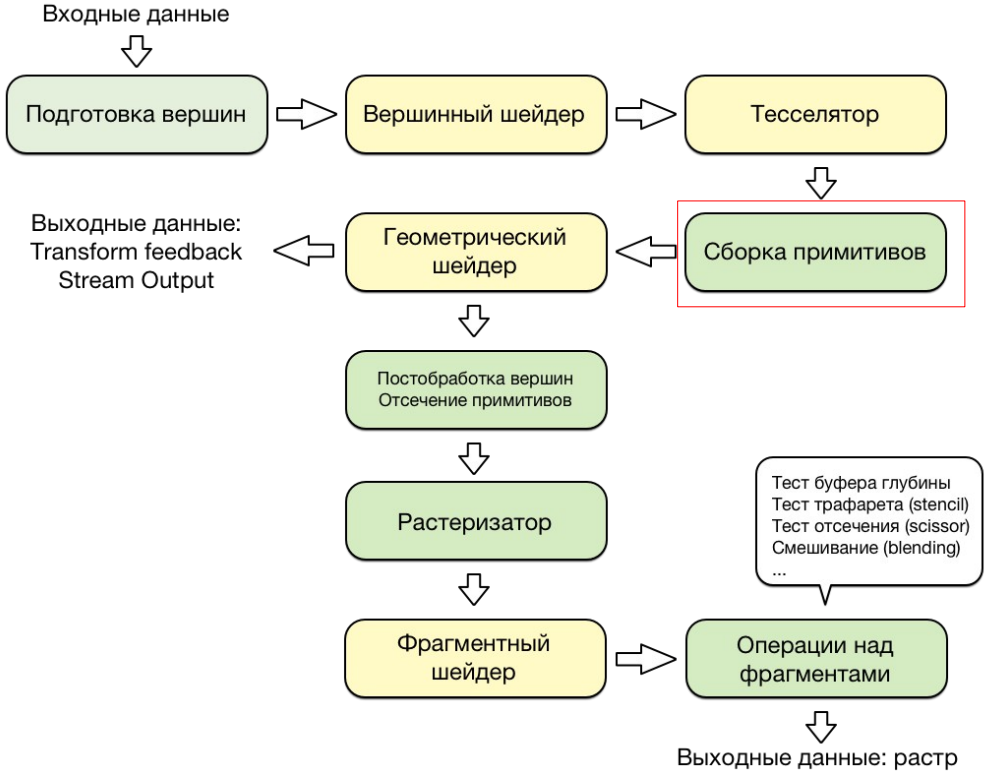
Презентация AMD «Direct3D 11 In-Depth Tutorial: Tessellation», Bill Bilodeau
http://www.gdcvault.com/play/1012740/direct3d

Тесселятор: пример использования

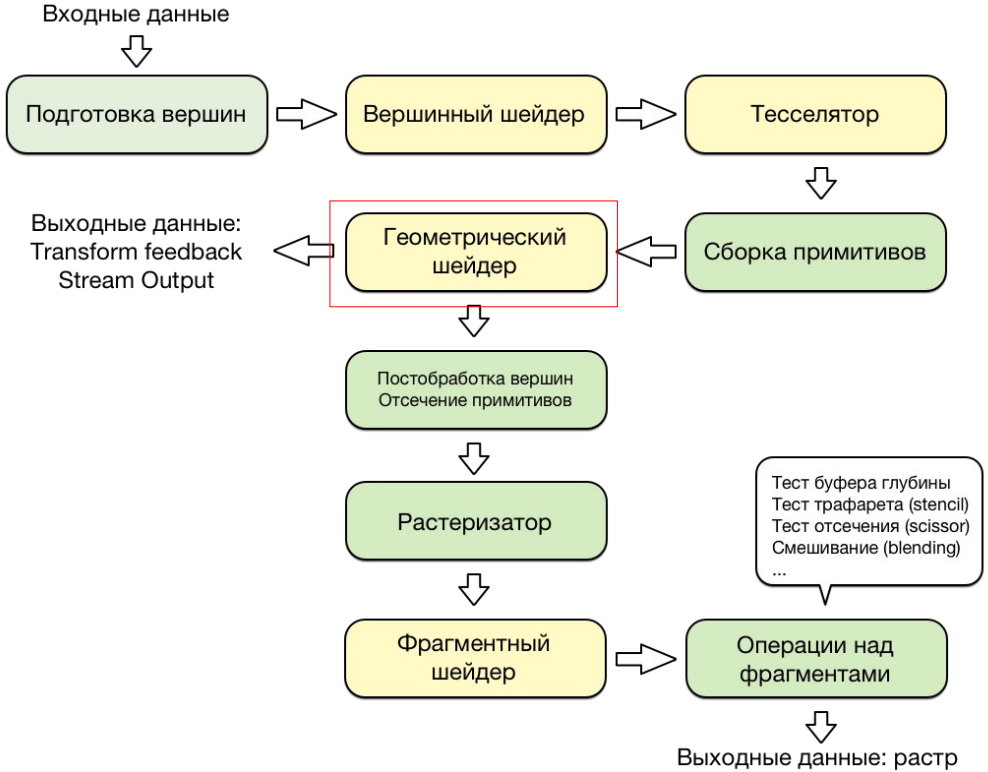


Презентация AMD «Direct3D 11 In-Depth Tutorial: Tessellation», Bill Bilodeau
http://www.gdcvault.com/play/1012740/direct3d

Графический конвейер



Графический конвейер



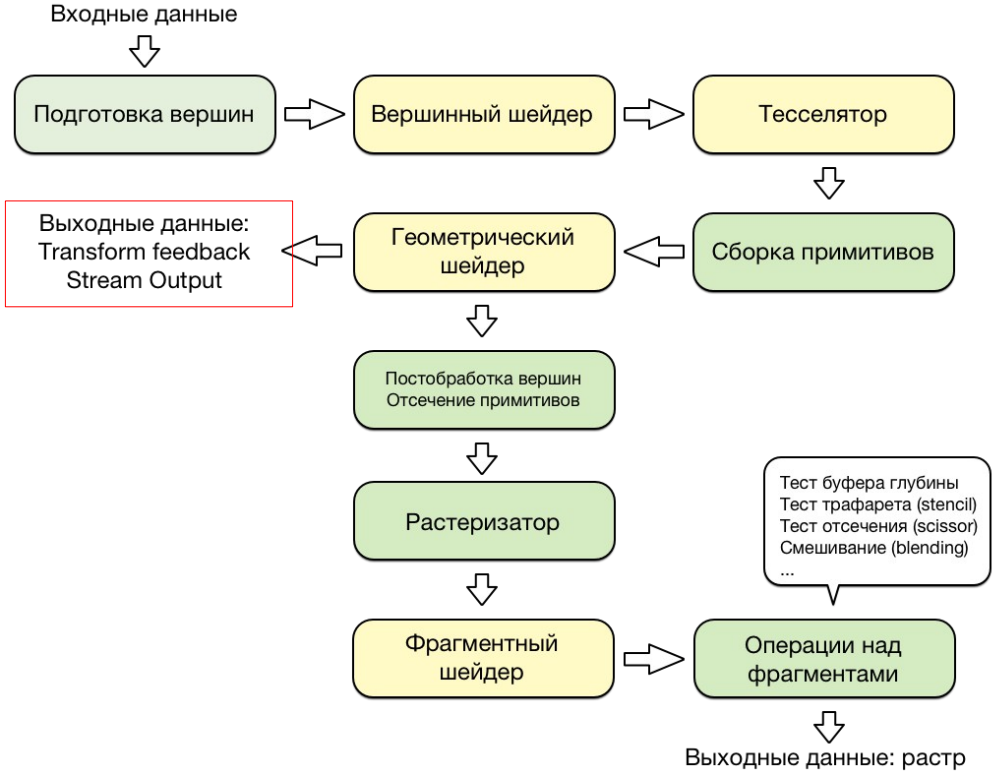
Геометрический шейдер

```
layout(points) in;
layout(triangle_strip, max_vertices = 3) out;

void main()
{
  gl_Position = vec3(-0.5, 0.5, 1.0, 1.0);
  EmitVertex();
  gl_Position = vec3(0.5, 0.5, 1.0, 1.0);
  EmitVertex();
  gl_Position = vec3(0.0, 0.0, 1.0, 1.0);
  EmitVertex();

  EndPrimitive();
}
```

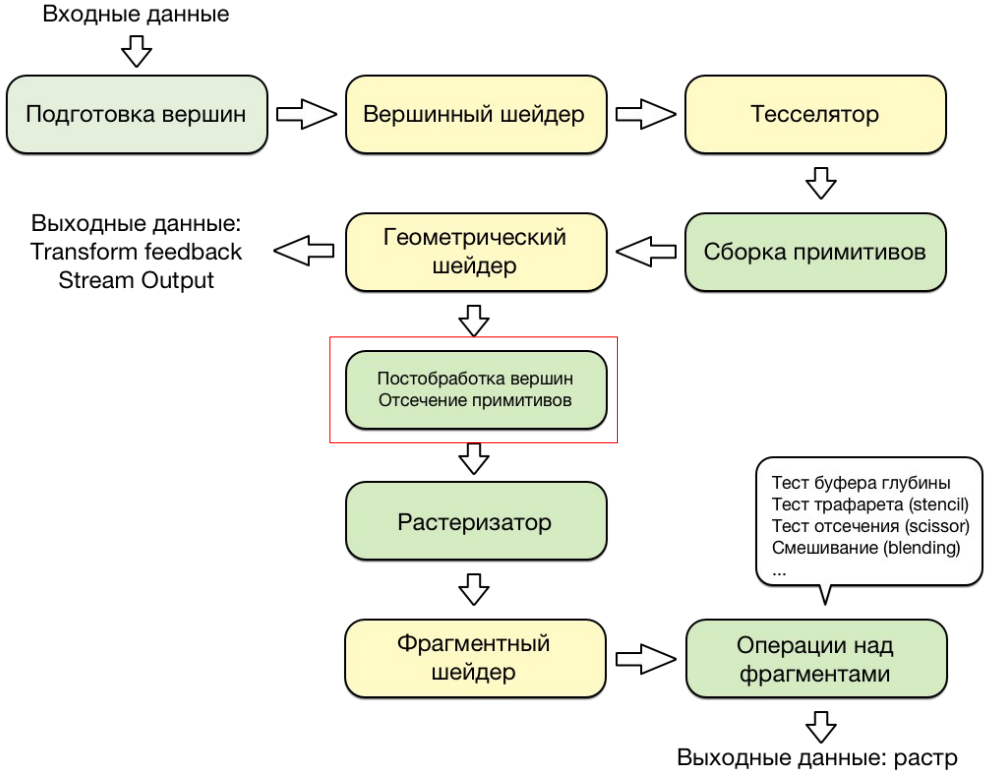
Тип входного примитива

Тип выходного примитива

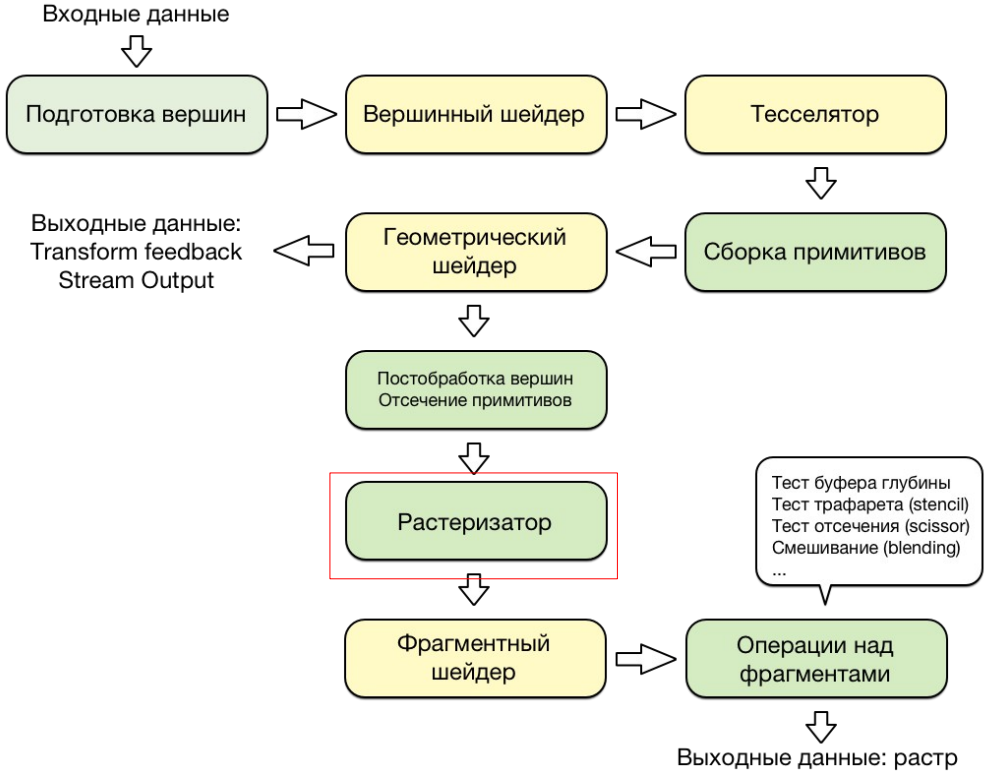
Графический конвейер



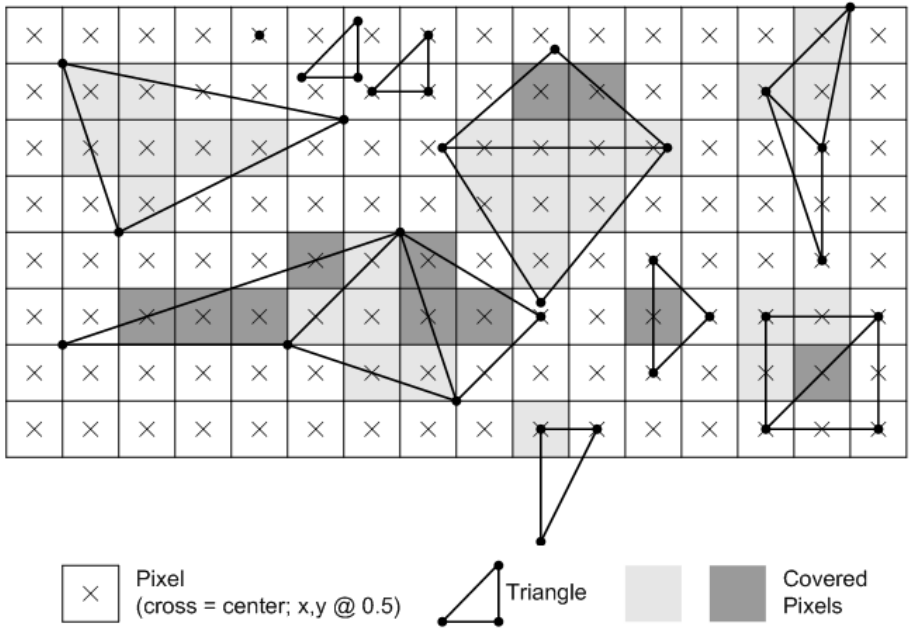
Графический конвейер



Графический конвейер

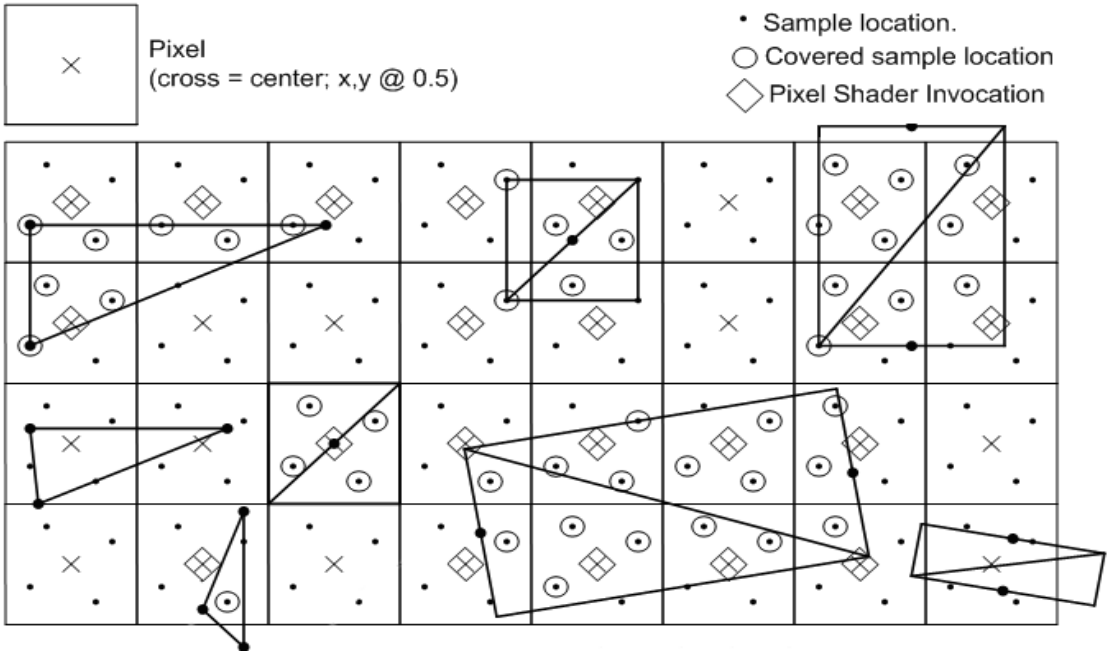


Растеризация



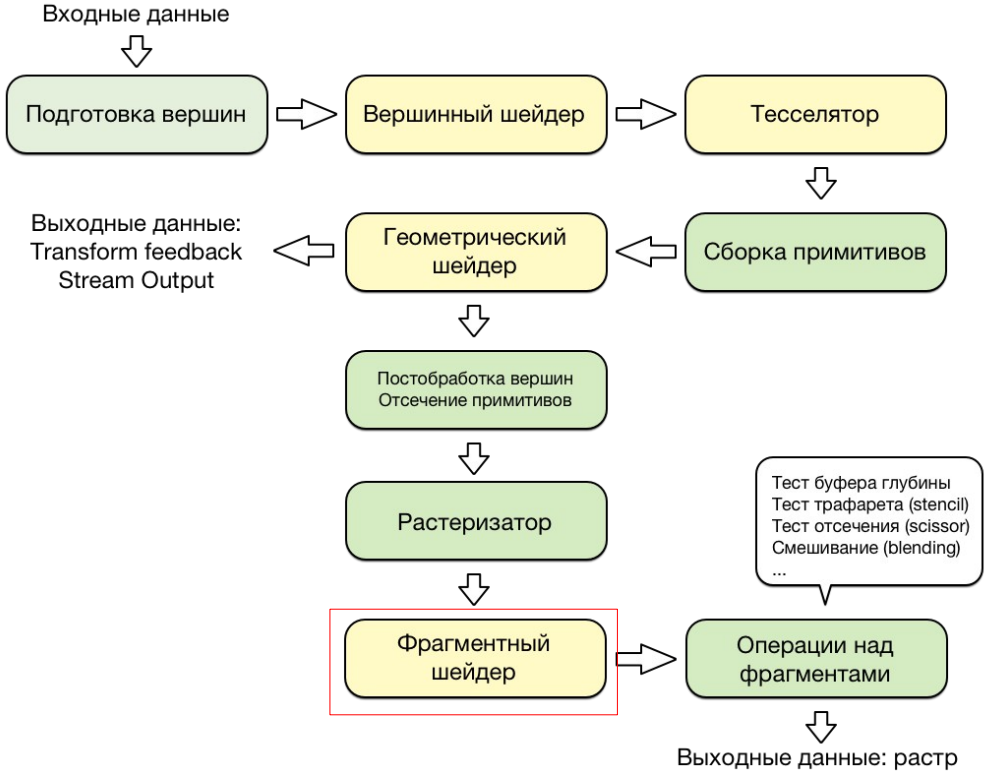
https://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx

Растеризация: антиалиасинг (MSAA)



https://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx

Графический конвейер



Фрагментный шейдер

Входные данные (интерполированные)

```
varying highp vec2 v_texCoord;
```

```
uniform sampler2D tex;
```

Текстура

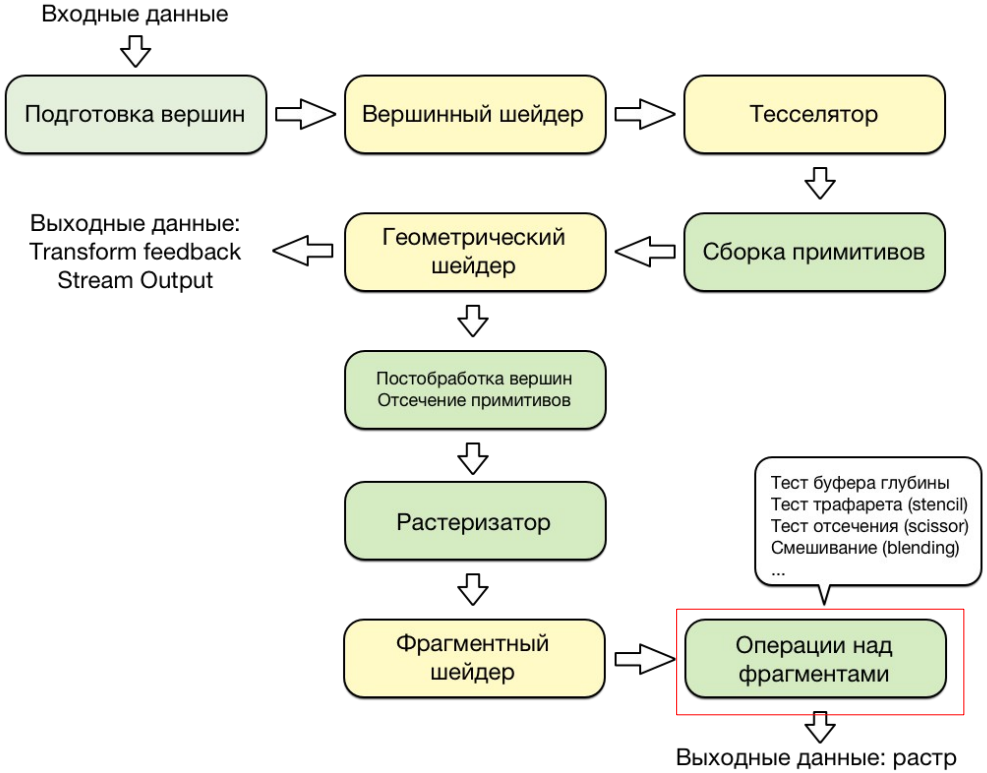
```
void main(void)
{
  highp vec4 color = texture2D(tex, v_texCoord);
  gl_FragColor = clamp(color, 0.0, 1.0);
};
```

Возвращает цвет

Фильтрация текстур

- Точечная;
- Билинейная;
- Трилинейная;
- Анизотропная.

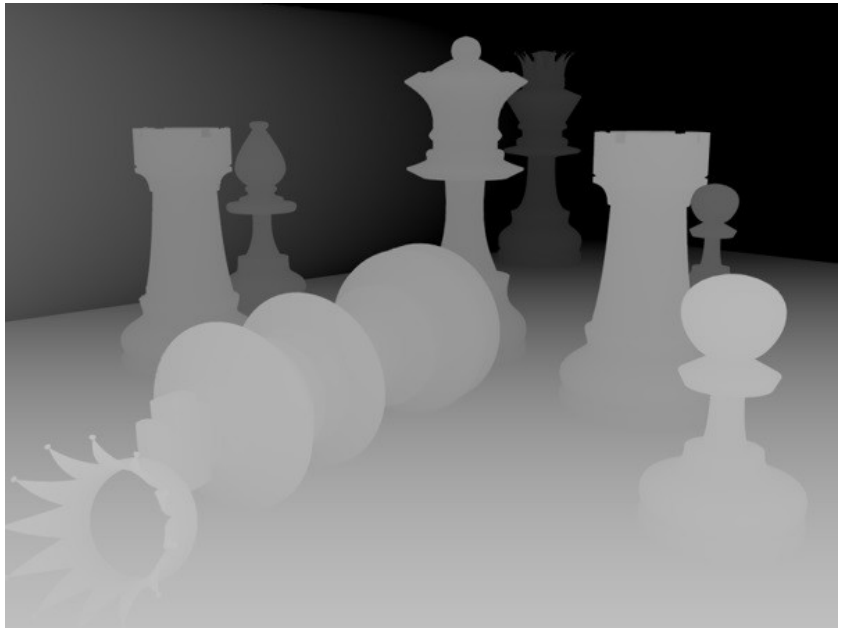
Графический конвейер



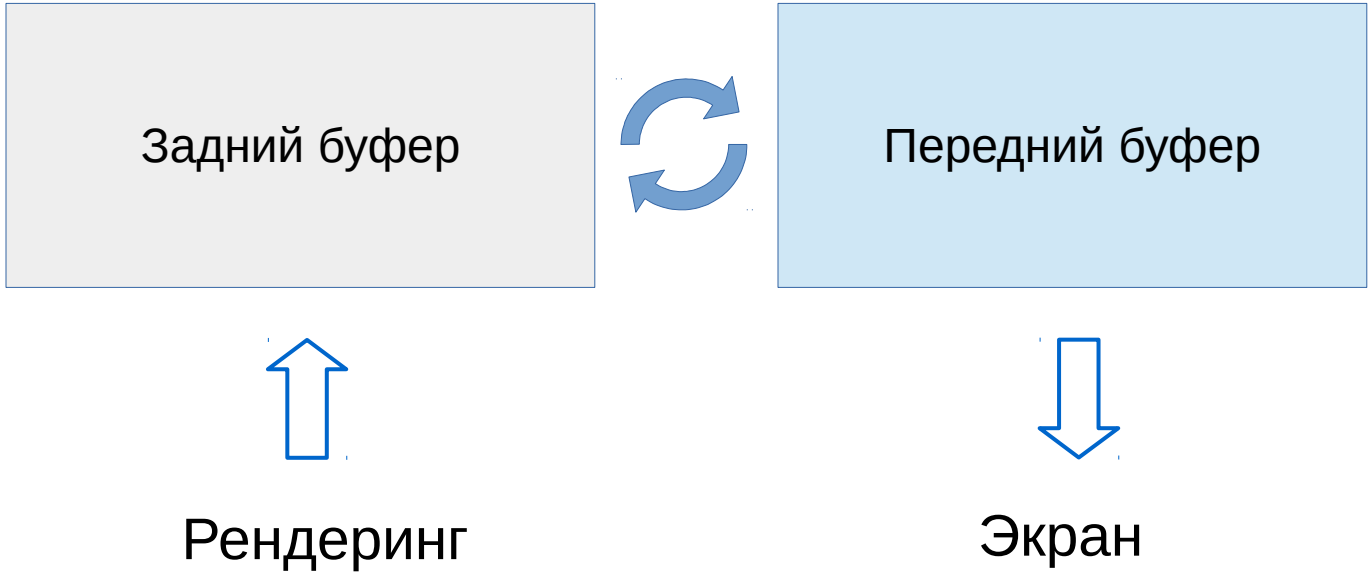
Буфер глубины

- Выполняется поиск минимума для каждого фрагмента по Z-координате в пространстве экрана;

- Может выполняться как до, так и после фрагментного шейдера.



Задний буфер (Back buffer)



Смешивание (blending)

Основное применение — реализация полупрозрачности

$$\begin{cases} \mathrm{out}_A = \mathrm{src}_A + \mathrm{dst}_A(1 - \mathrm{src}_A) \\ \mathrm{out}_{RGB} = \mathrm{src}_{RGB} + \mathrm{dst}_{RGB}\,(1 - \mathrm{src}_A) \end{cases}$$

OpenGL (Open Graphics Library)

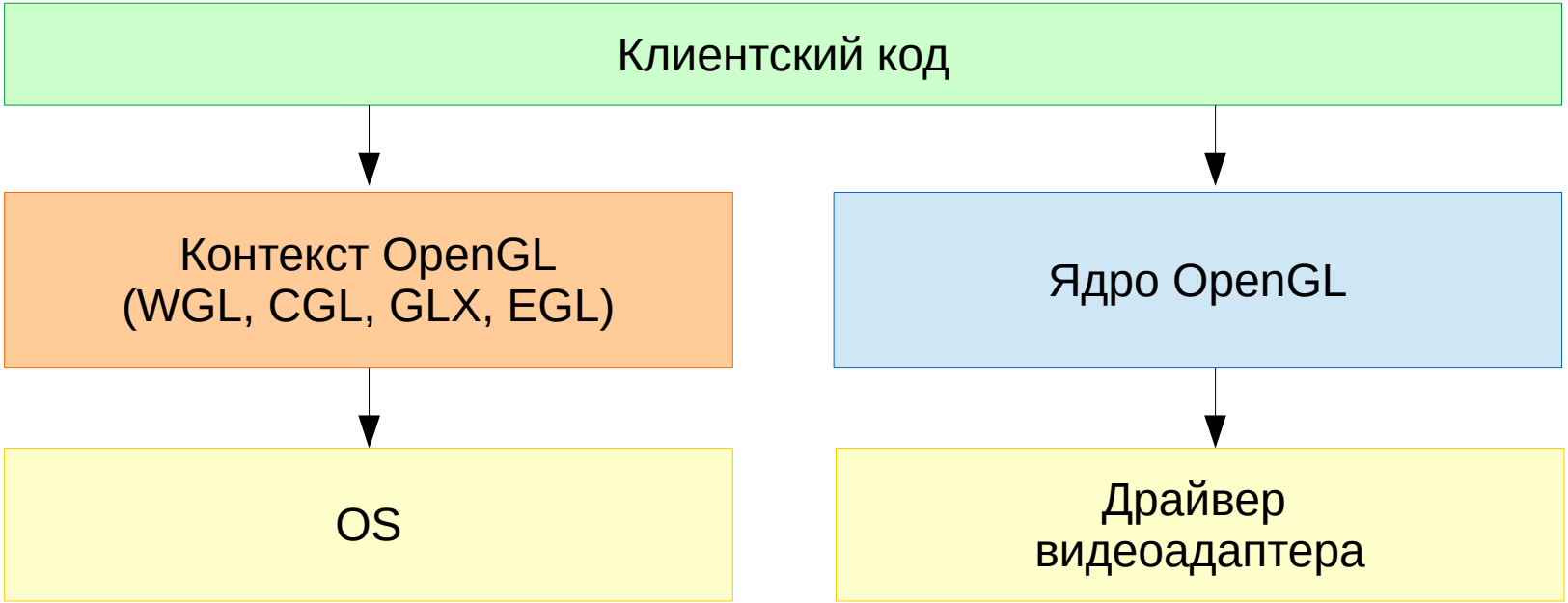
— спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Производители оборудования на основе этой спецификации создают реализации — библиотеки функций, соответствующих набору функций спецификации. Реализация призвана эффективно использовать возможности оборудования.

Основные преимущества OpenGL

- Кроссплатформенность (Windows, Linux, MacOS X, iOS, Android);
- Единый API (почти) для всех платформ;
- Механизм расширений;
- Комитет по стандартизации (Khronos Group).

Организация OpenGL



Core и Compatibility mode

```
glBindVertexArray(m_vertexArray);
glBindBuffer(GL_ARRAY_BUFFER, m_vertexBuffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_indexBuffer);
glDrawElements(GL_TRIANGLES, 12 /* indices count */,
                        GL_UNSIGNED_INT, (const GLvoid *)0);
```

```
glBegin(GL_POLYGON)
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

Не делайте так!

OpenGL 3+ (ES 2.0) API

- Функции в C-стиле с префиксом gl;
- Функции всегда обращаются к OpenGL-контексту текущего потока;
- Функции можно (а иногда нужно) импортировать самостоятельно;
- Функции зависят от контекста вызова (not pure).

Типы функций в OpenGL

- Порождающие (glGen*, glCreate*) и уничтожающие (glDelete*) функции;

- Функции, управляющими состоянием OpenGL (glBind*, glEnable, glActive*);

- Функции отрисовки (glDraw*);

- Прочие вспомогательные функции.

Объекты в OpenGL

1) **Буферы**

- Геометрические (VBO, VAO, Element Array);
- Пиксельные (PBO);

2) **Текстуры**

- 1D, 2D, 3D, кубмапы;
- Текстуры для рендеринга (FBO).

3) **Программы**

- Шейдеры.

Полезные библиотеки

- **GLFW** — современный аналог GLUT. Избавляет вас от работы с контекстами. http://www.glfw.org/;

- **Assimp** — загрузка 3D-моделей в различных форматах. https://github.com/assimp;

- **KTX** — загрузка текстур в форматах Khronos Group. https://www.khronos.org/opengles/sdk/tools/KTX/;

- Большой спектр библиотек по загрузке изображений (FreeImage, Stb Image, DevIL и т. д.).

OpenGL и Qt

Qt реализует полноценную поддержку OpenGL с возможностью интеграции рендеринга в оконную систему.

Класс **QOpenGLWidget** инкапсулирует всю работу с OpenGL-контекстом и по присоединению к оконной системе.

Класс **QOpenGLFunctions** предоставляет доступ к функциям OpenGL.

OpenGL и Qt

Смотрим код

https://github.com/rokuz/template/tree/window_demo

Домашнее задание

1) Реализовать мерцающие звезды в космосе.
   - Можно использовать класс TexturedRect как основу;
   - Звезды создаются в случайных местах экрана;
   - Мерцание звезды можно реализовать при помощи изменения прозрачности текстуры по времени (по функции синуса, например).

Срок сдачи: 21.11.2016 23:59:59

ТЕХНОАТОМ

Просьба оставить отзыв о данном занятии на портале!

Спасибо за внимание!

Роман Кузнецов

r.kuznetsov@mapswithme.com