Introducing Starling

1페이지

001 찜해주세요~ nassol : 0318 완료

First Flight

What Is Starling?
스탈링이 뭔가요**?**

Starling is an ActionScript 3 2D framework
developed on top of the Stage3D APIs
(available on desktop in Flash Player 11 and Adobe AIR 3).

스탈링은 액션스크립트 3 2D 프레임워크이다.
스탈링은 Stage3D API를 기반으로 해서 만들어졌다.
(데스크탑에서는 플래시 11과 어도비 에어3에서 제공된다.(??))

Starling is mainly designed
for game development,

but could be used
for many other use cases.

스탈링은 주로 게임 개발에 사용되게 설계되었으나, 다른 목적으로도 사용할 수 있다.

Starling makes it possible
to write fast GPU-accelerated applications
without having to touch the lowlevel Stage3D APIs.

스탈링을 사용하면 저수준의 Stage3D API를 직접 다루지 않고서도 그래픽 가속화된 빠른 어플리케이션을 만들 수 있다.

002 찜해주세요~ nassol 0318 완료

Why Starling?
스탈링이 왜 좋은가**?**

Most Flash developers
want to be able to leverage GPU acceleration (through Stage3D)
without the need
to write such higher-level frameworks (요부분 어떤 의미인지..)
and dig into the low-level Stage3D APIs.

플래쉬 개발자들은 대부분 저수준의 Stage3D API를 직접 파고들지 않고서 Stage3D를 사용한 GPU 가속화를 사용하고 싶어 한다.

Starling is completely designed
after the Flash Player APIs

and abstracts
the complexity of Stage3D (Molehill)

and allows
easy and intuitive programming for everyone.
스탈링은 플래시 플레이어 API에 맞게 설계되었다.
그리고 Stage3D의 복잡성을 추상화한다.
또한 스탈링을 사용하면 누구나 쉽고 직관적으로 프로그래밍을 할 수 있다.

Starling is for ActionScript 3 developers,
especially those involved in 2D game development,
so you will need to have a basic understanding of ActionScript 3.

스탈링은 액션 스크립트 3 개발자를 위한 것이다.
스탈링은 특히 2D 게임 개발을 하는 개발자를 위한 것이다.
따라서 스탈링을 사용하려면 액션스크립트 3에 대해서는 기본적으로 알고 있어야 한다.

003 찜해주세요~

By its design
(lightweight, flexible, and simple),

Starling can also be used
for other use cases like UI programming.

That said,
everything is designed to be
as intuitive as possible,

so any Java or .Net developer will get the hang of it
quickly as well.

004 찜해주세요~

Philosophy

Intuitive

Starling is easy to learn.

Flash/Flex developers will feel at home immediately,
since
it follows most of the ActionScript dogmas
and abstracts the complexity of the low-level Stage3D APIs.

Instead of **coding against concepts** like
vertices buffer,
perspective matrices,
shader programs
and assembly bytecode,
you will use familiar concepts
like
a DOM display list,
an event model,
and familiar APIs
like MovieClip, Sprite, TextField, and so on.

2페이지

005 찜해주세요~

Lightweight
스탈링은 가볍다.

Starling is a lightweight bird in many ways.

The amount of classes is limited (around 80k of code).

There are no external dependencies
beside Flash Player 11 or AIR 3

(mobile support will come in a future release).

This keeps
your applications small
and your workflow simple.

006 찜해주세요~

Free
스탈링은 무료다.

Starling is free and alive(의미??).

Licensed under the Simplified BSD license,
you can use it freely
even in commercial applications.

We are working on it every day
and we count on an active community
to improve it even more.

007 찜해주세요~

How

Behind the scenes,
Starling uses the Stage3D APIs,
which are low-level GPU APIs
running on top of
OpenGL and DirectX **on desktop**
and OpenGL ES2 o**n mobile devices.**

As a developer,
you have to know that
Starling is the ActionScript 3 port of Sparrow
(http://www.sparrow-framework.org),

the equivalent library for iOS
relying on OpenGL ES2 APIs (Figure 1).

Figure 1. Starling layer on top of Stage3D (Molehill)

008 찜해주세요~

Starling re-creates many APIs
that Flash developers are already familiar with.

Figure 2 illustrates the APIs
exposed by Starling
when it comes to graphical elements.

3페이지

009 찜해주세요~

Many people think that
the Stage3D APIs are strictly limited to 3D content
(and, to be fair, the name is a bit confusing),
but you can in fact also create 2D content with them.

Figure 3 illustrates the idea.

How can we draw something like a MovieClip
with the drawTriangles API?

Actually, it is very simple.

GPU are extremely efficient at drawing triangles,
so the drawTriangles API will draw two triangles,

and we will then
sample a texture
and apply it
to the triangles using UV mapping.

We will then end up with our textured quad,
which represents our sprite.

4페이지

010 찜해주세요~

By updating the texture every frame on our triangles,
we would end up with a MovieClip.

Pretty cool, huh?

Now, the good news is
that we will not even have to go through those details
when using Starling.

We will just provide our frames, supply them to a Starling MovieClip, and voila (Figure 4)!

011 찜해주세요~

To give you an idea of
how Starling reduces the complexity,
let’s see what code we would have to write
to display a simple textured quad(사각형)
using the low-level Stage3D APIs:

```
// create the vertices
var vertices:Vector.<Number> = Vector.<Number>([
−0.5,−0.5,0, 0, 0, // x, y, z, u, v
−0.5, 0.5, 0, 0, 1,
0.5, 0.5, 0, 1, 1,
0.5, −0.5, 0, 1, 0]);
```

012 찜해주세요~

```
// create the buffer to upload the vertices
var vertexbuffer:VertexBuffer3D = context3D.createVertexBuffer(4, 5);

// upload the vertices
vertexbuffer.uploadFromVector(vertices, 0, 4);

// create the buffer to upload the indices
var indexbuffer:IndexBuffer3D = context3D.createIndexBuffer(6);

// upload the indices
indexbuffer.uploadFromVector (Vector.<uint>([0, 1, 2, 2, 3, 0]), 0, 6);

// create the bitmap texture
var bitmap:Bitmap = new TextureBitmap();
```

5페이지

013 찜해주세요~

```
// create the texture bitmap to upload the bitmap

var texture:Texture = context3D.createTexture(bitmap.bitmapData.width,
bitmap.bitmapData.height, Context3DTextureFormat.BGRA, false);

// upload the bitmap

texture.uploadFromBitmapData(bitmap.bitmapData);
// create the mini assembler
var vertexShaderAssembler : AGALMiniAssembler = new AGALMiniAssembler();

// assemble the vertex shader
```

014 찜해주세요~

vertexShaderAssembler.assemble(Context3DProgramType.VERTEX,
"m44 op, va0, vc0\n" + // pos to clipspace
"mov v0, va1" // copy uv
);

// assemble the fragment shader

fragmentShaderAssembler.assemble(Context3DProgramType.FRAGMENT,
"tex ft1, v0, fs0 <2d,linear, nomip>;\n" +
"mov oc, ft1"
);

// create the shader program

var program:Program3D = context3D.createProgram();

// upload the vertex and fragment shaders

program.upload(vertexShaderAssembler.agalcode, fragmentShaderAssembler.agalcode);

015 찜해주세요~

// clear the buffer

context3D.clear (1, 1, 1, 1);

// set the vertex buffer

context3D.setVertexBufferAt(0, vertexbuffer, 0, Context3DVertexBufferFormat.FLOAT_3);
context3D.setVertexBufferAt(1, vertexbuffer, 3, Context3DVertexBufferFormat.FLOAT_2);

// set the texture

context3D.setTextureAt(0, texture);

// set the shaders program

context3D.setProgram(program);

// create a 3D matrix

```
var m:Matrix3D = new Matrix3D();
```

016 찜해주세요~

```
// apply rotation to the matrix
        to rotate vertices along the Z axis

m.appendRotation(getTimer()/50, Vector3D.Z_AXIS);

// set the program constants (matrix here)

context3D.setProgramConstantsFromMatrix(Co
ntext3DProgramType.VERTEX, 0, m, true);

// draw the triangles

context3D.drawTriangles( indexBuffer);
```

6페이지

017 찜해주세요~

```
// present the pixels to the screen

context3D.present();
```

And we would end up with the result shown in Figure 5.

Figure 5. A simple textured quad

Pretty complex code for this, right?

That is the cost of having access to low-level APIs.

You get to control a lot of things, but at the cost of low-levelness.

018 찜해주세요~

With Starling, you will write the following code:

// create a Texture object
        out of an embedded bitmap

var texture:Texture = Texture.fromBitmap (new embeddedBitmap());

// create an Image object our of the Texture

var image:Image = new Image(texture);

// set the properties

image.pivotX = 50;
image.pivotY = 50;
image.x = 300;
image.y = 150;
image.rotation = Math.PI/4;

// display it

019 찜해주세요~

addChild(image);

As an ActionScript 3 developer
        already accustomed to the Flash APIs,
                you will feel pretty much at home
                        with these APIs exposed,
                                while all the complexity of the Stage3D APIs
                                        is done behind the scenes.

If you try to use the redraw regions feature,
        you will see that
                Starling, as expected,
                        renders everything on Stage3D,

not the classic display list.

020 찜해주세요**~**

Figure 6 illustrates the behavior.

We have a quad rotating on each frame;
the redraw regions only show the FPS counter
sitting in the display list (running on the CPU).