整个简略过程:

1.controller中：

self.context = [[EAGLContext alloc] initWithAPI:kEAGLRenderingAPIOpenGLES3]; GLKView \*view = (GLKView \*)self.view; view.context = self.context;

2.窗口：esCreateWindow( esContext, ”test”, 320, 240, ES\_WINDOW\_RGB); 绘制/回调: esRegisterDrawFunc(esContext, Draw);

3.编译加载着色器: glCreateShader(GL\_VERTEX\_SHADER); glShaderSource (shader,1,&shaderSrc,NULL); glCompileShader(shader); glGetShaderiv(shader, GL\_COMPILE\_STATUS, &complied);

4.创建程序连接着色器: programObject = glCreateProgram(); glAttachShader (programObject, vertexShader); glLinkProgram(programObject);

5.在Draw函数中：

glviewport；

在每个帧的开始glClear(GL\_COLOR\_BUFFER\_BIT)清除颜色缓冲区, 清除颜色由glClearColor指定；

glUseProgram (programObject);

将数据加载到顶点属性0: glVertexAttribPointer(0,3,GL\_FLOAT,GL\_FALSE,0, vVertices);

绘制图元：glDrawArrays(GL\_TRIANGLES,0,3);

EGL 写在#ifndef \_\_APPLE\_\_ 等等里面，不然会报编译错误

窗口系统接口EGLDisplay display = eglGetDisplay( EGL\_DEFAULT\_DISPLAY )

初始化EGL：eglInitialize(EGLDisplay display, EGLint \*egl主版本号, EGLint \*egl次版本号)

确定渲染表面类型和配置：eglGetConfigs(EGLDisplay display, EGLConfig \*configs, EGLint 指定configs的大小, EGLint \*返回configs的大小)

查询EGLConfig属性：eglGetConfigAttrib(EGLDisplay display, EGLConfig config, EGLint 返回的特定属性, EGLint \*返回值)

EGL选择配置: eglChoose(EGLDisplay display, const EGLint \*attribList, EGLConfig \*configs, EGLint maxReturnConfigs, EGLint \*numConfigs)

创建EGL窗口：EGLSurface eglCreateWindowSurface(EGLDisplay display, EGLConfig config, EGLNativeWindowType 原生窗口, NULL)

屏幕外渲染: Pbuffer

创建渲染上下文：EGLContext eglCreateContext(EGLDisplay display, EGLConfig config, EGL\_NO\_CONTEXT, const EGLint \*attribList)

关联渲染表面和上下文：eglMakeCurrent(EGLDisplay display, EGLSurface 绘图表面, EGLSurface 读取表面, EGLContext context)

20示例Hello\_Triangle

创建着色器: GLuint glCreateShader(GL\_VERTEX\_SHADER)

提供源码：glShaderSource(GLuint shader, 1, &字符串数组指针, NULL)

编译: glCompileShader(GLuint shader)

查询编译结果: glGetShaderiv(GLunit shader, GLenum 系统常量, GLint \*查询结果)

日志信息：glGetShaderInfoLog(GLuint shader, GLsizei maxLength, NULL, GLchar \*infoLog)

创建链接程序: GLuint glCreateProgram()

连接着色器: glAttachShader(GLuint program, GLuint shader)

链接程序：glLinkProgram(GLuint program)

设置为活动程序(在Draw方法中)：glUseProgram(GLuint program)

统一变量uniform: 程序通过API传递给着色器的只读常数值的变量

获取统一变量(page55)：glGetActiveUniform glGetActiveUniformsiv

查询统一变量的位置：glGetUniformLocation

57加载统一变量：glUniform1f(GLint location, GLfloat x)

列/行跨距(GL\_UNIFORM\_MATRIX\_STRIDE): 矩阵中两个向量之间的偏移量

std140统一变量块布局(page61):

检索统一变量块索引：glGetUniformBlockIndex(GLuint program, const GLchar \*blockName)

用index获取name：glGetActiveUniformBlockName(GLuint program, GLuint index, GLsizei 名称数组中的字符数, GLsizei \*将写入名称数组中的字符数, GLchar \*blockName)

用pname和index获取相应结果：glGetActiveUniformBlockiv(GLuint program, GLuint index, GLenum pname, GLint \*params)

将索引和程序中的统一变量块绑定点关联glUniformBlockBinding(GLuint program, GLuint blockIndex, GLuint 统一变量缓冲区对象绑定点)

将统一变量缓冲区对象绑定到GL\_UNIFORM\_BUFFER目标和程序中的统一变量块绑定点(page63)：glBindBufferRange(GL\_UNIFORM\_BUFFER或GL\_TRANSFORM\_FEEDBACK\_BUFFER, GLuint index, GLuint 缓冲区对象句柄, GLintptr 以字节数计算的缓冲区对象起始偏移, GLuint 可以从缓冲区对象读取或者写入缓冲区对象的数据量)

检索程序二进制代码：glGetProgramBinary(GLuint program, GLsizei 可以写入binary的最大字节数, GLsizei \*二进制数据的字节数, GLenum 供应商专用二进制格式标志, GLvoid \*着色器编译器生成的二进制数据指针)

将程序二进制码保存到文件系统或读回OpenGL ES实现(page65): glProgramBinary(GLuint program, GLenum 供应商,const GLvoid \*binary, GLsizei 二进制数据的字节数)

布局限定符: layout (row\_major) uniform layout (std140) uniform

(page80)平滑/插值着色smooth 平面着色flat 质心采样centroid

预处理命令： 扩展 #extension extension\_name : behavior

不变性: invariant 防止伪像。由于优化,多个着色器的等价计算不能保证完全相同的结果

指定顶点属性: glVertexAttrib1f(GLuint index, GLfloat x)

89顶点数组：glVertexAttribPointer(GLuint index, GLint 分量数量, GLenum 数据格式, GLboolean 非浮点转化为浮点是否该规范化, GLsizei 跨距, const void \*客户端顶点数组的指针或顶点缓冲区对象内的偏移量)

91结构数组：指针访问高效，缺点是跨距更新。 数组结构：单独的缓冲区

93启用通用顶点属性数组: glEnableVertexAttribArray(GLuint index)

96获取活动顶点属性数量：glGetProgramiv(program, GL\_ACTIVE\_ATTRIBUTES, &numActiveAttribs)

获取活动顶点属性列表和它们的数据类型：glGetActiveAttrib(GLuint program, GLuint index, GLsizei 可以写入结果的最大字符数, GLsizei \*返回写入结果的字符数, GLenum \*返回的属性类型, GLint \*返回数组大小, GLchar \*属性变量名称)

98将通用顶点属性索引绑定到顶点着色器的变量: glBindAttribLocation(GLuint program, GLuint index, const GLchar \*name)

查询绑定：GLint glGetAttribLocation(GLuint program, const GLchar \*name)

100 glGenBuffers(GLsizei 返回的缓冲区对象名称数量, GLuint 分配的缓冲区对象指针)

glBindBuffer(GLenum target, GLuint 被绑定的缓冲区对象指针)

glBufferData(GLenum target, GLsizeiptr 缓冲区数据存储大小以字节数表示, const void \*缓冲区数据指针, GLenum 如何使用缓冲区存储数据的提示)

glBufferSubData(GLenum target, GLintptr 缓冲区数据存储中的偏移, GLsizeiptr 被修改的数据存储字节数, const void \*缓冲区数据指针)

99保存组成图元的元素索引GL\_ELEMENT\_ARRAY\_BUFFER

102 demo: VertexBufferObjects用缓冲区和不用缓冲区

106 VAO提供包含在顶点数组/顶点缓冲区对象配置之间切换所需要的所有状态的单一对象（只需要在Draw方法中Bind，而不需要再写缓冲区相关代码）

创建顶点数组对象glGenVertexArrays(GLsizei 顶点数组元素数量, GLuint \*顶点数组指针)

绑定顶点数组对象glBindVertexArray(GLuint 被指定为当前顶点数组对象的对象), 还是需要先bufferData

107 demo：VertexArrayObjects顶点数组对象绘图

109映射缓冲区对象：返回GPU存储缓冲区地址空间的直接指针

返回指向所有或者一部分缓冲区对象数据存储的指针void \*glMapBufferRange(GLenum target, GLintptr 缓冲区数据存储中的偏移量以字节数计算, GLsizeiptr 需要映射的缓冲区数据的字节数, GLbitfield 访问标志的位域组合) 还要结合memcpy??

更新部分映射：glFlushMappedBufferRange(GLenum target, GLintptr 缓冲区数据存储中的偏移量, GLsizeiptr 从偏移点开始刷新刷新的缓冲区字节数)

111 demo：映射缓冲区对象MapBuffers

112复制缓冲区对象: glCopyBufferSubData(GLenum 读取的缓冲区对象, GLenum 写入的缓冲区对象, GLintptr 需要复制的读缓冲偏移量, GLintptr 需要复制的写缓冲偏移量, GLsizeiptr 复制的字节数)

116 图元glDraw…等等

画线段：glLineWidth(GLfloat 线宽)

查询支持的线宽: GLfloat lineWidthRange[2] glGetFloatv (GL\_ALIASED\_LINE\_WIDTH\_RANGE, lineWidthRange);

绘图默认原点是左下，点精灵是左上

117绘制图元glDrawArrays(GLenum mode, GLint 起始顶点索引, GLsizei 顶点数量)

glDrawElements(GLenum mode, GLsizei count, GLenum 索引数组中元素索引类型, const GLvoid \*索引数组的指针)

glDrawRangeElements(GLenum mode, GLuint 索引数组中的最小数组索引, GLuint 索引数组中的最大数组索引, GLsizei 索引数量, GLenum 索引类型, const void \*索引数组)

119图元重启：渲染不相连图元 启用glEnable(GL\_PRIMITIVE\_FIXED\_INDEX)

驱动顶点: 没有插值的时候按哪个顶点输出

120 ??几何形状实例化：glDrawArraysInstanced(GLenum mode, GLint 顶点数组中的起始顶点索引, GLsizei 索引数量, GLsizei 图元实例数量)

121 demo: 几何形状实例化Instancing

glDrawElementsInstanced(GLenum mode, GLsizei 索引数量, GLenum 元素索引类型, const GLvoid \*元素索引数组指针, GLsizei 图元实例数量)

访问实例: glVertexAttribDivisor(GLuint 着色器索引, GLuint 对应顶点位置的实例数量)

?? 矩阵要分多个顶点着色器索引加载

视口：glViewport(GLint 视口左下角x像素, GLint y, GLsizei w像素, GLsizei h)

126 ??设置深度范围值: glDepthRangef(GLclampf n, GLclampf f)

指定正面三角形的方向:glFrontFace(GLenum GL\_CW/GL\_CCW)

指定要被剔除的三角形的面:glCullFace(GLenum mode)

启用剔除glEnable ( GL\_CULL\_FACE );

demo: 立方体多面纹理 Simple\_TextureCubemap

??为避免伪像,多边形偏移量:glPolygonOffset(GLfloat 因数, GLfloat 单位数)

130遮挡查询: glGenQueries(GLsizei 生成的查询名称对象的数量, GLuint \*存储查询名称对象的列表) glBeginQuery(GLenum target, GLuint 查询对象的名称)

检索查询对象的结果: glGetQueryObjectuiv(GLuint 查询对象的名称, GLenum 要检索的查询对象参数, GLuint \*params)

134内建变量：

gl\_VertexID输入变量，保存顶点的整数索引

gl\_InstanceID输入变量，保存实例化绘图调用中图元的实例编号

gl\_Position输出顶点位置的裁剪坐标

gl\_PointSize用于写入以像素表示的点精灵尺寸

gl\_FrontFacing根据顶点着色器生成的位置值和渲染的图元类型生成

内建统一状态：uniform gl\_DepthRangeParameters gl\_DepthRange;深度范围

内建常量：

const mediump int gl\_MaxVertexAttribs >= 16可以指定的顶点属性的最大数量

gl\_MaxVertexUniformVectors>=256可以使用的vec4统一变量项目的最大数量

gl\_MaxVertexOutputVectors>=16输出向量的最大数量

gl\_MaxVertexTextureImageUnits>=16可用纹理单元的最大数量

gl\_MaxCombinedTextureImageUnits>=32顶点和片段着色器中可用纹理单元最大数量的总和

着色器语言中尽量用常量或变量代替字面值，防止字面值计算多次，超出支持

裁剪esFrustum esPerspective

141 ??光

146 ??生成纹理坐标

146 ??顶点蒙皮

150 变换反馈glTransformFeedbackVaryings(GLuint program, GLsizei 顶点输出变量的数量, const char \*\*输出变量字符串组成的数组, GLenum 捕捉输出变量的模式)

进入变换反馈模式glBeginTransformFeedback(GLenum 图元输出类型)

152 ??顶点纹理

163创建纹理对象:glGenTextures(GLsizei 要生成的纹理对象数量, GLuint 保存纹理对象ID的数组)

绑定纹理对象:glBindTexture(GLenum target, GLuint 纹理对象句柄)

加载图像数据:glTexImage2D(GLenum target, GLint mip级别, GLenum 纹理存储的内部格式, GLsizei 图像的像素宽度, GLsizei height, GLint 0, GLenum 输入的纹理数据格式, GLenum 输入像素数据的类型, const void\* 包含图像的实际像素数据(自定义的颜色数组或fread读取的char \*类型))

着色器中的纹理统一变量(采样器): uniform sampler2D s\_texture

内建函数outColor = texture(s\_texture, 片段着色器输入的v\_texColor(由顶点着色器输出)) 详见335，可以旋转下当前页pdf

166设置解包对齐glPixelStorei(GLenum 像素存储类型, GLint 指定包装或解包选项)

168 mip贴图：主要体现在缩小的时候，一个位置的像素根据上一个放大级别相同位置的四个像素的平均值给定，防止伪像

过滤(缩小/放大)模式glTexParameteri(GLenum target, GLenum pname, GLint param)

demo:不同的纹理包装模式 TextureWrap 即glTexParameteri最后一个参数:GL\_REPEAT重复纹理 GL\_CLAMP\_TO\_EDGE限定读取纹理的边缘 GL\_MIRRORED\_REPEAT重复纹理和镜像

171 demo:过滤的纹理参数设置的影响: MipMap2D

xyzw中的w是齐次坐标， w=0时两条平行直线在无穷远交汇于一点 笛卡尔坐标系的X=x/w

glUniform1i ( userData->samplerLoc, 0 );//这句话的意思是设置采样器纹理单元的索引值为0，默认也是0，这是为了可能会有多个采样器纹理单元

171自动mip贴图生成glGenerateMipmap(GLenum target)

172 ??纹理坐标包装GL\_TEXTURE\_WRAP\_T，对超出纹理

demo: 针对超出纹理 TextureWrap

181激活纹理单元glActiveTexture(GLenum texture)

183立方图纹理：3个分量，采样器类型samplerCube

3D纹理:glTexImage3D(GL\_TEXTURE\_3D/GL\_TEXTURE\_2D\_ARRAY), GLint mip级别, GLenum 纹理存储的内部格式, GLsizei 像素图像宽度, GLsizei height, GLsizei 3D纹理的切片数量, GLint 0, GLenum 纹理数据格式, GLenum 像素数据的类型, const void\* 像素数据)

186压缩纹理glCompressTexImage2D(GLenum target, GLint mip贴图级别, GLenum 纹理存储的内部格式, GLsizei 像素图像宽度, GLsizei height, GLint 0, GLsizei 字节数图像大小, const void\* 像素数据)

glCompressTexImage3D(GLenum target, GLint mip贴图级别, GLenum 纹理存储的内部格式, GLsizei width, GLsizei height, GLsizei 像素图像深度, GLint 0, GLsizei 字节数图像大小, const void\* 像素数据)

查询支持的压缩格式glGetIntergerv(GL\_COMPRESSED\_TEXTURE\_FORMATS)

188更新图像子区域:glTexSubImage2D(GLenum target, GLint mip级别, GLint 开始更新的纹素x索引, GLint yoffset, GLsizei 更新的图像子区域宽度, GLsizei height, GLenum 输入纹理数据格式, const void\* 像素数据)

更新压缩的2D纹理图像子区域(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, Glsizei height, Glenum format, const void\* pixels)

3D纹理更新子区域glTexSubImage3D glCompressTexSubImage3D

191读取颜色缓冲区glReadBuffer(GLenum mode)

复制到纹理glCopyTexImage2D(GLenum target, GLint mip级别, GLenum 图像的内部格式, GLint 读取的帧缓冲区矩形左下角的x窗口坐标, GLint y, GLsizei 读取区域的宽, GLsizei height, GLint 0)

194采样器对象glGenSamplers(GLsizei 采样器对象数量, GLuint \*采样器对象数组)

绑定采样器对象glBindSampler(GLenum 采样器对象绑定到的纹理单元, GLuint 所要绑定的采样器对象的句柄)

设置采样器对象的状态glSampleParameteri(GLuint 采样器对象, GLenum pname, GLint 设置的纹理参数值)

196不可变纹理(加载数据之前指定纹理的格式和大小): glTexStorage2D(GLenum target, GLsizei mip贴图级别数量, GLenum 纹理存储内部格式, GLsizei width, GLsizei height, )

glTexStorage3D(…同2D…, GLsizei 基本图像深度)

201片段着色器内建变量：gl\_FragCoord窗口相对坐标(x,y,z,1/w)

gl\_FrontFacing片段是正面图元的一部分时为true，否则为false

gl\_PointCoord保存点精灵的纹理坐标

gl\_FragDepth覆盖片段的固定功能深度值

内建常量:  
gl\_MaxFragmentInputVetors>=15 片段着色器输入的最大数量

gl\_MaxTextureImageUnits>=16 可用纹理图像单元的最大数量

gl\_MaxFragmentUniformVectors>=224 vec4统一变量项目的最大数量

gl\_MaxDrawBuffers>=4 多重渲染目标(MRT)的最大支持数量

gl\_MinProgramTexelOffset/gl\_MaxProgramTexelOffset通过内建ESSL函数texture\*Offset()偏移参数支持的最大和最小偏移量

203 demo：多重纹理 MultiTexture

204 雾化 20 alpha测试(使用Discard) 根据判断在平面上/下裁剪

211请求缓冲区。在esCreateWindow最后一个参数, ES\_WINDOW\_RGB颜色缓冲区，ES\_WINDOW\_DEPTH深度缓冲区，ES\_WINDOW\_STENCIL

212清除缓冲区glClear(GLbitfield GL\_mask)

设置清除值glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)

设置清除深度缓冲区glClearDepthf(GLfloat [0,1]内的值)

设置清除模版缓冲区glClearStencil(GLint [0,2n-1]内的值) n是模版缓冲区可用的位数

清除绘图缓冲区:glClearBufferiv(GLenum buffer,GLint 要清除的绘图缓冲区名称, const GLint 用于清除缓冲区的数组或数值指针)

控制RGB的写入glColorMask(GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha)

在渲染透明物体之前，调用glDepthMask(GL\_FALSE)来禁用深度缓冲区的写入

模版缓冲区的写入:glStencilMask(GLuint 说明缓冲区中哪些位可以修改的位掩码[0,2n-1]) n是模版缓冲区位数

214根据图元的面顶点顺序设置位掩码glStencilMaskSeparate(Glenum face, GLuint [0,2n-1])

214 测试目的:选择哪些片段成为像素并影响最终的图像

启用片段测试glEnable(GL\_DEPTH\_TEST或其他)

裁剪测试glScissor(Glint x, GLint y, GLsizei width, GLsizei height) glEnable(GL\_SCISSOR\_TEST)

215 模版缓冲区测试:glStencilFunc(Glenum 模版测试的比较函数, GLint 模版测试的比较值, GLuint 指定在与参考值比较之前与模版缓冲区中各位进行按位与运算的掩码) glStencilFuncSeparate

glStencilOp(GLenum 片段不能通过模版测试时应用到模版位的操作, GLenum 通过模版测试但是没有通过深度测试时应用的操作, GLenum 片段在模版和深度测试中都通过时应用的操作)

219深度缓冲测试glDepthFunc(GLenum func)

220颜色混合:glBlendFunc(GLenum 输入片段的混合系数, GLenum 目标像素的混合系数)

glBlendFuncSeparate(GLenum 输入rgb系数, GLenum 目标rgb系数, GLenum 输入Alpha系数, GLenum 目标Alpha系数)

指定颜色glBlendColor(rgba)

混合颜色glBlendEquation(Glenum mode)

222 ??多重采样，抗锯齿glSampleCoverage(GLfloat [0,1], GLboolean 确定掩码值后,掩码中的所有位是否将被反转)

质心采样:确保片段数据从落在三角形内部的样本中选取

顶点着色器中smooth centroid out vec3 v\_color

223在帧缓冲区读取数据glReadPixels(Glint 左下角x, GLint y, GLsizei width, Glsizei height, GLenum 返回的像素格式, GLenum 返回的像素数据类型, Glvoid \*读取的值)

226多重渲染目标MRT方法

227为渲染指定颜色附着:glDrawBuffers(GLsizei bufs中的缓冲区数量, const Glenum\* 指定片段颜色或者数据值将要写入的缓冲区的符号常量数组)

233分配渲染缓冲区对象glGenRenderbuffers(GLsizei 渲染缓冲区对象名称的数量, GLuint \*渲染缓冲区名称的数组)

分配帧缓冲区对象名称glGenFramebuffers(GLsize 帧缓冲区对象名称的数量, GLuint \*帧缓冲区数组)

234设置帧缓冲区glBindRenderbuffer(GL\_RENDERBUFFER, GLuint 渲染缓冲区对象)

指定保存在渲染缓冲区的图像大小和格式glRenderbufferStorageMultisample (GL\_RENDERBUFFER, GLsizei 用于渲染缓冲区对象存储的样本数, GLenum 可用于颜色缓冲区、深度缓冲区或者模版缓冲区的格式, GLsizei 以像素数表示的渲染缓冲区宽度, GLsizei height)

237设置帧缓冲区对象glBindFramebuffer(GL\_READ\_FRAMEBUFFER/ GL\_DRAW\_FRAMEBUFFER/GL\_FRAMEBUFFER, GLuint 帧缓冲区对象名称)

238连接渲染缓冲区作为帧缓冲区附着glFramebufferRenderbuffer (GL\_READ\_FRAMEBUFFER/GL\_DRAW\_FRAMEBUFFER/GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENTi/GL\_DEPTH\_ATTACHMENT/GL\_STENCIL\_ATTACHMENT/GL\_DEPTH\_STENCIL\_ATTACHMENT, GL\_RENDERBUFFER, GLuint 作为用作附着的渲染缓冲区对象)

238连接2D纹理作为帧缓冲区附着glFramebufferTexture2D(GLenum target, GLenum attachment, GLenum 纹理目标, GLuint 纹理对象, GLint 纹理图像的mip级别)

240连接3D纹理glFramebufferTextureLayer(Glenum target, GLenum attachment, GLuint 纹理对象, GLint 纹理对象的mip级别, GLint 纹理图像层次)

241验证帧缓冲区对象是否完整GLenum glCheckFramebufferStatus(GLenum target)

242帧缓冲区位块传送glBlitFramebuffer(GLint srcX0, GLint srcY0, GLint srcX1, GLint srcY1, GLint dstX0, GLint dstY0, GLint dstX1, GLint dstY1, GLbitfield 缓冲区被复制的标志, Glenum 图像被拉伸时应用的插值GL\_NEAREST/GL\_LINEAR)

demo: MRTs

244使帧缓冲区的像素子区域实效(降低内存带宽需求,省电) glInvalidateFramebuffer(GLenum target, GLsizei attachments列表中的附着数量, const GLenum \*attachments数组, GLint x, GLint y, GLint width, GLint height)

253在GL命令流中插入栅栏命令并创建同步对象 GLsync glFenceSync(GLenum 向同步对象发送信号必须符合的条件GL\_SYNC\_GPU\_COMMANDS\_COMPLETE, GLbitfield 控制同步对象行为的标志按位组合0)

删除同步对象glDeleteSync(GLsync 要删除的同步对象)

阻塞客户并等待一个同步对象收到信号 GLenum glClientWaitSync(GLsync 等待其状态的同步对象, GLbitfield 控制命令刷新行为的位域GL\_SYNC\_FLUSH\_COMMANDS\_BIT, GLuint64 等待同步对象获得信号的超时时间)

glWaitSync立即返回且阻塞GPU，直到同步对象收到信号

267混合效果glEnable(GL\_BLEND) glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE)这里片段着色器中产生的Alpha值与片段的颜色进行调制。然后，该值被加到片段目标中保存的值，结果得到粒子系统的相加混合效果

demo ParticleSystem 粒子爆炸效果

271 确保传入的输出变量用于变换反馈glTransformFeedbackVaryings()

demo ParticleSystemTransformFeedback 启用变换反馈的粒子效果//page

272启用变换反馈渲染glBeginTransformFeedback(GL\_POINTS)

300字符串查询const GLubyte\* glGetStringi(GLenum 指定要返回的参数, GLuint 指定要返回的字符串索引)

301查询限制glGetBoolean glGetFloatv glGetIntegerv glGetInteger64v

306修正各种功能的操作glHint,指定3D图形算法偏重视觉质量还是速度

307 查询是否是有效实体glIs… 如glIsTexture

308与着色器有关的查询glGetAttachedShaders(GLuint program, GLsizei 返回的最大着色器名称的数量, GLsizei \*返回的实际着色器名称数量, GLuint \*着色器名称的数组)

查询着色器源码glGetShaderSource(GLuint shader, GLsizei 着色器源码数组中可用字节数, GLsizei \*着色器字符串长度, GLchar 保存着色器源码的数组)

检索统一变量glGetUniformfv(GLuint program, GLint location, GLfloat \*统一变量的对应类型数组)

309查询着色器语言的范围和精度glGetShaderPrecisionFormat

顶点属性查询glGetVertexAttribPointerv 数据元素的状态glGetVertexAttribfv

纹理状态查询glGetTexParameterfv

采样器查询glGetSamplerParameterfv

311查询异步对象的信息glGetQueryiv

查询同步对象的信息glGetSynciv

顶点缓冲区查询glGetBufferParameteriv glGetBufferPointerv

查询渲染缓冲区glGetRanderbufferParameteriv

查询帧缓冲区glGetFramebufferAttachmentParameteriv

326常用内建函数

339注册键盘输入处理回调函数esRegisterKeyFunc

340加载一个8、24或32位TGA图像esLoadTGA(char \*磁盘上的文件名, int \*以像素表示的加载图像的宽度, int \* height)

为一个球体生成几何形状int ESUTIL\_API esGenSphere(int 球体中的垂直和水平切片数量, float radius, GLfloat \*\* float3位置数组, GLfloat \*\* float3法线数组, GLfloat \*\* float2 texCoords数组, GLuint \*\* 三角形条带索引数组) 返回以GL\_TRIANGLE\_STRIP的形式渲染缓冲区时需要的索引数量

341为立方体生成几何形状int ESUTIL\_API esGenCube(float 立方体大小, GLfloat \*\* vertices, GLfloat \*\* normal, GLfloat \*\* texCoods, GLuint \*\* indices)

生成由三角形组成的方格网int ESUTIL\_API esGenSquareGrid(int 立方体大小, GLfloat \*\* vertices, GLuint \*\* indices)

记录平台调试输出信息void ESUTIL\_API esLogMessage()

342将result表示的矩阵乘以透视投影矩阵,并在result中返回新矩阵 void ESUITL\_API esFrustum(ESMatrix \*result, GLfloat left裁剪平面坐标, GLfloat right, GLfloat bottom, GLfloat top, GLfloat nearZ近深度到裁剪平面的距离, GLfloat farZ)

目的同上void ESUTIL\_API esPerspective(ESMatrix \*result, GLfloat 以度数表示的视野, GLfloat 渲染窗口的纵横比, GLfloat nearZ, GLfloat farZ)

乘以正交投影矩阵esOrtho 乘以比例缩放矩阵esScalse 乘以平移矩阵esTranslate 乘以旋转矩阵esRotate 两矩阵相乘esMatrixMutiply

得到单位矩阵esMatrixLoadIdentity

用眼睛位置、视线向量和上向量生成的一个视图变换矩阵esMatrixLookAt