张少勇1

Thank you for coming everyone! My name's Abel and I'm a VFX artist from NetEase Games in Hangzhou, China.

Three years ago, I worked at Sledgehammer games. Right after we shipped Call of duty: Advance Warfare, we had a week or two for VFX research . I tried to figure out the flow map shader using particle subUV textures that first developed by Gurrilla Games for Killzone2. But I failed because the complexity of the shder logic.

Soon after that I moved to China and joined NetEase Games. I have kept digging up flow map technique. in fact, flow map is a such essential technique for VFX artist: Animators animate arms and legs, We VFX artist animate pixels, and flow map is our tool.

One of the world's top 10 highest earning game companies with revenue of $4.03 billion in 2016.

More than 8000 employees with offices in Hangzhou, Beijing, Shanghai, Guangzhou, San Francisco, Seoul and Tokyo.

Released 220 of our own games as well as 47 licensed games.

In the top 3 grossing companies on the App Store (iOS) May, 2016

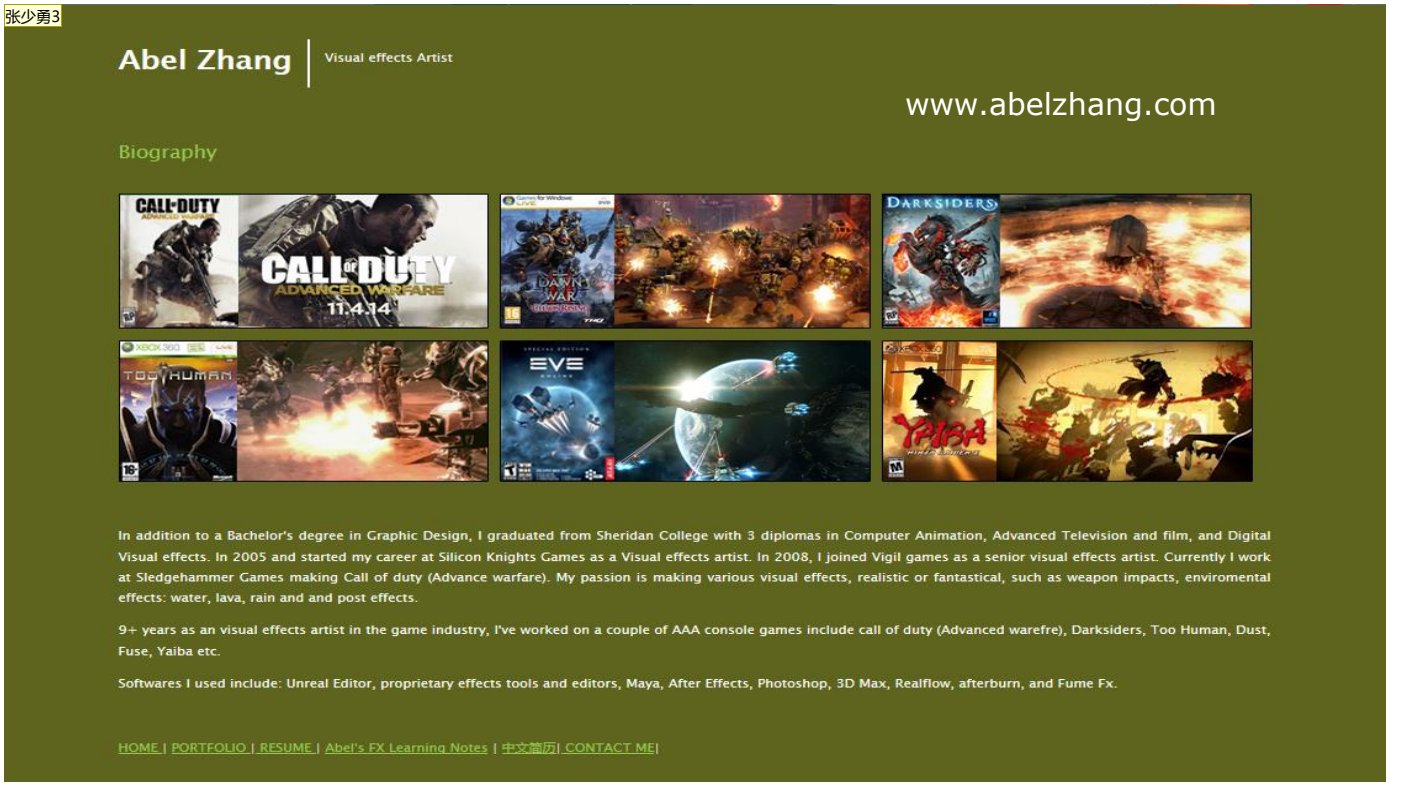
You may not know NetEase that well at the moment, but as we can see from this overview: it's a large and influential company in China. And we're hoping to increase our presence in the west in time.

Before joining in NetEase, all the projects I worked on were AAA games, such as *Call of Duty: Advanced Warfare*, *Darksiders 1*, *Too Human* and and several others. On switching to mobile, naturally I start applying techniques from AAA development to mobile. But I have to tailor them to these devices.

In China, most of our game players use lower end cell phones. So to me, good VFX is one that runs on all types of mobile devices, and good techniques are simple and effective ones.

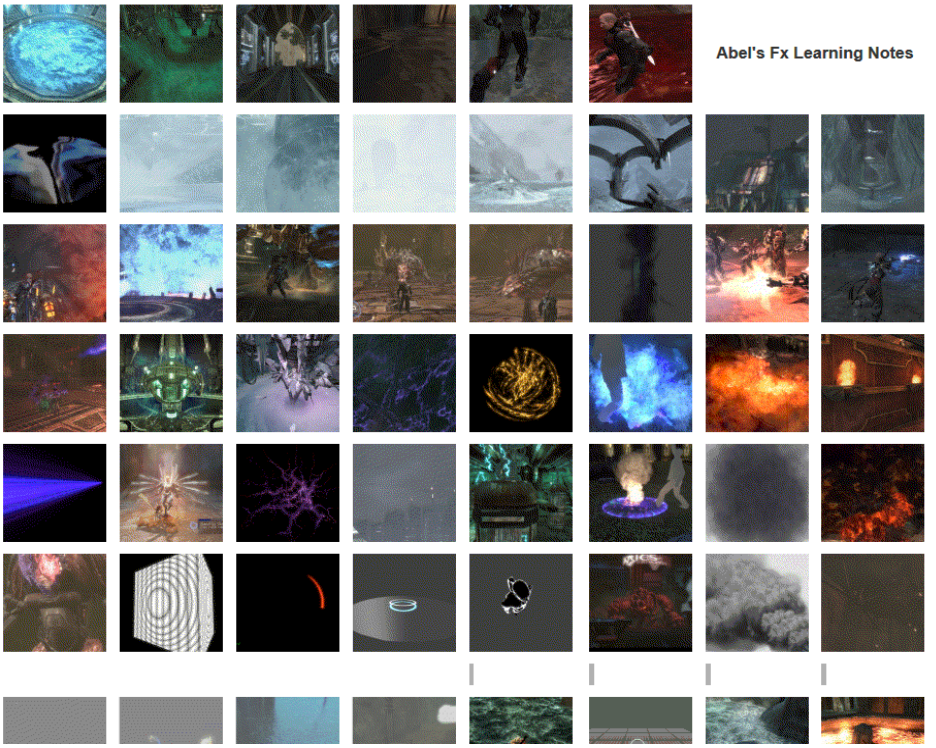
So today I want to talk about one example: a mobile effective, easy-to-use Flow-map shader – and how I understand it and how I've applied them to our current mobile project: *Galactic Frontline*.

张少勇3

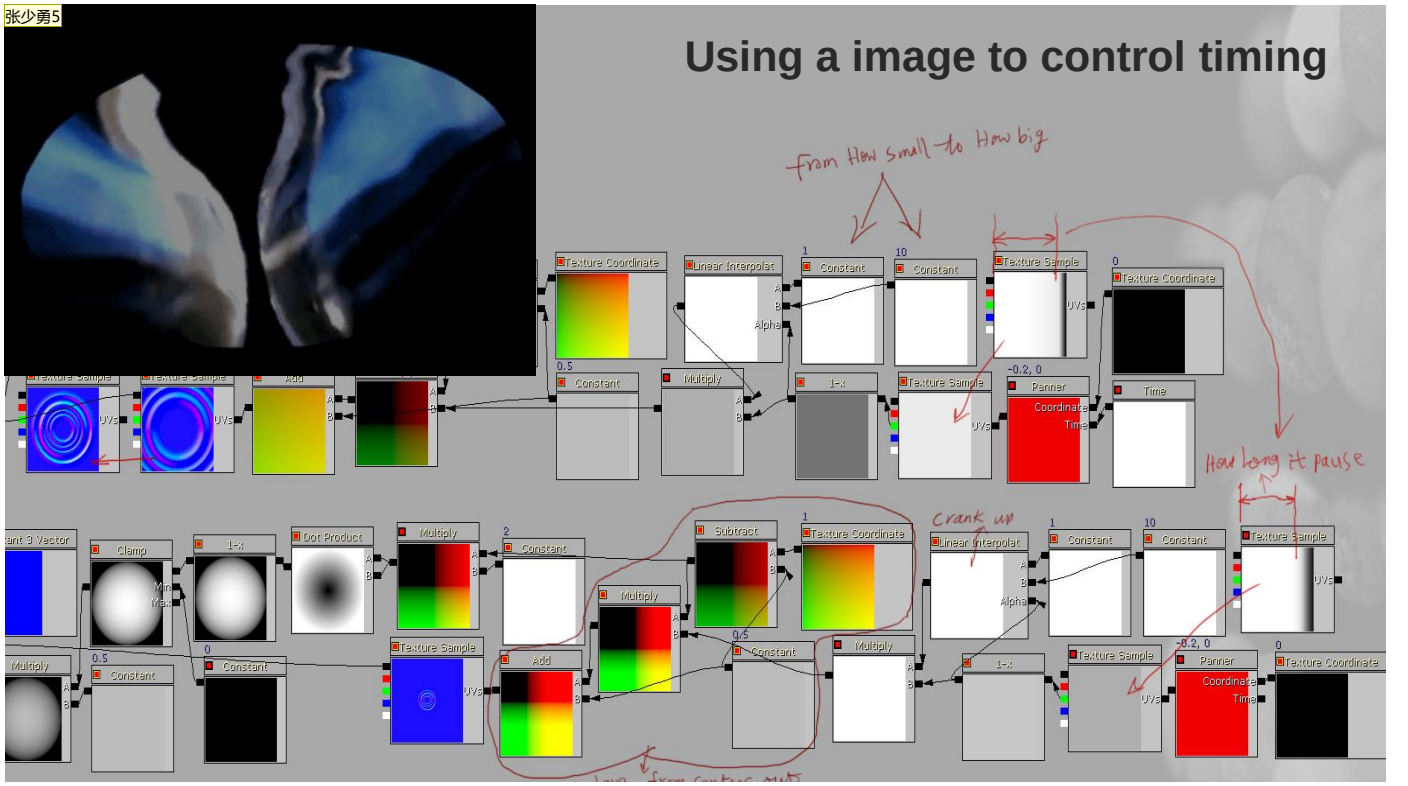


Before anything, I need to confess that I have zero coding knowledge and math was never my favourite subject. But I've always been interested in art and VFX techniques. I learned VFX by taking Picasso's advice. He said "good artists borrow, great artists steal". I experimented and emulated the techniques of other technical artists, VFX artists and programmers. I toyed and tweaked the interesting techniques to find my way. All the techniques that I've collected over the years, my treasures, so to speak are on my website: www.abelzhang.com. Just click *Abel's FX Learning Notes* and you will see this page.

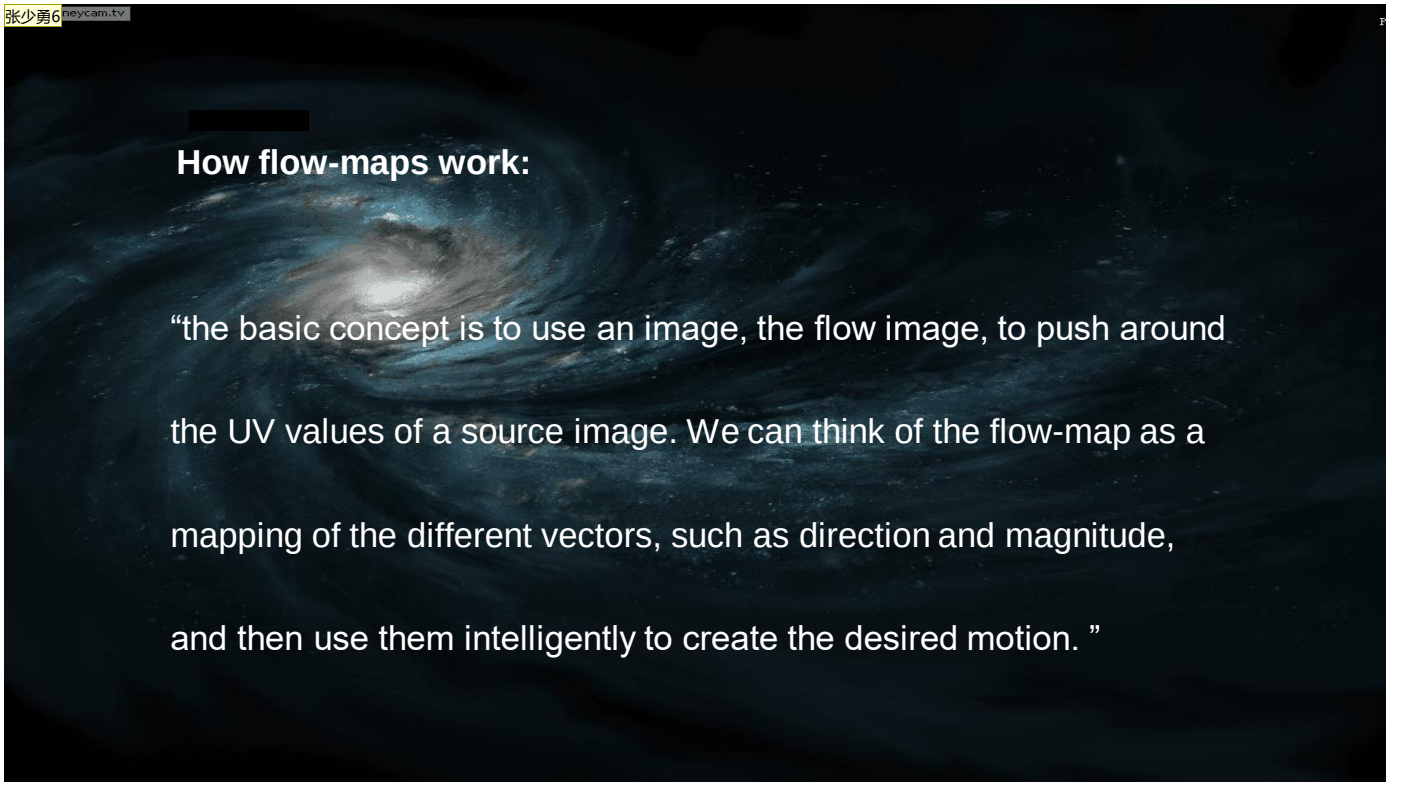
张少勇4



The techniques shown here have been collected over the course of around ten years. I used them during my time at Silicon Knights, Vigil Games, Sledgehammer, and now, NetEase Games. Most of them are shader tricks for VFX. Shaders are in fact the node base programming; which is not an artist good at, so I have to take notes to remember, whether I understand it or not. I'd say taking notes is the big part that got me survived 13 years career as an visual effects artist.



For example, this sequential ripple action caused by a water drop. The rebounce timing is controlled by a gradient image. I noted this down 10 years ago and it was the first time I discovered that animation could be controlled by a gradient image. If you asked me to recreate this fx, I am sure it will take much more time without checking my notes.



As I mentioned, today I want to talk about one of my favourite of these techniques,

This swirling galaxy is a menu background from *Galactic Frontline*.

Creating this kind of animation easily, quickly and relatively cheaply is what I’m going to delve into today.

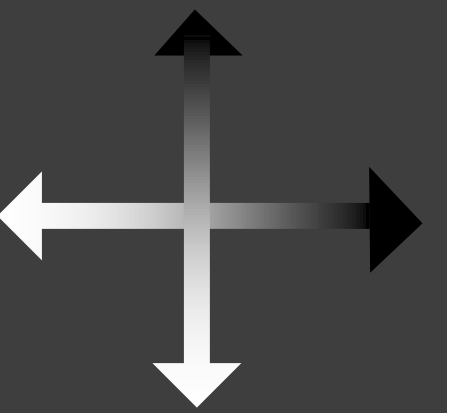
张少勇7

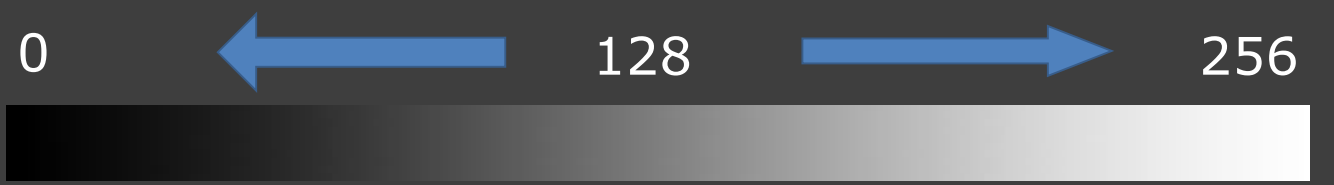
How flow-map change the UV of the source image:

128 Middle Gray creates no motion.

Black creates motion in one direction.

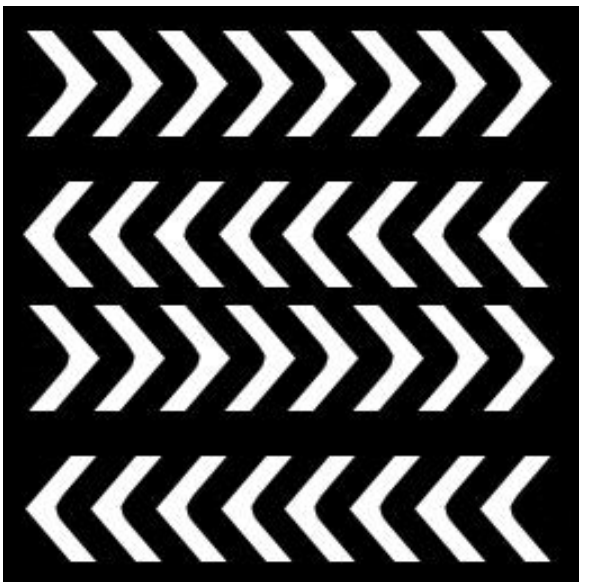
White creates motion in the opposite direction.





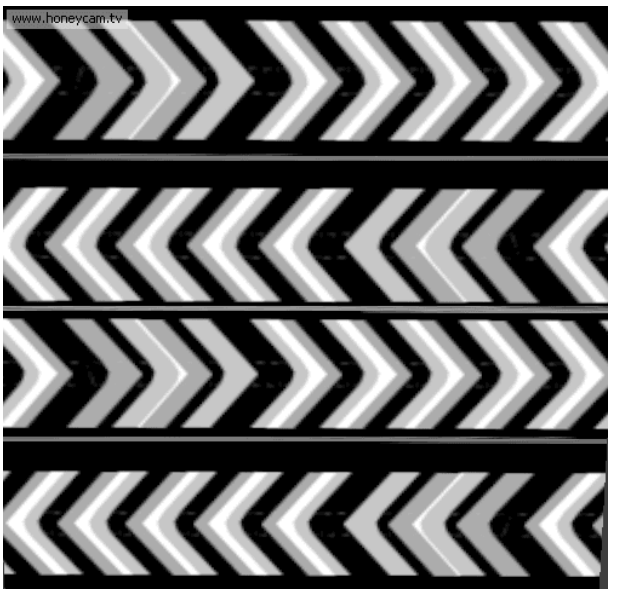
Think of the gray scale of the flow-map as an array of numbers from 0 to 256. We can use these numbers to manipulate the position of each pixel of a source image:

The Source Image



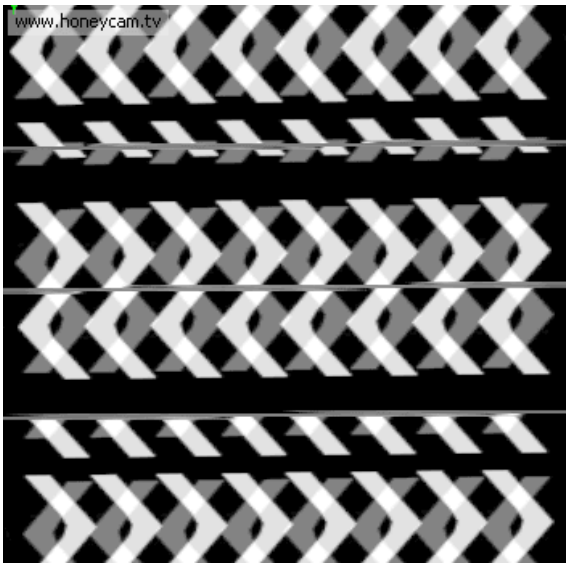
Say we have this image, the source image. It’s not moving.

Animated Left and Right



How are we going to make it move, both from the left and from the right?

Animated Up and Down

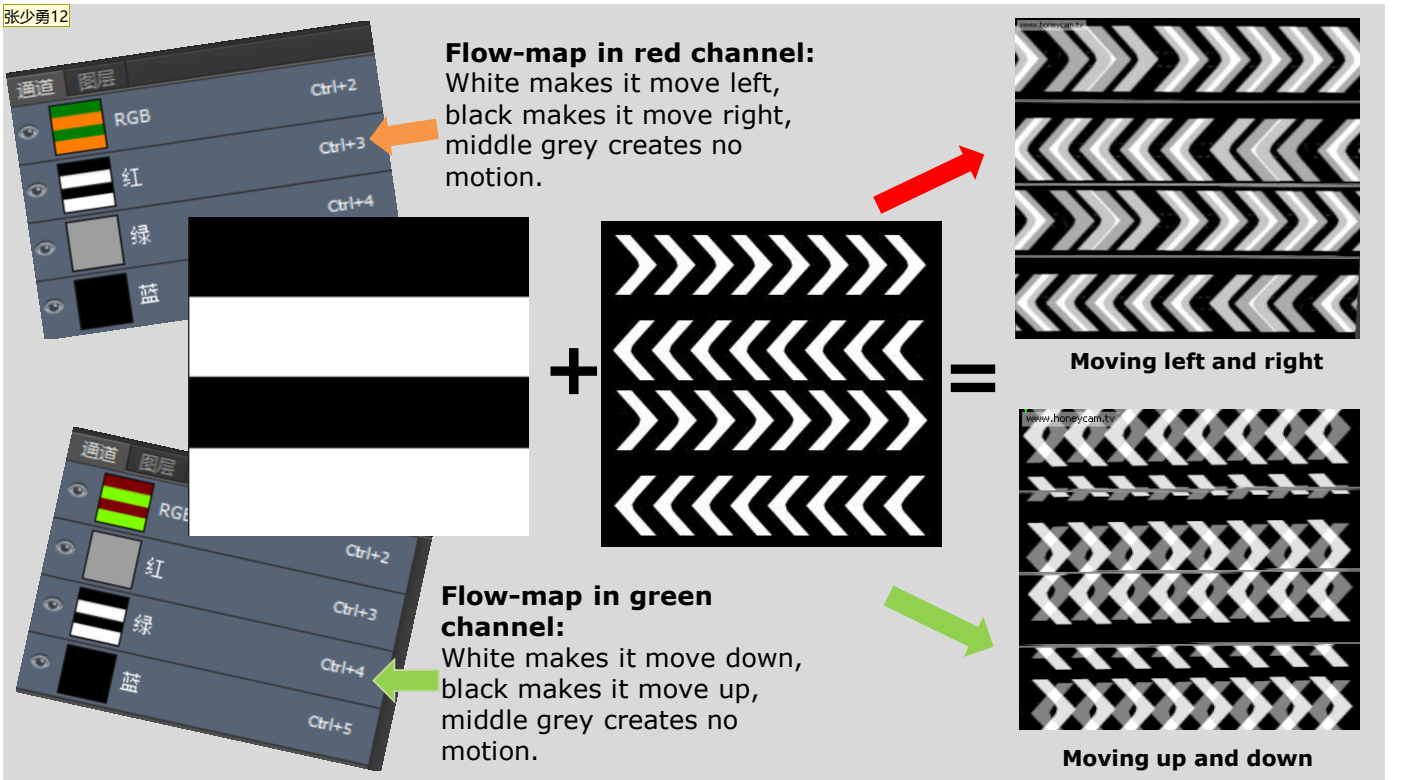


Or up and down?

The Flow-Map Image

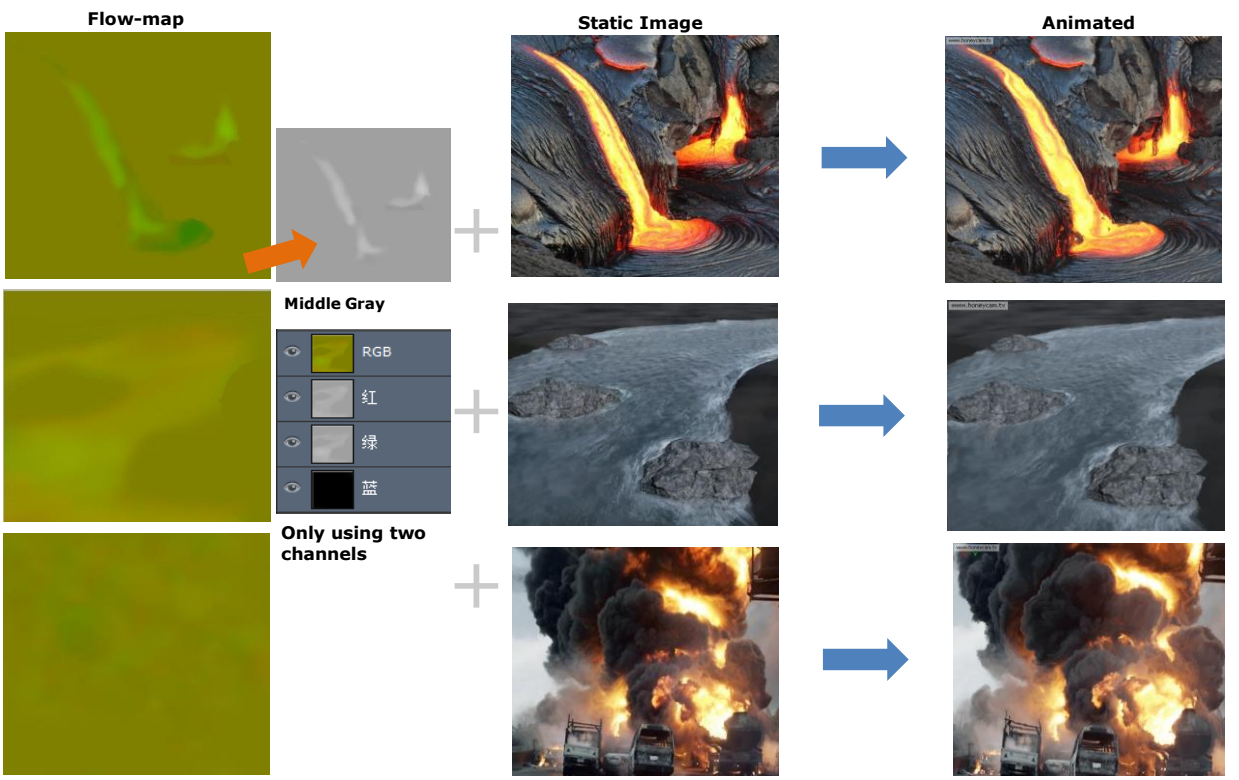
The ‘how’ is by using a flow-map and a flow-map shader.

This is the flow-map image, which is a black and white image in its red and green channel, and black in its blue channel.



The black in its red channel makes the source image move to the right and the white makes it move to the left. (Note that the green channel is 128 middle gray.)

And now switching it's red and green Channel we see the black in its green channel makes the source image move up and white move down. (This time the red channel is 128 gray, so that the motion is only one dimensional)



Here we can see that Flow-maps are able to generate a variety of complex, specific motions such as swirls, smoke, clouds and explosions.

I just quickly grabbed three pictures online: lava, a river and smoke. I quickly smeared out the flow-map textures in FlowMapPainter to demonstrate what we can do with this simple flow-map shader.

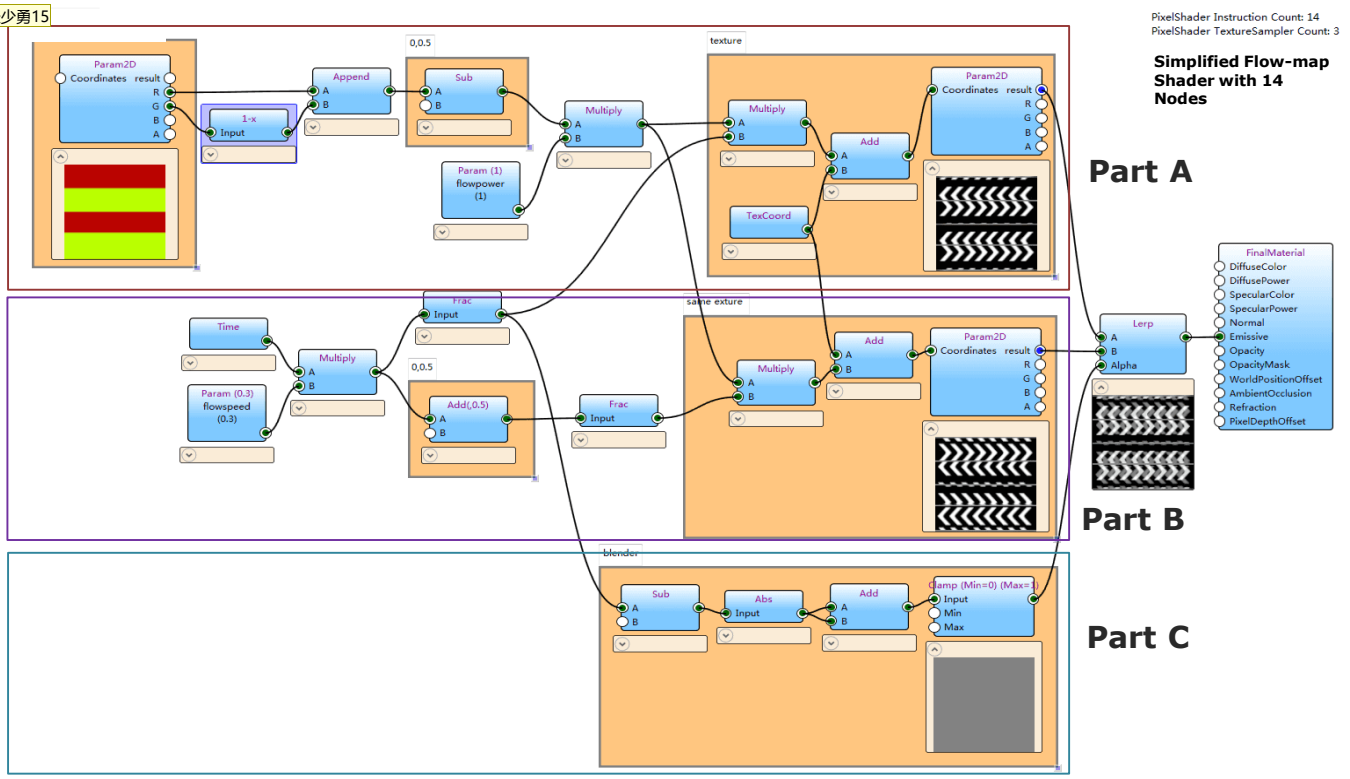
Note that in flow-maps, the 128 middle grey covers the static area of the source image. And the part that creates motion in the flow-maps are the part slightly lighter or darker than the middle grey.

The flow-map is a not a new technique. Our goal here is to simplify things down to the most fundamental level so that our games runs on mobile devices of different capabilities, as users have devices from the low-end to the high-end.

And we want to apply flow-map trick cheaply and quickly.

The flow-map is not a new technique. Our goal here is to simplify things down to the most fundamental level so that our games run on mobile devices of different capabilities, as users have devices from the low-end to the high-end. And we want to be able to do that cheaply and quickly.

Now that we've looked at the basic concept of flow-maps, let's check out an actual flow-map shader and see how to create flow-map textures.

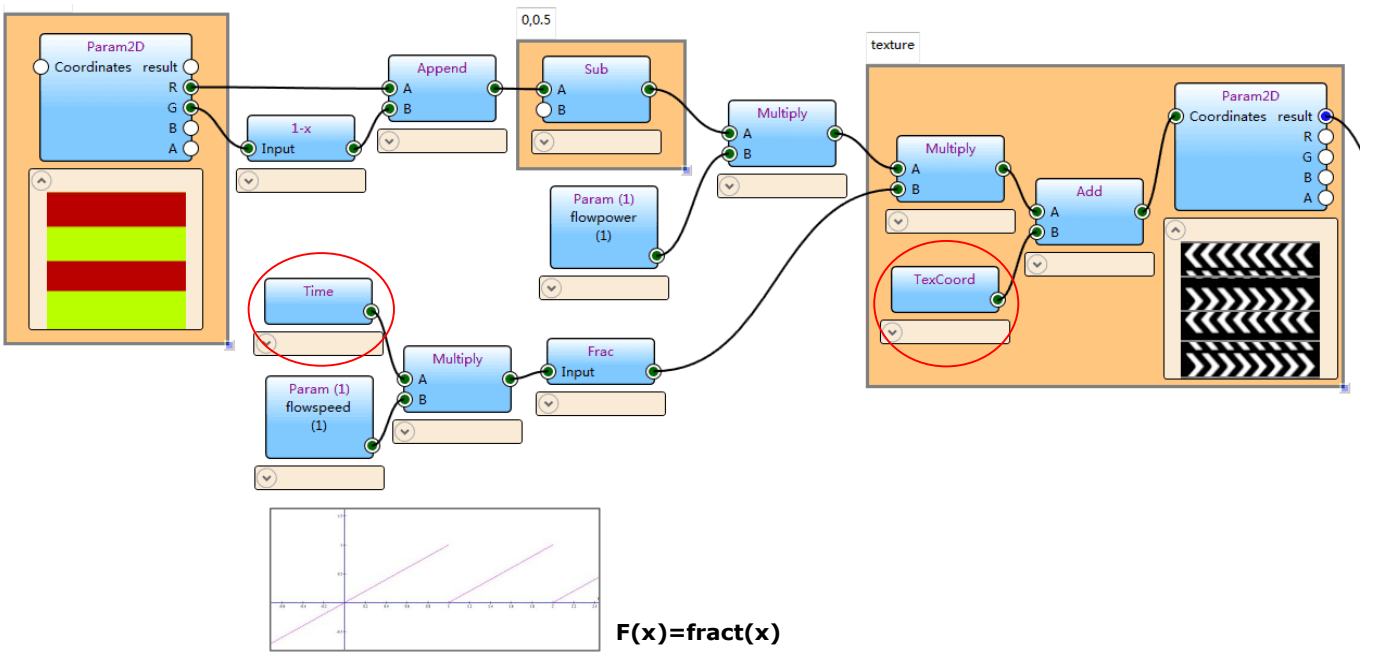


This is a simple 14 node flow-map shader. It works nicely in mobile development as its simplicity would allow the VFX to run smoothly on most mobile devices.

To better understand the shader, I've split it into three parts: A, B and C.

张少勇16

Part A: The source texture's UV is animated by a flow-map image and a Time node



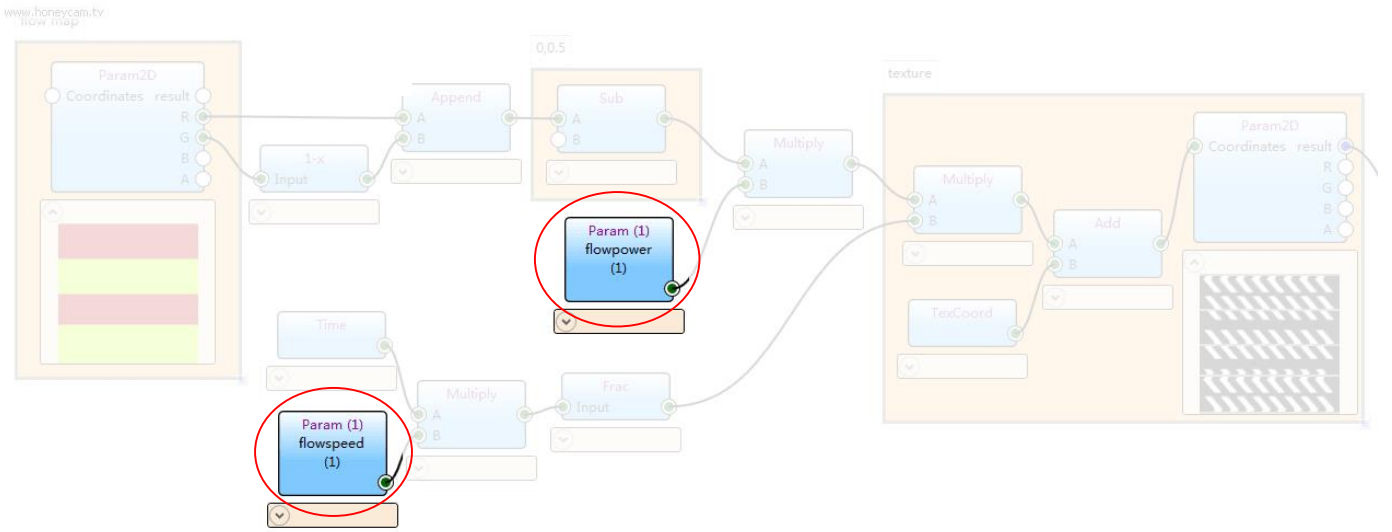
F(x)=fract(x)

This is Part A: as with most shaders, our aim is to manipulate the *Texture coordinates* of the source image. By using the flow-map image multiplied by the *Time* node to create motion.

But note that the animation is popping at the end of the cycle.

张少勇17

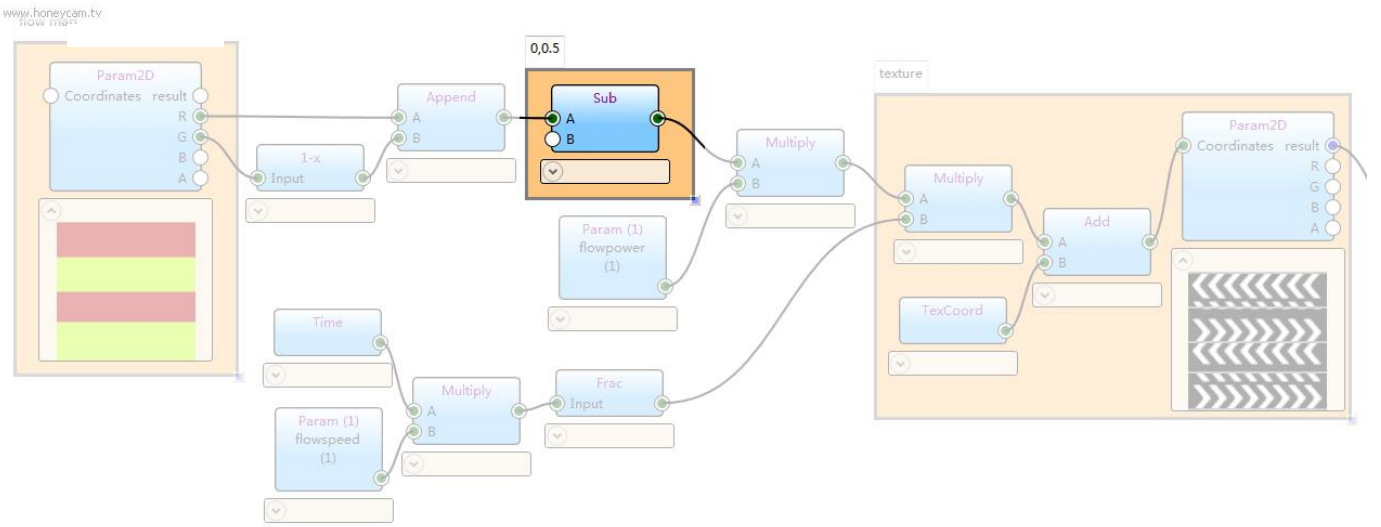
Part A: Flow speed control and distortion control



The only parameters added to the shader are flow speed and distortion volume. The rest is a little simple math.

张少勇18

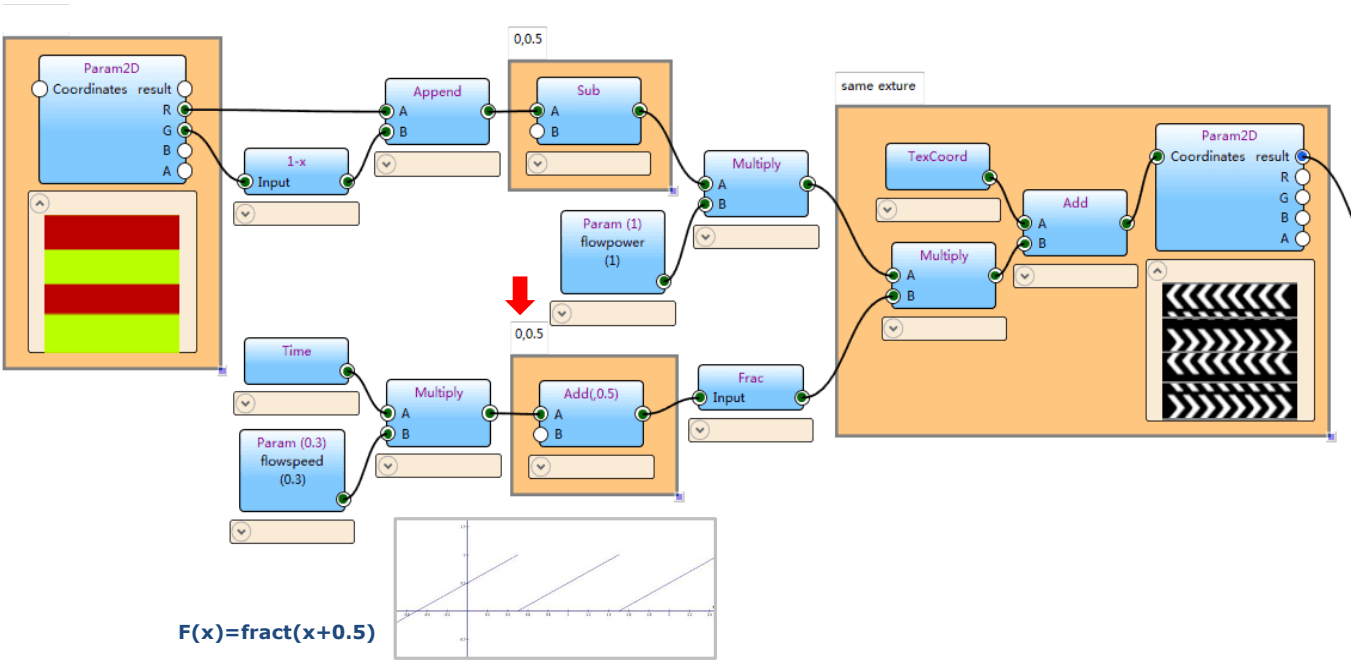
Part A: Subtract 0.5 to get opposite direction. Float range (0 - 1) becomes (-0.5 - 0.5)



The reason for subtracting 0.5 here is to change the float range from (0 – 1) to (-0.5 to 0.5) so that we gain an opposite direction vector, keeping the distortion centered.

张少勇19

Part B: Same as Part A but with 0.5 second delay in motion

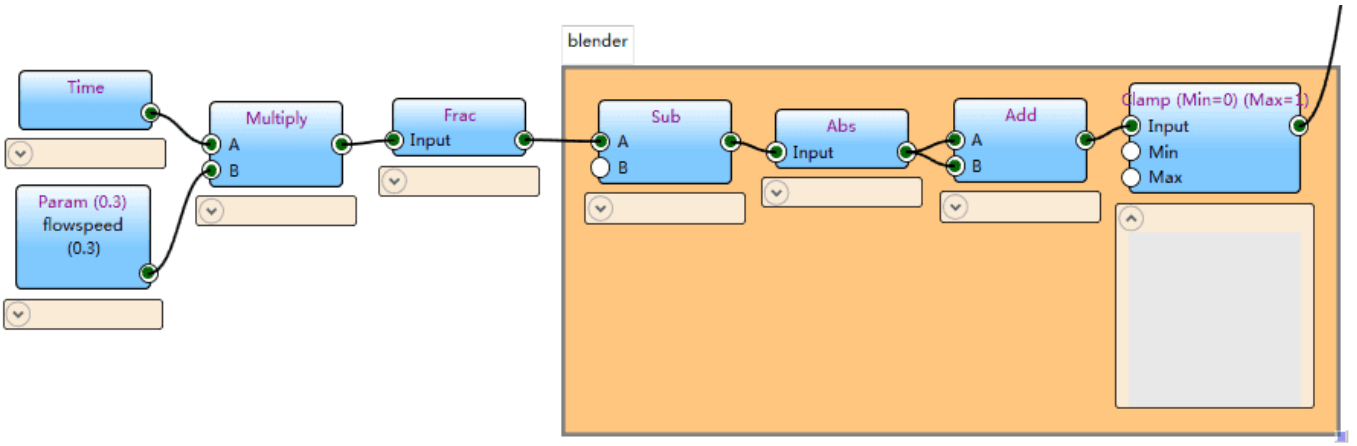


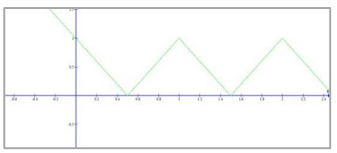
Part B is a duplicate of Part A. The only difference is that a 0.5 second delay has been added.

As we've seen part A is popping at the end of the cycle and is also part B with 0.5 second delay. What can we do to get rid of the popping so we achieve a continuous looping animation?

张少勇20

Part C: Blending Part A and Part B

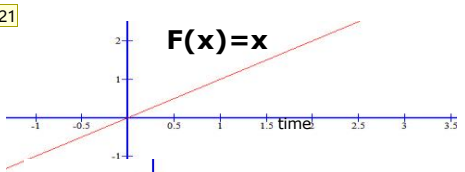


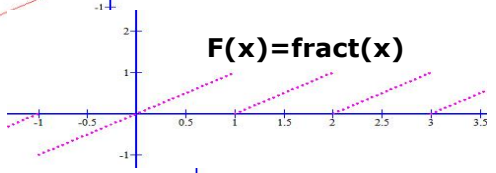


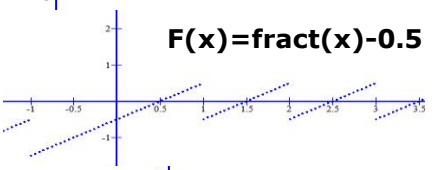
F(x)=abs((fract(x)-0.5)*2)

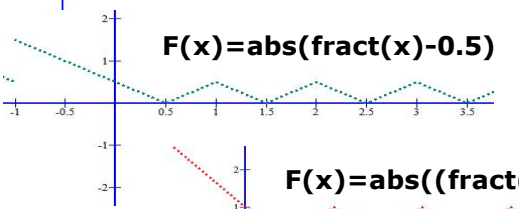
This is where part C comes in. It generates a fade-in and fade-out image that works as an alpha channel or mask.

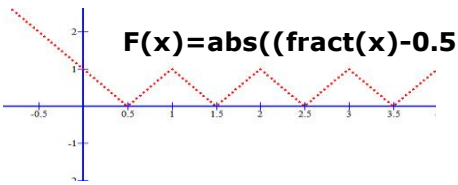
张少勇21







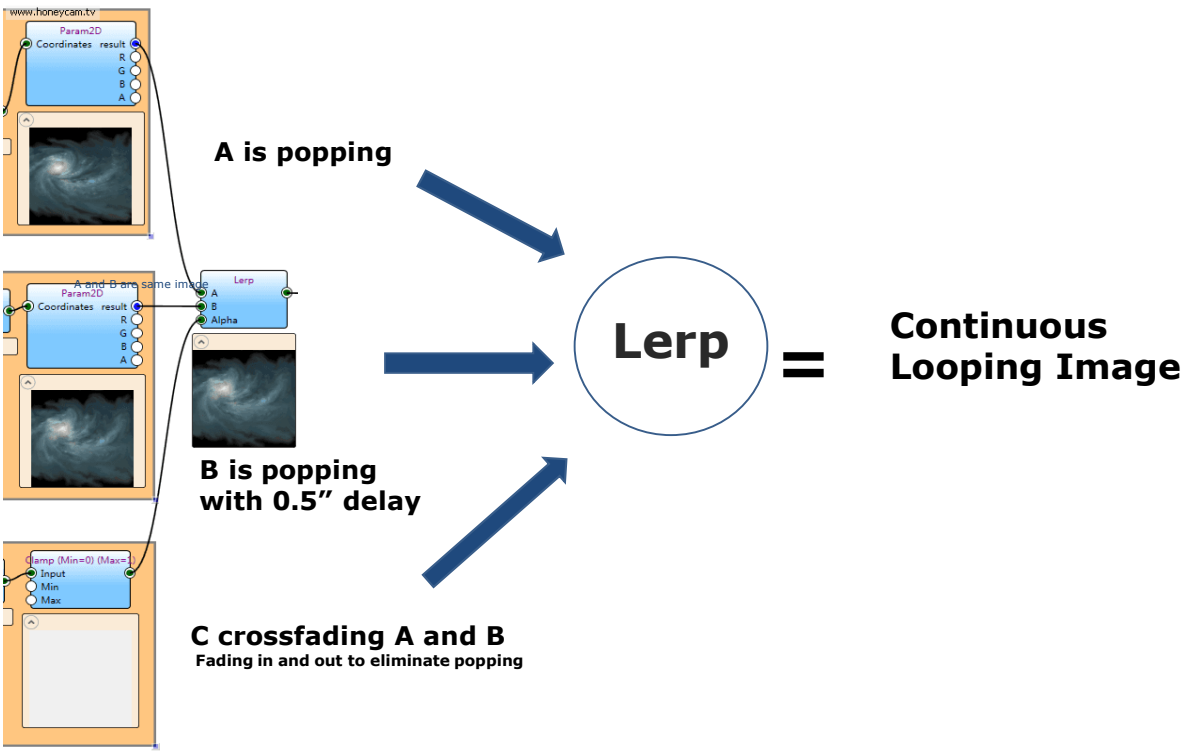




Images are numbers, numbers are images.

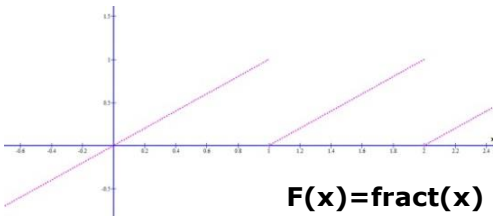
=

Here we see how part C works: step by step, the *Time* node turns to (0 to 1) repetition. Just like a controllable sin wave. And we see how math has turned into an image at the end. Obviously In a node based shader, images and numbers are same thing!



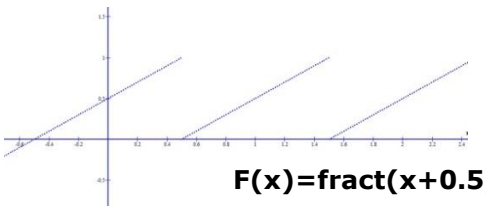
The relationship of the three branches: parts A and B are the same, but the motion of B has been delayed by half a second. part C blends A and B together using a Linear Interpolate node, creating the final effect.

张少勇23



F(x)=fract(x)

A

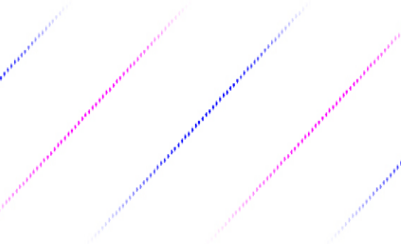


F(x)=fract(x+0.5)

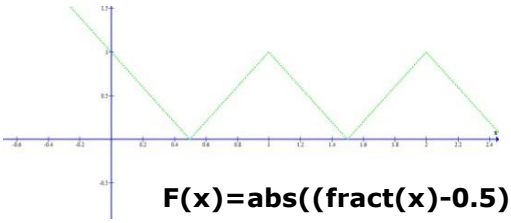
B

lerp

=



The popping parts have been faded out

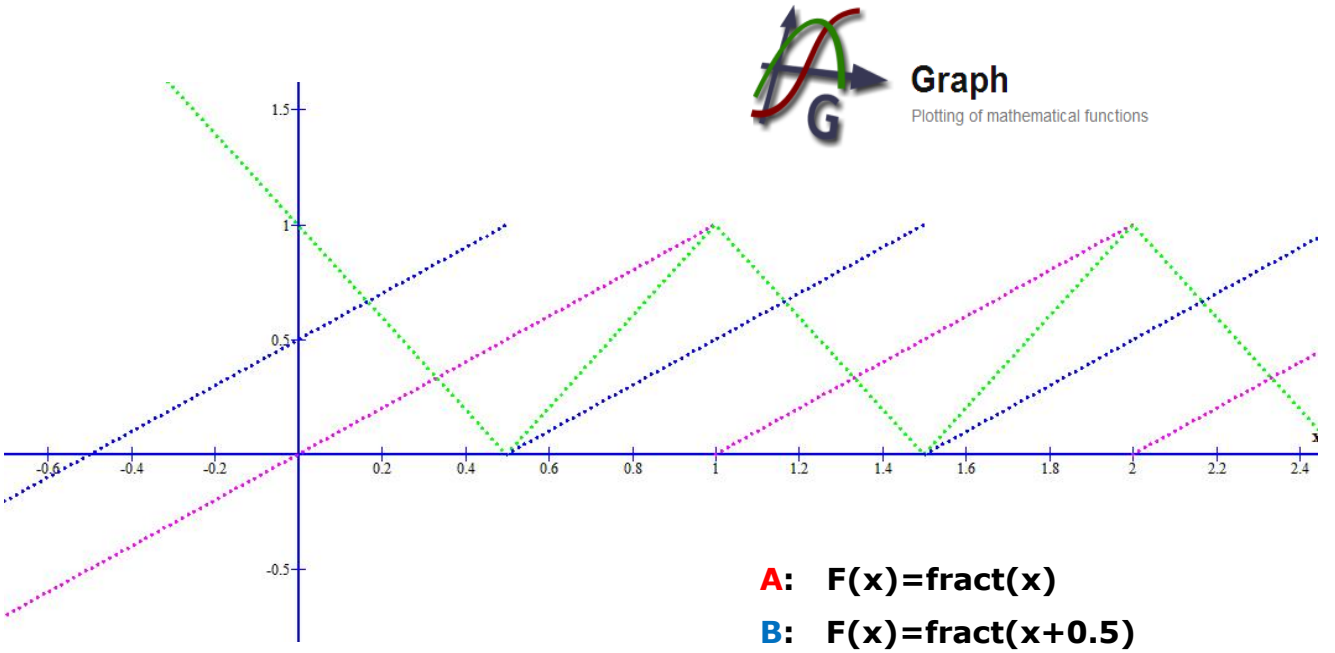


F(x)=abs((fract(x)-0.5)*2)

C

Here we see the whole Shader expressed mathematically.

张少勇24



A: F(x)=fract(x)

B: F(x)=fract(x+0.5)

C: F(x)=abs((fract(x)-0.5)*2)

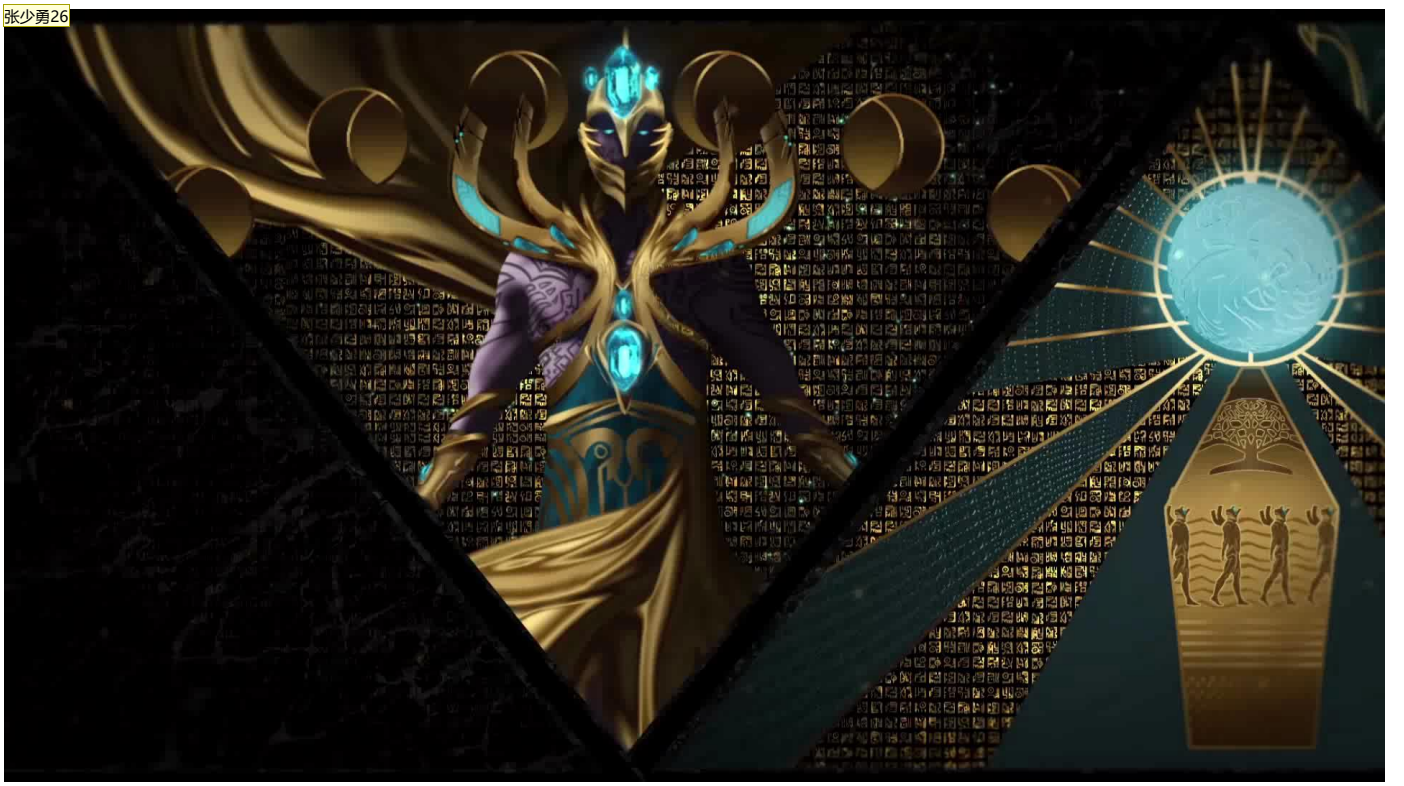
The graphs explain the shader so well. To me, this is the fun part. One picture is worth a thousand words, as they say. The Graph application, as it's name suggests, it turns math into graphics. It has really helped me, to understand each node's function, and to show why that each node is needed for the shader when I trained new vfx artist at work.

it amazes me that images and numbers can be same thing. After figuring out how a flow map shader works mathematically, seeing that how little bit of understanding of math can achieve the visual we wanted, I don't hate math as I once did. I only wish I was better at math then I could create better visual effects.

So how did I apply the flow-map technique into our game?

First let me quickly introduce our game, *Galactic Frontline*, is a sci-fi strategy game that takes you to the heart of a conflict that threatens the entire galaxy. Players will assume command of a ship representing one of three playable species and battle for supremacy against their enemies. Each fleet is equipped with a broad range of combat units and tactical skill vessels. Battles will take place in a wide variety of different environments. Not only can players uncover the secrets of the galaxy in story mode but they can also test themselves against players from all over the world in multiplayer mode.

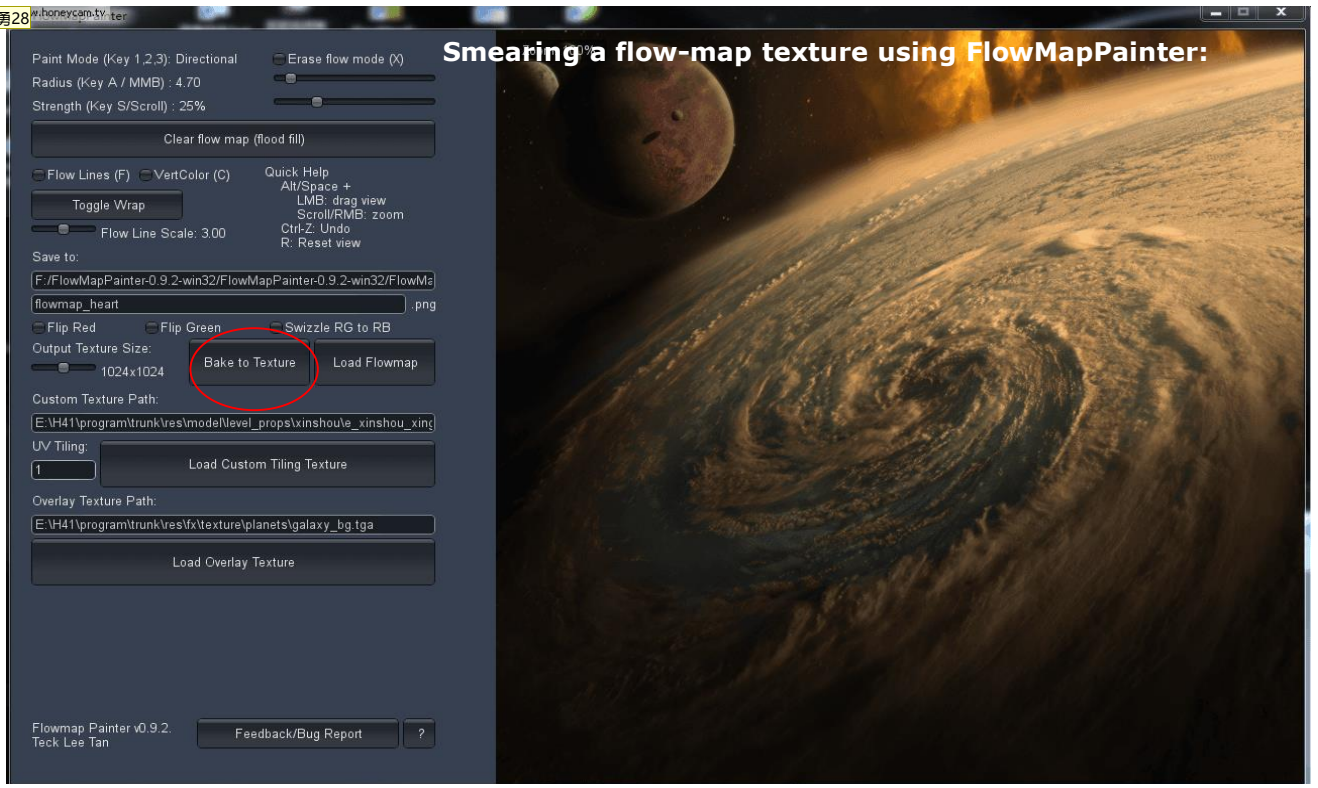
张少勇26



Galactic Frontline is still in production. This teaser will give you an idea of it.

I'll now show you how I've used flow-maps in the game's development.

This is one of the battle scene backgrounds from *Galactic Frontline*. This huge storm raging over the planet's surface was created by adding a swirling motion to a static image with a flow-map shader. Although it covers the full screen, it was cheap to run as it is one single layer. This effect was achieved in one hour or so with the flow map shader.

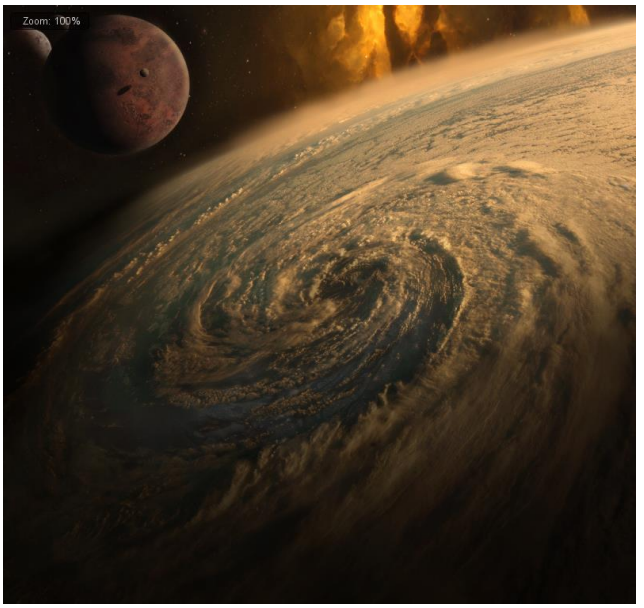


Having spoken about flow-map shader, let’s take a look at how to create flow-map textures.

In this case I imported the source image, the swirling storm, into FlowMapPainter as a background and smeared out the flow-map texture on top of it, almost as if I were painting the movement.

张少勇29

From the source image to the flow-map

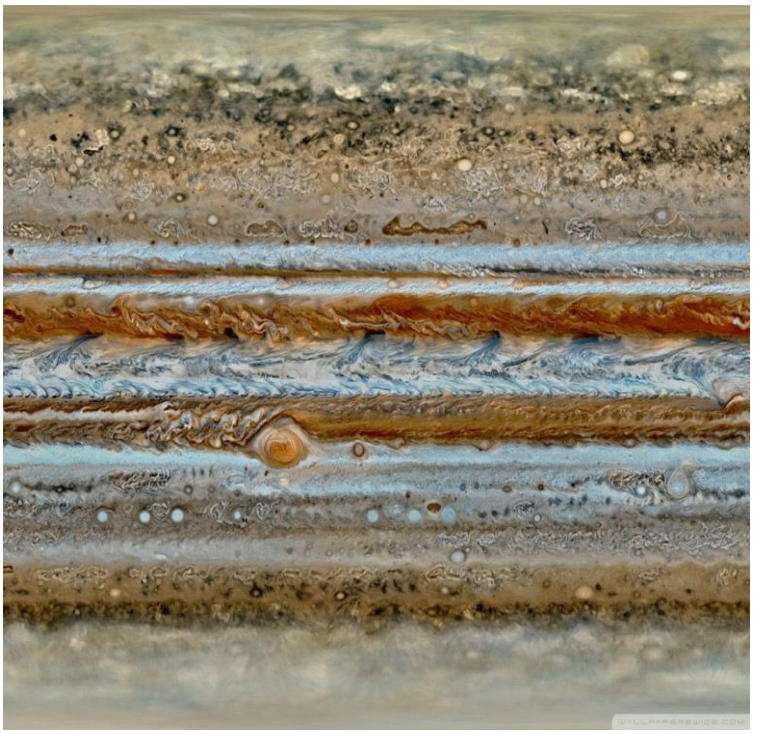


Here is the flow-map texture smeared out in 10 minutes using FlowMapPainter!

Now let’s consider more complex motion, with more details to think about, like the atmospheric phenomena on Jupiter, with all its raging storms.

张少勇31

Source Image

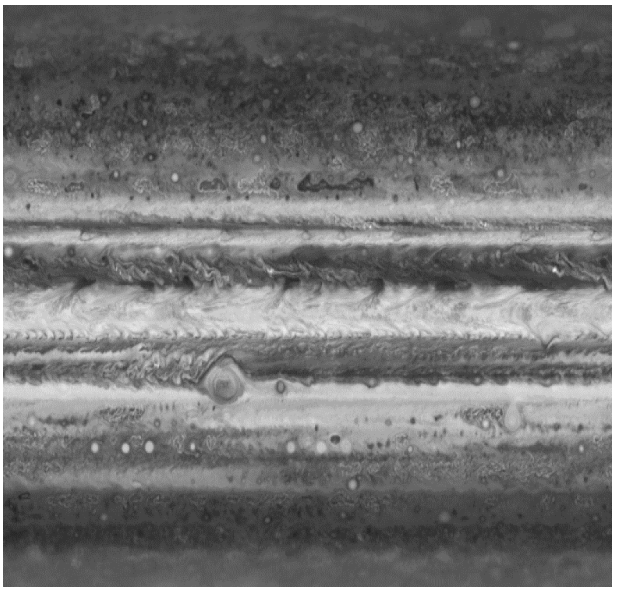


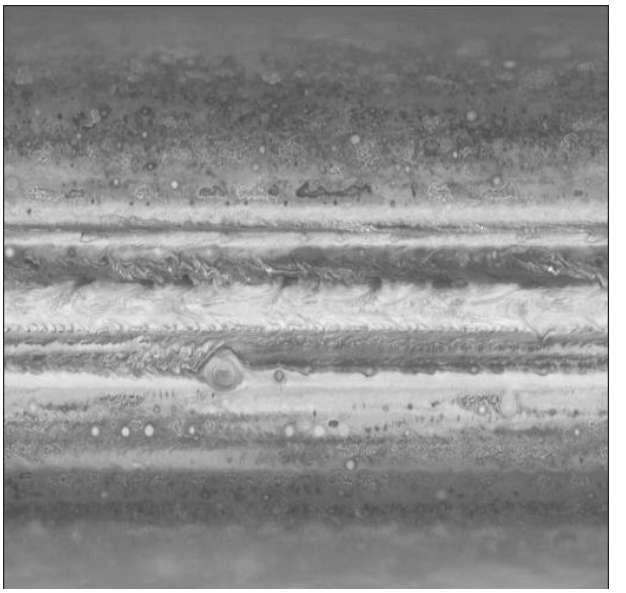
This is the source image of Jupiter, a very complex texture. It would take too long to brush out all the detailed motions in FlowMapPainter. Is there anyway to cheat?

张少勇32

Making flow-map texture out of the source image

The red channel of the source image → The red channel of the flow-map image



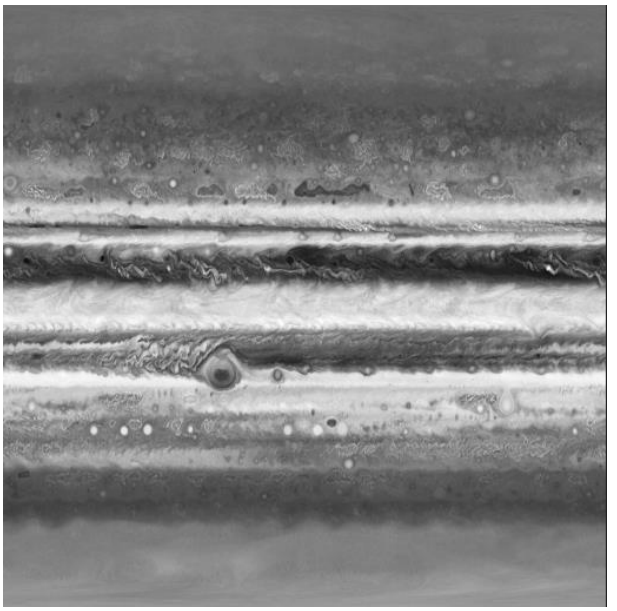


What I did is this: Using Photoshop, I adjusted the grey scale of image's red channel, Gaussian blurred one pixel, and brushed the area with 128 grey where I wanted there to be no motion.

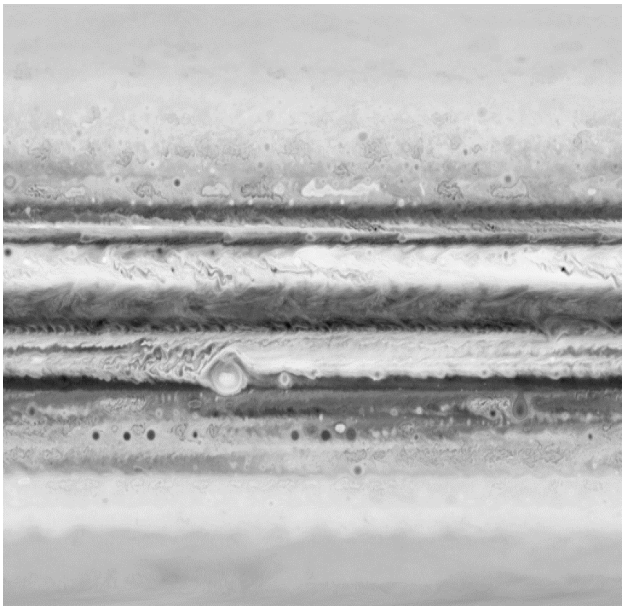
张少勇33

Inverted

The green channel of the source image



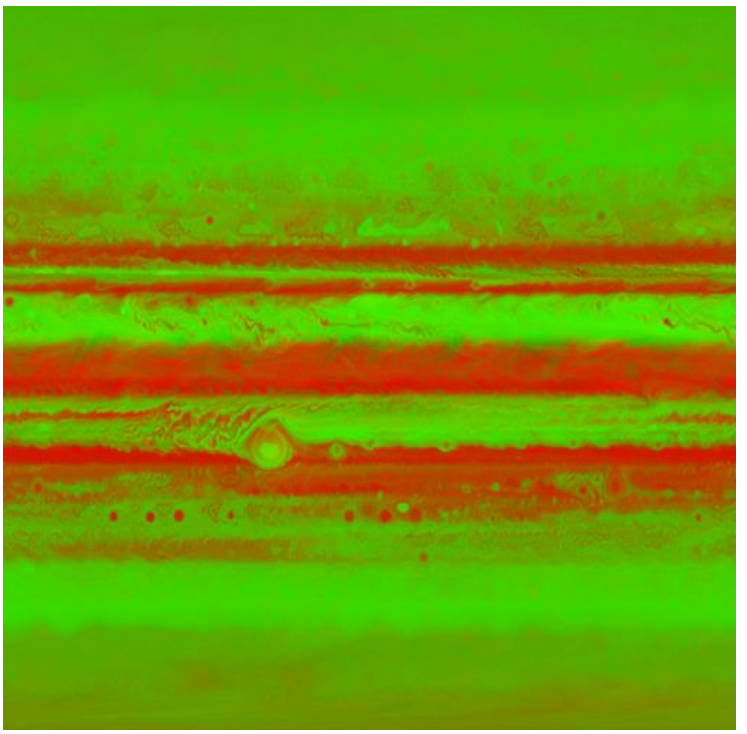
The green channel of the flow-map image



Then I inverted the green channel, and did the same.

张少勇34

Flow-map made out from a source image



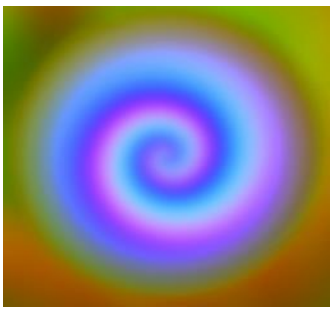
The result, a flow-map texture made out of a source image in Photoshop without brushing any of the details.

There are so many quick and easy ways in which we can make flow-maps textures as long as we know the basic principle how flow map works.

Flow-maps generate very specific or random motion in one single particle or mesh as the smoke effect in the image shows.

张少勇36

A flow-map used with particles effects



Here we see how this flow-map has made the dots into swirling lines in the vfx.

They can be used to animate environmental features such as rivers, or water falls.

Anything that flows!

Applying a flow-maps to Van Gogh's *The Starry Night*

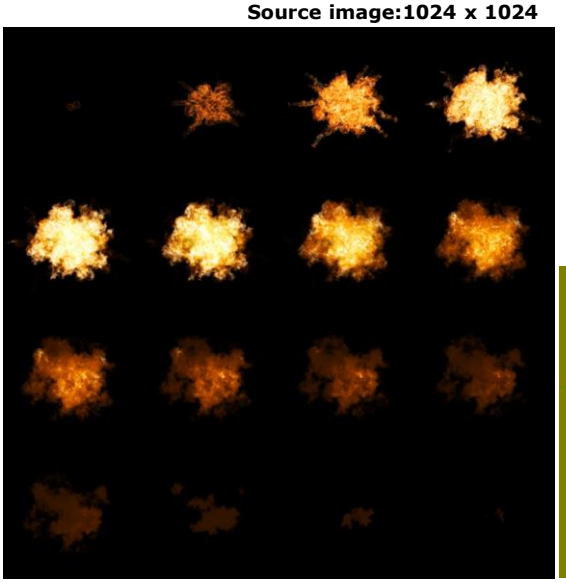
Let us imagine what Vincent saw in his mind's eye while painting the starry night.

We 've seen the simple version of a flow map shader, let's take one step further: replacing the single frame flow-map with a subUV texture, what change should we make to the shader?

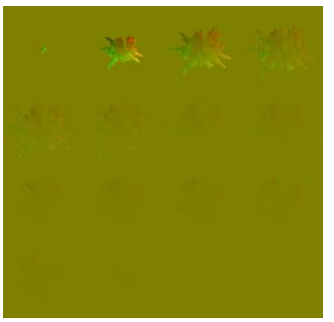
张少勇40

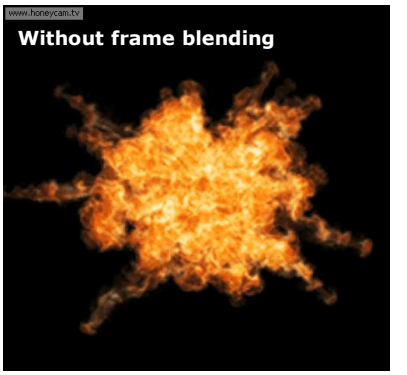
Limit the particle subUV texture to 1024 x 1024, 4x4 =16 frames.

Source image:1024 x 1024



Flow map512 x 512





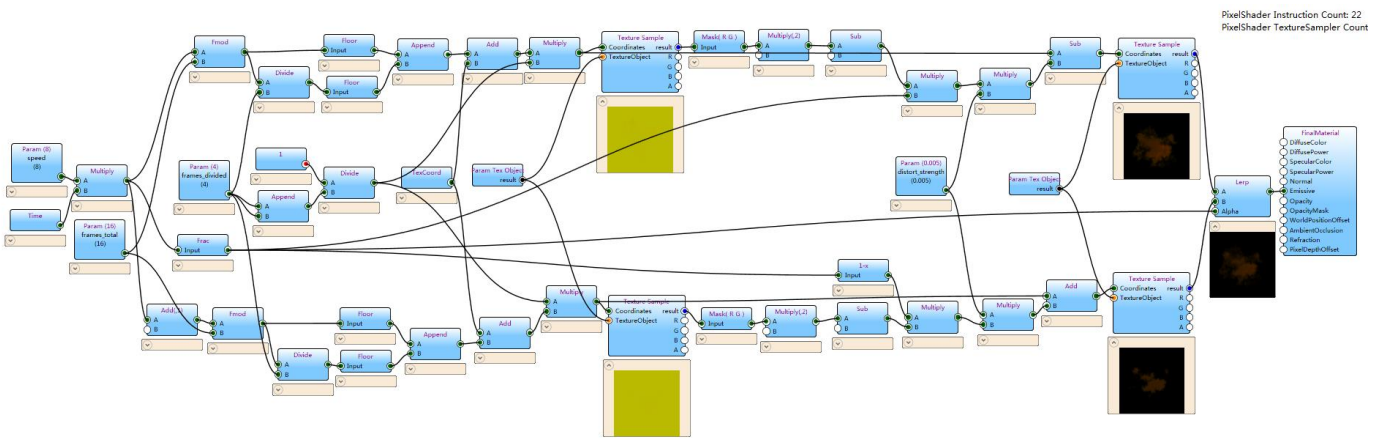
Cellphone is much smaller than Xbox. for a mobile project, texture memory is big concern. Our project Galactic Frontline restricts the subUV texture to no bigger than 1024 x 1024 pixels. Divided by 4, each unit would be 256 x 256 pixels. Smaller than that, the resolution will not be acceptable. So, 16 frames is all we've got for subUV particles, that is not enough frames for a smooth animation.

But with a flow-map shader for frame blending, we're able to use small sizes and small numbers of subUV textures to achieve smooth animations in mobile games.

张少勇41

Using flow-map for frame blending

Flow-map shaders for frame blending enable us to use small size and small numbers of subUV images to generate smooth, evolving images for mobile games using less texture memory



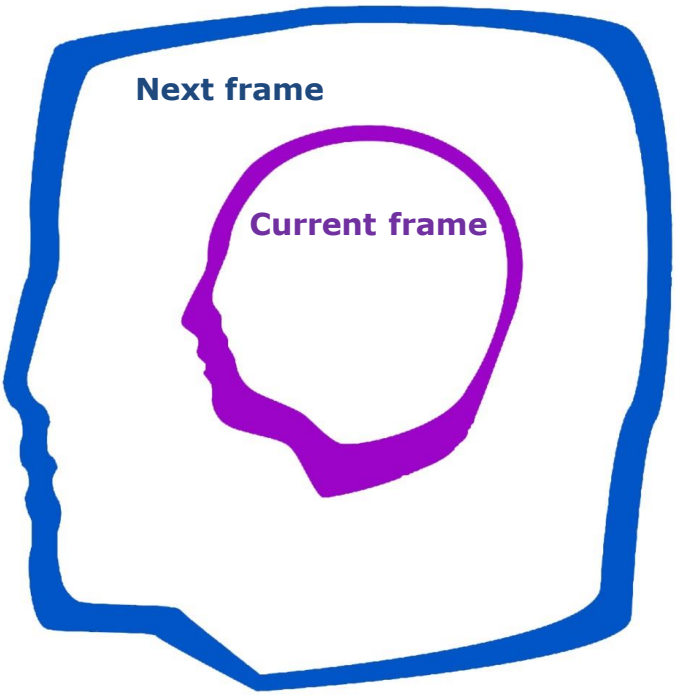
This is the flow-map shader for frame blending using subUV textures. It originally used for AAA games. I tailor it for mobile as you can see, it has only the emissive input for the whole shader.

it is much more complex than the single frame one. But it has the same basic concept of part C blending parts A and B.

(This was the exact problem I couldn’t resolve at Sledgehammer games until I saw the webpage of Klemen Lozar.)

“The aim is to extend the utility of animated textures by distorting them with motion vectors to procedurally generate the in-between frames”

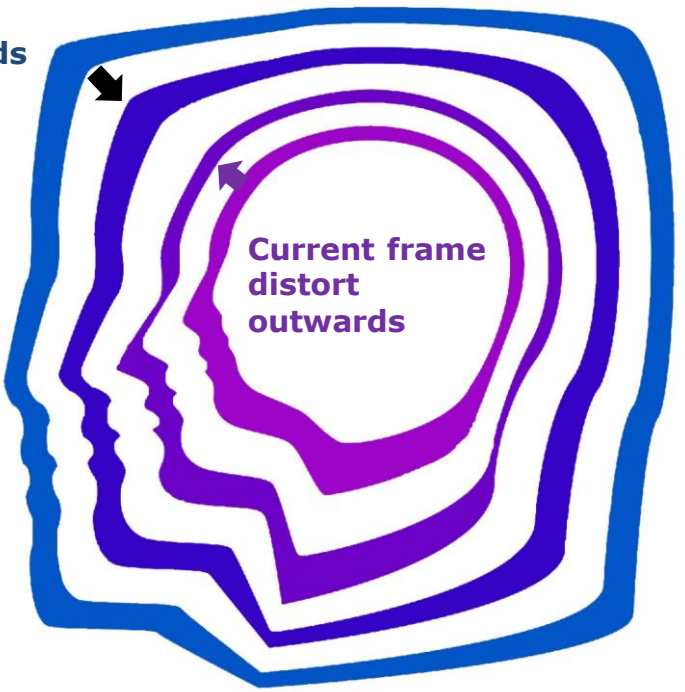
张少勇42



Say we have this two frames: If the inside frame distorts outwards and the outside frame distorts inwards.

张少勇43

Generated in-between frames.

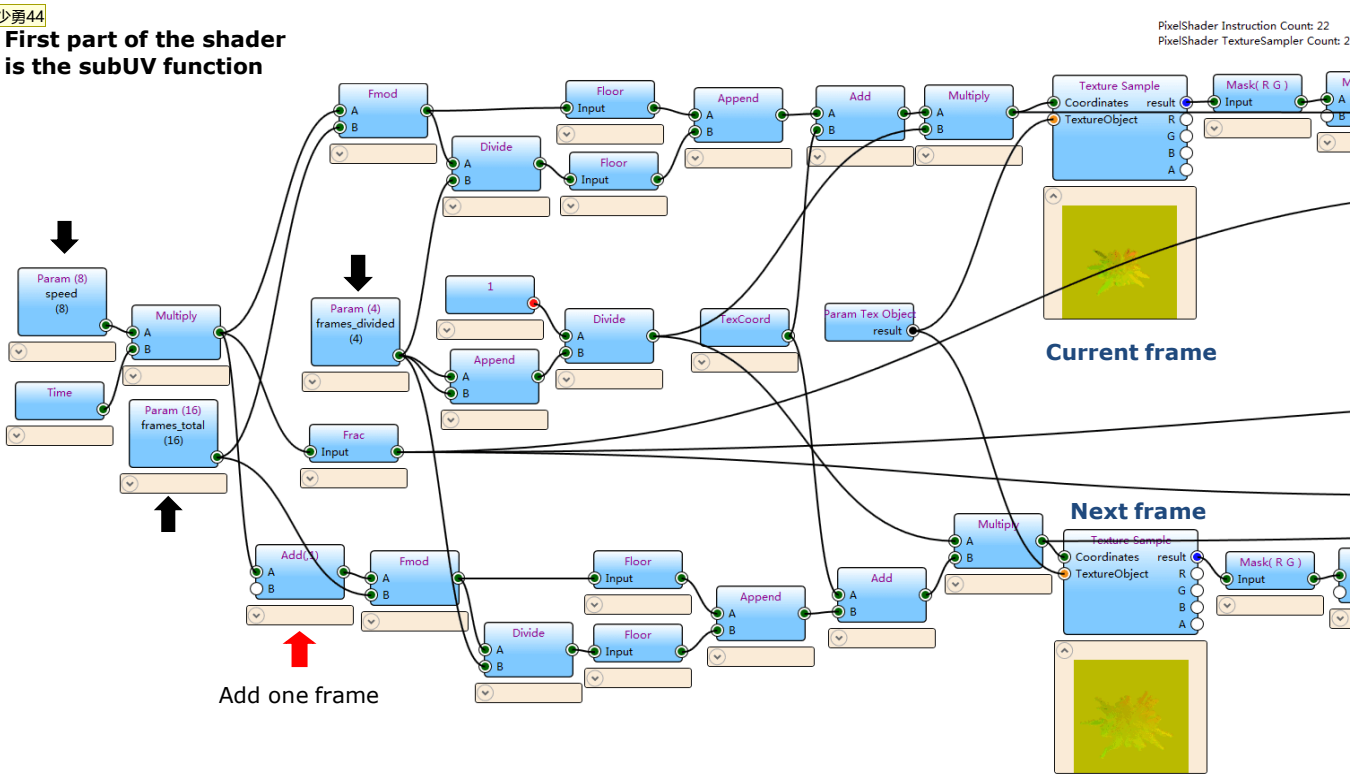


The image now makes more sense with the frames generated in-between.

(A rounded head gradually becomes a squared head)

张少勇44

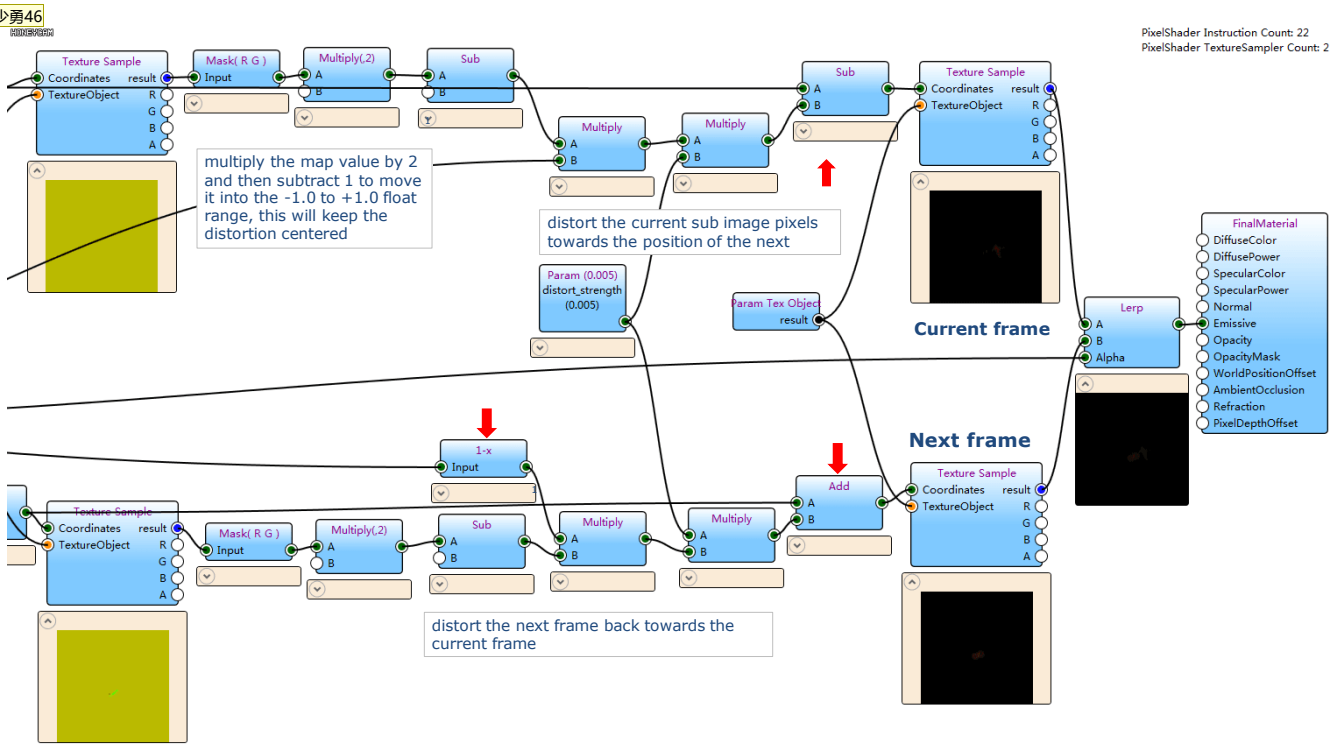
First part of the shader is the subUV function



This is the first part of the shader. It produces custom Particle subUV function for frame control. There are three key parameters to control the frame speed, total frame number and subUV image number.

The upper row is “part A”, the current frame. The lower row is “Part B”, the next frame. Note that 1 has been added to the “Next” frame value to offset the SubUVs by one frame. If you want know more detail of the first part of the shader:

Ben Cloward gave a very detailed explanation of the “particle subUV ” function in his talk at GDC 2016: Atlas walk Example



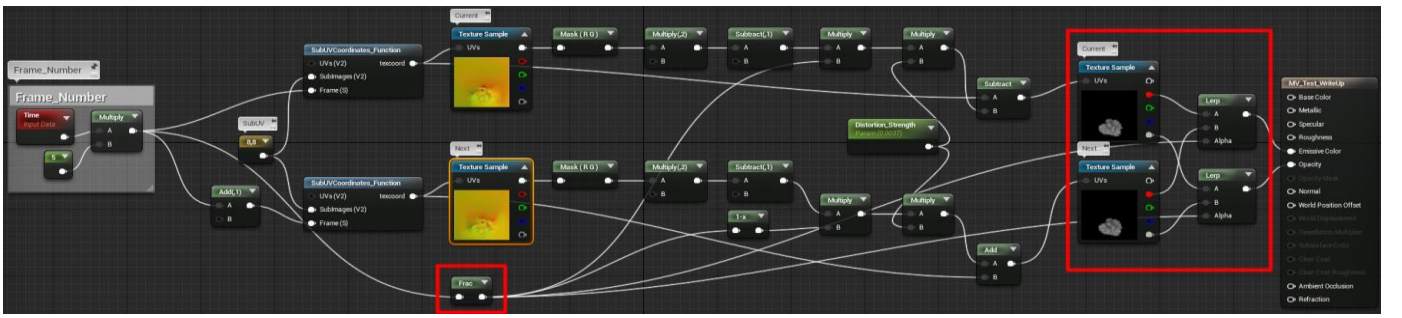
The second part of the shader: In addition to interpolating from one sub image to the next we need to distort the current sub image pixels towards the position of the next and similarly distort the next one back towards the current one so they look like meet in the middle.

It is hard to grasp the logic. I just want to know enough to use it to achieve the visual, and don’t want go too deep because that will drive an artist crazy. But,

Flow-map shaders for AAA games can be more complex for more detailed visual

It need to use multiple applications: Unreal 4, FumeFx for 3ds Max and After Effects with a Twixtor pro plug-in.

And It takes time and effort to render motion vectors (flow-maps) using FumeFX, Maya or Houdini



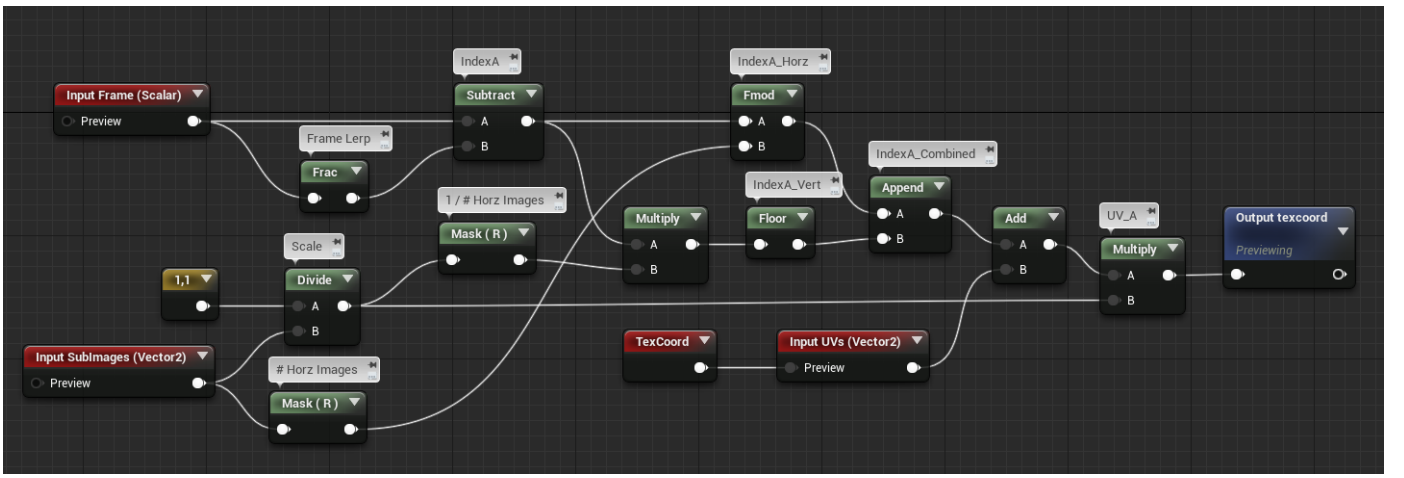
http://www.klemenlozar.com/frame-blending-with-motion-vectors/

Thanks to Klemen lozar! who shared his flow-map shader for Unreal. There is very detailed explain of the shader logic on his website. Check the page http://www.klemenlozar.com/frame-blending-with-motion-vectors/

This shader for AAA games can be much more complex while achieving greater visual effects. But It needs to use multiple applications: Unreal, FumeFx for 3ds Max or Maya, and After Effects with a Twixtor pro plug-in.

Output texture coordinates instead of an RGB channel

Modifying the existing SubUV functionality that comes with Unreal 4 so it outputs texture coordinates instead of an RGB channel.

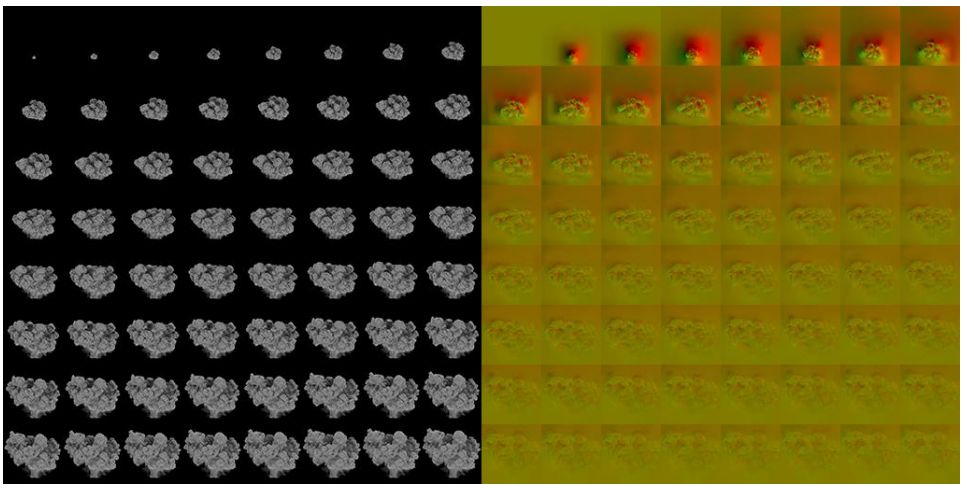


it needs manually make the particle subUV function so it outputs texture coordinates instead of an RGB channel.

张少勇49

Flow-map shaders for AAA games

SubUV textures for flow-maps and source images, at least two 2048x2048 textures with 28 MB memory cost.

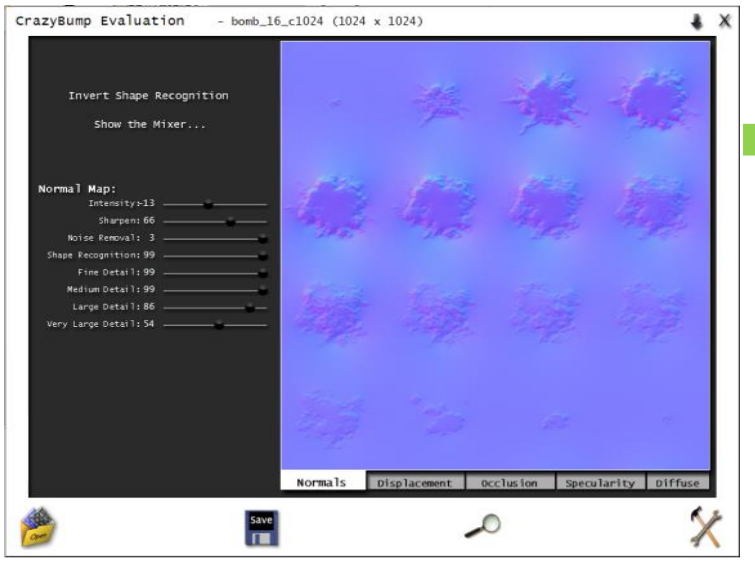


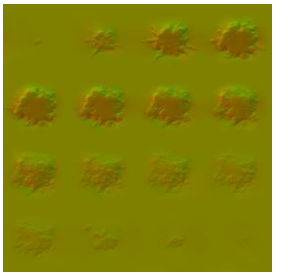
AAA games allow for much larger subUV textures. Take *Call of Duty: Advanced Warfare* for example, The subUV textures can be up to 2048 x 2048 pixels, even 4096 x 4096, which was not a problem for Xbox.

On a mobile game, such as *Galactic Frontline*, we're normally limited subUV texture to 512 x 512.

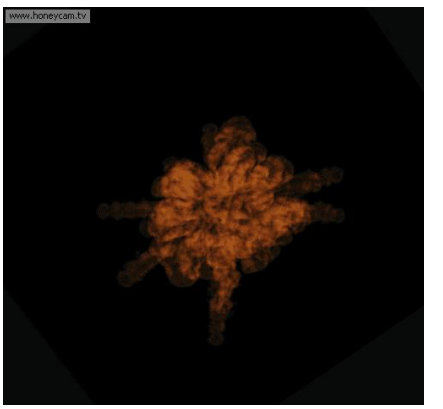
张少勇50

5 minutes with CrazyBump vs 5 hours on 3Dmax rendering





A flow-map made from CrazyBump in 5 minutes



The distortion effect has no big difference on small screen

it could take 5 hours or more rendering out a set of source and flow-map subUV textures manually in Maya, 3D Max or Houdi. In game production, Most of the case we already have the subUV source texture, all we need is the matching subUV flow map texture. A quick way is using CrazyBump to fake a flow-map texture out of a source texture we already have. it only cost a few minutes. Just use the red and green channel of the Normal map generated from the source texture. For distortion purposes, the texture is good enough.

When you have to get things done in a hurry all the time, it is good to have a way to cheat. Mobile projects are generally less budget, and short developing cycle and for much smaller screen.

张少勇51

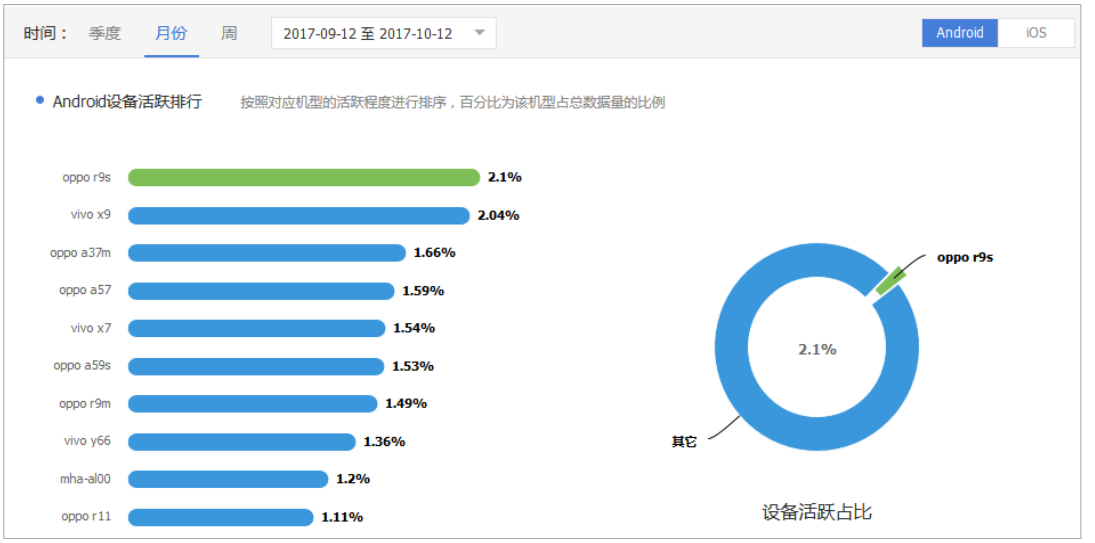
Frame blending flow-map shader used for explosion effects.

This intensive explosion effects in Galactic Frontline uses a flow-map shader for frame blending. It still runs at a decent frame rate considering there are so many effects being played at the same time. And it was made very quickly with a subUV flow map texture made from CrazyBump.

张少勇52

Only apply AAA techniques that are going to work on the majority of targeted users' devices, including low-end phones. Good VFX is the VFX that runs smoothly on all levels of devices.

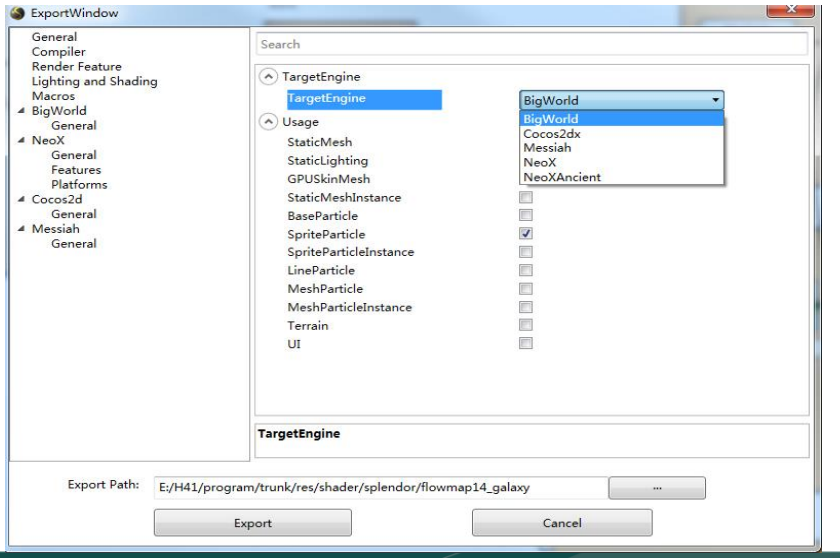
The chart shows what kinds of cellphones our potential users have.



In a conclusion, for mobile games, the easier the technique, the better things will run on lower-end devices.
There is a lot of diversity in terms of the phones and devices that users play on, and we want to make sure that everyone can enjoy the game.

This chart shows what kinds of cellphones our potential users have, mostly are the low end cellphones.

Outputting cross-platform and cross-engine material



programmer

TA

Game designner

NetEase as a large company, we have lots of different projects going on at the same time, and different teams often use different engines in the development process. Our engineers and technical artists were able to ensure the materials that we create, such as flow-maps, can be shared between teams, and can be exported in such a way that it works across various platforms and game engines.

There are so many talented people all working towards the same goals and I want to say thank you to, especially our programmer李冰, TA吴振丹 and game designer Matthew Aitken.

NetEase is a great company to work at, (anybody here want to work in China? please let me know). And Thank you to GDC and all those who have shared their expertise! I am standing here talking because I've watched a couple of the previous GDC presentations ☺

at the end, Let’s see a few more flow-map examples in some other projects from NetEase to see its broad use on vfx, character, environment and UI.

Overview

Pros: Simplified flow-map shaders for mobile games are quick, cheap, effective and relatively easy to create for very specific motions.

Flow-maps texture can be created in FlowMapPainter, CrazyBump, or Photoshop easily. All we need to know is the basic principle that the middle gray creates no motion, black creates motion in one direction and white in the other.

Flow-map shaders for frame blending enable us to use small size and small numbers of subUV images to generate smooth animated images for mobile games with a low texture memory cost.

Cons: Motion patterns are repeated.

Lots of negative space in particles overdraw.

Lastly, Let us quickly recap the good and bad about the flow-map technique:

Thank you again for being here. I hope this simple and easy technique for mobile development would be helpful to you. Any questions?

Thank you!

Shaoyong (Abel) Zhang
Vfx artist from NetEase Games

www.abelzhang.com

as6769@yahoo.com