



Imagination

**Migrating to Vulkan™ with the
PowerVR Framework**

February 2017

www.imgtec.com

A new generation of APIs

Why?

- **Greatly reduced API overhead**
 - Driver simplification
 - Reduced CPU usage, leading to reduced power consumption
- **Ubiquity of multi-core processors**
 - Room for a modern API to fully utilise all available CPU cores
- **Legacy APIs are bloated**
 - 20+ years of evolution



What are the core goals of Vulkan?

Clean, efficient, modern

- **Built for multi-threading**

- Work can be effectively split over multiple threads

- **Stateless**

- Easy to write our own layers on-top of the API

- **Less driver ‘magic’ – more predictable / deterministic**

- Achieved through verbosity – the driver does not need to guess about state

- **Maps to modern hardware architecture**

- Especially mobile platforms with unified memory and tile based renderers

- **Cross-platform**

- **Simplified API design**

- Removal of legacy features / requirements

What are the costs?

- **OpenGL ES drivers are smart**
 - Automatic handling of multi-buffering/ghosting/stalling/synchronisation
- **Extremely verbose API**
 - Everything is explicit – no defaults
- **Requires a deep understanding of graphics**
 - GPUs perform optimally when pipelining and that requires smart use of buffers
 - Steep learning curve

How do we overcome these costs?

- **Vulkan is intended to be wrapped into a framework or middleware**
 - Should be smart like an OpenGL ES driver
 - Should facilitate multi-threading as the priority
 - Should be succinct
 - Should remove the burden of device setup from the user

PowerVR Framework

Overview

PowerVR

by Imagination

PVR Framework

Goals of the new Framework

- **Makes Vulkan fun**
 - No boiler plate code – which Vulkan adds a lot of
- **Focus on high level design**
 - Succinct expression of renders enabling focus on multi-threading and correctly pipelined render paths
- **Extremely permissive license**
 - MIT Licence

PVR Framework

Goals of the new Framework

- **Cross-platform, cross-API abstraction based on Vulkan**
 - Back-end supports Vulkan and OpenGL ES
- **Supports all PowerVR API extensions**
- **Supports all PowerVR asset files**
 - POD, PVR Textures, PFX
- **GLM used throughout for vector/matrix/quaternion math**

PVR Framework

Advanced C++ features

■ C++11

- Name-spaces, smart pointers, OOP, STL

■ Fully documented

- Both in code and stand-alone documentation
- Extra care taken with function names for clarity

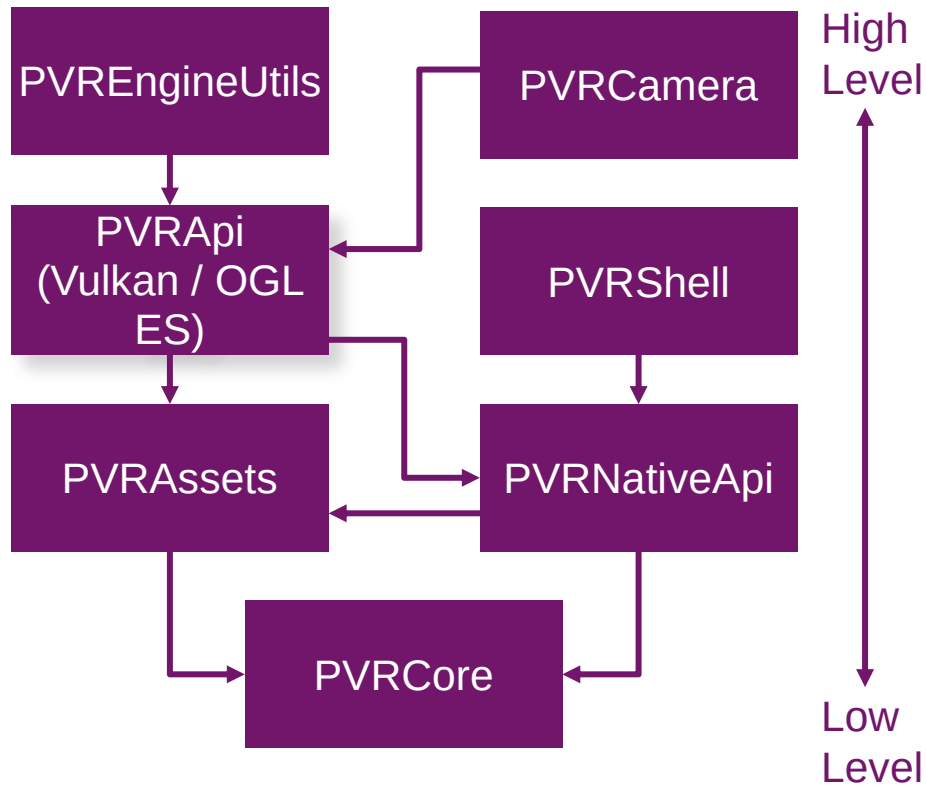
```
pvr::Result::Enum OGLESIntroducingPVRApi::initView() {  
    pvr::GraphicsContext gc = getGraphicsContext();  
    gc->
```

- createBuffer
- createBufferAndView
- createBufferView
- createCommandBufferOnDefaultPool**
- createCommandPool
- createComputePipeline
- createDescriptorPool
- createDescriptorSetLayout
- createDescriptorSetOnDefaultPool

```
public : pvr::api::CommandBuffer pvr::IGraphicsContext::createCommandBufferOnDefaultPool()  
\brief Create a new primary CommandBuffer on the default CommandPool  
\return A newly created CommandBuffer. Null if failed.  
File: IGraphicsContext.h
```

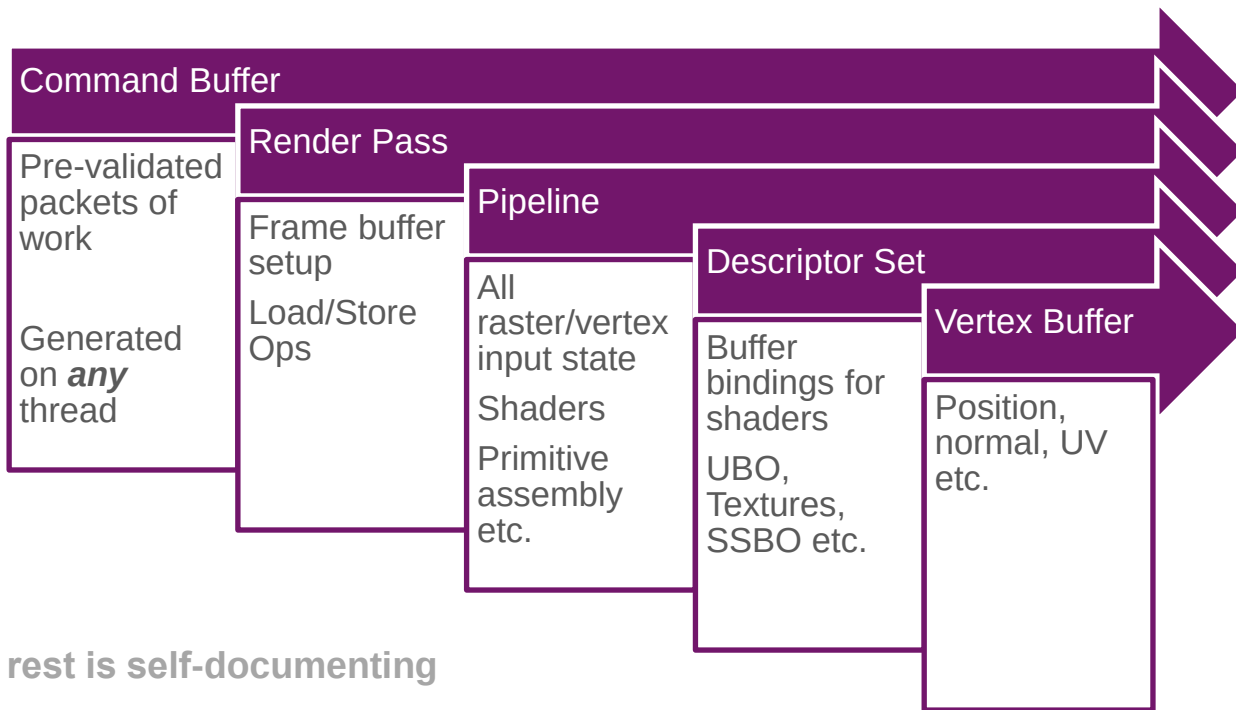
PVR Framework modules overview

Separate (compiled) libraries providing groups of functionality

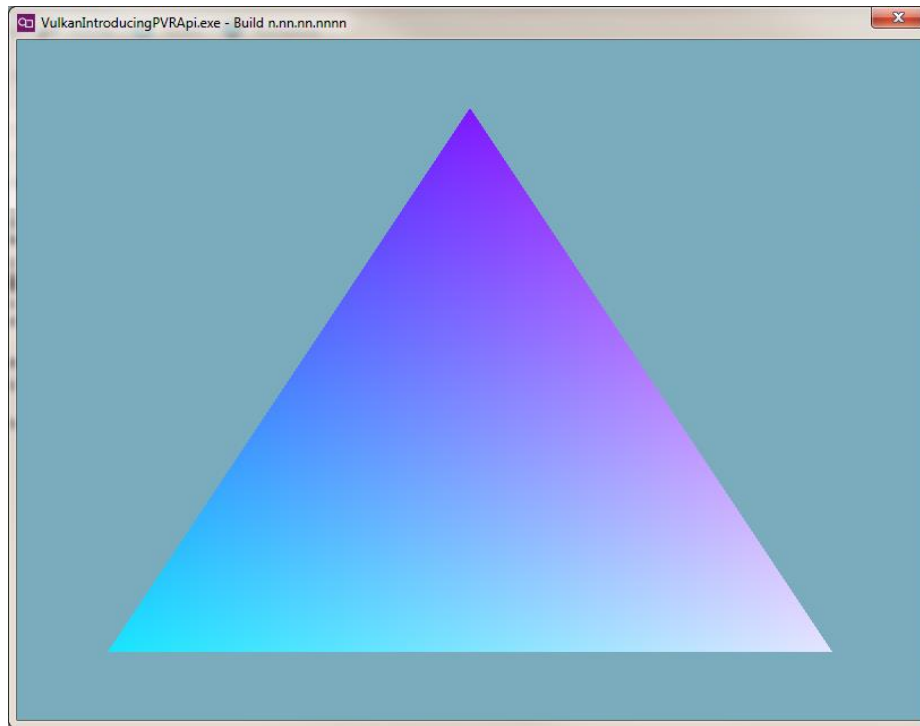


PVRApi

Follows the abstractions of modern graphics APIs



PVR Framework drawing a triangle



PVRShell getting started

Drawing a triangle

```
class OGLESIntroducingPVRApi : pvr::Shell
{
    pvr::Multi<pvr::api::CommandBuffer> onScreenCBAArray;
    pvr::Multi<pvr::api::Fbo> onScreenFBArray;
    pvr::api::GraphicsPipeline opaquePipeline;

    virtual pvr::Result initApplication() { return pvr::Result::Success; }
    virtual pvr::Result initView();
    virtual pvr::Result releaseView() { return pvr::Result::Success; }
    virtual pvr::Result quitApplication() { return pvr::Result::Success; }
    virtual pvr::Result renderFrame();
};
```

Drawing a triangle

PVR Framework exposed differences between OGLES and Vulkan – co-ordinates

- Screen y co-ordinate is inverted

```
const float OGLESIntroducingPVRApi::verts[] = { 0.0,  
#ifdef VULKAN_DEMO  
    -0.5  
#else  
    0.5  
#endif  
    ,0.0 ,  
    -0.5, 0.0, 0.0,  
    0.5, 0.0, 0.0 };
```

**asset loading classes handle this automatically*

Drawing a triangle

PVR Framework exposed differences between OGLES and Vulkan – shaders

- Vulkan uses SPIR-V shaders, OGLES uses GLSL shaders

```
pvr::api::Shader vert = getGraphicsContext()->
    createShader(*getAssetStream("vert.spirv"), pvr::types::ShaderType::VertexShader);

pvr::api::Shader frag = getGraphicsContext()->
    createShader(*getAssetStream("frag.spirv"), pvr::types::ShaderType::FragmentShader);
```

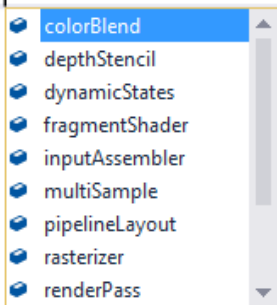

Drawing a triangle

Pipeline Setup

- Pipeline contains shaders and fixed function states

```
using namespace pvr::api;  
GraphicsPipelineCreateParam opaquePipeParameters = GraphicsPipelineCreateParam();  
  
opaquePipeParameters.fragmentShader.setShader(frag);  
opaquePipeParameters.vertexShader.setShader(vert);
```

opaquePipeParameters.



Drawing a triangle

Pipeline Creation

```
opaquePipeline = gc->createGraphicsPipeline(opaquePipeParameters);
```

Drawing a triangle

Command buffer setup

■ Command buffer creation and recording of draw calls

```
pvr::Result OGLESIntroducingPVRApi::initView()
{
    onScreenFBArray = getGraphicsContext()->createOnScreenFboSet();

    for (pvr::uint32 i = 0; i < getSwapChainLength(); ++i)
    {
        onScreenCBAArray.add(getGraphicsContext()->createCommandBufferOnDefaultPool());
        pvr::api::CommandBuffer &cb = onScreenCBAArray[i];

        cb->beginRecording();
        cb->beginRenderPass(onScreenFBArray[i], pvr::Rectanglei(0, 0, getWidth(), getHeight()), false,
            glm::vec4(123.0f / 255.0f, 172.0f / 255.0f, 189.0f / 255.0f, 1.0f));

        cb->bindPipeline(opaquePipeline);
        cb->bindVertexBuffer(vertexBuffer, 0, 0);
        cb->drawArrays(0, 3);

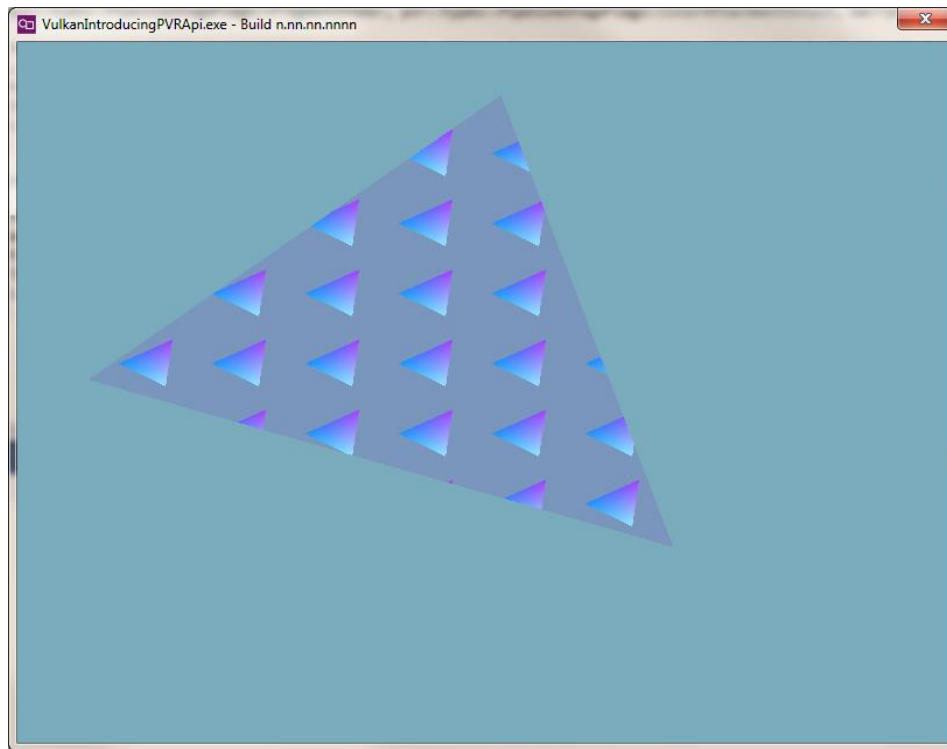
        cb->endRenderPass();
        cb->endRecording();
    }
    return pvr::Result::Success;
}
```

Drawing a triangle

Submit everything

```
pvr::Result OGLESIntroducingPVRApi::renderFrame()
{
    onScreenCBArray[getSwapChainIndex()]->submit();
    return pvr::Result::Success;
}
```

PVR Framework render to texture



Render to texture

Creating a texture

```
pvr::api::TextureStore offScreenTexture = getGraphicsContext()->createTexture(); //Create texture object

offScreenTexture->allocate2D(pvr::api::ImageStorageFormat(), //Initialise texture object
    getWidth(),
    getHeight(),
    pvr::types::ImageUsageFlags::ColorAttachment | pvr::types::ImageUsageFlags::Sampled);

pvr::api::TextureView offScreenTexView = getGraphicsContext()->createTextureView(offScreenTexture);
```

Render to texture

Creating a Framebuffer object (FBO)

■ Creating a render pass and FBO with colour attachment

```
pvr::api::RenderPassCreateParam offScreenPassParam; //Render pass configuration
pvr::api::FboCreateParam offScreenFBOParam; //FBO configuration

offScreenPassParam.setColorInfo(0, pvr::api::RenderPassColorInfo(offScreenTexture->getFormat(), //Format of texture i.e. RGB8
    pvr::types::LoadOp::Clear, //Load operation
    pvr::types::StoreOp::Store)) //Store operation
    .setSubPass(0, pvr::api::SubPass().setColorAttachment(0, 0));

pvr::api::RenderPass offScreenRenderPass = getGraphicsContext()->createRenderPass(offScreenPassParam); //Create a render pass

offScreenFBOParam.setColor(0, offScreenTexView) //Colour attachment
    .setNumLayers(1)
    .setDimension(getWidth(), getHeight()) //Dimension of FBO
    .setRenderPass(offScreenRenderPass); //Render pass that will be used when this FBO is bound

pvr::api::Fbo offScreenFBO = getGraphicsContext()->createFbo(offScreenFBOParam); //Create a FBO
```

Render to texture

Image Barriers

```
cb->beginRecording();

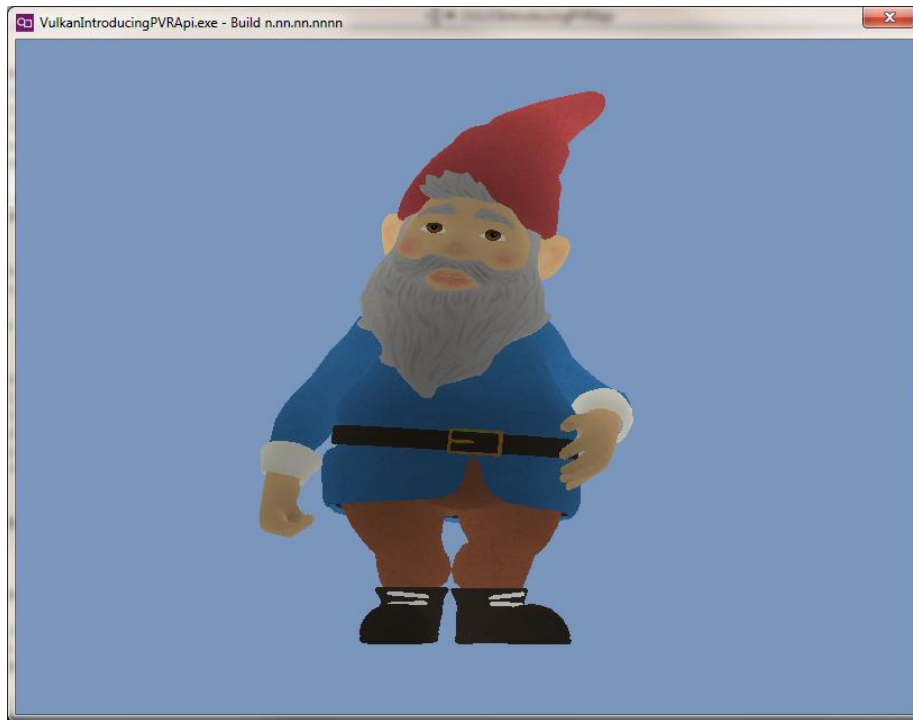
//Render to texture
cb->beginRenderPass(offScreenFBO, pvr::Rectanglei(0, 0, getWidth(), getHeight()), true,
    glm::vec4(123.f / 255.f, 150.f / 255.f, 189.f / 255.f, 1.f));

cb->bindPipeline(gp); //Graphics pipeline
cb->bindDescriptorSet(gp->getPipelineLayout(), 0, descSet); //Descriptor set
cb->bindVertexBuffer(buf, 0, 0);
cb->drawArrays(0, 3);
cb->endRenderPass();

//Add a memory barrier to force preceeding commands to finish before continuing
cb->pipelineBarrier(pvr::types::PipelineStageFlags::ColorAttachmentOutput,
    pvr::types::PipelineStageFlags::FragmentShader, sampleBarrier);

//.....
```


PVRAssets



PVRAssets

Asset loading

- **Classes to simplify loading of various asset & resource objects**
 - Models, Textures, Effects
- **Natively supports all PowerVR formats**
 - POD (Models / scenes, exported with PVRGeoPOD)
 - PVR (Textures, exported with PVRTexTool)
 - PFX (Shaders / effect files created with PVRShaman)
- **Easily extensible source code**

PVRAssets

Loading an object – importing data

```
//Load Gnome Asset

assets.init(*this); //Asset manager initialisation

assets.getTextureWithCaching(getGraphicsContext(),
    pvr::StringHash("gnome_texture.pvr"), &gnomeTex, nullptr); //Load texture

pvr::assets::ModelHandle gnomeModelHandle =
    pvr::assets::Model::createWithReader(pvr::assets::PODReader(getAssetStream("gnome.pod"))); //Load mesh

//Read vertex data from a mesh and create a single interleaved VBO & IBO
pvr::utils::createSingleBuffersFromMesh(getGraphicsContext(),
    gnomeModelHandle->getMesh(0), gnomeVBO, gnomeIBO);
```

PVRAssets

Loading an object – vertex attribute setup

```
pvr::utils::VertexBindings attributeBindings[] =  
{  
    { "POSITION", 0 },  
    { "NORMAL", 1 },  
    { "UV", 2 },  
};  
  
pvr::utils::createInputAssemblyFromMesh(gnomeModelHandle->getMesh(0),  
    attributeBindings, 3, opaquePipelineParam);
```

PVRAssets

Drawing

- Add draw commands to the command buffer

```
cb->bindPipeline(opaquePipeline);  
  
cb->bindVertexBuffer(gnomeVBO, 0, 0);  
  
cb->bindIndexBuffer(gnomeIBO, 0,  
    gnomeModelHandle->getMesh(0).getFaces().getDataType());  
  
cb->drawIndexed(0, gnomeModelHandle->getMesh(0).getNumIndices());
```

- Submit the command buffer to draw

PVREngineUtils

Render Manager

- **Automatic generation of render-able objects**
 - Data driven approach to pipeline setup & creation
- **Utilises existing PFX & POD file formats**
 - PFX used to define pipeline layout, POD stores model data
- **Fast prototyping**
 - Streamlined creation of graphics demo's

PVREngineUtils

PFX

```
<pipeline name="DefaultEffect">
  <depthstencil depthTest="true"/>
  <rasterization faceCulling="back"/>

  <attribute location="0" variable="inVertex" semantic="POSITION" dataType="vec3"/>
  <attribute location="1" variable="inNormal" semantic="NORMAL" dataType="vec3"/>
  <attribute location="2" variable="inTexCoord" semantic="UV0" dataType="vec2"/>

  <uniform variable="ModelMatrix" semantic="MODELMATRIX" dataType="mat4x4" scope="node"/>
  <uniform variable="MVPMatrix" semantic="MVPMATRIX" dataType="mat4x4" scope="node"/>
  <uniform variable="ModelWorldIT3x3" semantic="WORLDITMATRIX" dataType="mat3x3" scope="node"/>
  <uniform variable="LightPos" semantic="LIGHTPOSITION" dataType="vec3" scope="model"/>

  <texture set="0" binding="0" variable="sTexture" semantic="DIFFUSEMAP"
  minification="linear" magnification="linear" mipmap="linear" wrap_s="clamp" wrap_t="clamp"/>

  <shader name="DefaultVertexShader"/>
  <shader name="DefaultFragmentShader"/>
</pipeline>

<shader type="vertex" name="DefaultVertexShader">
  <file apiVersion="VULKAN" path="DefaultVertShader_vk.spv"/>
</shader>

<shader type="fragment" name="DefaultFragmentShader">
  <file apiVersion="VULKAN" path="DefaultFragShader_vk.spv"/>
</shader>
```

PVREngineUtils

PFX + POD = 3D demo

```
/* Rendering manager, puts effects together
with Models to render objects*/
pvr::utils::RenderManager mgr;

...
//Load PFX file
pvr::assets::pfx::PfxParser rd("effects.pfx", this);

//Add an effect
mgr.addEffect(*rd.getAssetHandle(), getGraphicsContext(),
             assetManager);
//Add POD model for rendering
mgr.addModelForAllPasses(scene);
//Create all API objects
mgr.buildRenderObjects();
scene->releaseVertexData(); //Cleanup
//Create semantic links between API objects
mgr.createAutomaticSemantics();

...
mgr.toPass(0, 0).recordRenderingCommands(cmdBuffers[swapidx], swapidx, false);

...
//Update the scene animation
mgr.toSubpassGroupModel(0, 0, 0, 0, 0).updateFrame(currentFrame);
//Update buffers, pipeline state etc.
mgr.updateAutomaticSemantics(getSwapChainIndex());
cmdBuffers[getSwapChainIndex()->submit();
```



PVR Framework

Summary

- **Use and learn new explicit APIs without their verbosity**
 - Vulkan-like interface
 - Maintains backward compatibility with OpenGL ES
 - Platform agnostic
 - Great asset loading code
 - Great starting point for your own engine, game or demo – easily extensible code
 - Great tool for learning / teaching
- **Fully permissive MIT licence – use as you like**

Visit us at GDC

Looking forward to meeting you in San Francisco !

At the Expo – Booth #532

01 – 03rd Mar 2017, Moscone Center, San Francisco

Visit the Imagination Technologies booth (#532) at the GDC expo for a personal look at the latest in mobile graphics and to get your questions answered by our experts.

In the Conference

Panel: The Future of VR and Mobile Graphics

14:40-15:40, 28th February: Room 2011 West Hall

This is a panel that will be examining the future of VR and mobile graphics – anything from the near future to the far future is allowed, wild speculation or solid beliefs. This panel will focus on the continuing rise of VR, and discuss where it's heading over the next few years – with particular attention to mobile. What are the end goals? What new technology is on the horizon? What kind of future is in store for mobile VR adopters? What will the content be like? We'll be asking these questions of our expert panellists to get their take, leading to no doubt some lively debate.



Imagination

Thank You

Resources

- **PowerVR support portal**

- <https://pvrsupport.imgtec.com>

- **PowerVR forum**

- <https://community.imgtec.com/forums/cat/powervr-insider-graphics/>

- **PowerVR SDK**

- <https://www.imgtec.com/blog/powervr-tools-sdk-2016-r2-package-now-live/>

- **Vulkan on PowerVR devices**

- <https://www.imgtec.com/blog/tiling-positive-or-how-vulkan-maps-to-powervr-gpus/>

- **Vulkan high efficiency for mobile**

- <https://www.imgtec.com/blog/vulkan-high-efficiency-on-mobile/>

- **Vulkan scaling on multiple threads**

- <https://www.imgtec.com/blog/vulkan-scaling-to-multiple-threads/>