**第七章：着色器**

**高效的GPU渲染方案**

本章介绍着色器的基本知识以及Geiv下对其提供的支持接口，并以“渐变高斯模糊”为线索进行实例的演示讲解。

[背景信息]

[计算机中央处理器的局限性]

在大学的“数字图像处理”课程中，老师讲解了高斯模糊的基本算法，并使用C#进行了基本实现。

高斯模糊，简单地说，就是使用高斯权重模板对图像的每一个像素进行再计算、填充，以达到模糊的效果。

在课程中，对于给定的模板与模糊度系数，对一副800X600的图像进行模糊处理，需要计算48万个像素点，尽管当时机房已经普及了酷睿系列CPU，但这个过程依旧耗时2~3秒。

于是有这样的一个问题：在游戏中，我们经常看到画面进行模糊到清晰的动态转变，即使是次世代的GBA\SFC等游戏主机依然有这样的效果实现，而论硬件性能，我们的PC应该远远高于这些次世代游戏机，而2、3秒的运算结果已经明确告诉我，想要进行更为高效的模糊计算，显示地挨个计算像素点是不可能达到预期效率的，想要一秒钟完成60幅不同模糊度的图像计算，还要另寻他路。

为什么计算性能普遍较低的游戏机却能实现高性能CPU都无法流畅做到的运算呢？笔者认为还是因为CPU的结构差异导致的，PC的任务种类多，过程复杂，需要复杂的指令集系统；但游戏机的CPU主要为图像而生，只需使用针对图像的少数精简指令系统即可。

总而言之，为了保证CPU在复杂任务下的通用性，计算机的CPU并不擅长进行图形运算，并且在一些场合下，图形运算的效率相当低下。

[图形处理器GPU]

为了弥补CPU处理图形的缺陷，GPU便诞生了，GPU是针对图形运算而生的处理器，如今的主流游戏PC都会有一个高性能的GPU。具体的概念笔者就不在这里提了，相信搞计算机的都会对其了解一二。

[在编程中使用GPU运算资源]

在大多数开发场合下，无论是C还是Java，我们敲的代码都无疑地运行在CPU之上，那我们怎样才能显示地调用GPU的资源呢？在OPENGL中，除了常用的固定管线式编程外，还提供了更为灵活的功能，它允许开发者使用着色语言（GLSL，一种类C语言）编写名为“着色器”的程序，经编译运行于GPU之上并接管一部分GPU内置功能。

[Opengl着色器简述]

介于笔者履历有限，这里只做简单介绍，详细概念可以参考《Opengl着色语言》一书，它对着色器和GLSL有着非常详细的介绍。

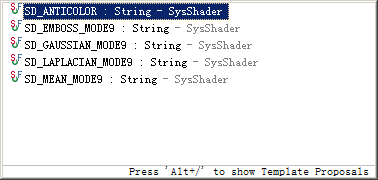
一个Opengl着色器（下简称着色器）由一个顶点着色器与片段着色器构成，它们负责的工作大部分是不同的，仅有少部分交集。

Opengl为着色器的校验、编译、连接和设置参数提供了完整的API，只要显卡支持，在Opengl上下文中即可由源文件得到着色器程序，无需其他环境。

编写着色器的目的，是为了实现诸如模糊、雾化、发光等固定管线不易叙述的功能，进而丰富图像的表现能力。很多游戏引擎将Opengl中与着色器相关的API包装到了上层，我在设计时也采用了这种方案。

[使用系统着色器]

在GEiv中，内置了5个系统着色器，它们是在设计之初进行着色器测试时所保留下来的，并不是由于“常用”（笔者感觉着色器的编写往往很有针对性，一般很难有常用一说）。



依次介绍：

SD\_ANTICOLOR：将目标色改为反色

SD\_EMBOSS\_MODE9：浮雕效果，后面的MODE9表示它使用的是3X3的模板，可以适应一般的需求，若要提高精确度，可以将模板扩充至4X4\5X5等。

SD\_GAUSSIAN\_MODE9：高斯模糊

SD\_LAPLACIAN\_MODE9：锐化

SD\_MEAN\_MODE9：均值化

[设置着色器]

在GEiv中，使用着色器的对象是图元，使用其setShaderProgram方法将着色器绑顶到对应图元上，如：

Obj T = UES.creatObj(UESI.BGIndex);

T.addGLImage(0, 0, "./mdls.jpg");

T.setShaderProgram(SysShader.SD\_GAUSSIAN\_MODE9);

[设置Uniform参数]

Uniform修饰词（不是数据类型）指明一个GLSL中的变量是外部传入的，并且在整个着色器执行过程中不会改变（区别于attribute），例如对于实现模糊变化来讲，“模糊度”这个变量属于Uniform型无疑。

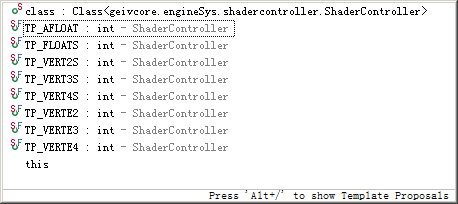
在GLSL中，数组有很多存在形式，float[]可以表示一维线性浮点数组，vect2则表示了一个点对，而vec2[]则表示了一维线性浮点“数对”组，类似的还有vect3\4等等这样在java中没有现成的对应结构的数据，也就是说，设置Uniform参数时必须给定我们的数据源（一维线性数组）和映射到GLSL中变量的方式。

在GEiv的图元类中，setShaderUniform(String uniformName, Object value, int TPFLAG)方法可以设置该图元上绑定的着色器参数，这个方法的参数依次是：

uniformName是着色器中对于的变量名称。

value是数据源，目前只支持float[]、Float[]或单个的Float型。

TPFLAG 是指定的映射方式，你可以直接从ShaderController的静态值部分直接引用。它可以是下列值之一：



AFLOAT与FLOATS分别对应单个float值与一维float数组。

VERTXS表示由X个点对组成的数组，即该数组每个元素包含一个X维度向量。所给数据源必须是一个X整数倍大小的一维线性数组，该数组会按照顺序依次填充这些X维度向量，例如使用TP\_VERT2S时，数据源一维数组的前两个元素构成了着色器向量数组的第一个向量元素。

VERTEX表示一个X维度向量。要求给定的一维数据源数组大小必须为X。

[自定义用户着色器]

除了使用SysShader中现成的着色器外，还可以使用用户自定义的着色器程序。

在引擎的句柄UESI中，包装并简化的Opengl产生着色器的API：

UESI.createShaderProgram(String spName, String vpPath, String fpPath);

参数介绍：

spName是用户为着色器起的名字。

vpPath是顶点着色器的源文件路径，如”./vp.txt”。

fpPath是片段着色器源文件路径。

之后我们只需要使用设置着色器的API对图元绑定spName就行了。

[实例-渐变高斯模糊]

您可以再[GitHub]Sample\Sample-Shader下找到这个例子及使用的资源。

Main.java:

package com.geiv.test;

import geivcore.R;

import geivcore.UESI;

public class Main{

public static void main(String[] args) {

UESI UES = new R();

new Guassion(UES);

}

}

Guassion.java:

package com.geiv.test;

import geivcore.SerialTask;

import geivcore.UESI;

import geivcore.engineSys.shadercontroller.ShaderController;

import geivcore.engineSys.shadercontroller.SysShader;

import geivcore.enginedata.canonical.CANExPos;

import geivcore.enginedata.obj.Obj;

import java.awt.event.KeyEvent;

import java.util.Arrays;

public class Guassion implements SerialTask{

UESI UES;

float[] g\_aryVerticalOffset;

float[] vertstatic;

float bur = 600f;//bur作为模糊度因子，使用静态偏移量(vertstatic)除以模糊度因子得到不同的偏移量参数(g\_aryVerticalOffset)，由于GLSL上下文是使用的OPENGL坐标式，与GEiv有所不同，因此初始因子经计算转换后定为600f

Obj T;

public Guassion(UESI UES) {

this.UES = UES;

T = UES.creatObj(UESI.BGIndex);

T.addGLImage(0, 0, "./mdls.jpg");

T.setShaderProgram(SysShader.SD\_GAUSSIAN\_MODE9);//设置着色器

T.setShaderUniform(SysShader.NA\_WEIGHTARGS,

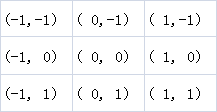
SysShader.BR\_GAUSSIAN\_MODE9, ShaderController.TP\_FLOATS);//设置Unfiorm参数，其中规格化后的权重模板及其在GLSL上下文中的名称已经给定，它们可以由SysShader直接引用。

vertstatic = new float[]{-1,-1, 0,-1, 1,-1

-1, 0, 0, 0, 1, 0

-1, 1, 0, 1, 1, 1};

其实它是这样的模板↓



g\_aryVerticalOffset = Arrays.copyOf(vertstatic, vertstatic.length);

for(int i = 0;i < g\_aryVerticalOffset.length;i++)

{

g\_aryVerticalOffset[i] = vertstatic[i]/bur;//使用模糊度因子处理

}

T.setShaderUniform(SysShader.NA\_OFFSETARGS, g\_aryVerticalOffset,

ShaderController.TP\_VERT2S);//设置Uniform偏移量

T.setPosition(CANExPos.POS\_CENTER);

T.show();

UES.addSerialTask(this);

}

@Override

public void Serial(int clock) {

//挂载一个扫描式的键盘输入，当Z键按下时bur增加，重置偏移量并重设Uniform。当X键按下时则相反。

if(UES.getKeyStatus(KeyEvent.VK\_Z))

{

T.setShaderUniform(SysShader.NA\_OFFSETARGS,g\_aryVerticalOffset,ShaderController.TP\_VERT2S);

bur+=4f;

for(int i = 0;i < g\_aryVerticalOffset.length;i++)

{

g\_aryVerticalOffset[i] = vertstatic[i]/bur;

}

}

else if(UES.getKeyStatus(KeyEvent.VK\_X))

{

T.setShaderUniform(SysShader.NA\_OFFSETARGS,g\_aryVerticalOffset,ShaderController.TP\_VERT2S);

bur-=4f;

for(int i = 0;i < g\_aryVerticalOffset.length;i++)

{

g\_aryVerticalOffset[i] = vertstatic[i]/bur;

}

}

}

}

除此之外，还有高斯模糊着色器的两部分

顶点着色器：

attribute float sys\_pIndex;

void main(void)

{

gl\_TexCoord[0] = gl\_MultiTexCoord0;

gl\_Position = ftransform();

}

↑这个顶点着色器没有写任何实质性的功能，它被很多片段着色器共用。

高斯-片段着色器:

const int g\_iWeightNumber = 9;//权重模板大小

uniform sampler2D g\_FilterTexture;//我们的纹理

uniform float g\_aryWeight[g\_iWeightNumber];//权重数组

uniform vec2 g\_aryOffset[g\_iWeightNumber];//偏移量-即使使用9个格子，也没有规定必须是相邻的格子；偏移量叙述着权重模板格子间的实际像素距离，取值越高，模糊度越高。

void main(void)

{

vec4 vec4Sum = vec4(0.0);

for(int i = 0; i < g\_iWeightNumber; ++i)

{

vec4Sum += texture2D(g\_FilterTexture, gl\_TexCoord[0].st + g\_aryOffset[i])\*g\_aryWeight[i];//这里进行实质的计算过程，也就是替换原先我们让CPU挨个计算像素点的部分。gl\_TexCoord[0]是绑定的0号纹理（Opengl中图形是可以绑定多个纹理的），它的st就是位置，一个vec2类型量，位置与偏移量进行加和，得到偏移位置。

之后根据纹理和偏移位置取出对应的rgba四维向量，它是一个vec4类型。

进一步，将我们的vec4乘以规格化后的权重，并加和到vec4Sum上，由于是规格化后的权重，因此RGBA都不会越界。

}

gl\_FragColor = vec4Sum;// 最后，我们将这个计算完毕的RGBA结果填入纹理中。

}

执行结果（按住X键时）：







↑可以观察到由清晰到模糊的动态过程。