中国大学生计算机设计大赛



交互媒体设计创作文档

作品编号：　　　　　　2024035705

作品名称：　　ICmd-指令行交互应用开发通用框架

作　　者：　王永淳 蒋伟豪 蒋忱迅 任川渝 罗钧元

版本编号：　　　　　　　　V1.3

填写日期：　　　　　2024年4月25日

目 录

[第一章 行业背景与意义价值 3](#_Toc165054668)

[第二章 作品概述 4](#_Toc165054669)

[第三章 技术难点与算法实现 5](#_Toc165054670)

[第四章 作品功能和实现说明 9](#_Toc165054671)

[第五章 运行结果 17](#_Toc165054672)

[第六章 应用场景与未来展望 18](#_Toc165054673)

[参考文献 19](#_Toc165054674)

# 行业背景与意义价值

1.1当前各类交互界面背景

物联网已被国务院列为我国重点规划的战略性新兴产业之一，在相关政策带动下，我国物联网产业呈现高速发展的态势。同时2021 年 3 月 23 日，国家发展改革委等十三个部门发布《关于加快推动制造服务业高质量发展的意见》。《意见》中强调了各类信息技术的重要性，强调了智能制造的发展需求和前景。毫无疑问，在未来物联网将会得到更加深入的发展。

但是在当前的物联网还有工业互联网内中，大量的设备都不像我们常用的Windows系统一般拥有图形化的交互界面，例如在物联网的控制服务端多是使用Centos之类的Linux操作系统，此类操作系统通常只有命令行界面以及简单的颜色显示能力。

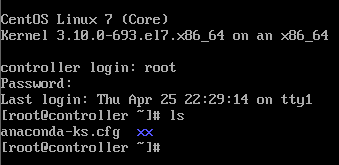


图1.1 centos仅有命令行界面显示

除此之外，物联网终端的各类机器，其主要功能并不需要复杂的显示，更有使用单片机之类的终端，仅仅只有字符显示的功能，并不具备搭载各类图形化界面或者交互式界面的功能，只能做到简单的字符显示。

所以，当前工业互联网以及物联网市场中存在的终端与服务端的机器普遍存在着缺乏图形化界面的痛点。这要求各行各业的操作工人要能够对现场的终端，服务器显示的参数和信息拥有敏锐的观察能力和警觉性，从而发现机器出现的问题和故障。这不仅提高了人工培训的成本，也增加了出现误判的概率，降低了对于突发事件的处理效率。

所以当前物联网中存在的各式设备中，存在着如上亟待解决的痛点。市场需要一个能够解决痛点，在各种设备上做到图形显示功能的交互式框架。这是我们提出全新交互式框架的背景。

1.2需求分析

随着物联网的发展，无人机，单片机，各类传感器等等各类设备也在不断迅猛发展，在这些硬件系统中，常常缺乏了拥有图形化能力、交互能力的界面系统。

与此同时，此类设备的操作系统通常为只拥有命令行界面的操作系统，设备本身通常也并不拥有专门的渲染硬件，只有CPU一类的计算硬件，这些客观条件决定了此类设备并不具备安装并运行类似QT等图形化界面的能力。

而一个完整的拥有较强显示功能和交互功能的界面系统，将会很好的为物联网终端、服务端设备赋能，提升使用者的开发和设计效率，起到降低维护成本的效果。

因此，针对当前并非所有设备都拥有图形化交互界面的难题和痛点，一个能够在命令行界面实现便于理解、便于使用的拥有交互功能的框架是当前市场存在的需求。

我们开发的命令行交互应用开发通用框架（ICmd）便针对了这样的现状，提出了使用在命令行界面中通过字符打印和监控各类输入方式的形式，实现通用开发框架的解决方案，旨在通过为各类系统提供通用框架的形式，在较低的系统占用的前提下，实现图形化界面，并从此角度实现图形化交互界面的革新。

1.3竞品分析

在当前的市场上，已经存在了大量类似的图形化界面，这里选用同样适用C/C++的QT和EASYX进行比较。

|  |  |  |  |
| --- | --- | --- | --- |
| 功能\框架名 | EASYX | QT | ICmd |
| 主要应用 | 实现图形库 | 制作GUI | 基于字符画绘制GUI |
| 操作系统 | 基于Windows | 多平台兼容 | 未来将多平台兼容 |
| 生态 | 完善度低 | 允许组件编程 | 允许组件编程且提供了丰富的自定义接口 |
| 对资源占用 | 占用较大 | 占用较大 | 占用低，适配各类机型 |

表1.3 竞品分析

与其他的常用图形化框架相比，ICmd拥有更好的适配性，打印字符的图形渲染方式虽然会在清晰度上略有缺失，但会占用更少的系统资源，可以更好的在一些设备上适配。

# 作品概述

2.1作品简介

命令行交互应用开发通用框架（以下简称ICmd）是一款基于C++语言，为命令行界面所设计的，拥有XML支持的，用于进行交互应用开发的，通过打印字符的形式实现图形化的通用框架。当前已可在Windows平台中运行，通过修改基础的原生库的调用，就可以在其他系统中使用。

此框架通过在命令行窗口中打印字符的方式进行图形和界面的渲染，避免了常用的图形化界面使用各类框架会占用大量内存与计算资源的问题。使得用户可以在命令行窗口中实现可视化的界面，以及完成鼠标点击，键盘操控，等等一系列操作。从而通过字符串的形式，实现并满足了在缺少图形化界面的操作系统中（如Centos）、在占用较少资源的情况下，进行交互式应用的开发的需求。

2.2创作立意

基于以上提到的需求和市场现状，我们可以发现当前的各种交互式界面存在的问题。

因此我们设计了这样的一个交互应用开发通用框架。他基于C++语言和CMAKE技术，仅需要使用少量系统自带的库函数进行调用的特点，使得此框架在微调后，可以在各种操作系统中使用，从而可以满足在不同操作设备中的图形化交互的需求。

而这些通过ICmd实现图形化交互的系统也将只会被占用较少的资源，保证了它可以在执行原本的任务的同时，使用可视化的形式进行交互。这有效的降低了开发者和系统维护人员的学习成本，提高了设计效率。而我们也将通过这样的方式，助力我国物联网产业加速发展，为无人机，大型机器等各类机械的发展提供助力。

2.3开发原则与创新点

1.脱离原生系统，降低跨平台成本

本框架仅仅在命令行输出与监听鼠标/键盘的地方使用了原生系统自带的库，其他成分包括图形库，消息循环以及各种UI的底层实现，底层渲染，事件调度，以及更多的对于一个成熟UI框架必不可少的部分，均为使用C++20标准库，由开发人员从0开发，有效降低了对各类函数库的依赖性。本文档也会在以下篇幅中扼要地阐述每个部分的设计思路。

2.减少心智负担，追求每个开发者最舒服的开发姿态

拥有Android或者其他大型项目开发经验的开发者，都会对一个开发友好型框架有着深切的向往。此框架也提供了一整套面向开发者的开发工具包（ICmd Development Kit），不仅仅支持XML进行布局设计，也提供了高度响应式，高度拓展性以及高度简明性的用户接口，力图在保证性能优秀的同时，给予每个应用开发者一个舒服简便的开发环境。

3.优化的编程手段，先进的编程思想

采用了元编程技术，函数式开发思想，节点式布局设计，在优化了整体程序的同时，也增加了框架的容错性，提高了可拓展性，提供了丰富的自定义方式，并减少了开发者的心智负担。

4.命令行中也可以有丰富的交互

在传统命令行界面中只有输入与输出以及黑屏和白字，用户和开发者在使用时常会出现疲惫和倦怠感，此框架在鼠标交互提供了非完备的监听机制，在颜色与样式更是实现了高度的风格化，让用户可以对界面的色彩，样式等进行自定义，让原本平淡无奇的命令行可以为用户带来如应用程序UI般的体验。

# 技术难点与算法实现

3.1难点一：元编程

1.简述

ICmd实现了字符串为索引的元组。

众所周知元组的类型是编译时确定的，也是编程中编译时多态性的体现（Compile-time Polymorphism），而传统元组是以数字为索引的，例如mTuple.get<1>()，本框架将类似于mTuple.of<”hello”>()的代码成为了可能。



3.1.1 元编程流程图

2.实现

元编程可以和运行时的代码相互比较学习，例如using A = int与int i = 0这种赋值语句类似， is\_enable<true, A>::type与printf(“xxx”)这种调用语句类似。通过学习元编程中的基本概念，可以在C++中进行类的任意多态处理

定义编译时字符串struct CompTimeStr

将编译时字符串的模板（template <CompTimeStr... Keys>）和类型模板（template <typename... Args>）以嵌套结构体方式放进字符串索引元组的定义

定义中声明元组tuple\_raw作为背后的容器

定义中嵌套tuple\_helper，用模板类多态推断（Type Deduction）的方法，将Keys遍历，成员为Key，设置静态常数index，获取当Key与传来的Key值相等时的迭代器I的值并给予index

获取tuple\_raw中位置为I的元素与类型，返回。

3.元编程在ICmd中的应用

字符串索引元组可非常方便地作为每个节点或节点组件的参数的容器，替代的std::map，允许多类型，且在运行前完成类型推导，既保证了输入值的类型检查，也保证了取出值的类型推断，全程无类的转换与指针参与，是一种非常优雅而实用的参数寄存方法。

3.2难点二：图形库渲染的实现

1.简述

对于各种复杂的图形，都可以被分解为一些简单的图元的组合。本节我们将描述如何在ICmd中实现对于各种类型图形的渲染实现。我们在内核从最底层实现一个图像引擎，为命令行渲染实现轻量化渲染。在应用层，用户定义的各种图形将分解为图元，传入内核，在内核中实现渲染。

2.实现

基于轻量化的思想，我们的图像引擎流程为：图元装配→光栅化→片段着色器→测试混合。

对于图像渲染流程来说，流行的图像引擎总是会经过：顶点着色器→几何着色器→形状匹配→光栅化→片段着色器→测试混合。

对于这些图像引擎来说，顶点属性通常包括：顶点位置，反射率，法向量，切线向量。

由此可见，对于传统的图像引擎来说，其中大部分顶点属性和流程是用于光渲染的，需要计算顶点的弧面弧度或者透明度。虽然对于我们的任务来说，命令行能展示的字符的色彩范围通常很有限，所以一般不考虑复杂的光渲染以及透视。因此我们需要自定义一个轻量的图像引擎。

为了实现轻量化，我们将顶点着色器和片段着色器合并，即直接将图形光栅化并根据着色器的输入选择填色方案，不对顶点做额外的定位和处理。综上所述，我们应该对缓冲区（缓冲区）进行直接着色。既将步骤简化为：图元装配→光栅化→片段着色器→测试混合。

顶点属性简化为：顶点位置，图元装配的作用是将顶点匹配对应的形状（线段，多边形等等）。我们可以匹配的形状有：点集，线段集，闭合线段，三角形集，多边形。

对于不同的图形，我们将采用不同的渲染方案，以尽可能减少渲染的开支。对于这些图形的具体渲染方案将在光栅化一节详细描述。

光栅化的作用主要是将坐标映射到缓冲区上，即将矢量图形式的图形转化为位图形式的图像。我们会在内核定义一个渲染器，以规定渲染的风格。接下来我们具体介绍不同的图形对应的渲染算法：

点集，

对于点集，我们依次对其中的每一个点渲染。渲染器会给出这个点集中点的大小。一个顶点数为n的点集，我们将生成n个点。

对于半径设定为R的顶点，我们的渲染区域计算公式为：

线段集，

对于线段集，我们依次对其中的每两个点渲染，生成一条直线。渲染器会给出这个线段的宽度。一个顶点数为n（n 为偶数）的线段集，我们将生成n/2个线段。

对于宽度设定为R的一对顶点，我们的渲染区域计算公式为：

闭合线段集，

对于闭合线段，我们依次对其中的每个点和下一个点渲染，生成一条直线，并将最后一个点和第一个点生成一条直线以生成闭合线段集。渲染器会给出这个线段的宽度。一个顶点数为n（n > 2）的线段集，我们将生成n个线段。

我们的渲染区域计算公式与线段集相同，不再赘述。

三角形集，

对于三角形集，我们依次对其中的每三个点渲染，生成一条三角形。一个顶点数为n（n 为3的倍数）的三角形集，我们将生成n/3个三角形。

三角形的渲染区域计算较为复杂。对于每个三角形，我们采用根据拆分填充算法。

此处的标准算法将一个三角形拆违两个更容易绘制的三角形，一个底边平行，一个顶边平行。对于这两种三角形，我们可以采用简单的遍历计算出每一个需要渲染的区域，以底边平行()的三角形为例，即：

具体的拆分方法即为找到三个顶点中y坐标第二大的点，然后以该顶点划一条平行于x轴的直线，计算该直线与三角形对边的交点，以此将一个三角形分割为两个三角形。

多边形。

对于多边形，我们先对该多边形的每个顶点依据其重心进行逆时针排序，然后生成一个多边形。我们的多边形渲染算法支持凸多边形和凹多边形。一个顶点数为n（n > 2）的多边形，我们将生成1个多边形。

多边形的渲染区域计算较为复杂。我们采用根据扫描线填充算法（有序边表法）。

扫描线填充算法的基本思想是：用水平扫描线从上到下（或从下到上）扫描由多条首尾相连的线段构成的多边形，每根扫描线与多边形的某些边产生一系列交点。将这些交点按照x坐标排序，将排序后的点两两成对，作为线段的两个端点，以所填的颜色画水平直线。多边形被扫描完毕后，颜色填充也就完成了。扫描线填充算法也可以归纳为以下4个步骤：

（1）求交，计算扫描线与多边形的交点

（2）交点排序，对第1步得到的交点按照x值从小到大进行排序；

（3）颜色填充，对排序后的交点两两组成一个水平线段，以画线段的方式进行颜色填充；

（4）是否完成多边形扫描？如果是就结束算法，如果不是就改变扫描线，然后转第1步继续处理。

对于每一条扫描线，如果每次都按照正常的线段求交算法进行计算，则计算量大，而且效率底下。为了减少计算量，扫描线算法需要维护一张由“活动边”组成的表，称为“活动边表（AET）”。

“活动边表”是扫描线填充算法的核心，整个算法都是围绕者这张表进行处理的。要完整的定义“活动边表”，需要先定义边的数据结构。每条边都和扫描线有个交点，扫描线填充算法只关注交点的x坐标。每当处理下一条扫描线时，根据△x直接计算出新扫描线与边的交点x坐标，可以避免复杂的求交计算。一条边不会一直待在“活动边表”中，当扫描线与之没有交点时，要将其从“活动边表”中删除，判断是否有交点的依据就是看扫描线y是否大于这条边两个端点的y坐标值，为此，需要记录边的y坐标的最大值。

为了方便活性边表的建立与更新，我们为每一条扫描线建立一个“新边表（NET）”，存放该扫描线第一次出现的边。当算法处理到某条扫描线时，就将这条扫描线的“新边表”中的所有边逐一插入到“活动边表”中。“新边表”通常在算法开始时建立，建立“新边表”的规则就是：如果某条边的较低端点（y坐标较小的那个点）的y坐标与扫描线y相等，则该边就是扫描线y的新边，应该加入扫描线y的“新边表”。

片段着色器的作用是绘制和填色，我们直接在帧上将需要渲染的区域的属性填入渲染器规定的属性。

测试混合的作用是将我们需要渲染的多个图形按照给定的顺序依次渲染，设定这些图形之间的覆盖关系，同时对超出画布的部分进行裁剪。

综上所述，我们在定义顶点的时候就已经包含了它在帧上的绝对位置，所有的顶点和顶点数组都不需要长，宽以及图形位置的属性。但是在应用层时，我们会将复杂的图元具象化为一个图形对象，这个对象包含了长宽和图形位置的属性，便于用户进行自由地调整需要渲染的图形。

# 作品功能和实现说明

在对此框架进行设计时，分为内核层与用户层。从而在保障框架的稳定运行的同时暴露足够多的可用接口给用户从而实现更加丰富的自定义组件。

4.1内核层实现说明

内核层主要分为了3个线程，2条队列，分别是内核线程，渲染线程，还有监听器线程。队列则分为了帧循环队列和消息循环队列。帧循环队列用于更新帧和处理各类信息，消息循环用于处理窗体收到的各式消息。

首先介绍两条队列的实现逻辑：



图4.1.1消息处理机制示意图

其中，消息队列用于存储当前的系统中出现的信息，并通过先进先出的形式管理其中的信息。消息队列由一条队列和数个handler组成，handler向消息队列不断发送信息进入队列，而一条消息包括了消息类型，从哪里来，到哪里去，以及其中的信息。一般消息队列收到信息后，就将相关的信息发送给要发送到的handler处。然后等要处理相关信息的handler处理完后，通过update函数将此消息弹出消息队列。



图4.1.2帧循环机制示意图

帧循环队列则是将渲染好的帧放进队列中，通过渲染线程不断先进先出的放入帧和取出要打印的帧。

与帧循环队列深切相关的是canvas相关函数，通过start开始绘制，加入节点，绘制内容的参数等等等等，在绘制完成后调用finish函数，在通过着色后，会将绘制的内容发送为MT\_DATE信息到渲染线程中，而渲染线程则将这个帧压入到帧循环队列中，帧循环队列则不断将已有的帧弹出并绘制。

接下来分别介绍3个线程的实现逻辑：



图4.1.3 消息循环流程图

消息循环：由looperThread负责，其中是消息循环，用于处理线程间的通讯，生命周期与资源加载。

整个消息循环从startloop函数进入，并在启动后获取窗口句柄c，在此时向render线程发送load\_over信息。之后从消息队列中取出信息msg，根据msg的类型决定如何处理该信息。而在处理了需要处理的信息后，将会更新一次消息队列，使消息队列的下一个信息到达队列的最顶层。



图4.1.4 渲染逻辑流程图

图形渲染：由renderThread负责，其中是图形库与渲染程序，用于处理各种图元信息，以及输出帧信息，字符打印。

从startTick函数进入，从中先初始化帧信息的参数后，取出msg后，根据msg的数据类型进行不同的处理，比如收到load\_over时会将信息发送给消息队列并要求更新。收到shutdown会推出线程，而收到update会存储帧的信息并压入帧循环队列中，而帧循环队列则不断的弹出顶帧进行更新和渲染。



图4.1.5 监听循环流程图

监听循环：由hookThread负责，其中是对于键盘与鼠标的循环监听。

监听循环时刻监听是否有需要安装hooks的需求，如果有则获取当前节点和句柄以及消息队列，然后设置按键的处理函数，并进入控制台模式遍历所有的事件类型，之后再按情况执行各类型的函数，并将处理结果输出，让系统可以进行监听处理。



图4.1.4 三个线程的逻辑流程

三个线程中，由消息循环线程启动，进行onCreate的处理，帧循环线程则不断等待帧信息，更新帧队列，监听线程则等待hooks信息，将监听到的信息传递给帧队列，从而不断响应用户的操作。

4.2用户层重要概念讲解

1.view.hpp中的概念讲解

view.hpp中提供了每个节点的更新，绘制等接口。

Node（节点）: 应用层面渲染的最小单位，相当于Android中的View，但是有着比View更高的响应性与组件性

NodeComponent（节点组件）：规范每个节点行为的单位。每个节点的渲染，布局，监听都可由节点组件提供

在进行组件的自定义编写时，通常通过继承这两个基类实现。

2.canvas.hpp中的概念讲解

Canvas中提供了绘制所需的基本函数，例如drawLine，drawCricle等。

其中使用的参数l,t,r,b代表了绘制图形外接矩形的上顶，下底，左边，右边四个边，通过四个数字决定绘制图形各个点位的位置，如原型就以四边为外接矩形绘制内接圆。

而使用canvas进行绘制时，则需要按照start()…绘制函数…finish()的格式进行绘制。

3.os.hpp中的概念讲解

在os中主要定义了从xml文件读取node信息的函数，以及上下文管理器（context），从而实现函数的定义。

Context（上下文管理器）：在构造时需要输入应用名，作者名，窗口高度，窗口宽度等，承担了上下文信息传输的功能。

4.3用户创作的应用程序的生命周期

1. 资源加载 – 内存地址的里应外合：



图4.3.1 内核初始化的缩略概念图

* 1. 程序开始时，首先加载的资源是用户定义的onCreate方法，窗口/应用参数，以及XML文件，资源性文件（图片，音频等等），和监听器回调统一进入**上下文管理器**（Context）中进行热加载。
  2. 上下文管理器将加载结果（内存地址）封装成回调，通过调用**内核主函数**的方法给予内核，封装进**内核资源上下文**（CoreContext）中

1. 内核的初始化 – 消息握手的左拥右抱：
   1. 创建**帧队列**（FrameQueue）与**消息队列**（MessageQueue），将三大线程，互斥锁等资源进行初始化，并将**资源上下文信息**（CoreContext）与**应用信息**（CoreInfomation）给三个线程
   2. looperThread（消息循环线程）中资源加载完毕后，发送该线程初始化完毕（LOAD\_OVER）的信息给renderThread（渲染线程）
   3. renderThread收到消息后，初始化**命令行窗口**（Console），发送初始化完毕（LOAD\_OVER）的信息给looperThread；发送安装Hook函数（INSTALL\_HOOKS）的信息给hookThread（监听线程）
   4. looperThread收到消息后，从内核资源上下文提取用户的onCreate方法并执行，并将消息循环句柄以参数形式反射到应用层面
   5. hookThread收到消息后，开启监听循环，侦测鼠标和键盘的状态
   6. 至此内核启动并持续运行
2. 节点的加载 – 应用层面的翩翩起舞
   1. 节点在构造中捆绑**上下文**（Context）
   2. 将必须组件加载到节点上（例如Background组件使得节点有纯色背景）
   3. 将节点附属到上下文中的**顶层节点**（parent）中
   4. 顶层节点进行更新



图4.3.2 layout与draw的工作方式

* + 1. 调用layout函数，自父到子调用每个子节点的layout函数，确定每个节点的绝对坐标与长宽
    2. 调用draw函数，自父到子调用每个子节点的draw函数，实现提取节点的绘制结果，渲染为帧信息后，封装为一个更新（UPDATE）消息，发送给renderThread
  1. renderThread将帧信息输出到命令台，结束节点到命令台的渲染全流程

1. 节点的监听 – 交互响应的心有灵犀
   1. hookThread中的循环侦测到事件后，将事件各种参数进行包装
   2. hookThread调用内核资源上下文中传来的全局回调
   3. 全局回调中将事件进行二次包装，包装为提供给用户事件回调的**鼠标事件**（MouseEvent）与**键盘事件**（KeyEvent）
   4. 全局回调中遍历所有顶层节点的子节点，并查找他们的**监听器**(Listener)，将符合事件条件的监听器回调触发
   5. 至此从底层到应用层面的事件处理全部完成。
2. 应用的结束 – 华丽呈现的爽快收尾

各个Node析构，将上下文中各种数据类型进行释放，发送SHUTDOWN消息至looperThread，内核开始释放各种资源，退出各种线程，最后关闭窗口

# 运行结果

5.1 图形绘制应用

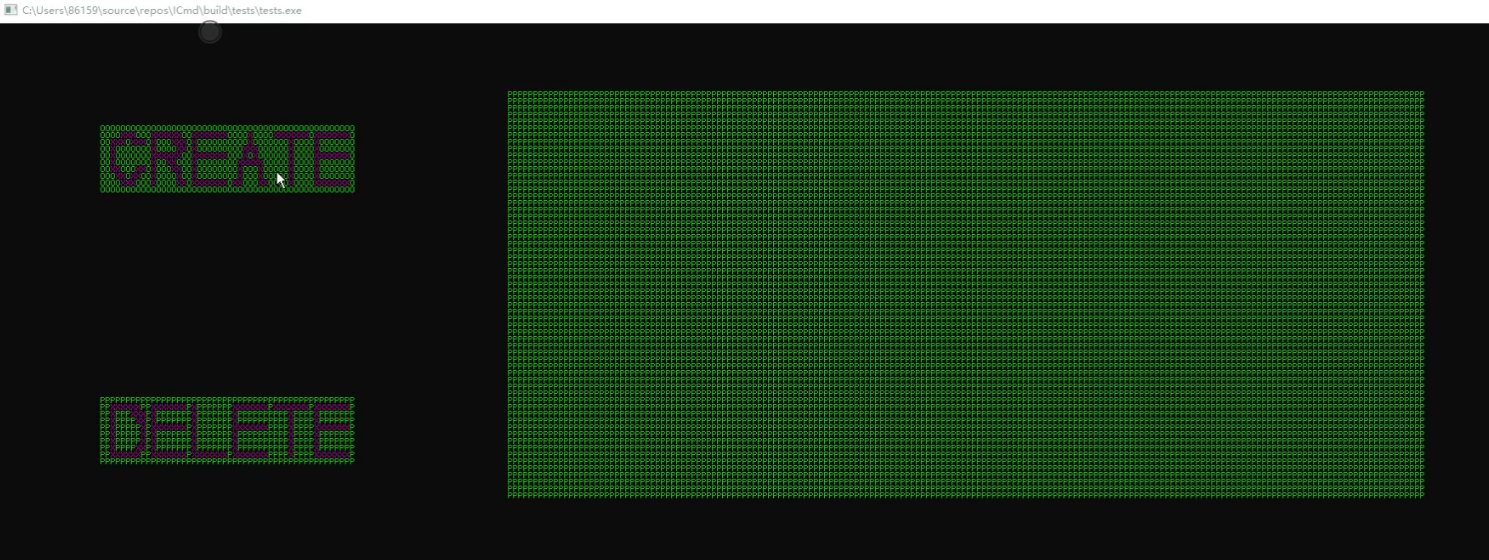


图5.1.1图形绘制界面

此应用用于展示基于本框架实现的图形绘制功能和点击功能。此时为应用刚启动的模式。

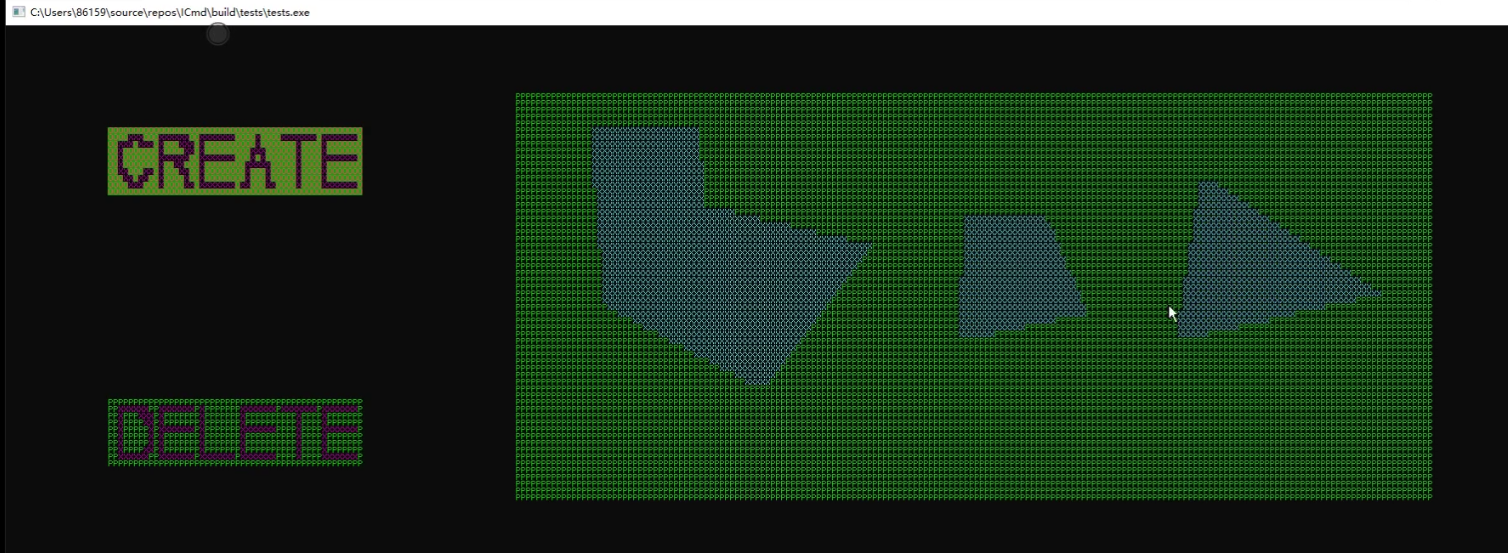


图5.1.2多边形绘制功能展示

在点击左侧的create后，可以进入绘图模式，create键高亮，此时为绘制模式，在窗口内通过鼠标点击的方式进行多边形的绘制，在点选各个点位后，画板内会显示出对应的图像。

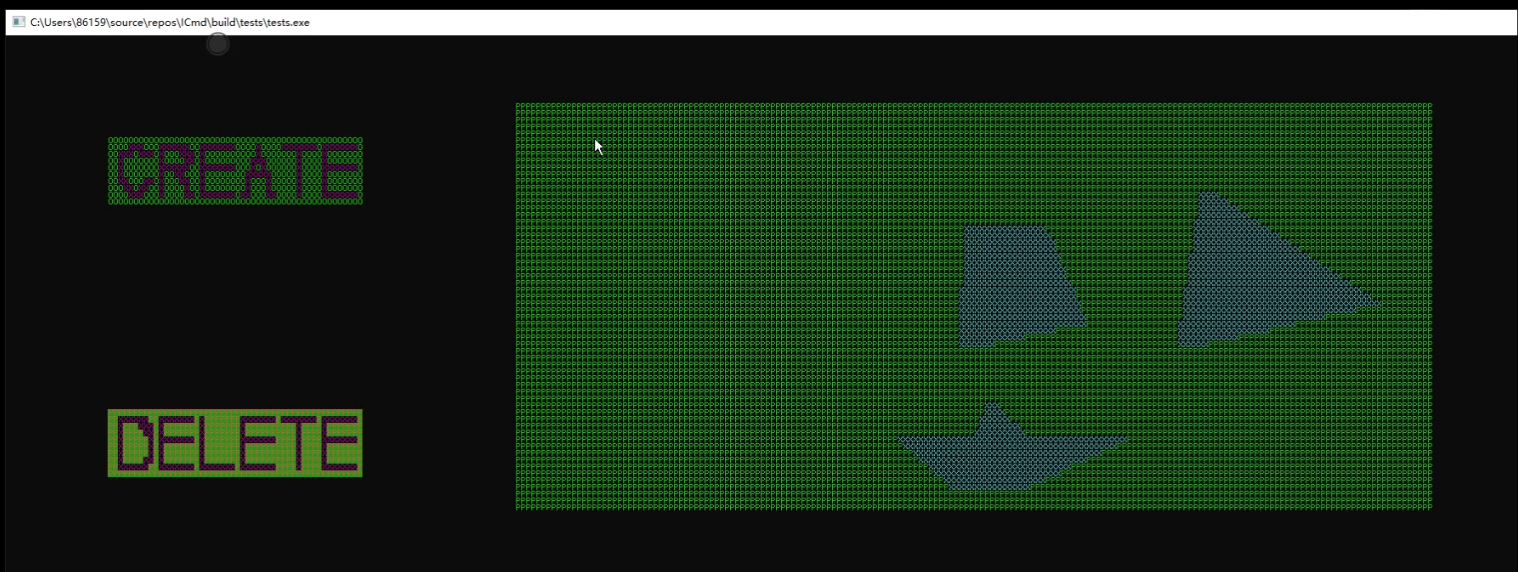


图5.1.2多边形绘制功能展示

点击Delete后，进入删除模式，delete键高亮，此时点击已经绘制好的图像，可以删除该图像。

5.2 桌面功能

此应用主要作为一个桌面，进行各项应用的统筹安排，可以通过鼠标拖动的形式让内部的应用进行移动。

# 应用场景与未来展望

6.1应用场景

1.初学者入门C/C++

当前，为面向科学技术的现代化，C/C++语言课程已经成为了大部分工科学子必定会学习的课程，而在大部分工科学子学习相关内容时，也将会面临需要使用图形化交互界面的程序设计要求。但对于初学者而言，诸如QT等复杂的图形化界面框架有着过高的学习成本，难以快速掌握。在这种时候，像此框架一样可以快速学习的图形化界面将会更好的帮助他们完成相关程序设计。

同时，此框架的丰富的组件的基类提供了大量的可使用接口，拥有强大的自定义能力，可以为初学者提供很好的面向对象编程范例，在用户使用的同时，也能从中学习和实践到C/C++语言的语言特性，从而更好的掌握相关的语言。

2.虚拟机以及服务器应用

随着云计算相关科技的不断深入和发展，虚拟机在各方面都变得更加常用，而其中大多数的虚拟机或是服务器都采用的是Centos一类的Linux操作系统，通常都缺乏图形化的界面，只能通过传统的命令行界面进行交互。在这种情况下，本框架将可以提供丰富的自定义接口和整体可视化交互开发框架，从而让开发者可以基于此框架，在此类并没有图形化界面的系统中进行可视化界面的开发。

3.工业互联网以及物联网终端设备的应用

在工业互联网以及物联网的终端设备，例如机械臂，大型生产器械，传感器等设备上，往往显示功能只是用于进行故障排查和显示基本参数，芯片和计算模块的主要算力都用于进行工业生产活动或者接受处理信息。

因此在此类机器上，专门为了拓展显示功能分配算力和更新显示屏并不现实，在这样的情况下，诸如QT等框架就显得较为笨重和庞大，而我们的框架则可以在如此缺少算力和显示能力有限的情况下，提供足够多的自定义模块和组件，从而让开发者可以针对设备设计专用的显示模块，将终端设备的信息图形化，让故障排查人员能够更好的发现终端设备的问题。或者开发专用的交互程序，降低使用者的学习成本和应用难度。

6.2社会效益

近年来，国家出台多项政策鼓励应用物联网技术来促进生产生活和社会管理方式向智能化、精细化、网络化方向转变。在这样的转变过程中，物联网技术的大发展，势必会带动物联网设备的发展，而同时也会带动更多用户对物联网设备的使用，从而导致物联网设备的扩大化和普及化。

在扩大化的过程中，物联网设备的交互显示功能，也将会成为一个重大的需求，在这种情况下，ICmd框架将会更好的在设备的硬件不做出调整的情况下，占用较少的资源进行更好的交互显示的开发。

除此之外，物联网技术的发展也将带动云计算服务的发展，而对于云计算使用Linux系统的服务端，我们的设备也将很好的改善开发者的使用体验，更好的在命令行界面的系统中为开发者提供良好的图形化框架，从而为开发降本增效。

同时，此框架的开放性接口和自定义组件简单易懂，让初学者可以借此框架了解C/C++语言的设计思想和理念，更好更快的入门C/C++语言。

总而言之，这通过此框架的应用推广，将可以在一定程度上提高国民经济和社会生活信息化水平、改善当前终端与服务端设备交互能力欠缺现状，带动相关学科发展和技术创新能力增强，为国家信息化发展提供动力。

6.3未来展望

当前本框架仍然处于起步阶段，未来将会致力于搭建更加良好的开发社区环境，促进开发者互相交流与更新框架版本，做到多平台兼容的同时，实现尽可能多的可用自定义组件，提供更加丰富的绘制与交互接口，实现多种帧率适配，动态灵活布局等目标。并与当前的渲染引擎走出差异化赛道，进一步做大做强，为国家的科教兴国人才强国战略做出贡献。

# 参考文献

【请按照标准参考文件格式填写】

[1] tzb592825420 (2015)多边形区域填充算法－－扫描线填充算法（有序边表法）[OL]<https://blog.csdn.net/u013044116/article/details/49737585>

[2] Sunshine(2012) Software Rasterization Algorithms for Filling Triangles[OL]<http://www.sunshine2k.de/coding/java/TriangleRasterization/TriangleRasterization.html#algo2>