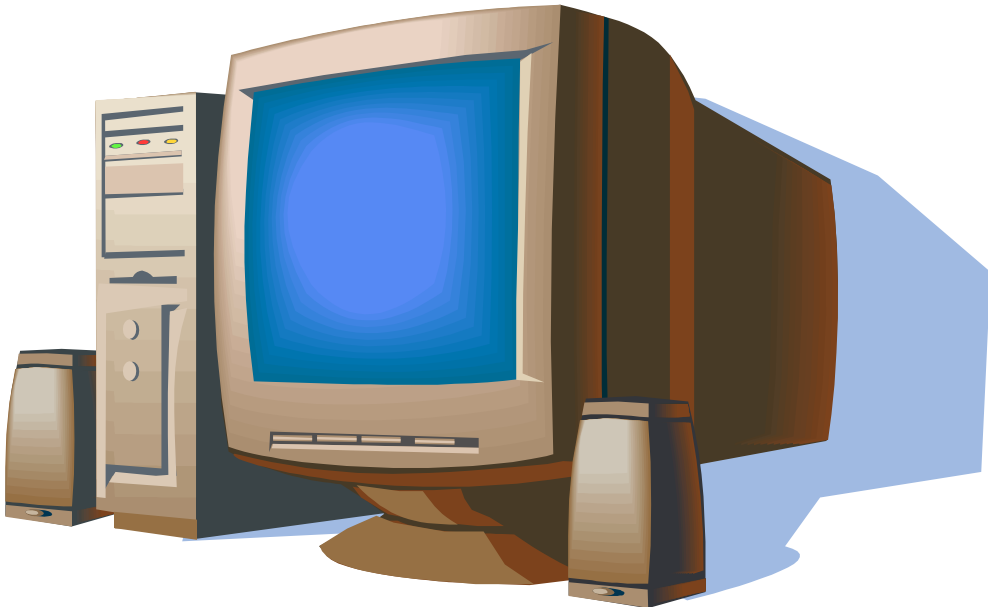
BigWorld 技术

客户端培训

概要

- 客户端配置
- 实体（Entity）
- 模型（Model）
- 动画与动作（Action）
- 摄像机与视角
- 3D 引擎
- 游戏相关
- 粒子系统与特效
- GUI
- 任务（Job）系统
- 性能分析和引擎统计数据

一、客户端配置



客户端配置– 概览

- 配置文件
- Personality 脚本
- 连接服务器

配置文件

- XML格式的配置文件
- 引擎配置：
  - 用于配置客户端引擎（C++）
  - ***<resources_folder>/engine_config.xml***
- 脚本配置：
  - 用于在启动时配置脚本
  - ***<resources_folder>/scripts_config.xml***
- 用户选项：
  - 配置图像、显示以及一些特定于游戏的设置
  - ***<working_folder>/preferences.xml***

配置文件

- 实体（entity）：
  - 列出所有游戏中用到的entity
  - ***<resources_folder>/scripts/entities.xml***
- 资源：
  - 指定游戏中用到的资源
    - 例如：加载界面
  - 也可以根据客户的需要覆盖这些设置
  - ***<resources_folder>/resources.xml***

Personality 脚本

- 用于启动客户端的全局脚本
- 加载与更新设置
- 初始化游戏
- 处理全局逻辑
  - 游戏菜单
  - 连接服务器
- 在 cellapp 和 baseapp 上也有 personality 脚本
- 文件位置：
  - ***<resources_folder>/scripts/client/<script_name>.py***

Personality 脚本

- 通过通知方法和回调函数来进行交互

方法	描述
init(...)	在引擎开始初始化时被调用 *(引擎、脚本和用户配置文件会作为参数传入)*
start(...)	当引擎初始化完毕、将要开始运行时被调用
fini(...)	当引擎退出时被调用
handleKeyEvent(...)	当用户按键时被调用
handleMouseEvent(...)	当用户移动鼠标时被调用
handleAxisEvent(...)	当用户移动手柄时被调用
onChangeEnvironments(...)	当玩家在室内和室外间切换时被调用
onRecreateDevice(...)	当屏幕分辨率改变时被调用
....	请参阅客户端 Python 文档中的 BWPersonality 部分

与服务器连接

- 与服务器的连接是由一个 BigWorld 模块中的 Python 方法来实现的
- ***BigWorld.connect(hostName, args, connectionCallback)***
  - ***hostName***：该字符串中是登录服务器的 IP 或是域名
  - ***args***：可选的 Python 对象，包括：
    - ***username***：玩家名称，用于显示和表示该玩家
    - ***password***：密码以确保玩家的真实性
    - ***inactivityTimeout***：如果在该成员所指定的时间（单位：秒）内没有得到更新，玩家将断开连接
  - ***connectionCallback***：可选的 Python 方法，其参数表示登录过程中的各种状态

连接回调函数

- ***def connectionCallback(self, stage, status, msg)***
  - ***stage:*** 定义连接在当前所处的阶段
    - **0** 客户端不能初始化登录
    - **1** 客户端已连接到登录服务器
    - **2** 客户端已收到来自登录服务器的数据（连接完毕）
    - **6** 客户端断开与登录服务器连接
  - ***status:*** 表示这一阶段的状态
    - 'NOT_SET' ：状态没有被设置
    - 'LOGGED_ON' ：帐号登录成功
    - 'CONNECTION_FAILED' ：登录失败：无法联系登录服务器
    - ...
  - ***msg:*** 用来描述该响应的文本信息

连接信息

- ***BigWorld.LatencyInfo().value***
  - 一个只读的 Vector4，描述客户端网络连接的延迟信息
  - ***isOnline、minLatency、maxLentency、avgLatency***

- ***BigWorld.serverDiscovery***
  - ***servers:*** 可用服务器列表（每个服务器都是一个serverDiscovery::Detail对象）
  - ***searching:*** 1-开始在局域网上搜索BigWorld服务器

    0-停止服务器搜索并清除服务器属性
  - ***changeNotifier:*** Python 函数，每当创建一个 serverDiscovery::Detail 对象或是对 servers 列表里的 serverDiscovery::Detail 对象更新时被调用

连接信息

- ***serverDiscovery::Details*** *(只读)*
  - ***hostName:*** 主机名
  - ***ip:*** 主机的 IP 地址，以整数形式表示，第一个字节为最高字节
  - ***port:*** 主机监听连接的端口
  - ***uid:*** 主机的用户 id，用于区分在一台物理服务器上运行的多个服务器程序
  - ***ownerName:*** 启动该服务器的用户的用户名
  - ***usersCount:*** 任何客户端与该主机尝试进行连接的总次数
  - ***universeName:*** 当前在服务器上运行的 `universe` 名称
  - ***spaceName:*** 当前在服务器上运行的游戏空间（space）名称

演示时间

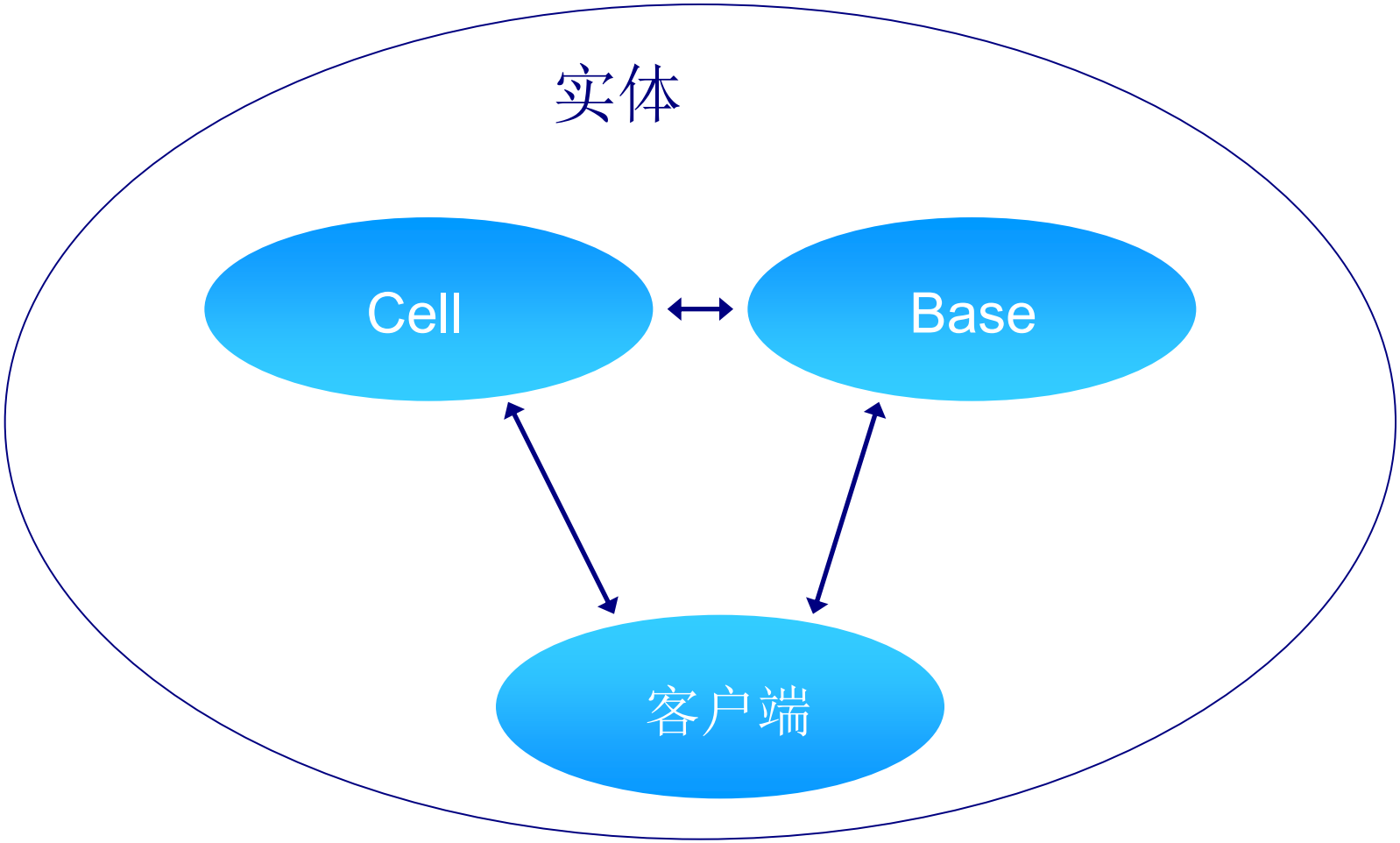
- 演示：
  - XML 配置文件
  - Personality 脚本
  - 连接服务器
  - 演示程序：

二、实体（Entity）

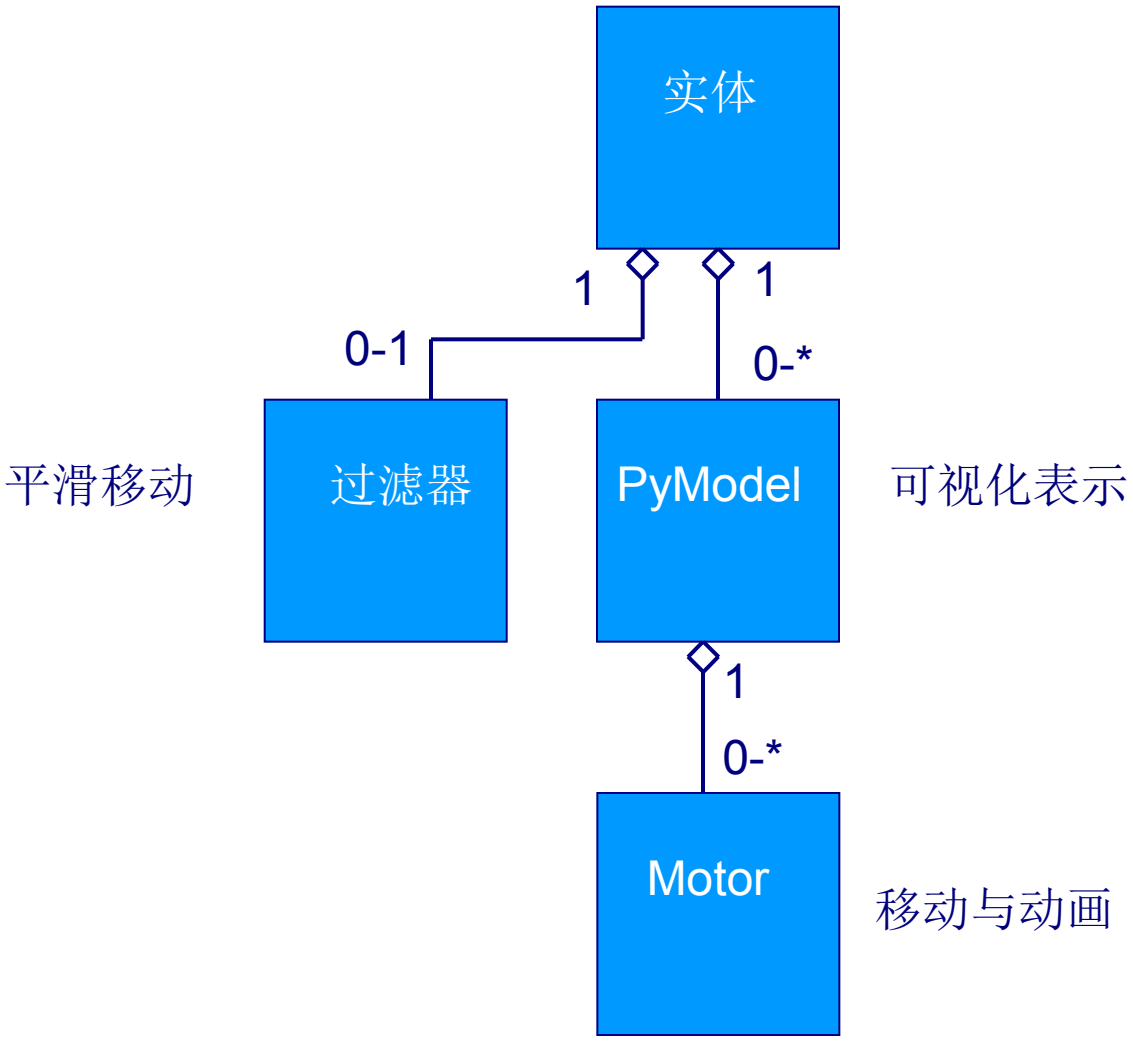
实体 – 概览

- 实体架构
- 客户端实体
- 实体脚本
- 客户端 – 服务器通信
- 物理属性
- 移动过滤器（filter）
- 玩家实体

实体架构



客户端实体



实体脚本

- 回忆《服务器培训》中的的分布式的实体模型
- 实体可以存在于 Cell、Base 和客户端
- 接口（定义）、Python（实现）
- 脚本实现了：
  - ***<ClientMethods>***，如定义文件(.def)所描述的
  - 属性通知方法：
    - ***set_<propertyName>() # From self.health = 20***
    - ***setNested_<propertyName>() # From self.myFixedDict["x"] = 2***
    - ***setSlice_<propertyName>() # From self.myArray.append(7)***
  - 常规回调函数
  - 其它客户端行为

实体回调函数

函数	描述
onEnterWorld(…)	当 BigWorld 把该实体加入游戏世界时被调用
onLeaveWorld(…)	当该实体要离开游戏世界时被调用
targetFocus(…)	我们获得了一个新的目标
targetBlur(…)	我们失去了我们的目标
targetModelChanged(…)	目标的模型改变了
reload(…)	当脚本被重新加载时调用，我们可以根据需要在这里重新设置我们的状态
onControlled(…)	让我们知道我们的受控状态已被改变

客户端 – 服务器通信

- 客户端实体用 base 和 cell mailbox 来与服务器通信
- ***Entity.base.methodName()***
  - 调用在 base 实体上的方法
  - 只能用于玩家实体
- ***Entity.cell.methodName()***
  - 调用 cell 实体上的方法
  - 可用于所有有 cell 部分的实体

物理属性

- 实体物理属性用于操作实体的移动
- 只有受控实体才有物理属性
- 实体必须给物理属性初始化一个物理属性类型
- 示例：
  - ***self.physics.type = BigWorld.STANDARD_PHYSICS***
    - 设置物理属性为标准的玩家风格物理
  - ***self.physics.collide = True***
    - 激活碰撞检测

类型	描述
0 – 标准	受重力影响，与场景碰撞
1 - Ripper	气垫车辆
2 - Limpet	用 **physics.chase** 来追随一个实体

使用物理属性

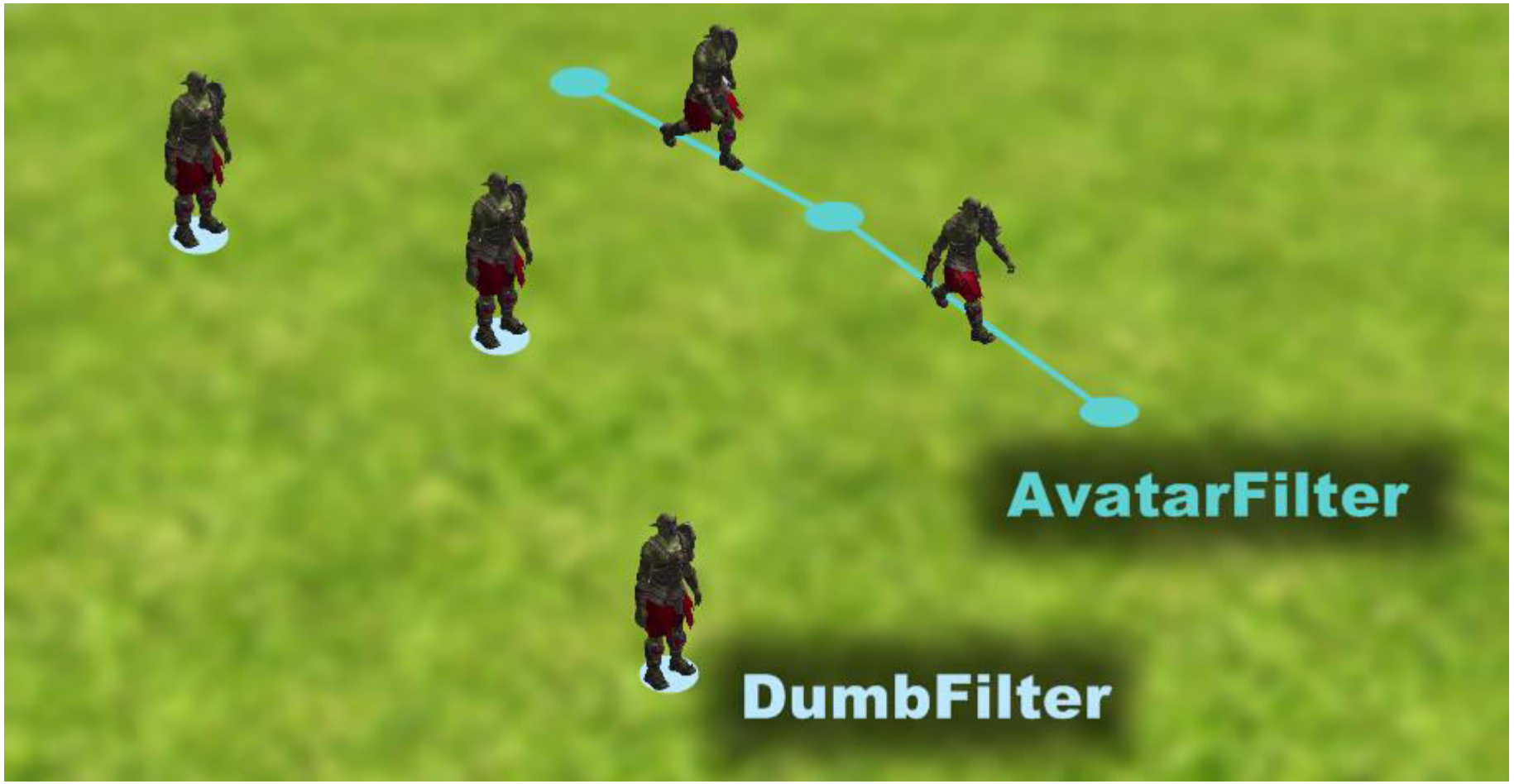
方法	描述
seek()	把玩家移动到某个位置
stop()	停止 seek 操作

属性	描述
velocity	设置玩家速度
userDirected	允许实体的 yaw 受 DirectionCursor （也就是鼠标）的影响
dcLocked	暂时使实体的 yaw 不受鼠标控制
gravity	设置重力值，譬如说 9.8
fall	设为 True 使得实体会下落
collide	允许与场景碰撞

移动过滤器（filter）

- 过滤器可以平滑游戏中的移动
- 来自服务器的位置更新也许不够频繁
- 用一个实体过滤器来提供实时的位置
- 示例：
  - ***Entity.filter = BigWorld.AvatarFilter()***
  - 创建一个简单的过滤器，并连接到实体上
  - 对服务器发送的位置进行线形插值

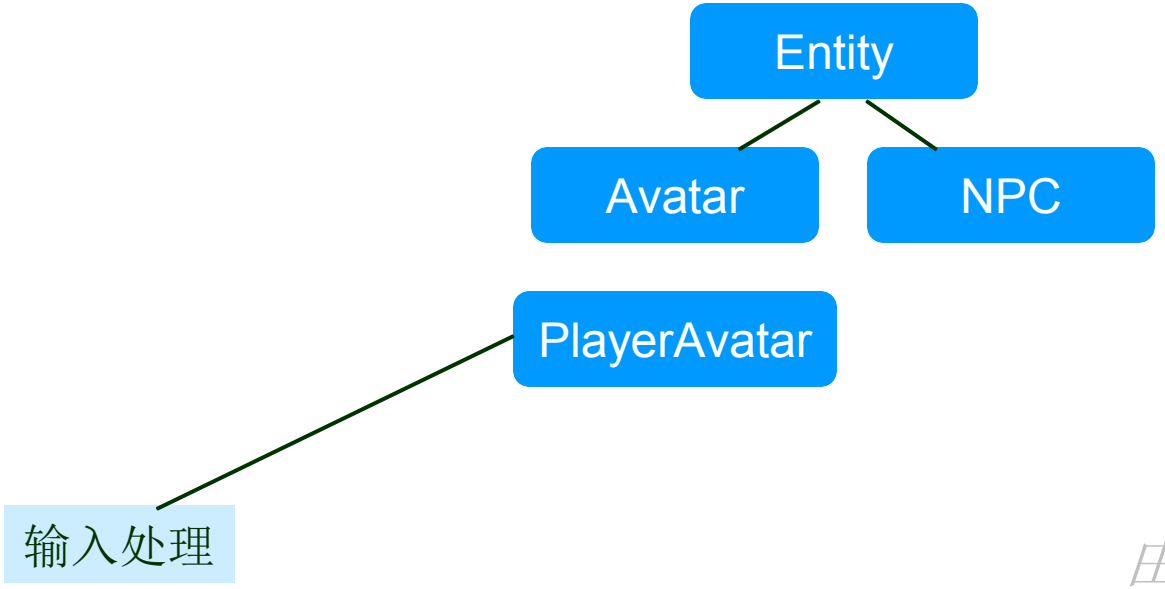
移动过滤器



移动过滤器

过滤器	描述
DumbFilter	什么都不做
AvatarFilter	对（来自服务器的）最后的8个位置进行线形插值
AvatarDropFilter	同上，但是同时把实体拉到地上
PlayerAvatarFilter	使用玩家的输入，但是也接受服务器的覆盖
BoidsFilter	用生物群（flock）算法控制一群模型

玩家实体



由玩家控制

玩家实体

- 玩家实体的 Python 对象必须：
- 从一个实际的Entity类派生而来的类
- 被命名为***Player<ParentClassName>***
  - 例如：***PlayerAvatar***
- ***BigWorld.player()***
  - 返回当前的玩家实体
- ***BigWorld.player(entity)***
  - 设置新的玩家实体
  - 如果不存在***Player<entity>***类则无效
  - 在上一个玩家实体上调用***onBecomeNonPlayer()***
  - 在新的玩家实体上调用***onBecomePlayer()***
  - 任何时候只能有一个玩家实体存在

受控实体

- 一个受控实体由客户端控制，并且发送位置更新到服务器 (例如玩家)
- 如果一个实体不是受控实体，则接受服务器的位置更新
- 可以存在任意多个受控实体（也就是说，受用户输入影响）
  - 大船上的玩家
  - A “Redeemer” weapon
  - A “Lost Vikings” game

演示时间

- 演示：
  - 实体
  - 物理属性
  - 过滤器
  - 玩家
  - 受控实体
  - 演示程序：

三、模型

模型 – 概览

- 模型概览
- 模型创建
- PyModel
- PyModelNode
- PyAttachment
- PyFashion
- 模型LOD
- Motor
- Tracker

模型 - 概览

- 模型代表一个 3D 对象
- Entity 可能有 0 个或多个模型, 通常为 1 个
- Entity 可以有一个主模型, 和 *n* 个次要模型
- 模型也可以被连接到其它的模型，稍后会给出更多细节...
- 示例：
  - ***pyModel = BigWorld.Model(modelFile,…)***
    - modelFile 接受 **.model** 的路径，返回一个 **PyModel** 对象
    - 可以传任意多个**.model** 路径, 它们根据node结构结合在一起产生一个**PyModel**
  - ***pyModel = BigWorld.PyModelObstacle(modelFile,…)***
    - 创建一个模型并加入碰撞场景

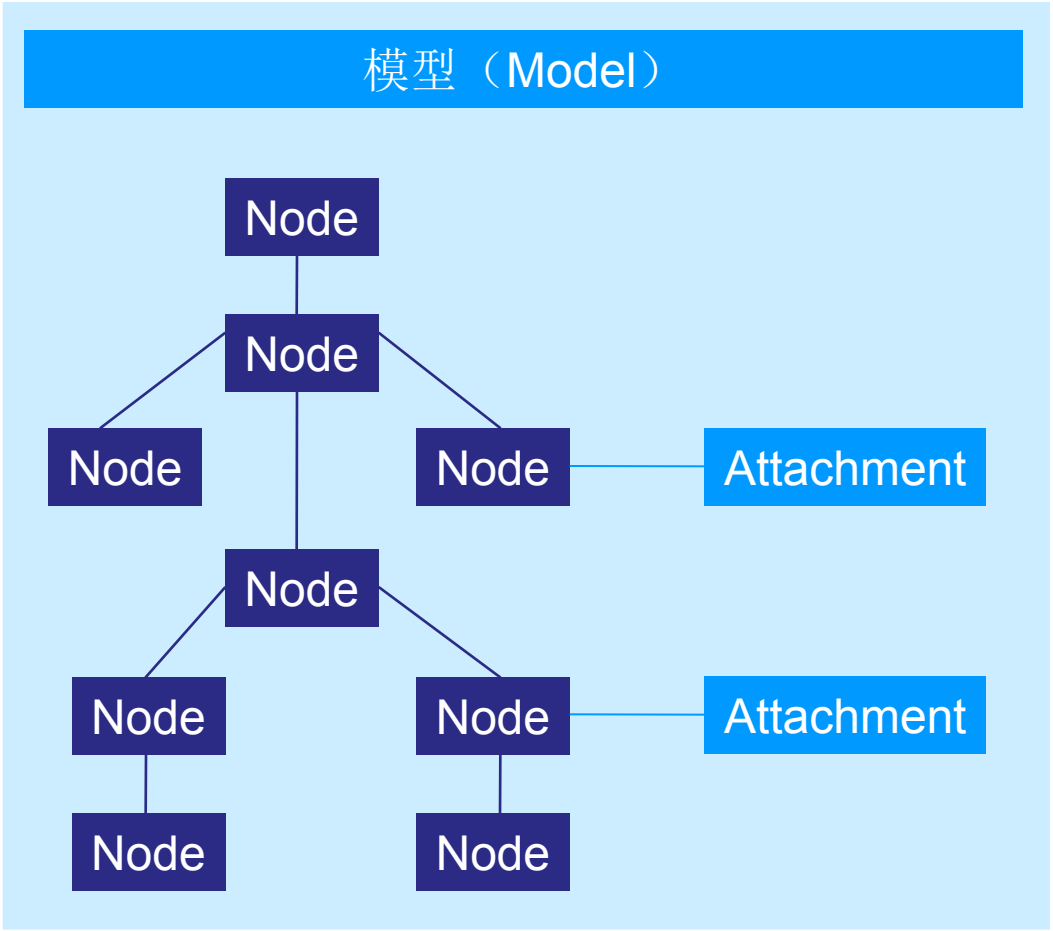
模型 - 概览

- ***entity.model = pyModel***
  - 设置实体的主模型
- ***entity.addModel(pyModel)***
  - 给实体添加次要模型（例如，一个魔法效果）
- ***entity.delModel(pyModel)***
  - 从实体中删除该次要模型
- ***entity.models***
  - 包含次要模型的 Python 列表
- ***model.position = (x,y,z)***
  - 设置模型的位置

模型创建

- 模型和动画用 3ds Max 创建并用 BigWorld 插件导出
  - ***.primitives*** 原始网格数据
  - ***.animation*** 原始关键帧动画数据
  - ***.visual*** 节点层级结构、渲染信息、包围盒.
    如果一个节点的名字以 **HP_** 开头，那么它是一个 hard point 属性，会返回一个***PyModelNode***。请参见示例文件.
  - ***.model*** 会引用 **.visual** 文件，其内容包含了动画、动作（action）和染色
    请参见示例文件.

PyModel



PyModelNode

- 用节点（node）为模型添加 attachment
- 两种访问 ***PyModelNode*** 的方法：
  - ***model.HardPointName*** 属性
    - 返回名为 ***HP_HardPointName*** 的节点
  - ***model.node("nodeName")*** 方法
    - 返回名为 **nodeName** 的节点
- ***model.myHardPoint = pyModel***
  - 附加一个次要模型在主模型的节点上 (例如一把枪)
- ***node.attach(PyAttachment)***
- ***node.detach(PyAttachment)***
- ***node.attachments***

PyAttachment

- 任意一个 ***PyModelNode*** 上都可以连接任意多个 ***PyAttachment*** 对象
  - ...但是一个 ***PyAttachment*** 同时只能被连接到一个 **PyModelNode** 上
- ***PyAttachments*** 包括:
  - ***PyModel*** 任何一个 ***PyModel*** 都可以被连接到另一个 PyModel 上
  - ***ParticleSystem*** 描述粒子行为
  - ***PySplodge*** 由阳光投射出的通用阴影
    - 可以设定宽、高和LOD
    - 影子的贴图在 ***engine_config.xml*** 中设置
  - ***GuiAttachment*** 可以连接 GUI 组件

PyFashion

- ***PyFashion*** 对象可以改变 ***PyModel*** 的外貌
- ***PyDye(matter, tint)***
  - 改变 ***PyModel*** 的部分材质，该材质在 **.model** 文件中设定. 参看 python_client.chm.

模型LOD

- 可以使用模型 LOD 来提高游戏性能
- 可以为复杂的模型定义简化的版本
- BigWorld 自动根据摄像机的位置切换模型
- LOD：
  - 材质：减少 shader 数量
  - Mesh：使用较少的多边形来创建模型
  - 节点：减少骨架节点的影响

Motor

- 可以用 Motor 来移动或是控制模型的方向，也可以用它来播放动画
- Entity 的模型缺省时会画在原点
- 当一个 ***PyModel*** 被赋值给 ***Entity.model*** 时，会自动被加上一个类型为 ***ActionMatcher*** 的 motor
- action matcher 会把模型移动到 entity 所处的位置
- 它也会播放动画 – 我们会在稍后讨论

Motor

- 示例:
  - ***motor = BigWorld.Propellor(parameters)***
    - 创建一个 ***propellor*** motor
  - ***self.model.addMotor(motor)***
    - 添加 ***motor*** 到模型
    - 如果有多个 motor，每个 motor 会依次影响 **PyModel**，这可能会产生冲突
  - ***self.model.delMotor(motor)***
    - 从模型删除 ***motor***

Motor 类型

Motor	描述
ActionMatcher	移动 PyModel 到 entity 所处的位置并播放相应的动画
Propellor	在一个点上运用推力来移动 PyModel
Oscillator	前后转动模型（监控摄像头）
Oscillator2	绕着世界的原点震荡
LinearHomer	笔直地向目标移动
Homer	以指定的角速度和速度行进到目标
Bouncer	用物理属性来反弹模型（手榴弹）
Servo	使用 MatrixProvider 来移动 PyMode

Tracker

- Tracker 改变动画上的一个节点
- 使用一个 direction provider 来改变 pointingNode 的 yaw 和 pitch
- 使用 ***TrackerNodeInfo,*** 把连接到 pointingNode 上的其它节点混合到中间位置
  - 例如：转动头使之面向另一个实体，同时也部分地转动颈部和肩部
- tracker 会在动画对节点施加影响后把节点转向一个方向

Tracker

- 示例:
  - ***tracker = BigWorld.Tracker()***
  - ***tracker.directionProvider =***
    BigWorld.DiffDirProvider(sourceMatrix,
    targetMatrix)
  - ***tracker.nodeInfo =***
    BigWorld.TrackerNodeInfo(self.model,
    "Head", [("Neck", 0.5)],
    "None", -100, 100, -100, 100, 100)
- 添加与删除 tracker:
  - ***self.model.tracker = tracker***
  - ***del self.model.tracker***

演示时间

- 演示：
  - 模型创建
  - PyModel 与 PyModelNode
  - Attachment
  - Fashion
  - LOD
  - Motor
  - Tracker
  - 演示程序：

四、动画与动作（action）

动画与动作 – 概览

- 播放动作
- 动作架构
- ActionQueuer
- ActionMatcher

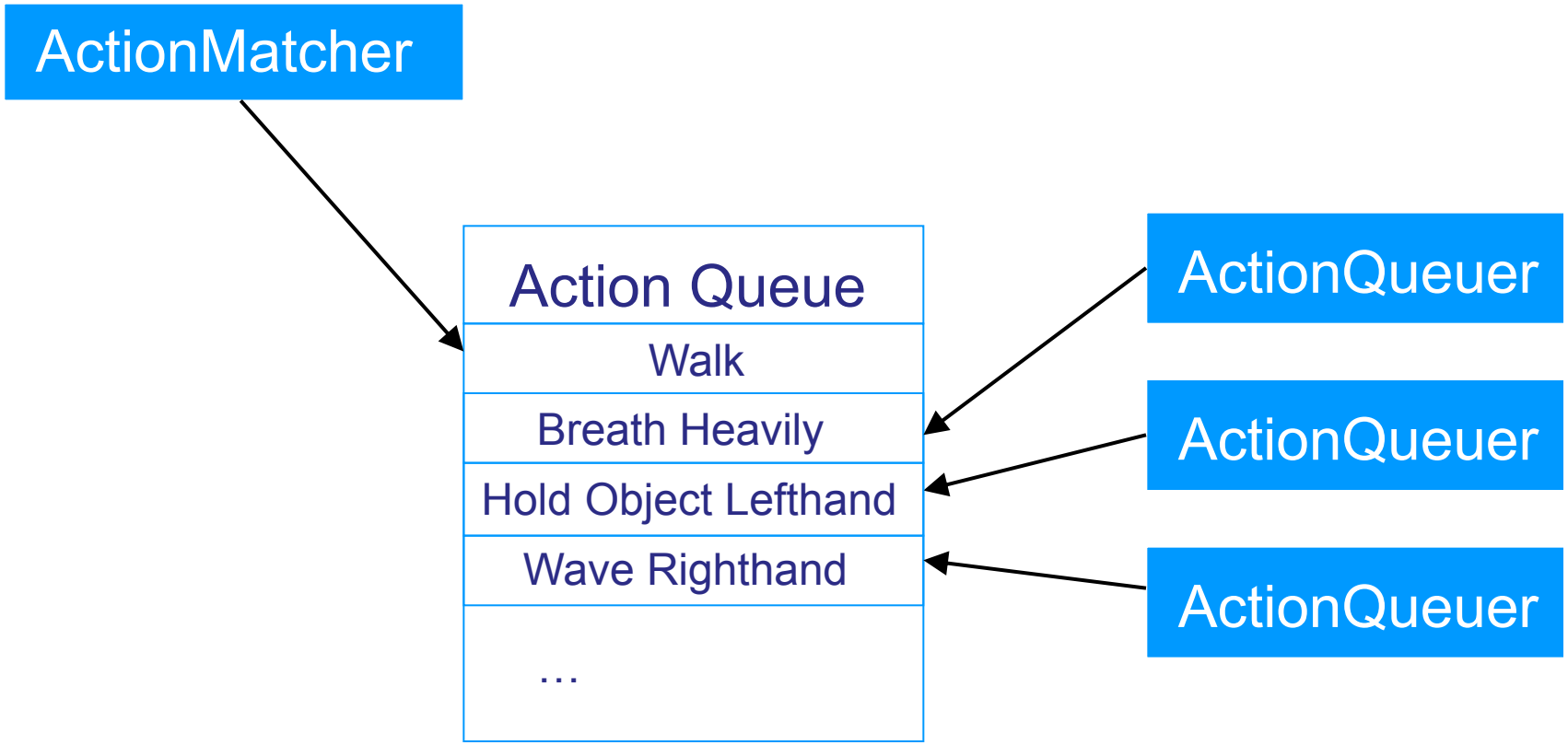
动画和动作 - 概览

- 动画用骨骼或变形动画去移动 3D 模型
- 可以使用动作在模型上播放动画，它还具有一些附加属性

播放动作

- **ActionQueuers** 可以被用来在 Python 中对动画进行控制
- 立刻播放一个动作：
  - `Model.Jump()`
- 1 秒后播放一个动作：
  - `Model.Jump(1.0)`
- 1 秒后播放一个动作，并且在动作结束时回调脚本：
  - `Model.Jump(1.0, self.onJumpComplete)`
- 在一个动作播放完毕后紧接着播放另一个动作：
  - `Model.Jump().Land()`

动作架构



ActionQueuer

- 可以把一个动作保存下来以便在任何时候对其进行调用
  - `myJumpAction = Model.action("Jump")`
  - `myJumpAction = Model.Jump`
- **`myJumpAction(afterTime, callBack, promoteMotion)`**
  - 把 **`ActionQueuer`** 加入到 **`ActionQueue`** 中
  - 所有参数都是可选的
    - **`afterTime`** - 在把动作加入对列前延迟的时间
    - **`callBack`** - 当动作完成时所调用的回调函数
    - **`promoteMotion`** - 布尔类型，指定动画是否会改变实体的位置

ActionQueuer

- 所有的动作都是混合的
  - 动画不会突然发生改变
  - 淡入淡出
  - 与正在播放的其他动作混合
  - 混合都是在各个节点上进行的

ActionQueuer 属性

属性	描述
track	整数类型。在同一个 track 上的动作会取代在该 track 上正在播放的其它动作。-1 表示不属于任何 track
blendInTime	一个动作完全影响该模型所需要的秒数（淡入时间）
blendOutTime	一个动作完全停止对模型的影响所需要的秒数（淡出时间）
duration	整个动作播放完毕所需的时间
displacement	动作使模型移动的距离（米）
rotation	动作使模型旋转的弧度

ActionMatcher

- 基于服务器发送的位移更新，选择和显示动画
  - 当把 **PyModel** 赋值给 **Entity.model** 的时候，ActionMatcher 会被加为缺省的 motor
- 每个动作都可以有一个 **<match>** 项，它可以包含一个 **<trigger>** 和一个 **<cancel>** 项
  - Match 的条件有 minSpeed、maxSpeed 等
  - 示例：base.model
  - 这些都可以在模型编辑器里进行设置
- 在游戏里查看 **ActionMatcher**：
  - **BigWorld.debugAQ(BigWorld.player())**
  - **Tests.ActionMatcher.test()**

ActionMatcher 属性

- **matcherCoupled**
  - 缺省为 1，设为 0 则关闭
- **inheritOnRecouple**
  - 缺省为 1。玩家实体会被移到 **PyModel** 的位置以避免视觉问题
- **lastMatch**
  - 上一个匹配动作的名称
- **matchCaps**
  - 用户可以为 **ActionMatcher** 定义一些状态，这些状态可以用 0 到 31 之间的数字表示。matchCaps 就包含了一个数字列表表示这些状态。
  - 动作可以指定一些状态作为其被选中匹配的前提

ActionMatcher 属性

- **turnModelToEntity**
  - 决定 **ActionMatcher** 是否应该调整 **PyModel** 的 yaw 与实体的方向一致
    - 如果设为 0，ActionMatcher 将不会改变 yaw
- **footTwistSpeed**
  - 对 **PyModel** 的 yaw 进行调整使之朝向所属实体当前方向的角速度

演示时间

- 演示：
  - 动画
  - 动作
  - ActionQueuer
  - ActionMatcher
  - 演示程序：

五、摄像机与视角

摄像机与视角 – 概览

- 摄像机类型
- 创建和使用摄像机
- 设置视角

摄像机类型

- CursorCamera
  - 用 **`DirectionCursor`** 在离实体指定的距离跟踪视线
- FreeCamera
  - 没有限制，没有碰撞，可以在游戏世界中自由移动，由方向键和鼠标控制
  - **`freeCamera.fixed = 1`** 将停止任何移动

创建和使用摄像机

- **BigWorld.CursorCamera()**
  - 创建一个 **CursorCamera**
- **camera.target = BigWorld.PlayerMatrix()**
  - **BigWorld.PlayerMatrix()** 返回 **MatrixProvider**
  - **target** 会随着玩家位置的更新而更新
- **BigWorld.camera(*camera*)**
  - 在任何时刻只能使用一个摄像机
- **BigWorld.firstPerson(true)**
  - Cursor camera 可以为第一人称或第三人称模式

创建和使用摄像机

- **camera.set(*transformMatrix*)**
  - 设置摄像机的世界变换
- **camera.position**
  - 摄像机当前的位置
- **camera.direction**
  - 摄像机当前的朝向
- **camera.matrix**
  - 世界坐标系 -> 摄像机坐标系
- **camera.invViewMatrix**
  - 摄像机坐标系 -> 世界坐标系

设置视角

- **BigWorld.projection()**
- 返回唯一的投影对象
- 包括:
  - **nearPlane**
    - 小于该距离的面都会被裁减掉
  - **farPlane**
    - 大于该距离的面都会被裁减掉
  - **fov**
    - 当前摄像机的视野领域（在 0 到 pi 之间取值）
  - **rampFov(*newFOV*, *timeAllowed*)**
    - 在指定的时间内过渡到指定的视野领域值

六、3D 引擎

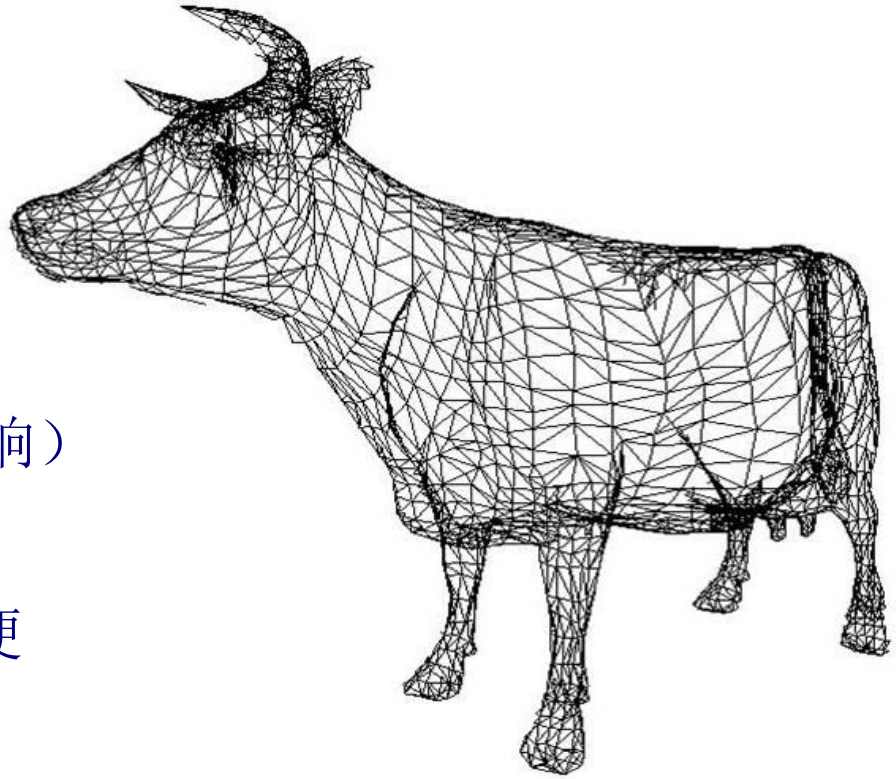
bigWORLD™ TECHNOLOGY

3D 引擎 – 概览

- 引擎概览
- Visual
- EffectMaterial
- 光照系统
- 贴图
- 基于高度图的地形

3D 概览 – 概览

- DirectX9 引擎
- 左手坐标系统
  - X – 右、Y – 上、Z – 指向屏幕内部
- 大量地使用 .fx 文件
- 基于高度图的地形
- 骨骼动画
- 蒙皮（每个顶点受到三根骨骼影响）
- 通过渲染通道来实现延迟渲染
- 命名 Moo 的原因是因为键入方便



Visual

- 底层的可渲染对象，从内容创建软件中导出
  - 3ds Max, Maya
- 大部分游戏对象都是以 visual 方式渲染的
  - 角色、场景物件、壳体（shell）、等等...
- 包括：
  - 图元　　顶点及索引
  - 骨骼　　节点层次数据
  - 材质　　**`EffectMaterial`**
  - 碰撞数据　　BSP 树

EffectMaterial

- 使用 ID3DXEffect
- 变量
  - 可由美工编辑的变量
  - 自动变量（使用变量语义（semantics））
- 技术标记（technique annotation）
  - 将蒙皮、法线贴图、渲染通道以及图像设定等信息传递给渲染系统

光照系统

- 8 个动态灯
  - 2 个方向灯（directional light）
  - 4 个点光源（point light）
  - 2 个探照灯（spot light）
  - 会被物体遮挡
- 静态的顶点光照
  - 壳体和室内物件

属性	方向灯	点光源	探照灯
颜色	✓		
方向	✓	×	✓
位置	×	✓	✓
内径	×	✓	✓
外径	×	✓	✓
锥形角度	×	×	✓

贴图

- 引擎根据 **TextureDetailLevel** 或是 **.texformat** 文件定义的规则对贴图进行压缩和缩放
- **TextureDetailLevel** 将资源名（部分或全部）与某条规则匹配以决定：
  - 贴图格式
  - 压缩后的贴图格式
  - 贴图是否需要被缩放
- **.texformat** 文件是一个 XML 文件，它与贴图的质量设定相关，可以被用来决定贴图的格式
  - 如果存在一个与贴图文件同名的 **.texformat** 文件，那么将由它来决定这个贴图的格式

贴图

- 经过压缩和缩放的贴图会被保存为与源贴图文件同名的**.dds** 文件（如果是压缩的则为 **.c.dds**）

基于高度图的地形

- 每个地形块 100x100 米
  - 与 chunk 大小一样
- 每个游戏空间（space）可以定义不同的高度网格分辨率
- 任意层的贴图
  - 每个 pass 4 层
  - 每个游戏空间可以设定自己的地形层的mask细节
  - 贴图的分辨率与高度网格分辨率无关
- 贴图的 alpha 通道存储了高光亮度值

演示时间

- 演示：
  - Effect 文件
  - 光照系统
  - 贴图
  - 演示程序：

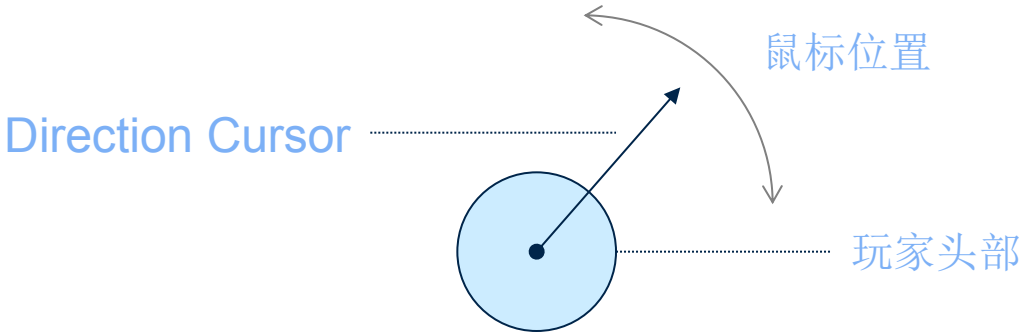
七、游戏相关

游戏相关 – 概览

- Direction Cursor
- Targeting
- 陷阱（trap）
- MatrixProvider 和 Vector4Provider
- 性能分析和引擎统计数据

Direction Cursor

- Direction Cursor 是一个从玩家头部延伸的向量
- 鼠标的输入会影响这个向量的角度，从而产生一个可能的视线
- 提供一个完整的动态矩阵（位置和方向）



Direction Cursor

- **`dc = BigWorld.dcursor()`**
  - 获得一个 Direction Cursor 对象
- **`BigWorld.dcursor(dc)`**
  - 激活 direction cursor（缺省是激活的）
- 许多 BigWorld 对象可以使用 Direction Cursor
  - Trackers 能使身体的部分转向视线的方向
  - ActionMatcher 使 PyModel 与 pitch/yaw 匹配
  - Cursor Camera 位于头的后方，沿着视线的方向观看

Targeting

- **`BigWorld.target`**
  - 访问 BigWorld Targeting 系统
- **`BigWorld.target()`**
  - 获得离屏幕中央最近的实体
  - 你也可以用你自己定义的视角向量来进行 targeting
- 用位域来设置可否被 targetting
  - **`BigWorld.target.caps( [NPCs, Monsters] )`**

陷阱

- 当有玩家实体进入或离开陷阱时，会触发回调函数
- **BigWorld.addPot(*centre, radius, callback*)**
  - 返回一个可以被存储的陷阱 ID
  - 根据给定的 **MatrixProvider** 和 ***radius,*** 创建一个球型的、仅对玩家有效的陷阱，当陷阱被触发时激活回调函数 ***callback***
- **def trapCallback(*entered, trapID*)**
  - 如果玩家进入陷阱 ***entered*** 为 1，如果离开陷阱则 ***entered*** 为 0
- **BigWorld.delPot(*trapID*)**
  - 删除陷阱

MatrixProvider 和 Vector4Provider

- provider 提供了一个抽象接口，可以用它来查询某个特定类型的值
  - 因为 provider 是一个类对象，并且其查询方法是虚的，因而实现 provider 接口的类可以具有任意的动态性
  - BigWorld 使用这一范式从一个对象到另一个对象之间持续地传递变化的值
    - 对象不需要知道 provider 实现类的类型
  - 这使得我们可以在 Python 中建立控制器（controller）而不需要在每一桢中对任何 Python 代码进行调用

MatrixProvider 和 Vector4Provider

- Vector4Provider
  - 一些示例：
    - Vector4Animation - 在两个或多个 Vector4 之间随着时间进行插值
    - Vector4LFO - 提供一个随着时间变化的波形
    - Vector4Morph - 在两个值之间随着时间进行渐变（morph）
- MatrixProvider
  - 一些示例：
    - MatrixAnimation - 在两个或多个矩阵之间随着时间进行插值
    - PyModelNode - 这个 MatrixProvider 提供了节点在世界坐标系里面的变换
    - MouseTargetngMatrix – 它所提供的矩阵可以在世界坐标系中表示鼠标（也就是说，可以被用来做选取）

Vecto4Provider 用法示例

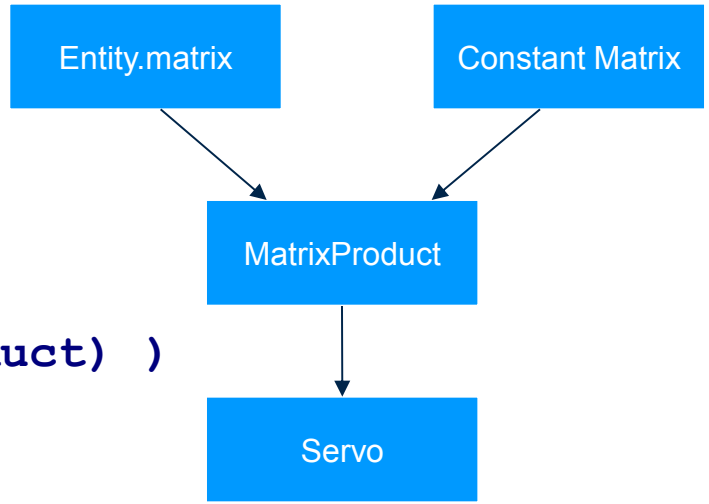
- 对暴露给 Python 的 shader 参数进行动画：
  - 譬如说，用 Vector4LFO 为模型创建一个闪动效果

```
lfo = Math.Vector4LFO()
lfo.period = 0.5
lfo.waveform = 'SINE'
$p.model.Single_material_skinned = 'Merchant'
$p.model.Single_material_skinned.clothesColour3 = lfo
```

MatrixProvider 用法示例

- 使用 motor 来移动一个模型
  - 譬如说，让一个盒子漂浮在玩家上空两米处

```
model = BigWorld.Model("sets/town/props/axe.model")
t = Math.Matrix()
t.setTranslate( (0,2,0) )
product = Math.MatrixProduct()
product.a = $p.matrix
product.b = t
model.addMotor( BigWorld.Servo(product) )
$p.addModel( model )
```



演示时间

- 演示：
  - Direction Cursor
  - Targeting
  - 陷阱
  - MatrixProvider 和 Vector4Provider
  - 示范程序：

八、粒子系统和特效

粒子系统和特效 – 概览

- 粒子系统
- 花草（flora）
- 天气
- 水
- 后处理（Post Processing）

粒子系统

- **SpriteParticleRenderer()**
  - 把每个粒子都作为一个 sprite 来渲染
- **PointSpriteParticleRenderer()**
  - 把每个粒子都作为一个 point sprite 来渲染（最大尺寸为 64 像素）
- **AmpParticleRenderer()**
  - 渲染一系列扭动的线条到起始点以模拟电流等
- **BlurParticleRenderer()**
  - 渲染拖尾效果（开销较小）
- **TrailParticleRenderer()**
  - 渲染拖尾效果（开销较大）
- **MeshParticleRenderer()**
  - 把每个粒子都作为一个网格来渲染

粒子系统的action

Action	描述
Source	根据时间、移动或是调用请求来创建粒子
Sink	根据时间或速度来删除粒子
Barrier	形成一个看不见的形状来反弹或停止粒子
Force	对粒子施加一个常量加速度
Stream	使粒子的速度向着一个流收敛
Jitter	给粒子添加随机的颤动
Scaler	随着时间对粒子进行缩放
TintShader	根据粒子存活的时间来改变其颜色
NodeClamp	把粒子连接到 **`PyModel`** 的节点
Orbiter	使粒子绕某一点盘旋
Flare	在一个或多个粒子上画光晕
Collide	使粒子与 3D 场景碰撞
Splat	与 Collide 类似, 但是在每个碰撞时都会调用脚本函数
MatrixSwarm	使粒子在目标周围云集
Magent	加速粒子到某一点

使用粒子系统

- **Pixie.create(*fileName*)**
  - 返回一个 ***ParticleSystem*** 或 ***MetaParticleSystem***
  - 从名为 ***fileName*** 的 XML 文件加载粒子定义
- **Pixie.ParticleSystem(*capacity*)**
  - 创建一个空的 **ParticleSystem**
  - 初始化容量为 ***capacity***（缺省为100）
- **Pixie.MetaParticleSystem()**
  - 创建一个空的 **MetaParticleSystem**
- **model.rightHand = *ps***
  - 把粒子系统 *ps* 连接到 **rightHand** 节点

天气

- 暴露给 Python 的底层天气 API：
  - **BigWorld.addSkyBox** – 淡入/淡出天空盒模型
  - **BigWorld.weatherController** – 控制雨量
  - **BigWorld.sunlightController** – 日光颜色乘数
  - **BigWorld.ambientController** – 环境光颜色乘数
  - **BigWorld.fogController** – 雾色彩和密度乘数
- 由 Python 实现的高层天气 API：
  - 基于 XML 的天气类型
  - 所有参数可以在实际编辑器中编辑
  - 在不同天气类型中进行平滑过渡
- 客户端通过环境同步来保持同步
- 通过创建天气实体来控制局部天气

地表物

- 这一系统可以在玩家周围一定半径内高效地渲染大量的细节对象
  - 譬如说：草、地衣、石砾或是残骸
    - 各个地表物对象的形状是由 .visual 文件定义的。
  - 无交互 - 也就是说玩家和地表物之间没有碰撞
  - 可以通过使用一个 Perlin 噪音发生器使之随时间进行变化
- 当玩家移动时根据距离进行淡入淡出
- 根据所在地形的贴图自动放置
  - 譬如说，一个草地贴图所创建的细节对象会是草
  - 贴图 -> 地表物的映射由 flora.xml 文件定义

水

- 可以在地图编辑器中放置各自独立的水体
- 可以调整 shader 参数
  - 譬如说，颜色、菲涅尔指数（Fresnel exponent）、贴图缩放、波纹大小/速度/方向
- 反射/折射
- 基于法线贴图的波纹物理仿真，会对物体在水面的移动作出反应
- 使用 MRT 深度缓存来创建软边界（soft edge）和泡沫效果
  - 高级特性

后处理

- 完全可定制的后处理链
  - 一个链包含了一系列效果
  - 一个效果包含了一系列阶段
- 可以在地图编辑器中编辑
- 在任何时候都会有一个链被激活
  - 在链中的效果可以被打开或关闭（旁路）
  - 在定义效果如何叠加时次序非常重要
- 每个阶段都是由一个 pixel shader 定义的
  - 一个阶段的输出会被输入到下一个阶段

演示时间

- 演示：
  - 粒子系统
  - 天气
  - 花草
  - 水
  - 后处理
  - 演示程序：

九、GUI

GUI – 概览

- GUI 组件
  - 提供一个 2D 四边形体系，它们可以在屏幕上按照正确的次序进行渲染并检测输入事件
  - 可以被串行化到 XML 文件
- 组件脚本
  - 一个 Python 类的实例，它可以关联到某个特定的组件上以处理逻辑
  - 譬如说，按钮、滚动条和操作栏
- 输入处理
  - 组件会向所关联的脚本对象发送键盘和鼠标事件
- 鼠标光标

组件类型

- **Simple(*texture*)**
  - ***texture*** 可以是一个贴图的路径或一个 **PyTextureProvider**
  - 所有其它的组件都从这一类型派生而来
- **Window(*texture*)**
  - 子窗口会继承窗口的位置
  - 会对子窗口按照窗口范围进行裁减
  - 可以被用来对窗口内的子窗口进行滚动
- **Frame2(texture)**
  - 可以改变大小的顶层窗口，它可以避免贴图拉伸
- **BoundingBox(*texture*)**
  - 在一个实体（投射到屏幕上的）包围盒的角上渲染 ***texture***

组件类型

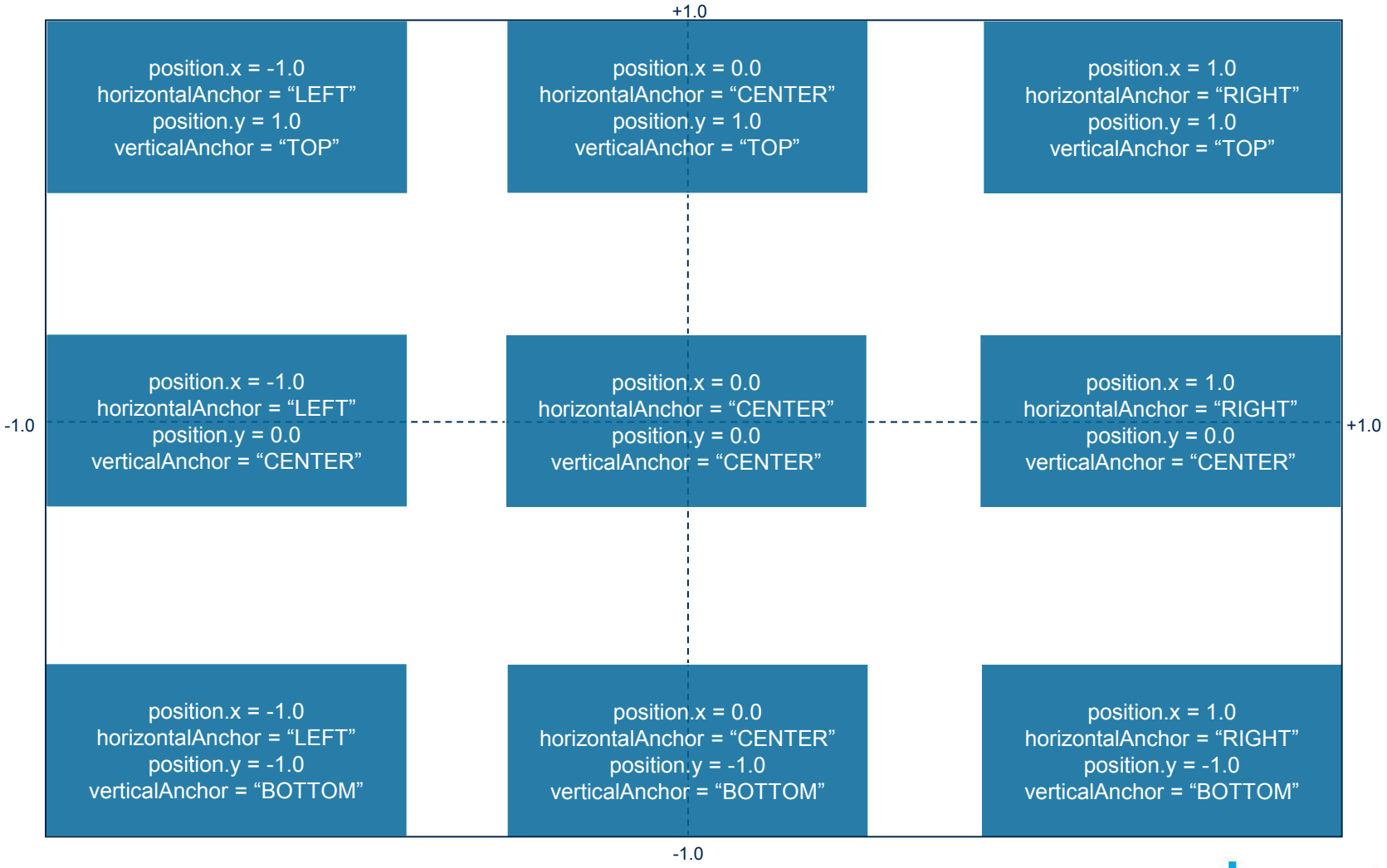
- **Text(*text*)**
  - 使用一个位图字体渲染一行文字
  - 可以在运行时修改当前所使用的位图字体
- **MeshAdaptor()**
  - 包含了一个**PyModelAttachment**，可以被加入到界面树中

组件布局

- 缺省情况下，组件（的大小和位置）是定义在裁减空间（clip space）中的
  - 因此，0, 0 是屏幕的中心，Y 轴正方向向上，X轴正方向向右
  - 裁减空间在两个轴上的范围都是 -1.0 到 +1.0，因此屏幕的宽和高都是 2.0
  - 这使得 GUI 可以被设计得与分辨率无关
  - 这也使得我们可以把一个组件的位置停靠（anchor）不同的位置
    - 也就是说，我们可以把某个组件在裁减空间的横坐标位置设为 1.0、横向的停靠方式设为 RIGHT，使得它漂浮在屏幕的右边。

组件布局

裁减空间的位置和停靠方式



组件布局

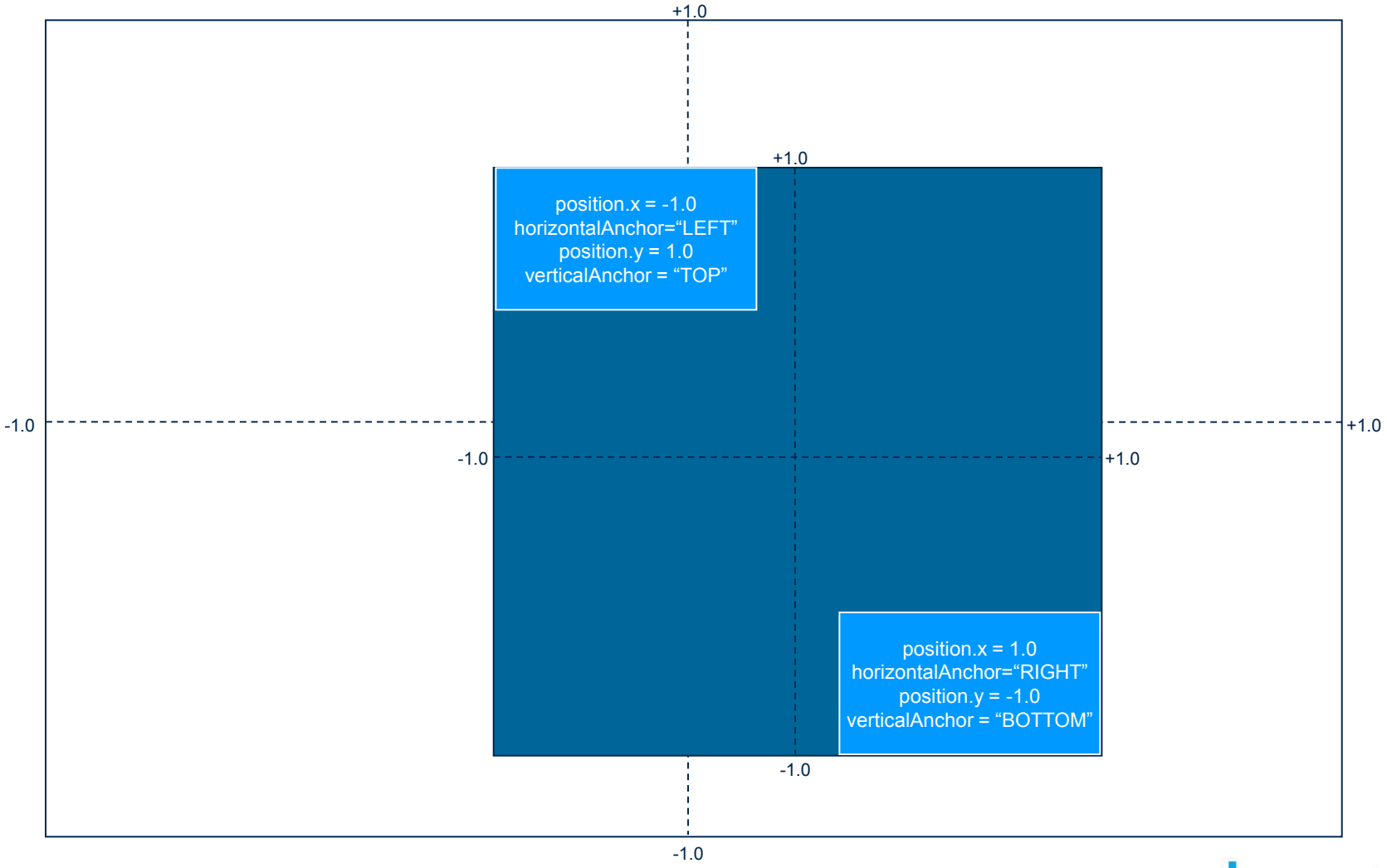
- 一个 SimpleGUIComponent 不会从它的父组件那里继承位置
- ...除非它的父组件是一个 Window 组件
  - 如果一个子组件是在裁减空间里定义的，那么这个裁减空间是指窗口所包含的区域
    - 因此，0,0 是窗口的中心，-1.0 是窗口的左边，+1.0 是窗口的右边，以此类推

组件布局

- 我们也可以使用像素来为组件定义位置
  - 相对于屏幕的左上方（如果它的父组件是 Window 组件，则相对于其父组件）

组件布局

Window 变换的继承



GUI Shader

- 修改 GUI 组件的外观
  - 不要和 Vertex/Pixel shader 混淆
  - 可以把多个 shader 和一个组件关联起来
- **AlphaShader()**
  - 控制 GUI 组件的 alpha 混合
- **ClipShader()**
  - 裁减一个组件，例如一个进度条
- **ColourShader()**
  - 动态地改变或调整 GUI 组件的颜色
- **MatrixShader()**
  - 移动和缩放界面部件

示例

- 创建和配置组件

```
guiBG = GUI.Simple("bg.bmp")
guiBG.position = (-1, -1, 0.5)
guiText = GUI.Text("blah")
guiText.position = (-1, -1, 0.4)
guiText.font = "default_small.font"
```

- 把组件添加到 GUI 组件树中

```
guiBG.addChild(guiText)
GUI.addRoot(guiBG)
```

- 为 GUI 组件应用 shader

```
alphaBlend = GUI.AlphaShader()
guiBG.addShader(alphaBlend)
```

把 GUI 关联到模型

- **GuiAttachment** 继承于 **PyAttachment**
  - 能被关联到一个 **PyModelNode**
  - 可以把 GUI 组件关联到 **GuiAttachment** 上
- 示例

```
guiBG = GUI.Simple("bg.bmp")
guiAttach = GUI.Attachment()
guiAttach.component = guiBG
model.rightHand = guiAttach
```

输入处理

- 每个 GUI 组件都可以通过设定合适的标志并实现所需的回调函数来处理输入事件。
- 输入回调函数：
  - handleKeyEvent(…)
  - handleMouseEvent(…)
  - handleAxisEvent(…)
  - handleMouseClickEvent(…)
  - 等等

鼠标光标

- MouseCursor 对象封装了所有的鼠标光标行为
- 鼠标的形状是在鼠标定义文件里定义的
  - 缺省文件：gui/mouse_cursors.xml
  - 或是在 resources.xml 文件的 gui/cursorsDefinitions 下定义
- 我们也可以使用一个 PyTextureProvider 来定义鼠标光标的形状
  - 例如，你可以集合一个PyModelRender来实现一个动态地渲染的三维的鼠标光标

演示时间

- 演示：
  - 界面部件
  - 输入的处理
  - 鼠标光标
  - 演示程序：

十、任务（job）系统

任务系统 – 概览

- CPU 核的利用
- 任务
- Direct3D 包装器（wrapper）
- 同步块

CPU Core Utilisation

- 在多核系统中，可以把工作分割并让多个核来分担
  - **核 1**：　　主线程
  - **核 2**：　　渲染线程
  - **核 3**：　　后台载入线程
  - **核 4..N**：　　任务线程
- 双核被视为一个特例
  - **核 1**：　　主线程
  - **核 2**：　　渲染和任务线程
  - 后台载入线程可以在两个核上移动
- 在单核系统上禁用任务系统

任务

- 任务是一个离散的处理单位，它可以被任务核分担
- 任务总是依次开始，但是可以以任何次序结束
- 当完成一个任务后，任务核会到任务队列中按顺序取出下一个需要执行的任务来执行
- 可以给每个任务一个指针，它可以在此写入输出数据。
  - 使用 **`JobSystem::allocOutput`** 来创建

Direct3D 包装器

- Direct3D 在它自己的核上运行
  - 这可以防止显卡空闲
- 在 API 级别对设备进行包装，对于用户来说是透明的
  - 譬如说，可以像往常一样调用 **`DrawPrimitive`** 而不用担心它在何时何地对设备进行真正的调用
  - 调用会在一个命令缓存中排队以在下一帧中执行

Direct3D 包装器 – 资源锁定

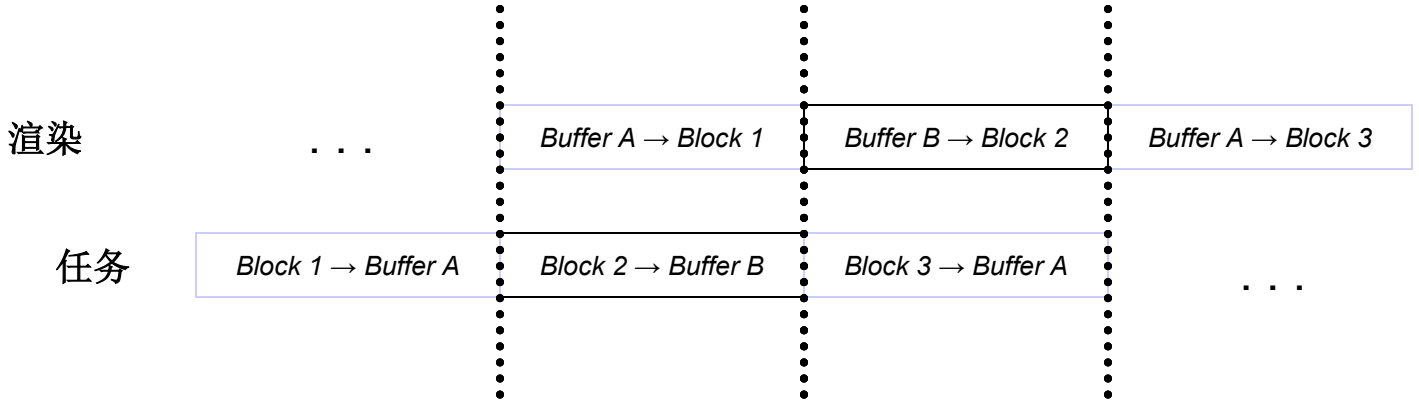
- 像顶点缓存（vertex buffer）这样的资源可以利用延迟锁定（deferred locking）
  - 这可以被用来在一个或多个任务中对缓存进行填充。
  - 使用 **`JobSystem::allocOutput`**，因此该锁定所得到的指针必须只在任务的执行方法中被使用！

同步块

- 每一帧都会被分成“块”
  - 依次渲染，一个接着另一个（同步地）
  - 每个块都有一组相关联的 D3D 命令和任务
  - 可以用任意数量的任务来为一个特定的块创建输出

同步块

- 任务线程总是比渲染线程先执行一块
  - 希望任务线程在渲染线程完成渲染时已经完成！
  - 如果渲染线程先遇到同步壁垒，它会等待所有任务的完成
- 任务输出是双缓冲的，因而任务线程最多也只能比渲染线程先执行一个块



同步块

- 典型用法：
  - 创建新块
  - 使用延迟锁定来锁定顶点缓存
  - 创建 N 个任务
  - 设置渲染状态
  - 发送绘制调用
- 注意，这里的次序并不重要，任务总会在渲染开始之前完成执行。

11、性能分析和引擎统计数据

性能分析和引擎统计数据 – 概览

- 实时性能监测
- 情景性能测试
- 浸泡测试（Soak testing）
- 监视器（Watcher）

实时性能监测

- 实时性能分析控制台
  - 在游戏中使用 DEBUG + F5 打开
  - 显示最后几帧平均而得的各种统计数据
    - 帧频
    - 图元数量
    - 展开性能监视器层次
- 性能监测器（Dog Watcher）
  - 为一段代码命名以便计时
  - 可以在代码的不同部分进行层次化的性能分析

性能分析

- 摄像机在一定帧中按照指定的路径移动
- 把每帧的数据都输出到一个 CSV 表格中以便分析
- 可以在“历史（profiler history）” 模式下运行
  - 和标准的性能分析类似，只是游戏保持可交互状态
  - 可以被用来测试那些无法自动完成的用例

浸泡测试

- 浸泡测试
  - 在一定时间内不停移动摄像机
  - 不限制帧频 – 在测试完毕后会提供平均帧频
  - 可以被用来测试游戏是否可以运行很长时间
  - 每 6 秒钟输出 MemTracker 状态
  - 会输出一个和性能分析测试中一样的 CSV 文件，但是也会包含内存数据

观察器（watcher）

- 可以在观察器控制台（DEBUG + F7）中通过观察器来监视引擎内部的变量
  - 譬如说，网络子系统的统计信息是以观察器的方式暴露的
  - 按照类别、层次来保存
  - 使用 PGUP 和 PGDN 在观察器之间移动
- 观察器既可以是只读的，也可以是可读写的
  - 修改某些观察器的值会改变引擎的行为
  - 使我们可以访问一些调试功能
    - 譬如说，显示骨骼、门户等
- 可以通过 `BigWorld.setWatcher` 和 `BigWorld.getWatcher` 在 Python 中访问
- 在 Consumer Release build 不可用

总结

- 客户端培训到此结束
- 可以在 Client Programming Guide 和 Client Python API 文档中得到更细节的信息
- 感谢您的参加