高级计算机图形学

中国科学技术大学计算机学院
黄章进
zhuang@ustc.edu.cn

第十章之第三节

GLSL(2)

内容

- GLSL语法
  - 数据类型
  - 运算符
  - 流控制
  - 函数
  - 顶点着色器
  - 片段着色器

基础知识

- GLSL的语法来自于C/C++
  - 字符集，记号(tokens)，标识符（identifiers）
  - 分号， 大括号嵌套
  - 控制流： if-else, for, while, do-while
  - 关键字
  - 注释： //…和/*…*/
  - 入口函数： void main()

数据类型

- 标量
  - float 浮点型，IEEE754单精度
    - 不支持双精度，字面浮点常数的后缀f或F是可选的
  - int 整型，至少16位精度[-65535,65535]
    - 十进制、八进制、十六进制： 42, 052, 0x2A
    - 不支持长整型，不支持按位运算
    - GLSL1.5精度为32位，支持无符号整型uint
  - bool 布尔型：true/false
    - 关系运算和逻辑运算产生布尔型
    - 条件跳转表达式 (if, for, ?:, while, do-while）只接受布尔型

数据类型

- 向量
  - 二维到四维的float、int、bool型
    - vec2、vec3、vec4
    - ivec2、ivec3、ivec4
    - bvec2、bvec3、bvec4
  - 分量访问
    - (x, y, z, w) 位置或方向
    - (r, g, b, a) 颜色
    - (s, t, p, q) 纹理坐标

数据类型

- 矩阵
  - 2 × 2, 2 × 3, 2 × 4, 3 × 2, 3 × 3, 3 × 4, 4 × 2, 4 × 3, 和 4 × 4浮点矩阵
    - mat2, mat3, mat4
    - mat2x2, mat2x3, mat2x4
    - mat3x2, mat3x3, mat3x4
    - mat4x2, mat4x3, mat4x4
  - 第一个数指定列数，第二个数指定行数
  - 列优先。例如4列3行矩阵： mat4x3 M;
    - M[2]是第3列，类型vec3；M[3][1]是第4列第2行元素

数据类型

- 采样器（sampler）
  - 着色器访问纹理的不透明句柄
    - sampler[1,2,3]D 访问1D,2D,3D纹理
    - samplerCube 访问立方图纹理
    - sampler[1,2]DShadow 访问1D,2D阴影图
  - 例子:

```
uniform sampler2D Grass;
vec4 color = texture2D(Grass, coord);
```

数据类型-结构(struct)

- 结构成员可以是基本类型和数组
- 不支持位字段

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

light lightVar2;
```

- 不支持嵌套的匿名结构
- 不支持嵌入的结构定义

```
struct S { float f; };

struct T {
S; // Error: 匿名结构
struct { ... }; // Error:
S s; // Okay: 嵌套结构
};
```

数据类型

- 数组
  - 可以声明基本类型和结构的数组
  - 除了作为函数参数外，声明时可以不指定大小
  - 只能声明一维数组
  - 可用length方法来查询数的大小

1. 可通过再次声明指定数组大小，但之后不能再次声明

```
vec4 points[];
vec4 points[10]; //大小为10
vec4 points[20]; // 非法
vec4 points[]; // 非法
```

2. 编译时根据最大下标确定数组大小

```
vec4 points[]; //
points[2]=vec4(1.0); //为3
points[7]=vec4(2.0); //为8
int size=points.length();
```

数据类型

- void
  - 声明函数不返回任何值
  - 没有缺省的函数返回值
  - 除了再用于空的函数形参列表外，不能用于其他声明

```
void main()
{
    …
}
```

隐式类型转换

- 只支持以下整型到浮点型的隐式转换
  - int -> float
  - ivec[2,3,4] -> vec[2,3,4]
- 不支持数组和结构的隐式转换
  - int数组不能隐式转换为float数组

作用域

- GLSL的作用域规则和C++相似
- 在所有函数定义之外声明的变量具有全局作用域
- 花括号内声明的变量作用域在花括号内
- while测试和for语句中声明的变量作用域限于循环体内
- if语句不允许声明新变量

存储限定符

- 声明变量时可以在类型前指定存储限定符
  - < none: default > 局部读写或函数输入参数
  - const 只读的编译时常量或函数参数
  - attribute 应用程序传递给顶点着色器的每顶点数据
  - uniform 应用程序传递给着色器的每图元数据
  - varying 顶点着色器经光栅化器传递给片段着色器的插值数据

存储限定符

- 全局变量只能指定为限定符const、attribute、uniform和varying中的一个
- 局部变量只能使用const限定符
- 函数参数只能使用const限定符
- 函数返回值和结构字段不能使用限定符
- 缺省限定符
  - 没有限定符的全局或局部变量不能和应用程序和其他着色器交换信息

const限定符

- const变量必须在声明时初始化
  - const vec3 zAxis = vec3 (0.0, 0.0, 1.0);
- 结构变量可以声明为const，但结构的字段不能限定为const

attribute限定符

- OpenGL用attribute变量向顶点着色器传递每顶点的数据
- 在顶点着色器中声明为全局变量，只读
- 不能在片段着色器中声明使用
- 使用OpenGL顶点API或顶点数组传递属性变量值给顶点着色器
- 类型限制为浮点标量、向量和矩阵，不能声明为数组或结构
- float标量内部存储为vec4

uniform限定符

- unifrom用于声明在图元处理时保持不变的全局变量
  - 不能在glBegin和glEnd间改变其值
- 在顶点和片段着色器中只读
- 任何类型的变量，包括结构和数组

varying限定符

- varying变量是顶点和片段着色器交流数据的唯一方式
  - 顶点着色器输出的每顶点数据由光栅化器插值为每片段数据输入到片断着色器
- 全局变量，顶点着色器可读写，片段着色器只读
- 类型限制为浮点标量、向量和矩阵，以及这些类型的数组，不能为结构

初始化和构造函数

- 变量可以在声明的时候初始化
  - 整型常量可以用八进制、十进制和十六进制
  - 浮点值必须包括一个小数点，除非是用科学计数法，如3E-7，后缀f或F可选
  - 布尔型为true或false
- 聚合类型（向量、矩阵、数组和结构）必须用构造函数进行初始化

构造函数

```
vec4 color = vec4(0.0, 1.0, 0.0, 1.0);
vec4 rgba = vec4(1.0); // sets each component to 1.0
vec3 rgb = vec3(color); // drop the 4th component
mat2 m = mat2(1.0, 2.0, 3.0, 4.0);
```

$$\begin{bmatrix} 1.0 & 3.0 \\ 2.0 & 4.0 \end{bmatrix}$$

```
struct light {
float intensity;
vec3 position;
};
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));

const float c[3] = float[3](5.0, 7.2, 1.1);
const float d[3] = float[](5.0, 7.2, 1.1);
```

类型转换

- GLSL不支持C风格的强制类型转化，只能用构造函数进行显式类型转换
- true转换为1或1.0，false转换为0或0.0
- 0或0.0转换为false，非0转换为true

```
float f = 2.3;
bool b = bool(f);

float f = float(3); // convert integer 3 to floating-point 3.0
float g = float(b); // convert Boolean b to floating point
vec4 v = vec4(2); // set all components of v to 2.0
```

访问向量元素

- 可以用 []或者选择运算符(.)逐个索引向量的元素
  - x, y, z, w 位置或方向数据
  - r, g, b, a 颜色数据
  - s, t, p, q 纹理坐标，注意OpenGL是s,t,r,q
  - **a[2], a.b, a.z, a.p** 是一样的
- 混合（**Swizzling**）运算符可以用来操纵每个分量

```
vec4 a;
a.yz = vec2(1.0, 2.0);
```

混合操作

- 各分量名称可打乱顺序，但只能使用同一组名称，且长度不能大于4
- 左值中名称可以重复，右值不可以

```
vec4 v4;
v4.rgb; // is a vec3
v4.xgba; // is illegal

vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);
vec4 dup = pos.xxyy; // dup = (1.0, 1.0, 2.0, 2.0)
pos.xw = vec2(5.0, 6.0); // pos = (5.0, 2.0, 3.0, 6.0)
pos.xx = vec2(3.0, 4.0); // illegal - 'x' used twice
```

运算符

- GLSL中使用的运算符优先级和结合性和C是一致的
  - ++ --
  - - !
  - 算术运算符： * / + -
  - 关系运算符： < > <= >=
  - 逻辑运算符： == != && ^^(异或) ||
  - 选择运算符： ?:
  - 赋值： = += -= *= /=

逐分量运算

- 一般地，向量和矩阵运算是逐分量进行的
- 例外情形：矩阵乘向量、向量乘矩阵以及矩阵相乘按线性代数运算规则

```
vec4 v, u;  float f;
v = u + f; // v = (u.x + f, u.y + f, u.z + f, u.w + f);

mat4 m;
v * u; // 逐分量乘，不是内积
v * m; // 行向量乘矩阵
m * v; // 矩阵乘列向量
 m * m; // 矩阵相乘
```

流控制

- GLSL的流控制和C++非常相似，着色器的入口点是main函数
- 循环结构：for、while和do-while，for和while语句中可以声明变量，作用域持续到子语句结束。break和continue同C
- 选择结构：if和if-else，只是if语句不能声明变量。还有选择操作符(?:)。
  - GLSL1.5支持switch-case结构
- discard 仅用在片段着色器中丢弃片段

函数

- 函数的使用和C++类似，函数名可以通过参数个数和类型进行重载
- 函数声明:

returnType functionName (type0 arg0, type1 arg1, ..., type*n* argn);

- 函数定义:

```
returnType functionName (type0 arg0, type1 arg1, ..., typen argn)
{
// do some computation
return returnValue;
}
```

参数限定符

- GLSL按值-返回（value-return)调用函数
- 参数类型type*n*必须指定类型和可选的参数限定符
  - < none: default > 同in
  - in　　复制到函数中，函数内可写
  - const in　复制到函数中，函数内只读
  - out　　返回时从函数中复制出来
  - inout　　复制进函数并在返回时复制出来

参数和返回值类型

- 参数和返回值可以是任意类型
  - 数组必须显式指定大小
  - 数组传入或返回时只需使用变量名，且大小和函数声明中匹配
- 函数必须声明返回值类型，无返回值时声明为void
- 输入参数从左往右求值
- 函数不能递归调用

内置函数

- 角度和三角函数：逐分量
- 指数函数：逐分量
- 常用函数：逐分量
- 几何函数：向量
- 矩阵函数
- 向量关系函数
- 纹理访问函数
- 片段处理函数：只在片段着色器使用
- 噪声函数

顶点着色器

- 顶点着色器必须负责如下OpenGL固定功能的逐顶点操作:
  - 模型-视图矩阵和投影矩阵没有应用于顶点坐标
  - 纹理矩阵没有应用于纹理坐标
  - 法向量没有变换到观察坐标，且没有重新缩放或规范化
  - 没有对启用GL_AUTO_NORMAL执行法向规范化

顶点着色器

- 没有自动生成纹理坐标
- 没有执行每顶点的光照计算
- 没有执行彩色材料计算(glColorMaterial)
- 没有执行彩色索引光照计算
- 当设置当前光栅位置时，将应用上述所有操作

顶点着色器

- 下述操作将应用到顶点着色器所得到的顶点值
  - 颜色钳位(clamping)和掩模（masking）
  - 裁剪坐标的透视除法
  - 包括深度范围缩放的视口映射
  - 包括用户定义裁剪平面的裁剪
  - 前向面确定
  - 平面明暗处理
  - 颜色、纹理坐标、雾、点大小等一般属性裁剪
  - 最终颜色处理

内置顶点属性(attribute)

- 应用程序提供顶点信息：位置、法向、颜色、纹理坐标
  - glVertex, glNormal, glColor, glTexCoord
  - glDrawArrays
- 内置顶点属性

```
attribute vec4 gl_Vertex;   // glVertex
attribute vec4 gl_Color;    // glColor
attribute vec4 gl_SecondaryColor;   // glSecondaryColor
attribute vec3 gl_Normal;  // glNormal
attribute vec4 gl_MultiTexCoordn;//glMultiTexCoord(n,…), n=0,…7
attribute float gl_FogCoord; // glFogCoord
```

特殊输出变量

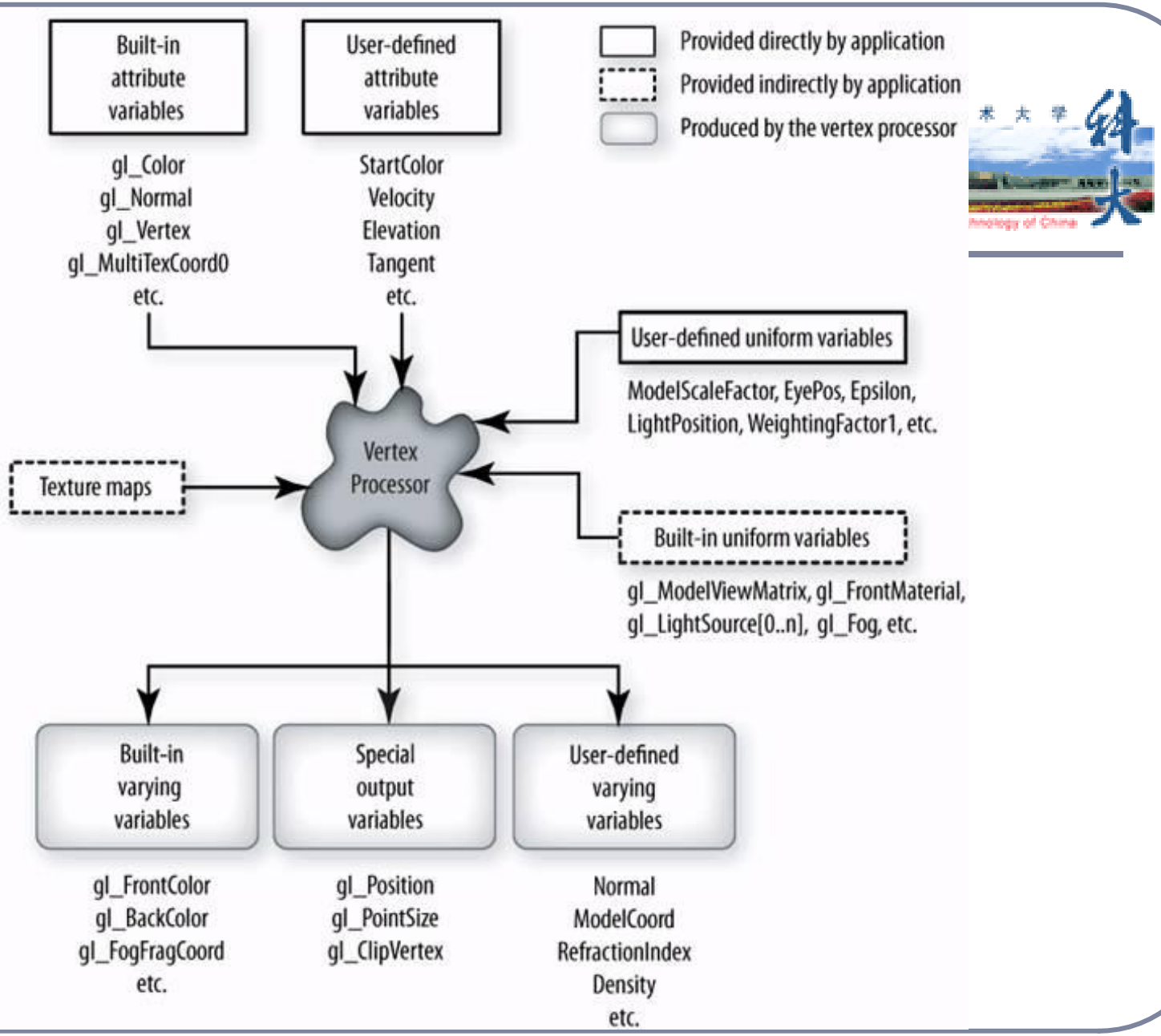
- gl_Position: 顶点着色器必须写入的齐次裁剪坐标，将用于图元装配、裁剪、剔除
- gl_PointSize: 将被光栅化的点大小
- gl_ClipVertex: 相对于用户裁剪平面的顶点位置

```
vec4 gl_Position; // must be written to
float gl_PointSize; // may be written to
vec4 gl_ClipVertex; // may be written to
```

内置varying变量

```
varying vec4 gl_FrontColor;
varying vec4 gl_BackColor;
varying vec4 gl_FrontSecondaryColor;
varying vec4 gl_BackSecondaryColor;
varying vec4 gl_TexCoord[]; // at most will be gl_MaxTextureCoords
varying float gl_FogFragCoord;
```

- 根据图元的朝向，固定功能从gl_FrontColor, gl_BackColor, gl_FrontSecondaryColor和gl_BackSecondaryColor计算片段着色器使用的gl_Color和gl_SecondaryColor


Built-in attribute variables
User-defined attribute variables
Provided directly by application
Provided indirectly by application
Produced by the vertex processor
gl_Color
gl_Normal
gl_Vertex
gl_MultiTexCoord0
etc.
StartColor
Velocity
Elevation
Tangent
etc.
User-defined uniform variables
ModelScaleFactor, EyePos, Epsilon,
LightPosition, WeightingFactor1, etc.
Texture maps
Vertex Processor
Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial,
gl_LightSource[0..n], gl_Fog, etc.
Built-in varying variables
Special output variables
User-defined varying variables
gl_FrontColor
gl_BackColor
gl_FogFragCoord
etc.
gl_Position
gl_PointSize
gl_ClipVertex
Normal
ModelCoord
RefractionIndex
Density
etc.

片段着色器

- 片段着色器取代固定功能的纹理、颜色求和、雾等片段操作
  - 没有应用纹理环境和纹理函数
  - 没有执行纹理应用程序
  - 没有应用颜色求和
  - 没用应用雾化

片段着色器

- 下述操作的行为不会改变
  - 纹理图像规格
  - 压缩纹理图像规格
  - 按照指定方式工作的纹理参数
  - 纹理状态和代理状态
  - 纹理对象规格
  - 纹理比较模式

内置varying变量

```
varying vec4 gl_Color;
varying vec4 gl_SecondaryColor;
varying vec4 gl_TexCoord[]; // at most will be gl_MaxTextureCoords
varying float gl_FogFragCoord;
varying vec2 gl_PointCoord;
```

- gl_PointCoord用于点块纹理(point sprites)

特殊变量

- 特殊输入变量，只读

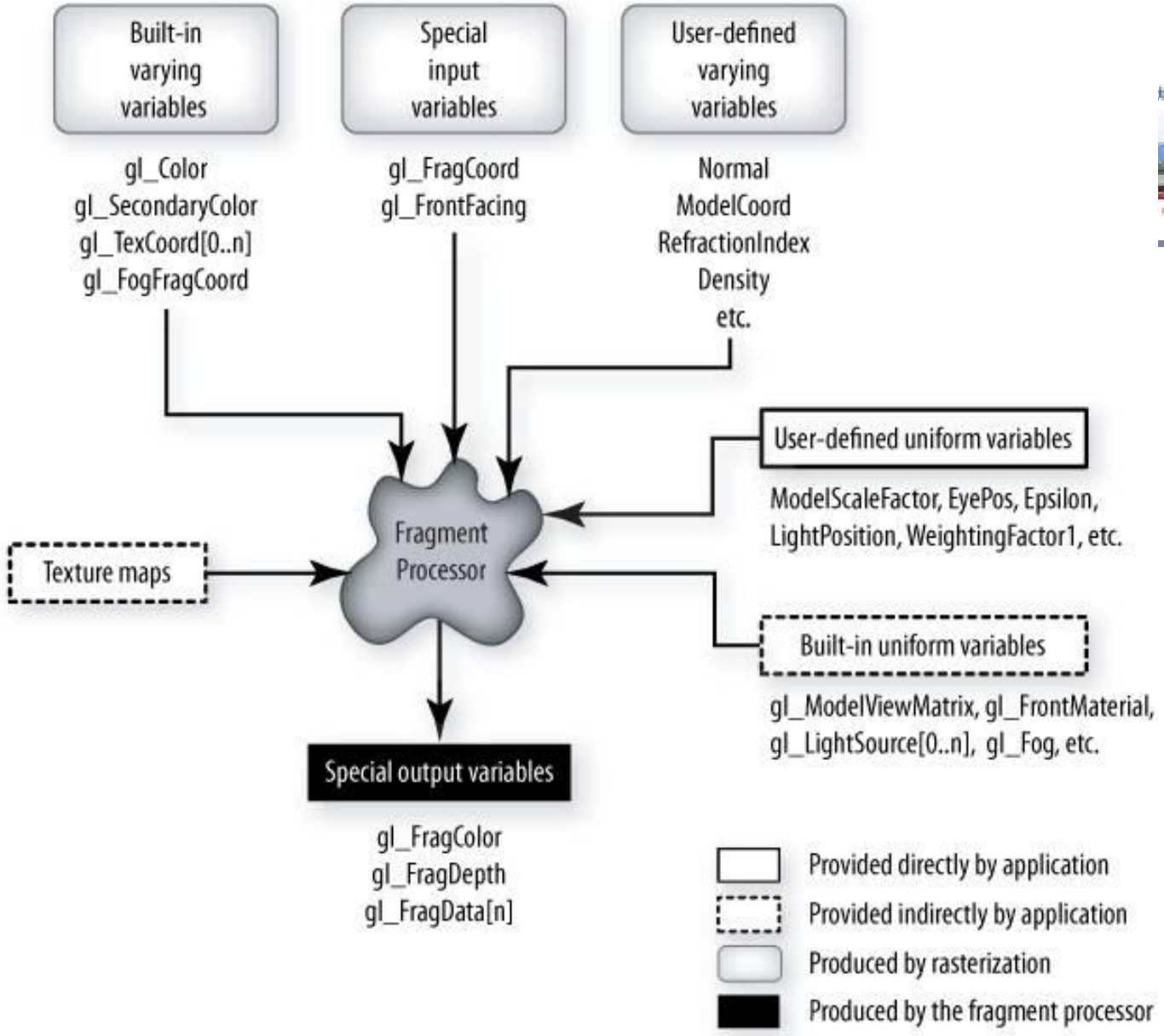
```
vec4 gl_FragCoord;
bool gl_FrontFacing;
```

  - gl_FragCoord：片段的窗口坐标(x,y,z,1/w)
  - gl_FrontFacing：片段所属图元的朝向

- 特殊输出变量

```
vec4 gl_FragColor;
vec4 gl_FragData[gl_MaxDrawbuffers];
float gl_FragDepth;
```

  - gl_FragColor：写入颜色缓冲区的颜色
  - gl_FragData：写入离屏缓冲区（使用glDrawBuffers)的数据
  - 只能写入其一


Built-in varying variables
Special input variables
User-defined varying variables
gl_Color
gl_SecondaryColor
gl_TexCoord[0..n]
gl_FogFragCoord
gl_FragCoord
gl_FrontFacing
Normal
ModelCoord
RefractionIndex
Density
etc.
User-defined uniform variables
ModelScaleFactor, EyePos, Epsilon, LightPosition, WeightingFactor1, etc.
Texture maps
Fragment Processor
Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource[0..n], gl_Fog, etc.
Special output variables
gl_FragColor
gl_FragDepth
gl_FragData[n]
Provided directly by application
Provided indirectly by application
Produced by rasterization
Produced by the fragment processor

内置常量


```
const int gl_MaxLights = 8;
const int gl_MaxClipPlanes = 6;
const int gl_MaxTextureUnits = 2;
const int gl_MaxTextureCoords = 2;
const int gl_MaxVertexAttribs = 16;
const int gl_MaxVertexUniformComponents = 512;
const int gl_MaxVaryingFloats = 32;
const int gl_MaxVertexTextureImageUnits = 0;
const int gl_MaxTextureImageUnits = 2;
const int gl_MaxFragmentUniformComponents = 64;
const int gl_MaxCombinedTextureImageUnits = 2;
const int gl_MaxDrawBuffers = 1;
```

内置uniform变量


```
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_ModelViewProjectionMatrix;
uniform mat4 gl_TextureMatrix[gl_MaxTextureCoords];
uniform mat3 gl_NormalMatrix;

uniform vec4 gl_ClipPlane[gl_MaxClipPlanes];

uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];
uniform gl_LightModelParameters gl_LightModel;
uniform gl_FogParameters gl_Fog;
```

固定功能着色器

- 变换
  - 把顶点位置变换到裁剪空间，有两种方式

```
// Transform vertex to clip space
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

//or this:
gl_Position = ftransform();
```

    - 第二种方法保证以与固定功能完全相同的方式计算变换的位置

视点坐标位置计算

- 光源位置被OpenGL用模型视图矩阵变换到视点坐标系。下述代码把顶点位置变换为视点坐标

```
vec4 ecPosition;
vec3 ecPosition3; // in 3D space

// Transform vertex to eye coordinates
if (NeedEyePosition)
{
    ecPosition = gl_ModelViewMatrix * gl_Vertex;
    ecPosition3 = (vec3(ecPosition)) / ecPosition.w;
}
```

法向变换

- 在视点空间计算光照时，法向也必须变换到视点空间

```
normal = gl_NormalMatrix * gl_Normal;
```

- 法向规范化： glEnable(GL_NORMALIZE)

```
normal = normalize(normal);
```

- 法向缩放：缩放比例因子为内置uniform变量gl_NormalScale

```
normal = normal * gl_NormalScale;
```

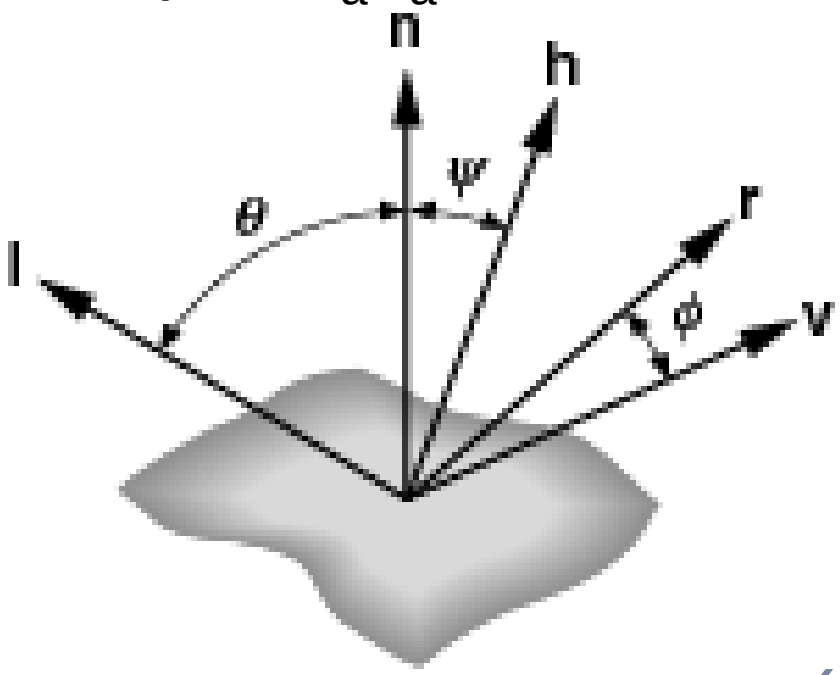
纹理坐标变换

- OpenGL的每一个纹理单元都定义了一个纹理矩阵，可用内置的uniform变量gl_TextureMatrix访问

```
gl_TexCoord[0] = gl_TextureMatrix[0] * gl_MultiTexCoord0;
```

光照

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{h} \cdot \mathbf{n})^{\alpha} + k_a I_a$$



点光源


```
void PointLight(in int i, in vec3 eye, in vec3 ecPosition3, in vec3 normal, inout
vec4 ambient, inout vec4 diffuse, inout vec4 specular)
{
    float nDotVP; // normal . light direction
    float nDotHV; // normal . light half vector
    float pf; // power factor
    float attenuation; // computed attenuation
    factor float d; // distance from surface to light source
    vec3 VP; // direction from surface to light position
    vec3 halfVector; // direction of maximum highlights

    // Compute vector from surface to light position
    VP = vec3(gl_LightSource[i].position) - ecPosition3;

    // Compute distance between surface and light position
    d = length(VP);
```

点光源


```
        // Compute attenuation
        attenuation = 1.0 / (gl_LightSource[i].constantAttenuation +
                             gl_LightSource[i].linearAttenuation * d
                             + gl_LightSource[i].quadraticAttenuation * d * d);

        halfVector = normalize(VP + eye);

        nDotVP = max(0.0, dot(normal, VP));
        nDotHV = max(0.0, dot(normal, halfVector));

        if (nDotVP == 0.0) pf = 0.0;
        else pf = pow(nDotHV, gl_FrontMaterial.shininess);
        ambient += gl_LightSource[i].ambient * attenuation;
        diffuse += gl_LightSource[i].diffuse * nDotVP * attenuation;
        specular += gl_LightSource[i].specular * pf * attenuation;
}
```

ShaderGen

- 自动生成模拟固定流水线功能着色器源代码的程序
  - http://mew.cx/glsl/shadergen/