用到的Pyside基本模块介绍：

1、QtCore模块

用于核心的非 GUI 功能函数,用于以下方面:日期、文件和目录、数据结构、数据流、URL、MIME、线程和进程。

2、QtGui模块

用于绘图组件以及与绘图相关的类,比如按钮、窗口、状态栏、工具栏、滑块、位图、颜色、字体等。

3、QtXml模块

用于处理 XML 文件的类,该模块提供了 SAX 和 DOM API 两种 XML 文件处理方式的实现。

4、QtOpenGL 模块

用于渲染使用 OpenGL 库创建的 3D 或 2D 图形。并且它支持 Qt GUI 库和 OpenGL 库的无缝结合。

<https://doc.qt.io/qtforpython/PySide2/QtOpenGL/QGLWidget.html>

3月29日

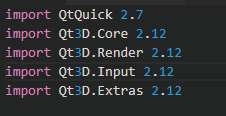
<https://doc.qt.io/qt-5/qt3d-qml.html#qt-3d-extras-module>



3月30日 参加两场考试，所以没更新。

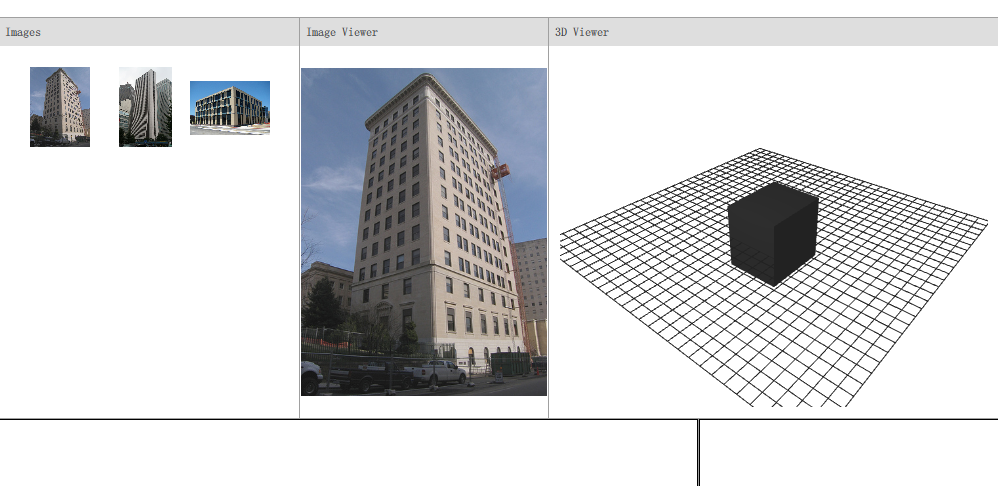
3月31日

import部分：

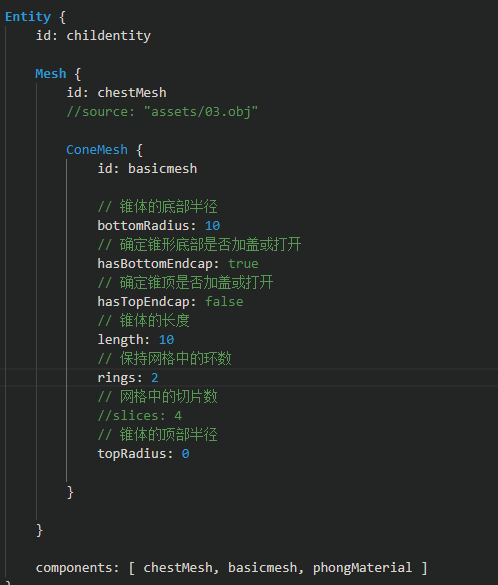


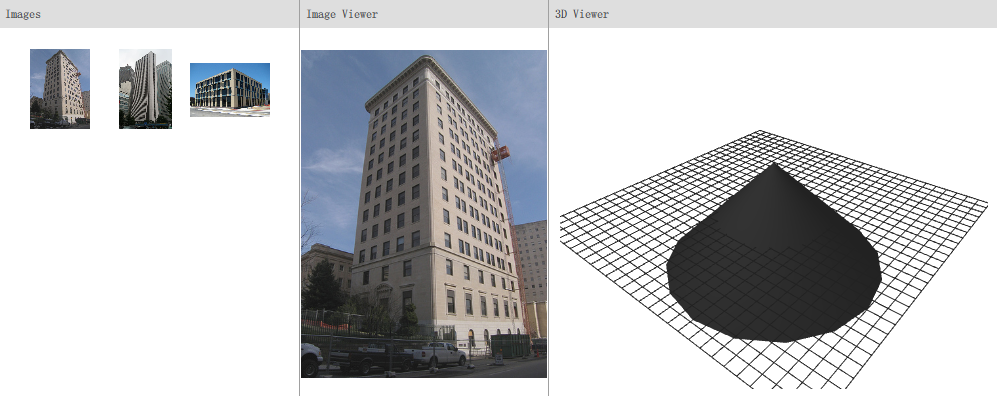
1. CuboidMesh



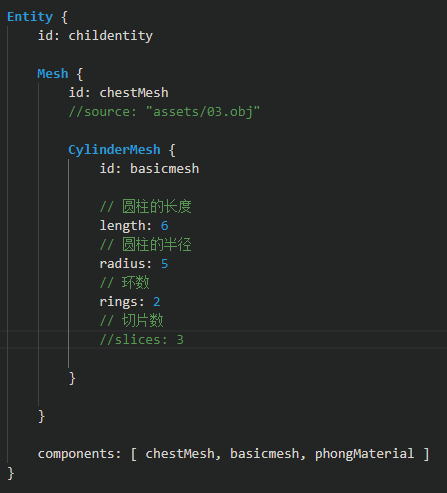


1. ConeMesh





1. CylinderMesh



注意：slics=3时,柱体为三棱柱，slice=4时，为四棱柱即长方体（如下图1），未设置slices时，柱体为圆柱体（图2）。slices控制底边多边形的边数（取值>=3）。椎体也是如此。

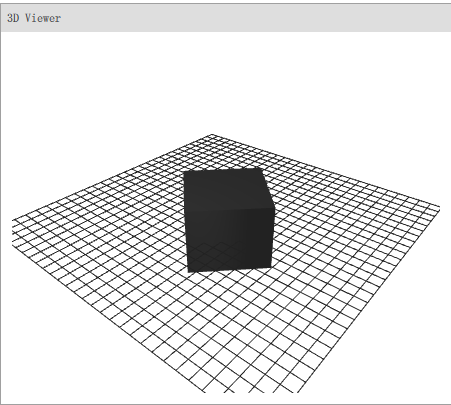


图1

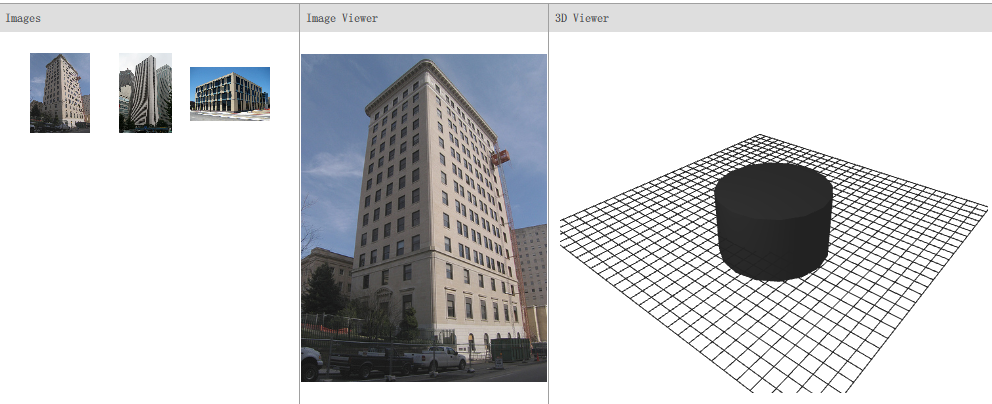


图2



PyOpengl环境搭建：

<https://blog.csdn.net/BigBoySunshine/article/details/80218245>

4月1日 QtXML——读取XML文件

1 创建一个QDomDocument类对象，代表整个XML文档

2 使用QFile打开要读取得xml文档，使用QDomDocument类的setContent()函数来设置整个文档的内容，它会将XML解析成一个DOM树,并保存在内存中。

QFile file(filename);

QDomDocument doc;

doc.setContent(&file, true);

3 获取根节点元素,QDomDocument类也是QDomNode的子类，使用firstChild()函数可以获得它的第一个子节点。使用documentElement()可以获得他的根节点，这是访问XML文档的入口。

QDomElement root = doc.documentElement();

QDomNode child\_node = root.firstChild();

4 后面就可以通过QDomNode来遍历整个文档。

QDomNode的函数：

firstChild()：获得第一个子节点

lastChild()：获得最后一个节点

childNodes()：获取该节点的所有孩子节点的一个列表

nextSibling()：获取下一个兄弟节点

previousSibling()：获取前一个兄弟节点

// 对xml文件的每个grammar节点进行分析

while (!child\_node.isNull()) {

if (child\_node.toElement().tagName() == "grammar") {

Grammar grammar;

parseGrammar(child\_node.toElement(), grammar);

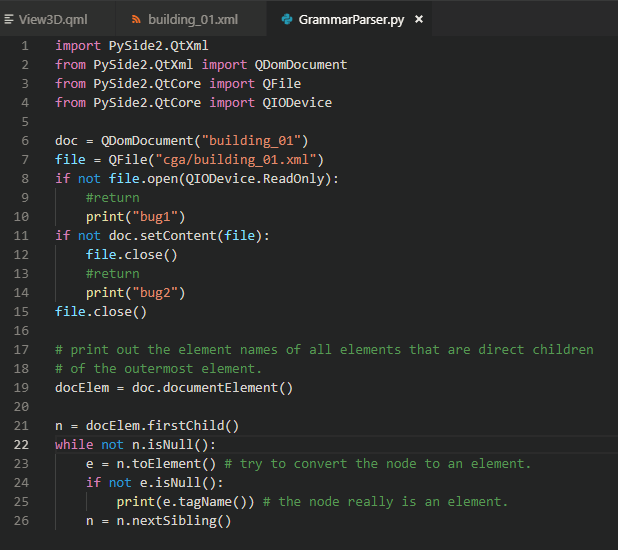
grammars.push\_back(grammar);

}

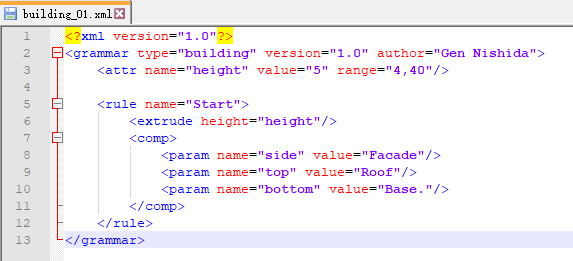
child\_node = child\_node.nextSibling();

}

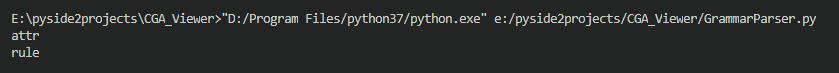
新建GrammarParser.py，第一个示例：



输入：



输出：



后续的代码还没调试好。。。

4月2日 敲代码中，rule部分C++转python，代码提交到GitHub的CGA\_Viewer项目中。

4月3日 上午科目一考试，下午和晚上继续码。。。依旧是rule中对xml进行语法分析部分，这段敲完之后会进行调试。

4月4日-4月8日 编码

4月9日 问题：python+qml怎么在QML场景中用OpenGL渲染多边形?

Cube OpenGL ES 2.0 example：

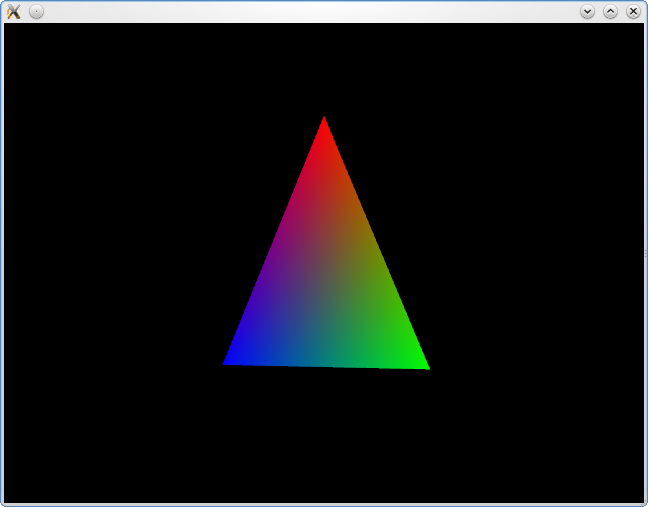
<https://doc.qt.io/qtforpython/overviews/qtopengl-cube-example.html?highlight=glenable>

有许多方法可以在OpenGL中渲染多边形，但最有效的方法是仅使用三角形条带基元并从图形硬件存储器中渲染顶点。 OpenGL有一种机制可以为这个内存区域创建缓冲区对象，并将顶点数据传输到这些缓冲区。在OpenGL术语中，这些被称为顶点缓冲对象（VBO）。

这是立方体面分解为三角形的方式。以这种方式对顶点进行排序，以使用三角形条带使顶点排序正确。 OpenGL根据顶点排序确定三角形正面和背面。默认情况下，OpenGL对前面使用逆时针顺序。该信息由背面剔除使用，其通过不渲染三角形的面来改善渲染性能。这样，图形管线可以省略三角形的不面向屏幕的渲染边。

使用QOpenGLBuffer创建顶点缓冲区对象并将数据传输到它们非常简单。 MainWidget确保使用OpenGL上下文当前创建和销毁GeometryEngine实例。这样我们就可以在构造函数中使用OpenGL资源，并在析构函数中执行适当的清理。

OpenGL Window Example：<https://doc.qt.io/qt-5/qtgui-openglwindow-example.html>



Scene Graph - OpenGL Under QML：

演示如何在Qt Quick场景下渲染OpenGL

<https://doc.qt.io/qtforpython/overviews/qtquick-scenegraph-openglunderqml-example.html?highlight=qmlregistertype>

Scene Graph - Custom Geometry：

自定义几何示例显示如何创建QQuickItem使用场景图API为场景图构建自定义几何的方法。它通过创建一个BezierCurve项来实现这一点，该项是CustomGeometry模块的一部分，并在QML文件中使用它。

<https://doc.qt.io/qtforpython/overviews/qtquick-scenegraph-customgeometry-example.html?highlight=qmlregistertype>

QGLWidget类：

<https://doc.qt.io/qtforpython/PySide2/QtOpenGL/QGLWidget.html?highlight=glenable>

用于Python交互的QML和Qt：

<https://blog.qt.io/blog/2018/05/14/qml-qt-python/>

如何用Python for Qt / PyQt：

<https://machinekoder.com/how-to-not-shoot-yourself-in-the-foot-using-python-qt/>

QML与Python通信：

<https://my.oschina.net/u/1275030/blog/186341>

4月10日

实现跨平台的QML和OpenGL混合渲染：

<https://blog.csdn.net/gamesdev/article/details/38024327>

PySide2 port of the opengl/contextinfo example from Qt v5.x：

D:\Program Files\python37\Lib\site-packages\PySide2\examples\opengl

4月11日

import QtQuick 2.7

import Qt3D.Core 2.12

import Qt3D.Render 2.12

import Qt3D.Input 2.12

import Qt3D.Extras 2.12

Entity{

id: rootEntity

Camera {

id: mainCamera

projectionType: CameraLens.PerspectiveProjection

fieldOfView: 45

nearPlane: 0.01

farPlane: 1000.0

position: Qt.vector3d(24.0, 20.0, -20.0)

upVector: Qt.vector3d(0.0, 1.0, 0.0)

viewCenter: Qt.vector3d(0.0, 0.0, 0.0)

}

FirstPersonCameraController { camera: mainCamera }

components: [

RenderSettings {

activeFrameGraph: ForwardRenderer {

camera: mainCamera

clearColor: "transparent"

}

},

InputSettings {}

]

PhongMaterial {

id: phongMaterial

ambient: "#FFF"

diffuse: "#222"

specular: diffuse

shininess: 0

}

Entity{

id: childentity

GeometryRenderer{

id: geometryRenderer

instanceCount: 1

primitiveType: GeometryRenderer.Triangles

geometry: Geometry{

Attribute{

attributeType: Attribute.VertexAttribute

vertexBaseType: Attribute.Float

vertexSize: 3

byteOffset: 0

byteStride: 3 \* 4

count: 3

buffer : Buffer{

id: vertexBuffer

type: Buffer.VertexBuffer

data: [ 0.0, 0.0, 0.0, 0.0, 10.0, 0.0, 10.0, 10.0, 0.0]

}

}

}

}

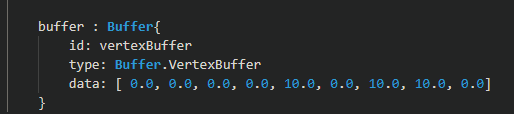
components: [phongMaterial, geometryRenderer]

}

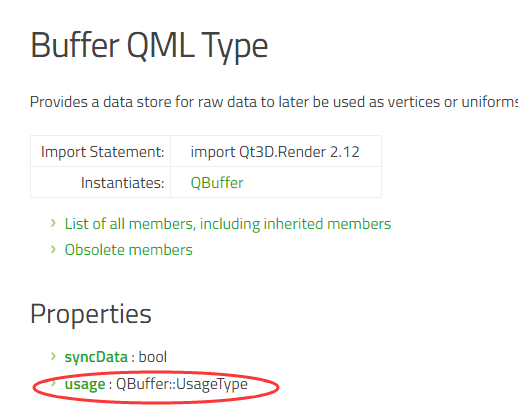
Grid3D {}

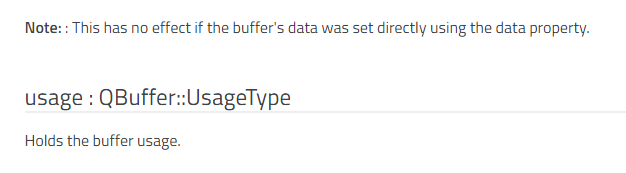
}

错误：



原因：

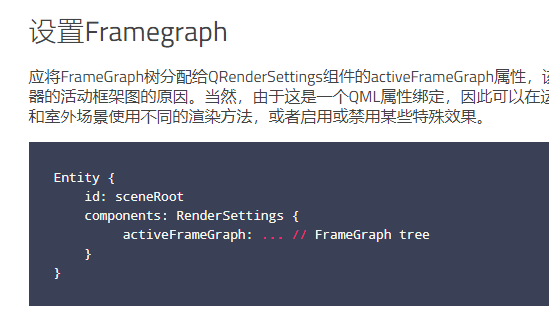




4月12日

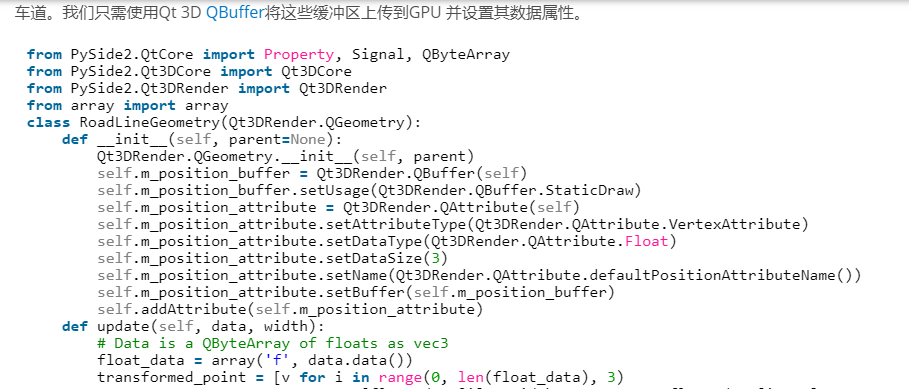
Qt 3D渲染帧图

framegraph是控制场景渲染方式的数据结构



<https://doc.qt.io/qtforpython/overviews/qt3drender-framegraph.html>

PySide2.Qt3DRender例子：（代码不完整）



<https://www.kdab.com/python-tron-demo/>

An Introduction to Qt3D：

<https://www.ics.com/blog/introduction-qt3d>

4月13日 修正11日的bug



