**实 验 一 报 告**

**（2020-2021学年第二学期）**

**3D游戏引擎架构设计基础**

**（Foundations of 3D Game Engine Architecture Design）**

|  |  |  |
| --- | --- | --- |
| 学生姓名：胡嘉鸿 | 学号：201930380501 | 年级，班级：2019级1班 |
| 成绩： | 任课教师签名： | 日期：2022.3.17 |
| 实验题目：OGRE或Panda3D开源游戏引擎场景、渲染、资源等核心类的分析 | | |
| 目录：   1. 实验内容描述: 引擎名称和版本,将分析的引擎模块名称 2. 将分析的引擎模块的主要功能描述 3. 将分析的引擎模块的主要类和类关系描述 4. 类的数据成员和成员函数描述 5. 总结 | | |
| 实验报告：   1. 主要分析的是场景组织体系模块 2. 该模块主要负责管理出现在屏幕上的所有物体的承载节点，保持对这些节点的跟踪。 3. 本次主要分析该模块的Node、SceneNode、Bone这三个类。   这三个类的关系如下图所示：    Node类为场景图的公共基类，派生两个场景节点类：普通场景节点类（SceneNode）和角色骨骼节点类（Bone）。  4、  Node类：  数据成员：   |  |  | | --- | --- | | Vector3 mInitialPosition; | 作为关键帧动画的基础的位置 | | Quaternion mInitialOrientation; | 用于作为关键帧动画的基础的方向 | | Vector3 mInitialScale; | 作为关键帧动画的基础使用的缩放 | | Listener\* mListener; | 节点监听器-只有一个允许(没有列表)的大小和性能原因。 | | Node\* mParent; | 父节点的指针 | | ChildNodeMap mChildren; | 指向direct子指针的集合 | | ChildUpdateSet mChildrenToUpdate; | 要更新的子列表，如果self不是out of date，但children是 | | String mName; | 该节点的名称 |   主要成员函数：   * 构造函数   Node::Node(const String& name)  构造函数产生一个string类型的节点名称，并且初始化节点变量   * 跟子节点有关的函数   Node\* Node::createChild(const Vector3& inTranslate, const Quaternion& inRotate)  Node\* Node::createChild(const String& name, const Vector3& inTranslate, const Quaternion& inRotate)  void Node::addChild(Node\* child)  Node\* Node::getChild(unsigned short index) const  Node\* Node::removeChild(unsigned short index)  Node\* Node::removeChild(Node\* child)  void Node::removeAllChildren(void)  Node\* Node::getChild(const String& name) const  Node\* Node::removeChild(const String& name)  Node::ChildNodeIterator Node::getChildIterator(void)  Node::ConstChildNodeIterator Node::getChildIterator(void) const   * 和变换有关的函数   void Node::setScale(const Vector3& inScale)  void Node::scale(const Vector3& inScale)  Matrix3 Node::getLocalAxes(void) const  void Node::translate(const Vector3& d, TransformSpace relativeTo)  void Node::translate(const Matrix3& axes, const Vector3& move, TransformSpace relativeTo)  void Node::roll(const Radian& angle, TransformSpace relativeTo)  void Node::pitch(const Radian& angle, TransformSpace relativeTo)  void Node::yaw(const Radian& angle, TransformSpace relativeTo)  void Node::rotate(const Vector3& axis, const Radian& angle, TransformSpace relativeTo)  void Node::rotate(const Quaternion& q, TransformSpace relativeTo)  SceneNode类：  数据成员：   |  |  | | --- | --- | | Vector3 mAutoTrackOffset; | 跟踪偏移 | | Vector3 mAutoTrackLocalDirection; | 局部的“法线”方向向量 | | Vector3 mYawFixedAxis; | 固定的轴周围偏航 | | bool mYawFixed : 1; | 是否绕固定轴偏转。 | | bool mIsInSceneGraph : 1; | 这个节点是场景图的当前部分吗? | | bool mShowBoundingBox : 1; | 决定是否显示节点的边界框的标志 |   主要成员函数：   * 构造函数   SceneNode::SceneNode(SceneManager\* creator) SceneNode::SceneNode(SceneManager\* creator, const String& name)  //构造函数，仅由创建者SceneManager调用。创建具有生成名称的节点或者特定名称的节点。   * 跟object有关的函数   //AttachObject函数将场景对象的实例添加到此节点。场景对象可以包括实体对象，相机对象，灯光对象，粒子系统对象等。  void SceneNode::attachObject(MovableObject\* obj)  //获取附加到此节点的对象数。  unsigned short SceneNode::numAttachedObjects(void) const  //获取指定对象  MovableObject\* SceneNode::getAttachedObject(unsigned short index)  MovableObject\* SceneNode::getAttachedObject(const String& name)  //分离指定对象  MovableObject\* SceneNode::detachObject(unsigned short index)  MovableObject\* SceneNode::detachObject(const String& name)  void SceneNode::detachObject(MovableObject\* obj)  void SceneNode::detachAllObjects(void)   * 和子节点有关的函数   //创建子节点  Node\* SceneNode::createChildImpl(void)  Node\* SceneNode::createChildImpl(const String& name)  //删除子节点  void SceneNode::removeAndDestroyChild(const String& name)  void SceneNode::removeAndDestroyChild(unsigned short index)  void SceneNode::removeAndDestroyChild(SceneNode\* child)  void SceneNode::removeAndDestroyAllChildren(void)  SceneNode\* SceneNode::createChildSceneNode(const Vector3& inTranslate,  const Quaternion& inRotate)  SceneNode\* SceneNode::createChildSceneNode(const String& name, const Vector3& inTranslate,  const Quaternion& inRotate)   * 跟操作相关的函数   void SceneNode::setDirection(Real x, Real y, Real z, TransformSpace relativeTo,  const Vector3& localDirectionVector)  void SceneNode::lookAt( const Vector3& targetPoint, TransformSpace relativeTo,  const Vector3& localDirectionVector)  void SceneNode::\_autoTrack(void)  SceneNode\* SceneNode::getParentSceneNode(void) const  //使得悬挂在这个节点上的对象变得可见/不可见，第一个参数是设置可见性，第二个参数是设置是否级联到子节点。  void SceneNode::setVisible(bool visible, bool cascade)  //反转所有可见性  void SceneNode::flipVisibility(bool cascade)  Bone类：  数据成员：   |  |  | | --- | --- | | Skeleton\* mCreator; | 返回创建者的指针，用于创建子对象(不是智能ptr，因此子对象不保留父对象 | | Vector3 mBindDerivedInverseScale; | 绑定姿势时骨的反向衍生尺度 | | Quaternion mBindDerivedInverseOrientation; | 在绑定姿势时骨骼的反向派生方向 | | Vector3 mBindDerivedInversePosition; | 绑定姿势时骨骼的反向派生位置 |   主要成员函数：   * 构造函数   Bone::Bone(unsigned short handle, Skeleton\* creator) Bone::Bone(const String& name, unsigned short handle, Skeleton\* creator)   * 设置骨骼方面的相关函数   //将当前位置/方向设置为“绑定姿势”，即骨骼最初绑定到网格的布局。  void Bone::setBindingPose(void)  //重置为默认位置/方向  void Bone::reset(void)  //设置是否手动控制骨骼  void Bone::setManuallyControlled(bool manuallyControlled)  bool Bone::isManuallyControlled() const  //获取此骨骼的控制句柄  unsigned short Bone::getHandle(void) const   * 跟创建子骨骼方面相关的函数   //创建新骨骼作为该骨骼的子骨骼  Bone\* Bone::createChild(unsigned short handle, const Vector3& inTranslate, const Quaternion& inRotate)  Node\* Bone::createChildImpl(void)  Node\* Bone::createChildImpl(const String& name)  5、总结：  本次实验我主要分析了Node，SceneNode，Bone这三个类之间的关系以及这三个类有什么样的功能函数，了解了ogre的对于场景的设计思想：将实体与挂载节点分离开来，减少内存的消耗，受益颇多。  要求：  1. 尽可能用图、表等设计工具描述代码，包括类关系图、时序图、状态转换图、流程图、功能模块图等；  2. 描述类功能时，不要简单列出类的成员函数和解释，要按功能类别分析和描述；  3. 最好能解释系统这样设计的原因，如：为什么这样设计，为什么用这样的设计模式等。  总之，想象你是开发团队的一份子，你的报告是要让其他开发成员看懂你的设计思想，而不是读懂你的代码。 | | |