# 1 学习途径

## 1.1 Qt 官方文档

QtCreator中按F1查阅相应文档

## 1.2 OpenGL官网

www.opengl.org

## 1.3 OpenGL ES及GLSL API文档

https://www.khronos.org/registry/OpenGL-Refpages/es3/

## 1.4 GLSL ES 3.2文档

https://www.khronos.org/registry/OpenGL/specs/es/3.2/GLSL\_ES\_Specification\_3.20.html

## 1.5 GLSL Core 4.6文档

https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.html

## 1.6 OpenGL 官方入门文档

https://www.khronos.org/opengl/wiki/Getting\_Started

## 1.7 常用技术文档

<https://www.opengl.org/archives/resources/code/samples/sig99/advanced99/notes/node1.html>

# 2 OpenGL与一般绘图API的差异

## 2.1 图形特征处理方式不同

OpenGL（包括其他GPU绘图API）采用的是一种大道至简的处理方式：将图形数据进行拆解——通过顶点数据+图元类型（点、线、三角面）来描述任意的图形。普通API则是划分一些基本图形（点，线，圆，矩形...），用最少的特征来进行描述这些图形（比如描述一个圆只需要两个参数：圆的中心坐标和半径）。

## 2.2 硬件不同

OpenGL使用GPU进行图形绘制。普通API使用CPU绘图（一些普通API也会将代码转换成GPU API以通过GPU进行绘图）。

## 2.3 性能差异

GPU绘图明显高于CPU绘图。

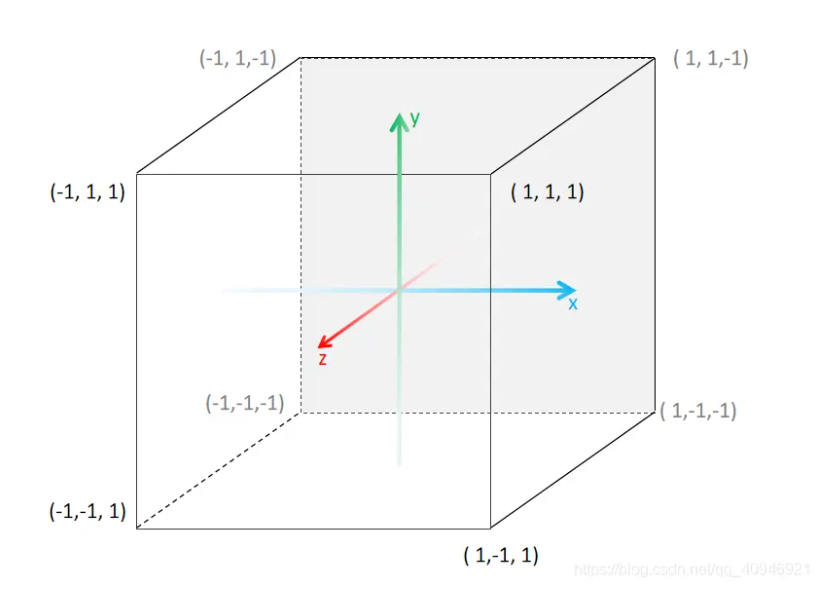
## 2.4 用法差异

普通绘图API可以使用高级函数提供几个简单的参数就能绘制图形，GPU绘图API则要求你必须完成图形渲染中的核心阶段（以圆为例，普通API可以直接通过圆的中心坐标和半径进行绘制，而GPU绘图API关注的是圆上所有点的坐标）。

# 3. OpenGL图形绘制流程

## 3.1 预处理

标准化设备坐标(Normalized Device Coordinates, NDC)，OpenGL为了让坐标运算不受显示器分辨率的影响，将xyz坐标标准化到了[-1,1]之间。任何落在范围外的坐标都会被丢弃/裁剪，标准化设备坐标称为顶点数据（Vertex Data）。



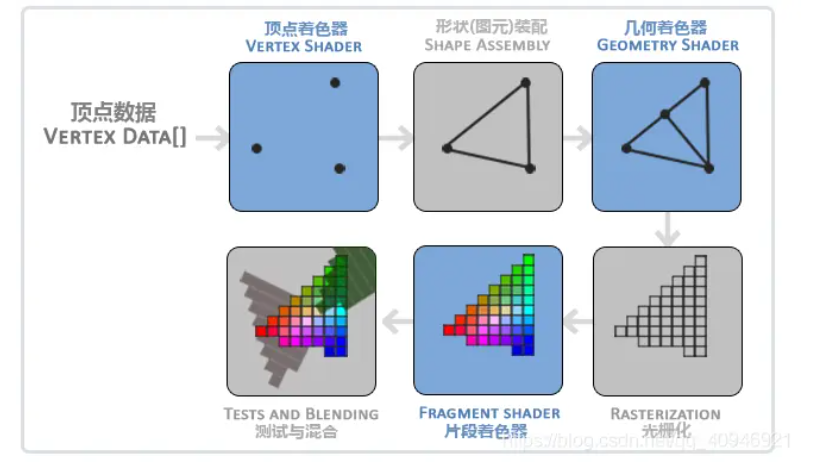
标准化设备坐标

顶点数据并不要求一定有顶点坐标，它可以是顶点的颜色，法向量，纹理坐标等一系列用来绘制顶点的数据的统称。这些顶点数据将会从CPU被传入到GPU中，传输是一个开销非常大的过程，因此传输的数据越少越好，

顶点着色器，几何着色器都会对数据进行处理，真正使用这些数据是从光栅化阶段开始，也就是说你可以在数据处理的过程中再生成顶点数据，（比如你要画圆的话，可以只传入一个顶点坐标和一个半径长度，在之前的处理阶段生成圆形的顶点数据）。

## 3.2 图形渲染管线

OpenGL的图形绘制流程称为图形渲染管线，包括以下六个处理阶段，将预处理中标准化的坐标绘制出来（其中蓝色的部分需要编程实现）。



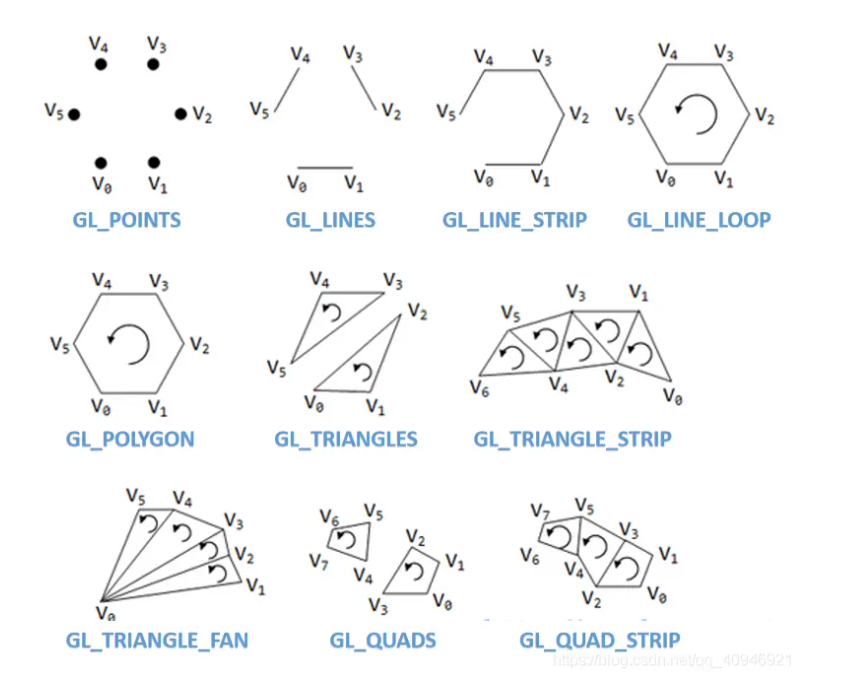
图形渲染管线

### 3.2.1 顶点着色器（Vertex Shader）

加工顶点数据，一般是指顶点数据的变换。shader翻译成着色器很容易让人产生误解：着色器是不是就是用来填充颜色的？并不是！（或许称之为顶点处理器会更好一些）。这一阶段是用来处理一个单独的顶点数据，因此这阶段其实是并行操作（可能有很多顶点数据都同时在使用这个顶点着色器程序进行处理），通过代码实现如何处理上一步的顶点数据，比如平移、旋转、缩放，修改各种顶点数据，比如纹理坐标，法向量等，原则只有一个：需要保证最终只有一个顶点坐标，并且将一些会在其他阶段使用的数据传递输出。

### 3.2.2 图元装配（Shape Assembly）

OpenGL支持四种基本图元类型：点，线，三角形，四边形。并提供了一些扩展图元供使用，如下图：



图元装配

### 3.2.3 几何着色器（Geometry Shader）

根据顶点数据和图元信息生成所需顶点数据，它能起到一定的裁剪作用、同时也能产生比顶点着色器输入更多的基础图元，包括：

（1）生成新的顶点数据（例如，输入一个顶点，能够生成0个以上新的顶点数据）；

（2）修改输出的图元类型（例如，输入图元是点，能够输出线条图元或者三角形图元）。

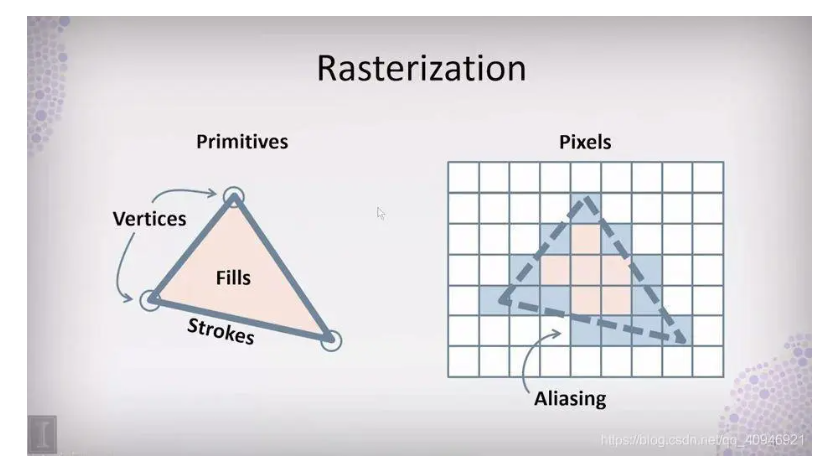
例如：

粒子系统：一个粒子系统往往是通过GL\_POINTS图元来进行绘制，在几何着色器中再生成各种形状的粒子（它可以是2D的，也可以是3D）。

图形扩展：在learn OpenGL教程——加载3D模型的章节中，可以使用几何着色器生成模型各个顶点的法向量（由点→线扩展）。

### 3.2.4 光栅化（rasterization）

将前两阶段处理之后的几何数据经过一系列变换转换为像素。例如，如果我们要绘制的是一个三角形，那么经过前两阶段，现在拥有的是三角形三个顶点的几何数据，光栅化的目的就是将几何数据经过一系列变换后最终转换为像素，从而呈现在显示设备上的过程，如下图：



光栅化

### 3.2.5 片段着色器（Fragment Shader）

计算光栅化中产生片段（“像素”）的颜色，带有一定的裁剪功能（discard）。

### 3.2.6 测试（Tests）与混合（Blending）

测试为筛选并处理从片段处理器中传递过来的像素，测试的目的就是为了通过设置一些测试的方法，按照某种机制来筛选出我们需要的方法。

OpenGL提供了四种测试方法：

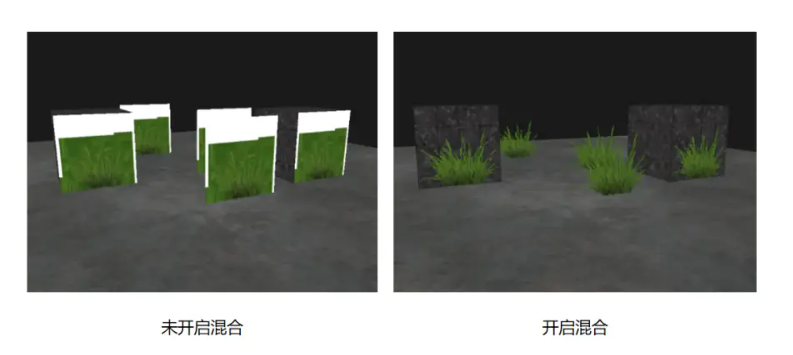
（1）深度测试：当绘制立体物体的时候，使用深度测试可以丢弃物体背面的像素

（2）Alpha测试：可以通过alpha值来丢弃像素。

（3）剪裁测试：通过设置裁剪区域来丢弃像素。

（4）模板测试：模板测试是根据一个模板来进行测试，其目的为了让开发者能够定义自己的测试方法。

混合技术常用于表现物体的透明度，透明的物体的颜色常与位于其之后的物体的颜色进行混合，来达到透明的效果。物体的透明度由颜色的alpha通道的值定义，如下图：



混合模式

# 4 QtOpenGL的使用

## 4.1 创建Opengl窗口

在QT中创建Opengl窗口需新建类继承于QOpenGLWidegt，要将ui界面中的QOpenGLWidegt提升为新建的Opengl子类，并通过函数重写实现QOpenGL提供的三个虚函数来调用窗口。

### 4.1.1 initializeGL()

初始化建立OpenGL的资源和状态。在第一次调用resizeGL()或paintGL()之前调用一次。

### 4.1.2 resizeGL()

设置OpenGL视口，投影等。每当调整Widget的大小时（第一次显示窗口Widget时会调用它，因为所有新创建Widget都会自动获得调整大小的事件）。

### 4.1.3 paintGL()

渲染OpenGL场景，需要更新Widget时就会调用。

## 4.2 搭建渲染管线

需要通过编程实现的着色器程序分为三个部分：顶点着色器、几何着色器和片段着色器，着色器使用着色器语言GLSL(OpenGL Shading Language)编写代码，编译着色器之后，再链接着色器程序即可使用。

一般按照以下顺序进行绘制：

### 4.2.1 绑定着色器程序

### 4.2.2 设置顶点数据的数据来源

一般是调用QOpenGLShaderProgram::setAttribute开头一些方法，有setAttributeValue（装载一个数据），setAttributeArray（装载一组数据），setAttributeBuffer（从缓存区装载一组数据））。

### 4.2.3 调用draw函数

glDraw开头的一些函数，常用的有glDrawArrays（通过数值绘制），glDrawElements（通过索引进行绘制）等）。

## 4.3 OpenGL对象

### 4.3.1 顶点数组对象（Vertex Array Object，VAO）

VAO是一个管理顶点数组的对象，它包含了多个顶点属性指针和绑定到缓冲区对象的顶点数据。作用是存储顶点数据的来源和解析格式。

### 4.3.2 缓冲区对象（Buffer Object，BO）

BO是一种用于存储数据的对象，它可以存储顶点数据（VBO）、索引数据、纹理数据等。作用是一段可供使用的GPU内存。

### 4.3.3 着色器对象（Shader Object）

着色器是一种用于执行图形渲染的程序，它可以编写顶点着色器、片段着色器、几何着色器、计算着色器等不同类型的着色器。

### 4.3.4 纹理对象（Texture Object）

纹理是一种用于存储图像数据的对象，它可以被应用到几何图形上，用于实现贴图的效果。

### 4.3.5 帧缓冲对象（Frame Buffer Object，FBO）

FBO是一种用于渲染到纹理的对象，它可以将渲染结果输出到纹理上，用于实现后期处理和特效。

### 4.3.6 渲染缓冲对象（Render Buffer Object，RBO）

RBO是一种用于存储渲染结果的对象，它可以存储深度信息、模板信息等。

### 4.3.7 查询对象（Query Object）

查询对象可以用于获取 OpenGL 状态或执行操作的结果，例如获取GPU的性能信息、检测渲染完成状态等。

### 4.3.8 同步对象（Sync Object）

同步对象可以用于同步CPU和GPU的操作，例如等待GPU完成渲染操作后再进行下一步操作。