# 整体架构

1. 整个程序由main函数中运行一个App的run函数来启动，这个App相当于是整个渲染器的入口和整合工厂
2. 渲染器的功能由组件类(FzbComponent)来实现，如前向渲染、光线追踪、SVO划分；同时主组件（FzbMainComponent）负责维护一些全局变量，如设备，交换链
3. 对于vulkan渲染的重要部分，都由一个容器类来维护，如buffer、image。由负责具体功能的组件创建、调度和销毁。

## 组件FzbComponent

### 组件FzbComponent

1. 组件是实现具体功能的父类，一个具体功能类需要继承FzbComponent类
2. 组件类包括一些需要用到的全局变量和一些功能函数
3. 全局变量（无法创建，则能由构造函数传入）
   1. 设备、队列
   2. 交换链
   3. 命令缓冲池
4. 自己维护的变量
   1. 命令缓冲区
   2. 描述符池
5. 功能函数
   1. 封装的创建各种缓冲区函数
   2. 封装的创建描述符函数
   3. 创建信号量和栏栅函数
6. 所有继承组件类的功能类，需要注意
   1. 需要一个功能设置结构体，xxxSetting，通过MainComponent告诉组件需要做什么。
   2. 需要实现一个addExtensions函数，这个函数主要是为MainComponent创建设备前添加功能所需要的扩展和设备功能
   3. 需要init函数来构造功能类，而不是构造函数。因为addExtensions在主组件创造设备前，因此我们需要先创造组件类通过addExtensions函数为主组件创建设备后，才能初始化
   4. 需要activate函数，这个是调用功能的接口
   5. 可以通过presentPrepare和present实现渲染，主要是用来debug的，如SVO后查看划分结果
   6. 需要一个clean函数，用来清理组件维护的资源（全局变量由主组件清理）

### 主组件FzbMainComponent

1. 整个程序只能有一个主组件，即App
2. 主组件维护的全局变量，包括
   1. 窗口和回调函数（鼠标、键盘）
   2. 相机
   3. 表面surface
   4. 设备
   5. 交换链
   6. 实现具体功能的子组件
3. 主组件的run函数，由main函数调用，执行整个程序
   1. 初始化相机
   2. 初始化窗口initWindow
   3. 初始化vulkan initVulkan
   4. 渲染循环 mainLoop
   5. 清理 clean

#### 初始化相机

其实就是设置相机的各种属性，如位置、fov、近远平面，只需要调用相机类的构造函数即可

#### 初始化窗口

1. 初始化glfw库，设置glfw窗口的设置，如是否禁止改变窗口大小
2. 创建glfw窗口，设置大小和名字
3. 将当前调用的mainComponent绑定到窗口上，并设置回调函数

#### 初始化vulkan

##### 初始化设置

1. 创造组件
   1. 根据每个组件的setting，决定是否创建组件
   2. 创建需要的组件，并通过组件的addExtensions函数，初始化mainComponent的实例扩展、设备扩展和设备功能
2. 初始化设置：根据实例扩展和应用层（根据是否是debug模式决定）创建vulkan实例，可以决定vulkan的版本等
3. 创建debug信使：将debug的信息传递出来
4. 创建表面surface
5. 创建设备
   1. 根据设备扩展和设备功能，从所有的GPU中挑选出最优的那个
   2. 并创建相应的逻辑设备，开启相应的设备功能
   3. 创建相应的队列
6. 创建交换链
   1. 根据表面surface获取各种设置（色彩空间、精度等）；获取展示模式（一般是三缓冲）和范围。
   2. 创建交换链，将之绑定到表面上
7. 初始化缓冲
   1. 创建命令缓冲池
   2. 创建命令缓冲（传入数量）

##### 创建场景

从指定地址加载模型、材质以及各种资源，目前有3种方式

1. 直接通过函数得到mesh，这适合像球、三角形等简单mesh
2. 给定一个obj文件的地址，直接加载mesh以及它所需的材质（给出mtl可得到纹理）
3. 给定一个目录，从该目录下的scene.xml文件中加载所有资源

##### 激活组件

调用组件的init和activate函数，激活组件

##### 创建渲染前的准备资源

1. 主要是创建相应image，如深度缓冲，renderPass，信号量
2. 如果组件需要present，则调用组件的presentPrepare函数

#### 渲染循环mainLoop

## 整体流程

### 场景创建（读取XML）

1. 创建scene类，然后调用addSceneFromMitsubaXML(path)函数，读取指定路径的xml文件
2. addSceneFromMitsubaXML会读取所有的material，其所需的资源不会立刻读取，但会保留资源地址和类型等，如纹理地址，过滤方式。
3. 根据material的类型，添加相应的shader。然后根据material所需的资源和shader所有的资源类型，开启相应的宏。
4. 根据material的顶点格式，加载mesh的顶点属性。

### 创建shader

1. 在为material添加shader时，需要创建shader
2. Shader同样是通过一个xml文件读取的，但是不需要用户指定（当然也可以自己指定），可以获得该shader的各种需要的资源（所需纹理，如albedoMap、normalMap；数值属性，如roughness）、所需开启的宏以及pipeline相关的信息（如背面剔除，深度测试），**其实就是按照unity的shader来的**
3. 在material调用addShader后，会和shader进行信息同步，保证shader只会开启有实际资源的宏，如shader一开始启动normalMap，但是material没有传入normlaMap，则会关闭。其实就是material和shader资源的并集。

### 场景创建资源

1. 遍历所有的material，创建其所需要的各种纹理和缓冲区
2. 压缩mesh的顶点，去掉冗余顶点。
3. 遍历所有的mesh，创建其所需的各种缓冲区（mesh特有数据和顶点缓冲区，索引缓冲不由场景创建）

### 创建renderPass

1. 根据所需的subPass创建FzbSubPass类，subPass会根据shader创建meshBatch和相应的pipeline

# Shader

## 注意事项

1. 每个shader的第一行必须是版本号，如#version 450

## 变量命名规范

1. 所有的纹理都以Map结尾，如albedoMap，normalMap
2. 所有的宏，按照use+变量名命名，变量名首字母大写，如useAlbedoMap，useNormalMap
3. 所有的shader之间传递的变量，按照shader阶段+变量名命名，如vertexWorldPos，geometryWorldPos