**Protoss引擎**

**软件项目计划书**

2013-06-09

目录

[1项目概述 3](#_Toc359223320)

[1.1 背景 3](#_Toc359223321)

[1.2 目的 3](#_Toc359223322)

[1.3 项目目标 4](#_Toc359223323)

[2 项目团队组织 4](#_Toc359223324)

[2.1 组织结构 4](#_Toc359223325)

[2.2 人员分工 5](#_Toc359223326)

[2.3 协作与沟通 5](#_Toc359223327)

[3 实施计划 5](#_Toc359223328)

[3.1 风险评估及对策 5](#_Toc359223329)

[3.2 工作流程 6](#_Toc359223330)

[3.3 总体进度计划 6](#_Toc359223331)

[4 支持条件 7](#_Toc359223332)

[5 附录 7](#_Toc359223333)

[5.1 引擎框架描述 7](#_Toc359223334)

[5.2 引擎API使用示例（待修改） 10](#_Toc359223335)

# 1项目概述

## 背景

目前公司的UI产品主要包括Launcher，解锁，壁纸三类。

其中Launcher使用的是libgdx引擎，libgdx对3D的支持仍未正式发布，所以3D Widget的研发进度较慢。

解锁因载入速度原因，基本没有使用OpenGL。

壁纸使用的引擎为JPCT，但因为引擎未开源，扩展难度大。

这三类产品中，壁纸对3D的要求最高，因为壁纸的交互少，画面一定要精美才可以吸引用户，打开市场。但是本公司目前在3D方面的积累不足，工程师和美术的沟通成本高，导致了3D壁纸的开发成本非常巨大，粗略估计，一款3D壁纸的开发，往往需3-4个星期的时间。

如果有一款能完全掌握的3D引擎和PC端设计工具，可以大幅缩短公司3D产品的开发周期，例如可以将3D壁纸的纯开发周期从3-4个礼拜，缩短到3-4天。同时，拥有此类核心技术，长远来看，对公司的UI产品线的横向（包括直接出售引擎）纵向扩展都会有很大的帮助。

## 目的

短期：加快3D产品的开发速度，降低工程师3D工程难度。

中期：引入桌面特效工具，增加特效模板，增加界面设计师的参与度，降低工程师和设计师的沟通成本，大大缩短开发周期

长期：加强公司在3D方面的研发能力。

## 项目目标

项目可以分为两个阶段：

* 1. 从无到有的设计一个3D壁纸引擎，其架构会参考现有的桌面和移动端游戏引擎，并基于壁纸的应用需求增加裁减模块。引擎的将以C++为基本开发语言，为中期的PC端设计器作准备。引擎的提供一系列的应用接口，降低3D壁纸的工程难度。这个阶段的验收标志点以可以使用本引擎通过编程方式做出花园动态壁纸为准。
  2. 从优秀到卓越。提供设计器和特效模板框架，界面设计师可以直接在PC调整设计，并直接得到预览，设计器可以直接倒出动态壁纸应用程序，整个流程基本不需要工程师的参与。

# 项目团队组织

## 组织结构

基于Scrum软件工程开发方式，分为：

1. Product owner. 确定产品的功能和完成时间，在每个Sprint开始前，Product owner可以修改需求的内容和优先级，每个Sprint结束时，Product owner可以决定是否接受Team的工作成果
2. Scrum master. 负责监督整个Scrum项目进程，调整项目计划；确保开发团队成员的能力能够胜任产品的开发；促进团队中不同角色的成员间充分交流和沟通，并负责为项目的进行扫清障碍；保证开发团队不受外力的干扰和阻扰；掌握产品开发进度，参与每日Scrum会议、Sprint计划会议和 Sprint评审会议。
3. Team. 负责开发，测试，文档设计。

## 人员分工

项目的中前期，由于集中在引擎端，只需要工程师

Product owner: 前期由宏伟代理，后期转交给小关。

Scrum master：苏宇

Team：宏伟，苏宇，纪昊

项目的中后期，增加设计器的实现，需要设计师的加入。这部分人员安排待定。

## 协作与沟通

在Sprint中，每天都会有一个几分钟的小会议，不超过15分钟。在会议上，每个团队成员需要回答三个问题：今天你完成了哪些工作？明天你打算做什么？完成你的目标是否存在什么障碍？（Scrum Master需要记下这些障碍）。

每一个Sprint完成后，都会举行一次Sprint回顾会议，在会议上所有团队成员都要反思这个Sprint。举行Sprint回顾会议是为了进行持续过程改进。会议的时间限制在4小时。

# 实施计划

## 风险评估及对策

风险一：阶段一未能实现使用本引擎未足够完善达到做出花园壁纸的级别

对策：使用JPCT引擎来制作花园壁纸，苏宇会在sprint1基于JPCT完成花园壁纸

## 工作流程

工作流程采用scrum模型进行开发。

以一周为一个迭代长度，周首制定工单，由成员自由认领，周尾做冲刺回顾。

每个冲刺阶段完成后都可产生一个可交付成果。

## 总体进度计划

第一版本开发周期从2013年6月7号开始到2013年7月12号为止，总共分为5个sprint。

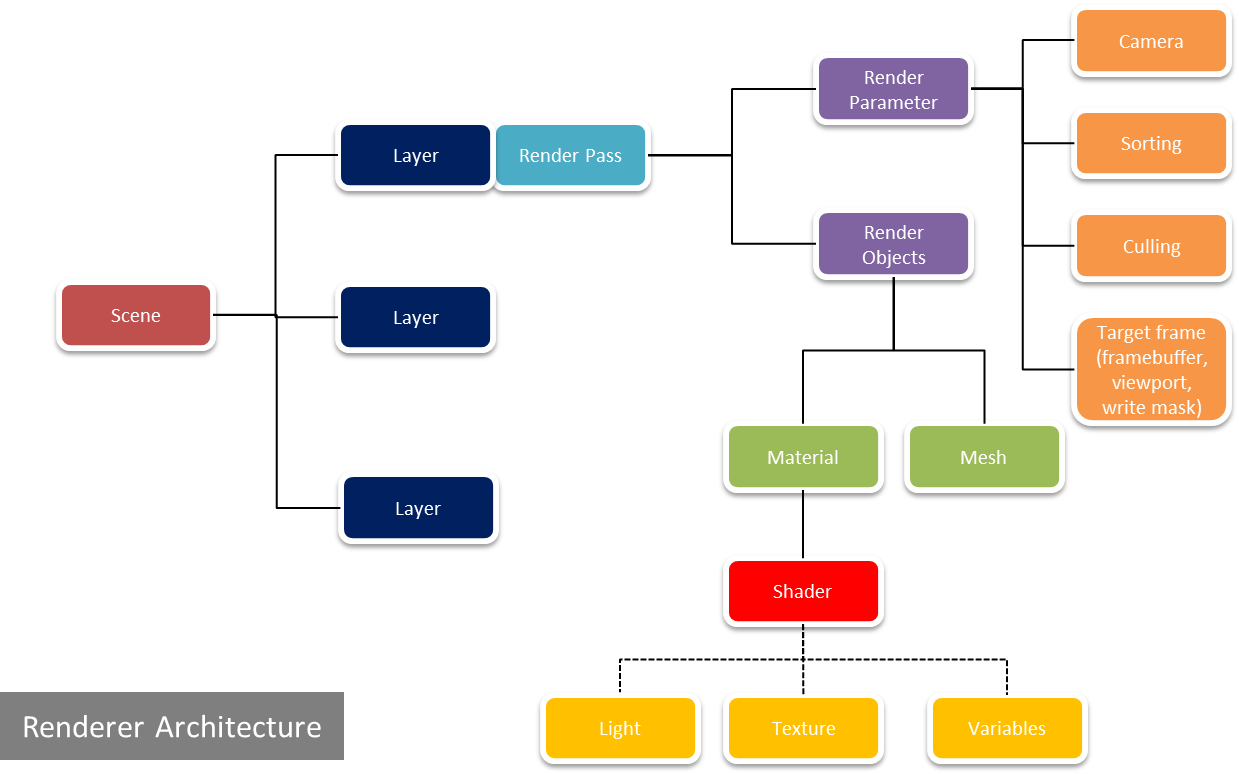
|  |  |
| --- | --- |
| Sprint 0  6.7 – 6.14 | 统一开发协作理念和工程规范，建立scrum机制，分配角色，设计工程架构，制作架构设计文档，分配sprint 0任务，编写基本框架 |
| Sprint 1  6.17 – 6.21 | - 完善文档 (lihw)  - mesh file format and loader (suyu)  - resource management (lihw)  - camera (jihao)  - render target (suyu)  - 基于JPCT制作花园动态壁纸 (jihao) |
| Sprint 2  6.24 – 6.28 |  |
| Sprint 3  7.1 – 7.5 |  |
| Sprint 4  7.8 - 7.12 |  |

# 支持条件

# 附录

## 引擎框架描述

基本渲染引擎架构



Scene: 场景。场景包括了所有的可见物体，特效，及其动画的动态数据结构。一个动态壁纸通常只含有一个场景，如果未来将引擎扩展至其他应用，可以在场景之上再加入一个层次，用于实现多场景。

Layer: 层。层保证物体以批次的方式绘制，这个层的绘制前，上一个层的绘制必然结束。一个场景可以包含任意多个层。

Render Pass: 渲染批次。指的是一个从物体3D模型到帧缓冲的运算过程。渲染批次由渲染参数和待渲染物体组成，参数包括Camera，Sorting，Culling，Target framebuffer。物体有Mesh和Material组成。渲染批次同时也是Technique（技术，用于特效）的组成单位。

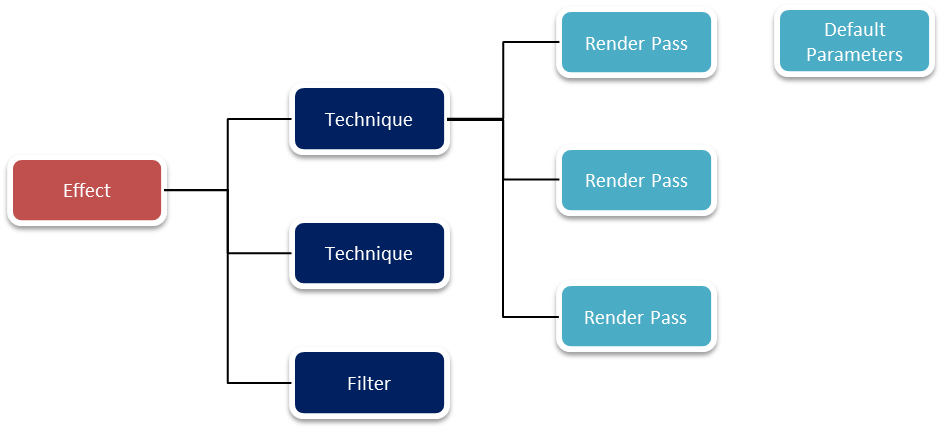
Camera: 相机参数，确定了相机的位置和投影方式，它给出两个4x4矩阵，相机矩阵和投影矩阵。

Sorting: 渲染物体排序方式，如z-sorting, sorted based on material等，排序方式可以叠加，按照顺序依次排序。例如先按照深度排序，再按照material排序。

Culling: Culling方式，如face culling和camera culling，还有其他BSP等culling方式，Culling的方式可以叠加，叠加效果与叠加顺序无关。

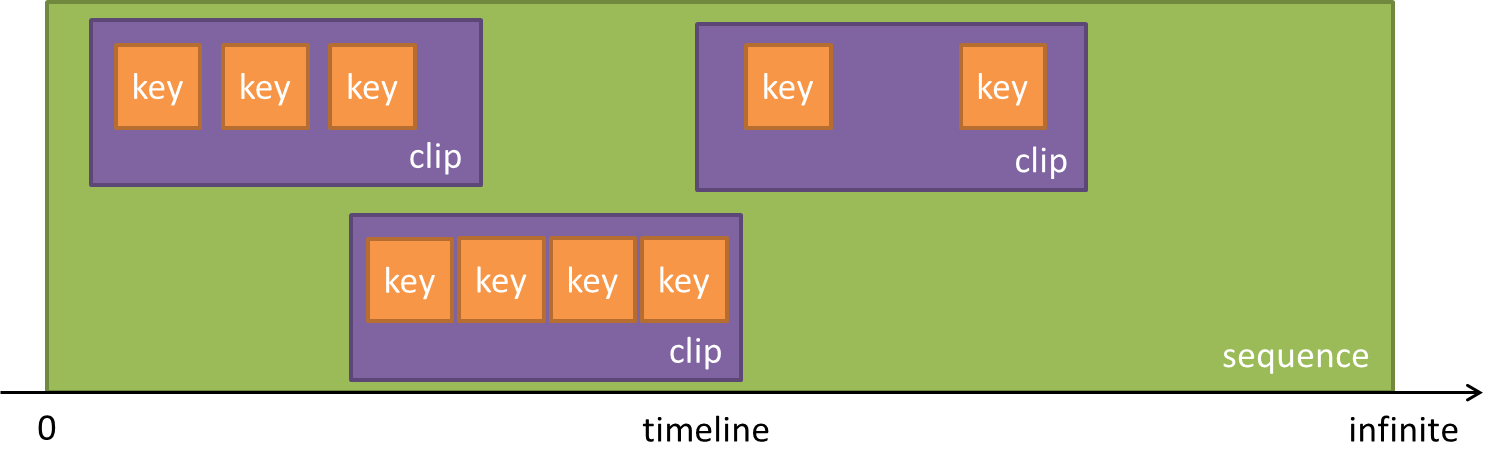
Target framebuffer: 渲染目标缓冲。其属性包括framebuffer object（帧缓冲），viewport（视窗大小）和write mask（写入模板）。

特效系统架构



通常特效Effect包含一个或多个Technique。每一个Technique是实现同一特效的不同方法。Technique由多个Render Pass组成和一组默认参数。

动画结构



动画结构从上至下分别为sequence, clip和key。Key就是一个关键帧，一个关键帧是针对一个物体中某种属性的值。几个关键帧组合成一个clip。Clip根据时间或一个变量插值关键帧形成一组连续变化的属性值，这是推动动画的基本数据结构。Clip组合成sequence。比如一个物体移动的动画，其实是三个clip的组合后的sequence，三个clip分别对应x，y和z值在时间轴上得改变。Clip可以放在sequence上的任意位置，当时间或者驱动变量到达这个Clip开始位置，Clip开始驱动对应的数据改变，从而影响绘制结果，形成动画。

## 引擎API使用示例（待修改）

**Animation:**

|  |
| --- |
| Animation \*animation = Animation::create(“filepath/animation.ani”);  Object3D \*obj = Object3D::create(\*animation);  obj.animate(time); |

**Scene:**

|  |
| --- |
| PScene\* scene = context->getScene();  PLayer\* layer = pNew(PLayer(scene));  PMesh\* mesh = context->getResourceManager()->retainMesh("object.mes");  PTexture\* texture = context->getResourceManager()->retainTexture("texture.png");  PShader\* shader = context->getResourceManager()->retainShader("vert.glsl", "frag.glsl");  PDrawable\* drawable = pNew(PDrawable(layer));  drawable->getMaterial()->setShader(shader);  drawable->getMaterial()->addTexture(texture);  drawable->getGeometry()->setMesh(mesh);    layer->getCamera()->setPosition(10, 0, 0);  layer->getCamera()->setOrientation(0, 0, 0, 0, 1, 0);  layer->getCamera()->setProjection(P\_CAMERA\_PROJECTION\_PERSPECTIVE);  layer->getCulling()->clear();  layer->getCulling()->addCulling(P\_CULLING\_DEFAULT);  layer->getSorting()->clear();  layer->getSorting()->addSorting(P\_SORTING\_DEFAULT);    layer->getRenderTarget()->setViewport(-1, -1, -1, -1);  layer->getRenderTarget()->setStencilWritable(false);  layer->getRenderTarget()->setFrameBuffer(P\_NULL); |

**Material & Light:**

|  |
| --- |
| Object3D \*obj1 = Object3D::create(“filepath/object.obj”, “filepath/object.mtl”);  PoingLight \*pLight = PointLight::create();  pLight->setColor(color);  pLight->setPos(1, 1, 1);  pLight->setIntensity(128);  DirectionLight \*dLight = DirectionLight::create();  dLight->setColor(color);  dLight->setDirVector(1 , 1, 1);  dLight->setIntensity(128); |

**Renderer:**

|  |
| --- |
| renderer->render(deltaTime); |

**Resource:**

|  |
| --- |
| Int textureId = Resourse::loadTexture(“filepath/texture.png”);  obj->bind(textureId);  Mesh \*mesh = Resourse::loadMesh(“filepath/mesh.obj”);  Shader \*shader = Resource::loadShader(“filepath/VertexShader.glsl”, “FragmentShader.glsl”);  Obj->setShader(\*shader); |