# Введение в GLSL

**Наталья Татарчук**
**ATI Research, Inc**
**3D Application Research Group**

# Обзор Материала

- Введение в GLSL
- Обзор структуры языка
- Примеры шейдеров в GLSL

# GLSL – Новый Стандарт для Программирования Шейдеров в OpenGL

- Поддерживается драйверами с выпуском OpenGL 1.5
- Позволяет разработчикам использовать программируемые конвейеры с использованием OpenGL API
- Язык программирования шейдеров высокого уровня
- Стандартное расширение ARB

# Обзор Языка GLSL

- Базируется на основе ANSI C
  - Сохранено большинство возможностей языка, за исключением тех случаев, когда страдает производительность
- Добавлена поддержка векторов и матрих
- Заимстованы некоторые возможности C++
  - Фунции, перегруженные по аргументам вызова (Overloaded functions)
  - Возможность декларирования переменных в любом месте по мере надобности

# Обзор Языка GLSL (продолж.)

- GLSL является набором двух схожих языков для программирования VPU
  - Программирование вершинного процессора
  - Программирование фрагментного процессора
- Большая часть функциональных возможностей одинакова

# Обзор Вершинного Процессора

- Конвейер, оперирующий вершинными данными
- Используется для:
  - Трансформации вершин
  - Трансформации нормалей и их нормализации
  - Генерации текстурных координат
  - Рассчета освещения

- *Вершинные шейдеры* – это программы, запускаемые на вершинном процессоре
  - Программа знает тольно об одной вершине

# Обзор Фрагментного Процессора

- Конвейер, обрабатывающий интерполируемые данные для:
  - Наложения текстур
  - По-фрагментного рассчета цвета материала
- *Фрагментный шейдер* - это программа для фрагментного процессора, работающая с одним фрагментом
  - С помощью FS можно симулировать FFP полностью, но нельзя их использовать одновременно
  - Некоторые части FFP нельзя заменить с FS
    - Alpha and depth test
    - Scissor
    - Stencil test
    - Alpha blending
  - Знает текущее состояние конвейера OpenGL

# Функциональные Возможности Языка GLSL

- Директивы препроцессора
- Типы данных
- Операторы и выражения
- Структура языка
- Встроенные переменные и типы данных

# Директивы Препроцессора

- `#define`
- `#undef`
- `#if`
- `#ifdef`
- `#ifndef`
- `#else`
- `#elif`

Так же, как в C++

- `#endif`
- `#error:` Вывод диагностического сообщения
- `#pragma:` Зависит от имплементации компилятора
- `#line:` Вывод номера строки при запуске макро

# Зарезервированные Слова в GLSL

```
attribute const uniform varying
break      continue do for while
if else    in out inout
float int void bool true false
discard return
mat2 mat3 mat4
vec2 vec3 vec4 ivec2 ivec3 ivec4
 bvec2 bvec3 bvec4
sampler1D sampler2D sampler3D
 samplerCube sampler1DShadow
 sampler2DShadow
struct
```

# Зарезервированные Слова в GLSL для Будущего Использования

- Использование на данный момент вызывает ошибку:

```
asm
class union enum typedef template
goto switch default
inline noinline volatile public
static extern external
long short double half fixed unsigned
input output
hvec2 hvec3 hvec4 dvec2 dvec3 dvec4
  fvec2 fvec3 fvec4
sampler2DRect sampler3DRect
sizeof cast
namespace using
```

# Типы Данных в GLSL

- `void` для функций без возвращаемого значения
- `bool` булевское значение, **true** или **false**
- `int` целочисленное значение со знаком
- `float` число с плавающей точкой
- Многокомпонентные вектора в форме `*vecN`, где {`*`} – идентификатор типа данных (`b` для булевских векторов, `i` для целочисленных векторов и просто `vecN` для векторов с плавающей точкой) и `N` – это количество компонент.
  - `vec2` – двух компонентный вектор с плавающей точкой
  - `bvec4` – четырех компонентный булевский вектор
  - `ivec3` – трех компонентный целочисленный вектор

# Матрицы и Чтение Текстур

- Матрицы: только NxN матрицы с плавающей точкой (2x2: `mat2`, 3x3: `mat3`, 4x4: `mat4`)

- Сэмплеры:
  - `sampler1D`
  - `sampler2D`
  - `sampler3D`
  - `samplerCube`
  - `sampler1DShadow`
  - `sampler2DShadowэ`

# Целочисленные Типы Данных

- Поддержка в железе не обязательна: на данный момент не поддерживается существующим железом
- Может использоваться для циклов и индексации массивов
- Точность 16 битов + плюс бит знака
  - Переполнение 16ти битов создает несовместимость
- Может быть переведены в данные с плавающей точкой для подсчетов
  - Контролируется драйверами
- Константы могут быть заданы в десятичной, восьмеричной и шестнадцатеричной системе отсчета

# Вектора и Матричные Данные

- 2-, 3-, и 4-компонентные вектора
- Матрицы могут быть только с плавающей точкой:
  - Адресуются в column major order
- Например:
  - `vec2 texcoord1, texcoord2;`
  - `vec3 position;`
  - `vec4 myRGBA;`
  - `ivec2 textureLookup;`
  - `mat4 viewMatrix;`
- Инициализация с помощью конструкторов

# Структуры Данных и Массивы

- **`struct`** может быть использован для создания новых типов данных:

```
struct light
{
    float intensity;
    vec3  position;
} lightVar;
```

Имя нового типа данных

Имя переменной

- Однотипные переменные могут быть представлены массивом:

```
float myLights[8];
```

or

```
const int nNumLights = 2;
light lights[nNumLights];
```

# Обозначение Использования Переменных

- < none: default >
  - Просто переменная
- **`Const`**
  - Константа, либо параметр для фукции с неизменяемым значением
- **`Attribute`**
  - Обозначает входные вершинные данные
- **`Uniform`**
  - Неизменяемый внутри шейдера значение
- **`Varying`**
  - Интеролируемые данные между вершинным и фрагментным шейдерами
- **`In`**
  - Входные параметры функций
- **`Out`**
  - Выводимые значения функций
- **`Inout`**
  - «Входит-Выходит»
- Глобальные переменные могут изпользовать обозначения “**`const`**, “**`attribute`**”, “**`uniform`**”, or “**`varying`**” – только один сразу
- Локальные переменные могут использовать только “**`const`**”

# Обозначение Использования `attribute`

- Для декларации переменной, как вершинных данных
  - Разрешено только в вершинном шейдере для чтения
  - Данные передаются в вершинный шейдер через vertex API либо как часть вершинного массива данных в OpenGL
  - Нельзя использовать как структуры данных либо как массивы: только как `float, vec2, vec3, vec4, mat2, mat3,` или `mat4`
    - `attribute vec4 position;`
    - `attribute vec3 normal;`
    - `attribute vec2 texCoord;`
- Максимальное количество вершинных аттрибутов – 16

# Стандартные Вершинные Атрибуты в OpenGL

- `attribute vec4 gl_Color;`
- `attribute vec4 gl_SecondaryColor;`
- `attribute vec3 gl_Normal;`
- `attribute vec4 gl_Vertex;`
- `attribute vec4 gl_MultiTexCoord0;`
- `attribute vec4 gl_MultiTexCoord1;`
- `attribute vec4 gl_MultiTexCoord2;`
- `attribute vec4 gl_MultiTexCoord3;`
- `attribute vec4 gl_MultiTexCoord4;`
- `attribute vec4 gl_MultiTexCoord5;`
- `attribute vec4 gl_MultiTexCoord6;`
- `attribute vec4 gl_MultiTexCoord7;`
- `attribute float gl_FogCoord;`

# Обозначение Использования `uniform`

- Неизменяемые внутри шейдера значения для чтения
- Заданные непосредственно программой через команды API либо через OpenGL state:
  - Например все переменные в RenderMonkey заданы как uniform параметры для шейдеров

```
uniform vec4 lightPosition;
```

- Может быть использованы с любыми типами данных
- Конкретная имплементация OpenGL драйвера задает максимальное количество доступных `uniform` параметров
- **Внимание**: Заметьте, что в OpenGL uniforms задаются в объекте шейдерной программы, а не в самих шейдерах

# Обозначение Использования `varying`

- Интерполируемые данные между вершинным и фрагментным шейдером
- Вершинный шейдер просчитывает значение этих переменных на каждую вершину и передает из в фрагментный шейдер через интерполяторы
- Переменные `varying` интерполируются как perspective-correct значение
- Фрагментный шейдер может читать `varying` значения
  - Декларации **должны совпадать** в вершинном и фрагментном шейдерах

```
varying vec3 normal;
```

- Можно использовать только `float`, `vec2`, `vec3`, `vec4`, `mat2`, `mat3`, и `mat4`, либо массивы из этих типов данных.

# Переменные Вывода из Вершинного Шейдера

- `vec4 gl_Position`: homogenous позиция вершины – *вывод обязателен*
- `float gl_PointSize`: вершинный шейдер может вывести размер рисуемой точки в пикселях
- `vec4 gl_ClipVertex`: вершинный шейдер может выверсти координаты для использования с user clipping planes.

# Переменные Вывода из Фрагментного Шейдера

- `vec4 gl_FragColor`
  - Цвет фрагмента, *вывод обязателен*
- `float gl_FragDepth`
  - Значение глубины сцены

# Конструкторы

- `int(bool)` – из Boolean в int
- `int(float)` – из float в int
- `float(bool)` - из Boolean в float

- `vec3(float)` - initializes each component of a vec3 with the float
- `vec4(ivec4)` - makes a vec4 from an ivec4, with component-wise conversion
- `vec2(float, float)` - initializes a vec2 with 2 floats
- `ivec3(int, int, int)` - initializes an ivec3 with 3 ints
- `bvec4(int, int, float, float)` - initializes with 4 Boolean conversions
- `vec2(vec3)` - drops the third component of a vec3
- `vec3(vec4)` - drops the fourth component of a vec4
- `vec3(vec2, float)` - vec3.x = vec2.x, vec3.y = vec2.y, vec3.z = float

# Доступ к Компонентам

- Поддерживаемые компоненты векторов:
  - {*x, y, z, w*} - используются при считывании позиций или нормаль
  - {*r, g, b, a*} - используются при считывание цветов
  - {*s, t, p, q*} – используются при считывании текстурных координат
    - `vec4 v4;`
    - `v4.rgba;` - четырех компонентный вектор
    - `v4.rgb;` - трех компонентный вектор
    - `v4.b;` - знание с плавающей точкой
    - `v4.xgba;` - нелегально – *нельзя мешать компоненты из разных наборов!*
  - Порядок компонент может быть различным для swizzle либо повторен (replicated):
    - `vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);`
    - `vec4 swiz = pos.wzyx;` - swiz = (4.0, 3.0, 2.0, 1.0)
    - `vec4 dup = pos.xxyy;` - dup = (1.0, 1.0, 2.0, 2.0)

- Компоненты матриц могут быть считаны используя [] []
  - `mat4 myMatrix;`
  - `myMatrix[0][2] = 4.0;`

# Вход в Шейдер

- Используется функция *main*

- Она должна быть объявлена таким образом:

```
void main(void)
{
  ...
}
```

# Flow Control Semantics

- If / else поддерживается

```
if ( bool_expression)
   True expression
else
   False expression
```

- Также поддерживаются циклы

```
for (init-expression; condition-expression;
     loop-expression)
  sub-statement

while (condition-expression)
  sub-statement

do
  statement
while (condition-expression)
```

# Jump Statements

- `continue;`
- `break;`

  Только в циклах
- `return` и `return expression`
- `discard`
  - Только используется, когда нужно прекратить подсчет данных для фрагмента
    - Фрагмент выброшен и данные буфера кадра не изменяются
  - Например: Можно тестировать значение alpha для фрагмента и выбрасывать фрагмент на основе этого теста

# Встроенные OpenGL Константы

- Можно использовать в вершинном и фрагментном шейдере

```
const int gl_MaxLights = 8; - GL 1.0
const int gl_MaxClipPlanes = 6; - GL 1.0
const int gl_MaxTextureUnits = 2; - GL 1.2
const int gl_MaxTextureCoordsARB = 2; - ARB_fragment_program
const int gl_MaxVertexAttributesGL2 = 16; - GL2_vertex_shader
const int gl_MaxVertexUniformFloatsGL2 = 512; -
  GL2_vertex_shader
const int gl_MaxVaryingFloatsGL2 = 32; - GL2_vertex_shader
const int gl_MaxVertexTextureUnitsGL2 = 1; -
  GL2_vertex_shader
const int gl_MaxFragmentTextureUnitsGL2 = 2; -
  GL2_fragment_shader
const int gl_MaxFragmentUniformFloatsGL2 = 64; -
  GL2_fragment_shader
```

# Встроенное Состояние OpenGL

- Программа может задать OpenGL state, который может быть считан внутри шейдеров используя встроенные переменные
  - `uniform mat4 gl_ModelViewMatrix;`
  - `uniform mat4 gl_ProjectionMatrix;`
  - `uniform mat4 gl_ModelViewProjectionMatrix;`
  - `uniform mat3 gl_NormalMatrix; // derived`
  - `uniform mat4 gl_TextureMatrix[gl_MaxTextureCoordsARB];`
  - `uniform float gl_NormalScale;`
  - …
  - А так же константы для параметров глубины сцены, цвета материалов, параметры задачи освещения, clip plane parameters, point parameters (size, etc), текстурных параметров, тумана, и т.д.

# Встроенные Функции

| | | | |
|---|---|---|---|
| radians degrees | sin cos tan asin | acos atan atan | exp2 log2 sqrt |
| pow inversesqrt | abs sign floor | ceil fract mod | min max mix |
| clamp step dot | distance cross | smoothstep | length |
| lessThan lessThanEqual | greaterThan greaterThanEqual | equal notEqual | normalize |
| reflect | ftransform | any all not | faceforward |
| matrixCompMult | | | |

А так же для текстурного доступа:

- texture1D  texture2DProj texture1DProj texture1DLod
- texture1DProjLod …
- textureCube, textureCubeLod
- shadow1D, shadow2D , shadow1DProj, shadow2DProj, …