# 虚拟平台搭建

## 2.1 3Dmax简介

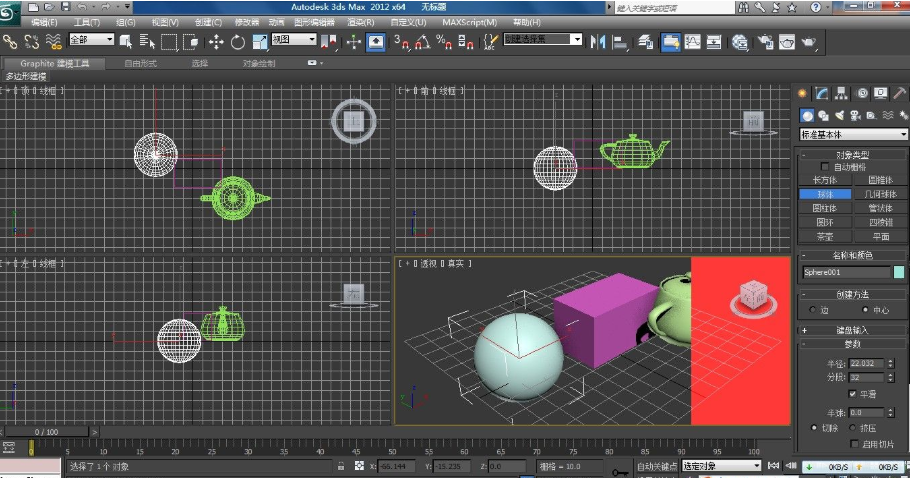
3D Studio Max（常简称为3D Max）也是最新版Autodesk 3ds Max的前身，由Autodesk Media and Entertainment开发和制作，是一款用于制作3D动画、模型、游戏和图像的专业3D计算机图形程序。3Dmax具有强大的建模能力和灵活的插件架构，可以在Microsoft Windows平台上使用。它被广泛应用于游戏开发、商业电视制作以及建筑可视化等工作，也用于电影特效的实现和电影预览的制作。对于它的建模和动画工具，3D Max的最新版本还包含渲染器（如环境遮挡和表面散射等）、动态模拟、粒子系统、辐射着色、贴图创建和渲染、全局照明等，可以定制用户界面、新图标和自己的脚本语言。

除了强大的建模和处理功能，3D Max还具有以下的突出特点。首先它具有非常高的性价比，相比较于它本身强大的建模、渲染功能，它的价格显得相对较低，一般的制作公司和使用者就能够承受并使用，这样也能够大大降低一些作品的制作成本。3D Max系列软件能够很好地适应Windows系统，而且对电脑硬件的要求也相对较低，一般电脑的配置就可以基本满足软件的使用需要，这也给广大软件使用者提供了极大的便利。其次，3D Max在我国拥有最多的使用者，网络上也有大大小小的活跃论坛在共同探讨、交流软件的教程、使用心得等，这为软件的入门提供了丰富的学习资源。此外，软件本身的使用门槛较低，使用3D Max的制作流程十分简洁高效，初学者也可以很快上手，制作出简单的作品。

在本次项目的虚拟平台设计中，主体虚拟手部分的模型原型使用3D Max平台进行设计，并将设计好的手部分导入到unity中完成后续制作。



**图2-1 3DS MAX**



**图2-2 3D MAX操作界面演示**

## 2.2 Unity简介

Unity是由Unity Technologies开发的跨平台游戏制作引擎，主要用于开发二维或三维游戏，也可以用于制作建筑可视化、实时三维动画等交互内容。Unity的编辑器可以在Windows和Mac OS X下运行，其发布的游戏可以在包括iOS、Android、Mac等27个平台运行，也可以利用其Unity web player插件发布网页游戏。



**图2-3 Unity标志**

Unity使用交互式的图形化开发环境，使用者通过可视化界面结合程序脚本来实现开发目标。Unity中支持大部分3D模型文件，自动瞬时导入，并将模型贴图转化成Unity适合的格式文件，方便开发者现有资源的移植。此外，Unity的脚本文件适用的编程语言包括JavaScript、C#、Boo等，给使用者提供多种选择，便捷高效。

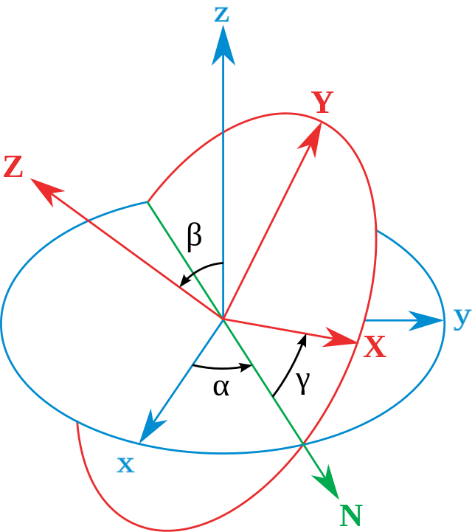
**图2-4 unity界面**

## 欧拉角与四元数

Unity中使用的角度是以四元数的形式表示，而不同于我们平时所常见的欧拉角形式，由于在Unity中控制角度要频繁地使用到四元数，故在此对欧拉角、四元数以及两者的关系进行介绍。

在传统以欧拉角来度量角度的三维空间坐标系中，任何刚体子坐标系的取向都可以使用三个欧拉角来表现。设全局坐标系的三根轴分别为x、y、z，称x-y平面与刚体坐标系中的X-Y平面的交线称为交点线，则欧拉角可以定义如图2-5所示：

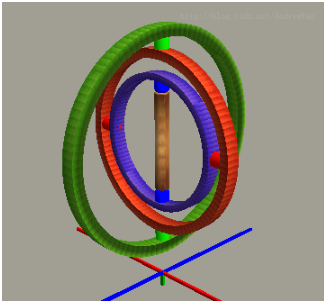
* 是x轴与交点线的夹角；
* 是z轴与Z轴的夹角；
* 是交点线与X轴的夹角。



**图2-5 欧拉角定义**

然而欧拉角在使用的时候会遇见两种问题，一种问题是关于旋转的次序问题，尽管用欧拉角表示的旋转十分直观，容易理解，但是不同的旋转顺序会导致完全不同的旋转结果，同样的欧拉角可以用不同组合来表示，这样给欧拉角使用时的定义带来了一点麻烦。

使用欧拉角的另一个重大缺陷则是出现万向节死锁（Gimbal Lock），即在某种特殊的情况下，欧拉角将失效的问题：旋转矩阵是依次进行的，假设先围绕x轴旋转，再围绕y轴旋转，最后围绕z轴旋转，这就导致物体其实是围绕自己的X轴旋转，而不是世界坐标的X轴旋转。【https://blog.csdn.net/andrewfan/article/details/60981437】万向节死锁是由欧拉旋转本身的定义所造成的，这种围绕选旋转前固定轴的先Z、再X、再Y的旋转操作，与其最终所预期的三个轴向可以旋转的结果并非一定是一对一的映射。某些情况下是多对一的映射，造成一些旋转自由度的缺失，也就是“死锁”。



**图2-6 万向节死锁示意图**

相比较欧拉角而言，用四元数表示角度则不存在万向节死锁的问题。四元数是由 William Rowan Hamilton 于 1843 年在爱尔兰发现的数学概念，并作为复数的扩展。1985年 Shoemake 将其引入计算机图形学领域。从此四元数在计算机图形学、计算机动画、计算机视觉和虚拟实现等领域获得了广泛运用。基于四元数的坐标旋转方法由于其较传统欧拉角法具有高效、简洁且意义明确的优点而得到了广泛使用。【四元数文献的引用】

四元数定义i 、j、k三个虚数单位参与运算，并有以下运算规则

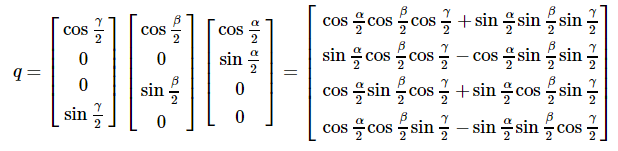
一个普通的四元数可以表示为：

又因为四元数的i 、j、k之间乘法的性质与向量之间的叉积结果形式很类似，所以四元数还有另一种表示形式：

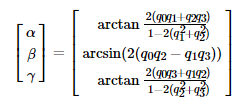
四元数的加法运算与复数类似，遵循实数和复数的所有交换律和结合律

四元数的点积等同于一个四维矢量的点积，即两个四元数中每个元素数值的一对一乘积：

当我们使用欧拉角实现物体在空间的旋转时，假设绕z、y、x轴依次旋转，使用分别表示物体绕x、y、z的旋转角度，则此时欧拉角转换成四元数的矩阵表示为：



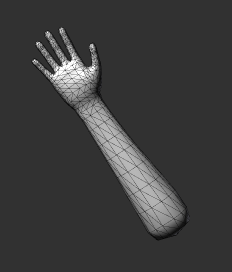
对公式求逆解即可得到由四元数到欧拉角的转换：



这样我们就得到了四元数和欧拉角之间的转换关系，并在之后Unity中调整虚拟手的初始旋转角度、最终旋转角度以及计算虚拟手的旋转角度数值时能够更好地理解和应用。

## 虚拟平台搭建

首先，我们在3D Max软件中搭建我们所需要虚拟手的基本结构。由于数据手套选择以右手为使用手来进行制作，因此我们虚拟平台对应的虚拟手也选择右手作为演示模型。从虚拟手各个手指开始，每个手指设置三个关节，包括远节指间关节（指甲盖下的第一个关节）、近节指间关节（远节指尖关节下面的关节）和掌指关节（手指与手掌连接处的关节），以关节为分界对手指段分别建立模型，并将各个手指关节之间连接、各手指与手掌部分连接，将手掌部分与前臂模型连接，这样我们就得到了一个简单的手部3D模型。在3D Max中对虚拟手原型进行网格划分，并选用了一种和人手类似颜色的纹理对整个模型进行渲染，让整个虚拟手显得更加逼真。

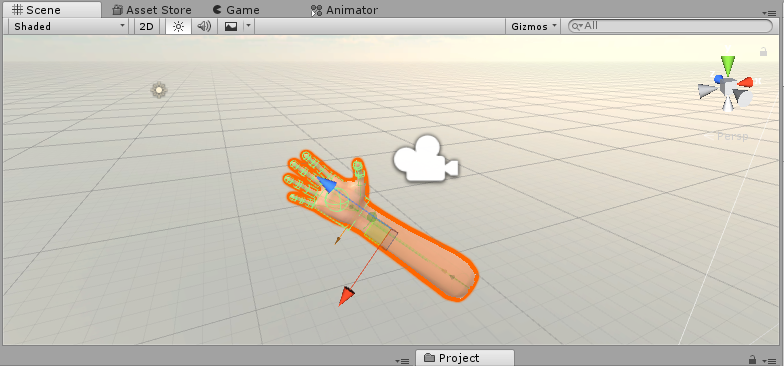


**图2-7 虚拟手的3D原型和网格划分**



**图2-8 着色后的虚拟手**

虚拟手的3D原型构建完毕，我们可以将其直接拖入Unity中作为Unity中的预设物体，这样就可以在之后的更改中直接更改预设对象，场景中的所有相同预设下的游戏对象（Game Object）也会随之改变，这样会使我们的设计工作更加简洁高效。



**图2-9 Unity场景中的虚拟手**

## 虚拟手活动功能设计

虚拟手的原型部分已经搭建完毕，接下来我们开始针对虚拟手活动所需要的基本功能进行脚本的编写。虚拟手进行活动的几项基本功能包括：

1. 各个手指的弯曲以及全部手指的抓握；
2. 手腕关节和小臂的旋转；
3. 对各个手指运动时与手掌所成夹角的读取。

由于后续与虚拟手进行数据通讯的数据手套使用角度传感器进行角度的传输，而并未使用提供位置信息的传感器，所以尽管在Unity平台中可以实现摄像机主视角下的手部平移，但是对于实际的演示并没有太大意义，故在设计时并未考虑手部的位移通讯传输，并采用固定小臂底部的方式获得更好的视角。



**图2-10 固定小臂后的视角**

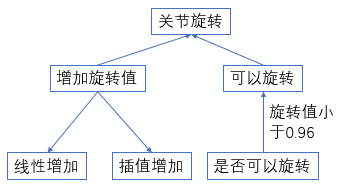
### 2.5.1 基础类的构建

首先我们需要构建FingerPart类来表示手指的各个关节部分，对于每一个关节，需要的参数包括初始时的关节角度、最终能够旋转到的角度、旋转曲线、旋转速度等数值参数，以及关节是否允许旋转、是否正在旋转、是否处于中性位置等逻辑参数。

**表2-1 FingerPart 类的变量说明**

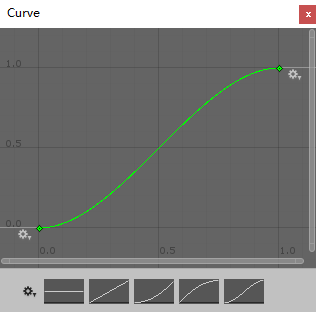
|  |  |  |
| --- | --- | --- |
| 变量名称 | 变量类型 | 说明 |
| OpenHandRotation | Quaternion | 初始时的关节角度 |
| TargetRotation | Quaternion | 最终能够旋转到的角度 |
| RotationCurve | AnimationCurve | 旋转时参照的曲线 |
| RotationValue | float | 旋转值 |
| IsRotating | bool | 关节是否正在旋转 |
| IsRotationAllowed | bool | 关节是否允许旋转 |
| \_mbendSpeed | float | 旋转速度 |
| IsNeutral | bool | 关节是否处于中性位置 |

接下来搭建FingerPart类中的几个核心函数，判断关节能否旋转、控制关节旋转以及增加旋转数值。将旋转值RotationValue归一化处理后限制在0到1之间，考虑到误差大小设置阈值为0.96，即当旋转值大于0.96时，即判定关节已经达到旋转终值，此时不能旋转。只有当关节正在旋转且允许旋转时，对关节的初始角度和最终角度按照手动调整的旋转值增加模式（包括线性和插值）进行处理，使关节旋转到指定位置。而要实现关节的复位，只需要将关节的目标角度设为复位角度，将关节的初始角度设为当前角度，即可实现关节复位的操作。



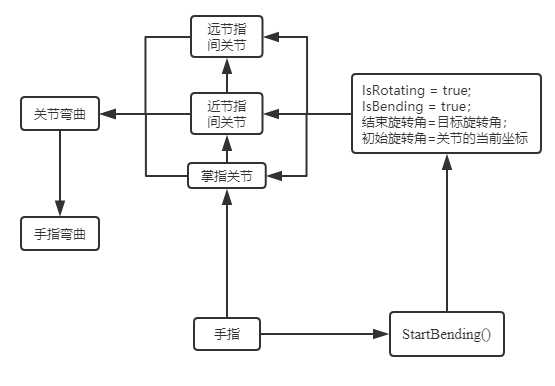
**图2-11 关节旋转控制流程**

在实际的脚本编写中，当旋转模式（RotationMode）使用线性增加时，新增的数值将直接累加在原有数值上，而当使用插值处理时，采用mathf中包含的smoothstep函数对旋转值进行处理，使其能够非线性的增加，其插值处理的旋转曲线由图2-12所示。



**图2-12 插值曲线**

下一步搭建用来表示整根手指的Finger类，在Finger类中导入表示关节的FingerPart类，这样我们就可以得到带有三个关节的手指表示。Finger类中的关键函数为控制整根手指共同弯曲的函数StartBending()，通过遍历类中所有的关节部分，将关节的开始旋转角度设置为关节的当前角度，终止旋转角度设置为目标角度。由于各关节是手指部分的子物体，近节指间关节采用的旋转角度坐标是基于掌指关节的子坐标系，远节指尖关节采用相对于近节指间关节的子坐标系，这样的坐标系关系可以在调整关节位置时更直观地显示某根关节在当前相对参考系的位置，方便旋转参数的不断调试。而当要实现手指的弯曲时，该手指各关节的IsRotating属性被同时设置为真，手指的IsBending属性也被设置为真，就可以实现各个关节共同旋转至目标旋转角度，进而实现整根手指的弯曲效果，如图2-13所示。



**图2-13 手指的控制示意图**

### 2.5.2 主要控制部分的搭建

主要控制部分的函数HandPhysicsController()包括对虚拟手腕部关节转动角度的控制、手指关节的运动控制、对某一根手指的运动控制等。主线程中首先初始化了各个变量，并将手指类、关节类实例化，设置手指包括拇指、食指、中指、无名指和小指，手腕在此处也作为手指类下的一个变量进行实例化，其中每根手指包含三个关节。

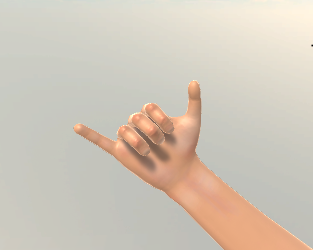
针对手腕处关节转动，首先初始化转动开始时的角度和旋转速度，然后对腕部的转动幅度作归一化处理，分别设置上下界为+1和-1，单位值即为旋转速度，通过角度累加的方式来实现腕部关节的旋转，如图2-14所示。



**图2-14 虚拟手腕关节旋转**

在手指关节的运动控制部分PrecisionRotate()中，通过设置关节的索引数和以四元数形式表示的旋转角度，确定要控制的具体关节和该关节旋转的具体角度，然后改变关节的逻辑属性IsRotating为真，并将目标角度赋值为需要的旋转角度，就可以实现对某一具体关节的旋转。此外，还可以通过重载函数的形式，增加旋转速度为函数的参数，更加精细地控制关节的旋转过程。

而在控制手指的部分，主线程中的大部分重载函数都用到了上述Finger基础类中的StartBending()成员函数。例如当我们想要弯曲某一根手指时，设置参数为某一根手指的索引值，然后在函数内部调用变量在该索引下的成员函数StartBending()，即可实现对该手指的弯曲，如图2-15所示。同理若要实现虚拟手的握拳效果（所有手指弯曲），只需要构建循环遍历所有的手指索引即可，最终效果如图2-16所示。



**图2-15 对特定手指的弯曲控制**



**图2-16 对五根手指的控制**

主线程中控制手指弯曲的函数已经搭建完毕，接下来我们需要接入鼠标和键盘的控制，便于后续程序的测试。Unity的库中提供了关于各个按键的键码（Keycode），首先我们将各个变量与按键键码相对应，方便后续控制的参数输入。设置键盘输入的勾选框，当勾选框被选中时，启用键盘输入。Unity提供了GetKeyDown和GetKeyUp两个函数用于检测对应的键盘按键是否按下或松开，所以只需要当检测到键盘（或鼠标）对应按键被按下时，调用对应手指的成员函数StartBending()，而当该按键松开时，调用成员函数StopBending()，就能够实现键盘和鼠标对虚拟手的控制。

**表2-2 各按键键码与控制**

|  |  |  |
| --- | --- | --- |
| 按键 | 键码 | 控制 |
| A | KeyCode.A | 小指 |
| S | KeyCode.S | 无名指 |
| D | KeyCode.D | 中指 |
| F | KeyCode.F | 食指 |
| Space | KeyCode.Space | 拇指 |
| 鼠标左键 | KeyCode.Mouse0 | 五指 |
| 鼠标右键 | KeyCode.Mouse1 | 手腕 |

## 本章小结

在第二章中，我们使用3DMax建立了虚拟手的原始3D模型并进行了模型渲染，在Unity中载入模型，确定了各个关节、手指的从属关系，并结合Unity的脚本编写，实现了用键盘对虚拟手的控制。通过手动调节各个关节旋转的最终角度，使得虚拟手各部分在实现弯曲效果时更加自然逼真。

# 数据手套通讯与控制

用户通过佩戴数据手套进行康复训练，数据手套上搭载的角度传感器获得用户手部的角度信息后通过USB连接的方式将其传输至电脑端，在电脑端对角度信息进行处理之后发送到服务器端，然后服务器端分析数据将角度信息同步至虚拟平台的虚拟手，实现可视化的效果。本章我们将搭建服务器端用于接收角度数据，实现数据手套和虚拟手的同步。

## 3.1 UDP简介

UDP是User Datagram Protocol的简称，协议全称是用户数据报协议。在网络中，它与TCP协议一样用于处理数据包，是一种无连接的协议。在正式通信前，不必与对方先建立连接，直接向接收方发送数据，也是一种不可靠的通信协议。正是由于UDP协议不关心网络数据传输的一系列状态，使得UDP协议在数据传输过程中节省了大量的网络状态确认和数据确认的系统资源消耗，大大提高了UDP协议的传输速度，而且UDP无需连接管理，可以支持海量并发连接。【可靠UDP数据】

UDP本身的控制机制并不是十分复杂，通过IP来提供面向无连接的通信服务。它在收到发送过来的数据的同时，将数据原样发送到网络上。所以这种“Best effort”的发送方式，在出现网络拥堵的情况时，UDP并不能提供有效的数据流量控制来避免阻塞。而如果在传输数据的过程中，出现了数据丢失的情况，那么UDP服务也不会重新发送数据，当传输数据到达服务器端的顺序被打乱时，也没有提供纠正的功能。

相对于TCP协议对信息安全的重视，UDP这种“不可靠”的协议看起来似乎并不安全。但是早期TCP复杂的连接建立方式使得数据传输的速度较慢，占用系统资源较高，数据传输的效率比较低。而UDP这种无连接的传输方式，其传输速度非常快，在一些对单条数据信息的可靠性要求不必太高的场景中获得了很好的应用，是分发信息的理想协议。在某些高频信息传输场景下，即使传输数据的过程中偶有信息丢失，在很短的时间内也会有另一条新的消息替换它。

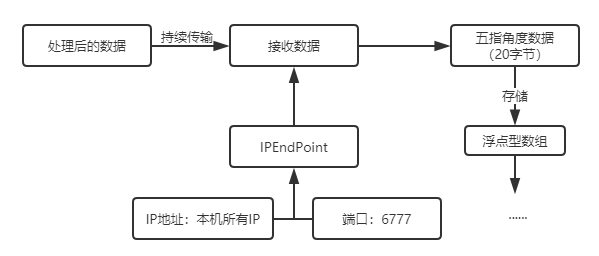
考虑到数据手套向电脑端发送数据的时间间隔为10ms，因此可以选用高速的UDP协议传输作为数据处理系统和虚拟平台之间的数据传输方式。

## 3.2 服务器端搭建

Unity的场景运行是逐帧刷新的，如果将接收UDP传输数据的接收部分放在主线程中，很容易出现程序的卡顿等情况，因此采用多线程技术，通过新建进程来接收数据。

首先我们实例化一个新进程，并在程序开始部分确定IP地址和端口，并将IP地址和端口绑定在一个IPEndPoint上。在此我们选择使用IPAddress.Any来表示本机上的所有IP地址，选择6777端口作为通信端口，并初始化了浮点型数组和计数变量用于接收传输的角度数据和对数据进行计数。

通过C#提供BitConverter.ToSingle函数对接收到的字节数据进行转换，使其转换成浮点型数据并存储在数组中。数据手套每次发送的数据是五根手指的数据，按照从拇指到小指的顺序排列，故用于存储数据的数组容量为5，需要调用时根据数组索引即可获取相应手指的角度值。数据手套发送的浮点型角度值转换成字节发送时，每个数据长度为4个字节，所以五指的数据总长为20字节，读取时按每四个字节进行数据存储。保持此线程的工作状态，即可实现对数据的接收，接收过程如图3-1所示。

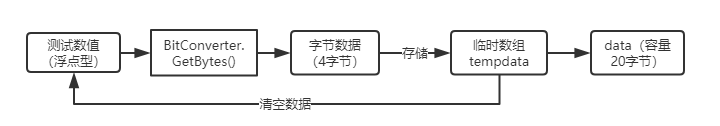


**图3-1 服务器端接收数据**

## 3.3 模拟用户端搭建

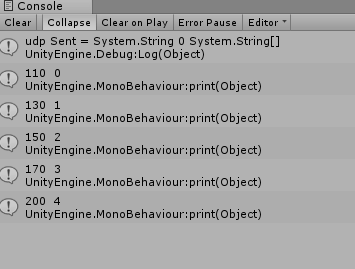
实际开发过程中需要测试接收数据的效果，因此我们在虚拟平台也搭建了一个UDP数据传输端来模拟数据手套的数据传输过程。同样地，使用本机IP地址“127.0.0.1”，将IP地址与6777端口绑定到同一个IPEndPoint上，然后向接收地址连续发送数据。

对于模拟数据的构建，我们首先创建了一个容量为20的字节数组data用于存储（实际传输时的数据大小为20字节）。Unity自带函数BitConverter.GetBytes()可以将单个浮点型数值转化成4个字节，所以我们还需要一个字节数组类型的临时变量tempdata用来转存数据。其具体转化过程如下：首先BitConverter.GetBytes()将第一个浮点数值转化为4个字节存储在tempdata，然后使用tempdata对data进行逐索引赋值，然后转化第二个浮点数值，并将新转化得到的字节添加在data数组中，以此类推。当整个过程完成后，我们就可以获得一个20字节的数组，整个流程如图3-2所示。



**图3-2 模拟数据的存储过程**

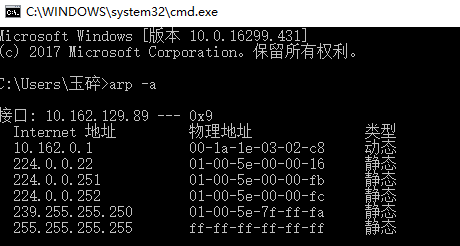
字节数组创建后，我们就可以利用Unity中自带的成员函数Client.Send()来编写发送测试数据的函数SendFloat()：首先在函数中创建临时变量tempdata，并通过循环添加的方式实现上述20字节数组的构建，最后将包含五个数值的字节数组通过Client.Send()发送到接收端的IPEndPoint，就可以从服务器端读取接收到的数据。实际测试数据传输时效果如图3-3所示。



**图3-3 服务器接收测试数据**

## 3.4 通讯测试

经过模拟用户端的数据测试，虚拟平台可以正常接收20字节的数组数据。接下来我们尝试服务器端和用户端建立连接并发送数据测试。我们将两台电脑接入同一无线网络信号，并通过命令行查询服务器端IP，以提供给用户端发送数据时所需要传输的网络地址，如图3-4所示。



**图3-4 服务器端IP地址查询**

在用户端输入服务器端的IP地址和开放端口后，通过实际用户端的程序控制发送数据包进行测试，并在控制台实时显示数据，如图所示，此处放发送数据

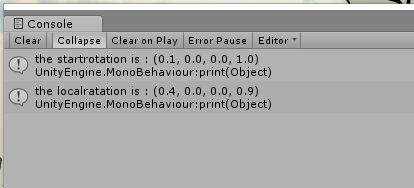
## 3.5 手套控制

### 3.5.1 数据处理

从数据手套发送至服务器端的数据是长度为20字节、包含五指角度信息的数据包，其中角度的形式是范围在0到180度之间的单精度浮点数值，而虚拟手单根手指的活动角度范围并不等于180度，且为了符合拟人化处理的要求，每根手指活动的角度范围各不相同。因此我们首先要对接收数据做归一化处理，然后与实际手指的角度等比例相乘。

以食指为例，当我们接收到食指的角度数据值之后，首先将其除以180，获得范围在0~1之间的比例因子p：

然后我们针对食指的三个关节，获取三个关节的初始角度以及限定的最终弯曲角度，通过四元数的插值函数Quaternion.Slerp()，使用比例因子对同一手指的三个关节在初始角度和最终角度之间进行插值处理，实现传感器接收角度和关节四元数值的一一对应。在实际使用时，数据手套需要首先进行角度校准，以五指伸张作为0度，以五指弯曲作为180度，如图3-6所示。然后再进行数据发送。这样每当接收到一个角度信息，虚拟手经归一化插值处理后都能实现对应手指弯曲相应的角度。



**图3-5 食指远节指间关节的角度信息**

数据手套传感器的采样时间是10ms，每分钟可以传输包含五指信息的100组数据。针对采样时间，我们分别设置虚拟手的采样时间为10ms、100ms、200ms进行测试，比较控制效果。实测中发现，当采样时间为10ms，即与数据手套传输数据速度相同时，由于数据手套传感器的数据波动和漂移现象，如图所示 此处放图，虚拟手的抖动非常严重，效果较差。当采样时间为200ms时，虚拟手相对数据手套会呈现一定的时间滞后性。当采样时间为100ms，则有相对较好的实时性效果，因此我们最终选择的采样时间为100ms。

## 3.6 本章小结

本章使用UDP协议传输，搭建了服务器端并建立了模拟用户端进行数据测试，然后建立了服务器端虚拟平台和数据手套用户端之间的数据通讯传输，通过对角度信息进行归一化处理，初步实现了通过数据手套控制虚拟手的效果。通过对不同采样时间进行测试，最终选择采样采样时间为100ms。通过数据手套控制虚拟手已经具有较好的实时性，下一章将考虑针对手功能障碍患者设计简单的康复训练小游戏，并完成游戏机制及图形界面的设计。

# 第四章 游戏设计

## 4.1 游戏目的

目前传统的康复训练治疗往往需要患者在医护人员的引导下做大量重复性康复训练，而通过设计不同的康复训练小游戏，能够让患者在容易感到枯燥的康复训练过程中获得趣味，并能够在训练过程中集中注意力，激发患者完成康复训练的东西，提升康复训练的效果。

基于数据手套提供的角度数据，我们初步构想设计一个抓取类的游戏，用户通过握紧手指抓取游戏中的目标，时间越短获得的分数越高，以此来锻炼患者手部五指的协调性和弯曲速度，帮助患者康复。此外，我们还在游戏中提供了难度分级，患者能够通过做出不同更有难度的手势，来获得更高的得分，提供给患者更多的选择。

## 4.2 抓取小游戏设计

### 4.2.1 Unity的刚体与碰撞器

在抓取条件的判定上，我们决定采用Unity提供的刚体和触发器组件来进行抓取判定的设置，首先简单介绍Unity中提供的刚体和碰撞器功能。

Unity中的Rigidbody刚体组件可以为游戏对象赋予物理属性，是游戏对象在屋里系统的控制下接受推力与扭力，从而能够像在现实世界中一样进行运动。当一个游戏对象被赋予刚体组件后，游戏引擎会对其进行物理效果模拟，实现和其他游戏物体的物理交互。

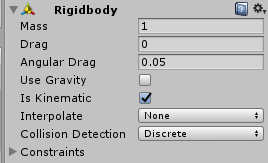


图 物体的刚体属性

触发器（Trigger）用来触发事件。在很多游戏引擎或者工具（例如RPG Maker）中都有触发器，例如在角色扮演游戏里，玩家走到一定的区域范围内就会触发事件，这种场景就可以使用触发器来实现功能。当绑定着碰撞器的游戏对象进入触发器的区域时，会运行触发器对象上的MonoBehaviour中的OnTriggerEnter()函数，通过函数内的代码来控制游戏事件的发生。

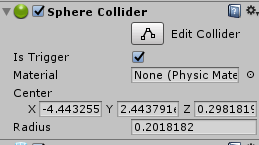


图 球形碰撞器

### 4.2.2 游戏内容设计

经过讨论，我们决定设计抓取小游戏的形式是通过抓取屏幕上飞舞的蝴蝶，并通过计算时间来衡量受试者的成绩。首先我们导入在网络上找到的蝴蝶素材，并在虚拟平台中测试效果。值得注意的是，此时应通过Unity游戏素材的Scale Factor属性来调整游戏对象的大小，而不能通过传统的Scale属性来调整大小，否则会出现游戏对象预览窗口和实际动画播放窗口大小异常的问题。

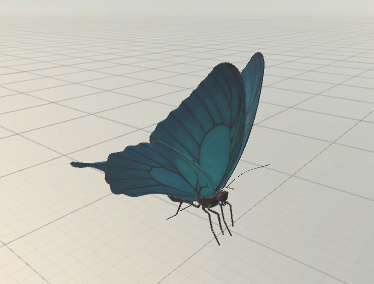


图 蝴蝶素材

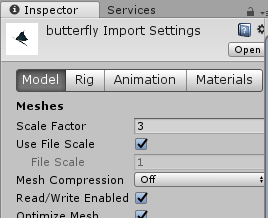


图 通过蝴蝶对象的Scale Factor属性来修改大小

然后我们通过Unity中提供的动画处理器Animation来给蝴蝶添加动画效果，在Animation界面，我们可以设置动画的播放时长，并可以通过设置关键帧（Key frame）的方式，对动画进行某一帧的细节设置，可以编辑的细节包括游戏对象的位置、角度、大小等信息。通过使用关键帧，我们对蝴蝶每一时刻的位置、翅膀角度信息进行修改，最终得到蝴蝶飞舞的动画效果。

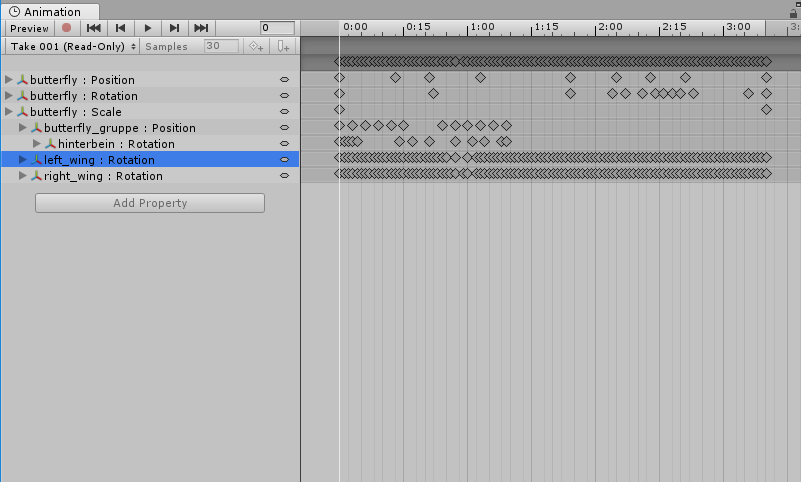


图 对蝴蝶动画的关键帧处理

在游戏功能的设计上，我们决定使用触发器来检测手指是否运动到指定位置。我们首先创建一个球形的游戏对象，并向其添加刚体附件，这样我们才能够继续向其添加触发器附件。将球体拖动到手掌掌心的合适位置，并向其添加触发器控件，通过调整合适的半径大小，使得五指能够在闭合时恰好接触到碰撞区域，就可以触发碰撞器内的OnTriggerEnter()函数，控制接下来的事件运行。

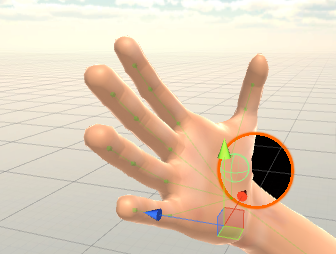


图 球体和触发器的设置

接下来将之前编辑好动画的蝴蝶对象设置为上述球体的子物体，而将球体的父物体设置为我们的虚拟手，这样蝴蝶的坐标就会使用相对球体位置的相对参考系而不使用相对世界原点的世界参考系，从而实现围绕球体飞舞的动画效果。然后我们设置球体的材质为透明材质，这样在实际的游戏视角中，就能呈现出蝴蝶围绕手部飞舞的效果。



图 蝴蝶飞舞效果

接下来我们编写蝴蝶消失的代码，我们首先定义一个初始值为假的逻辑参数if\_thistrial\_end，用来表示这一次的抓取动作是否完成，如果完成，则设置为真，否则设置为假。出于在游戏的准确性和效果考虑，我们设置当至少有三根手指进入到预先设置的触发器区域时，即判定为完成了一次抓取，此时在触发器的OnTriggerEnter()函数中，通过游戏对象的setActive()函数，将蝴蝶对象的Active值设置为false，即取消蝴蝶对象的活动状态，使其消失，从而实现“抓取成功”的效果，如图所示。



图 抓取成功，蝴蝶对象消失

### 4.2.3蝴蝶刷新函数

解决了蝴蝶对象的抓取问题，接下来我们设计关于蝴蝶刷新的函数。首先我们编写函数BallSpawn()，在函数中设置条件判断，当承载触发器的球体的激活状态为false时（即虚拟手已经完成了一次成功抓取），将球体激活状态重新设置为true（此时蝴蝶重新刷新），并将定义的每次抓取时间归零，抓取成功次数加一。注意此时函数仍然不能放在Update()中运行，因为Update()是每帧刷新，如果将函数放在此处运行，将会导致蝴蝶每帧刷新，无法实现抓取后消失的效果。我们在Unity初始化函数中使用Unity的内置成员函数InvokeRepeating()来实现重复效果，InvokeRepeating()函数可以使特定的函数延时执行，并间隔一定的时间重复执行。我们以5秒作为重复执行的时间间隔执行我们的刷新函数BallSpawn()，即每隔5秒检测球体的激活状态，若为false则刷新蝴蝶，否则继续等待五秒后执行下一次检测，如图所示

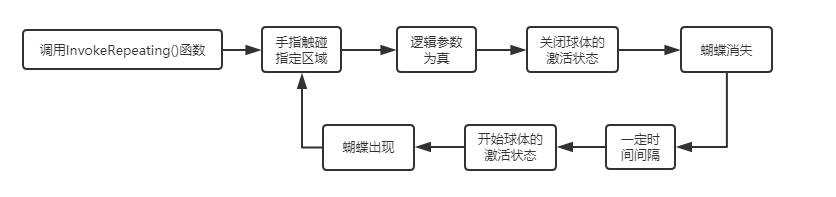


图 蝴蝶的刷新机制

### 4.2.4 游戏时间计算与存储

接下来我们计算用来衡量抓取成功速度的时间指标。值得注意的是，我们并不能直接调用Unity中的时间类Time.time来衡量游戏运行时间，因为Time.time计算的是游戏开始到现在的时间，而在游戏主要运行的Update()函数中，Update()是随着游戏的运行而每帧刷新，这样游戏中运行的时间实际上就会受到帧率的影响，而不能反映实际运行的时间。因此我们使用增量时间Time.deltatime来计算实际运行的时间，增量时间是以秒为单位的时间，计算的是完成上一帧所用的时间，因此当帧数与帧间时间相乘时，我们就可以获得实际对应的时间。

我们还可以通过归零的方式来计算上一次抓取所用时间。因为在未抓取状态下，逻辑参数if\_thistrial\_end为假，我们可以增加条件语句，新建变量UseTimePerTry，当if\_thistrial\_end为真时，存储经过的时间并停止记录，这样我们就得到了上一次抓取所用的时间。

此外，我们还设置了数据存储TimePacket，由于我们预先不知道用户会进行多少次抓取动作，因此我们采用动态数组ArrayList来对上一次所用的时间进行存储。对于传统的数组类型，在进行变量存储之前，需要声明数组的容量。而ArrayList则是一种动态的存储方式，可以在并不提前声明容量的条件下动态地增加或减少元素，而且可以存储任意类型的元素和任意个数。因此，我们可以使用一个ArrayList型的变量TimePacket来存储每一次成功抓取所花费时间，用于之后康复训练效果的统计和分析。

通过动态数组ArrayList类型提供的成员函数Sort()，我们可以对TimePacket中的元素进行从小到大的排序。这样当我们取第一个元素时，就可以得到历次抓取过程中用时最少的一次。

### 4.2.5 游戏分数的计算

本小节我们设计抓取蝴蝶游戏的分数计算。对于一次成功抓取的分数计算，我们以抓取所用的时间来作为分数的衡量标准。经过我们小组的讨论分析，我们决定将时间区分值定为1秒和5秒：如果用户能够在1秒内成功地完成抓取动作，则能够获得100分的加分；如果用户能够在5秒之内完成抓取动作，则能够获得50分的加分；在5秒之外完成的抓取动作，则只能够获得10分的加分。时间用上述的每次抓取时间衡量，分数则在之前的分数上累加。通过游戏分数的设置，可以鼓励用户进行康复训练，并努力实现更快的抓取动作。

### 4.2.6 特殊手势训练的设置

考虑到如果只使用单一的抓取动作来进行训练，即使是采用了游戏的形式，用户也会很快感到疲倦。因此我们在游戏中还加入了难度分级，即通过做出不同难度的手势动作，获得额外的得分，实现难度分级的效果。我们首先设计了三个难度不同的手势，即弯曲拇指和食指、弯曲拇指和小指以及弯曲食指和小指，这几个难度递增的动作得分分别为200分、400分和1000分，手势如图 所示



图 难度递增的三个手势

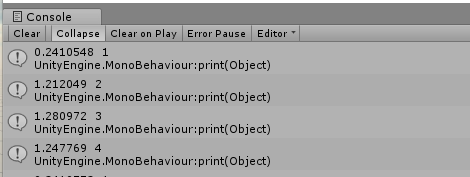
考虑到用手指关节之间的所成角度作为判断手势依据时，可能出现同一角度下的多种情况，而对于固定的虚拟手，不同手指远端指间关节指间的距离则相对固定，对于这种相对简单的场景来说显得更为准确。我们首先建立变量来存储各个手指上的远端指间关节所对应的角度值，通过Unity中游戏对象自带的成员变量transform.position即可获得当前游戏物体在世界坐标系下的绝对坐标，并保存为Vector3类型的变量，然后使用变量中包含的成员函数distance()来计算拇指与其余四个手指上的远端指间关节之间的距离。计算得到两根手指之间的最大距离和最小距离，然后分别计算各个手势下拇指与其余四指之间的距离。通过if条件语句的控制，综合考虑设置0.1为指间距离的允许误差，当拇指与其余各指的指间距离满足条件时，即判定已经做出了相应的手势，进而对分数进行累计。

以第一个“OK”手势为例，首先我们计算得到拇指与其余四指远端指间关节的最大距离和最小距离，如图所示



图

然后计算在“OK”手势下，拇指与其余四指远端指间关节的距离，如图 所示



图

得到了该手势下的距离值，我们就可以通过条件语句来对距离进行判断，针对此手势，我们判定当拇指与食指距离小于0.3，拇指与中指、无名指、小指的距离大于1.15时，即认为完成了一次对应的弯曲，然后计算对应的分数。

同样地，我们不能够把判断手势的函数放在Unity的Update()函数中，因为这同样会引发在每帧中判断进而导致在该手势下每帧判断都成功而并非判断一次的问题，所以我们仍然需要利用上述延时函数InvokeRepeating()，而将函数的重复检测时间相对缩短即可。这样在一次成功地手势判定后，会等待一定的时间再进行下一次判断。

## 4.3 GUI实现

游戏的基本内容和控制代码已经设计完毕，接下来我们还需要设计一个图形用户界面（GUI，Graphical User Interface）用于和用户展开交互。Unity提供了使用脚本创建GUI的能力，提供了包括按钮、文本框、文字信息等一系列控件供开发者使用。

首先我们需要设计游戏的开始界面，像大多数游戏一样，我们需要“开始”、“帮助”和“退出”三个按钮。首先我们寻找到合适的边框素材贴图，以作为纹理贴图在矩形选框上，用来作为我们的菜单面板，如图所示

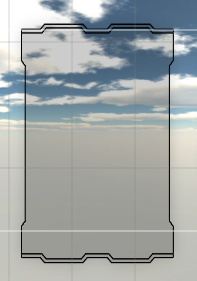


图 菜单面板

接下来我们在菜单面板上的合适位置，设置三个按钮控件，并在按钮控件上添加文字控件，文本内容分别为“Start”、“Help”和“Quit”，调整合适的颜色和字体大小，设置完成后如图所示

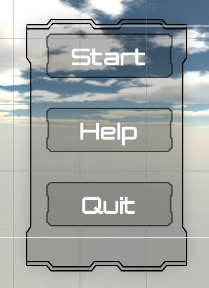


图 设置完成后的控制面板

开始菜单框架搭建完毕后，我们接下来设置各个按钮的点击触发函数。首先开始按钮的功能即为进入游戏，我们利用Unity提供SceneManagement库中的功能，设置当点击开始按键时，切换到我们的游戏主场景并开始运行，这样就实现了开始游戏的效果。

对于帮助菜单的设计，我们设置当我们点击“帮助”按钮时，能够弹出相应的游戏规则说明，指示用户如何才能完成游戏，并附加图片说明。通过添加文字组件来输入我们想要显示的文本，通过添加图片组件来添加我们的演示图片，并设置返回按钮，当点击返回按钮时触发返回主菜单的相应，如图所示



图 帮助菜单的显示页面

对于“退出”按钮，我们在其上绑定的脚本文件上设置，当点击按钮时，触发Unity自带库中的Quit()函数，从而退出函数。

在游戏界面，我们同样需要设置图形界面来显示用户当前的分数和时间。我们同样先建立一块风格统一的面板作为背景。此处我们采用脚本编写的方式，使用Unity进程中的OnGUI()函数，在屏幕右上角显示我们定义的分数、总用时、上次抓取用时和最短抓取用时。使用GUIStyle类还可以对显示文本的大小、颜色等格式进行调整，最终效果如图 所示



图 显示分数和时间信息

此外，针对几种难度不同的特殊手势，我们还可以设计特殊的动态效果使其更加醒目。以“OK”手势为例，我们首先在画布（Canvas）上新建一个文本对象，内容为+200，并调整合适的颜色和字体大小。然后我们打开Unity内自带的动画编辑器Animation，通过定义关键帧，在不同时刻改变字体的位置和透明度，形成一段淡入淡出的动画效果，更加醒目显眼，如图所示



图 “OK”手势的动画效果

## 4.4 本章小结

本章结合现有的硬件条件以及患者手部康复锻炼的需要，自主设计并制作了能够锻炼五指的抓取游戏，并在游戏中加入了难度递增的手势，通过难度分级来激发患者康复训练的动力。