

Лекция №8

Шейдеры

Рябинин Константин Валентинович

e-mail: icosaeder@ya.ru

jabber: icosaeder@jabber.ru

Пермь, 2011

Шейдер – это микропрограмма для одной из ступеней графического конвейера, используемая для определения окончательных параметров объекта или изображения

Свойства:

- Автономен, не является частью кода приложения
- На современном оборудовании выполняется на видеокарте (аппаратная поддержка)
- Написан на специфическом процедурном языке
- Предназначен для многократного вызова, но без использования многопоточности в явном виде
- Выполняет лишь узкоспециализированную задачу определения конкретных параметров объекта или изображения

Назначение шейдеров:

- Создание визуальных эффектов
 - Превращение графического конвейера из неуправляемого в управляемый
 - Значительное увеличение свободы управления результатом визуализации
- Унификация механизма создания визуальных эффектов любой сложности

Достоинства шейдеров:

- Очень высокая эффективность
 - Свобода создания визуальных эффектов
 - Децентрализация кода
 - Межпроектное переиспользование
- В современной практике шейдеры составляют основу всех визуальных эффектов, без их применения не обходится ни одна мультимедийная система

Наиболее частые эффекты, создаваемые при помощи шейдеров:

- **Моделирование реалистичных материалов**
 - Дерево, металл, пластмасса, . . .
- **Моделирование природный явлений**
 - Вода, огонь, дым, . . .
- **Процедурное текстурирование**
 - Различные узоры, как регулярные, так и не регулярные
- **Нефотореалистичный рендеринг**
- **Анимация формы**
 - Процедурное движение, интерполяция между ключевыми кадрами, системы частиц, . . .
- **Сложное текстурирование**
 - Мультитекстурирование, анимации текстуры, . . .
- **Сложное освещение**
 - Глобальные модели освещения, тени, каустика
- **Оптические эффекты**
 - Отражение, преломление, дифракция, дисперсия
- **Сглаживание изображения**

- **Геометрический шейдер** – микропрограмма, обрабатывающая за раз **один геометрический примитив**
 - Задача: определить положение и цвета примитива
- **Вершинный шейдер** – микропрограмма, обрабатывающая за раз **одну вершину**
 - Задача: определить положение и цвет вершины
- **Пиксельный (фрагментный) шейдер** – микропрограмма, обрабатывающая за раз **один «фрагмент» изображения**, то есть его атомарную часть (чаще всего атомарной частью изображения выступает пиксель)
 - Задача: определить цвет фрагмента

По сути шейдер – общее название для семейства специализированных микропрограмм

Для написания шейдеров используются специализированные языки программирования, характеризующиеся:

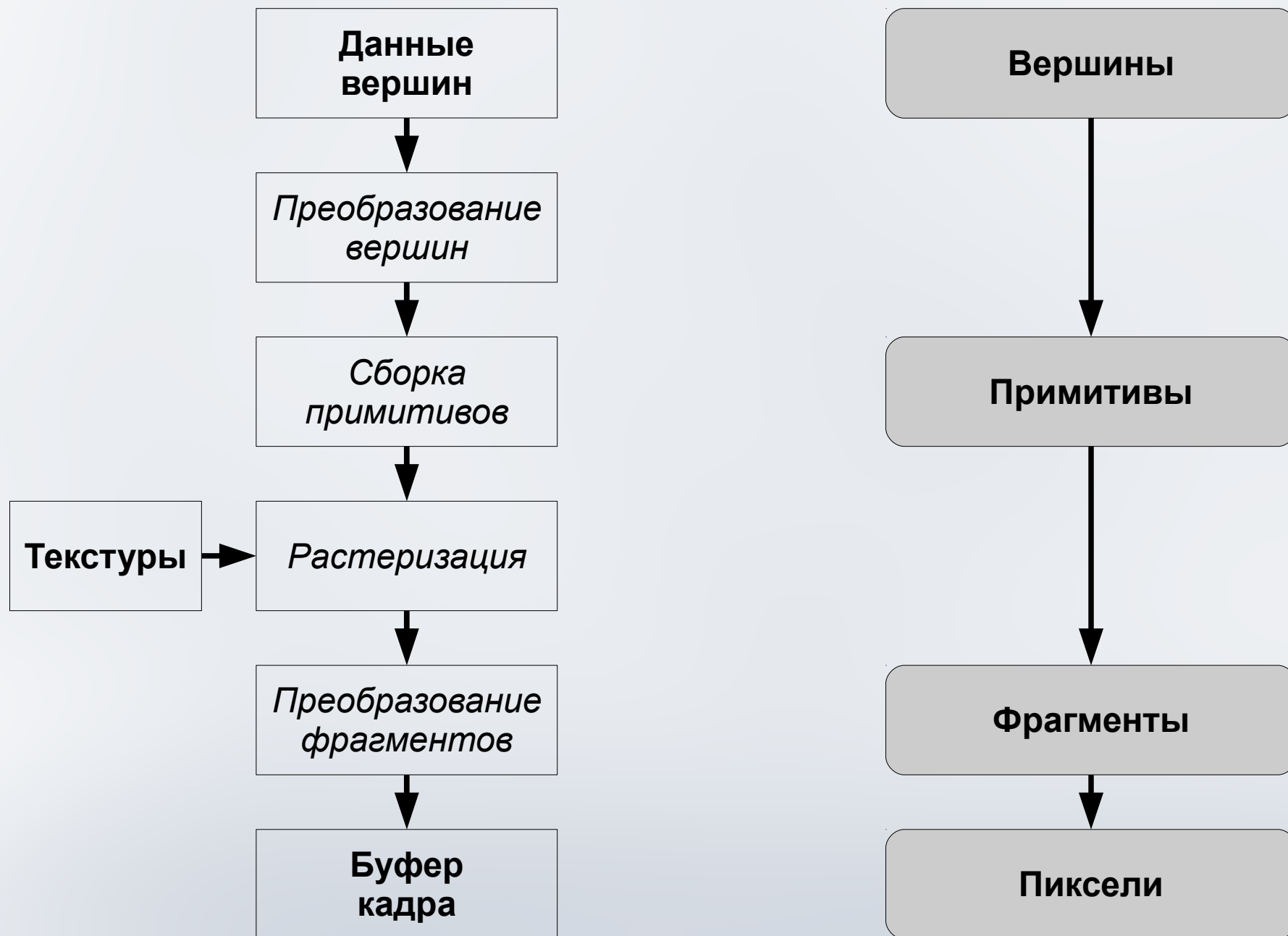
- **Процедурной парадигмой**
- **Тьюринг-полнотой**
- **Наличием специализированных типов данных и встроенных функций для работы с обрабатываемыми сущностями**
- **Как правило, за основу шейдерных языков берётся синтаксис C**

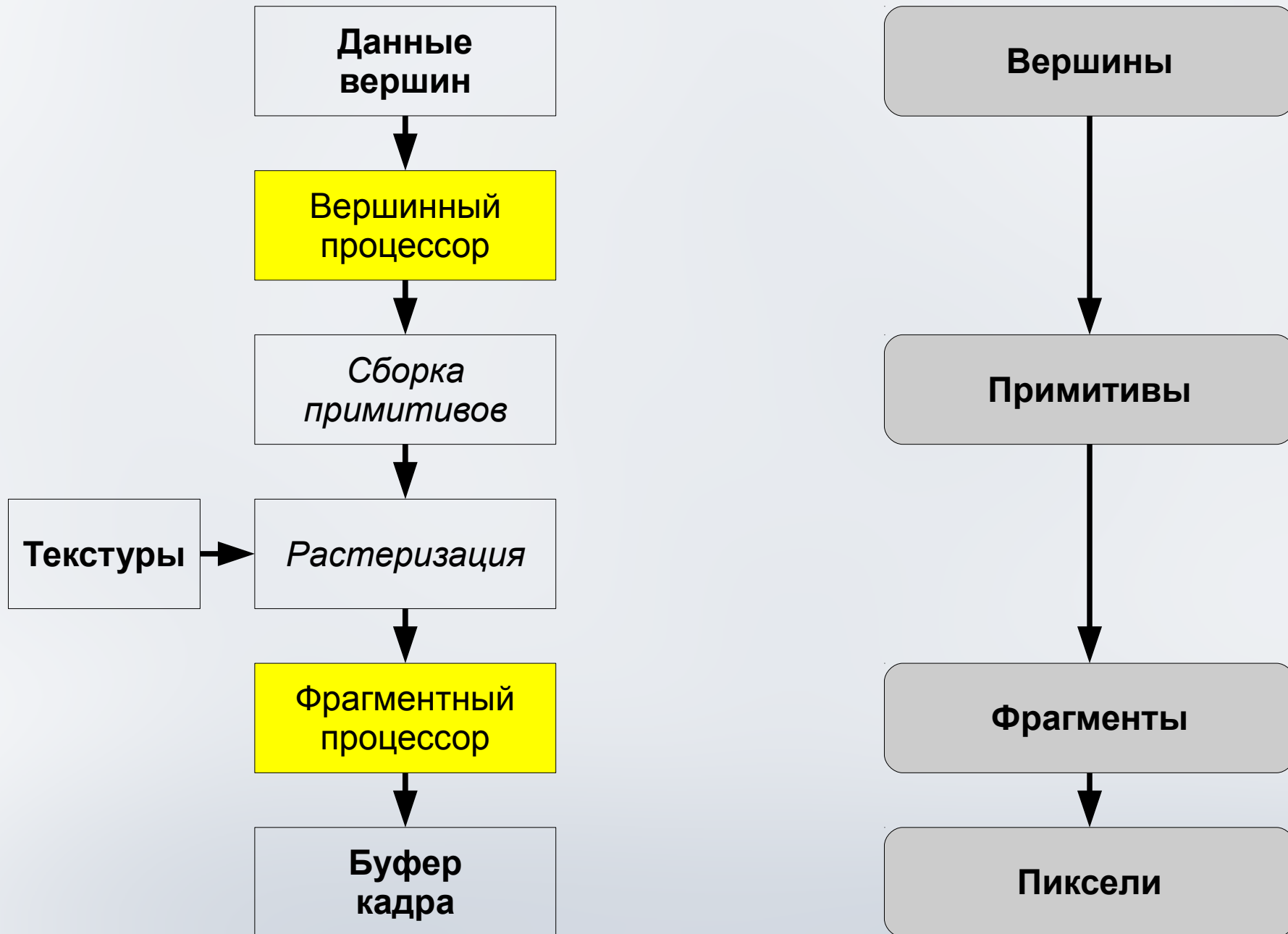
Примеры языков:

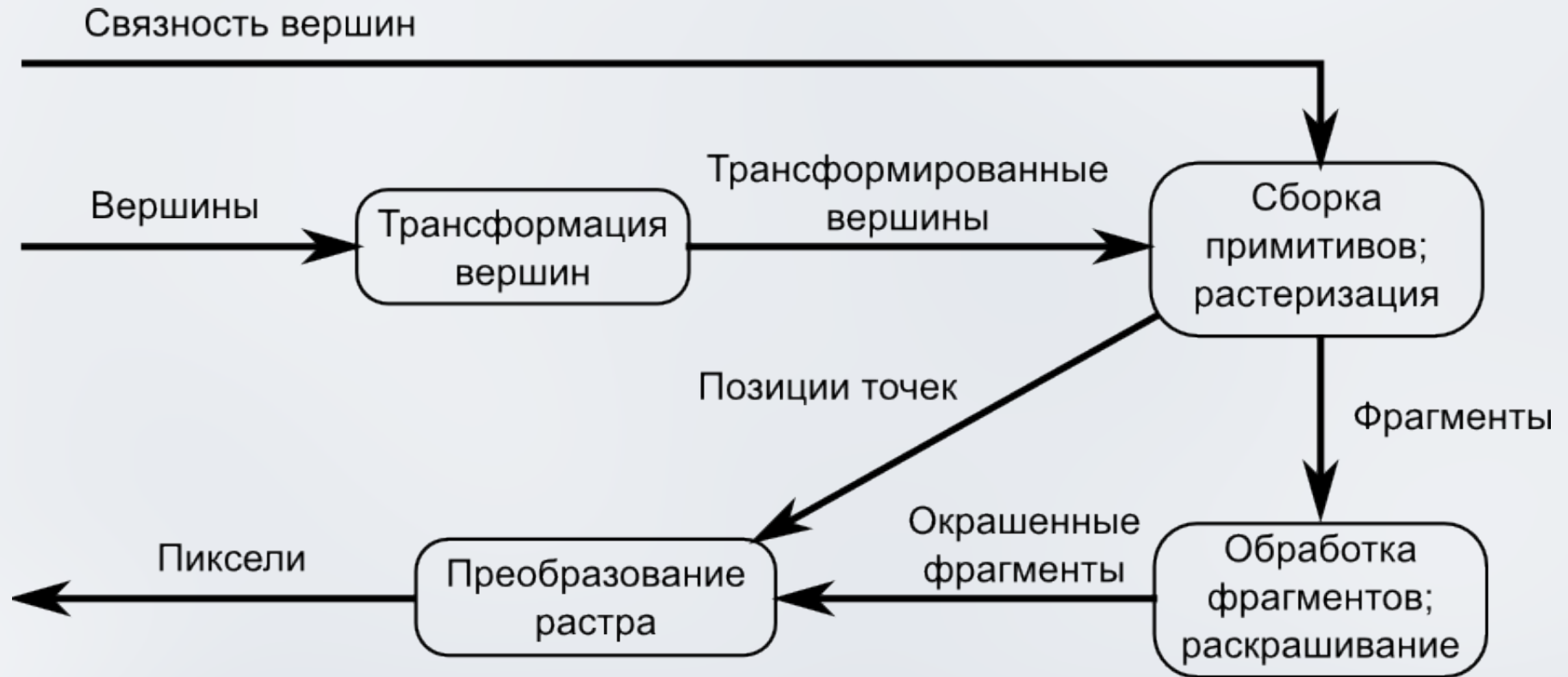
- GLSL (Graphics Library Shader Language)
 - Шейдерный язык от ARB
- HLSL (High-Level Shader Language)
 - Шейдерный язык от Microsoft
- DirectX ASM
 - Шейдерный ассемблер от Microsoft
- Cg (C for Graphics)
 - Шейдерный язык от nVidia для Microsoft
- RenderMan
 - Шейдерный язык от Pixar для художников
- Gelato
 - Шейдерный язык от nVidia для художников

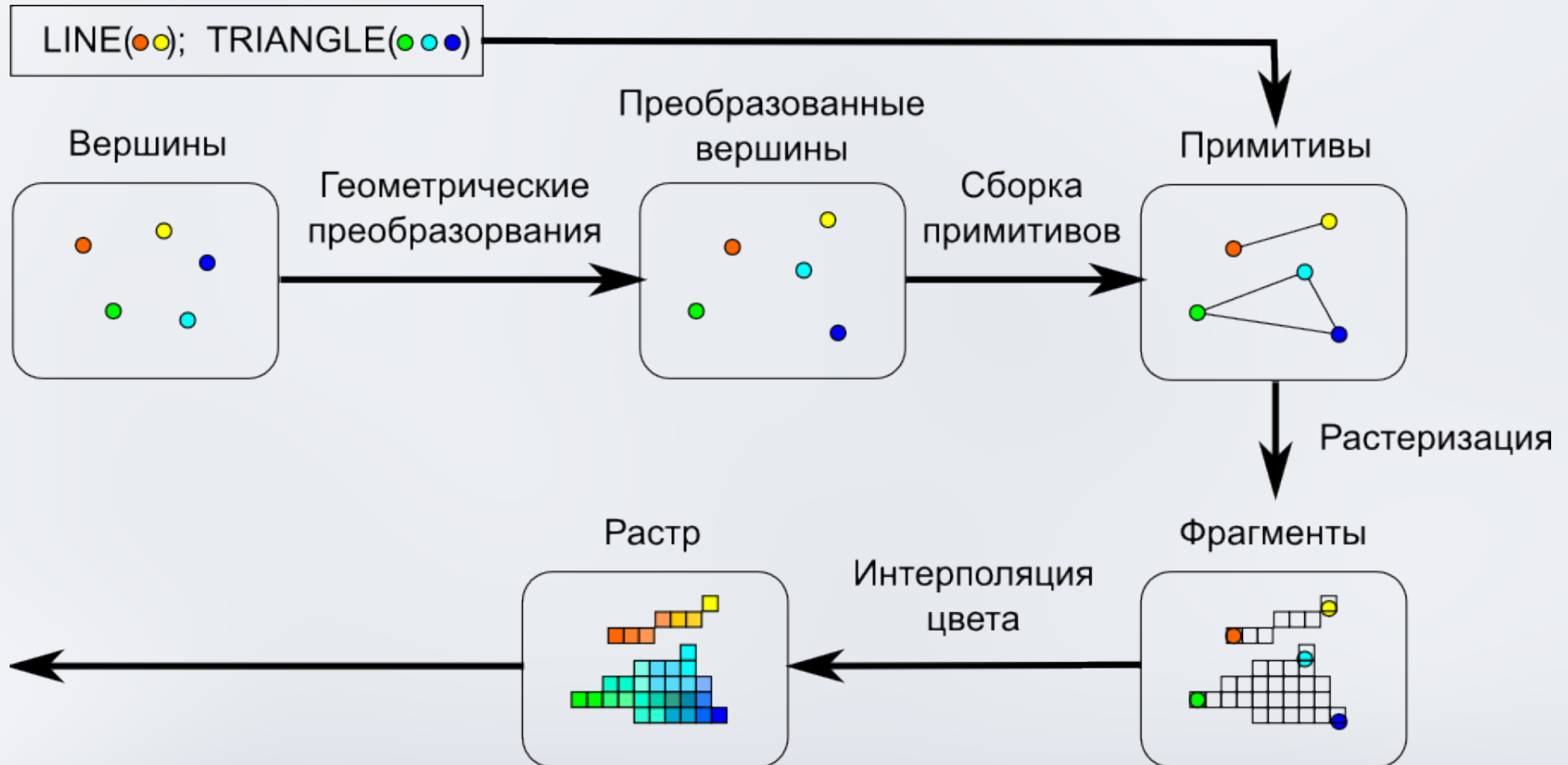
- Загрузка из файла (или из строковой константы)
- Компиляция («на лету», во время выполнения основной программы)
- Встраивание в конвейер («активация»)
- Множественное выполнение (для каждой обрабатываемой данной ступенью конвейера сущности за один рендеринг шейдер выполняется ровно один раз)
- Отсоединение от конвейера («деактивация»)
- Удаление из памяти

- Помимо данных об обрабатываемой сущности шейдеры могут получать из основной программы произвольные наборы данных
 - «глобальных» (одинаковых для всех обрабатываемых сущностей)
 - «локальных» (сцепленных с сущностью, как правило при помощи массивов, индексы в которых соответствуют номерам сущностей)
- Шейдеры более раннего этапа конвейера могут подготавливать и передавать параметры шейдерам более позднего этапа









Вершинный процессор – это программируемый модуль, обрабатывающий вершины и связанные с ними данные

Выполняемые операции:

- Преобразование вершин и нормалей
- Генерация и преобразование текстурных координат
- Расчёт цвета вершин (с учётом освещения и других произвольных параметров)
- Программа для вершинного процессора называется вершинным шейдером
- Вершинный шейдер готовит данные для фрагментного процессора

Вершинный процессор – это программируемый модуль, обрабатывающий вершины и связанные с ними данные

Входные и выходные данные:

- **Переменные-атрибуты** (*attribute*) – передаются вершинному шейдеру от приложения для описания свойств каждой вершины
- **Однообразные переменные** (*uniform*) – используются для передачи данных как вершинному, так и фрагментному процессору. Не могут меняться чаще, чем один раз за полигон
- **Разнообразные переменные** (*varying*) – служат для передачи данных от вершинного к фрагментному процессору. Данные переменные могут быть различными для разных вершин, и для каждого фрагмента будет выполняться интерполяция

Фрагментный процессор – это программируемый модуль, обрабатывающий фрагменты (пиксели) и связанные с ними данные

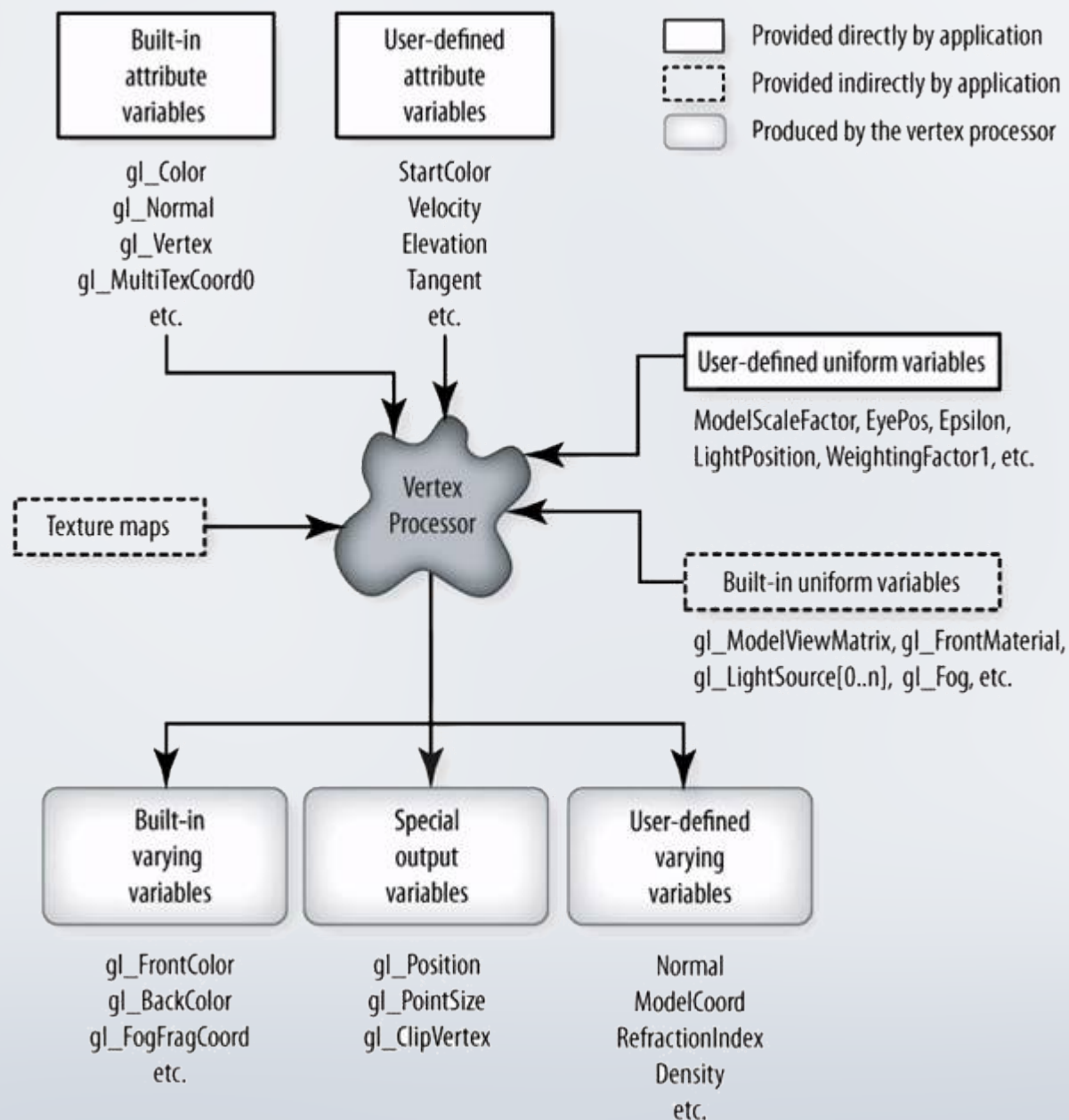
Выполняемые операции:

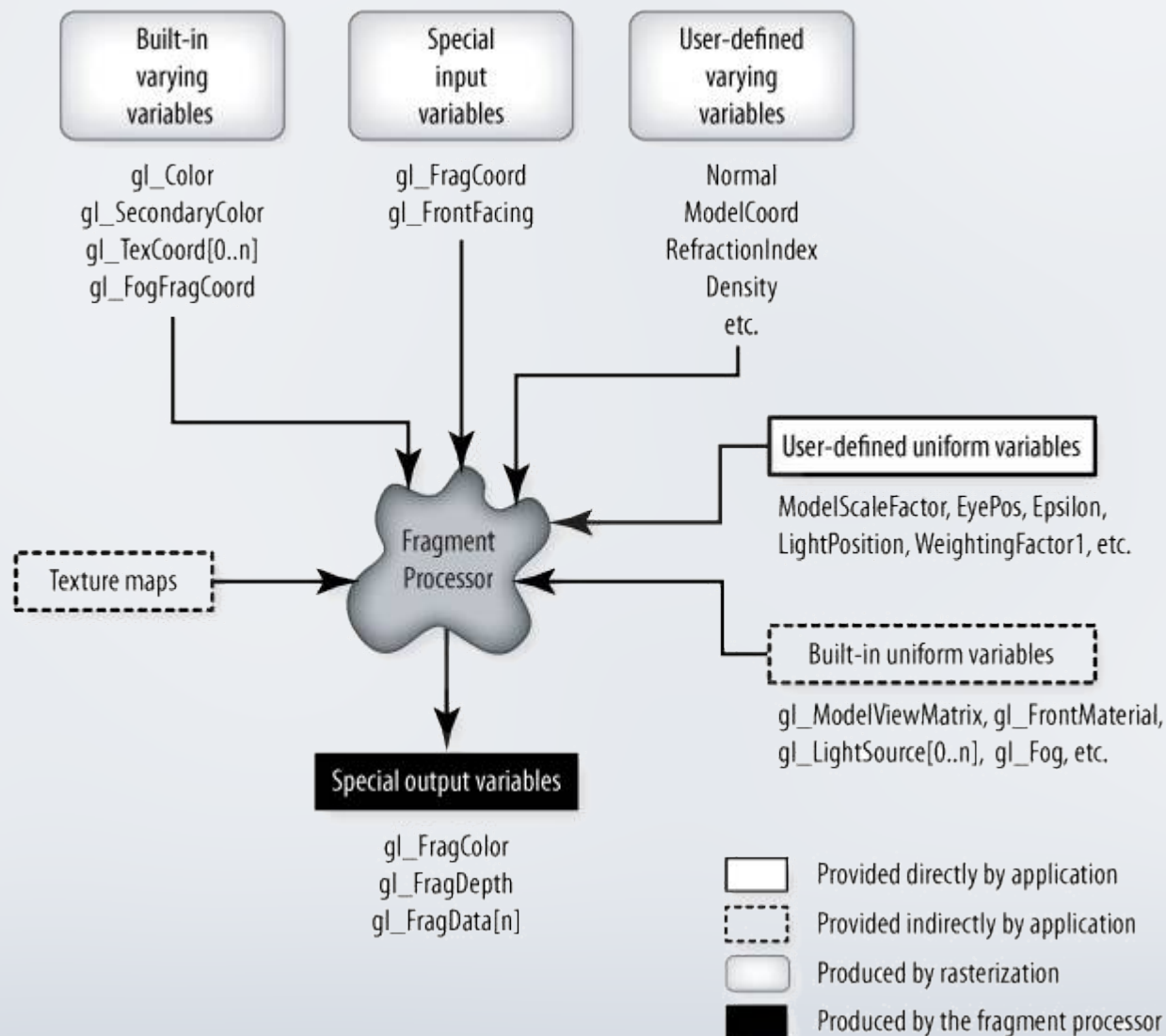
- Операции над интерполированными значениями, полученными от вершинного процессора
 - Наложение текстур
 - Расчёт цвета фрагментов (с учётом освещения и других произвольных параметров)
 - Координаты фрагмента есть константа
- Программа для фрагментного процессора называется фрагментным шейдером

Фрагментный процессор – это программируемый модуль, обрабатывающий фрагменты (пиксели) и связанные с ними данные

Входные и выходные данные:

- **Разнообразные переменные (*varying*)** – приходят от вершинного шейдера; как встроенные, так и определенные разработчиком
- **Однообразные переменные (*uniform*)** – для передачи произвольных относительно редко меняющихся параметров





Шейдерная программа – это специализированная структура данных в OpenGL, предназначенная для хранения нескольких шейдеров разных типов для их одновременного использования

- По факту шейдерная программа используется для связки **ровно двух шейдеров** – одного вершинного и одного фрагментного, так как программируемый конвейер в каждый момент времени нуждается ровно в двух шейдерах
- Теоретически возможна сборка шейдера из нескольких частей, но на практике это почти не используется
- В каждый момент времени может быть активна только одна шейдерная программа
- Шейдерная программа – единственный способ использования шейдеров в OpenGL, поэтому **шейдеры всегда существуют парами** (каждому вершинному должен соответствовать его фрагментный)
- Порядок вызовов:
`glCreateProgram → glAttachShader → glAttachShader → glLinkProgram → glUseProgram`

GLSL – это высокоуровневый процедурный язык программирования для вершинного и фрагментного процессоров OpenGL

Характеристика

- Программы на GLSL представляют собой абстракцию от аппаратного обеспечения
- Компилятор GLSL является частью библиотеки стандарта OpenGL и генерирует оптимизированный под конкретную видеокарту код
- GLSL основан на синтаксисе C и является чисто процедурным
- Начало выполнения программы – функция `void main()`
- Спецификация языка (от 02.09.2011):
<http://www.opengl.org/registry/doc/GLSLangSpec.4.20.8.clean.pdf>

● Для типов существуют квалификаторы точности

- lowp – низкая точность
- mediump – средняя точность
- highp – высокая точность

● Скалярные (базовые типы)

● float – вещественное число:

- lowp float: $[-2; 2]$ mediump float: $[-2^{14}; 2^{14}]$ highp float: $[-2^{62}; 2^{62}]$

● int – целое число

- lowp int: $[-2^8; 2^8]$ mediump int: $[-2^{10}; 2^{10}]$ highp int: $[-2^{16}; 2^{16}]$

● bool – логическое значение

● Особенности:

● Отсутствие неявных приведений типа:

```
float a = 1; // порождает ошибку  
float b = 1.0, c = float(1);
```

● Тип int не всегда поддерживается аппаратно (в общем случае – обёртка над float), поэтому результат переполнения, вообще говоря, не определён

● Отсутствуют побитовые операции

● Тип bool – обёртка над int, то есть, в общем случае, так же обёртка над float

Векторы

- `vec2`, `vec3`, `vec4` – вещественные вектора на 2, 3 и 4 компоненты
- `ivec2`, `ivec3`, `ivec4` – целочисленные вектора на 2, 3 и 4 компоненты
- `bvec2`, `bvec3`, `bvec4` – логические вектора на 2, 3 и 4 компоненты

Особенности:

- Реализована перегрузка операций сложения и вычитания векторов, причём код оптимизируется и выполняется GPU за $O(1)$
- Инициализация вектора может быть осуществлена при помощи конструкторов вида
`vec3 a = vec3(0.1, 0.2, 0.3);`
`vec4 b = vec4(a, 0.4);`
`vec2 c = vec2(a);` // будут взяты первые две компоненты
- Для доступа к компонентам можно использовать индекс:
`vec3 a; a[1] = 0.5;`
либо мнемонические поля (`x, y, z, w ~ r, g, b, a ~ s, t, p, q`):
`vec3 a; a.y = 0.5;`
- Существуют вспомогательные поля, предоставляющие доступ к любому подмножеству значений в любой последовательности:
`vec3 a, b;`
`a.xy = b.zy = vec2(0.5, 0.8);`
`vec3 c = vec3(0.9, a.xy);`
- Мнемоника полей существует лишь для пользователя, представляя собой обёртку доступа к данным

● Матрицы

● `mat2` – вещественная матрица 2x2

● `mat3` – вещественная матрица 3x3

● `mat4` – вещественная матрица 4x4

● Особенности:

● Реализована перегрузка операций сложения, вычитания и умножения матриц

● Реализована перегрузка операции умножения матрицы на вектор

● Матрица хранится по столбцам и могут быть рассмотрены как массив векторов-столбцов

● Как правило, матрицы приходят в шейдер из основной программы и используются для произведения аффинных преобразований

● Дискретизаторы – специализированные структуры данных для доступа к текстурам

- `sampler1D` – предоставляет доступ к одномерной текстуре
- `sampler2D` – предоставляет доступ к двумерной текстуре
- `sampler3D` – предоставляет доступ к трехмерной текстуре
- `samplerCube` – предоставляет доступ к кубической текстуре

● Особенности:

- Дискретизатор приходит в шейдер извне через *uniform*-переменную и не может быть изменён внутри шейдера
- Используется для доступа к текстуре
- Для извлечения данных из дискретизатора используются специализированные функции, например:

```
// fragment shader
uniform sampler2D tex;
void main()
{
    vec4 color = texture2D(tex, gl_TexCoord[0].st);
    gl_FragColor = color;
}
```

● Структуры

● `struct Light { vec3 position; vec3 color; };`

● Особенности:

- Структуры, фактически, полностью идентичны структурам в C
- *union* и *enum* зарезервированы в качестве ключевых слов, но пока не поддерживаются

● Массивы

● `float a[10];`
`vec4 points[5];`

● Особенности:

- Можно объявлять массивы любых типов
- Массивы являются статическими

● Void – тип для функций, не возвращающих значения

→ Более никаких типов в GLSL нет; динамическое выделение памяти (указатели) не поддерживается; строки и абстрактные типы не предусмотрены

- Перегрузка операций для векторных и матричных типов данных
- Функции над векторами:
 - `dot` – скалярное произведение
 - `normalize` – нормирование вектора
 - `reflect` – отражение вектора относительно вектора
 - `refract` – преломление вектора относительно вектора с коэффициентом преломления
 - `length` – длина вектора
 - `distance` – расстояние между двумя точками
 - ...

- **Функции над матрицами:**
 - **determinant** – определитель матрицы
 - **transpose** – транспонированная матрица
 - **inverse** – обратная матрица
 - ...
- **Тригонометрические функции**
 - **sin, cos, tan** – функции
 - **asin, acos, atan** – аркфункции
 - **radians, degrees** – перевод из градусов в радианы и обратно
 - ...
- **Гиперфункции**
 - **sinh, cosh, tanh** – функции
 - **asinh, acosh, atanh** – аркфункции
 - ...

● Математические функции

- `pow` – возведение произвольную в степень
- `exp` – экспонента
- `log` – натуральный логарифм
- `sqrt` – квадратный корень
- `clamp` – ограничение значения
- `abs` – модуль
- `ceil`, `floor`, `round` – округление в разные стороны
- `sign` – сигнум
- `min`, `max` – минимум, максимум
- ...
- ...