计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第二节 GLSL(I)

顶点着色器应用

- 顶点的移动
  - 变形（morphing）
  - 波动
  - 分形
- 光照
  - 更真实的模型
  - 卡通着色器

片段着色器应用

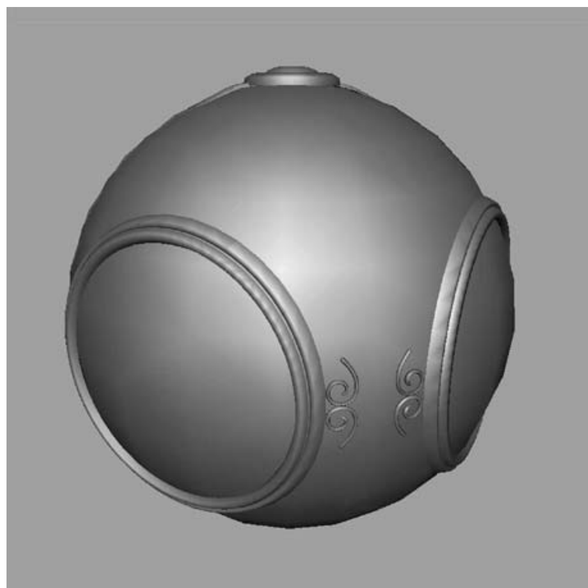
- 逐片段进行光照计算

逐顶点光照计算

逐片段光照计算

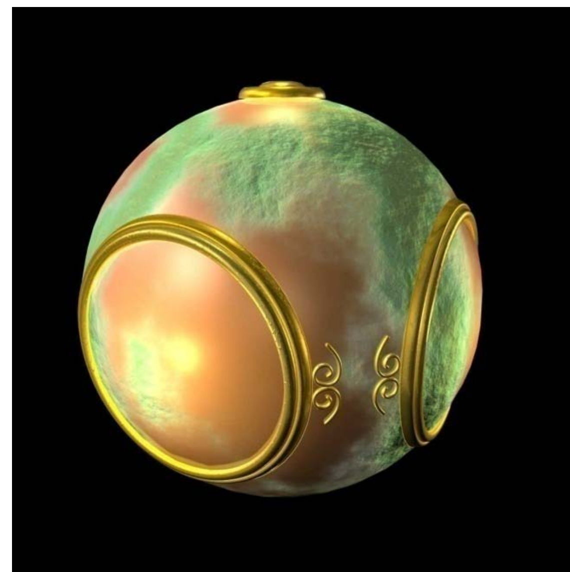
片段着色器应用

- 典型应用：纹理映射



光滑明暗处理

环境映射



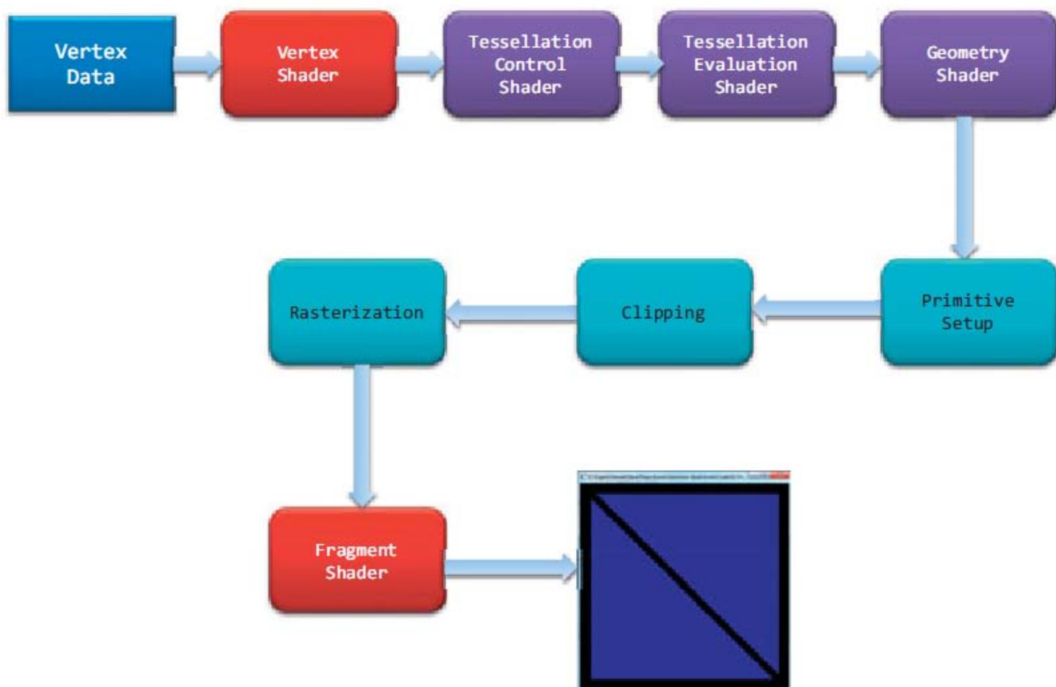
凹凸映射

着色器的编程语言

- OpenGL Shading Language的缩写
- OpenGL 4.6的一部分
- 高级类C语言
- 引进新的数据类型
  - 矩阵
  - 向量
  - 采样器（Samplers）
- OpenGL的状态通过内置变量传递

可编程流水线

- 利用GLSL语言编写着色器（Shader），用于替代固定流水线中顶点处理子流水线及片元处理子流水线
- 另外，最新版本的OpenGL还提供了细分着色器，几何着色器，计算着色器等，来提供更丰富的绘制功能



一个简单的顶点着色器

```
//filename: vGL_simple.vert
#version 330 core

layout(location = 0) in vec4 aPosition;

uniform mat4 ModelViewMatrix;
uniform mat4 ProjectionMatrix;

void main(void)
{
    gl_Position = ProjectionMatrix * ModelViewMatrix *
  aPosition;
}
```

一个简单的片元着色器

```
//filename: fGL_simple.frag
#version 330 core

layout(location = 0) out vec4 fColor;

void main (void)
{
    fColor = vec4(0.0f, 0.0f, 1.0f, 1.0f);
}
```

GLSL的基本语法

- 与C语言类似，着色器从main()函数开始运行；可自定义函数；注释方式相同
- 不同之处：
  - #version 330 core：指定所用的OpenGL语言版本，什么模式
  - uniform：变量修饰符，指变量为用户应用程序传递给着色器的数据，它对于给定的图元而言是一个常量
  - in：变量修饰符，指这个变量为着色器的输入变量
  - out：变量修饰符，指这个变量为着色器的输出变量
  - layout：布局限定符，指定变量的布局规范

GLSL的代码编写与命名

- GLSL的代码有两种编写方式
  - 利用C/C++语言的字符串数组（优点：不需要进行文件读写）

例如：

```
Const char *vShader = {
    "#version 330 core"
    "layout(location = 0) in vec4 aPosition;"
    …
};
```

  - 利用文本编辑器编写，然后利用文件读写载入程序中（优点：代码修改方便）

- 若采用文本编辑器，GLSL代码的文件后缀建议使用
  - .vert：顶点着色器
  - .frag：片元着色器
  - .tesc，.tese：细分控制/求值着色器
  - .geom：几何着色器
  - .comp：计算着色器

顶点着色器的数据输入与输出

- 输入逐顶点变量的基本步骤：
  - 初始化顶点数组对象：glGenVertexArrays, glBindVertexArray等
  - 分配顶点缓存对象：glGenBuffers，glBindBuffer等
  - 将数据载入缓存对象：glBufferData（或glBufferSubData）等
  - 将顶点数据与顶点着色器的in变量关联：glGetAttribLocation, glVertexAttribPointer，glEnableVertexAttribArray等
- 输入uniform变量的基本步骤：
  - 查询uniform变量的索引：glGetUniformLocation等
  - 设置uinform变量的值：glUniform, glUniformMatria等（根据类型选择合适的函数）
- 利用out修饰变量，在片元着色器可接收经过插值的相应变量
- 利用transform feedback机制，可将数据回传至缓存中

片元着色器的输入输出与输出

- 利用in修饰变量，可从顶点着色器获取经过插值的数据
- 利用out修饰变量，将数据输出至帧缓存中；位置可由layout指定；
- 输出到哪个颜色缓存可通过glDrawBuffer或glDrawBuffers指定

连接着色器与OpenGL

Program

glCreateProgram

glAttachShader

glAttachShader

glLinkProgram

glUseProgram

Vertex Shader

glCreateShader

glShaderSource

glCompileShader

Fragment Shader

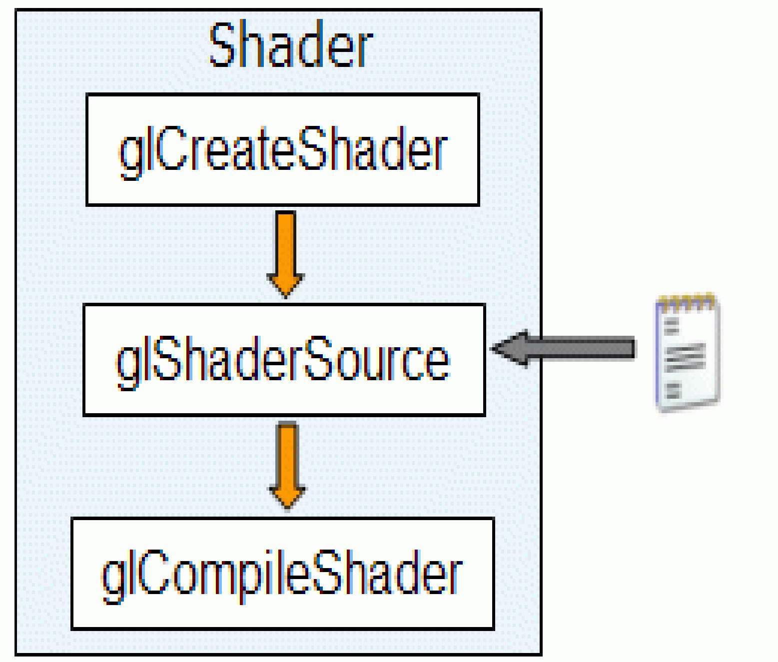
glCreateShader

glShaderSource

glCompileShader

创建着色器

- 创建着色器对象
- 加载着色器源代码
- 编译着色器对象
- 验证是否成功编译



创建着色器对象

- `GLuint glCreateShader(GLenum type);`
  - 创建一个空的着色器对象，着色器对象维护定义着色器的源代码字符串
  - type可取值为
    - GL_VERTEX_SHADER：顶点着色器
    - GL_FRAGMENT_SHADER：片段着色器
  - 返回非零整数，出错时返回零

加载着色器源代码

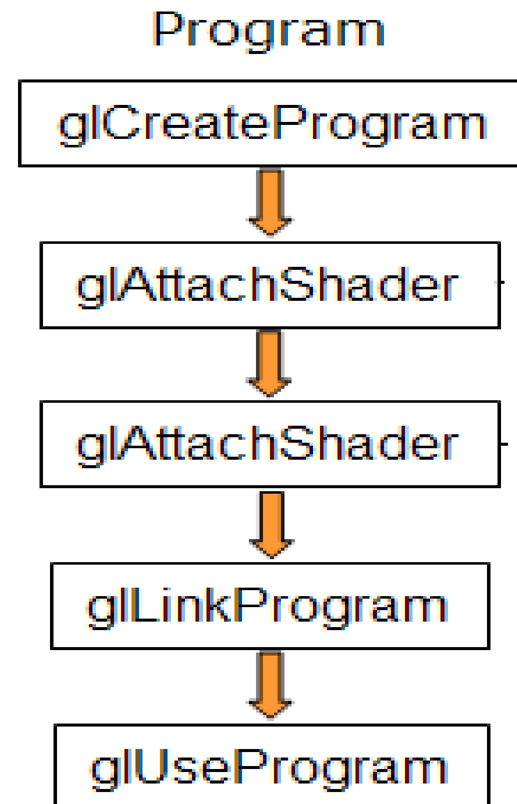
- `void glShaderSource(GLuint shader, GLsizei count, const CLchar **string, const GLint *length);`
  - 把存储在字符串数组string中的源代码加载到着色器对象shader，之前存储在着色器对象里的源代码将被完全替换
  - count - 字符串的个数
  - length - NULL（每个字符串都是以null结束）或字符串长度的数组（若有长度为负值，表示该字符串以null结束）

编译着色器对象

- `void glCompileShader(GLuint shader);`
  - 编译着色器对象shader的源代码
  - 可以调用glGetShaderiv()函数以GL_COMPILE_STATUS为参数查询编译结果
  - 可以调用glGetShaderInfoLog()函数查看编译信息日志

创建程序

- 创建程序对象
- 把着色器对象连接到程序对象
- 链接程序
- 验证链接是否成功
- 使用着色器



创建程序对象

- GLuint glCreateProgram(void);
  - 创建一个空的程序对象
  - 返回非零整数，出错时返回零
  - OpenGL用程序对象来封装管理可执行着色器
  - 可创建多个程序对象，并在渲染时切换以使用不同的着色器

连接着色器对象

- `void glAttachShader(GLuint program, GLuint shader);`
  - 把着色器对象shader连接（attach）到程序对象program
  - 可在加载源代码或编译前，把着色器对象连接到程序对象
  - 可连接顶点/片段着色器对，或只连接一类
  - 可连接多个同一类的着色器对象到程序对象，但只能有一个着色器对象有main函数
  - 一个着色器对象可连接到多个程序对象

链接程序

- void glLinkProgram(GLuint program);
  - 链接程序对象program，生成可执行程序
  - 链接前，连接到程序对象的着色器对象必须已成功编译
  - 可以调用glGetProgramiv()函数以GL_LINK_STATUS为参数查询链接结果
  - 可以调用glGetProgramInfoLog()函数查看链接信息日志

使用程序

- void glUseProgram(GLuint program);
  - 安装可执行程序program作为OpenGL渲染状态机的一部分进行顶点和/或片段处理
  - program为0时，使用固定功能流水线
  - 程序对象在使用中时，应用程序可以随意修改甚至删除程序对象，而不会影响作为当前状态机一部分的可执行代码
  - glLinkProgram()成功重链接使用中的程序对象后，将安装生成的可执行代码作为当前渲染状态机的一部分

清理函数

- `void glDeleteShader(GLuint shader);`
  - 删除着色器对象shader
  - 如果着色器对象没有连接到任何程序对象，立即删除；否则，标记为删除
- `void glDetachShader(GLuint program, GLuint shader);`
  - 把着色器对象shader从程序对象program分离
  - 如果着色器标记为删除，且分离后没有连接到其他着色器，则被删除

清理函数

- void glDeleteProgram(GLuint program);
  - 删除程序对象program
  - 如果程序对象不是正在使用的渲染状态机的一部分，立即删除；否则，标记为删除
  - 当程序对象被删除时，所有连接的着色器对象将被分离出来

查询函数

- void glGetShaderiv(GLuint shader, GLuint index, GLint *params);
  - 返回着色器对象shader参数为pname的属性值

pname	params
GL_SHADER_TYPE	着色器类型：GL_VERTEX_SHADER或GL_FRAGMENT_SHADER
GL_DELETE_STATUS	删除状态：GL_TRUE 当前标记为删除
GL_COMPILE_STATUS	编译状态：GL_TRUE 之前的编译成功
GL_INFO_LOG_LENGTH	信息日志的长度(含null)，0表示无日志
GL_SHADER_SOURCE_LENGTH	串联的源代码字符串的长度(含null字符)，0表示没加载源代码

查询函数

- void glGetProgramiv(GLuint program, GLenum pname, GLint *params);
  - 返回程序对象program参数为pname的属性值

pname	params
GL_DELETE_STATUS	删除状态：GL_TRUE 当前标记为删除
GL_LINK_STATUS	链接状态：GL_TRUE 之前的链接成功
GL_VALIDATE_STATUS	验证状态：GL_TURE 之前的验证成功
GL_INFO_LOG_LENGTH	信息日志的长度(含null)，0表示无日志
GL_ATTACHED_SHADERS	连接的着色对象的个数
GL_ACTIVE_ATTRIBUTES	活动属性变量的个数
GL_ACTIVE_UNIFORMS	活动一致变量的个数

查询函数

- void glGetShaderInfoLog(GLuint shader, GLsizei bufSize, GLsizei *length, GLchar *infoLog);
- void glGetProgramInfoLog(GLuint program, GLsizei bufSize, GLsizei *length, GLchar *infoLog);
  - infoLog返回着色器对象shader最后一次编译或程序对象program最后一次链接的信息日志
  - 缓冲区infoLog的大小为bufSize，实际返回的日志字符数存放在length中
  - 信息日志的字符数可由glGetShaderiv()或glGetProgramiv()查询得到

查询函数

- `void glGetShaderSource(GLuint shader, GLsizei bufSize, GLsizei *length, GLchar *source);`
  - source返回着色器对象shader的源代码字符串
  - 缓冲区source的大小为bufSize，实际的源代码字符数存放在length中
  - 源代码字符串的长度可用glGetShaderiv()查询得到

查询函数

- `GLboolean glIsShader(GLuint shader);`
- `GLboolean glIsProgram(GLuint program);`
  - 检查shader或program是否是着色器对象或程序对象的名称
- `void glValidateProgram(GLuint program);`
  - 验证程序对象program是否可以在当前OpenGL环境中运行
  - 如果验证通过，程序对象program的GL_VALIDATE_STATUS值置为GL_TURE

Thanks for your attention!