自制渲染引擎及在客户端应用

目 录 CONTENTS

为什么要造轮子

为什么要造轮子

UI Kit 不够用么？

业界流行的GUI引擎不够用么？

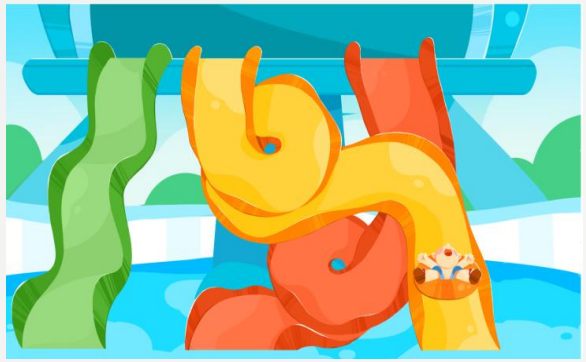
...

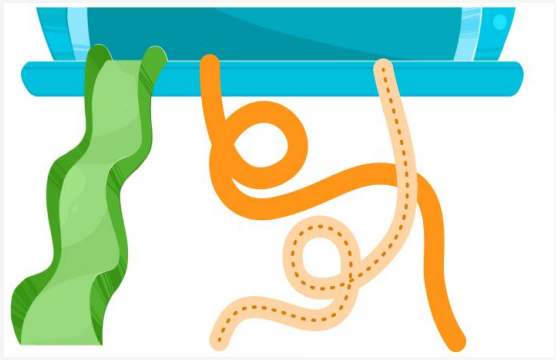
背景

UI 组件难处理的场景

该怎么做呢？

自定义场景绘制

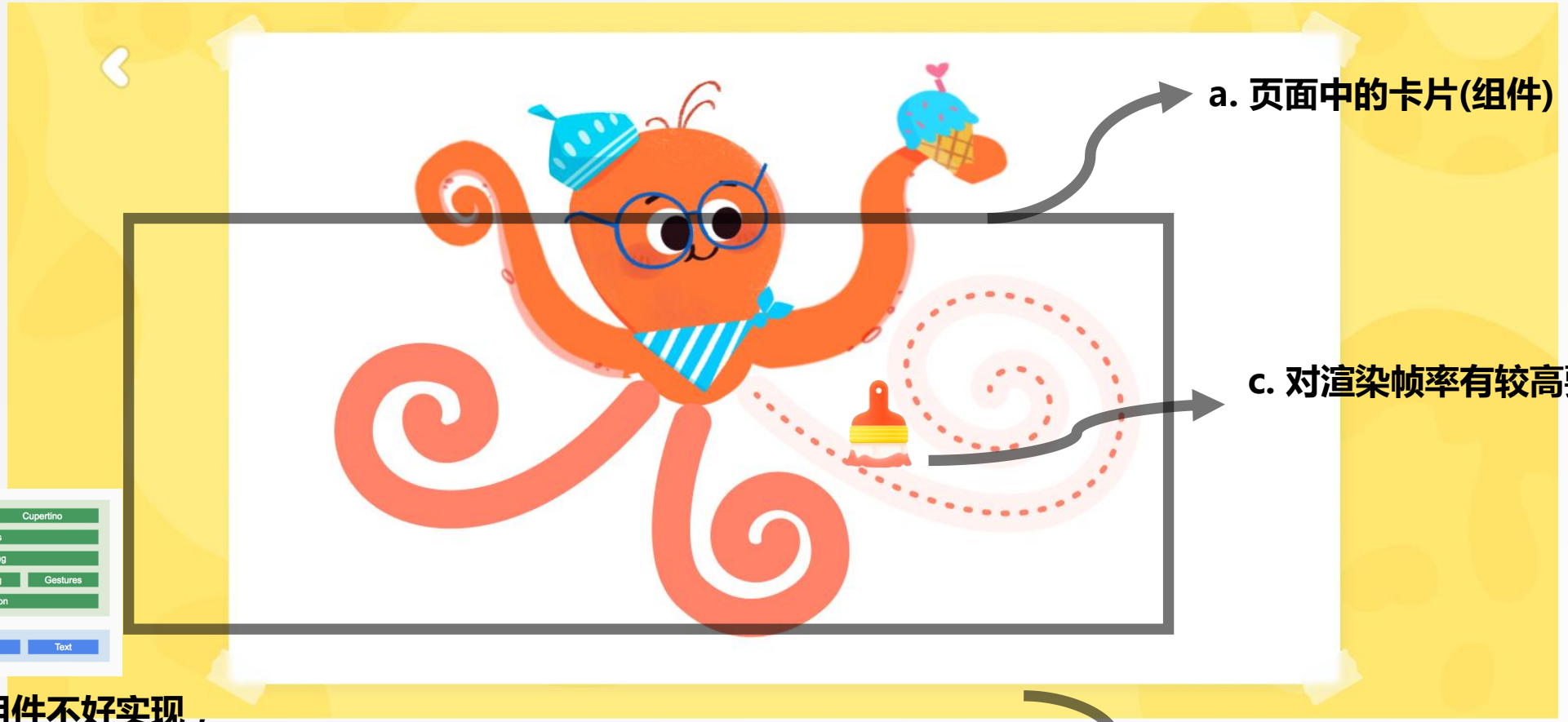




骨骼动画

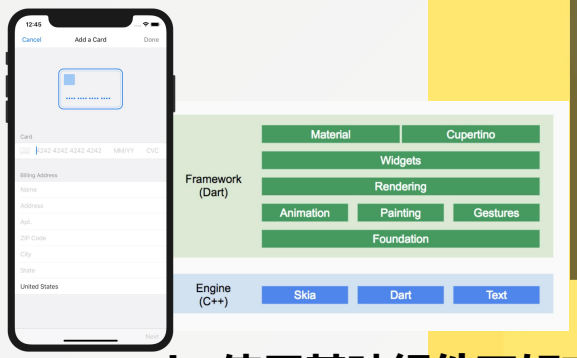
...

在业务场景中的使用方式



a. 页面中的卡片(组件)

c. 对渲染帧率有较高要求



b. 使用基础组件不好实现，
使用渲染引擎又太重

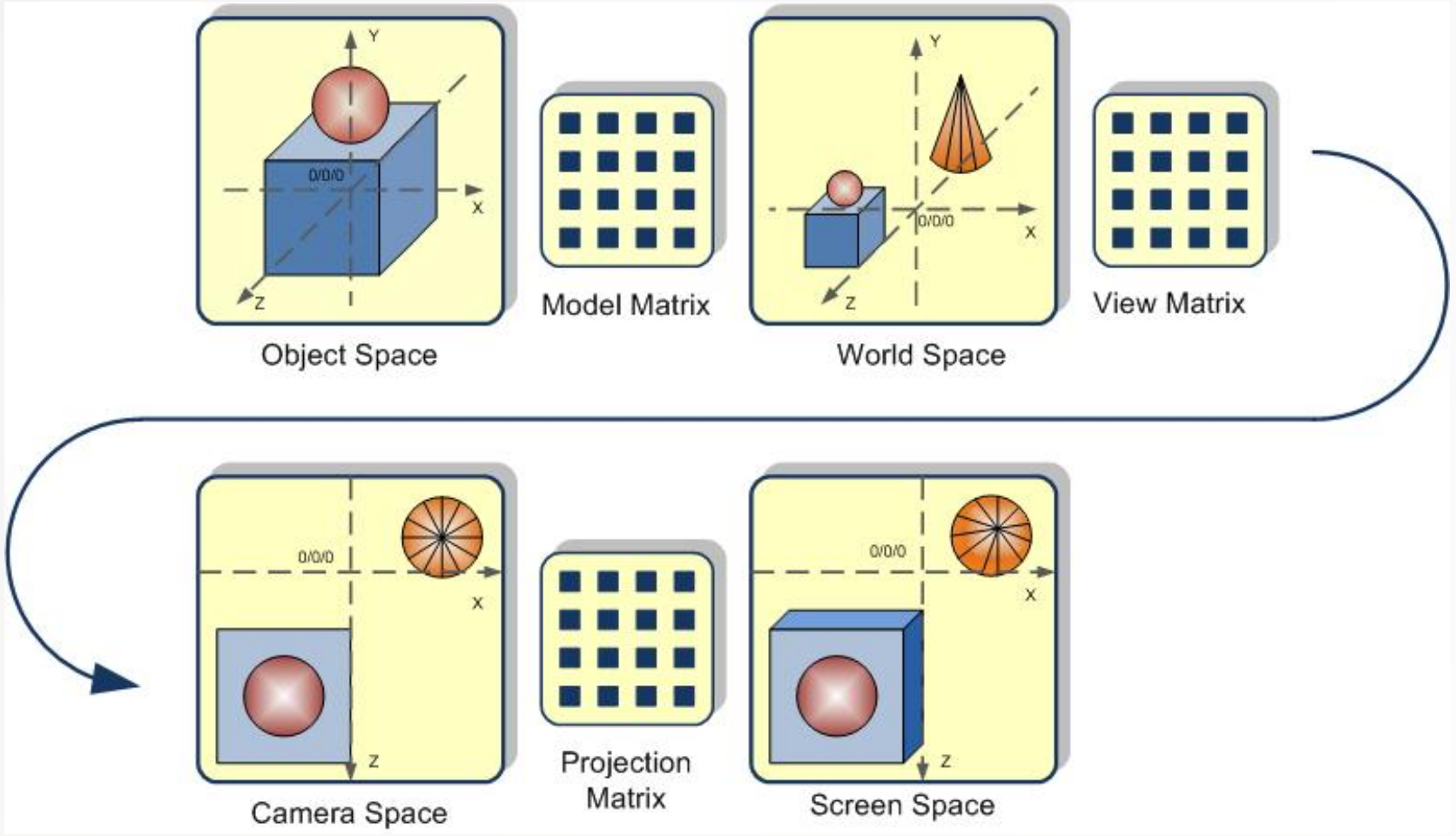
d. 整体业务的一小部分逻辑

Recap

MVP 变换

光栅化

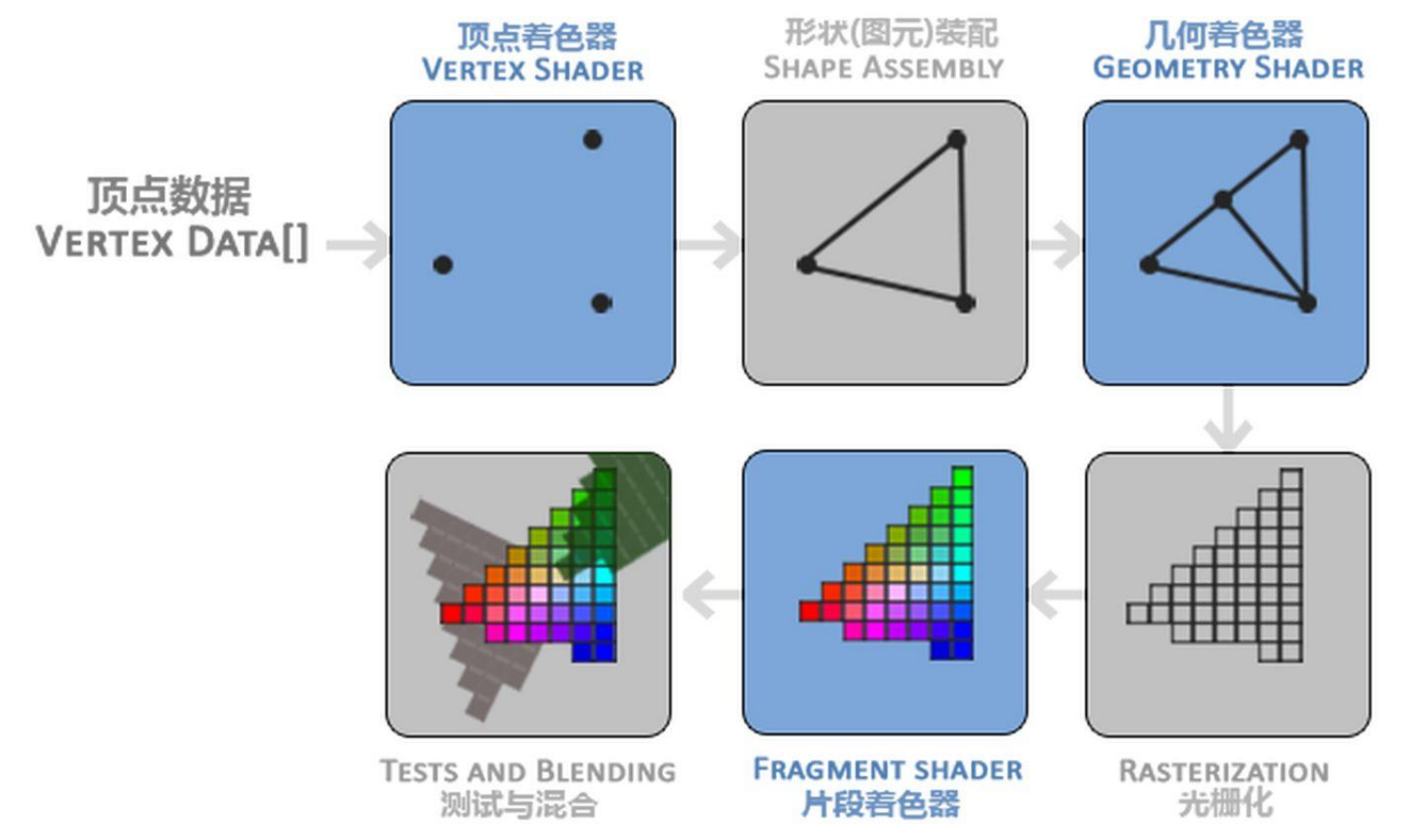
可编程的图形接口



MVP 变换

光栅化

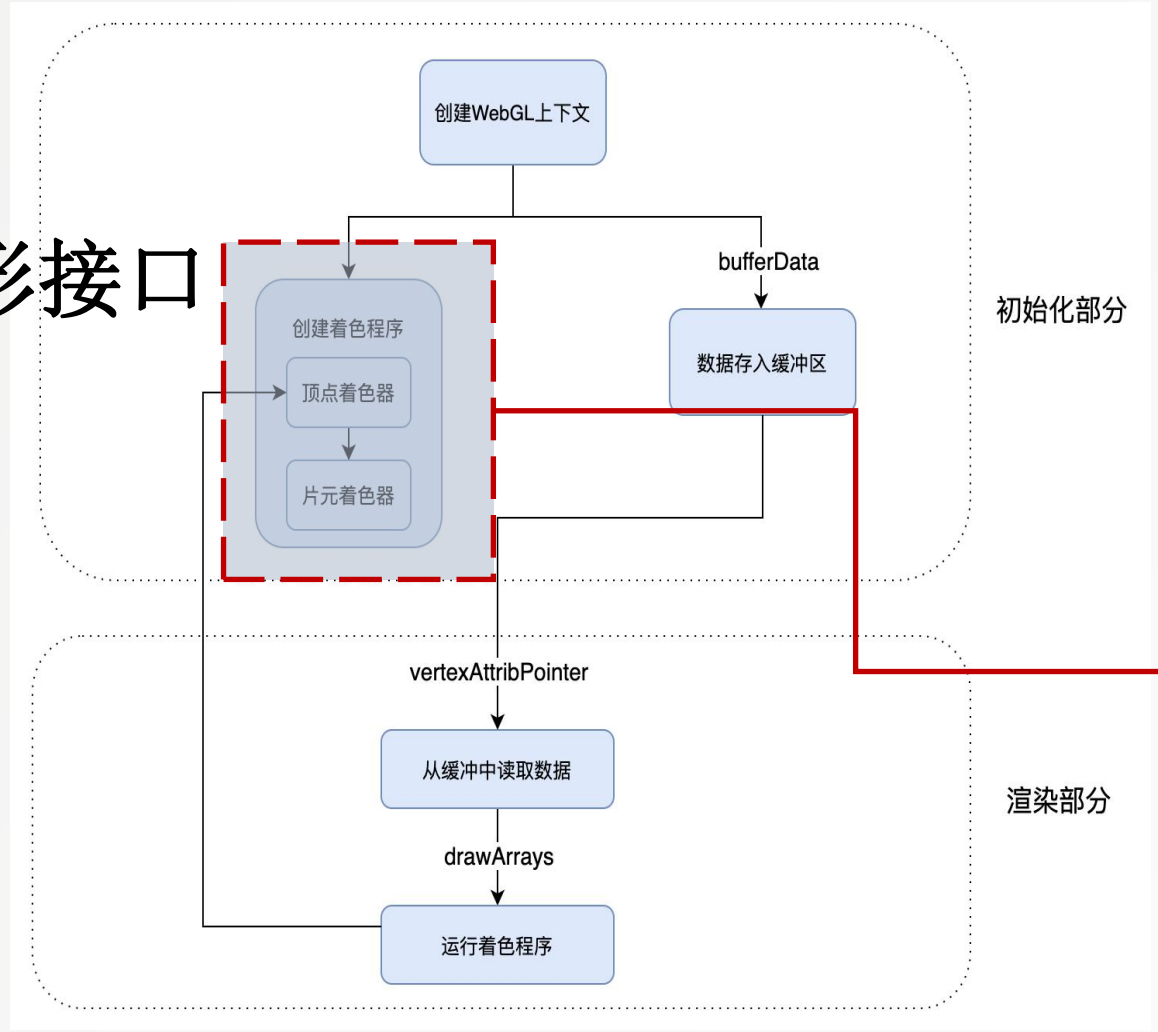
可编程的图形接口

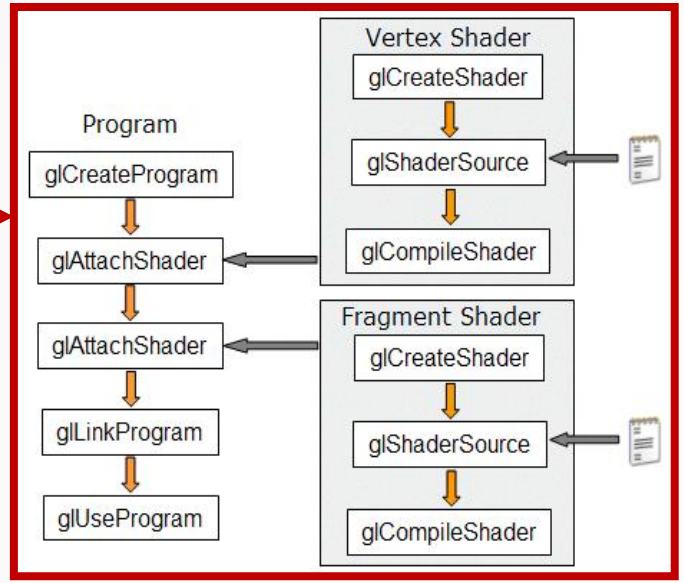


MVP 变换

光栅化

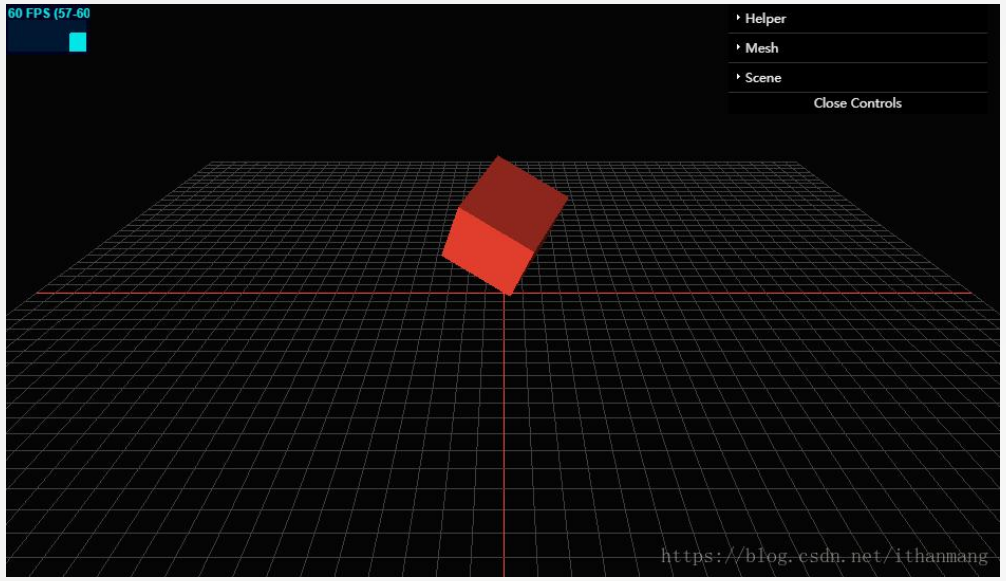
可编程的图形接口





温习渲染相关知识

Api 设计



three.js WebGLRenderer ⇒

1. 构建必要组件：

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.inner

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

2. 创建要渲染的元素：

```
// 创建立方体
var cubeGeometry = new THREE.CubeGeometry(100, 100, 100);// 立方体模型
var cubeMaterial = new THREE.MeshLambertMaterial({color : Math.random() * 0xffffff});//
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);// 创建网格实例
cube.position.y = 90;// 立方体的 y 坐标 +90
// 将立方体加入场景
scene.add(cube);
// 将光源加入场景
scene.add(ambientLight);
scene.add(directionalLight1);
scene.add(directionalLight2);
```

3. 执行渲染：

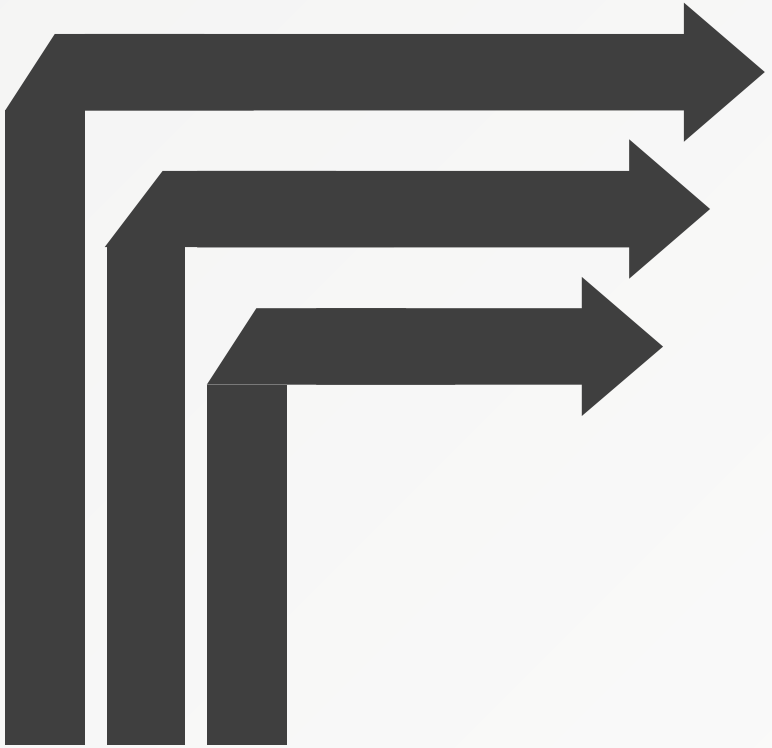
```
var animate = function () {
  requestAnimationFrame( animate );

  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;

  renderer.render( scene, camera );
};

animate();
```

⇒

如何实现 WebGLRenderer

怎么造轮子

实现引擎的目标



高性能与轻量级

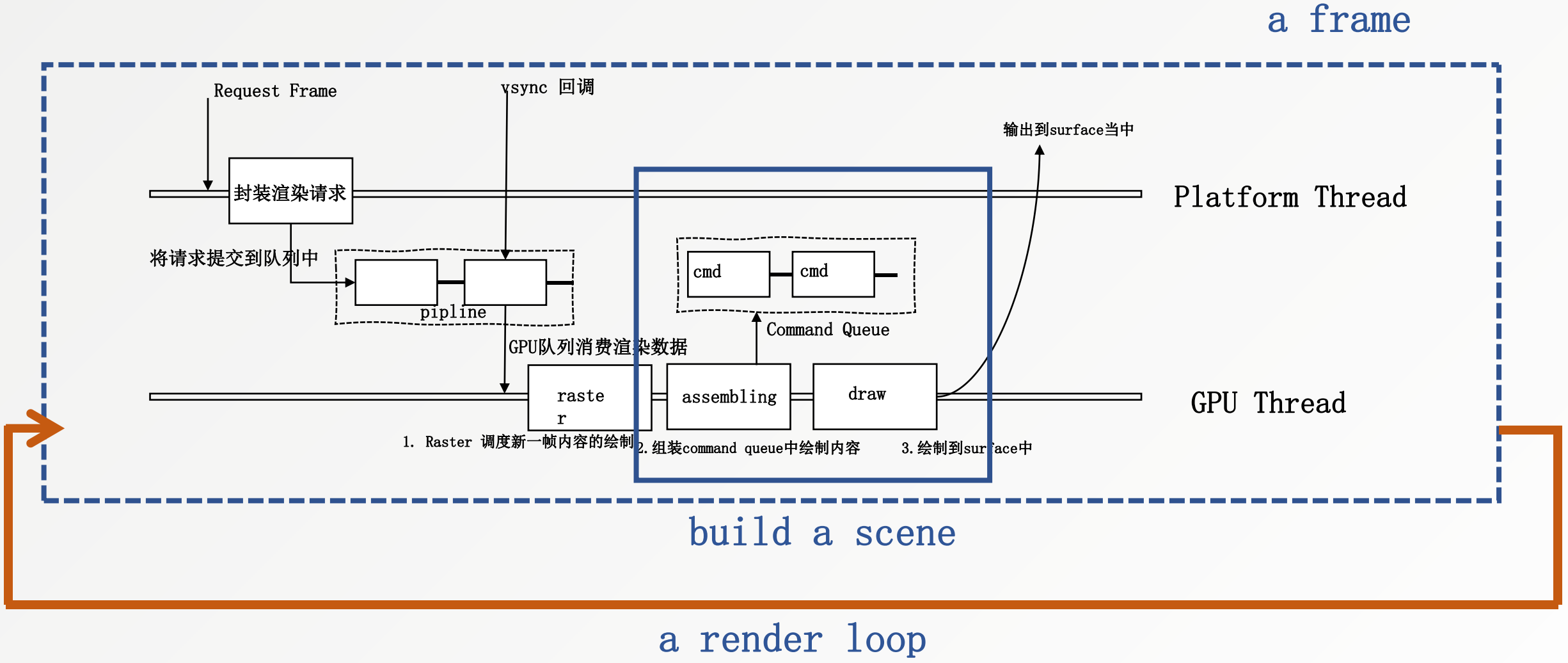
易于集成

方便定制

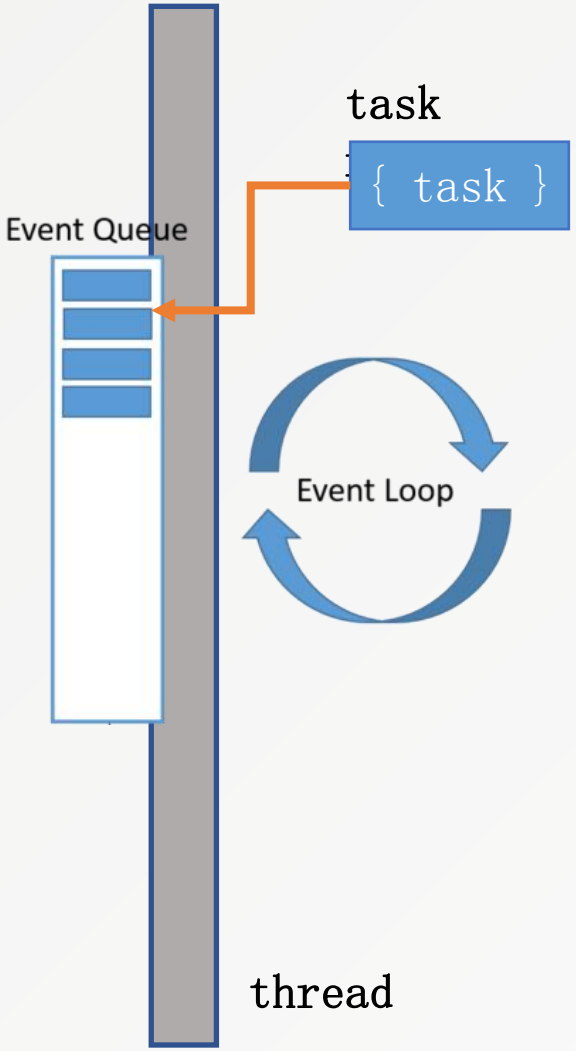
渲染循环

```
while (!quit)
{
    // Update the camera transform based on interactive
    // inputs or by following a predefined path.
    updateCamera();

    // Update positions, orientations and any other
    // relevant visual state of any dynamic elements
    // in the scene.
    updateSceneElements();

    // Render a still frame into an off-screen frame
    // buffer known as the "back buffer".
    renderScene();

    // Swap the back buffer with the front buffer, making
    // the most recently rendered image visible
    // on-screen. (Or, in windowed mode, copy (blit) the
    // back buffer's contents to the front buffer.
    swapBuffers();
}
```

引擎整架构框图


a frame
Request Frame
vsync 回调
输出到surface当中
封装渲染请求
Platform Thread
将请求提交到队列中
pipline
cmd
cmd
Command Queue
GPU队列消费渲染数据
raste
r
assembling
draw
GPU Thread
1. Raster 调度新一帧内容的绘制
2.组装command queue中绘制内容
3.绘制到surface中
build a scene
a render loop

高性能与轻量级



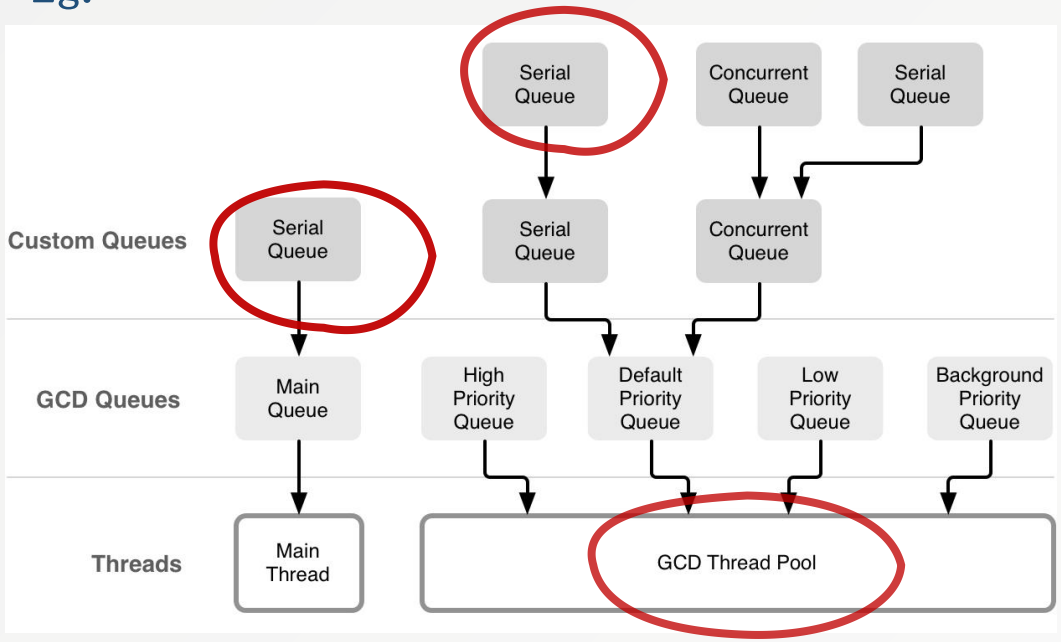
->Thread
->Message(Event) Loop
->Task Runner

各司其职保证高性能

高性能与轻量级

复用系统的
message queue
保证轻量级

Eg.



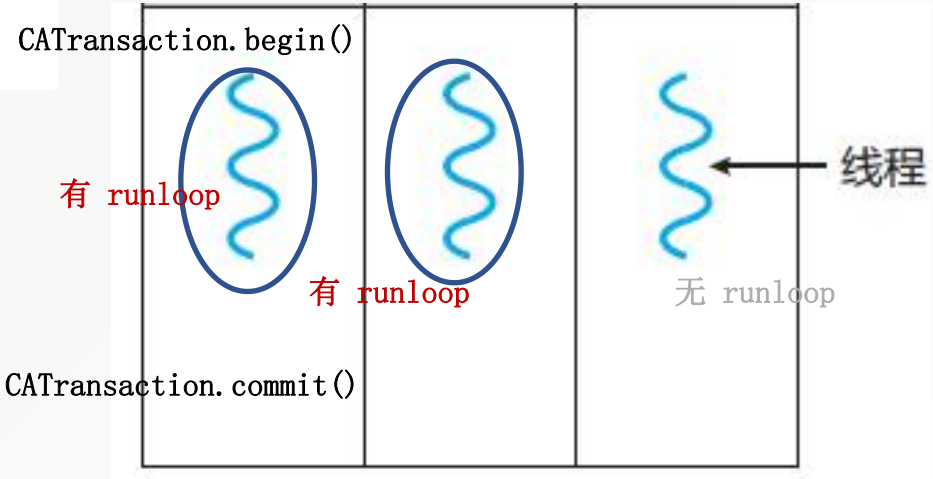
复用main queue 实现负责逻辑的task runner

复用系统线程池,减轻创建线程开销

基于GCD serial Queue创建用于渲染的task runner

多个线程用于渲染存在的问题:

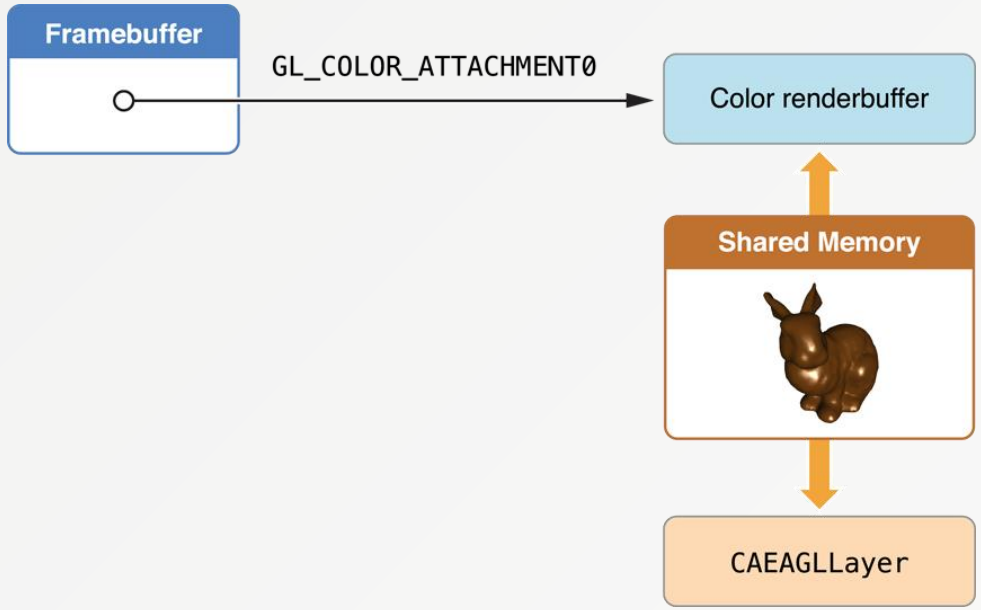
设置layer presentsWithTransaction为 true
会使得渲染无法更新

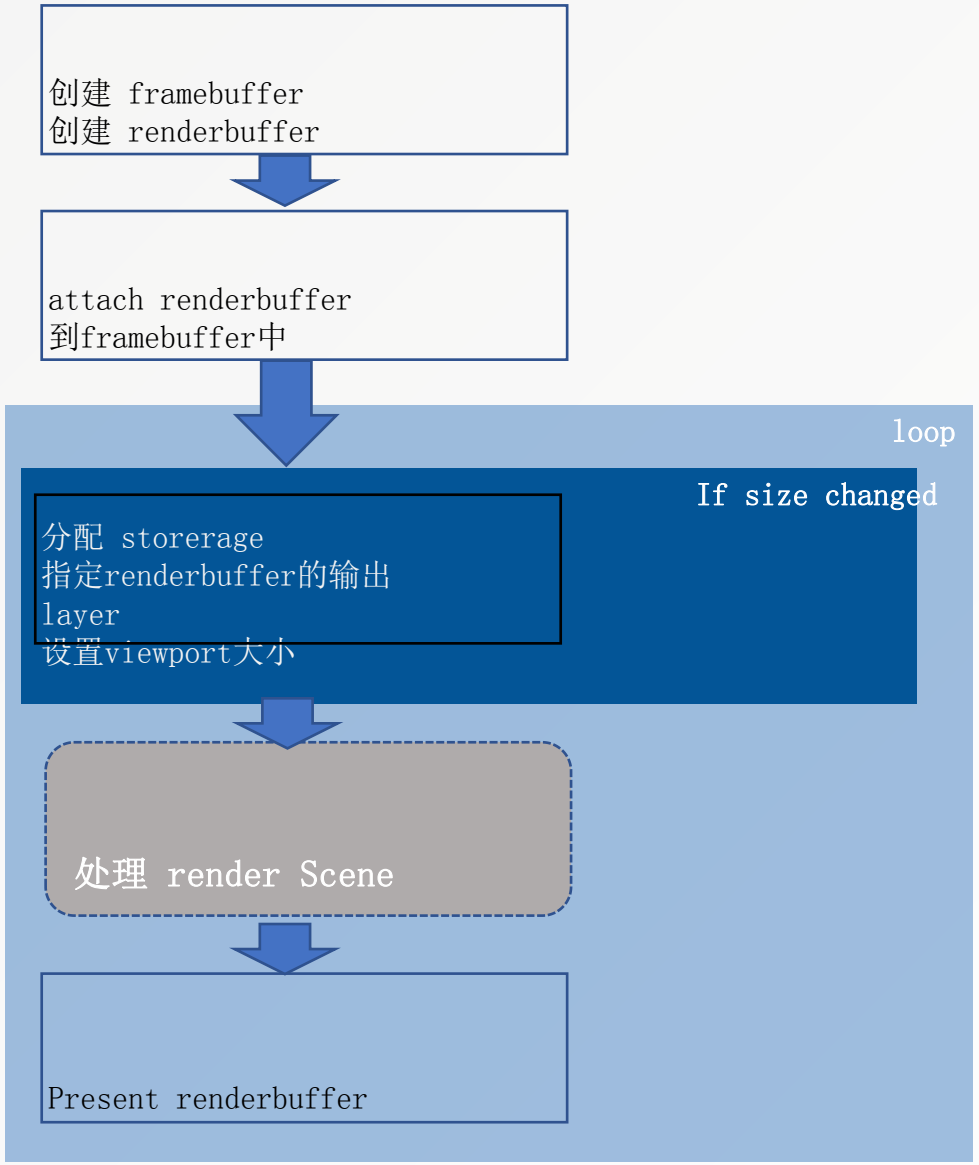


需主动调用！！！ CATransaction.flush()

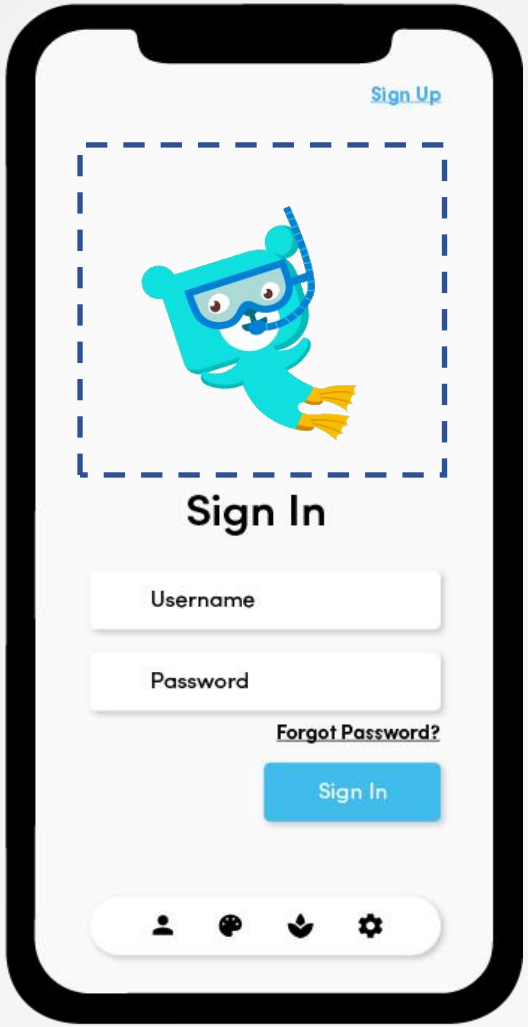
高性能与轻量级

iOS端基于OpenGL 工程上的实现：

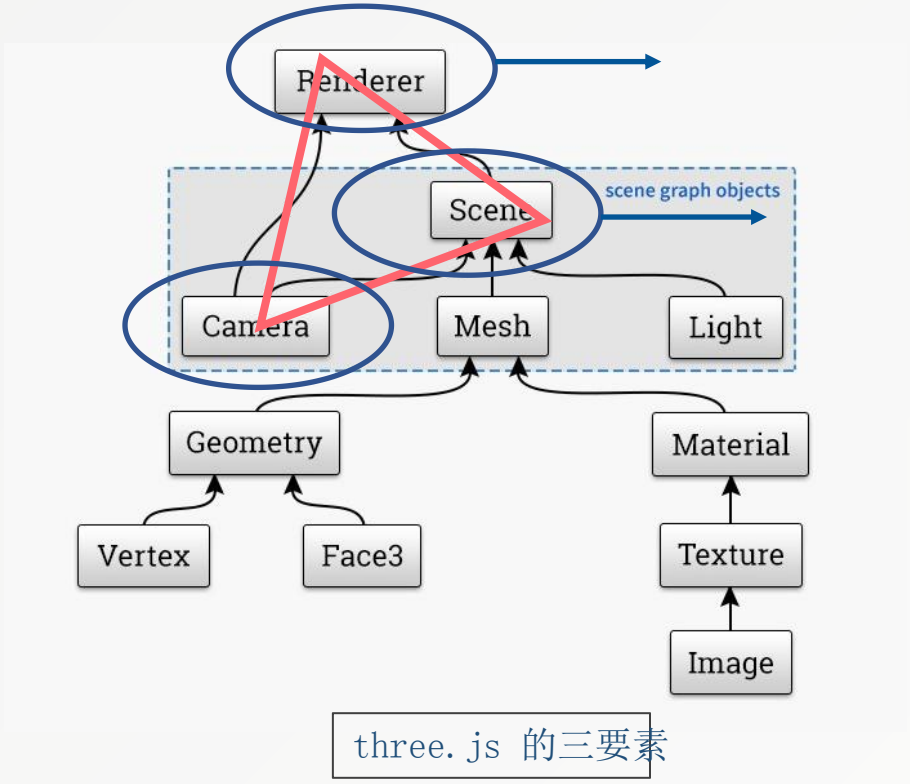




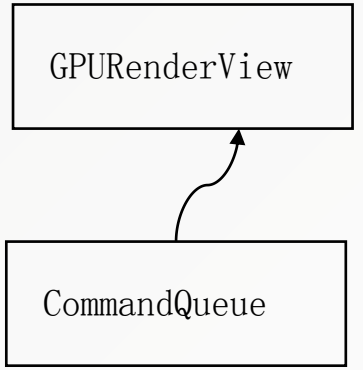
易于集成



组件形态



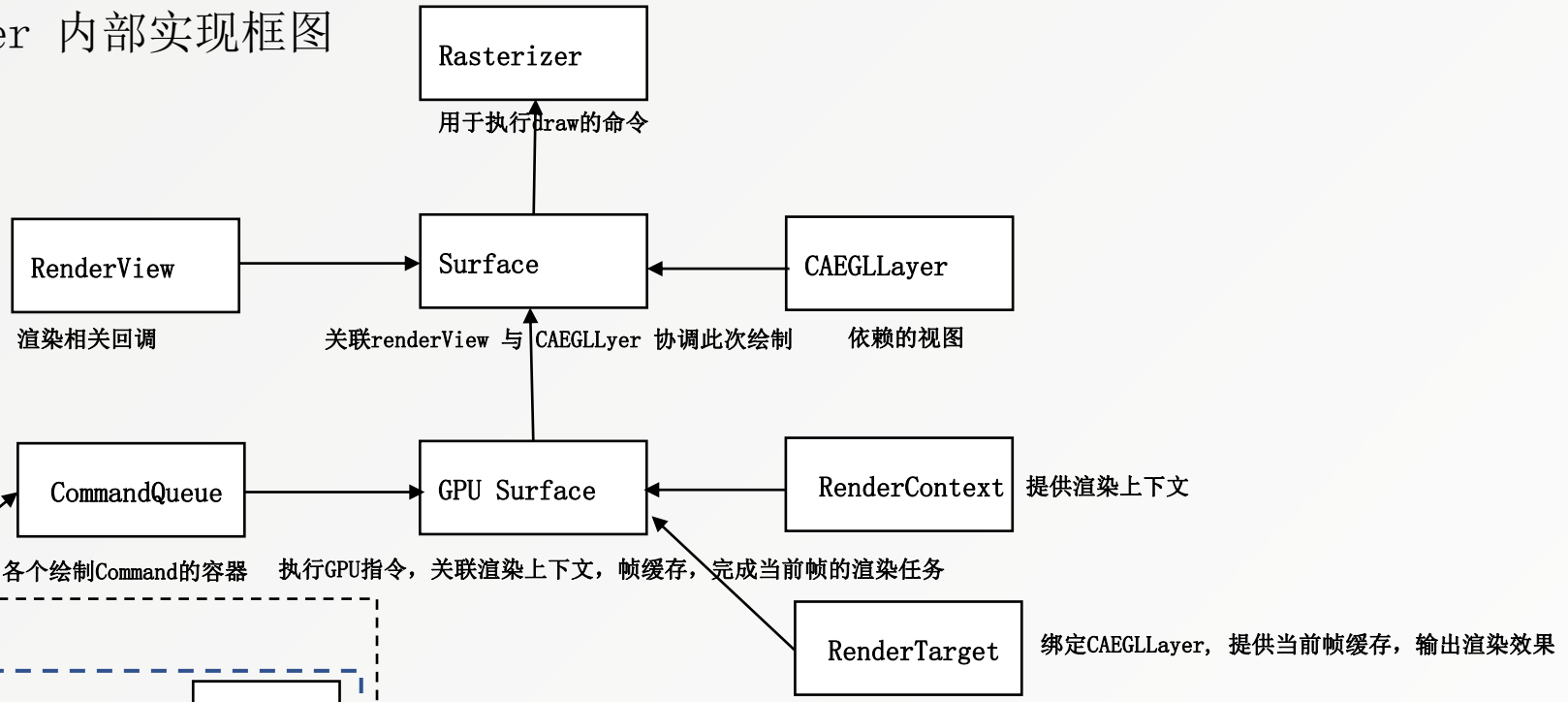
three.js 的三要素

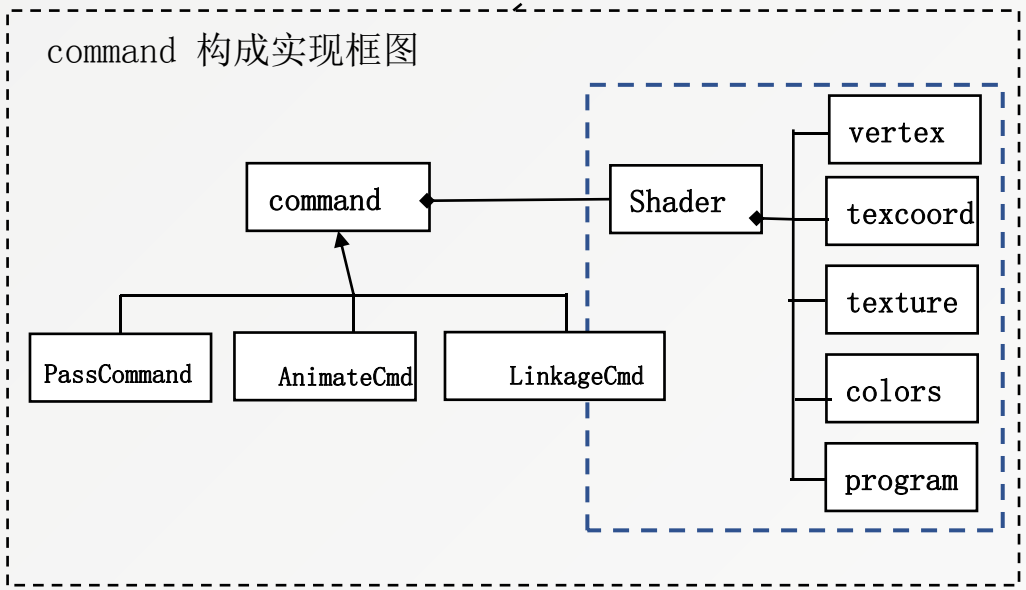


Prek GPU Engine 的二要素

方便定制

Rasterizer 内部实现框图



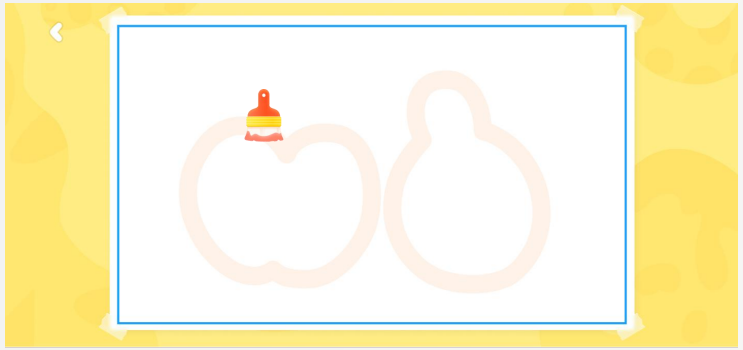


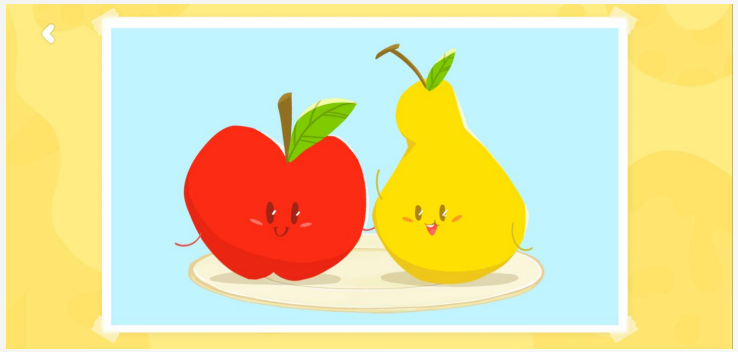
如何用轮子

场景1：　临摹题

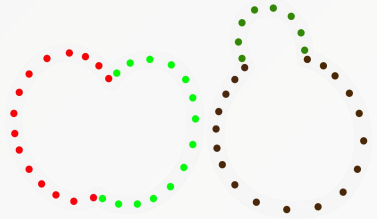
背景

实现下面的场景

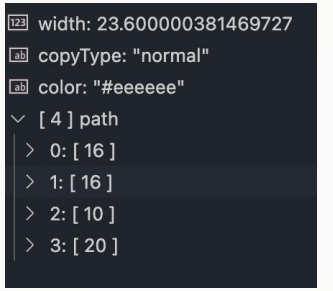




❶ UI给定绘制的采样点

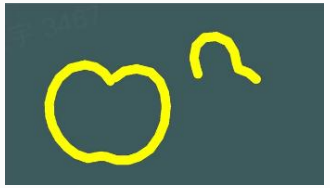


❷ 生成描述文件



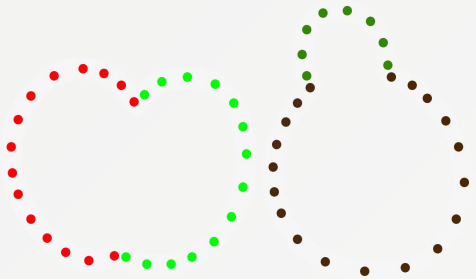
❸ 处理客户端逻辑

❹ 渲染端处理



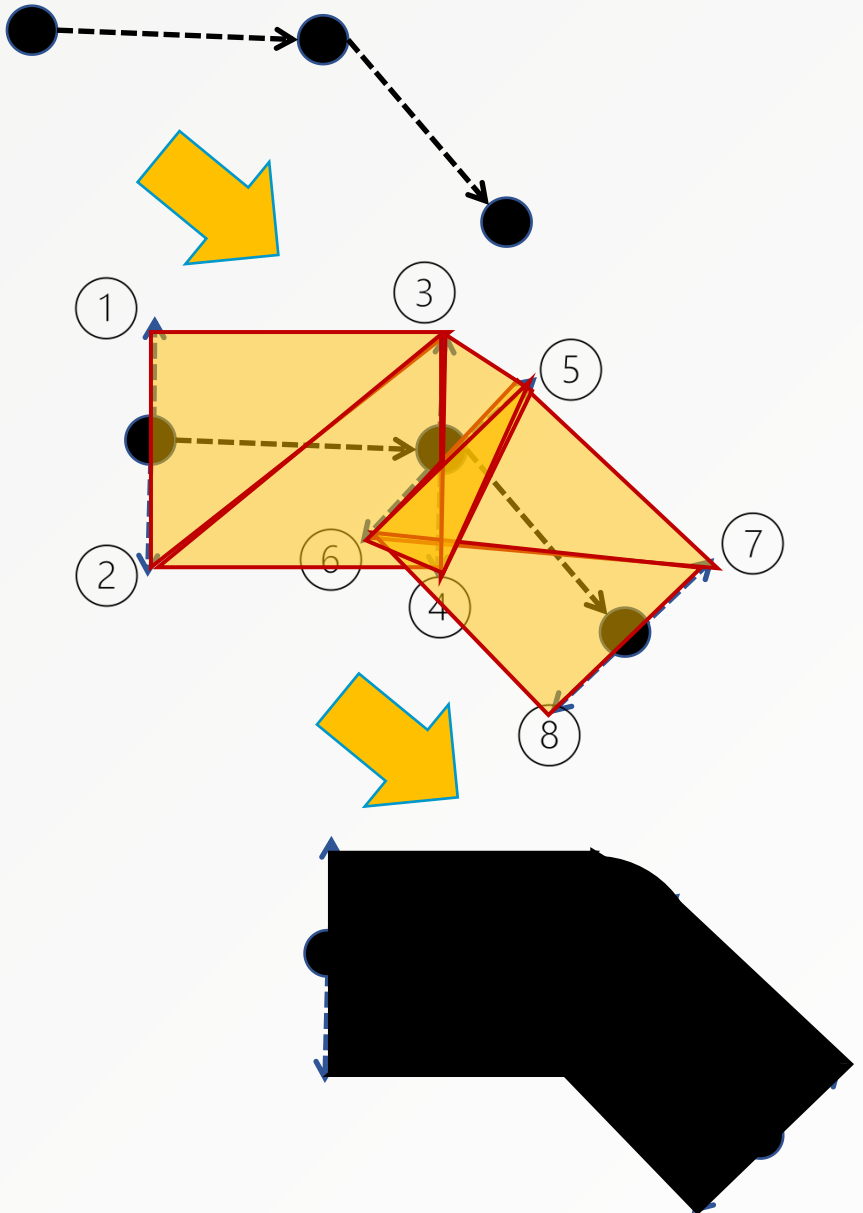
场景1:　临摹题

算法设计



将采样点平滑的连接起来

❶ 解析json文件
❷ 生成顶点属性数组
❸ 编写GL shader
❹ draw call

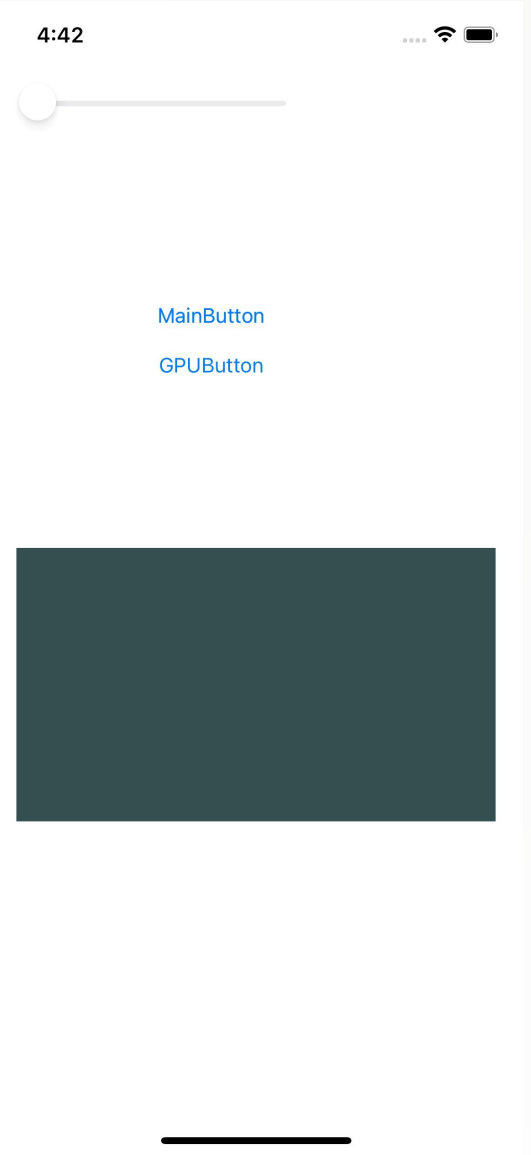


场景1:　临摹题

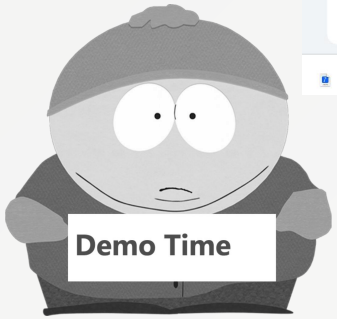
平台实现

生成:

预览:



特别感谢 prek h5 team



场景1:　临摹题

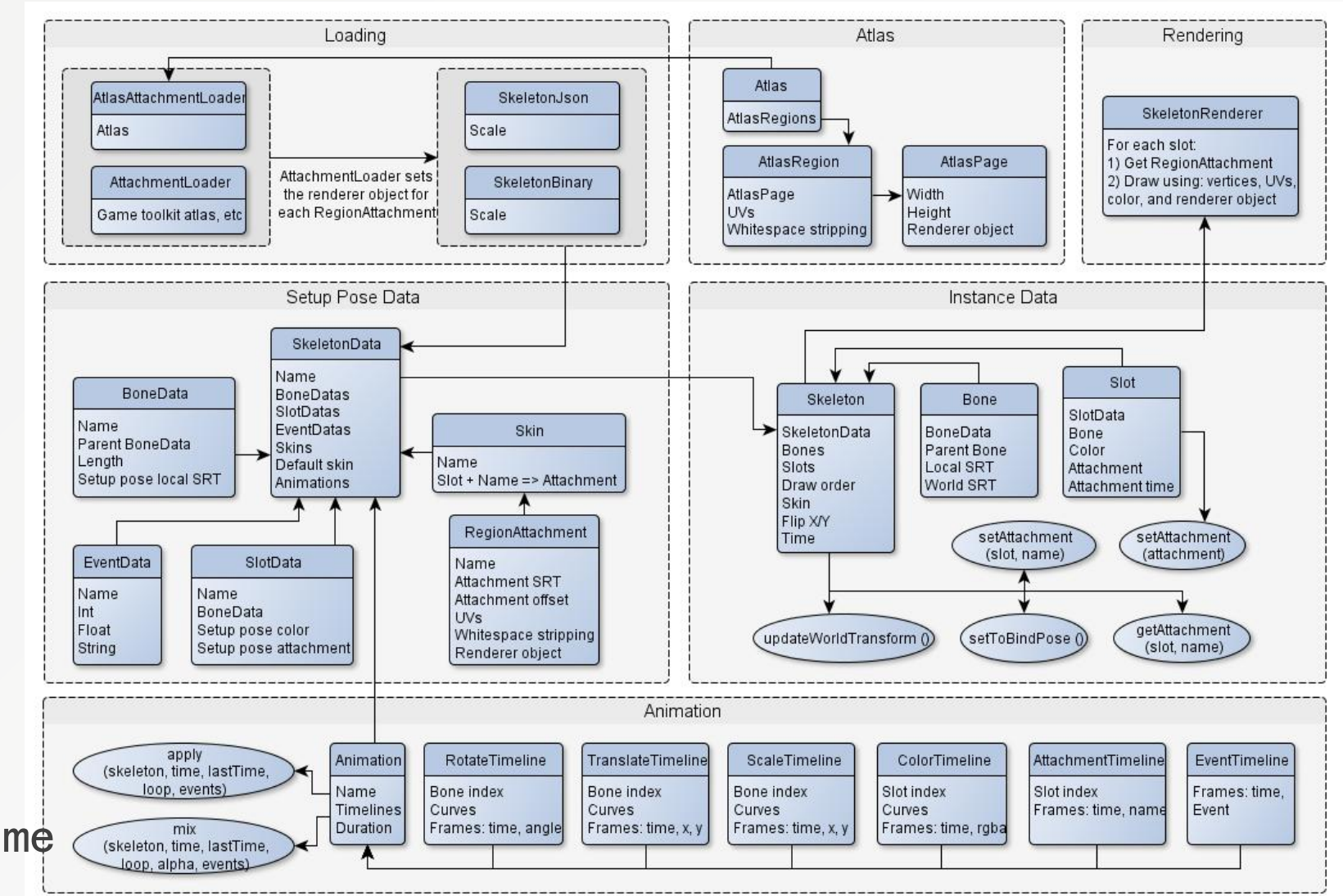
上线效果对比:

标准	优化后帧率	优化前帧率	效果提升
平均线	59.87fps	51.23fps	1.16倍
95线	58.99fps	21.98fps	2.68倍
99线	57.97fps	14.07fps	4.12倍

机型:
1.Ipad mini 3-5;
2.Iphone 6, 6s, 7, 8, x, SE,SE2;
3.Ipad 3, 4, air, 5, 6,air2, air3,a

场景2：spine动画

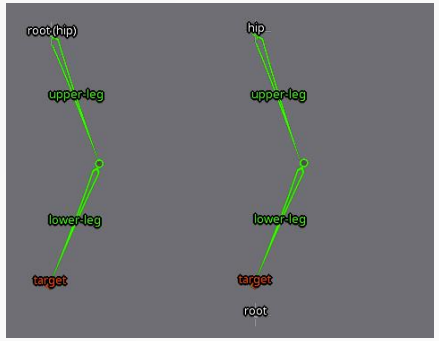
Spine Runtime



场景2：spine动画

atlas:

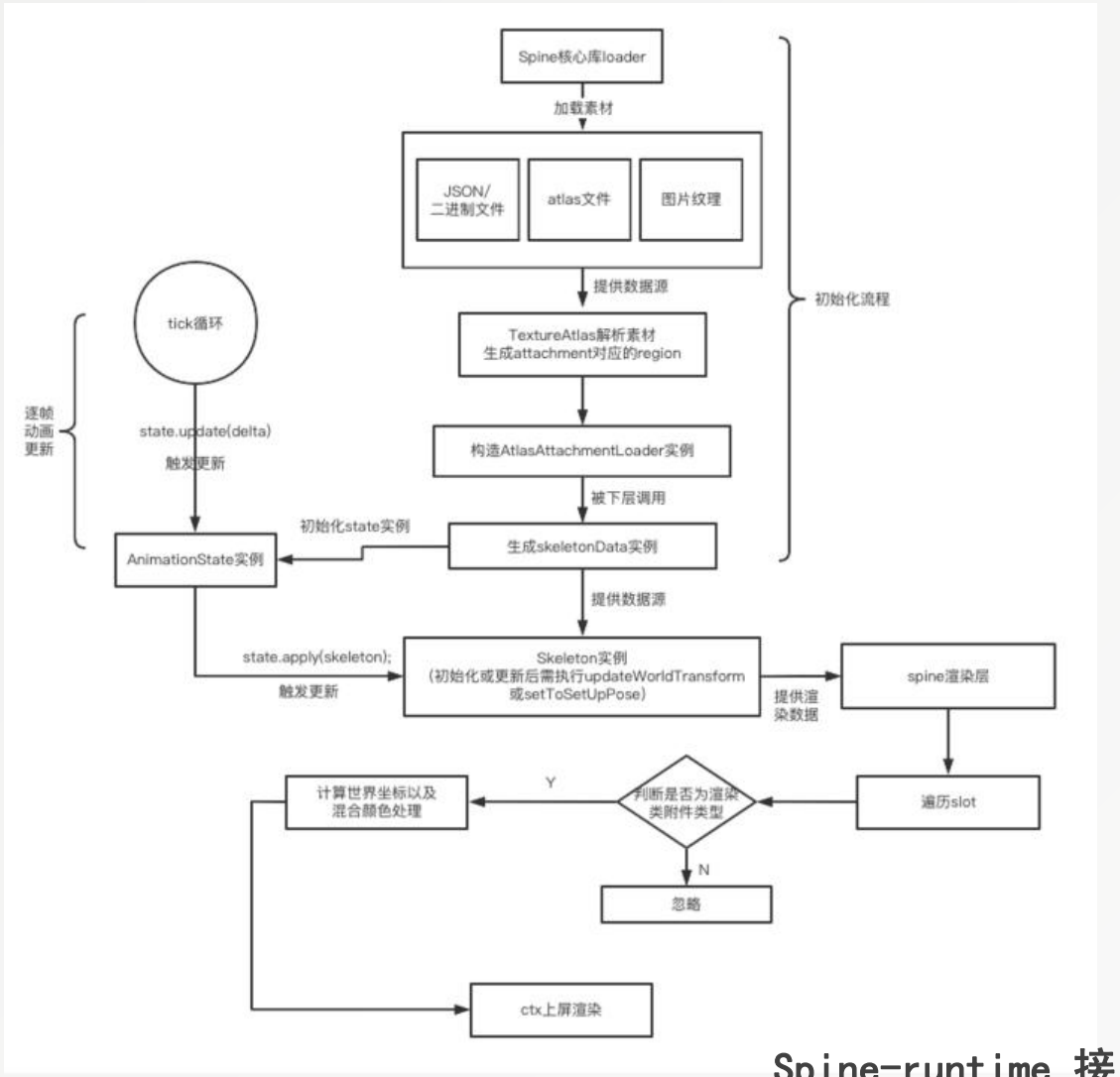
skeleton:



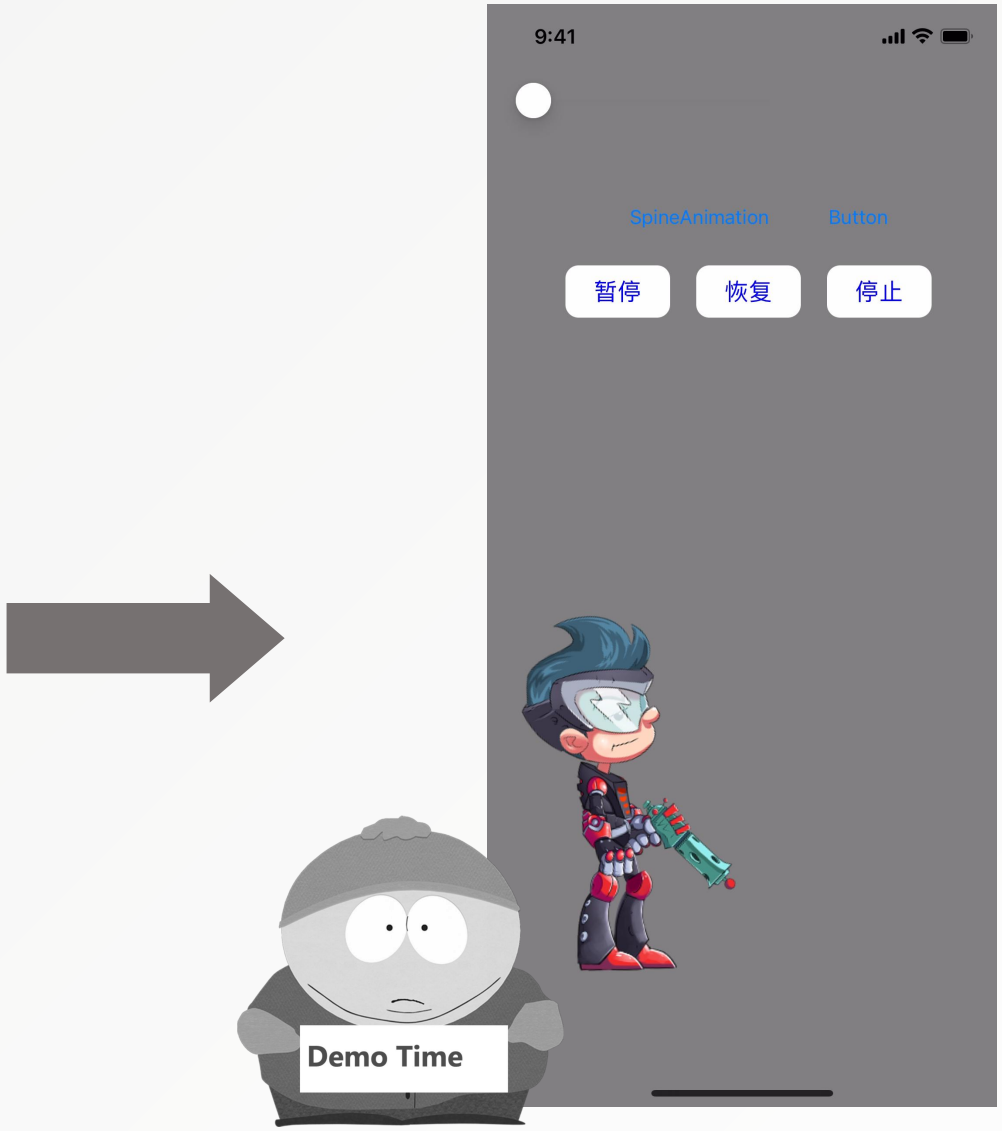
attachment:

分解：

场景2：spine动画



Spine-runtime 接入流程图



THANK YOU

Q&A