(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 107832108 B
(45)授权公告日 2021.05.07

(21)申请号 201610827353.8

(22)申请日 2016.09.14

(65)同一申请的已公布的文献号
申请公布号 CN 107832108 A

(43)申请公布日 2018.03.23

(73)专利权人 阿里巴巴集团控股有限公司
地址 英属开曼群岛大开曼资本大厦一座四层847号邮箱

(72)发明人 黄刚 潘华 李杰 胡吉嵩

(74)专利代理机构 北京清源汇知识产权代理事务所(特殊普通合伙) 11644
代理人 冯德魁

(51)Int.Cl.
G06F 9/451(2018.01)

(56)对比文件
CN 102254292 A,2011.11.23
CN 102591651 A,2012.07.18
CN 103336816 A,2013.10.02
CN 104978357 A,2015.10.14
CN 102254292 A,2011.11.23
方强.基于WebGL的3D图形引擎研究与实现.《中国优秀硕士学位论文全文数据库 信息科技辑》.2013,
王德生.基于HTML5和WebGL的三维WebGIS系统构建及应用.《中国优秀硕士学位论文全文数据库 基础科学辑》.2015,

审查员 韩典伯

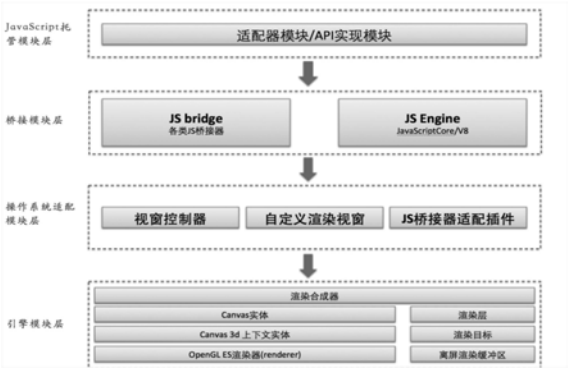
权利要求书3页 说明书17页 附图8页

(54)发明名称

3D canvas网页元素的渲染方法、装置及电子设备

(57)摘要

本申请公开了一种3D canvas网页元素的渲染方法、装置及电子设备,以及一种2D canvas网页元素的渲染方法、装置及电子设备。其中,3D canvas网页元素的渲染方法包括:截获对3D canvas网页元素的WebGL绘图指令;将WebGL绘图指令携带的指令信息传送到系统框架层;根据指令信息,生成与WebGL绘图指令对应的OpenGL绘图指令;将OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成3D canvas网页元素的渲染结果。采用本申请提供的方法,能够避免使用系统Webview对3D canvas网页元素渲染绘制,从而达到提高3D Canvas支持的兼容性与覆盖率的效果。



CN 107832108 B

1.一种3D canvas网页元素的渲染方法,其特征在于,包括:

从客户端脚本层截获对3D canvas网页元素的WebGL绘图指令;

将所述WebGL绘图指令携带的指令信息传送到系统框架层;

根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL 绘图指令;

将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

2.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述截获对3D canvas网页元素的WebGL绘图指令,采用如下方式:

通过所述3D canvas网页元素的替换DOM对象,截获所述WebGL绘图指令。

3.根据权利要求2所述的3D canvas网页元素的渲染方法,其特征在于,还包括:

将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象。

4.根据权利要求3所述的3D canvas网页元素的渲染方法,其特征在于,在所述将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象之前,还包括:

判断操作系统是否允许执行DOM对象替换的操作;若是,则进入下一步骤;若否,则使用Webview视窗绘制所述3D canvas网页元素。

5.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,包括:

从所述WebGL绘图指令中提取所述指令信息;

根据第一预设指令转换规则和所述指令信息,生成与所述WebGL绘图指令对应的渲染命令;

将所述渲染命令从传送到所述系统框架层。

6.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,采用如下方式:

通过JS bridge或者JavaScript引擎,将所述指令信息传送到所述系统框架层。

7.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,采用如下方式:

根据预设的时间间隔,将所述指令信息传送到所述系统框架层。

8.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述生成与所述WebGL绘图指令对应的OpenGL 绘图指令,包括:

将所述指令信息从所述系统框架层传送到系统本地层;

在所述系统本地层中,根据所述指令信息生成所述OpenGL 绘图指令。

9.根据权利要求8所述的3D canvas网页元素的渲染方法,其特征在于,所述将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,包括:

获取所述3D canvas网页元素对应的上下文信息及渲染目标;

根据所述上下文信息和所述OpenGL绘图指令,对所述渲染目标进行图形渲染;

将所述渲染目标输入到系统的离屏缓冲区。

10.根据权利要求9所述的3D canvas网页元素的渲染方法,其特征在于,所述将所述渲染目标输入到系统的离屏缓冲区,采用如下方式:

通过FBO技术,将多个所述渲染目标合成到所述离屏缓冲区;不同的所述渲染目标对应

不同的所述3D canvas网页元素。

11.根据权利要求9所述的3D canvas网页元素的渲染方法,其特征在于,在所述获取所述3D canvas元素对应的上下文信息及所述渲染目标之前,还包括:

判断所述上下文信息及所述渲染目标是否存在;若否,则创建所述上下文信息及所述渲染目标,以及对所述上下文信息和所述渲染目标进行绑定。

12.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,还包括:

对所述OpenGL渲染视窗和系统的Webview视窗进行合成,形成所述3D canvas网页元素所在网页的渲染结果。

13.根据权利要求12所述的3D canvas网页元素的渲染方法,其特征在于,所述对所述OpenGL渲染视窗和系统的Webview视窗进行合成,采用如下方式:

通过系统的窗口管理机制,将所述OpenGL渲染视窗和所述Webview视窗混合。

14.根据权利要求12所述的3D canvas网页元素的渲染方法,其特征在于,所述Webview视窗中绘制了所述网页内的非3D canvas网页元素。

15.根据权利要求1所述的3D canvas网页元素的渲染方法,其特征在于,所述系统包括Andriod系统或iOS系统。

16.一种3D canvas网页元素的渲染装置,其特征在于,包括:

WebGL指令截获单元,用于从客户端脚本层截获对3D canvas网页元素的WebGL绘图指令;

WebGL指令传送单元,用于将所述WebGL绘图指令携带的指令信息传送到系统框架层;

OpenGL指令生成单元,用于根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL 绘图指令;

OpenGL指令绘制单元,用于将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

17.根据权利要求16所述的3D canvas网页元素的渲染装置,其特征在于,所述WebGL指令截获单元,具体用于通过所述3D canvas网页元素的替换DOM对象,截获所述WebGL绘图指令。

18.根据权利要求16所述的3D canvas网页元素的渲染装置,其特征在于,所述WebGL指令传送单元包括:

指令信息提取子单元,用于从所述WebGL绘图指令中提取所述指令信息;

渲染命令生成子单元,用于根据第一预设指令转换规则和所述指令信息,生成与所述WebGL绘图指令对应的渲染命令;

渲染命令传送子单元,用于将所述渲染命令从传送到所述系统框架层。

19.根据权利要求16所述的3D canvas网页元素的渲染装置,其特征在于,所述OpenGL指令生成单元包括:

指令信息传送子单元,用于将所述指令信息从所述系统框架层传送到系统本地层;

OpenGL指令生成子单元,用于在所述系统本地层中,根据所述指令信息生成所述OpenGL 绘图指令。

20.一种电子设备,其特征在于,包括:

显示器;

处理器;以及

存储器,用于存储实现3D canvas网页元素的渲染方法的程序,该设备通电并通过所述处理器运行该3D canvas网页元素的渲染方法的程序后,执行下述步骤:从客户端脚本层截获对3D canvas网页元素的WebGL绘图指令;将所述WebGL绘图指令携带的指令信息传送到系统框架层;根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL 绘图指令;将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

3D canvas网页元素的渲染方法、装置及电子设备

技术领域

[0001] 本申请涉及移动互联网技术领域，具体涉及一种3D canvas网页元素的渲染方法、装置及电子设备。本申请同时涉及一种2D canvas网页元素的渲染方法、装置及电子设备。

背景技术

[0002] 随着Web技术的发展和HTML5标准的落地，HTML5开发越来越火热。HTML5开发具有跨平台、开发成本低、迭代快、易动态发布等优点，在移动App开发中占据的比重越来越大。Canvas元素作为HTML5的一个重要特性，在移动开发页面展示中，特别是图片处理、动画展示等场合，起了不可或缺的作用。

[0003] 在移动应用程序中，HTML5的3D canvas依靠系统Webview来完成渲染绘制，实现上依赖于系统Webview是否支持WebGL接口。WebGL是一种3D绘图标准，这种绘图技术标准允许把JavaScript和OpenGL ES 2.0结合在一起，通过增加OpenGL ES 2.0的一个JavaScript绑定，WebGL可以为HTML5Canvas提供硬件3D加速渲染，这样Web开发人员就可以借助系统显卡来在浏览器里更流畅地展示3D场景和模型了，还能创建复杂的导航和数据视觉化。显然，WebGL技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂3D结构的网站页面，甚至可以用来设计3D网页游戏等。

[0004] 然而，HTML5的3D canvas中的WebGL直到Android 5.0、iOS 8.0才被支持，并且支持的版本仅为WebGL 1.1，而非最新的WebGL 2.0。综上所述，现有技术存在部分移动操作系统不支持WebGL渲染的问题。

发明内容

[0005] 本申请提供一种3D canvas网页元素的渲染方法、装置及电子设备，以解决现有技术下部分移动操作系统不支持WebGL渲染的问题。本申请还提供一种2D canvas网页元素的渲染方法、装置及电子设备。

[0006] 本申请提供一种3D canvas网页元素的渲染方法，包括：

[0007] 截获对3D canvas网页元素的WebGL绘图指令；

[0008] 将所述WebGL绘图指令携带的指令信息传送到系统框架层；

[0009] 根据所述指令信息，生成与所述WebGL绘图指令对应的OpenGL绘图指令；

[0010] 将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述3D canvas网页元素的渲染结果。

[0011] 可选的，所述截获对3D canvas网页元素的WebGL绘图指令，采用如下方式：

[0012] 通过所述3D canvas网页元素的替换DOM对象，截获所述WebGL绘图指令。

[0013] 可选的，还包括：

[0014] 将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象。

[0015] 可选的，在所述将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象之前，还包括：

[0016] 判断所述操作系统是否允许执行DOM对象替换的操作;若是,则进入下一步骤;若否,则使用Webview视窗绘制所述3D canvas网页元素。

[0017] 可选的,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,包括:

[0018] 从所述WebGL绘图指令中提取所述指令信息;

[0019] 根据第一预设指令转换规则和所述指令信息,生成与所述WebGL绘图指令对应的渲染命令;

[0020] 将所述渲染命令从传送到所述系统框架层。

[0021] 可选的,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,采用如下方式:

[0022] 通过JS bridge或者JavaScript引擎,将所述指令信息传送到所述系统框架层。

[0023] 可选的,所述将所述WebGL绘图指令携带的指令信息传送到系统框架层,采用如下方式:

[0024] 根据预设的时间间隔,将所述指令信息传送到所述系统框架层。

[0025] 可选的,所述生成与所述WebGL绘图指令对应的OpenGL绘图指令,包括:

[0026] 将所述指令信息从所述系统框架层传送到系统本地层;

[0027] 在所述系统本地层中,根据所述指令信息生成所述OpenGL绘图指令。

[0028] 可选的,所述将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,包括:

[0029] 获取所述3D canvas网页元素对应的上下文信息及渲染目标;

[0030] 根据所述上下文信息和所述OpenGL绘图指令,对所述渲染目标进行图形渲染;

[0031] 将所述渲染目标输入到系统的离屏缓冲区。

[0032] 可选的,所述将所述渲染目标输入到系统的离屏缓冲区,采用如下方式:

[0033] 通过FBO技术,将多个所述渲染目标合成到所述离屏缓冲区;不同的所述渲染目标对应不同的所述3D canvas网页元素。

[0034] 可选的,在所述获取所述3D canvas元素对应的上下文信息及所述渲染目标之前,还包括:

[0035] 判断所述上下文信息及所述渲染目标是否存在;若否,则创建所述上下文信息及所述渲染目标,以及对所述上下文信息和所述渲染目标进行绑定。

[0036] 可选的,还包括:

[0037] 对所述OpenGL渲染视窗和系统的Webview视窗进行合成,形成所述3D canvas网页元素所在网页的渲染结果。

[0038] 可选的,所述对所述OpenGL渲染视窗和系统的Webview视窗进行合成,采用如下方式:

[0039] 通过系统的窗口管理机制,将所述OpenGL渲染视窗和所述Webview视窗混合。

[0040] 可选的,所述Webview视窗中绘制了所述网页内的非3D canvas网页元素。

[0041] 可选的,所述系统包括Andriod系统或iOS系统。

[0042] 相应的,本申请还提供一种3D canvas网页元素的渲染装置,包括:

[0043] WebGL指令截获单元,用于截获对3D canvas网页元素的WebGL绘图指令;

[0044] WebGL指令传送单元,用于将所述WebGL绘图指令携带的指令信息传送到系统框架层;

[0045] OpenGL指令生成单元，用于根据所述指令信息，生成与所述WebGL绘图指令对应的OpenGL绘图指令；

[0046] OpenGL指令绘制单元，用于将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述3D canvas网页元素的渲染结果。

[0047] 可选的，所述WebGL指令截获单元，具体用于通过所述3D canvas网页元素的替换DOM对象，截获所述WebGL绘图指令。

[0048] 可选的，所述WebGL指令传送单元包括：

[0049] 指令信息提取子单元，用于从所述WebGL绘图指令中提取所述指令信息；

[0050] 渲染命令生成子单元，用于根据第一预设指令转换规则和所述指令信息，生成与所述WebGL绘图指令对应的渲染命令；

[0051] 渲染命令传送子单元，用于将所述渲染命令从传送到所述系统框架层。

[0052] 可选的，所述OpenGL指令生成单元包括：

[0053] 指令信息传送子单元，用于将所述指令信息从所述系统框架层传送到系统本地层；

[0054] OpenGL指令生成子单元，用于在所述系统本地层中，根据所述指令信息生成所述OpenGL绘图指令。

[0055] 相应的，本申请还提供一种电子设备，包括：

[0056] 显示器；

[0057] 处理器；以及

[0058] 存储器，用于存储实现3D canvas网页元素的渲染方法的程序，该设备通电并通过所述处理器运行该3D canvas网页元素的渲染方法的程序后，执行下述步骤：截获对3D canvas网页元素的WebGL绘图指令；将所述WebGL绘图指令携带的指令信息传送到系统框架层；根据所述指令信息，生成与所述WebGL绘图指令对应的OpenGL绘图指令；将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述3D canvas网页元素的渲染结果。

[0059] 相应的，本申请还提供一种2D canvas网页元素的渲染方法，包括：

[0060] 截获对2D canvas网页元素的绘图指令；

[0061] 将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层；

[0062] 根据所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令；

[0063] 将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述2D canvas网页元素的渲染结果。

[0064] 可选的，所述截获对2D canvas网页元素的WebGL绘图指令，采用如下方式：

[0065] 通过所述2D canvas网页元素的替换DOM对象，截获所述2D canvas网页元素的绘图指令。

[0066] 可选的，所述将所述对2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层，包括：

[0067] 从所述2D canvas网页元素的绘图指令中提取所述指令信息；

[0068] 根据第一预设指令转换规则和所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的渲染命令；

[0069] 将所述渲染命令从传送到所述系统框架层。

[0070] 可选的，所述生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令，包括：

[0071] 将所述指令信息从所述系统框架层传送到系统本地层；

[0072] 在所述系统本地层中，根据将所述指令信息生成所述OpenGL绘图指令。

[0073] 可选的，所述将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，包括：

[0074] 获取所述2D canvas网页元素对应的上下文信息及渲染目标；

[0075] 根据所述上下文信息和所述OpenGL绘图指令，对所述渲染目标进行图形渲染；

[0076] 将所述渲染目标输入到系统的离屏缓冲区。

[0077] 可选的，还包括：

[0078] 对所述OpenGL渲染视窗和系统的Webview视窗进行合成，形成所述2D canvas网页元素所在网页的渲染结果。

[0079] 相应的，本申请还提供一种2D canvas网页元素的渲染装置，包括：

[0080] 指令截获单元，用于截获对2D canvas网页元素的绘图指令；

[0081] 指令传送单元，用于将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层；

[0082] OpenGL指令生成单元，用于根据所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令；

[0083] OpenGL指令绘制单元，用于将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述2D canvas网页元素的渲染结果。

[0084] 可选的，所述指令截获单元，具体用于通过所述2D canvas网页元素的替换DOM对象，截获所述2D canvas网页元素的绘图指令。

[0085] 可选的，所述指令传送单元包括：

[0086] 指令信息提取子单元，用于从所述2D canvas网页元素的绘图指令中提取所述指令信息；

[0087] 渲染命令生成子单元，用于根据第一预设指令转换规则和所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的渲染命令；

[0088] 渲染命令传送子单元，用于将所述渲染命令从传送到所述系统框架层。

[0089] 可选的，所述OpenGL指令生成单元包括：

[0090] 指令信息传送子单元，用于将所述指令信息从所述系统框架层传送到系统本地层；

[0091] OpenGL指令生成子单元，用于在所述系统本地层中，根据将所述指令信息生成所述OpenGL绘图指令。

[0092] 相应的，本申请还提供一种电子设备，包括：

[0093] 显示器；

[0094] 处理器；以及

[0095] 存储器，用于存储实现2D canvas网页元素的渲染方法的程序，该设备通电并通过所述处理器运行该2D canvas网页元素的渲染方法的程序后，执行下述步骤：截获对2D canvas网页元素的绘图指令；将所述2D canvas网页元素的绘图指令携带的指令信息传送

到系统框架层;根据所述指令信息,生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令;将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述2D canvas网页元素的渲染结果。

[0096] 与现有技术相比,本申请提供的方法,通过截获对3D canvas网页元素的WebGL绘图指令;将所述WebGL绘图指令携带的指令信息传送到系统框架层;根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL绘图指令;将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

[0097] 使用本申请提供的3D canvas网页元素的渲染方法,将截获的针对3D canvas网页元素的WebGL绘图指令携带的指令信息从客户端脚本层传送到系统框架层,并根据指令信息获取与WebGL绘图指令对应的OpenGL绘图指令,再将OpenGL绘图指令绘制到系统的OpenGL渲染视窗中,形成3D canvas网页元素的渲染结果;这种处理方式,避免使用系统Webview对3D canvas网页元素渲染绘制;因此,可以提高移动Web应用内HTML5 3D Canvas支持的兼容性与覆盖率,使得原先不支持WebGL或者支持WebGL不理想的移动设备中的应用均可使用WebGL进行动画渲染。

[0098] 本申请提供的3D canvas网页元素的渲染方法,通过引入JavaScript桥接层与系统框架层,充分解除了对于操作系统的依赖,能够兼容现有发布的移动操作系统,具有很强的兼容性,从而可以达到跨平台的技术效果。

附图说明

[0099] 图1是本申请提供的一种3D canvas网页元素的渲染方法的实施例的流程图;

[0100] 图2是本申请提供的一种3D canvas网页元素的渲染方法的实施例的结构示意图;

[0101] 图3是本申请提供的一种3D canvas网页元素的渲染方法的实施例的具体流程图;

[0102] 图4是本申请提供的一种3D canvas网页元素的渲染装置的实施例的示意图;

[0103] 图5是本申请提供的一种电子设备的实施例的示意图;

[0104] 图6是本申请提供的一种2D canvas网页元素的渲染方法的实施例的流程图;

[0105] 图7是本申请提供的一种2D canvas网页元素的渲染方法的实施例的结构示意图;

[0106] 图8是本申请提供的一种2D canvas网页元素的渲染方法的实施例的具体流程图;

[0107] 图9是本申请提供的一种2D canvas网页元素的渲染装置的实施例的示意图;

[0108] 图10是本申请提供的一种电子设备的实施例的示意图。

具体实施方式

[0109] 在下面的描述中阐述了很多具体细节以便于充分理解本申请。但是,本申请能够以很多不同于在此描述的其它方式来实施,本领域技术人员可以在不违背本申请内涵的情况下做类似推广,因此本申请不受下面公开的具体实施的限制。

[0110] 在本申请中,提供了一种3D canvas网页元素的渲染方法、装置及电子设备。在下面的实施例中逐一进行详细说明。

[0111] 本申请提供的3D canvas网页元素的渲染方法,其核心的基本思想为:截获对3D canvas网页元素的WebGL绘图指令,并将截获的WebGL绘图指令携带的指令信息从客户端脚本层传送到系统框架层,并根据指令信息获取与WebGL绘图指令对应的OpenGL绘图指令,再

将OpenGL绘图指令绘制到系统的OpenGL渲染视窗中，形成3D canvas网页元素的渲染结果。由于能够避免使用系统Webview对3D canvas网页元素进行渲染绘制，因而，可以提高移动Web应用内HTML5 3D Canvas支持的兼容性与覆盖率。

[0112] 请参考图1，其为本申请的3D canvas网页元素的渲染方法实施例的流程图。所述方法包括如下步骤：

[0113] 步骤S101：截获对3D canvas网页元素的WebGL绘图指令。

[0114] canvas标签是在HTML5中出现的，最先开始支持的是2D图形绘制，现在又开始有了基于WebGL的3D绘制。canvas元素是一个矩形区域，可以控制其中的每一像素。canvas元素本身是没有绘图能力的，所有的绘制工作必须在JavaScript内部完成，即：canvas元素使用JavaScript在网页上绘制图像。

[0115] WebGL是一套基于OpenGL ES 2.0的3d图形API，能够允许在网页中使用类似于OpenGL的WebGL实现3D渲染。WebGL完美地解决了现有的Web交互式三维动画的两个问题：第一，它通过HTML脚本本身实现Web交互式三维动画的制作，无需任何浏览器插件支持；第二，它利用底层的图形硬件加速功能进行的图形渲染，是通过统一的、标准的、跨平台的OpenGL接口实现的。

[0116] WebGL的图形API是通过HTML 5的3D canvas标签来使用的。本申请将JavaScript代码中对3D canvas网页元素的绘图操作指令，称为WebGL绘图指令。

[0117] 例如，一个canvas元素的代码为<canvas id＝"myCanvas"width＝"200"height＝"100"></canvas>，该元素包括id属性、宽度属性和高度属性；对该元素的JavaScript绘图代码可如下所示：

[0118] <script type＝"text/javascript">

[0119] var c＝document.getElementById("myCanvas")；

[0120] var cxt＝c.getContext("3d")；

[0121] cxt.fillStyle＝"#FF0000"；

[0122] cxt.fillRect(0,0,150,75)；

[0123] </script>

[0124] 由上述代码可见，JavaScript使用id来寻找canvas元素，然后，创建context对象，getContext("3d")对象是内建的HTML5对象，拥有多种绘制3D图形的方法；其中，cxt.fillRect(0,0,150,75)指令即为一条WebGL绘图指令；通过上述代码，可在canvas元素中绘制一个红色的矩形。

[0125] 在系统默认的情况下，是通过系统的Webview视窗绘制所述3D canvas网页元素的；而本申请提供的方法将截获对3D canvas网页元素的WebGL绘图指令，将所述3D canvas网页元素绘制在系统的OpenGL渲染视窗中。

[0126] 本申请提供的方法对WebGL绘图指令的截获方式不加以限制，可使用任意一种技术截获对3D canvas网页元素的WebGL绘图指令。

[0127] 作为一种可选的方案，本步骤可采用如下方式实现：通过所述3D canvas网页元素的替换DOM对象，截获所述WebGL绘图指令。

[0128] 所述3D canvas网页元素所在网页的DOM(Document Object Model，文档对象模型)包括所述3D canvas网页元素对应的DOM对象，可通过JavaScript(以及其他编程语言)

对DOM进行访问。

[0129] 所述替换DOM对象是指,所述3D canvas网页元素的DOM对象的替换对象。要通过所述替换DOM对象截获所述WebGL绘图指令,首先需要将所述3D canvas网页元素所在网页的DOM树中所述3D canvas网页元素的DOM对象替换为所述替换DOM对象。

[0130] 具体实施时,可首先通过getElementById(id)方法获取3D canvas网页元素,然后,再通过appendChild(node)方法插入所述3D canvas网页元素的替换DOM对象,该替换DOM对象可以为一个自定义的子节点,最后,再通过removeChild(node)方法删除3D canvas网页元素。经过上述处理后,对于该3D canvas网页元素的WebGL绘图指令,即可通过替换DOM对象截获。

[0131] 作为一种可选的方案,在所述将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象之前,还包括:判断所述操作系统是否允许执行DOM对象替换的操作的步骤;如果判断结果为是,则执行所述将所述3D canvas网页元素的DOM对象替换为所述替换DOM对象的步骤;否则,则使用Webview视窗绘制所述3D canvas网页元素。

[0132] 采用这种判断处理的方式,如果为了保证操作系统的安全性,预先做了不允许进行DOM对象替换的设置处理,则可以通过系统默认的Webview视窗绘制所述3D canvas网页元素,从而提高了用户体验。

[0133] 在截获到对3D canvas网页元素的WebGL绘图指令之后,就可以进入下一步骤,将截获的WebGL绘图指令携带的指令信息从客户端脚本层传送到系统框架层。

[0134] 步骤S103:将所述WebGL绘图指令携带的指令信息传送到系统框架层。

[0135] 所述WebGL绘图指令携带的指令信息,包括但不限于:指令名称及指令参数,例如,drawCircle(200,300,5)这个WebGL绘图指令所携带的指令名称为drawCircle,指令参数为:x坐标为200、y坐标为300,半径为5。

[0136] 本步骤S103可采用直接传递所述指令信息的形式,将所述WebGL绘图指令携带的指令信息从客户端脚本层传送到系统框架层,也可以采用下述优选的方案作为具体实施方案。

[0137] 作为一种优选的方案,本步骤S103可包括如下具体步骤:1)从所述WebGL绘图指令中提取所述指令信息;2)根据第一预设指令转换规则和所述指令信息,生成与所述WebGL绘图指令对应的渲染命令;3)将所述渲染命令传送到所述系统框架层。

[0138] 通过上述第2个步骤,可根据第一预设指令转换规则对各种指令信息进行拼接形成预设形式的渲染命令(即:指令信息的规范化表达形式),再通过上述第3个步骤以渲染命令形式将指令信息从客户端脚本层传送到系统框架层;这种处理方式,可以使得后续环节易于对该命令进行解析,进而获得与WebGL绘图指令对应的OpenGL绘图指令。

[0139] 本实施例中,设置了一个GCanvas类,该类包括与上述第2个步骤对应的指令转换程序代码如下所示:

[0140]
```
GContextWebGL.prototype.uniform4f = function(location, x, y, z, w) {
    this._drawCommands += (this.uniform4fId + location.id + "," + x + "," + y +
"," + z + "," + w + ";");
};
```

[0141] 例如,WebGL的绘图指令为:WebGLRenderingContext.uniform4f(location,x,y,z,w),经过上述代码处理后该指令转换为渲染命令(命令串):“17,3,100,200,300,400;”;其中,“17”指代uniform4f方法在GCanvas中的指令ID,“3”指代location对象的ID,“100,200,300”代表位置“x,y,z”,“400”代表宽度“w”,逗号“,”在GCanvas中是变量分隔符,分号“;”在GCanvas中表示命令结束符。

[0142] 需要说明的是,本步骤S103及上一步骤S101均运行在客户端脚本层。所述客户端脚本层可包括如下功能:1)查找HTML5页面中是否有可供替换的3D canvas元素,如有且能成功替换,则新建一个替换DOM对象将之插入原DOM树,并将原3D canvas元素删除,后续所有对于原3D canvas元素的调用、操作都将被新插入的DOM对象托管;2)截获对原3D canvas网页元素的WebGL绘图指令,并将该绘图指令携带的指令信息从客户端脚本层传送到系统框架层。此外,所述客户端脚本层还用于响应用户对网页元素的交互指令、响应回调函数、计算canvas网页元素的大小与位置、及多个canvas网页元素的层管理等。

[0143] 所述系统框架层是指,使用操作系统原生语言开发程序的应用程序框架层,例如,当本申请的方法应用在Android系统中时,系统框架层即为Java语言开发的程序层;当本申请的方法应用在iOS系统中时,系统框架层即为Object-C语言开发的程序层。

[0144] 具体实施时,可通过实时的方式将所述指令信息从客户端脚本层传送到系统框架层。然而,由于这种方式需要对截获的每一条指令进行系统框架层的渲染处理,因此,这种处理方法存在降低系统性能的问题。

[0145] 为了避免对系统性能产生负面影响,作为一种优选的方案,本步骤可采用如下方式实现:根据预设的时间间隔,将所述指令信息从客户端脚本层传送到所述系统框架层。这样,既可以保证一定的图像帧率,从而满足用户的视觉体验,又可以避免降低系统性能。

[0146] 此外,为了能够将所述指令信息从客户端脚本层传送到系统框架层,可采用如下技术实现:通过JS bridge或者JavaScript引擎,将所述指令信息传送到所述系统框架层。

[0147] 通过所述JS bridge或者JavaScript引擎,可以实现JavaScript代码与系统框架层代码的交互,从而将指令信息从客户端脚本层传送到系统框架层。

[0148] 如果采用JS bridge方法传递信息,可应用cordova等较为成熟的方案。如果采用JavaScript引擎方法传递信息,则当本申请的方法应用在Android系统中时,可采用V8引擎;当本申请的方法应用在iOS系统中时,可采用JavaScriptCore引擎。

[0149] 需要注意的是,如果使用了JS bridge的方案,那么针对所选取的JS bridge,还需要在所述系统框架层中实现JS bridge的适配插件,该适配插件用以接收由JS bridge传递给系统框架层的交互指令或渲染指令(即:指令信息),并且根据需要进行处理或者转发。

[0150] 需要说明的是,本申请提供的方法,通过引入JavaScript桥接层与系统框架层,充分解除了对于操作系统的依赖,能够兼容现有发布的移动操作系统,具有很强的兼容性,从而可以达到跨平台的技术效果。

[0151] 当所述WebGL绘图指令携带的指令信息被传送到系统框架层后,就可以进入下一步骤,根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL绘图指令。

[0152] 步骤S105:根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL绘图指令。

[0153] 本步骤根据接收到的所述指令信息,生成与所述WebGL绘图指令对应的OpenGL绘

图指令。通过本步骤及上述步骤的处理，能够将原始的WebGL绘图指令最终转换为对应的OpenGL绘图指令。

[0154] 需要说明的是，本步骤及下一步骤可直接在系统框架层实施，例如，Android系统框架层为应用程序的开发提供了两个类来实现对3D图形API OpenGL ES的支持，分别是GLSurfaceView和GLSurfaceView.Renderer；通过这两个接口类，就可以基于OpenGL ES来进行3D图形的开发了。然而，这种实施方式存在渲染效率较低的问题。

[0155] 作为一种优选的方案，本步骤及下一步骤可在系统本地层(如：Android系统的NDK，Native Development Kit)实施。所述系统本地层又称为C++引擎层。该方式和上述方式类似，以Android系统为例，该方式与上述方式的不同之处主要为：将GLSurfaceView.Renderer的实现部分放到本地来实现，然后通过JNI来实现Java端和C端的通信。

[0156] 在本实施例中，本步骤及下一步骤在系统本地层实施。为此，本步骤首先要将所述指令信息从所述系统框架层传送到系统本地层，然后，在系统本地层中，根据所述指令信息生成所述OpenGL绘图指令。通过在系统本地层生成并执行所述OpenGL绘图指令，可获得较高的渲染性能。

[0157] 具体实施时，本步骤S105可采用如下方式实现：根据第二预设指令转换规则和所述指令信息，生成所述OpenGL绘图指令。

[0158] 所述第二预设指令转换规则与上述第一预设指令转换规则相关，根据所述第一预设指令转换规则，将WebGL绘图指令转换为渲染命令；根据所述第二预设指令，将渲染命令形式的指令信息转换为与WebGL绘图指令对应的OpenGL绘图指令。

[0159] 例如，在客户端脚本层将所述WebGL绘图指令转换为渲染命令“B,1”，在经过系统框架层传递到系统本地层(C++引擎层)后，会被如下代码处理：

```
case 'B':
```

[0160]

```
{
    p ++;
    float tokens[1] = { 0};
    p = ParseTokens(p,tokens);
```

[0161]

```
    SetGlobalCompositeOperation(EJCompositeOperation(tokens[0]));
    break;
}
```

[0162] 通过上述代码处理后，“1”被转化为枚举值

[0163] “kEJCompositeOperationSourceOver”，然后，在SetGlobalCompositeOperation方法中继续处理，该方法代码如下所示：

[0164]
```
void EJCanvasContext::SetGlobalCompositeOperation(EJCompositeOperation
op)
    {
        SendVertexBufferToGPU();
        glBlendFunc( EJCompositeOperationFuncs[op].source,
EJCompositeOperationFuncs[op].destination);
        current_state_->global_composite_op = op;
    }
```

[0165] 经过上述代码解析出“glBlendFunc”,该指令为WebGL绘图指令真正对应的OpenGL的绘图指令。

[0166] 生成与所述WebGL绘图指令对应的OpenGL绘图指令后,就可以进入下一步骤,根据这些OpenGL绘图指令绘制所述3D canvas网页元素。

[0167] 步骤S107:将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

[0168] 本步骤将上一步骤生成的所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗中,以形成所述3D canvas网页元素的渲染结果。

[0169] 将OpenGL绘图指令绘制到系统的OpenGL渲染视窗为本技术领域的公职技术,本申请对具体绘制方式不加以限制。下面给出一种可选的具体实施方法。

[0170] 具体实施时,本步骤可包括如下步骤:1)获取所述3D canvas网页元素对应的上下文信息及渲染目标;2)根据所述上下文信息和所述OpenGL绘图指令,对所述渲染目标进行图形渲染;3)将所述渲染目标输入到系统的离屏缓冲区。

[0171] 1)获取所述3D canvas网页元素对应的上下文信息及渲染目标。

[0172] 要实现将OpenGL绘图指令绘制到系统的OpenGL渲染视窗的处理,需要系统本地层具有如下功能:1)创建、管理、销毁3D canvas实体(entity)与3D canvas的上下文实体(context entity),其中,上下文实体中保存有canvas元素的上下文信息,canvas实体保存有canvas元素的属性信息;2)负责管理将不同的canvas元素渲染到多个渲染层(render layer)的渲染目标(render target)中;3)将多个渲染目标通过OpenGL中的FBO技术合成到离屏渲染缓冲区。

[0173] 简单来说,渲染目标是一个表面,图形API可以往上面“画”东西,实质上,渲染目标是一个连续的内存区域,这样的内存区域可以同时的存在多个,也就是多个渲染目标。在本实施例中,不同的3D canvas网页元素对应不同的渲染目标。

[0174] 所述上下文信息及所述渲染目标是在第一次绘制3D canvas网页元素时生成的,因此,在所述获取所述3D canvas元素对应的上下文信息及所述渲染目标之前,还包括:判断所述上下文信息及所述渲染目标是否存在;如果不存在,则需要创建所述上下文信息及所述渲染目标,以及对所述上下文信息和所述渲染目标进行绑定。通过将新创建的所述上下文信息和所述渲染目标绑定,使得3D canvas元素、上下文信息及渲染目标三者相关联。

[0175] 2)根据所述上下文信息和所述OpenGL绘图指令,对所述渲染目标进行图形渲染。

[0176] 获取到3D canvas网页元素对应的上下文信息及渲染目标后，本步骤根据上下文信息和步骤S105生成的OpenGL绘图指令，对所述渲染目标进行图形渲染，以形成所述3D canvas网页元素的渲染结果。

[0177] 具体实施时，由canvas上下文实体根据生成的OpenGL绘图指令调用渲染器(renderer)对应的API，渲染器调用相应的OpenGL ES API并将结果渲染至渲染目标中。

[0178] 3)将所述渲染目标输入到系统的离屏缓冲区。

[0179] OpenGL中，GPU屏幕渲染有以下两种方式：1)当前屏幕渲染(On-Screen Rendering)；2)离屏渲染(Off-Screen Rendering)。其中，当前屏幕渲染是指，GPU在当前用于显示的屏幕缓冲区中进行渲染操作；离屏渲染是指，GPU在当前屏幕缓冲区以外新开辟一个缓冲区进行渲染操作。

[0180] 由于所述3D canvas网页元素的渲染结果最终需要融合到系统Webview视窗，因此，本步骤采用离屏渲染方式，将渲染后的渲染目标输入到系统的离屏缓冲区。

[0181] 需要注意的是，如果所述3D canvas网页元素所在的网页包括多个3D canvas网页元素，则所述将所述渲染目标输入到系统的离屏缓冲区，可采用如下方式：通过FBO(Frame Buffer Object)技术，将多个所述渲染目标合成到所述离屏缓冲区；这种情况下，不同的所述渲染目标对应不同的所述3D canvas网页元素，离屏缓冲区包括将多个3D canvas网页元素的渲染效果融合在一起的合成渲染效果数据。

[0182] 上述第1、2和3步骤在本实施例中的具体实现如下所述：对于3D canvas网页元素的canvas与相应webgl context，GCanvas内部会创建相应的两个类“GCanvas”与“GCanvasContext”与之对应；根据上下文信息，利用FBO技术，会将相应的图形绘制到FBO指定的纹理(texture)上，最终由系统的EGL的swap buffer机制渲染到屏幕。

[0183] 通过上述第1、2和3步骤将3D canvas网页元素的渲染效果输入到系统的离屏缓冲区后，具体实施时，还需要将离屏缓冲区的数据复制到系统框架层的OpenGL渲染视窗(如：Android系统的GlSurfaceView)中，以便系统框架层对所述OpenGL渲染视窗和系统的Webview视窗进行合成，形成所述3D canvas网页元素所在网页的整体渲染效果。

[0184] 具体实施时，可通过系统的窗口管理机制，将所述OpenGL渲染视窗和所述Webview视窗混合。其中，所述Webview视窗中绘制了所述网页内的非3D canvas网页元素。

[0185] 在本实施例中，通过HybridViewController类(继承自ViewController)，将WebView与GCanvasView(继承自GLSurfaceView)加入到原元素的ViewGroup中，部分源码如下：

[0186]
```
public void init(Activity activity, WebView webView, GCanvasView view) {
    super.init(activity, webView, view);
    mActivity.getWindow().getDecorView().setBackgroundColor(Color.WHITE);
    mWebView.setBackgroundColor(Color.TRANSPARENT);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
```

```
                savedWebViewLayerType = mWebView.getLayerType();
                mWebView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
            }
            attachViews();
            mWebView.bringToFront();
            mWebView.setVisibility(View.VISIBLE);
            mCanvasView.setVisibility(View.VISIBLE);
        }
```

[0187]

[0188] 综上所述，传统方法使用WebView对3D canvas网页元素进行渲染，而本申请提供的方法首先将3D canvas网页元素的动画内容渲染在系统的OpenGL渲染视窗中，然后，再将OpenGL渲染视窗和WebView视窗融合，形成网页的整体渲染效果。

[0189] 请参考图2，其为本申请的3D canvas网页元素的渲染方法实施例的结构示意图。在本实施例中，实现本申请的3D canvas网页元素的渲染方法有四层结构，分别为：JavaScript托管模块层、桥接模块层、操作系统适配模块层、引擎模块层。下面对这些层次分别作简要说明。

[0190] 1)JavaScript托管模块层：主要用于托管HTML5 3D canvas的API(WebGL绘图指令)和针对这些API实现本方案，本层的功能包括但不限于：canvas元素替换、渲染指令拼接与传递、交互指令响应、回调函数的响应、canvas元素大小与位置的计算、多个canvas的层管理等。

[0191] 2)桥接模块层：主要用于JavaScript与操作系统层(即：上述系统框架层)代码(android：Java/iOS：OC)进行交互，有两种可以实现JavaScript与底层代码交互的方案，分别为：JS bridge、JS引擎。

[0192] 3)操作系统适配模块层：主要用于实现自定义的渲染视窗(即：上述OpenGL渲染视窗)，该自定义的渲染视窗用于展现HTML5 3D canvas的渲染输出结果；以及，管理多个视窗用(自定义的渲染视窗与其他HTML5元素的渲染视窗)，以合成最终的渲染结果。如果在上一层(桥接模块层)中使用了JS bridge的方案，那么针对所选取的JS bridge，还需要实现JS bridge的适配插件用以接收由JS bridge传递给系统层的交互或渲染指令，并且按照需要进行处理或者转发。

[0193] 4)引擎模块层：即上述系统本地层，主要用以创建、管理、销毁canvas实体(entity)与canvas 3d的上下文实体(context entity)，根据接收到的渲染指令，调用相应的OpenGL ES API。在此层中还负责管理针对不同的canvas渲染到多个渲染层(render layer)的渲染目标(render target)中，最后将多个渲染目标通过OpenGL中的FBO技术合成到离屏渲染缓冲区。

[0194] 请参考图3，其为本申请的3D canvas网页元素的渲染方法实施例的具体流程图。图3所示的流程图对应上述方案的具体处理过程，结合图3能够更直观的理解上述方案的处理过程。下面结合图2对图3所示的具体流程图作简要说明。

[0195] 1)JavaScript托管模块层

[0196] 查找HTML5页面中是否有可供替换的3D canvas元素，如有且能成功替换，则新建一个自定义的DOM元素将之插入原DOM树，并将原canvas元素删除，后续所有对于原canvas元素的调用、操作都将被新插入的DOM元素托管。在接收到3D canvas的调用后，将该API调用转化为内部实现的一套命令，并通过定时器进行传入系统适配层。

[0197] 2)桥接模块层

[0198] 负责将JavaScript托管模块层生成的命令传给系统适配层。

[0199] 3)操作系统适配模块层

[0200] 接收到命令后，拼装用于渲染3D canvas的视图与其他HTML5元素的视图，并将命令传给引擎模块层。

[0201] 4)引擎模块层

[0202] 接收到命令后，进行解析，获取相应的3D canvas上下文(如若没有则新建)；合成器获取一个渲染目标(如果没有则新建)并且分配给3D canvas上下文；由3D canvas上下文根据解析的命令结果调用渲染器(renderer)对应的API，渲染器调用相应的OpenGL ES API并将结果渲染至渲染目标中；合成器将多个渲染目标通过FBO的技术进行合成并且输入到系统的离屏缓冲区。

[0203] 此时，本申请提供的3D canvas网页元素的渲染方法的主要工作均完成，操作系统将获得渲染控制权，并将渲染结果展示到屏幕上。

[0204] 在上述的实施例中，提供了一种3D canvas网页元素的渲染方法，与之相对应的，本申请还提供一种3D canvas网页元素的渲染装置。该装置是与上述方法的实施例相对应。

[0205] 请参看图4，其为本申请的3D canvas网页元素的渲染装置实施例的示意图。由于装置实施例基本相似于方法实施例，所以描述得比较简单，相关之处参见方法实施例的部分说明即可。下述描述的装置实施例仅仅是示意性的。

[0206] 本实施例的一种3D canvas网页元素的渲染装置，包括：WebGL指令截获单元101，用于截获对3D canvas网页元素的WebGL绘图指令；WebGL指令传送单元103，用于将所述WebGL绘图指令携带的指令信息传送到系统框架层；OpenGL指令生成单元105，用于根据所述指令信息，生成与所述WebGL绘图指令对应的OpenGL绘图指令；OpenGL指令绘制单元107，用于将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述3D canvas网页元素的渲染结果。

[0207] 可选的，所述WebGL指令截获单元101，具体用于通过所述3D canvas网页元素的替换DOM对象，截获所述WebGL绘图指令。

[0208] 可选的，所述WebGL指令传送单元103包括：

[0209] 指令信息提取子单元，用于从所述WebGL绘图指令中提取所述指令信息；

[0210] 渲染命令生成子单元，用于根据第一预设指令转换规则和所述指令信息，生成与所述WebGL绘图指令对应的渲染命令；

[0211] 渲染命令传送子单元，用于将所述渲染命令从传送到所述系统框架层。

[0212] 可选的，所述OpenGL指令生成单元105包括：

[0213] 指令信息传送子单元，用于将所述指令信息从所述系统框架层传送到系统本地层；

[0214] OpenGL指令生成子单元，用于在所述系统本地层中，根据所述指令信息生成所述

OpenGL绘图指令。

[0215] 请参考图5,其为本申请的电子设备实施例的示意图。由于设备实施例基本相似于方法实施例,所以描述得比较简单,相关之处参见方法实施例的部分说明即可。下述描述的设备实施例仅仅是示意性的。

[0216] 本实施例的一种电子设备,该电子设备包括:显示器101;处理器102;以及存储器103,用于存储实现3D canvas网页元素的渲染方法的程序,该设备通电并通过所述处理器运行该3D canvas网页元素的渲染方法的程序后,执行下述步骤:截获对3D canvas网页元素的WebGL绘图指令;将所述WebGL绘图指令携带的指令信息传送到系统框架层;根据所述指令信息,生成与所述WebGL绘图指令对应的OpenGL绘图指令;将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述3D canvas网页元素的渲染结果。

[0217] 与上述的方法相对应,本申请还提供一种2D canvas网页元素的渲染方法。下面首先对现有技术下的2D canvas网页元素的渲染方法及存在的问题作简要说明。

[0218] 现有技术下,HTML5 2D canvas在移动App中,依靠系统Webview来完成渲染绘制,性能上依赖于系统Webview。然而,HTML5的canvas元素在移动端的性能并不好,特别是在一些游戏、动画等高交互的页面上,canvas的性能问题更为明显,画面会出现卡顿、响应慢的现象。其中,由于Android手机的Android系统版本众多,各家手机厂商支持不一,定制化严重,HTML5canvas性能瓶颈在Android手机上尤为突出。综上所述,现有技术存在2D canvas网页元素渲染性能低的问题。

[0219] 请参考图6,其为本申请提供的一种2D canvas网页元素的渲染方法实施例的流程图,本实施例与第一实施例内容相同的部分不再赘述,请参见实施例一中的相应部分。本申请提供的一种2D canvas网页元素的渲染方法包括:

[0220] 步骤S101:截获对2D canvas网页元素的绘图指令。

[0221] 本步骤与上述实施例一的步骤S101相对应,两者不同之处在于:上述实施例一的处理对象为3D canvas网页元素,而本方法的处理对象为2D canvas网页元素;相应的,上述实施例一截获WebGL绘图指令,而本方法截获对2D canvas的绘图指令。

[0222] 本申请提供的2D canvas网页元素的渲染方法与上述3D canvas网页元素的渲染方法的核心思想及处理步骤完全一致,因此,本实施例不再对各个步骤做具体说明,相应部分请参见实施例一中的相关描述,此处不再赘述。下述描述的方法实施例仅仅是示意性的。

[0223] 作为一种优选的方案,本步骤可采用如下方式:通过所述2D canvas网页元素的替换DOM对象,截获所述2D canvas网页元素的绘图指令。

[0224] 步骤S103:将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层。

[0225] 作为一种优选的方案,本步骤可包括如下具体步骤:1)从所述2D canvas网页元素的绘图指令中提取所述指令信息;2)根据第一预设指令转换规则和所述指令信息,生成与所述2D canvas网页元素的绘图指令对应的渲染命令;3)将所述渲染命令从传送到所述系统框架层。

[0226] 步骤S105:根据所述指令信息,生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令。

[0227] 作为一种优选的方案,本步骤可包括如下具体步骤:1)将所述指令信息从所述系

统框架层传送到系统本地层;2)在所述系统本地层中,根据将所述指令信息生成所述OpenGL绘图指令。

[0228] 步骤S107:将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述2D canvas网页元素的渲染结果。

[0229] 作为一种优选的方案,本步骤可包括如下具体步骤:1)获取所述2D canvas网页元素对应的上下文信息及渲染目标;2)根据所述上下文信息和所述OpenGL绘图指令,对所述渲染目标进行图形渲染;3)将所述渲染目标输入到系统的离屏缓冲区。

[0230] 此外,本申请提供的方法还包括:对所述OpenGL渲染视窗和系统的Webview视窗进行合成,形成所述2D canvas网页元素所在网页的渲染结果。

[0231] 请参考图7,其为本申请的2D canvas网页元素的渲染方法实施例的结构示意图。在本实施例中,实现本申请的2D canvas网页元素的渲染方法有四层结构,分别为:JavaScript托管模块层、桥接模块层、操作系统适配模块层、引擎模块层。下面对这些层次分别作简要说明。

[0232] 1)JavaScript托管模块层:主要用于托管HTML5 2D canvas的API(绘图指令)和针对这些API实现本方案,本层的功能包括但不限于:canvas元素替换、渲染指令拼接与传递、交互指令响应、回调函数的响应、canvas元素大小与位置的计算、多个canvas的层管理等。

[0233] 2)桥接模块层:主要用于JavaScript与操作系统层(即:上述系统框架层)代码(android:Java/iOS:OC)进行交互,有两种可以实现JavaScript与底层代码交互的方案,分别为:JS bridge、JS引擎。

[0234] 3)操作系统适配模块层:主要用于实现自定义的渲染视窗(即:上述OpenGL渲染视窗),该自定义的渲染视窗用于展现HTML5 2D canvas的渲染输出结果;以及,管理多个视窗用(自定义的渲染视窗与其他HTML5元素的渲染视窗),以合成最终的渲染结果。如果在上一层(桥接模块层)中使用了JS bridge的方案,那么针对所选取的JS bridge,还需要实现JS bridge的适配插件用以接收由JS bridge传递给系统层的交互或渲染指令,并且按照需要进行处理或者转发。

[0235] 4)引擎模块层:即上述系统本地层,主要用以创建、管理、销毁canvas实体(entity)与canvas 2d的上下文实体(context entity),根据接收到的渲染指令,调用相应的OpenGL ES API。在此层中还负责管理针对不同的canvas渲染到多个渲染层(render layer)的渲染目标(render target)中,最后将多个渲染目标通过OpenGL中的FBO技术合成到离屏渲染缓冲区。

[0236] 请参考图8,其为本申请的2D canvas网页元素的渲染方法实施例的具体流程图。图8所示的流程图对应上述方案的具体处理过程,结合图8能够更直观的理解上述方案的处理过程。下面结合图7对图8所示的具体流程图作简要说明。

[0237] 1)JavaScript托管模块层

[0238] 查找HTML5页面中是否有可供替换的2D canvas元素,如有且能成功替换,则新建一个自定义的DOM元素将之插入原DOM树,并将原canvas元素删除,后续所有对于原canvas元素的调用、操作都将被新插入的DOM元素托管。在接收到2D canvas的调用后,将该API调用转化为内部实现的一套命令,并通过定时器进行传入系统适配层。

[0239] 2)桥接模块层

[0240] 负责将JavaScript托管模块层生成的命令传给系统适配层。

[0241] 3)操作系统适配模块层

[0242] 接收到命令后,拼装用于渲染2D canvas的视图与其他HTML5元素的视图,并将命令传给引擎模块层。

[0243] 4)引擎模块层

[0244] 接收到命令后,进行解析,获取相应的2D canvas上下文(如若没有则新建);合成器获取一个渲染目标(如果没有则新建)并且分配给2D canvas上下文;由2D canvas上下文根据解析的命令结果调用渲染器(renderer)对应的API,渲染器调用相应的OpenGL ES API并将结果渲染至渲染目标中;合成器将多个渲染目标通过FBO的技术进行合成并且输入到系统的离屏缓冲区。

[0245] 此时,本申请提供的2D canvas网页元素的渲染方法的主要工作均完成,操作系统将获得渲染控制权,并将渲染结果展示到屏幕上。

[0246] 在上述的实施例中,提供了一种2D canvas网页元素的渲染方法,与之相对应的,本申请还提供一种2D canvas网页元素的渲染装置。该装置是与上述方法的实施例相对应。

[0247] 请参看图9,其为本申请的2D canvas网页元素的渲染装置实施例的示意图。由于装置实施例基本相似于方法实施例,所以描述得比较简单,相关之处参见方法实施例的部分说明即可。下述描述的装置实施例仅仅是示意性的。

[0248] 本实施例的一种2D canvas网页元素的渲染装置,包括:指令截获单元101,用于截获对2D canvas网页元素的绘图指令;指令传送单元103,用于将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层;OpenGL指令生成单元105,用于根据所述指令信息,生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令;OpenGL指令绘制单元107,用于将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗,形成所述2D canvas网页元素的渲染结果。

[0249] 可选的,所述指令截获单元101,具体用于通过所述2D canvas网页元素的替换DOM对象,截获所述2D canvas网页元素的绘图指令。

[0250] 可选的,所述指令传送单元103包括:

[0251] 指令信息提取子单元,用于从所述2D canvas网页元素的绘图指令中提取所述指令信息;

[0252] 渲染命令生成子单元,用于根据第一预设指令转换规则和所述指令信息,生成与所述2D canvas网页元素的绘图指令对应的渲染命令;

[0253] 渲染命令传送子单元,用于将所述渲染命令从传送到所述系统框架层。

[0254] 可选的,所述OpenGL指令生成单元105包括:

[0255] 指令信息传送子单元,用于将所述指令信息从所述系统框架层传送到系统本地层;

[0256] OpenGL指令生成子单元,用于在所述系统本地层中,根据将所述指令信息生成所述OpenGL绘图指令。

[0257] 请参考图10,其为本申请的电子设备实施例的示意图。由于设备实施例基本相似于方法实施例,所以描述得比较简单,相关之处参见方法实施例的部分说明即可。下述描述的设备实施例仅仅是示意性的。

[0258] 本实施例的一种电子设备，该电子设备包括：显示器1101；处理器1102；以及存储器1103，用于存储实现2D canvas网页元素的渲染方法的程序，该设备通电并通过所述处理器运行该2D canvas网页元素的渲染方法的程序后，执行下述步骤：截获对2D canvas网页元素的绘图指令；将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层；根据所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令；将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述2D canvas网页元素的渲染结果。

[0259] 本申请提供的2D canvas网页元素的渲染方法、装置及电子设备，通过在移动App层面上截获对2D canvas网页元素的绘图指令，将2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层，再根据所述指令信息生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令；并将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述2D canvas网页元素的渲染结果。采用本申请提供的方法，能够避免通过系统Webview对2D canvas网页元素进行渲染绘制，从而达到提高渲染性能的效果。

[0260] 本申请虽然以较佳实施例公开如上，但其并不是用来限定本申请，任何本领域技术人员在不脱离本申请的精神和范围内，都可以做出可能的变动和修改，因此本申请的保护范围应当以本申请权利要求所界定的范围为准。

[0261] 在一个典型的配置中，计算设备包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。

[0262] 内存可能包括计算机可读介质中的非永久性存储器，随机存取存储器(RAM)和/或非易失性内存等形式，如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。

[0263] 1、计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括，但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光盘只读存储器(CD-ROM)、数字多功能光盘(DVD)或其他光学存储、磁盒式磁带，磁带磁磁盘存储或其他磁性存储设备或任何其他非传输介质，可用于存储可以被计算设备访问的信息。按照本文中的界定，计算机可读介质不包括非暂存电脑可读媒体(transitory media)，如调制的数据信号和载波。

[0264] 2、本领域技术人员应明白，本申请的实施例可提供为方法、系统或计算机程序产品。因此，本申请可采用完全硬件实施例、完全软件实施例或结合软件和硬件方面的实施例的形式。而且，本申请可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质(包括但不限于磁盘存储器、CD-ROM、光学存储器等)上实施的计算机程序产品的形式。

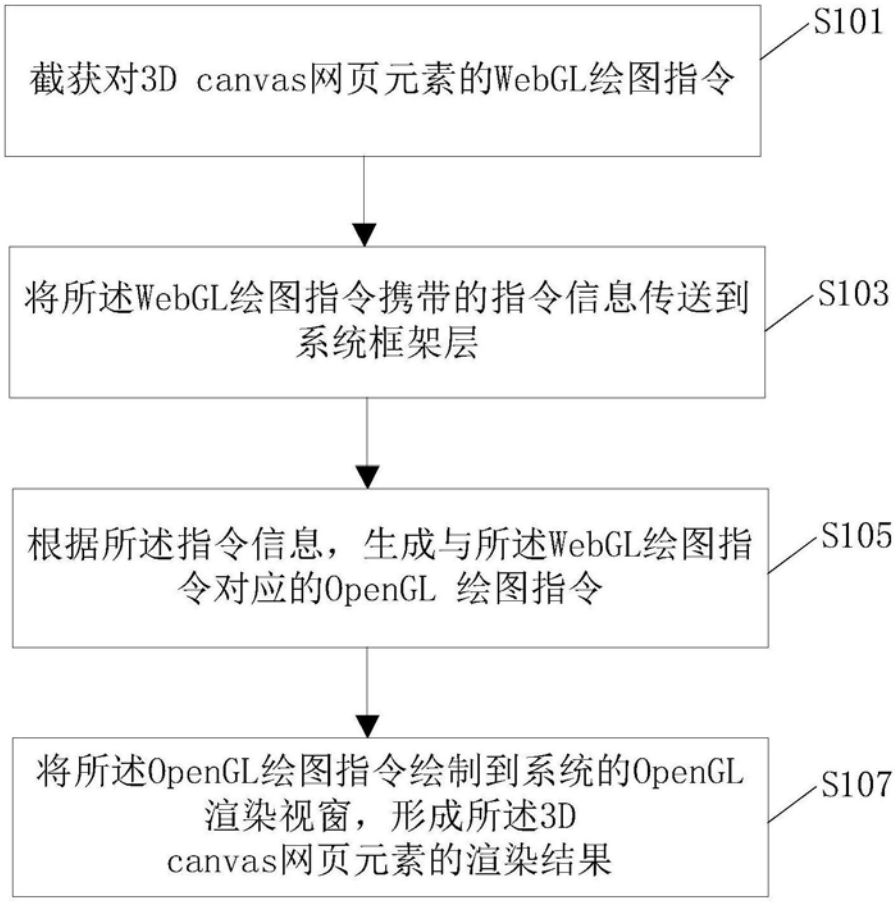

截获对3D canvas网页元素的WebGL绘图指令
S101
将所述WebGL绘图指令携带的指令信息传送到系统框架层
S103
根据所述指令信息，生成与所述WebGL绘图指令对应的OpenGL 绘图指令
S105
将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述3D canvas网页元素的渲染结果
S107

图1

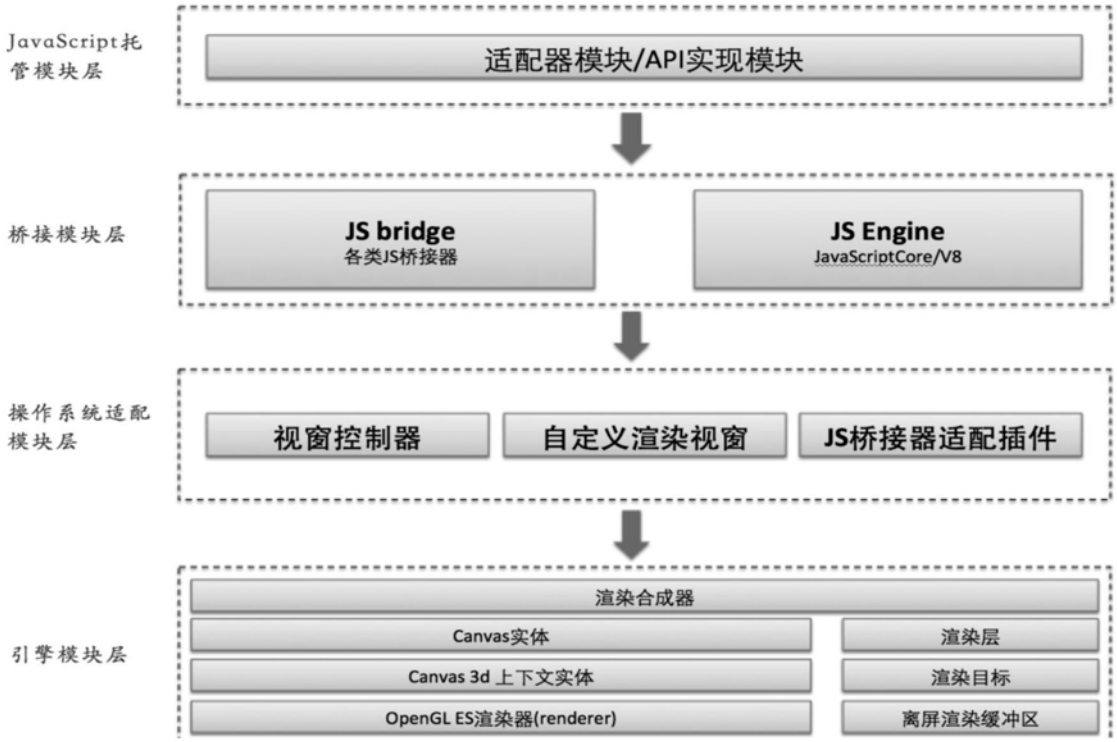

JavaScript托管模块层
适配器模块/API实现模块
桥接模块层
JS bridge
各类JS桥接器
JS Engine
JavaScriptCore/V8
操作系统适配模块层
视窗控制器
自定义渲染视窗
JS桥接器适配插件
引擎模块层
渲染合成器
Canvas实体
渲染层
Canvas 3d 上下文实体
渲染目标
OpenGL ES渲染器(renderer)
离屏渲染缓冲区

图2

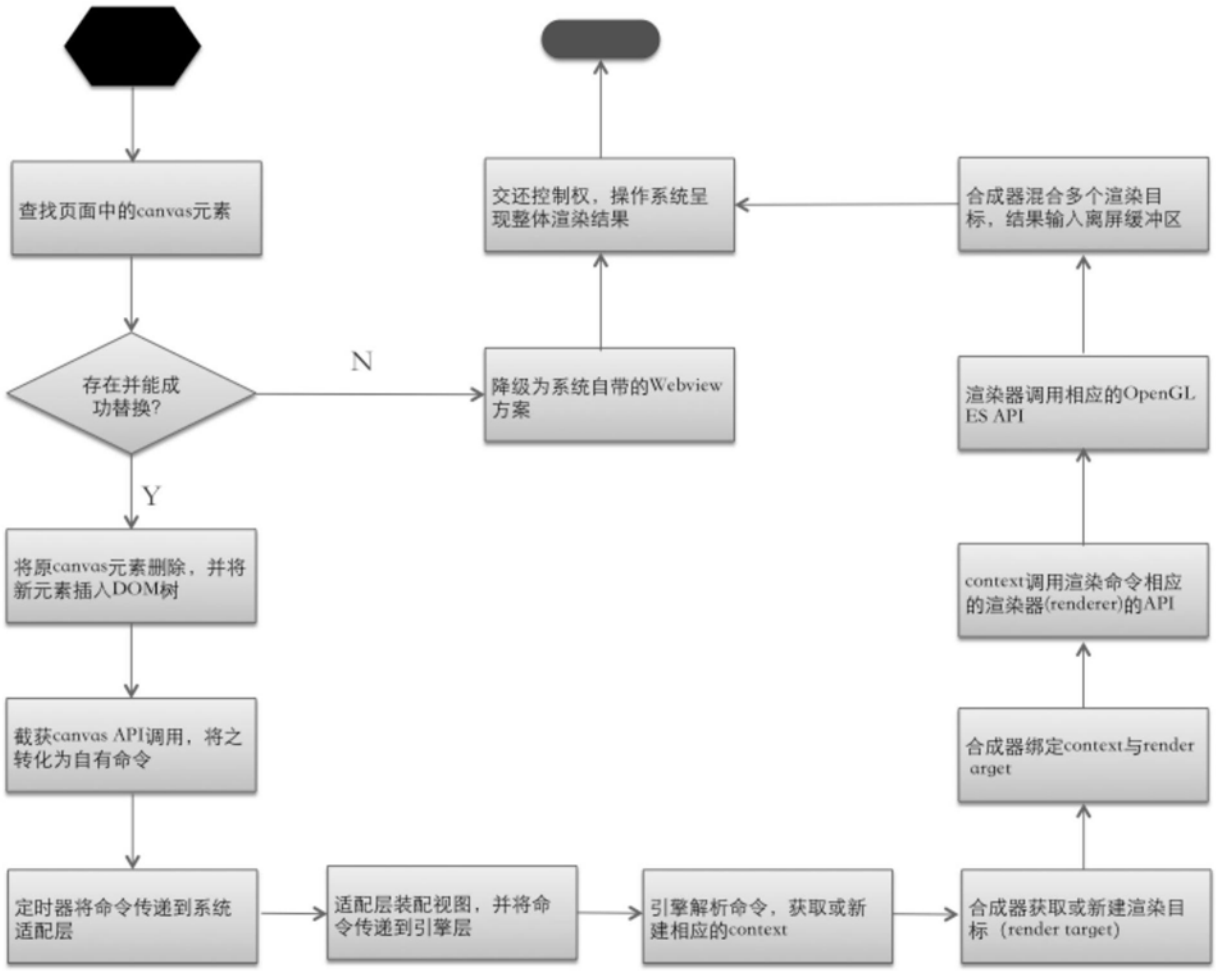

查找页面中的canvas元素
存在并能成功替换?
N
降级为系统自带的Webview方案
Y
将原canvas元素删除，并将新元素插入DOM树
截获canvas API调用，将之转化为自有命令
定时器将命令传递到系统适配层
适配层装配视图，并将命令传递到引擎层
引擎解析命令，获取或新建相应的context
合成器获取或新建渲染目标（render target）
合成器绑定context与render arget
context调用渲染命令相应的渲染器(renderer)的API
渲染器调用相应的OpenGL ES API
合成器混合多个渲染目标，结果输入离屏缓冲区
交还控制权，操作系统呈现整体渲染结果

图3

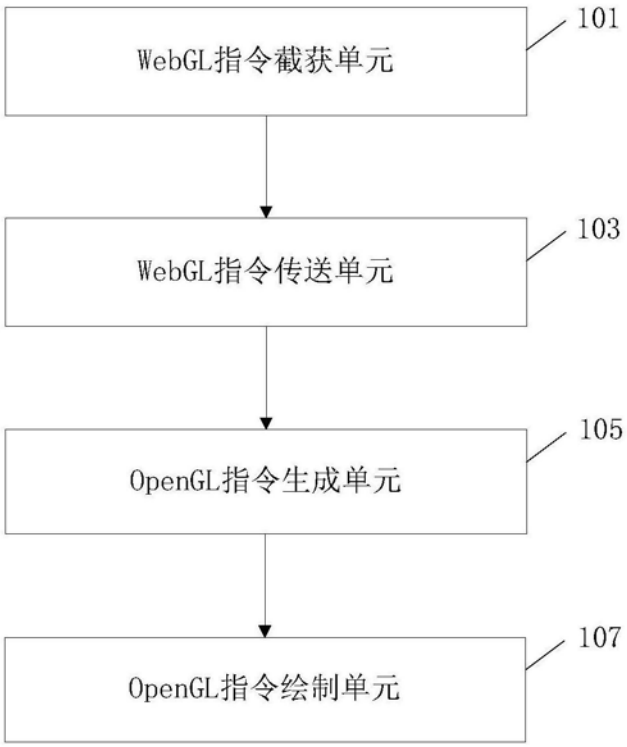

WebGL指令截获单元
101
WebGL指令传送单元
103
OpenGL指令生成单元
105
OpenGL指令绘制单元
107

图4

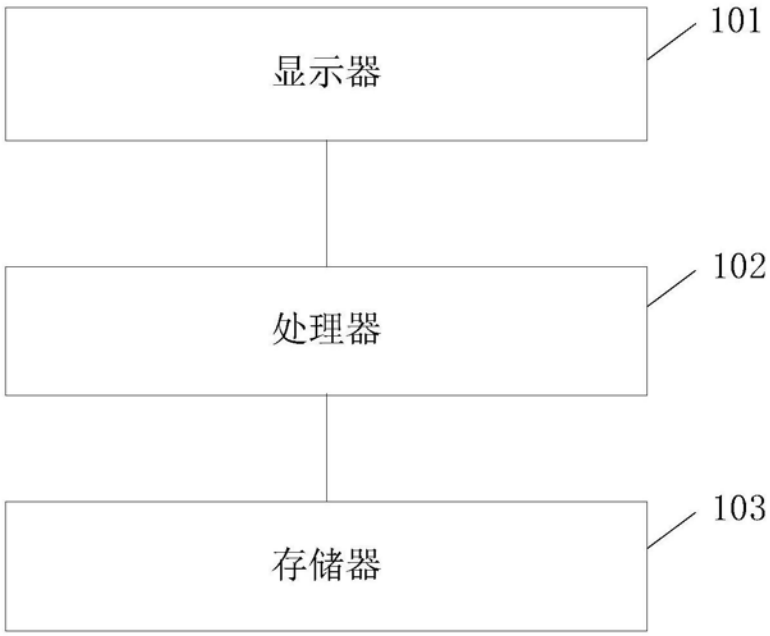

显示器
101
处理器
102
存储器
103

图5

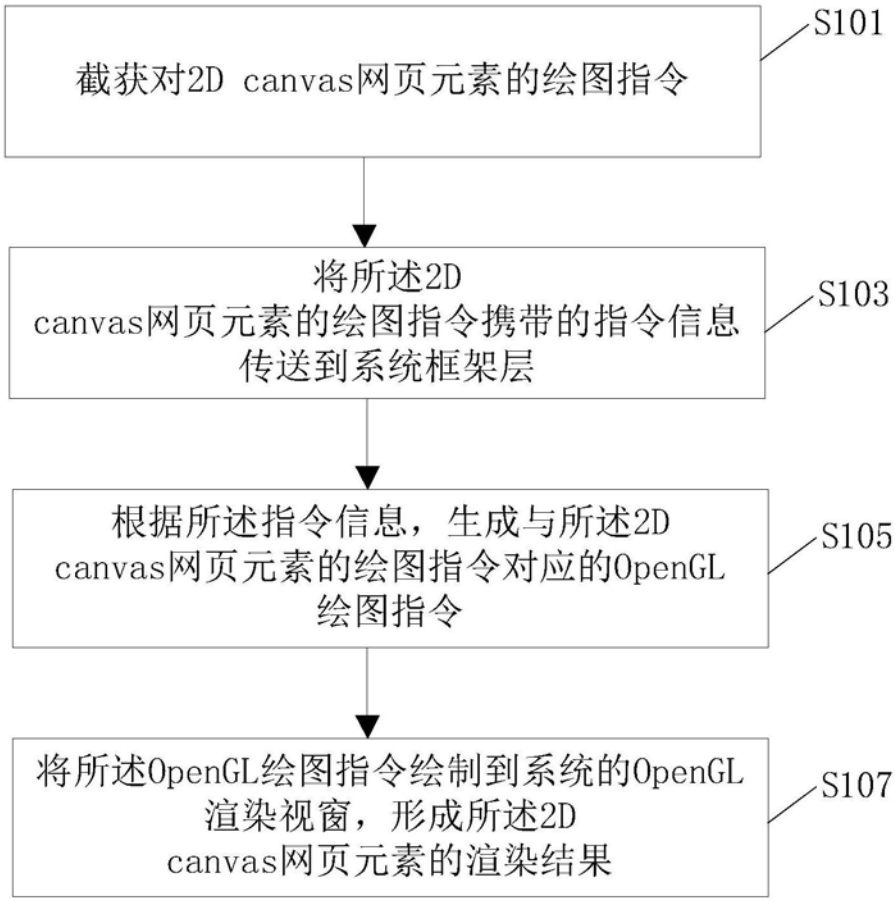

截获对2D canvas网页元素的绘图指令
S101
将所述2D canvas网页元素的绘图指令携带的指令信息传送到系统框架层
S103
根据所述指令信息，生成与所述2D canvas网页元素的绘图指令对应的OpenGL绘图指令
S105
将所述OpenGL绘图指令绘制到系统的OpenGL渲染视窗，形成所述2D canvas网页元素的渲染结果
S107

图6

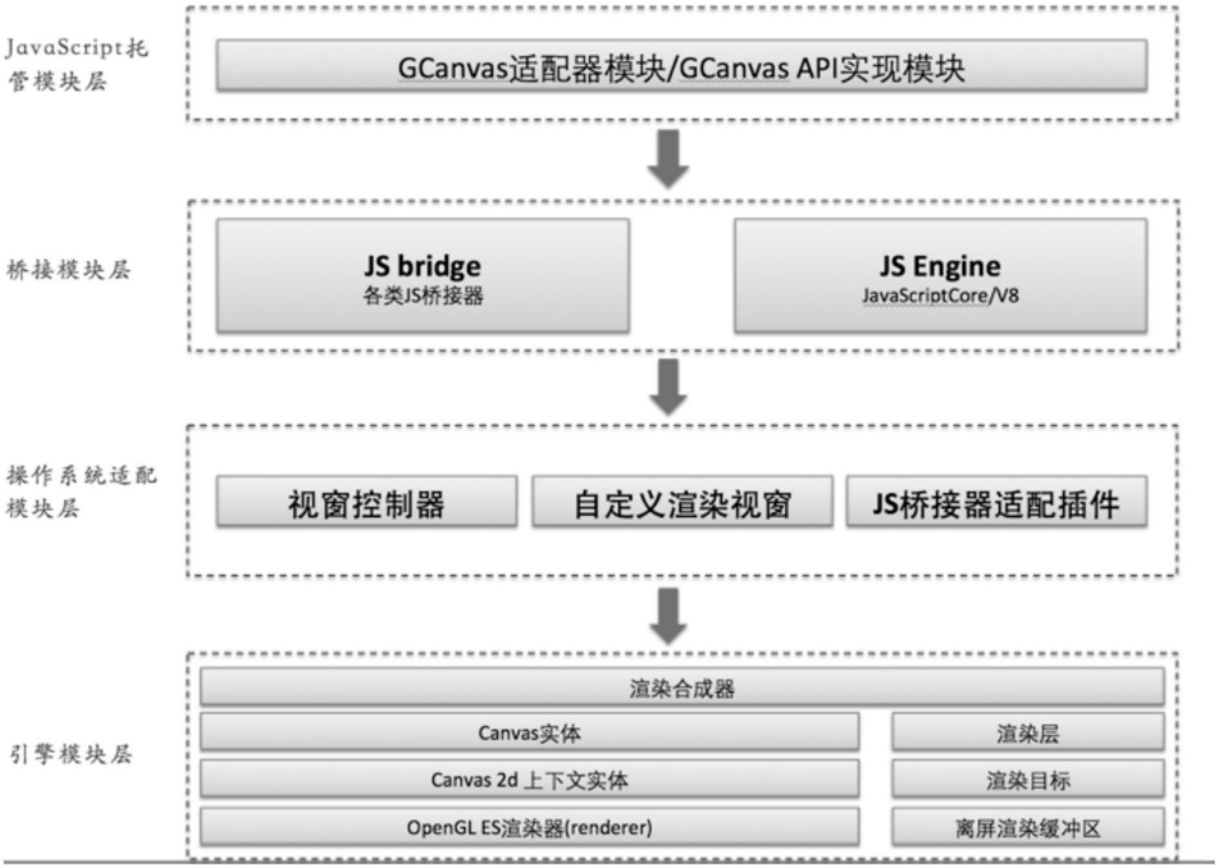

JavaScript托管模块层
GCanvas适配器模块/GCanvas API实现模块
桥接模块层
JS bridge
各类JS桥接器
JS Engine
JavaScriptCore/V8
操作系统适配模块层
视窗控制器
自定义渲染视窗
JS桥接器适配插件
引擎模块层
渲染合成器
Canvas实体
渲染层
Canvas 2d 上下文实体
渲染目标
OpenGL ES渲染器(renderer)
离屏渲染缓冲区

图7

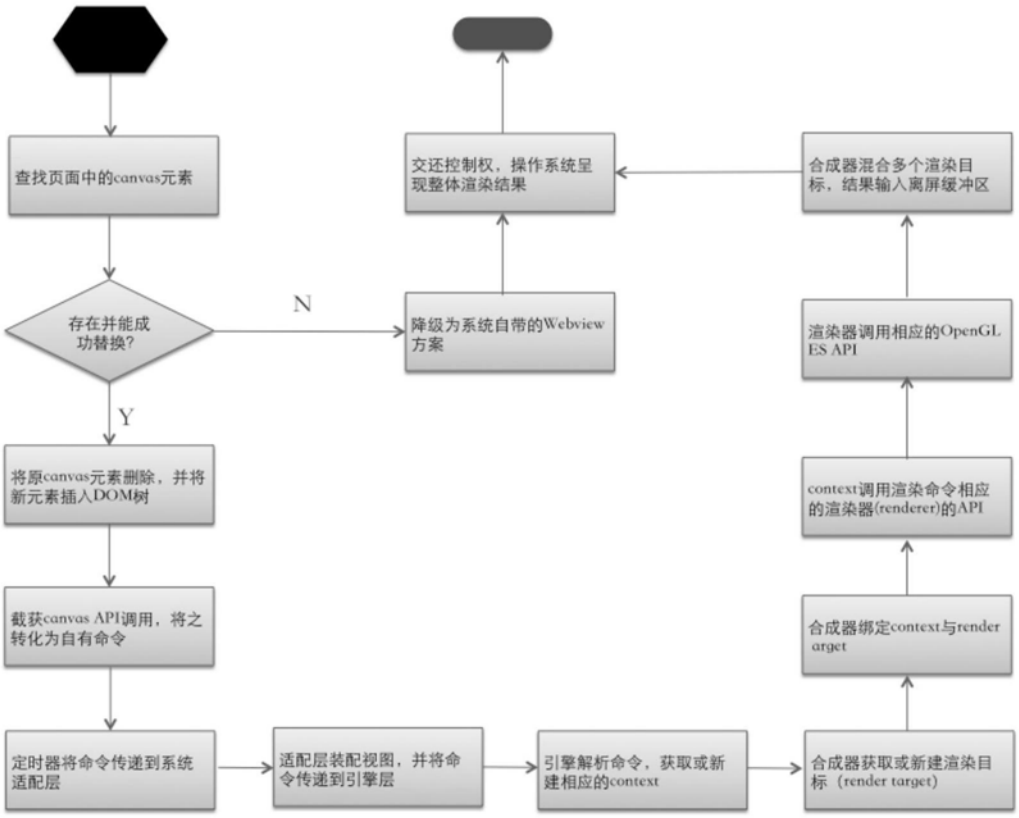

查找页面中的canvas元素
存在并能成功替换?
N
Y
降级为系统自带的Webview方案
将原canvas元素删除，并将新元素插入DOM树
截获canvas API调用，将之转化为自有命令
定时器将命令传递到系统适配层
适配层装配视图，并将命令传递到引擎层
引擎解析命令，获取或新建相应的context
合成器获取或新建渲染目标（render target）
合成器绑定context与render arget
context调用渲染命令相应的渲染器(renderer)的API
渲染器调用相应的OpenGL ES API
合成器混合多个渲染目标，结果输入离屏缓冲区
交还控制权，操作系统呈现整体渲染结果

图8

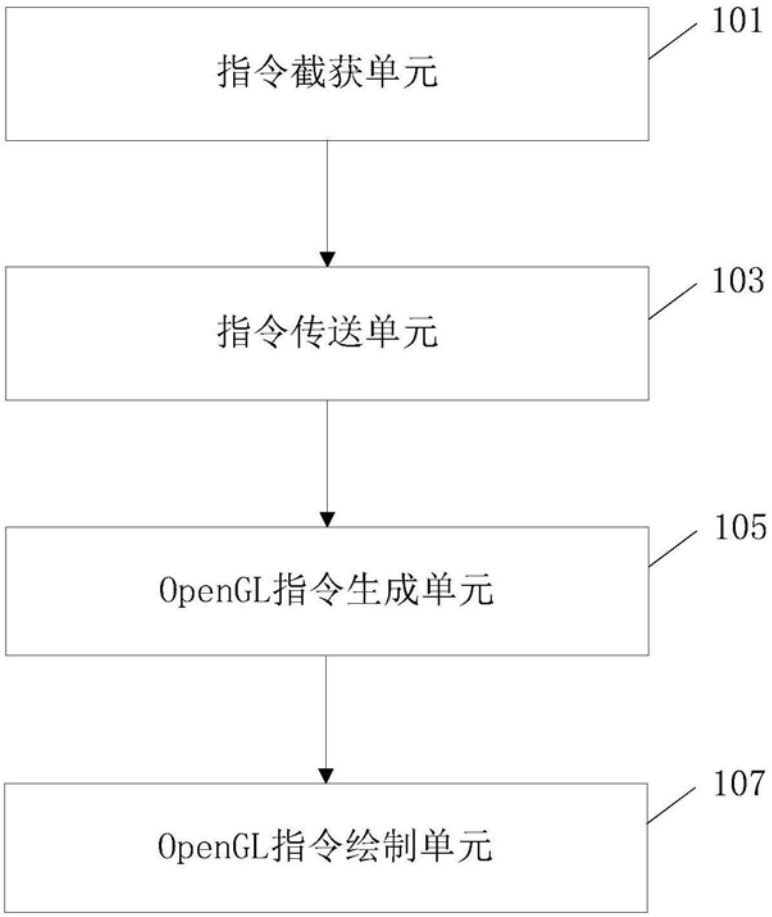

101
指令截获单元
103
指令传送单元
105
OpenGL指令生成单元
107
OpenGL指令绘制单元

图9

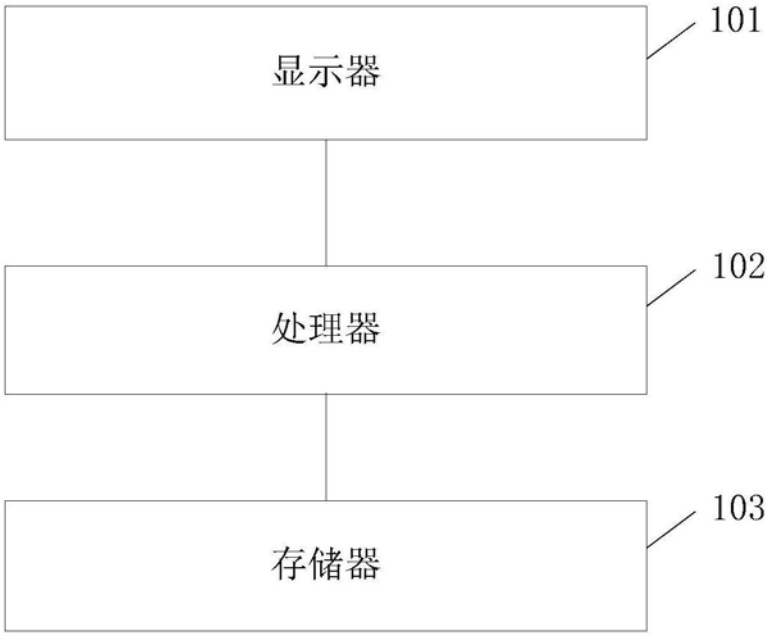

101
显示器
102
处理器
103
存储器

图10