**北 京 林 业 大 学**

**2023 学年— 2024 学年第 二学期 虚拟现实技术**

**实验报告书**

实验地点： 学研 任课教师： 杨 刚

实验题目： 增强现实技术开发

实验环境： unity，pycharm

# 实验内容概述:

开发过程中使用的开发包：

**在python中利用的工具如下：**

**Mediapipe：**是在python中用于实时的计算机视觉任务，例如手部追踪、面部识别、姿态估计、物体检测和手势识别等。

**Cvzone：**是封装了一系列常见的计算机视觉功能和模块，简化了基于 OpenCV 和 Mediapipe 进行计算机视觉任务的代码编写，例如人脸检测、手部追踪、姿势估计等。

还有OpenCV，numpy等相对来说熟悉的python工具包。

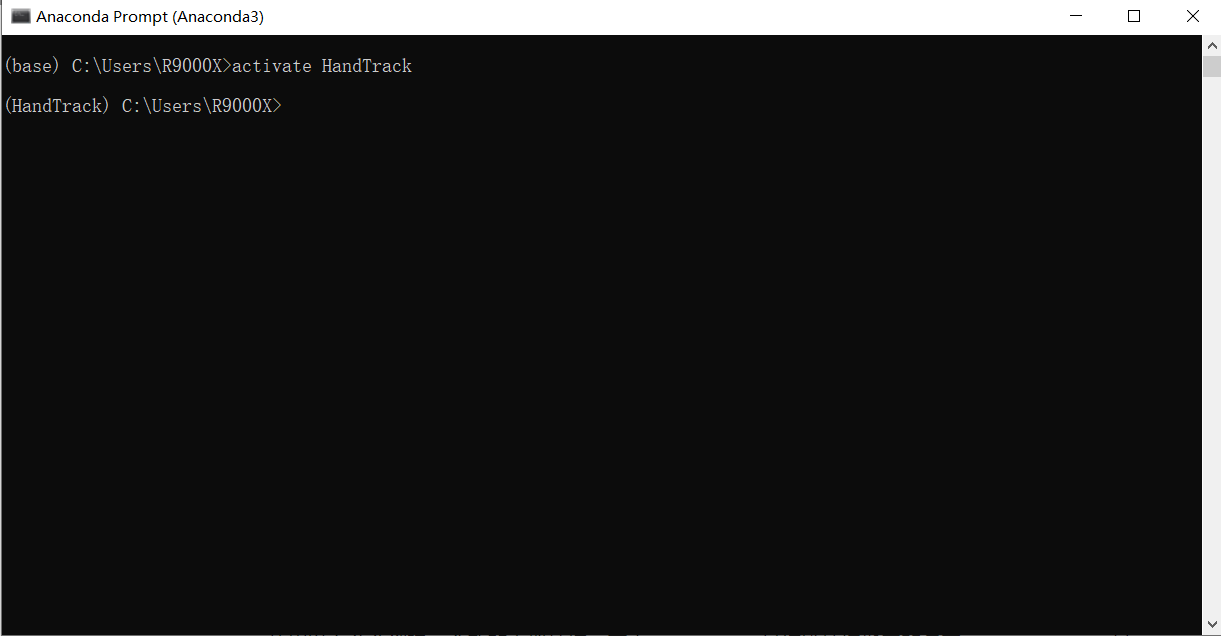
首先使用 OpenCV 启用摄像头并逐帧捕获视频，然后利用 Cvzone 中的手部追踪模块进行实时手部追踪。通过 udp传输协议，将捕获的手部信息**实时**传输到 Unity 中，并在 Unity 中，利用实时数据对手部以点和线条的形式进行模拟，并结合场景中的虚拟物体，为用户提供真实的手势操作体验，还可以让用户感受到手部击打虚拟物块时的物理效果。

# 实验方法:

#### 1-Python相关包的导入：

我利用anodonda安装虚拟的Python环境：

打开Anaconda 命令提示符：



创建虚拟环境：

conda create -n HandTrack python=3.9

激活创建的虚拟环境

conda activate HandTrack

考虑到由于 **cvzone** 可能不在 Anaconda 的默认仓库中，通常需要使用 **pip** 来安装。首先确保 **pip** 在虚拟环境中是最新的：

conda install pip  
然后使用 **pip** 安装 **cvzone**：

pip install cvzone

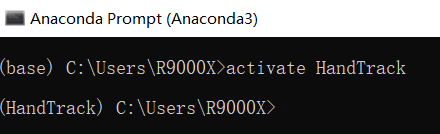
安装依赖项OpenCV：

pip install opencv-python

然后安装cvzone的依赖项：mediapipe

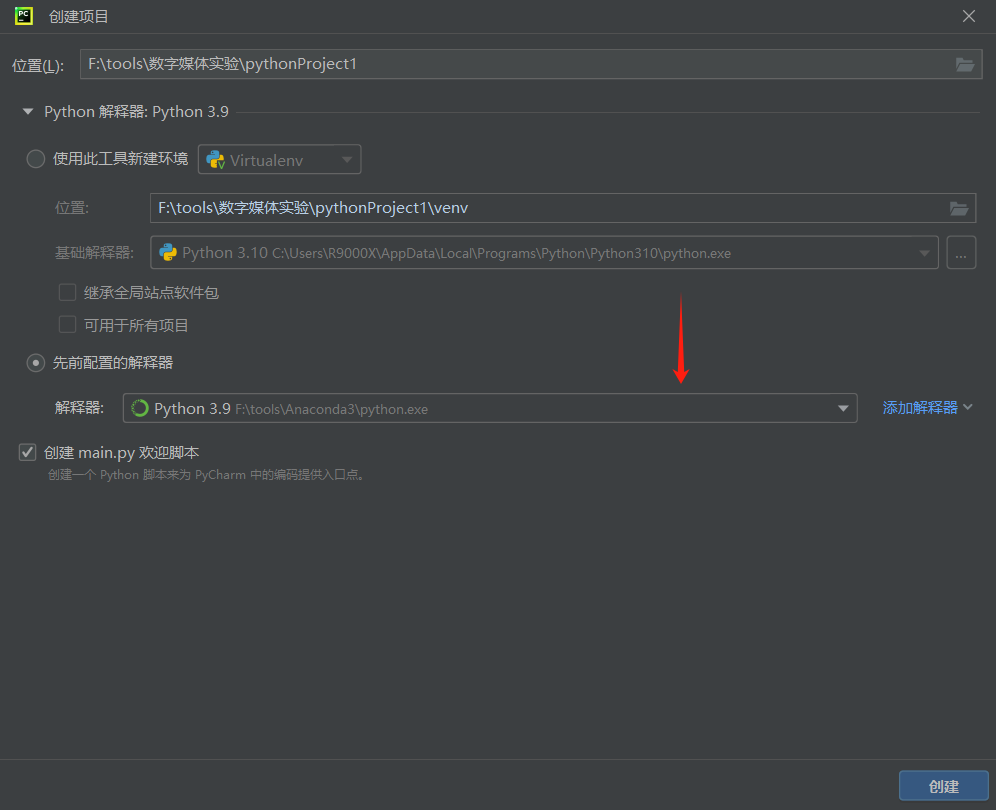
pip install mediapipe

**提示：**如果没有安装mediapipe，无法正确的调用cvzone中的手部识别的相关模块。

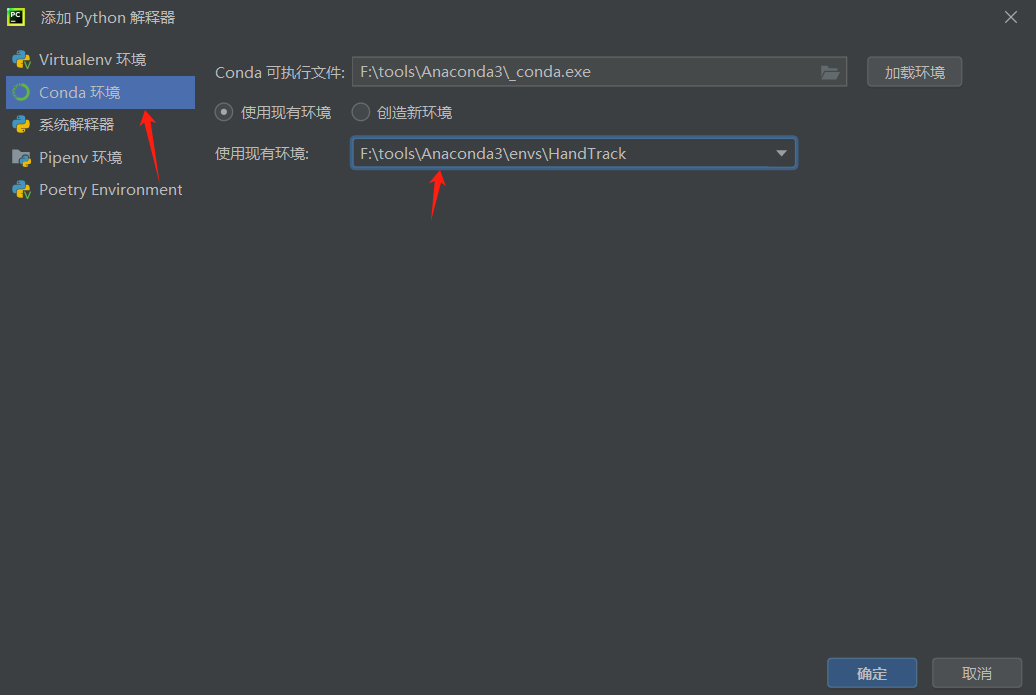


激活了HandTrack的虚拟环境。

#### 2-Python项目的创建：

打开pycharm，选择新建项目：  


选择添加解释器：



选择左侧的conda环境，同时在使用现有环境中选择刚刚创建好的虚拟环境：HandTrack

之后在创建好的新项目中，新建一个python文件：Test.py：



在python中编写的代码均在Test.py文件中呈现。

#### 3-利用python实现手部追踪：

##### 首先实现调用电脑摄像头实现每一帧的捕获：



Cap.set设置了捕获视频每一帧的长宽分辨率。此处设置了分辨率为 1280x720。

实时读取和显示摄像头捕捉的图像：

* while True：创建一个无限循环，用于持续读取摄像头捕获的视频帧。
* cap.read()：读取当前一帧图像并返回。success 表示读取是否成功，img 是读取到的帧图像。
* cv2.imshow：在一个名为 "Image" 的窗口中显示读取到的图像。
* cv2.waitKey(1)：在显示窗口中暂停 1 毫秒，允许用户进行窗口操作并使图像连续显示。如果用户在窗口中按下任何键，waitKey 就会返回对应的 ASCII 值。

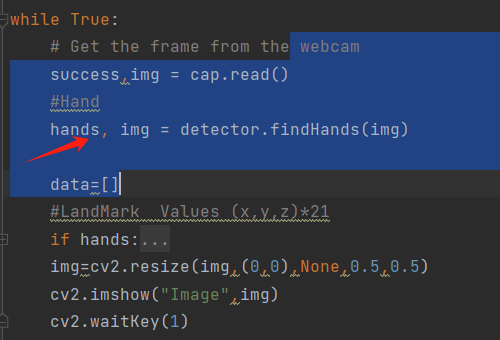
##### 实现手部获取：

定义手部探测器：



表述在一帧图像中最多只能捕捉一只手，同时置信区间为0.7

**如果置信区间设置为0.8，则由于实验场地相对昏暗，无法准确的识别手，识别效果差**



在实时捕捉视频帧中，检测手部，并且保存相关信息到hands中

#### 4-获取数据：

因为在cvzone中，捕获手其实是捕获手的21个关键的点。如实验结果中图一所示。

所以，我们要获取其中的21个关键的点的位置信息！

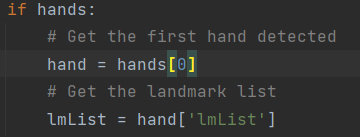
所以，在捕捉了手部之后，对齐数据进行如下操作：

* 初始化数据列表：用于存储从检测到的手部关键点中获取的坐标数据。



* 检查是否检测到手部，获取第一只手的关键点信息：

从hand 字典中提取关键点列表（lmList）。每个关键点是一个三元组 (x, y, z)，表示其在 3D 空间中的坐标。

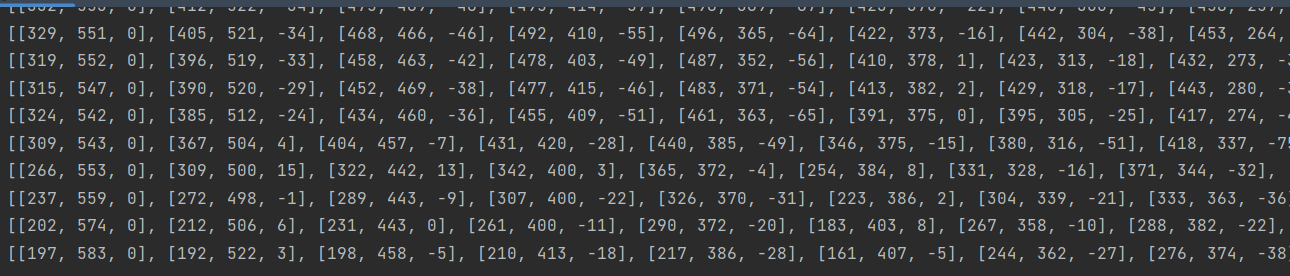


* 转换并添加关键点坐标：



data.extend([lm[0], height - lm[1], lm[2]])：将每个关键点的 x、翻转后的 y 和 z 坐标添加到 data 列表中。

**注意，这里如果用append，存储的格式中会有方框：（如下图所示）**



**同时，这里对于y坐标的处理，是因为OpenCV和unity中坐标原点的位置不相同。**

**在OpenCV中，坐标原点在左上角，而在unity中，坐标原点在右下角，正好相反，所以需要处理。**

#### 5-发送数据：（使用udp传送数据）

##### 创建套接字对象：



socket 是 Python 中提供网络通信功能的模块。

socket.AF\_INET：表示使用 IPv4 地址族。

socket.SOCK\_DGRAM：指明使用 UDP（用户数据报协议）来传输数据。

##### 设定目标服务器地址和端口：



这行代码将目标服务器地址和端口以元组形式赋值给 **serverAddressPort**，用作后续数据发送的目的地。

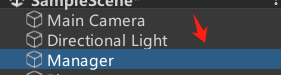
##### 发送数据：



* str.encode(str(data))：将数据 data 转换成字符串形式，再编码为 UTF-8 字节数组。这是因为套接字的发送和接收操作只支持二进制格式的数据。
* sock.sendto：将编码后的数据通过 sock 发送到指定的服务器地址和端口，即 serverAddressPort。

#### 6-Unity端接收数据：

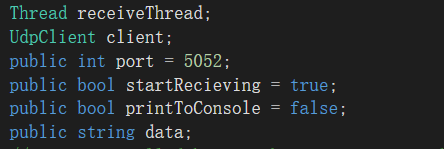
在新建创建的unity项目中，创建一个manager空对象，可以将接受脚本挂在在上面：





##### 具体的脚本实现如下：

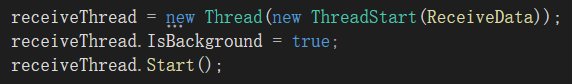
**定义需要的一些变量：**



* receiveThread：线程对象，用于异步接收数据。
* client：UdpClient 实例，用于在特定端口接收数据。
* port：接收 UDP 数据包的端口号，默认为 5052。（这里需要注意与在python程序中定义的端口号一致！）
* startRecieving：用于控制接收线程是否继续工作。
* printToConsole：决定是否将接收到的数据打印到 Unity 控制台。
* data：用于存储接收到的字符串数据。

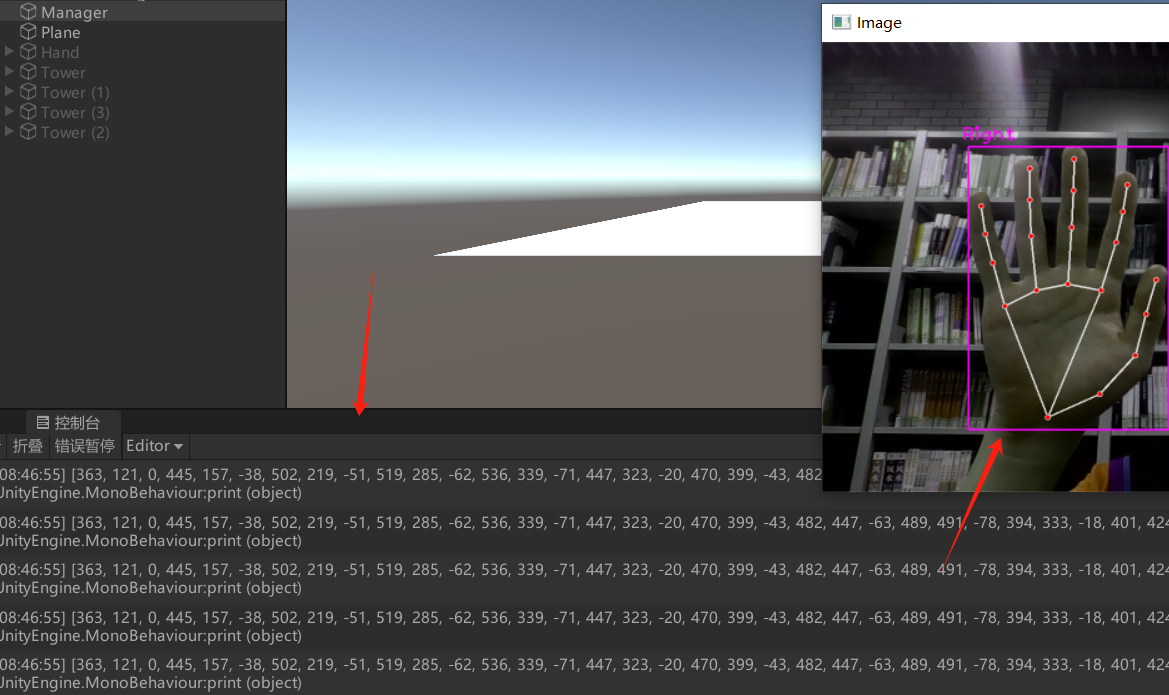
**Start 方法：**

* 创建一个新的线程 receiveThread，目标方法是 ReceiveData。
* 将 receiveThread 设置为后台线程，以便主线程结束后它也能自动结束。
* 启动线程，使 ReceiveData 方法开始执行。



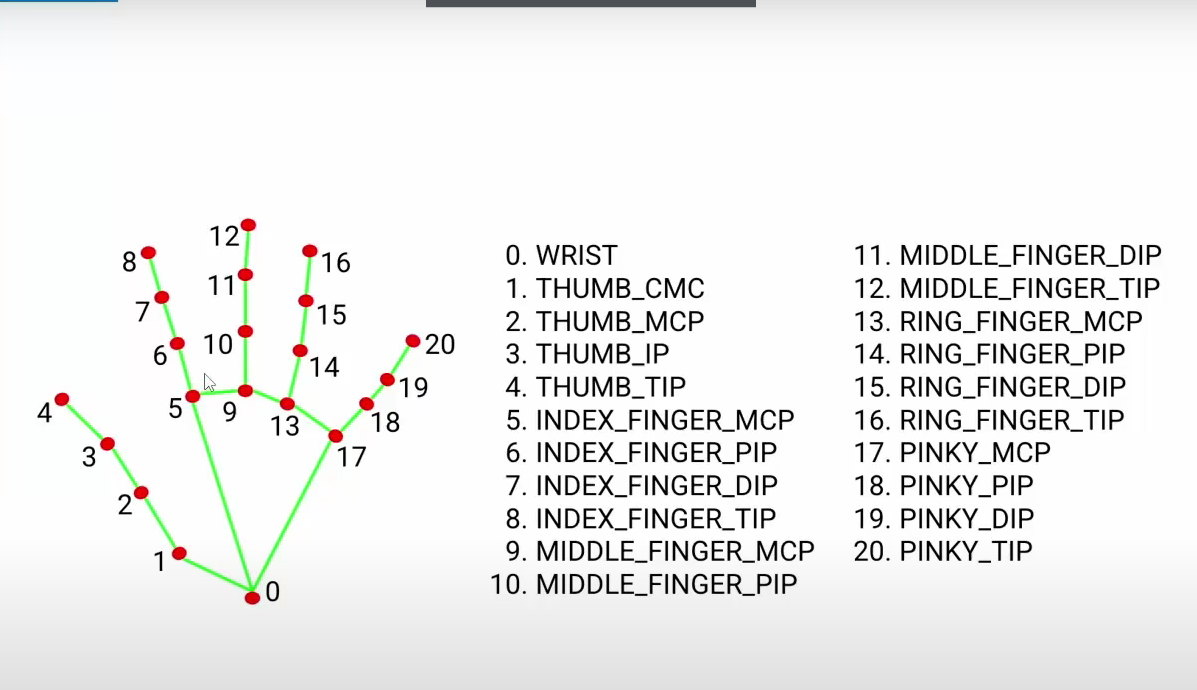
**ReceiveData 方法：**

* **UdpClient 初始化：**
  + client = new UdpClient(port)：创建 UdpClient 实例，并绑定到指定的端口号 port，用于监听该端口的数据包。
* **循环接收数据：**
  + while (startRecieving)：当 startRecieving 为真时，保持循环接收数据包。
  + IPEndPoint：用于存储接收到的数据包的来源地址。
  + client.Receive：接收一个 UDP 数据包，将来源地址存储到 anylP，并返回数据的字节数组。
  + Encoding.UTF8.GetString：将接收到的字节数组转换成 UTF-8 编码的字符串。
  + data：将字符串存储到类的公共字段 data 中，以便其他脚本或方法访问。
  + printToConsole 判断条件：如果 printToConsole 为真，则将接收到的数据打印到 Unity 控制台。
* **异常处理：**
  + try-catch：捕获接收过程中可能出现的异常，将异常信息打印到控制台。

将传送得到的数据在控制台中 输出如下：  


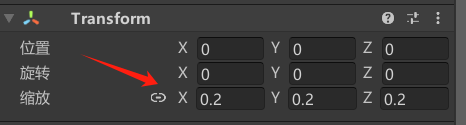
#### 7-在unity中创建手部模型：

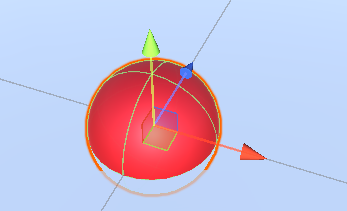
根据如下的示意图，我们需要创建21个关键点与21条在其中相连的直线。



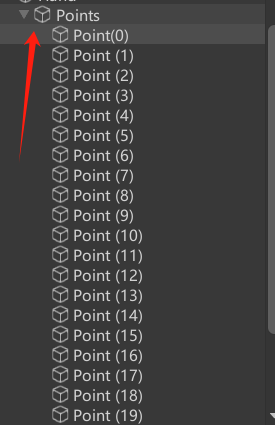
##### 创建关键点：

我们在场景中创建21个球体，并且将其缩放为0.2。为每个小球加上红色的材质：



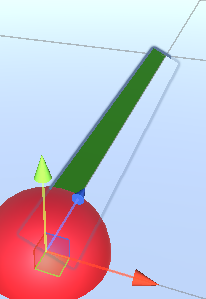


将这些小球放到Points空节点中方便管理。

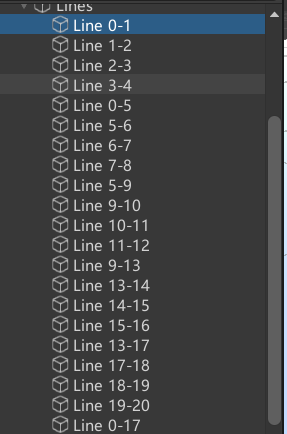


##### 创建连接关键点的线：

在场景中新建一条线：并给予它绿色的材质。



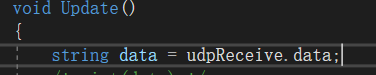
然后复制20条同样的线，把他们放到Lines空对象下方便管理：



#### 8-手部追踪代码实现：（关键点的绘制）

##### 从UDP接收传送过来的数据：

public UDPReceive udpReceive;



在update中实时接受传递过来的数据。

在unity属性面板中需要制定udp的对象：

将manager对象拖到如下位置即可。



##### 处理数据：

因为传过来的数据中开始和结尾有方括号，因此将其去除：



之后，因为传过来的是一个字符串，每一个坐标之间都用“,”隔开，因此，就需要去将其分割开来：



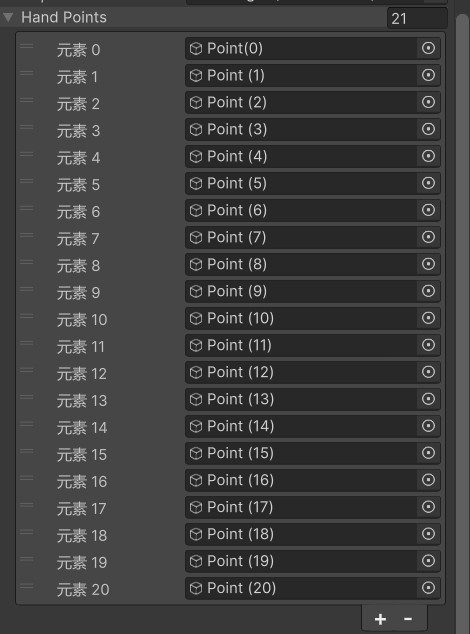
使其变成一个个的坐标，并且存入到points中。

##### 将数据依次赋值给点：

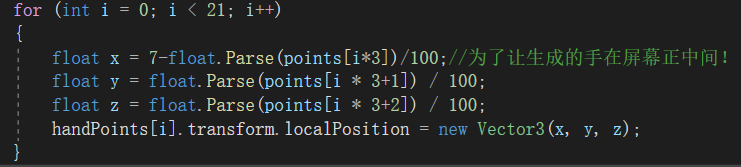
首先需要定义指定的点的变量：



在unity的属性面板中制定这些点：



然后定义循环，将这些值赋给这些点：



在其中，需要注意，数据的存放都是xyzxyz这样存放，因此，在循环的时候，获取每一个点的xyz坐标的时候循环变量都是如下的形式：

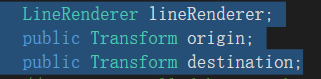
i\*3，i\*3+1，i\*3+2

后续的操作是为了尽可能的让手部模型出现在屏幕的中央，因此就需要我对于x的坐标进行处理。

#### 9-绘制直线：

##### 编写脚本：LineController

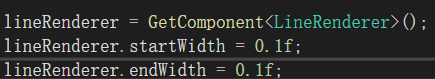
定义需要的变量：



* LineRenderer lineRenderer：一个私有变量，存储 **LineRenderer** 组件的引用。LineRenderer 是 Unity 中的组件，用于绘制3D空间中的线条。
* public Transform origin：公开的 Transform 变量，表示线段的起点
* public Transform destination：公开的 Transform 变量，表示线段的终点

start方法：

获取到同一游戏对象上的 LineRenderer 组件，同时设置绘制的直线起始和终止的宽度。



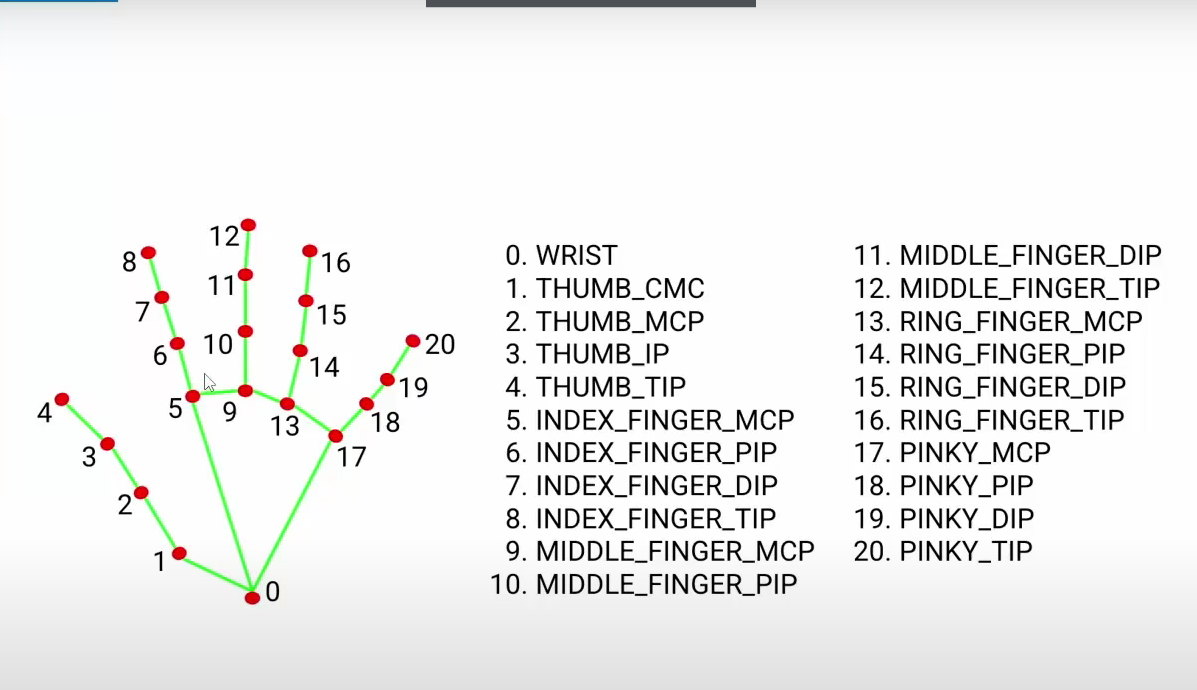
Update方法：

不断的更新执行案的起始和终止坐标。

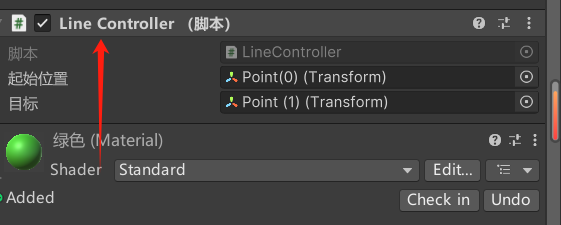


##### 指定每条线段开始和结束的点：

按照入下图所示，绘制关键点之间的线段，使其成为一个简易的手部模型。

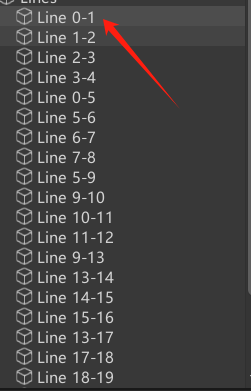


首先将这个lineController脚本挂载在每一条线段上：



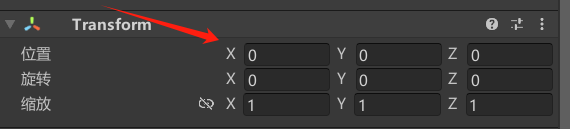
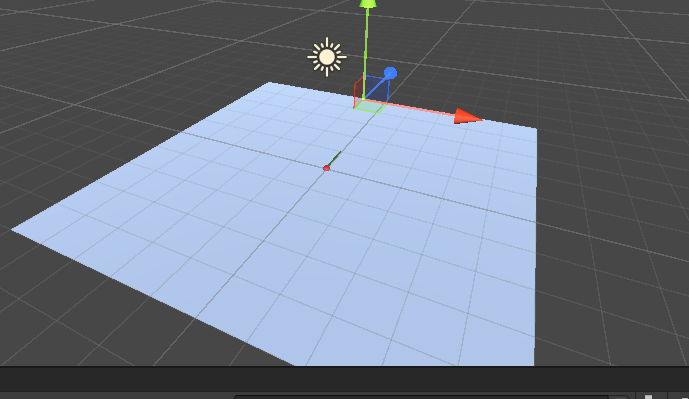
其次，根据上面的图为每一条线段指定对应的起点和终点坐标（直接拖动点到这个属性面板中即可）

为了防治点指定错误，可以给每条线重命名：



#### 10-创建虚拟环境：

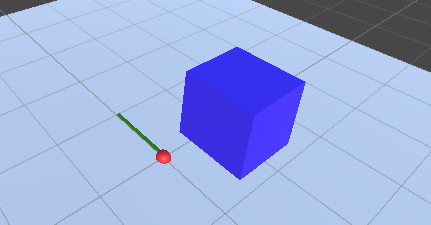
##### 在场景中创建一个平面，并重置其位置，使其中心在坐标原点



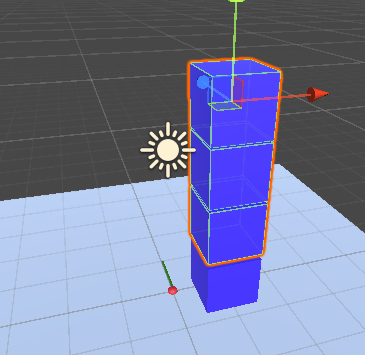
##### 创建塔楼模型：

首先创建一个立方体，然后调整其大小，并且给其添加上蓝色的材质。

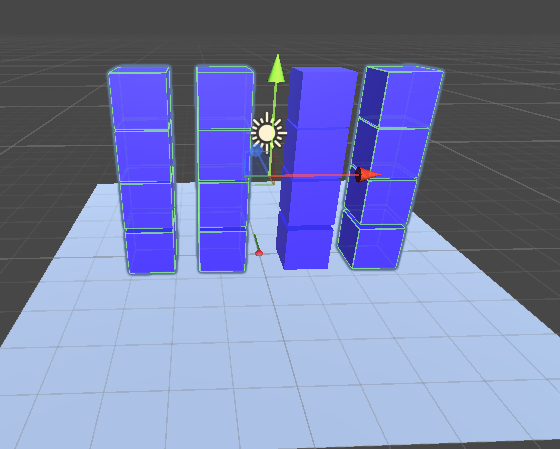
调整其位置使其能够被手部模型碰到



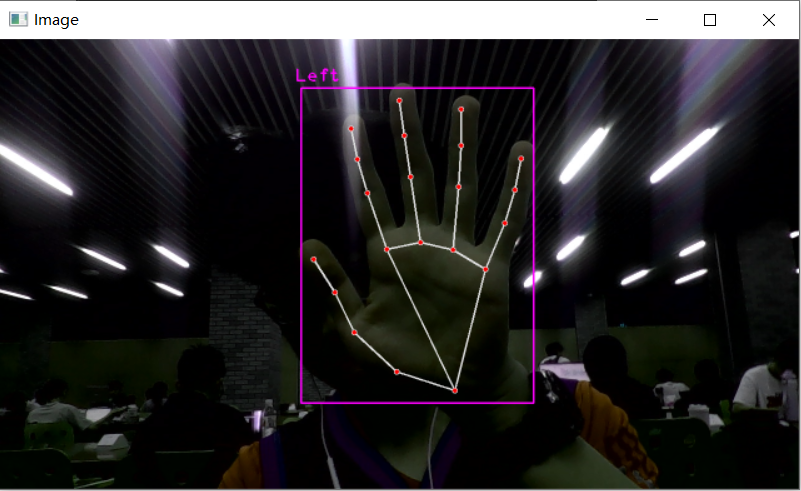
然后复制3个块，使其成为一个简单的塔。



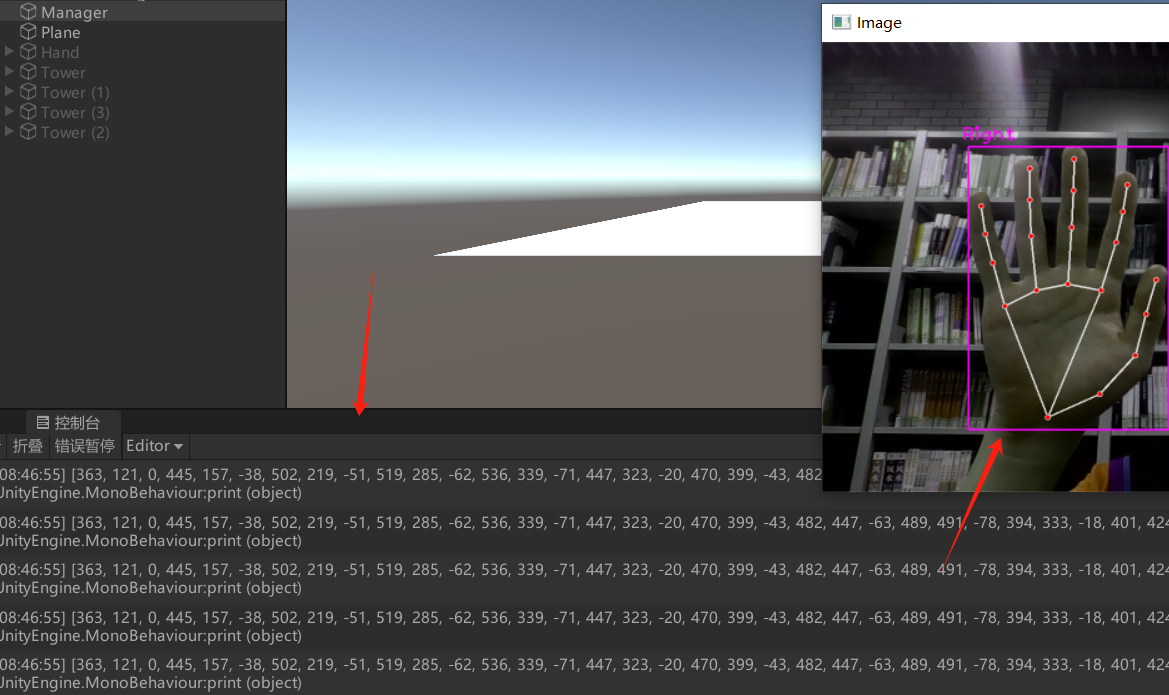
之后复制这个塔，实现最终的塔楼群模型。



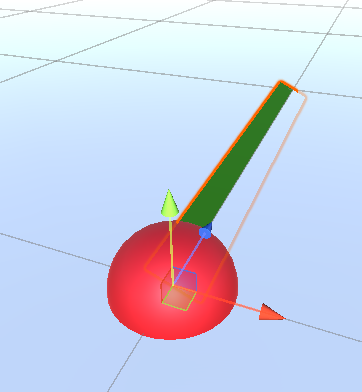
# 实验结果:



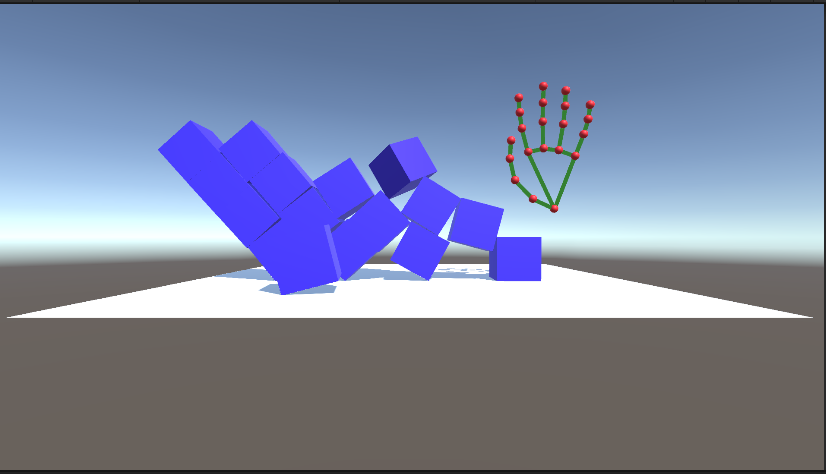
抓取得到的手部的相关信息



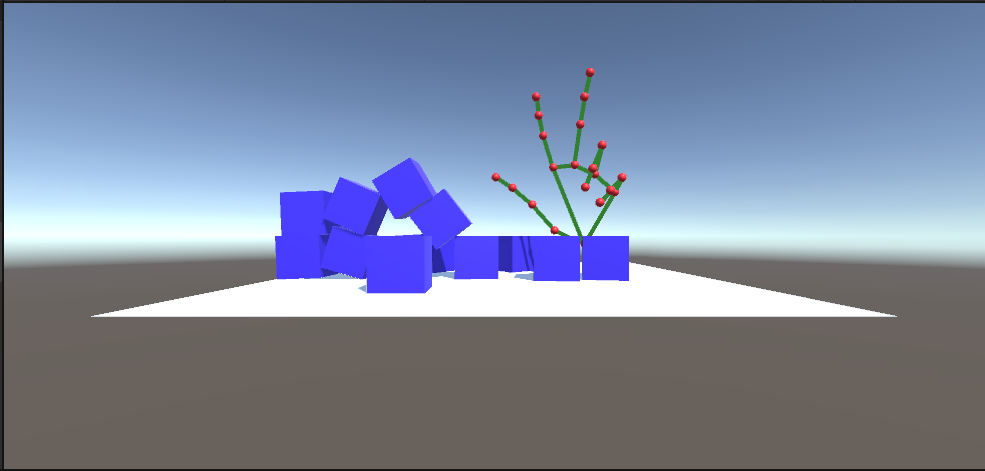
可以看到，在python程序中检测到手之后，其关键点的信息都会传送到unity中，相关的信息在unity中展现。

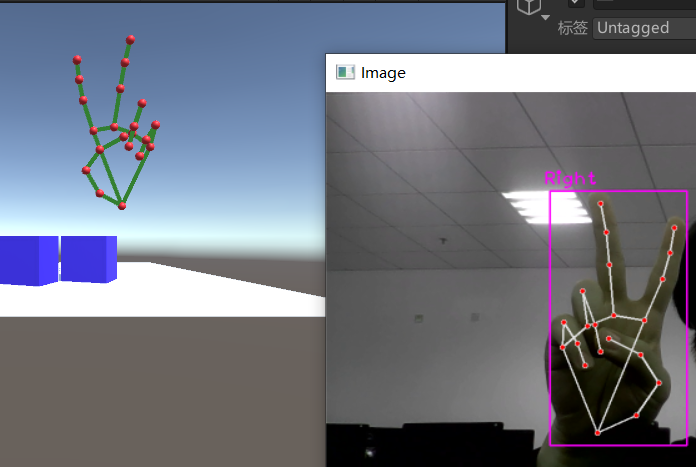


在场景中创建的点（球体）和线。



手部轻微碰撞塔楼之后塔楼倒塌





实时渲染手部动作以及其姿势。

具体实验结果见演示视频。

# 总结:

1. 手部识别过程中，环境是否足够亮非常影响其识别精度。

我一开始测试代码的时候，发现

1. 在实践的过程中了解到了opencv和unity中坐标系基础设置之间巨大的差异。

在一开始实验的时候，我对这两个开发工具的坐标设置没有注意，导致在unity中绘制的手部模型的位置与实际偏差太多！，后来通过上网查找后，明白了原有，也让我最终修改过来！

1. 在python中存储点的数据的时候一定要注意格式。

一开始实验的我没有注意到这一点，发现等在unity中接收到数据之后才发现在python中记录的数据的格式并不是我想要的，在这里我调试了很久代码，终于实现了对数据格式正确的修改！

在完成这次实验的时候，我对于增强现实，计算机视觉，计算机网络等相关知识有了更加深入的理解。因为我没有用传统的增强现实的sdk包进行开发，这反而能够让我从python编程的角度更加理解了那些sdk包中的各种功能。而我调用的**cvzone**和medipipe包，让我对计算机视觉有了更加深入的理解。让我明白了增强现实，其实就是一个计算机视觉识别到了指定物体之后，然后再在其环境中增加图形。

因为这次实验中我用了两种不同的语言，如何在两个不同程序间完成数据的传输，尤其是实时传输，是一个巨大的挑战。所幸我在查阅资料的时候，了解到了udp传输协议，这可以帮助我快速的实现数据在两个程序之间的传输，已达成预期的效果。

# 参考资源:

https://blog.csdn.net/CodeRoarX/article/details/132969372

<https://people.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html>

<https://blog.cloudflare.com/everything-you-ever-wanted-to-know-about-udp-sockets-but-were-afraid-to-ask-part-1>

<https://juejin.cn/post/7268955025211424779>

https://blog.csdn.net/weixin\_45930948/article/details/115444916