计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(5):76−81 [doi: 10.15888/j.cnki.csa.007399]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于可视化的 VR 编辑引擎①

蒋　宁, 徐济惠

(宁波城市职业技术学院 信息与智能工程学院, 宁波 315110)
通讯作者: 蒋　宁, E-mail: 48971892@qq.com

摘　要: 针对传统的虚拟现实 (VR) 应用开发平台所需技术门槛较高, 不利于 VR 的普及等问题, 采用基于状态机的可拖拽框架设计, 可扩展低耦合的 UI 界面设计以及组件形式的事件系统设计, 优化了 VR 中常用的碰撞盒以及模型细节层次的技术方案, 解决了具体实现中的数据结构、多线程实现、设计模式和数据存储等问题. 最终设计的 VR 编辑引擎采用完全可视化的“拖拽”设计方式, 发布后的程序既可以在 PC 上运行, 也可以在 VR 设备上运行. 该引擎主要应用于教育培训领域, 是具有易操作性、趣味性和先进性的实验教学辅助工具, 可激发学习兴趣、提高学习效果. 通过鼠标拖拽, 在即看即所得场景中, 方便的完成各种专业级的虚拟现实应用的搭建, 极大的降低了 VR 仿真软件的开发门槛和开发成本.

关键词: 可视化; 状态机; 碰撞盒优化; 细节层次

引用格式: 蒋宁,徐济惠.基于可视化的 VR 编辑引擎.计算机系统应用,2020,29(5):76–81. http://www.c-s-a.org.cn/1003-3254/7399.html

VR Editing Engine Based on Visualization

JIANG Ning, XU Ji-Hui

(School of Information and Intelligence Engineering, Ningbo City College of Vocational Technology, Ningbo 315110, China)

Abstract: Aiming at the high technical threshold of traditional Virtual Reality (VR) application development platform, which is not conducive to the popularization of VR, the draggable frame design based on state machine, extensible and low-coupling UI interface design and event system design in component form are adopted to optimize the common collision boxes and model details in VR. Hierarchical technical solutions, Solve the specific implementation of the data structure, multi-threaded implementation, design patterns and data storage and other issues. The final VR editing engine uses a fully visualized “drag-and-drop” design, and the released program can run on either PC or VR device. The engine is mainly used in the field of education and training. It is an experimental teaching assistant tool with easy operation, interesting and advanced. It can stimulate learning interest and improve learning effect. By dragging and dropping the mouse, the VR simulation software can be easily constructed in the “what you see is what you get” scene, which greatly reduces the development threshold and cost of VR simulation software.

Key words: visualization; state machine; collision boxes optimization; level of detail

通常虚拟现实 (VR) 应用的设计开发是需要开发人员具备一定的软件编程能力, 但是同时也限制了更多的没有软件编程技能但是却对设计虚拟现实应用感兴趣的这部分群体的创造能力[1], 因此如何实现一个能够使开发者无需掌握任何一门编程语言, 仅仅通过鼠标拖拽等“傻瓜”式操作, 在即看即所得场景中, 方便完成各种专业级虚拟现实应用搭建的编辑器已经成为当前虚拟现实市场的研究热点之一, 目前虚拟现实开发

① 收稿时间: 2019-10-11; 修改时间: 2019-11-07; 采用时间: 2019-11-18; csa 在线出版时间: 2020-05-07

工具市场份额最大的是 Unity3D 软件工具, 但是基于 Unity3D 工具开发的应用也是需要写大量代码的, 国外最为著名的可以通过鼠标拖拽就能完成一款虚拟现实应用的编辑器是 PlayMaker 编辑器[2], 但是它有几个不足地方: (1) 完全基于英文, 没有汉化版, 不方便国人使用. (2) 它本质上属于插件, 不能单独使用, 必须嵌入到 Unity3D 里才能使用. 国内目前也有几家公司致力于可视化的编辑器设计, 这几款编辑器中, 要么就是当用户使用时, 还是需要编写脚本代码的, 要么就是实现的设计应用还是太简单了, 难以达到商业应用[3]. 本文研究的可视化编辑引擎是源自本文作者开发的一款基于可视化的编辑引擎, 这款工具在一定程度上弥补了上述的不足.

1 可视化编辑器引擎设计

1.1 系统框架设计

本文研究的可视化编辑引擎的框架设计是基于有限状态机的思想进行设计开发. 比如开、关、开启、关闭、行走、空闲、攻击、防御等诸多状态, 每一个状态 (state) 由一个或者多个行为组成, 每一个行为 (action) 又包含了多个属性 (property), 然后通过事件 (event) 驱动不同状态之间的转换[4]. 因此驱动一个通过编辑器引擎搭建的任务流是由状态机、状态、行为、事件、转移、属性以及条件等要素构成. 任务流的数学模型可以表示成一个六元组 $(P, S, A, S0, \delta, \omega)$, 其中 P 是前驱状态集合, S 是后继状态集合, A 是行为的非空有限集合, $S0$ 是初始状态, δ 是状态转移函数, 可以表示为 δ: $P\times S \rightarrow P$, ω 是输出函数[5]. 在这个引擎里, 所有的状态机、状态和事件都是以可视化的方式通过拖拽等操作组合起来共同完成一个工作任务. 这里状态机 (state machine) 是 3 种状态的集合体, 这 3 种状态分别是前驱状态、当前状态以及全局状态; 而状态的数据结构则包含了 4 种列表: 分别是只执行一次的动作链表、在帧间不断重复的动作链表、事件链表以及动作界面链表; 事件包含 4 种类型: 自定义事件、系统事件、网络事件以及 VR 事件[6]. 上述几种元素面向使用者则是封装成可视化的矩阵节点 (node) 以方便用户通过拖拽的方式进行操作, 但是状态之间的转移则是通过一条有向线段标示, 每个状态可以包含多个动作, 如图 1 所示.

图 1 状态可视化示意图

图 1 表示的是当鼠标左键点击三维模型时, 进入到下一状态, 下一状态包含特写镜头、显示按钮、物理设置等行为. 等下一状态的这些行为全部执行完毕后, 触发结束事件, 通过转移有向线段又返回到初始状态. 其体现为一个状态变迁矩阵如表 1所示.

表 1 状态变迁矩阵

当前状态	事件	输出状态
开始状态	鼠标左键单击	新状态
新状态	结束	开始状态

1.2 UI 界面设计

UI 是 User Interface 的缩写, 就是用户界面的意思. UI 在一个软件系统中是人机交互的桥梁, UI 设计的优劣直接决定用户对于软件使用的体验感, 好的 UI 设计能够让用户对这款软件爱不释手, 反之如果 UI 界面设计不到位, 则即使你这款软件功能很强大, 也有可能不被用户所接受. 因此需要设计一套良好的 UI 框架, 在这款引擎中采用的是 MVC 模式来进行 UI 设计, 其优点包括 3 个方面: (1) UI 界面的属性只需要在 Inspector 面板中设置即可; (2) 业务逻辑与 UI 可以实现完全解耦, 提高了需求功能的可扩充性; (3) 可以很方便的从 NGUI 模式切换到 UGUI 模式.

本引擎中 UI 基于 MVC 的示意图如图 2 所示.

Model 的特点包括: (1) 与 UI 视图数据完全解耦; (2) 通过控制器或者其他 Model 以引用的方式访问; (3) 可以通过委托 (Delegate) 方式来触发外部事件.

本引擎中的 Model 是继承 Unity3D 自带的 Mono-behavior 类, 完全基于 C#编写的.

View 的特点包括: (1) 展示所有的 UI 元素; (2) 可以直接与用户进行交互; (3) 布局界面的设计; (4) 对 Models 层只拥有只读权限[7].

Controller 的职责包括: Controller 负责定义和调用 Model, 它可以决定要显示哪个 View. 控制器接受用户的输入并调用模型和视图去完成用户的需求. 控制器本身不输出任何东西和做任何处理. 它只接受请求

并决定调用哪个模型构件去处理请求, 然后决定用哪个视图来显示模型处理返回的数据.

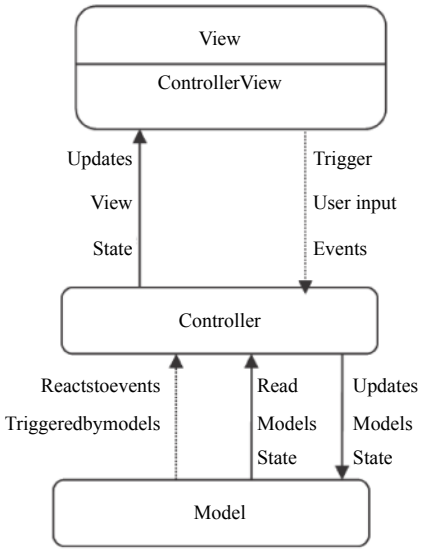

图 2 MVC UI 结构图

引擎中, 我们加入了另外一个中间层来进一步解耦 GUI 的 View 实现, 称之为: ControllerView, 一个 Controller View 位于 View 和 Controller 之间, 它提供了若干接口, 用来声明与 UI 具体实现无关的操作, 无论 View 本身是基于什么库来实现的 (NGUI, UGUI 等), ControllerView 的实现是基于适配器模式 (adapter pattern) 的, 这种类型的设计模式属于结构型模式, 它结合了两个独立接口的功能. 此时是作为业务逻辑代码和 UI 代码的桥梁, 也是作为两个不兼容的接口之间的桥梁.

1.3 事件系统设计

在本引擎中各个状态的转换和迁移都是通过事件驱动, 框架中定义了大量的事件, 比如模型碰撞、鼠标事件、键盘事件、网络事件等等, 需要一种机制来合理组织、管理和执行这些事件, 这种机制能够管理和执行一帧里面的多个事件或者是帧里一个时间片上事件的多次触发. 可以通过事件系统组件的方式来灵活的管理事件, 同时也能提升应用的模块化, 提供更好的可扩展性.

事件系统组件包括标准输入模块、触摸输入模块、物理射线投射模块以及图像投射模块.

事件系统组件主要负责处理输入、射线投射以及发送事件, 它是一种将基于输入的事件发送到应用程序中的对象, 无论是键盘、鼠标、触摸或自定义输入. 一个场景中只能有一个事件系统组件, 但是可以拥有多个输入类型组件, 事件系统在初始化的时候会通过字典的形式将所属对象下的输入模块类型依次添加进去, 并且在每个 Update 周期通过 UpdateModules 接口调用这些基本输入模块的更新模块接口, 然后输入模块会在 UpdateModule 接口中将自己的状态修改成“已更新”状态, 之后才可以调用输入模块的其他接口.

标准输入模块和触摸输入模块都源自于一个基类: 基础输入模块, 它是是一个基类模块, 负责发送输入事件 (点击、拖拽、选中等) 到具体对象. 事件系统下的所有输入模块都必须继承自基础输入模块组件.

除了以上两个组件, 还有一个很重要的组件就是基础射线投射组件, 它也是一个基类, 前面说的输入模块要检测到鼠标事件必须有射线投射组件才能确定目标对象. 系统实现的射线投射类组件有物理射线投射模块以及图像投射模块.

总的来说, 事件系统组件负责管理, 基础输入模块负责输入, 基础射线投射组件负责确定目标对象, 目标对象负责接收事件并处理, 然后一个完整的事件系统就形成了.

2 关键技术研究

2.1 包围碰撞盒优化

由于本引擎的一个重要特征是可视化, 因此用户在操作本引擎时会涉及到与大量可见模型进行交互操作, 这种与三维模型的交互需要附着在三维模型外面的碰撞检测盒的支持, 传统的碰撞盒采用的方式主要有: AABB 方式、包围球层次树、OBB 层次树等方式, 其中 AABB 方式特点是构造难度较低, 但是包围紧密度低; 包围球层次树的特点是构造难度比 AABB 方式低, 但包围紧密度不如 AABB 方式; OBB 方式的特点是构造难度要大于前两种方式, 但是包围紧密度要高于前两者. 这几种方式的误差度总体都较大, 不能给出最优范围的界限且实时检测性能不好. 在本引擎中, 采用了基于协方差矩阵方式优化碰撞检测盒以便提高准确度. 在场景中的模型可以被认为是由 n 个点$\{D1, \cdots, Dn\}$形成的一片点云集合, 这里将每个点的 x、y 和 z 值的集合通过偏差和协方差来进行处理. 偏差就是度量数据的点集和平均值的差异, 它计算的是平均值差的平方和. 其算式表示针对 X 轴如下, 针对 Y 或者

Z 轴也是类似的算式.

$$Var_x \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i-\bar{x})^2 \tag{1}$$

注意我们在估算中, 是用 n–1 代替 n 来纠正偏差的, 这样整体的平均值就是基于 n 个点集合的中心, 即 $\bar{p}=\frac{1}{n}\sum_{i=1}^{n}P_i$, 则在 X, Y, Z 3 个方向上的平均值则分别是:

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i;\ \bar{y}=\frac{1}{n}\sum_{i=1}^{n}y_i;\ \bar{z}=\frac{1}{n}\sum_{i=1}^{n}z_i \tag{2}$$

我们希望偏差尽量变小, 这样数据尽量是整体接近平均值的, 在偏差的基础上, 可以进一步采用协方差的度量方式以便更加精确的提高检测盒的精度, 协方差度量的是数据集合的独立性. 在三维场景中用的最多的就是 X 和 Z 坐标集合之间的协方差, 其算式如下表达:

$$cov_{xz}=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})(z_i-\bar{z}) \tag{3}$$

x 和 y 以及 y 和 z 之间的协方差的算式同上面是类似的, 协方差的值越低说明数据集之间数据的独立性越低, 通过整合每个坐标对的偏差和协方差, 可以确定一个协方差矩阵 M, 如式 (4).

$$M=\begin{pmatrix} Var_x & cov_{xy} & cov_{xz} \\ cov_{xy} & Var_y & cov_{yz} \\ cov_{xz} & cov_{yz} & Var_z \end{pmatrix} \tag{4}$$

这种协方差矩阵可以在其他特殊方向上确定对偏差的度量, 矩阵的结果向量如果偏向较小偏差的区域, 则该方向就会偏短些, 反之如果偏向较大偏差的区域, 则该区域方向就会偏长些. 对于一个单位圆, 如果用协方差矩阵进行转换, 则会得到一个椭球体, 这个椭球的长轴会沿着偏差值高的方向展开, 基本上对齐椭球体的长轴. 反之, 椭球的短轴则会沿着较低偏差的方向展开, 也就是点密集分布的地方, 基本上对齐于椭球体的短轴. 因此我们的策略就是用椭球的轴来代替标准轴创建场景中的包围对象, 这些轴就是已知的主轴, 如图 3.

为了获得这些主轴, 需先对角化这个矩阵, 即先对转换到主轴空间的向量进行旋转, 并应用到协方差矩阵, 最后旋转回来, 算式表达为: $D=P^{\mathrm{T}}CP$, 这里 P 就是正交矩阵, 所有列向量构成坐标系统的正交基. 它们等同于转换前的协方差矩阵的主轴, 因此可以用它们来创建碰撞包围盒.

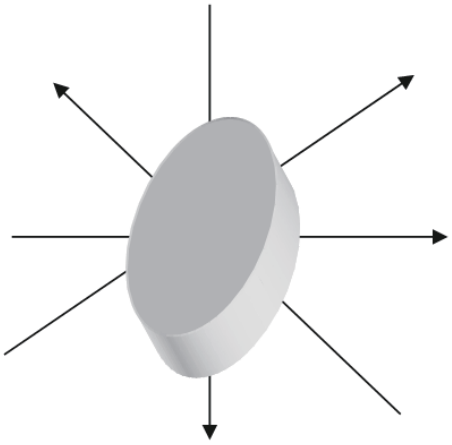

图 3 基于协方差矩阵的带主轴椭球体

为了得到这个旋转矩阵, 需要先获得其特征向量和特征值, 可以通过计算特征多项式, 对于三维场景坐标而言, 就是一个三次多项式, 然后直接解出方程的根. 矩阵 D-λI 的行列式为:

$$\begin{pmatrix} d_{11}-\lambda & d_{12} & d_{13} \\ d_{21} & d_{22}-\lambda & d_{23} \\ d_{31} & d_{32} & d_{33}-\lambda \end{pmatrix}=A\lambda^3+B\lambda^2+C\lambda+D \tag{5}$$

$$\text{s.t. } A=-1; B=d_{11}+d_{22}+d_{33} \tag{6}$$

$$C=-d_{11}d_{22}+d_{12}^2-d_{11}d_{33}-d_{13}^2-d_{22}d_{33}+d_{23}^2 \tag{7}$$

$$D=-d_{11}d_{22}d_{33}+2d_{12}d_{13}d_{23}-d_{11}d_{23}^2-d_{22}d_{13}^2-d_{33}d_{12}^2 \tag{8}$$

可以解出这个三项式的根来得到特征值, 然后代入每个特征值来计算出矩阵 $D_i=C-\lambda_i I$. 解出这个线性系统就可以获得对应的 3 个特征向量. 这些特征向量和中心构成了点云的坐标基就可以计算三维对象的碰撞检测盒了.

碰撞检测盒算法设计思路: 首先将构成碰撞检测盒的每个点减去中心, 得到差值向量, 接着用每个归一化的特征向量来做点积计算. 相对于每个基向量可以给出差值向量投影的长度. 计算出这些点击的最大最小值, 就会为点云创建对齐于新的基向量的包围盒, 经测试通过基于协方差矩阵方式优化包围盒要比上述 3 种方法在精确度和实时检测速度上都有明显的提高.

2.2 模型细节层次的优化

在引擎中, 专门有一个资源列表面板, 资源列表面板的作用是加载三维模型到场景中, 目前资源面板里已经包含了人物、动画、植物、动物、交通、器材、建筑等类型的将近 300 种三维模型, 如图 4 所示.

图 4 资源面板

每一种模型又分为高精度、中精度和低精度等 3 个细节层次的模型, 之所以采用这种方式, 是因为当场景模型与摄像机的位置相聚很远时, 可以不需要显示高精度的场景模型, 从而加速渲染效果. 反之当一个场景模型与摄像机的距离较近时, 则可以显示高精度的渲染效果以便改善视觉品质. 为了实现这种动态渲染效果, 需要使用 LOD 技术, 即细节层次技术, 为了实现细节层次渲染效果, 我们针对每个不同的细节层次分别创建三个, 每帧要渲染的模型应基于与摄像机间的距离进行选择的. 比较简单的规则是每个 LOD 应该有大约 2 倍于前一层次的多边形的数目, 但是这种规则比较粗略, 很容易产生抖动, 当最小化 LOD 变换的数目时, 需要解决何时改变 LOD 以便获得理想的性能和品质. 选择要渲染的 LOD 最简单方法是确定一个阈值, 这个阈值对应摄像机到模型的距离, 这种方法虽然容易计算, 但是存在两个问题: (1) 这种方法没有考虑摄像机的视野. 如果摄像机的视野过于狭窄或者过于广阔, 则可能会导致 LOD 渲染的频繁抖动现象的发生. (2) 如果对象与阈值很接近时, LOD 之间会有快速的反复转换. 为了解决上述两个问题, 我们使用“增大因子”技术, 增大因子是物体的屏幕尺寸与其物理尺寸的比例, 屏幕尺寸的确定是通过转换和投影对象上的极大值和极小值, 然后减去每一坐标的屏幕位置, 增大因子通过将对象的位置变换到视域中然后进行计算而得到:

$$S = x_{\text{scale}}/z_{\text{view}} \tag{9}$$

这里 x_{scale} 是用于投影方程的换算参数:

$$x_{\text{screen}} = (x_{\text{view}} \times x_{\text{scale}})/z_{\text{view}} + x_{\text{center}} \tag{10}$$

由于视点坐标只是一个世界坐标的旋转或平移, zview 由世界单位度量. x_{scale} 相对于摄像机视野, 以像素为单位; 因此, 增大因子 M 可以度量每个世界单位的像素. 当 M 增加时, 每世界单位有更多的像素, 对象相对来说在屏幕上更大, 因此应该使用一个更高的细节层次. 这样 M 既考虑了摄像机的距离也考虑了视野. 而且给出了一个比简单摄像机距离更好的确定细节层次的选择依据. 但是如果仅仅对 M 应用一个简单的阈值会产生频繁的抖动现象. 那么如何解决这种频繁的抖动现象? 在此我们采用滞变阈值来解决这种频繁抖动现象, 滞变阈值对应于一个值的范围的阈值, 其使用一个较高的和一个较低的阈值并且记录前一次的输出值. 如果输入值是在较高的和较低的阈值之间, 则输出值不变. 通过滞变阈值和放大率因子, 我们可以创建一个细节层次选择算法, 针对 3 个细节层次的低、中、高模型, 用于在高细节层次和中细节层次之间移动的滞变阈值为 Th_{upper} 和 Th_{lower}. 在中细节层次和低细节层次之间移动的滞变阈值为 Tm_{upper} 和 Tm_{lower}, 通过不断调试, 可以设定一个介于较高和较低滞变阈值之间的输入值, 采用这种方式选择细节层次意味着物体不会在一个点上进行反复的细节层次变换, 这样就确保我们所得到的是一个单一的移动而不是多个反复的变换行为, 从而避免了快速抖动的问题.

综上所述, 我们从可视化角度分别在包围盒碰撞检测和模型细节层次上进行了算法优化, 比常规的算法提高了精确度和实时性.

3 编辑器系统的实现

3.1 数据结构实现

本引擎完全采用 C#开发, 所写的 C#语言脚本接口是以 MonoBehaviour 这个类作为基础的, 主要使用的数据结构包括数组部分: List<T>; 链表部分: LinkedList<T>、哈希表和字典: HashTable 和 Dictionary<K,T>, 在实现代码里, 跟业务逻辑相关的所有的代码都是继承于 Action 类, 而跟 UI 相关的所有类都是继承于 ActionUI 类, 同时大量使用了泛型机制和插件机制, 泛型的引入可以使得大量的安全检查从运

行时转移到编译时进行, 提高了代码的运行速度. 本引擎中我们对于底层逻辑的开发都采用的泛型, 不需要指定特定的数据类型, 通过泛型的引入, 使得“逻辑复用”成为一种可能; 而插件的使用, 则提高了软件的内聚性, 降低了软件的耦合性.

3.2 设计模式实现

在引擎中, 我们大量采用的设计模式是观察者模式, 即定义了对象之间的一对多依赖, 即当一个对象改变状态时, 它的所有依赖者都会收到通知并且自动更新. 具体实现是通过委托 (delegate) 和事件 (events) 这两种回调函数机制完成的, 委托其实类似于 C++中的函数指针, 但它是以类的形式定义的, 它定义了方法的类型, 使得方法可以类似指针处理方式作为参数来进行传递. 事件是一种特殊的委托.

3.3 多线程实现

多线程能够在同一时间执行多于一个线程, 进而提升引擎整体处理性能, 在本引擎中, 通过某种机制也是可以实现类似多线程功能的效果的, 这种机制就是协程 (coroutine), 协程可以暂停逻辑的执行, 并且将控制权移交给引擎, 但是之后它还可以从暂停的位置继续开始执行余下的逻辑. 同时在引擎中的很多延时效果也是使用协程来实现的, 具体实现概括如下: 调用 StartCoroutine 方法开启协程, 协程的返回值必须是 IEnumerator 类型, 调用 WaitForSeconds 类实现延时效果.

3.4 数据存储的实现

我们采用序列化和反序列化技术实现数据的存储, 所谓序列化就是将对象转换为字节流的过程, 反序列化则是指的将字节流转换回对象的过程. 具体实现概括如下: 首先构造一个 System.IO.MemoryStream 对象, 提供了一个用来容纳经过序列化之后的字节块的容器. 接着创建一个格式化 (formatters) 对象, 最后调用序列化 (serialize) 方法和非序列化方法 (deserialize) 方法分别实现存储和加载数据的功能.

4 结束语

本文基于可视化的 VR 编辑引擎设计和研究, 围绕可视化、可扩充、可优化的设计目标, 以系统框架、UI 界面、事件系统等方面的设计工作为基础, 研究了包围碰撞盒技术和模型细节层次两个关键技术并进行了优化, 实验证明优化后的算法比常规的算法提高了精确度和实时性. 并解决了具体实现中的数据结构、多线程实现、设计模式和数据存储等问题, 最终实现了 VR 编辑引擎, 结果表明, VR 编辑引擎系统各项功能指标均符合实际要求, 具有完全可视化、“拖拽”设计方式、操作简单、可扩展性强、跨平台等优点, 极大的降低了 VR 仿真软件的开发门槛和开发成本. 下一步将继续深入研究该引擎的在线网络功能.

参考文献

1 倪乐波, 戚鹏, 遇丽娜, 等. Unity3d 产品虚拟展示技术的研究与应用. 数字技术与应用, 2010, (9): 54–55.
2 林深华, 范志尚, 蒋建兵, 等. 基于 Android 平台 Unity3D 游戏设计与实现. 企业科技与发展, 2013, (10): 40–42. [doi: 10.3969/j.issn.1674-0688.2013.10.017]
3 Finney K C. 3D 游戏开发大全. 齐兰博, 肖奕, 译. 北京: 清华大学出版社, 2010.
4 刘琼. 基于 MVC 架构的英语教学系统设计及应用. 微型电脑应用, 2018, 34(11): 107–109. [doi: 10.3969/j.issn.1007-757X.2018.11.034]
5 符红霞. Spring MVC 技术分析及在实践教学系统中的应用. 信息技术, 2012, (10): 42–46. [doi: 10.3969/j.issn.1009-2552.2012.10.013]
6 朱惠娟. 基于 Unity3D 的虚拟漫游系统. 计算机系统应用, 2012, 21(10): 36–39. [doi: 10.3969/j.issn.1003-3254.2012.10.009]
7 李婷婷, 王相海. 基于 AR-VR 混合技术的博物馆展览互动应用研究. 计算机工程与应用, 2017, 53(22): 185–189, 263. [doi: 10.3778/j.issn.1002-8331.1605-0369]