跨平台HMI工具介绍

大连致胜科技有限公司

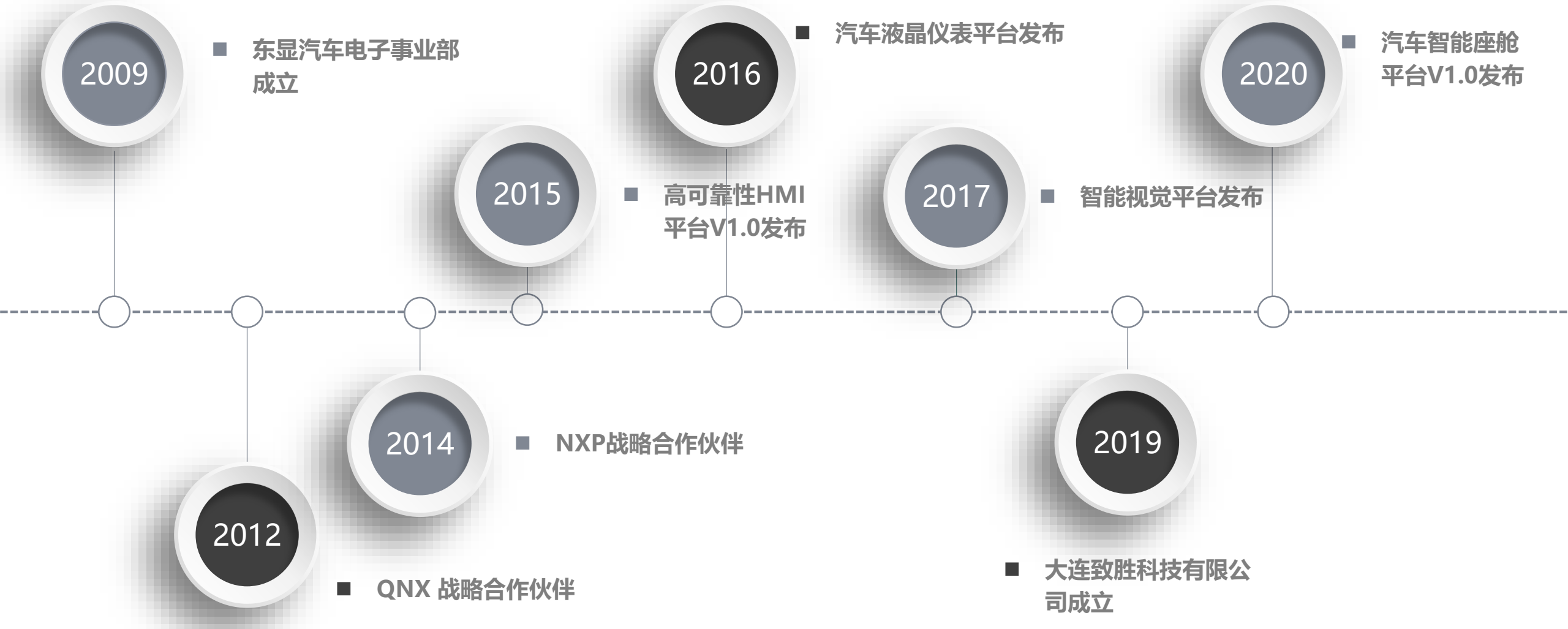
公司介绍

大连致胜科技有限公司前身是大连东显电子汽车电子事业部，定位高可靠性图形图像的嵌入式解决方案。经过10年的发展，事业部已经在工控、医疗和汽车电子行业展露头脚。为了更好的服务客户，面向未来的发展，事业部在2019年注册成为大连致胜科技有限公司。

大连致胜科技的研发团队拥有一流的算法能力、工程能力和完整的产品设计能力。公司以实时操作系统为核心技术，为客户提供从硬件设计、操作系统开发、核心技术授权到应用软件定制的整体解决方案和服务。

公司通过IATF16949和CMMI3质量管理体系，为客户提供专业的产品和技术服务。

发展历程



合作伙伴

01 OS

QNX Distributors & Resellers

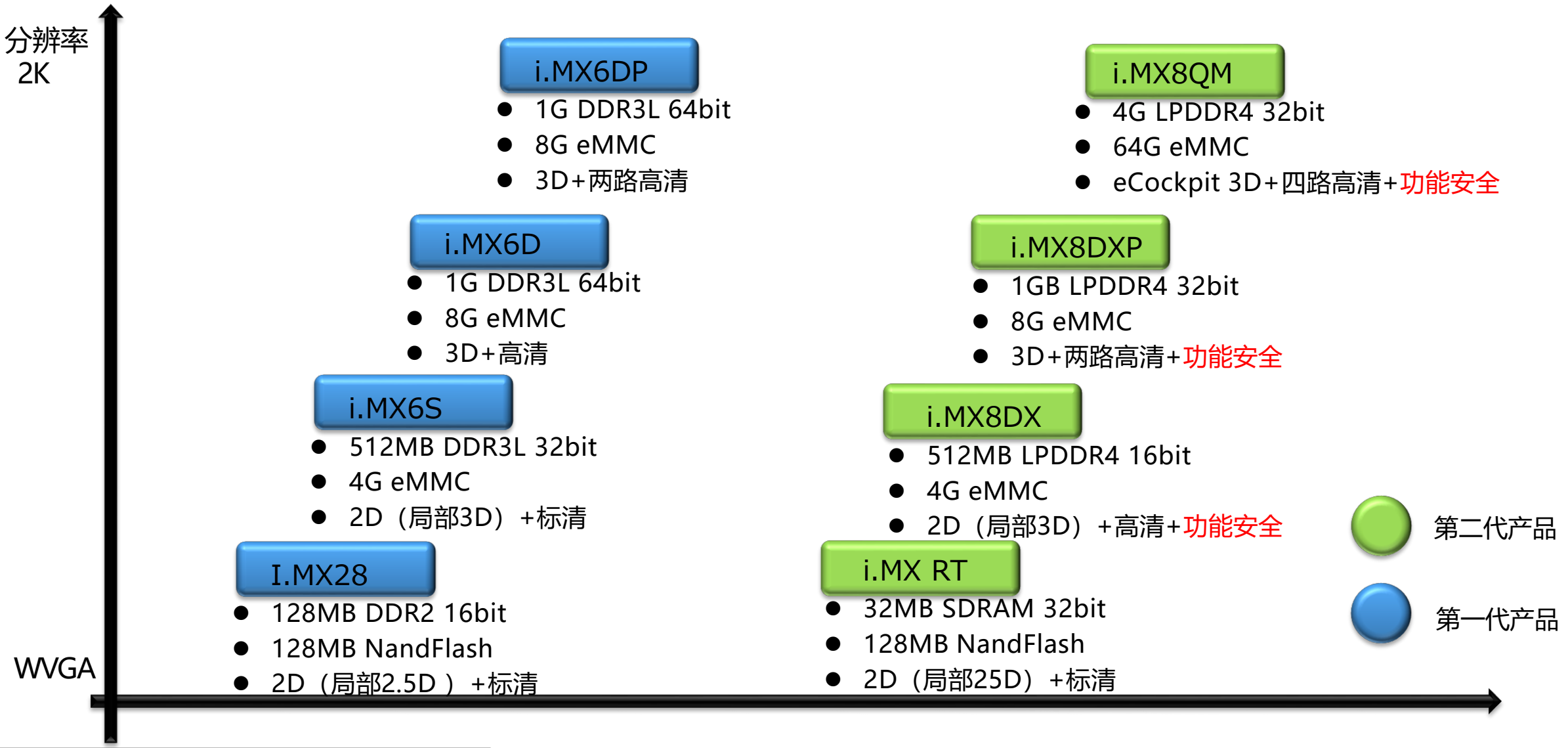
02 Silicon

NXP PARTNER PROGRAM

03 Tools

Crank software inc.

产品线



业务范围

系统开发

提供操作系统开发服务，满足客户多样化需求

应用定制

为产品提供应用程序开发以及提供产品周边增值软件开发

车规核心板

满足车规量产级的核心板，加速产品研发，降低技术风险

核心技术授权

提供互联互通、OTA等核心技术授权，缩短开发周期

测试与支持

以量产为目标的全方位测试服务，并在产品全生命周期内提供支持服务

HMI工具

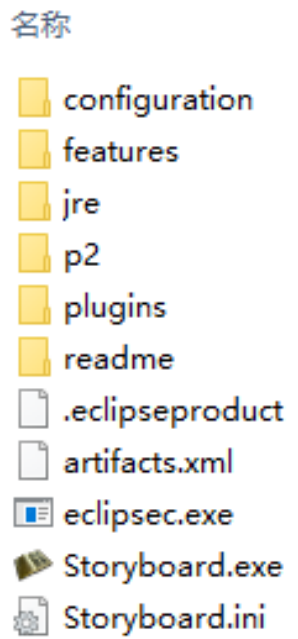
简介

Crank StoryBoard 是嵌入式图形界面开发套件，包含软件开发工具Storyboard Designer 和运行引擎Storyboard Engine。具有空间占用小，跨平台，可支持功能插件扩展和定制，并支持可视化开发，易学易上手等特点。

Storyboard Designer 使UI设计工程师能够轻松地对产品的界面进行设计，然后部署到嵌入式目标硬件平台上。UI设计工程师可以完全控制设计出的UI界面，而无需等待嵌入式系统工程师来将UI界面部署到硬件平台上。实现设计与开发并行。

Storyboard Engine 可将Storyboard Designer中开发设计的UI内容在嵌入式设备硬件平台上呈现。该引擎专门用于解决将丰富绚丽的UI引入资源能力有限的嵌入式设备的问题。Storyboard Engine插件架构使您可以轻松配置自己所需要的组件，从简单的显示到动画和交互式体验。它使您能够在所有产品中使用一个UI解决方案，因为它与设备上使用的硬件和操作系统无关。

文件安装结构



Crank

支持以下嵌入式系统(For MPU)

Android, IOS, Linux OS, Mac OS, QNX, VX Works, WEC 2013, Windows, Wince, and WinCompact 7...

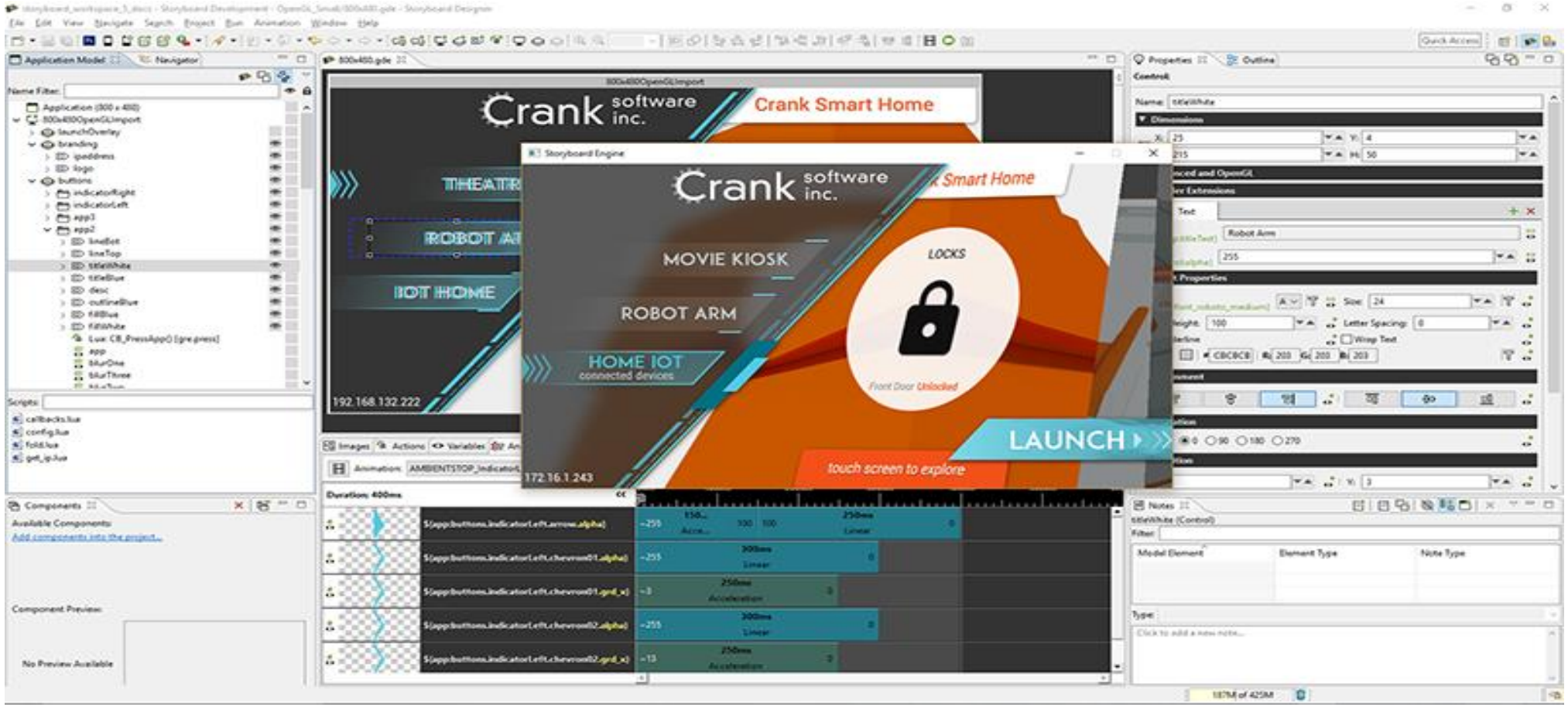
支持以下小型嵌入式系统(For MCU)

FreeRTOS, Integrity, iTron, MQX, Segger, ThreadX, µClinux, µC/OS-II, µC/OS-III, VX Works, and XIP Linux...



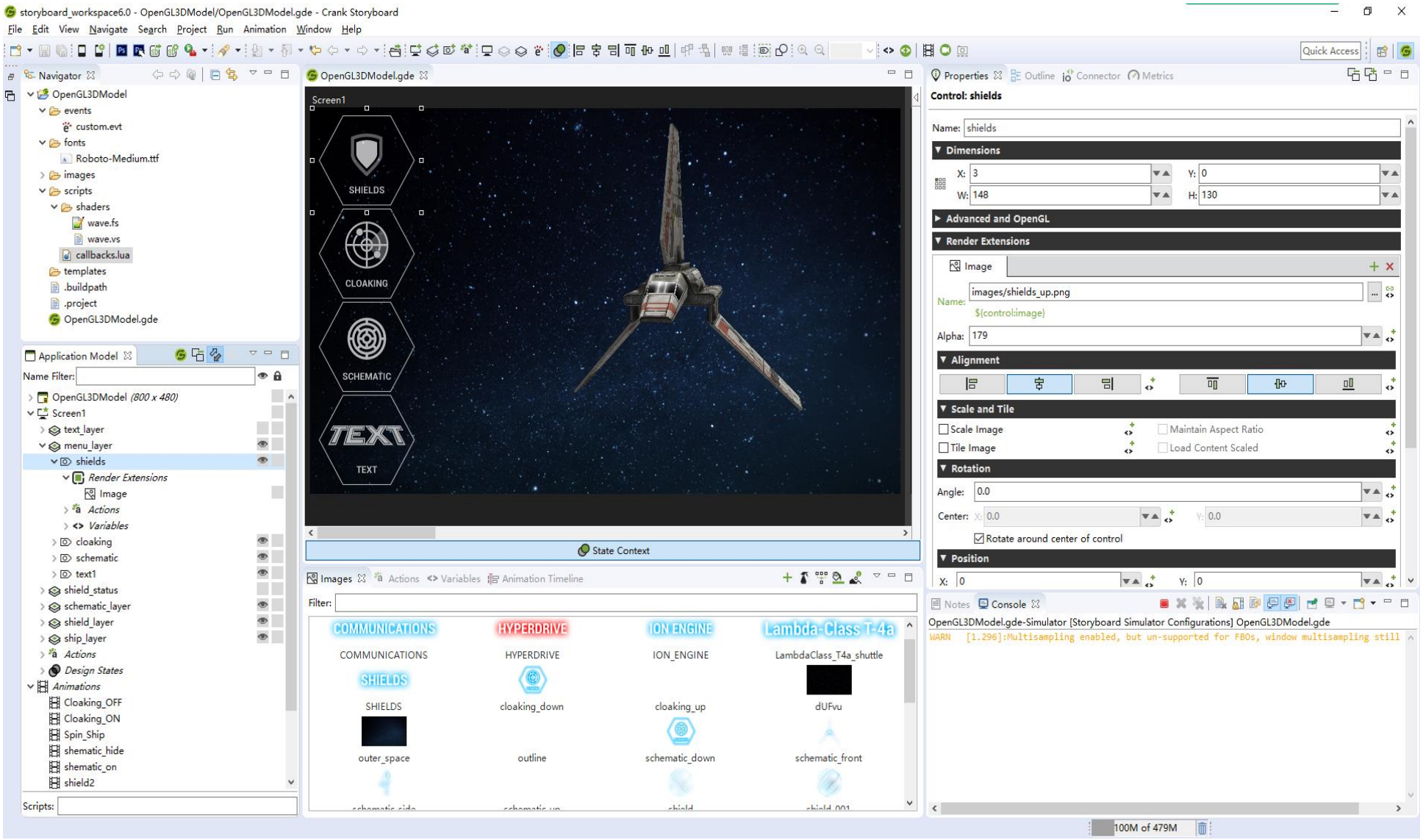
一 工具界面介绍

StoryBoard Designer采用一种以设计为中心的方法来创建嵌入式用户界面。用户在桌面开发环境中创建界面，可以直接从Photoshop等图形设计工具中导入内容，也可以通过拖放标准图像资源来构建屏幕。



Designer

二 工具布局介绍
1 顶部工具栏
2 左侧工程代码
3 左侧UI布局结构
4 中间实现效果
5 底部资源展示
6 右侧属性结构



三 控件介绍

应用程序组织为屏幕(**Screen**)、层(**Layer**)、组(**Group**)和控件(**Control**)的层次结构，统称为模型元素。模型元素只提供容器的位置、范围信息。显示内容由描画控件(**Render Extension**)提供。

Application Model

- Application
  - Screen
    - Layer
      - Group
      - Control
        - Render Extensions
        - Render Extensions
      - Table
        - Tabel Cell

Render Extensions

3D Model	3D模型
Arc	弧形控件
Canvas	画布
Circle	圆形控件
External	外部视频
Fill	颜色填充
Image	图片
Polygon	多边形
Rectangle	矩形
Text	文字

嵌套制约关系

四 事件(Event)和动作(Action)

事件触发动作。事件触发方法为 gre.send_event()。

1 事件类型：

① 标准事件：　　标准输入事件，如鼠标或触摸按下、抬起、移动、焦点变换等事件。

② UI事件：　　界面切换、动画开始结束、程序启动结束，定时器等。

③ 自定义事件：　　用户自定义消息。

2 动作类型：

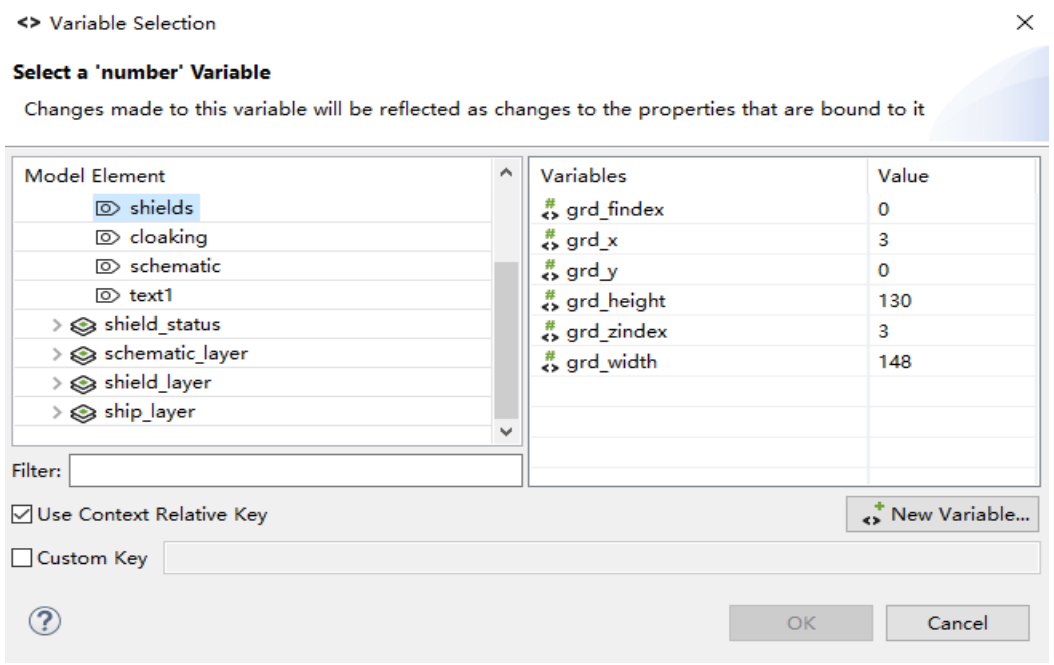
① crank内部动作：　动画处理动作、界面跳转动作、鼠标/焦点变换动画动作等。

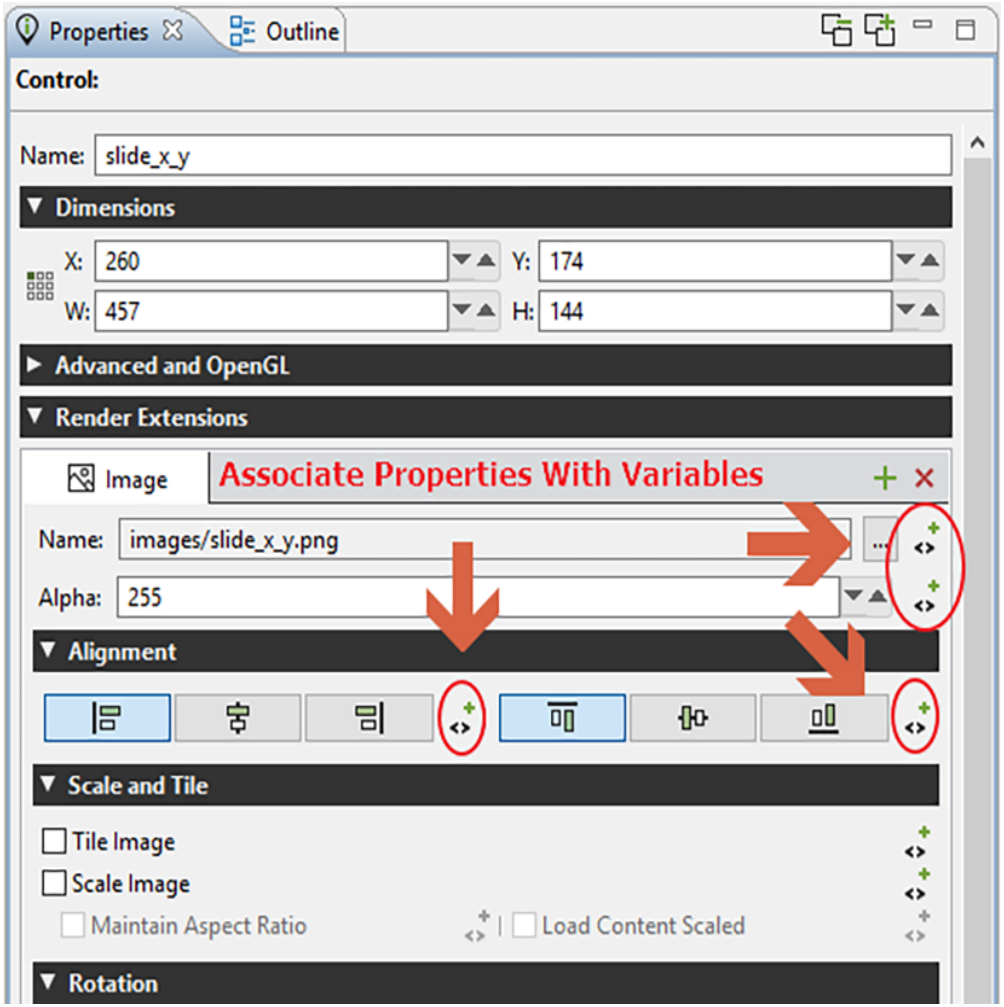
② 自定义动作：　　针对不同事件，用户可自定义处理动作。

五 变量类型及变量绑定

Crank提供**变量绑定**功能，可将变量与控件的属性绑定，来改变文字，图片，位置，颜色等信息。

变量类型有：Boolean、Color、Number、Image
Text String、Font Name、Screen Name、
File 、Float Number 、Lua Function。

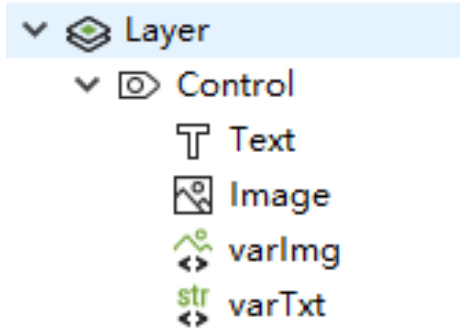


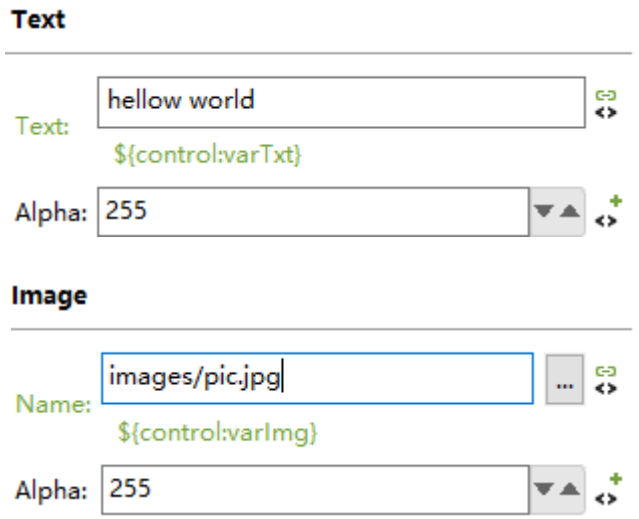


开发语言 Lua

Crank使用Lua语言进行开发，并有一套Lua API接口访问引擎。这个API是一个函数库，它允许通过操作数据、事件和用户接口组件与引擎进行交互。通过Lua API开发者可以:

1 从模型中获取和设置控件属性值，变量值等。
2 控制事件的发生和响应事件的动作，包括动画。
3 操作控件及其属性，比如Screen/Layer/Control等对象和显隐、透明度、位置信息等属性。





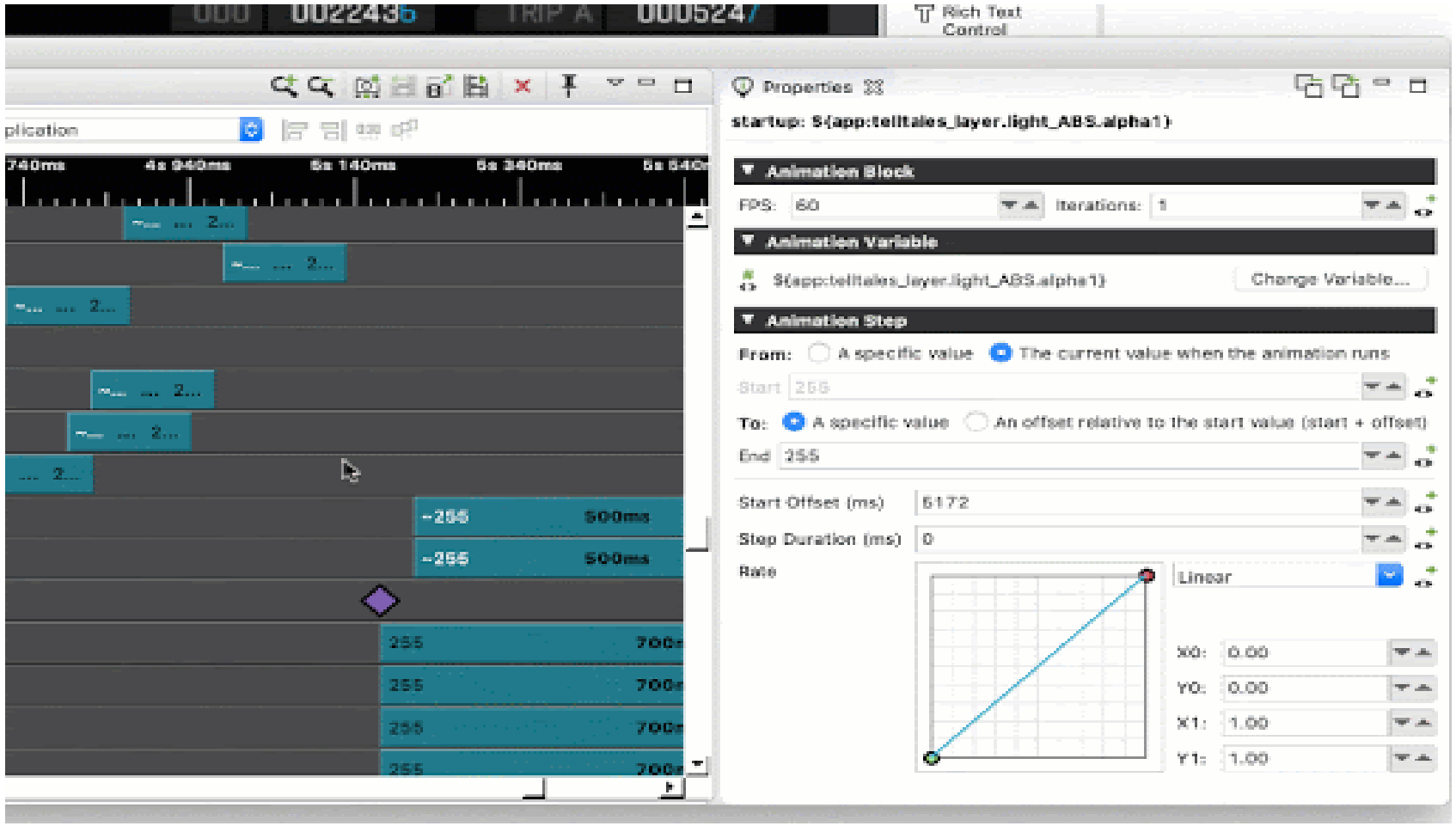

```lua
function changeVar()
  local data = {}
  data["Layer.control.varImg"] = "images/pic.jpg"
  data["Layer.control.varTxt"] = "hellow world"
  gre.set_data(data)
end
```

布局　　属性变量绑定　　代码控制

动画

Crank有两种方式进行动画控制

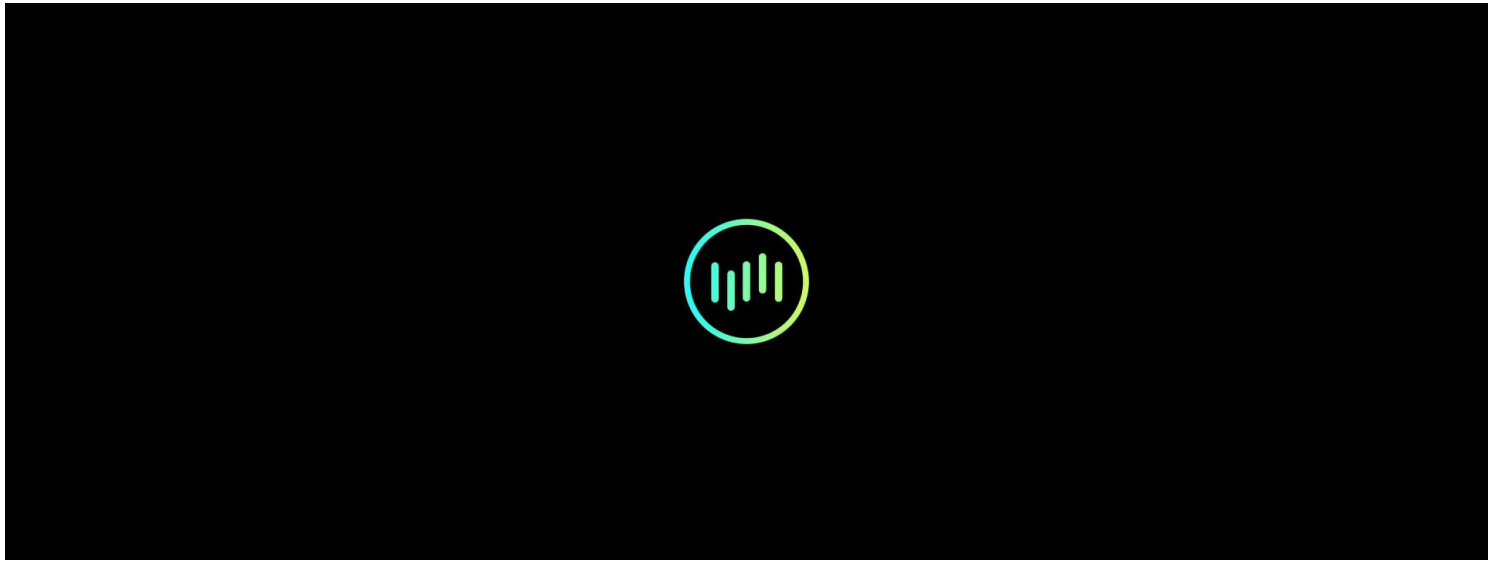
方式1：是利用Designer提供的timeline功能，通过鼠标操作即可组建动画并看到预览效果。



方式2：是通过lua代码自定义动画。

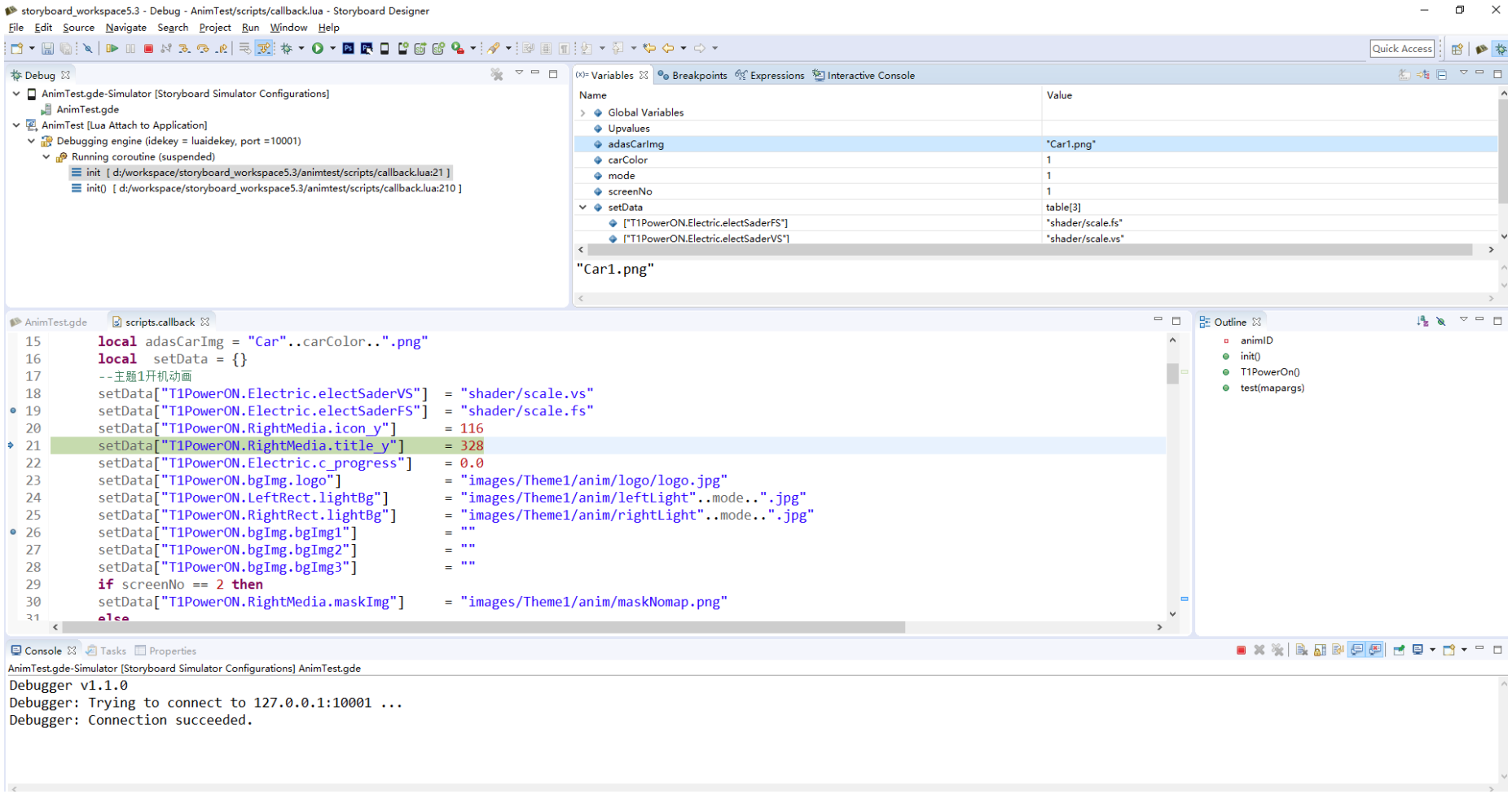
```lua
local animData = {}
animData[1] ={ rate = "easeout", interupt = 0, duration = 780, from = 920, to = 100,  offset = 320 + dur, key = "T1PowerON.LeftRect.grd_x"}
animData[2] ={ rate = "easeout", interupt = 0, duration = 780, from = 340, to = 141,  offset = 320 + dur, key = "T1PowerON.LeftRect.grd_y"}
animData[3] ={ rate = "easeout", interupt = 0, duration = 780, from = 80,  to = 800,  offset = 320 + dur, key = "T1PowerON.LeftRect.grd_width"}
animData[4] ={ rate = "easeout", interupt = 0, duration = 780, from = 45,  to = 500,  offset = 320 + dur, key = "T1PowerON.LeftRect.grd_height"}
animData[5] ={ rate = "easeout", interupt = 0, duration = 780, from = 920, to = 1000, offset = 320 + dur, key = "T1PowerON.RightRect.grd_x"}
animData[6] ={ rate = "easeout", interupt = 0, duration = 780, from = 340, to = 141,  offset = 320 + dur, key = "T1PowerON.RightRect.grd_y"}
animData[7] ={ rate = "easeout", interupt = 0, duration = 780, from = 80,  to = 800,  offset = 320 + dur, key = "T1PowerON.RightRect.grd_width"}
animData[8] ={ rate = "easeout", interupt = 0, duration = 780, from = 45,  to = 500,  offset = 320 + dur, key = "T1PowerON.RightRect.grd_height"}
animData[9] ={ rate = "easeout", interupt = 0, duration = 0,   from = 1,   to = 0,    offset = 320 + dur, key = "T1PowerON.LeftRect.grd_hidden"}
```

```lua
--创建动画id
local animID = gre.animation_create(60, 1, function() end)
--加入自定义动画
for i = 1, #animData, 1 do
  gre.animation_add_step(animID, animData[i])
end
--触发动画
gre.animation_trigger(animID)
```



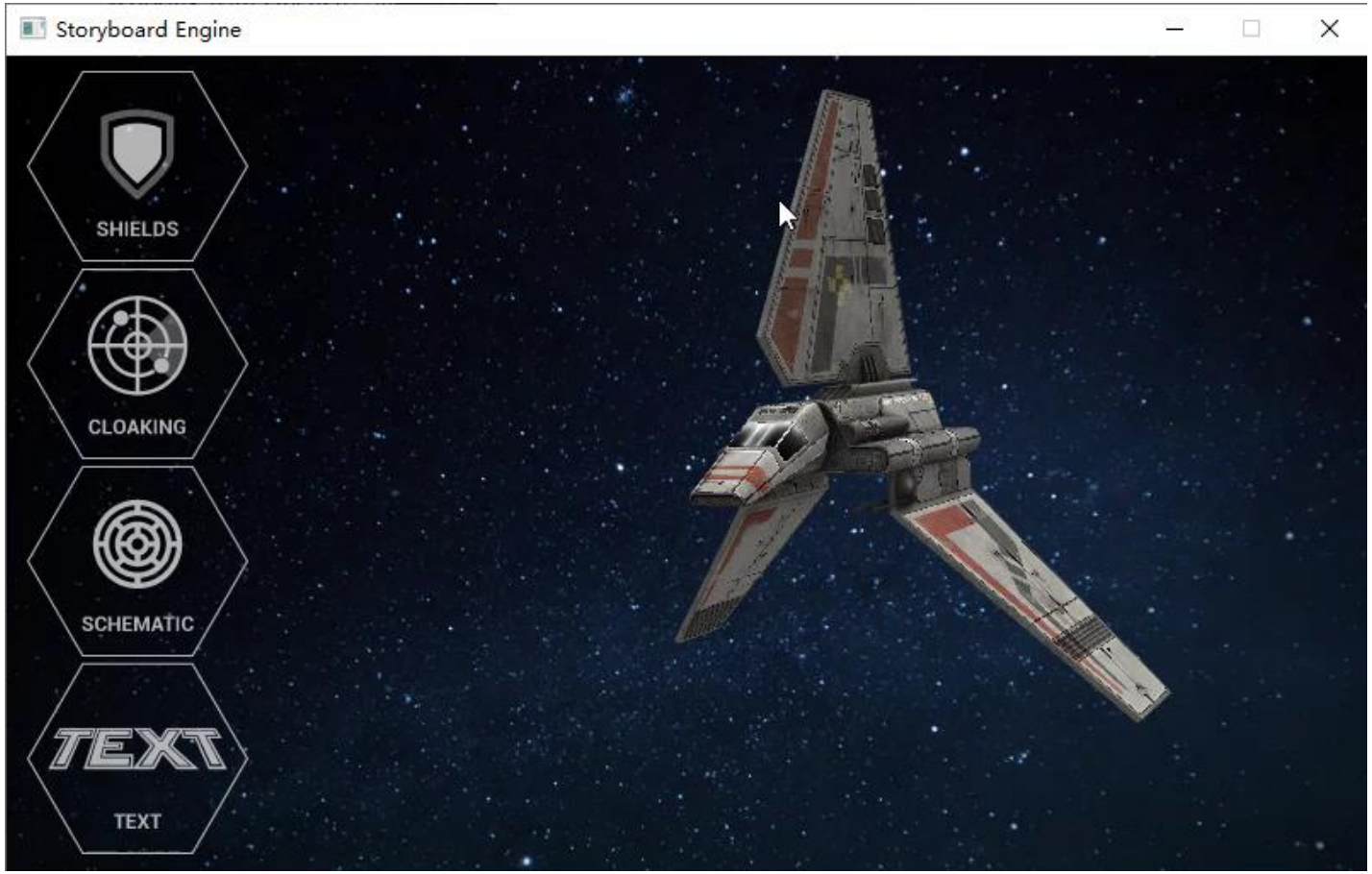
Lua Debug

为方便本地代码调试，StoryBoard支持Debug模式调试。



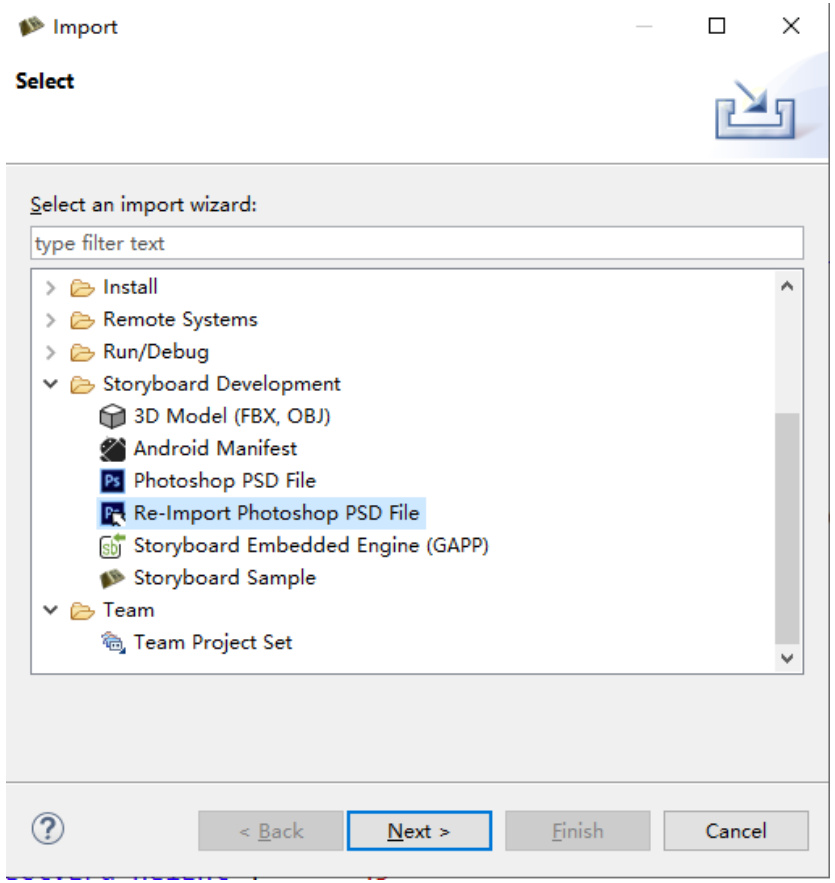
Designer

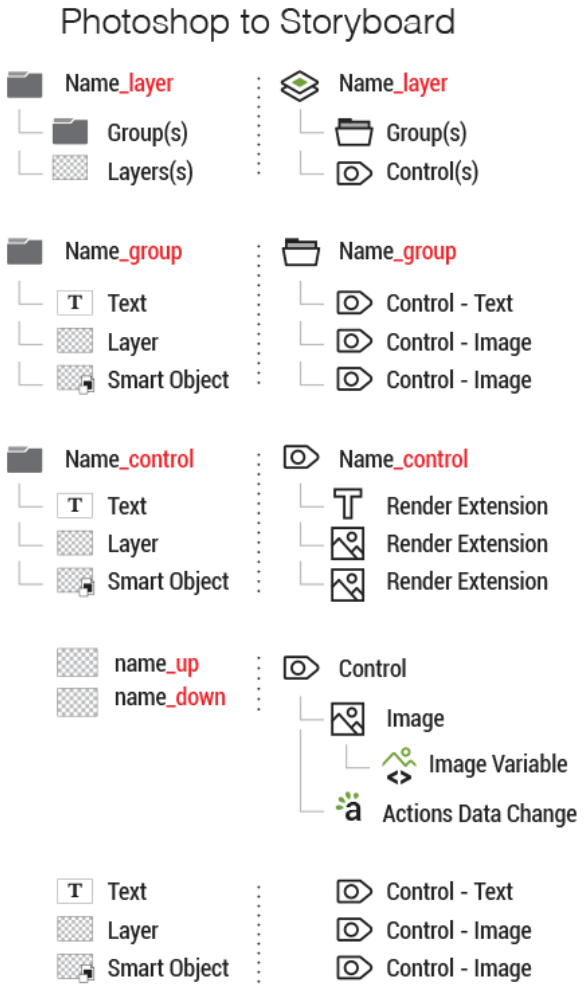
OpenGL和3D支持
Crank支持shader文件绑定和3D模型（obj，fbx）导入。用户可根据需求，进行3D相关开发。



导入PSD

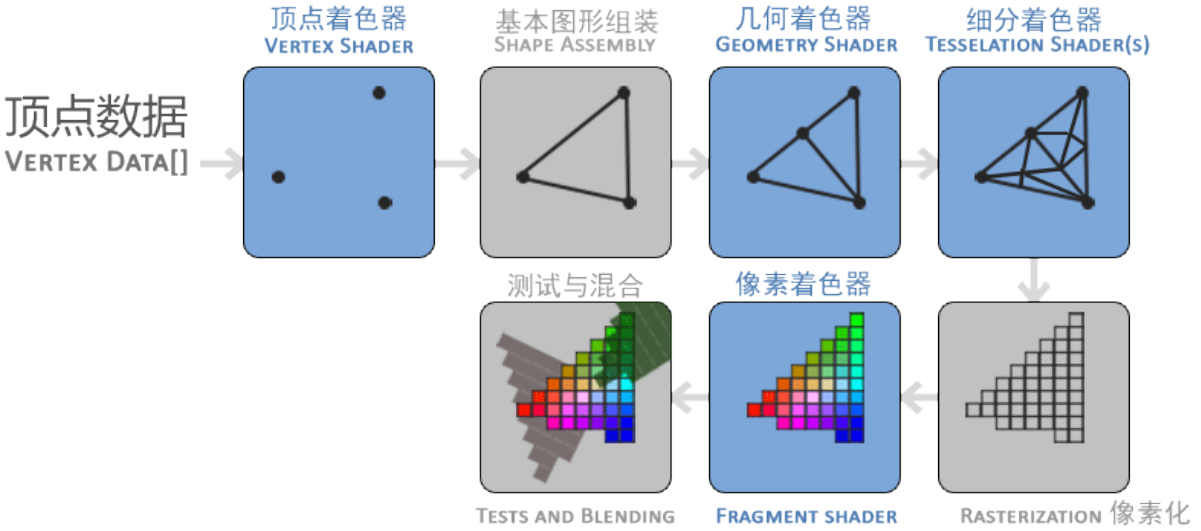
为便于开发人员快速布局UI界面结构，StoryBoard支持导入PSD文件，导入的UI结构与PSD中的层级关系、名称一致。





特效开发

- Shader是GPU流水线上一些可高度编程的阶段，而由着色器编译出来的最终代码是会在GPU上运行的。
- 依靠着色器可以控制流水线的渲染细节，例如用顶点着色器来进行顶点变换以及传递数据，用片元着色器来进行逐像素的渲染。



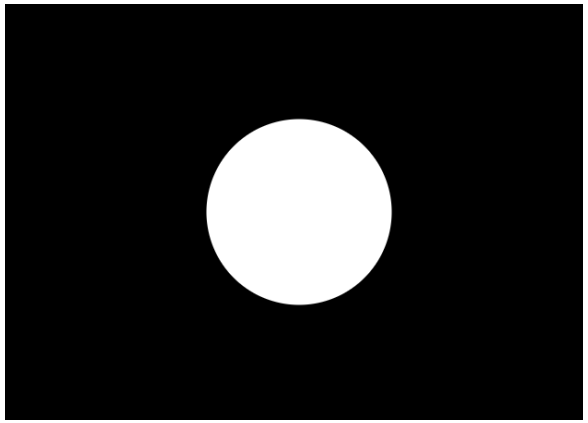
其中可编程着色器有顶点着色器、几何着色器、片段着色器(像素着色器)

图为Shader在渲染管线中的作用

- 简单图形的绘制

shader中利用图形函数，可以画出简单的图形。线可以根据线性方程或贝塞尔曲线进行绘制。圆可以用极坐标函数进行绘制。

- Shader入门：圆的实现



```
void main() {
    vec2 uv = fragCoord.xy;
    vec2 center = iResolution.xy * 0.5;
    gl_FragColor = vec4(1.0 - (length(uv - center) - 100.0));
}
```

特效开发

- 利用简单图形的绘制加上颜色渐变的效果，可以实现下面视频中的车道线。

特效开发

- sin，cos 函数的应用

shader中规律的变化都利用sin，cos函数进行。包括随机数也可以通过嵌套sin函数，取其小数部分，来用作伪随机数实现噪点等效果。

利用sin，点积等函数实现的伪随机数

```
float rand(vec2 co)
{
 return fract(sin(dot(co.xy, vec2(12.9898,78.233))) * 43758.5453);
}
```

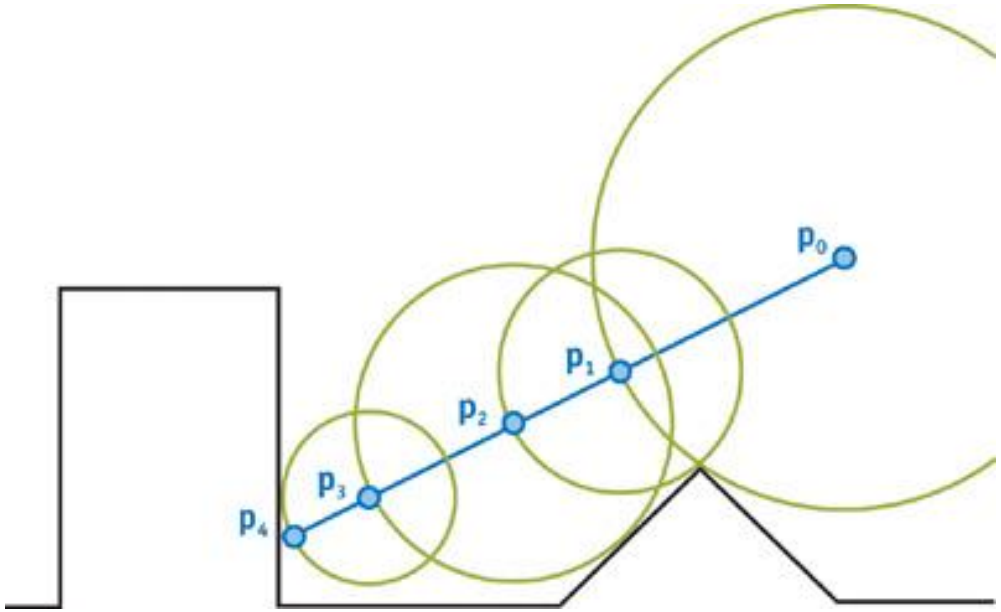
利用随机数实现的简单的粒子效果

Ray Marching算法

Ray marching如右图，算法大体思路是从P0点，按固定步长按射线方向递进，如果碰到物体则返回true，如果未碰触到物体则返回false。根据返回结果来判断P0点是否可以看到场景中的物体。

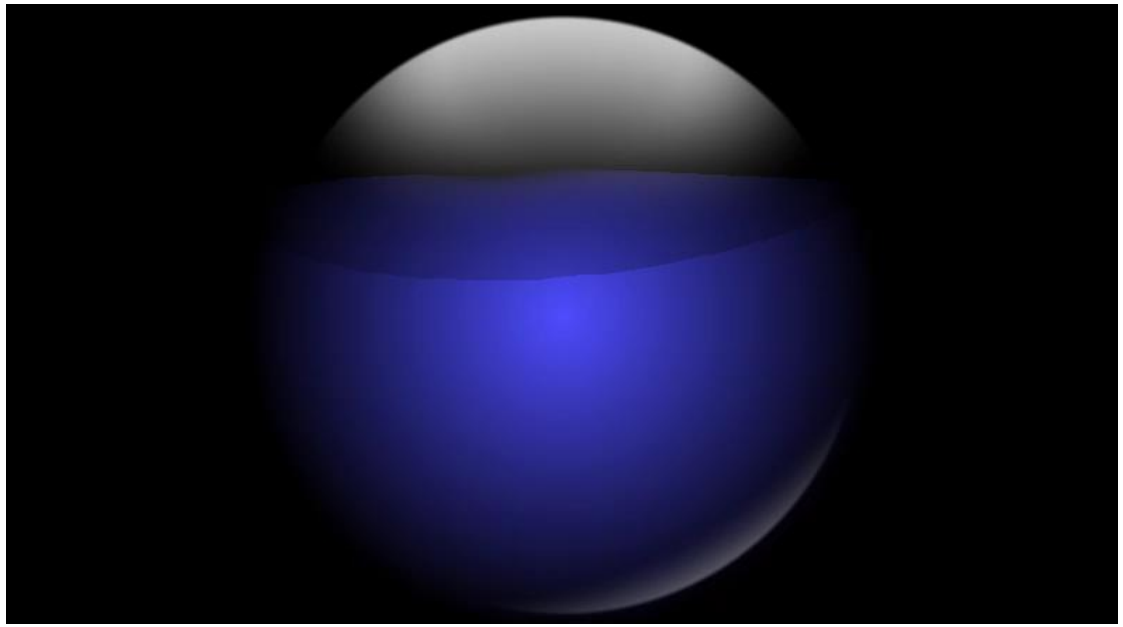
Distance field（距离场）

距离场绝大部分使用极坐标等方式，以数学公式的形式描述物体，包括三维物体。

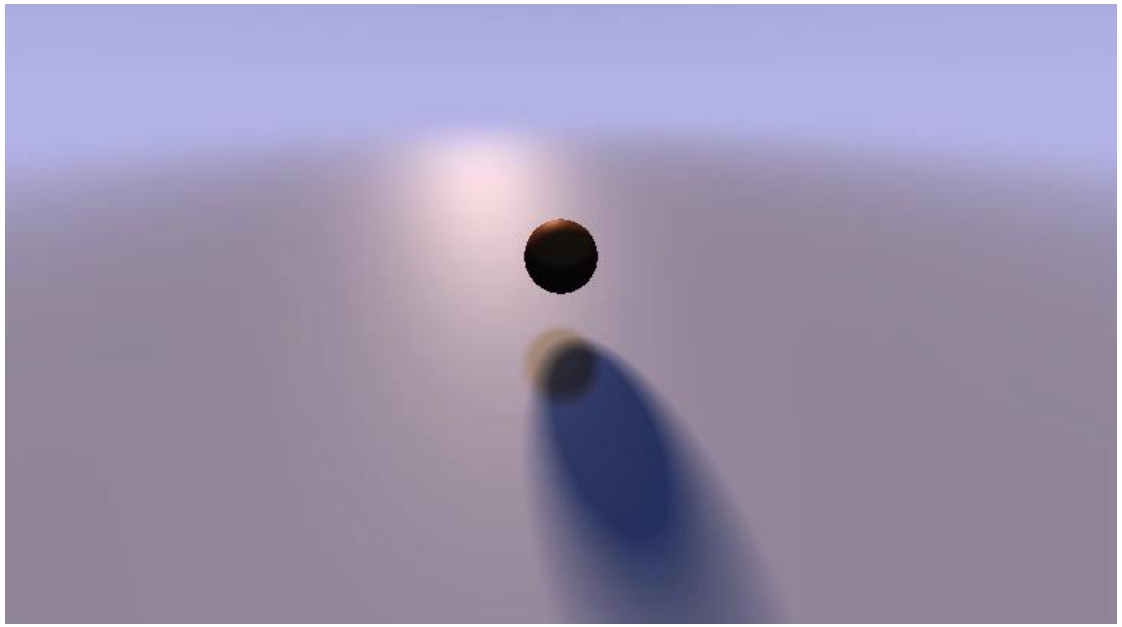


特效开发

用ray marching实现的3D水波纹效果。



用ray marching和distance field实现的3D小球和光照阴影效果。



特效开发

- 顶点着色器

利用顶点着色器进行模型的位置变换。
顶点着色器进行法线的计算并传入片段着色器。

- 边缘光的实现

计算法线和视角向量的夹角来实现边缘光。
法线和视角向量的夹角越大，则越接近边缘。

利用顶点着色器和边缘光实现的扫光效果

性能和效果的攻坚战

性能优化，是嵌入式平台不可避免的一个话题。一方面客户面的效果要求越来越高，越来越酷炫；另一方面本身硬件资源有限的情况下，又需要考虑成本；

如何在有限的资源下，最大化的实现酷炫的效果？

OpenGL渲染管线

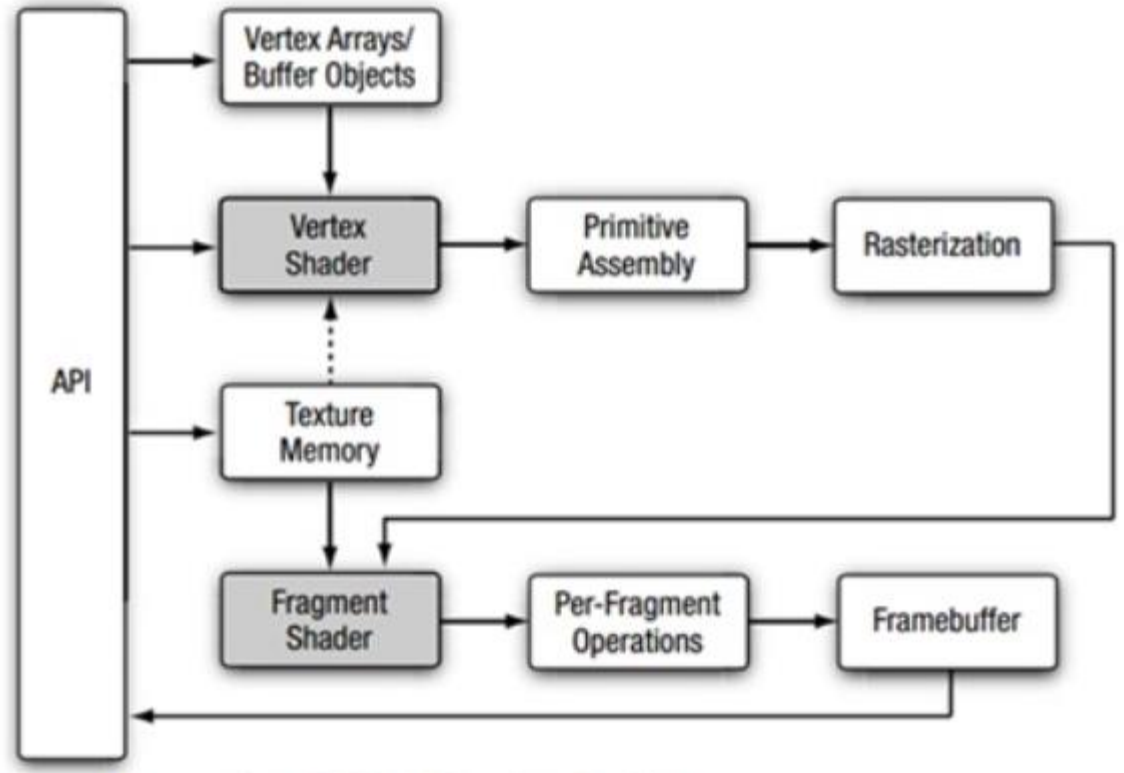


Figure 1-1 OpenGL ES 2.0 Graphics Pipeline

优化核心：降低GPU消耗

如何降低GPU资源的消耗，主要有以下几个方面：

1. 优化UI布局，避免不必要的叠层，合理布局减少资源浪费
2. 减少刷新负荷，比如局部刷新
3. 优化3D模型点面数
4. 简化或者优化光照效果
5. 优化Fragment Shader

优化Fragment Shader

1. GPU和CPU分工明确
2. Shader中减少判断分支
3. 简化Fragment Shader处理

优化Fragment Shader

1. GPU和CPU分工明确

   GPU适合做并行运算(矩阵变换等)，CPU适合做逻辑处理

   优化提醒：一些逻辑复杂的运算，特别是涉及到逻辑处理的可以在CPU处理完成，

   将结果传入Shader的Uniform变量，减轻GPU负荷

优化Fragment Shader

2. Shader中减少判断分支，例如使用内嵌函数step代替if分支

```
float a;
if (b > 0)
{
   a= 1.0;
}
else
{
   a= 2.0;
}
```

```
float a;
float tmp = step(b, 0);
a = 1.0 + tmp;
```

优化Fragment Shader

3. 能在Vertex Shader处理的就不要放到Fragment Shader处理

   例如光照处理，法向量(Normal Vector)等的计算可以放到Vertex Shader中进行。


感谢！
THANK YOU!