

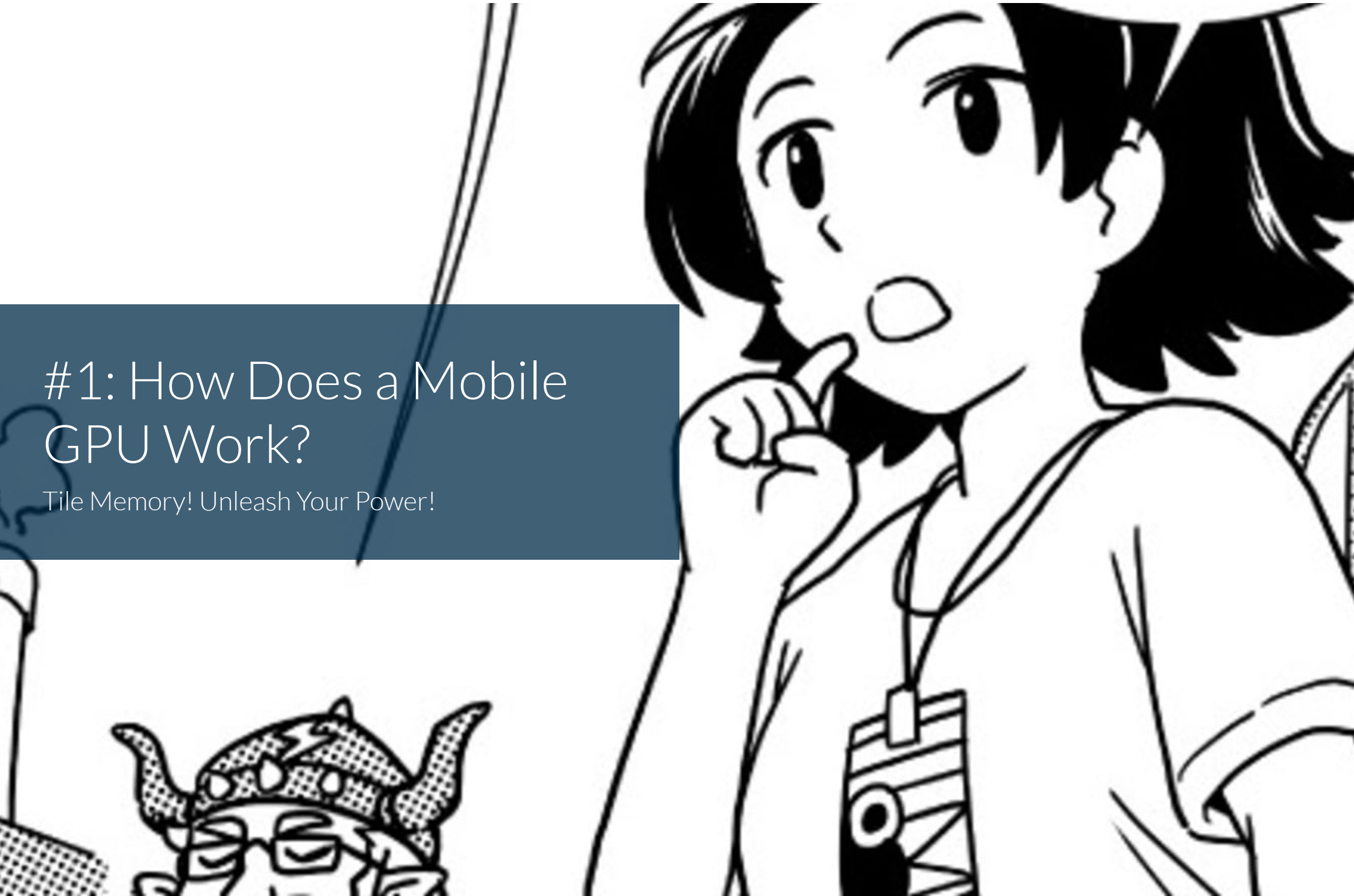
A manga-style illustration featuring a character with a long white beard and a purple horned helmet in the upper left. In the lower right, a character with dark blue hair and a yellow shirt is shown with a surprised expression, one arm raised. In the lower left, a character with long white hair is partially visible. The background consists of a purple and blue grid pattern.

The Arm Manga Guide to the Mali GPU

Meet Dr. Arm: The Graphics Superhero

#1: How Does a Mobile GPU Work?

Tile Memory! Unleash Your Power!

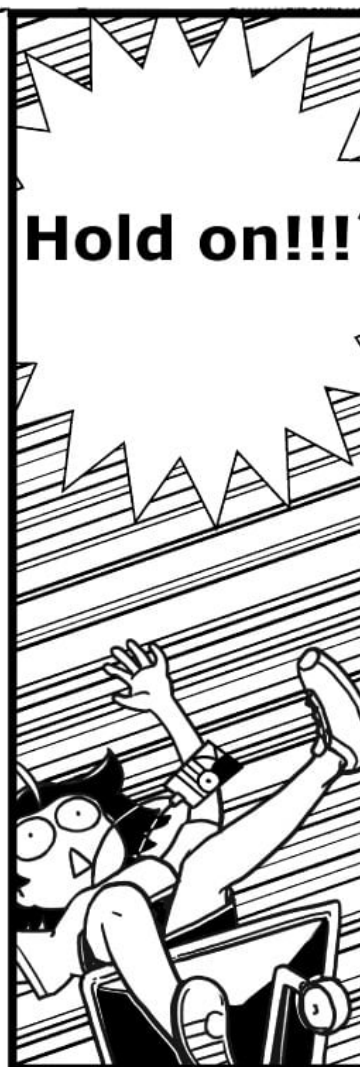


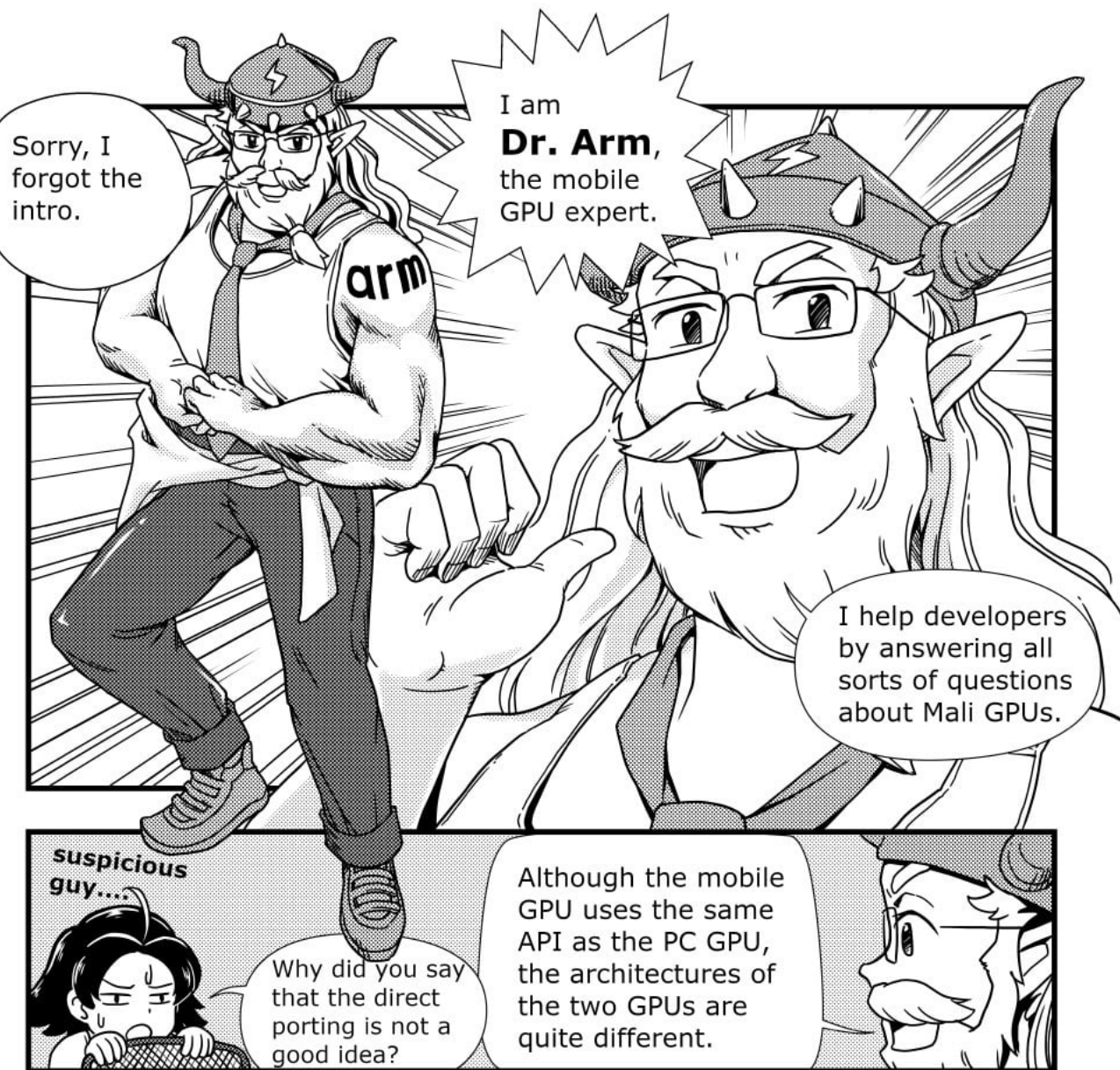
How does a Mobile GPU Work?

Comic: Ikaridon Yu

Playing
games on
mobile is so
convenient.











Here is the inside of a PC GPU.

Wow! That is a huge power station.



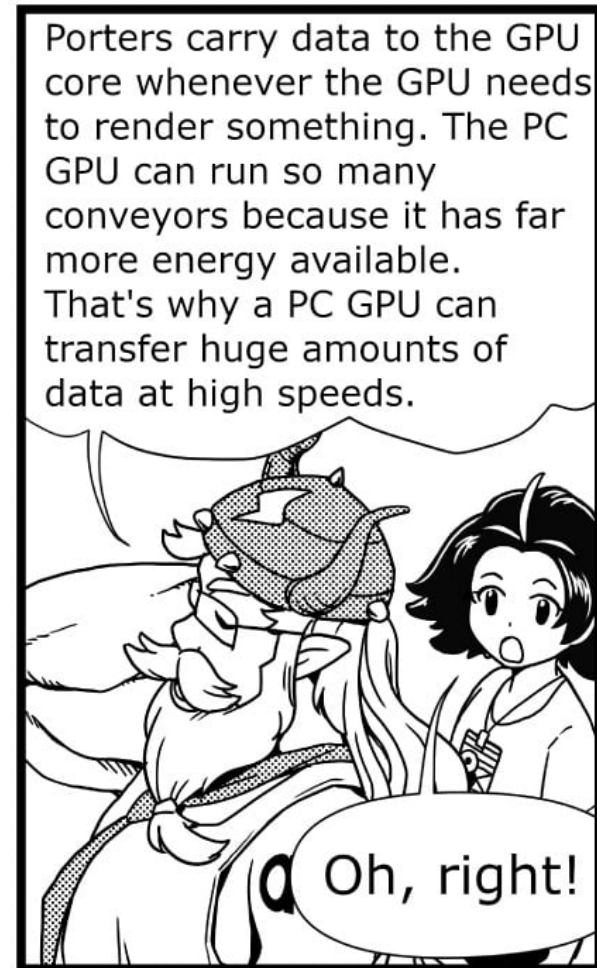
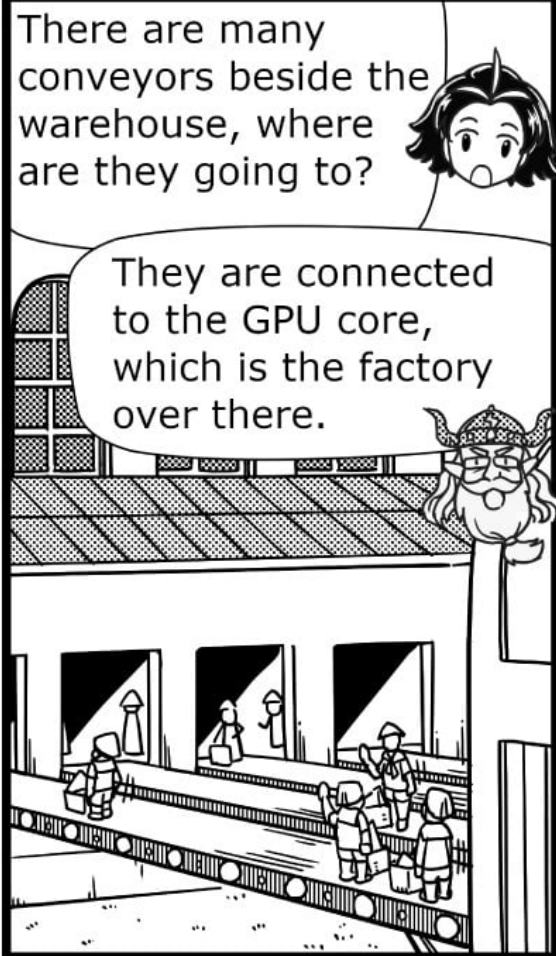
That's right. That's why the PC GPU doesn't need to worry about power consumption.

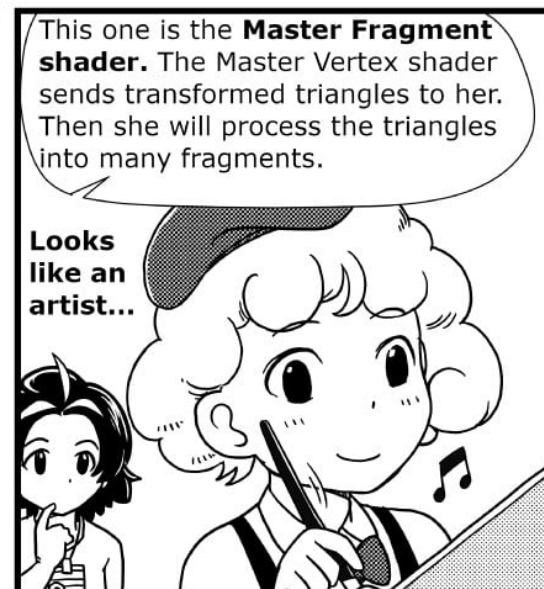
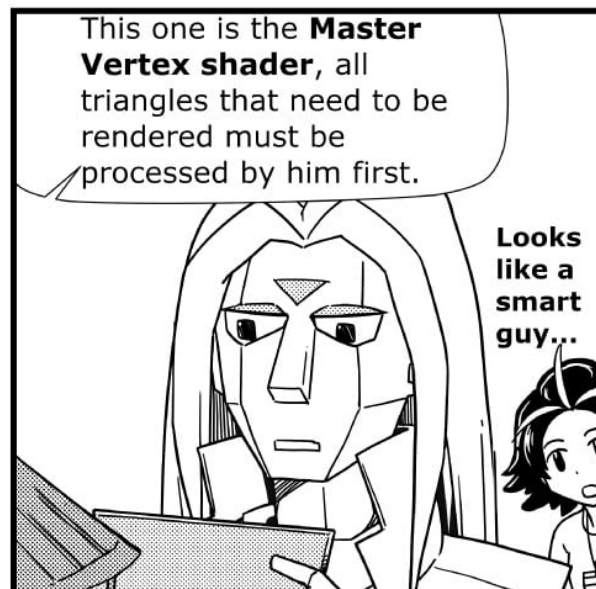
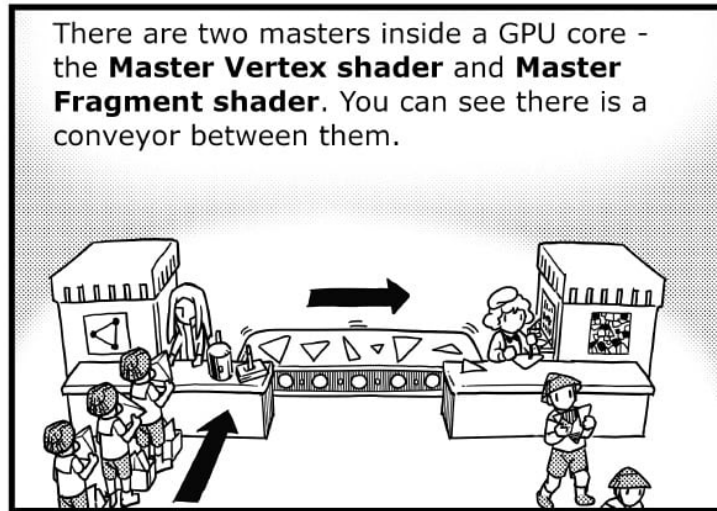


Did you see a big warehouse over there?

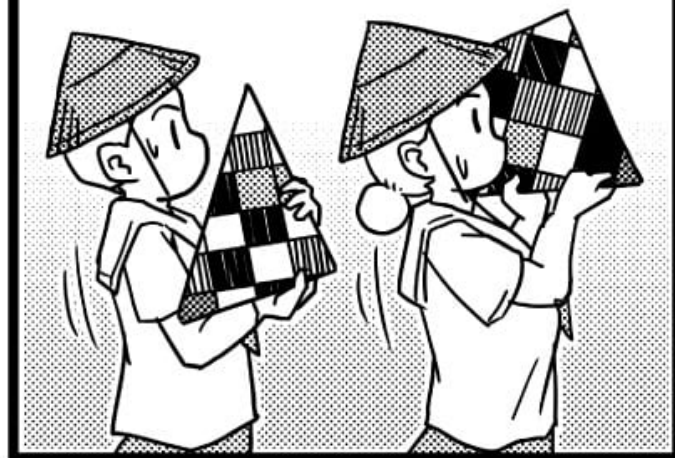
That is **video memory**.

All data that needs to be rendered is stored inside it.



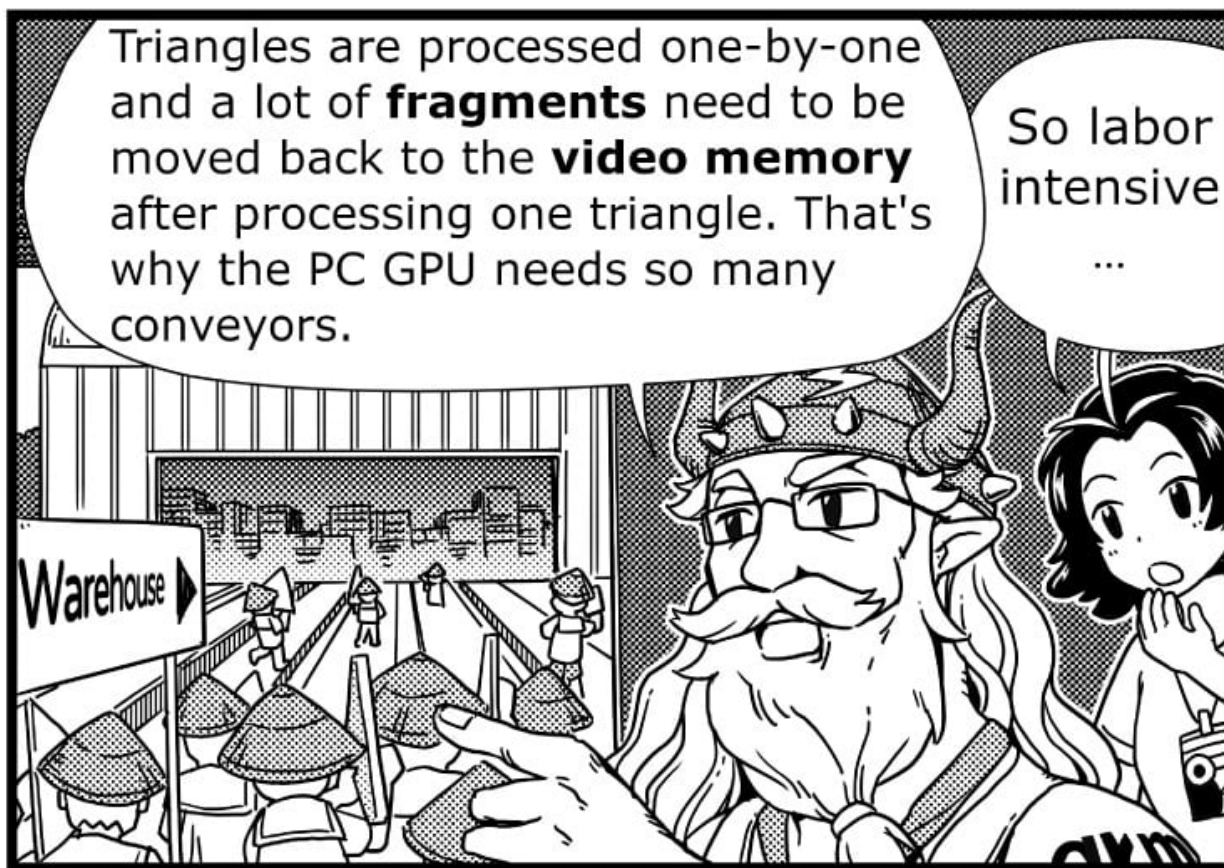


After that, the porters carry those fragments back to the video memory so the final image can be displayed on the screen.



Triangles are processed one-by-one and a lot of **fragments** need to be moved back to the **video memory** after processing one triangle. That's why the PC GPU needs so many conveyors.

So labor intensive ...



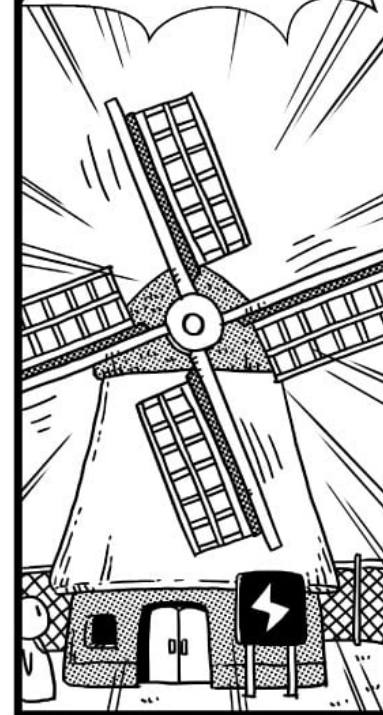
Now, let's go
to the inside of
a mobile GPU
for a
comparison.



Did you see any
difference from the
PC GPU?



**The power
station is
way too
small!!**



That's right. The power station of a PC can provide **200 ~ 300** Watts but the power station of a mobile device can only provide **2 ~ 3** Watts.

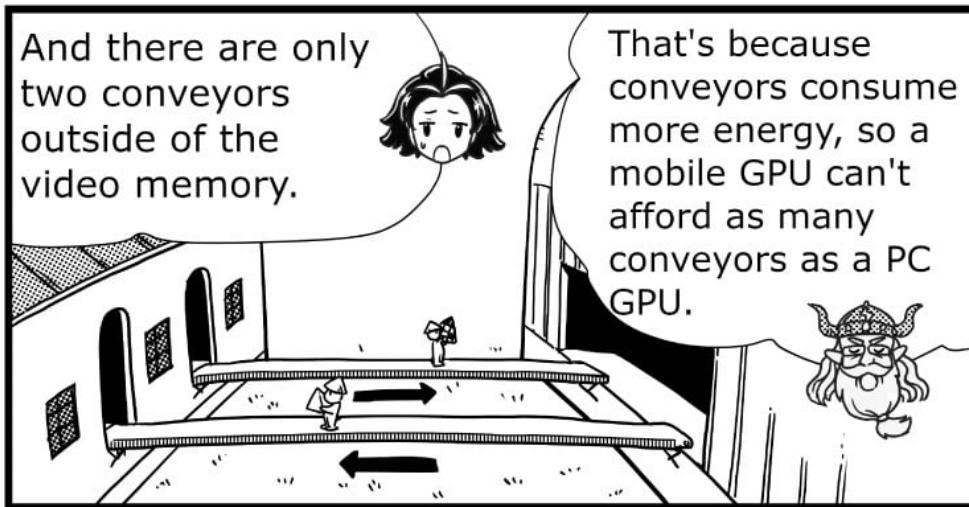
DESKTOP

MOBILE

about half a low-energy LED bulb



And there are only two conveyors outside of the video memory.



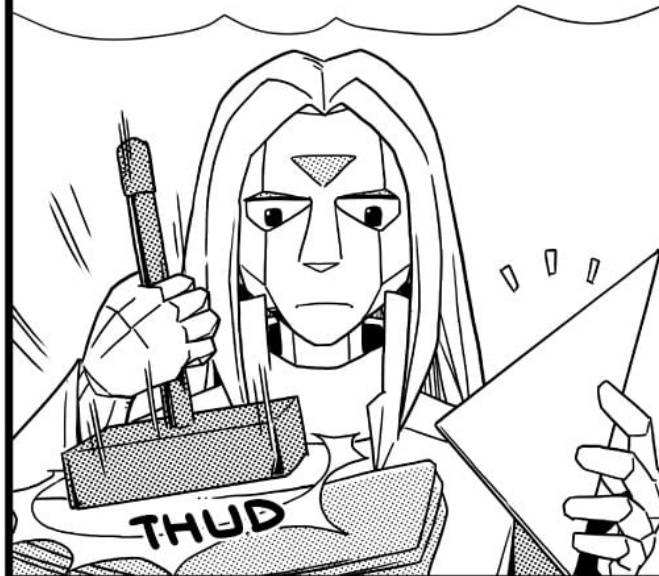
That's because conveyors consume more energy, so a mobile GPU can't afford as many conveyors as a PC GPU.

So if you port your PC game directly to mobile, the conveyors will stall almost straight away and the battery will also drain quickly.

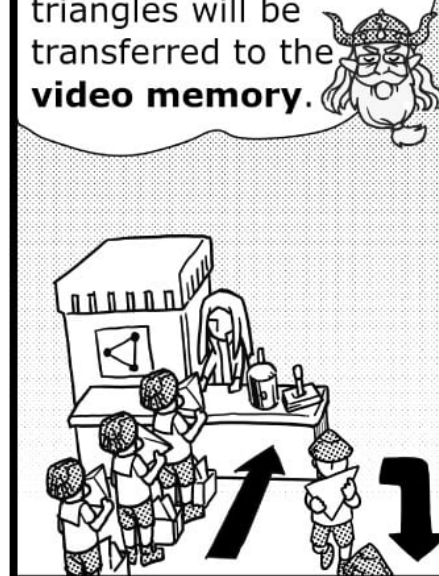
So how can a mobile GPU fix this?



Let's take a close look at the mobile GPU core. It has the **vertex shader** to process triangles, but...



The transformed triangles will not pass directly to the **fragment shader**. Instead, the triangles will be transferred to the **video memory**.



What?! So what should the **fragment shader** do then?



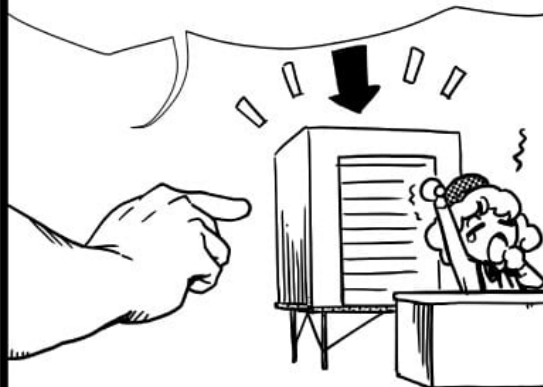
The **Fragment shader** will get the transformed triangles from the video memory then.



But would it increase the size of the data transfer?



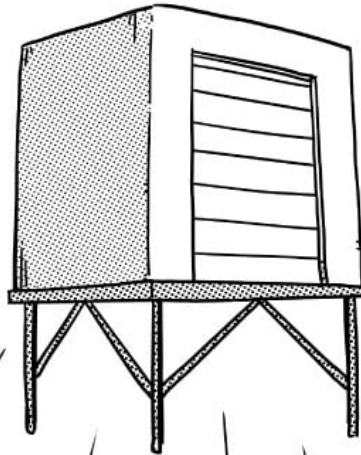
Exactly! Did you see a small warehouse besides the **fragment shader**?



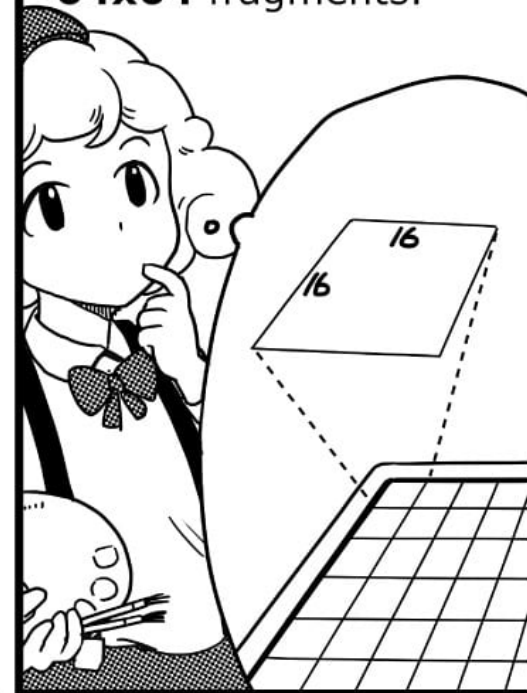
Yes, I didn't see such a thing at the PC GPU core.

It's **tile memory**, a special design for mobile GPUs.

Tile



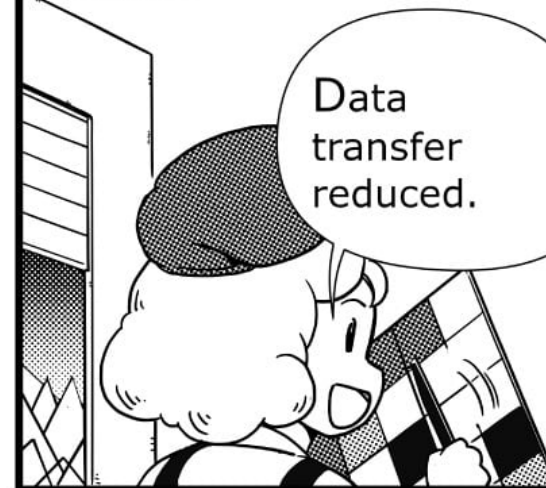
On a mobile GPU, the screen is split into many **tiles**. Each **tile** contains **16x16 ~ 64x64** fragments.



The **fragment shader** processes one **tile** at a time and then processes all triangles in the tile at once. So the frame buffer data just needs to be transferred to the **tile memory** at the beginning of the tile processing.



Then the **fragment shader** can directly access the data in the **tile memory** and process all the triangles at once.



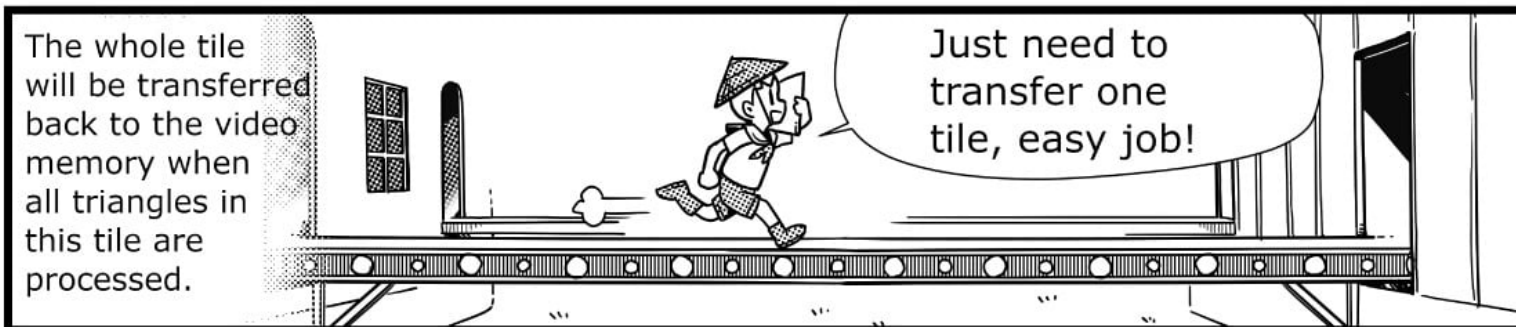
I got it now. Although the data transfer for vertex is increased, the tile design saves more of the data transfer for fragment.



If the GPU finds any triangle that is **occluded** by other triangles, those occluded triangles will be discarded without rendering.



The whole tile will be transferred back to the video memory when all triangles in this tile are processed.



Because the number of **tiles** on a screen is fixed, the size of the data transfer is also fixed.

My size is fixed



But on a PC GPU, the data transfer size is not fixed. It varies depending on the number of triangles that are rendered.

The occluded object also needs to be rendered, what a waste!



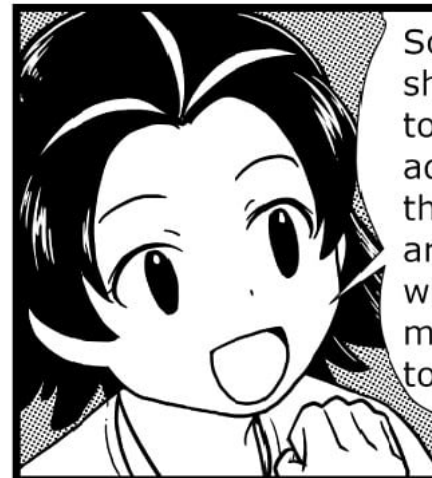
Why is the fps so low when there are not many objects on screen?



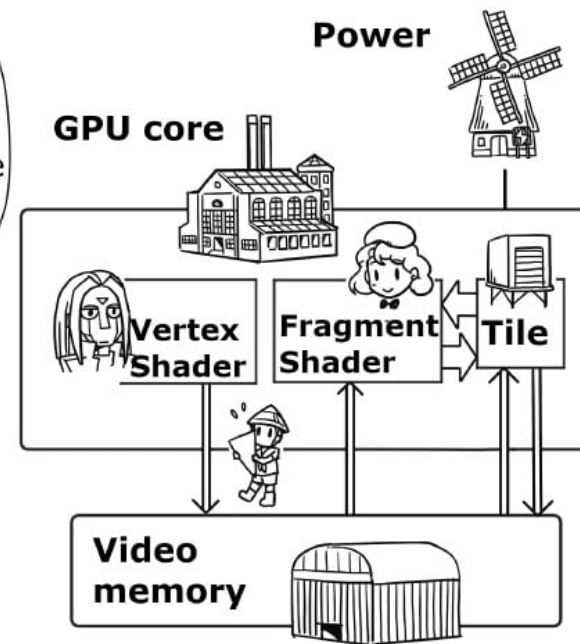
That's why the mobile GPU doesn't need so much energy.



It means the mobile GPU can run fast without consuming huge amounts of power.



So what should I do to take advantage of this architecture when porting my PC game to mobile?



Go to the Arm developer site for more detailed information :

<https://developer.arm.com/solutions/-graphics-and-gaming/developer-guides/-learn-the-basics/tile-based-rendering>



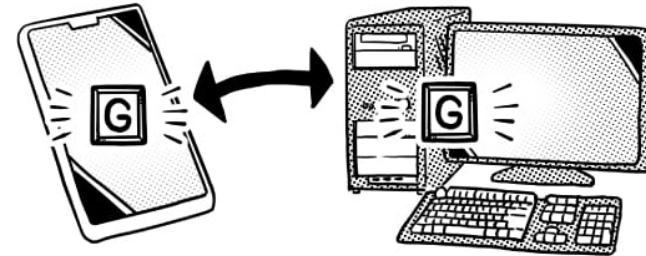
#2: Trouble at Render Pass

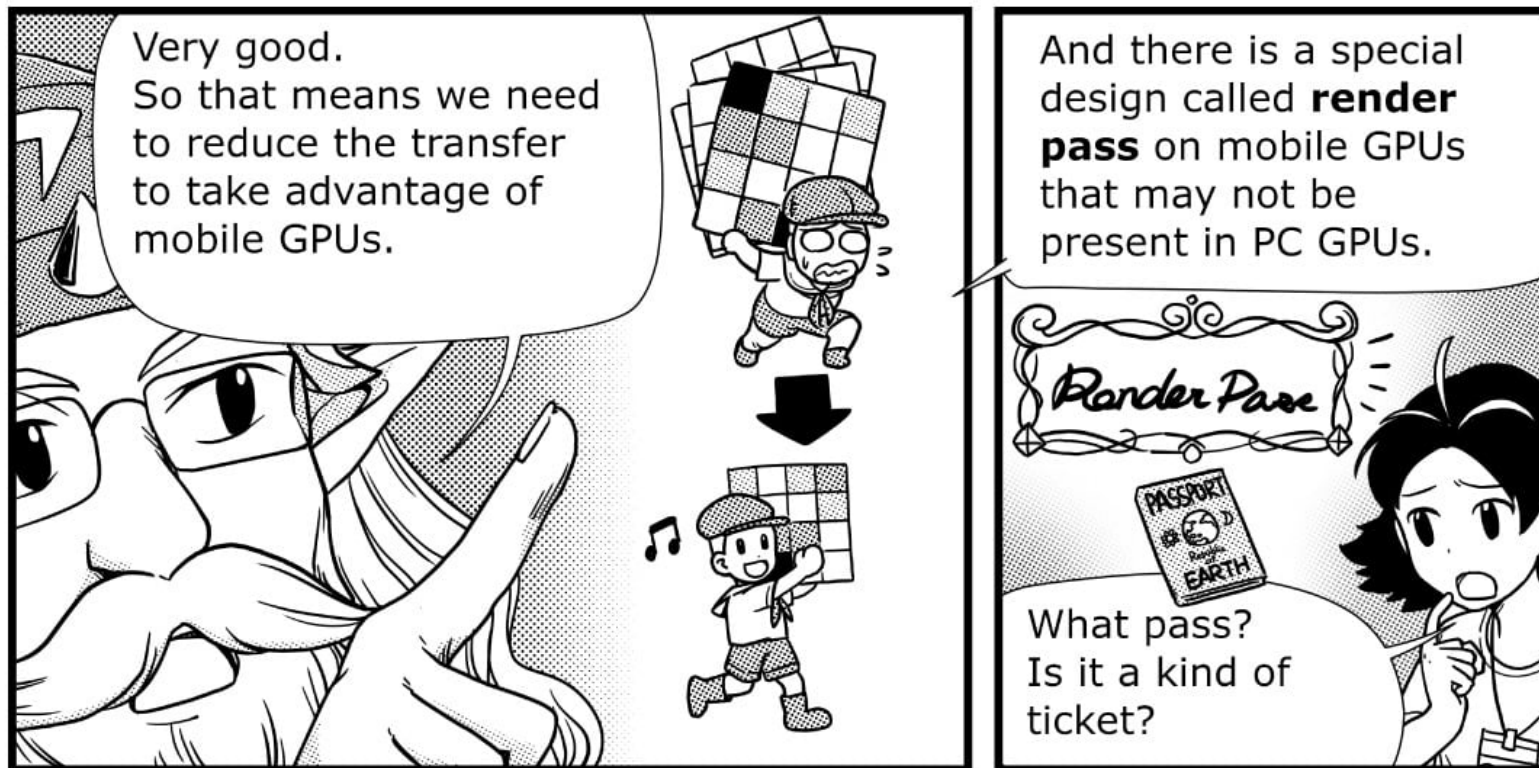
Graphics driver to the rescue!

What is a Render Pass?

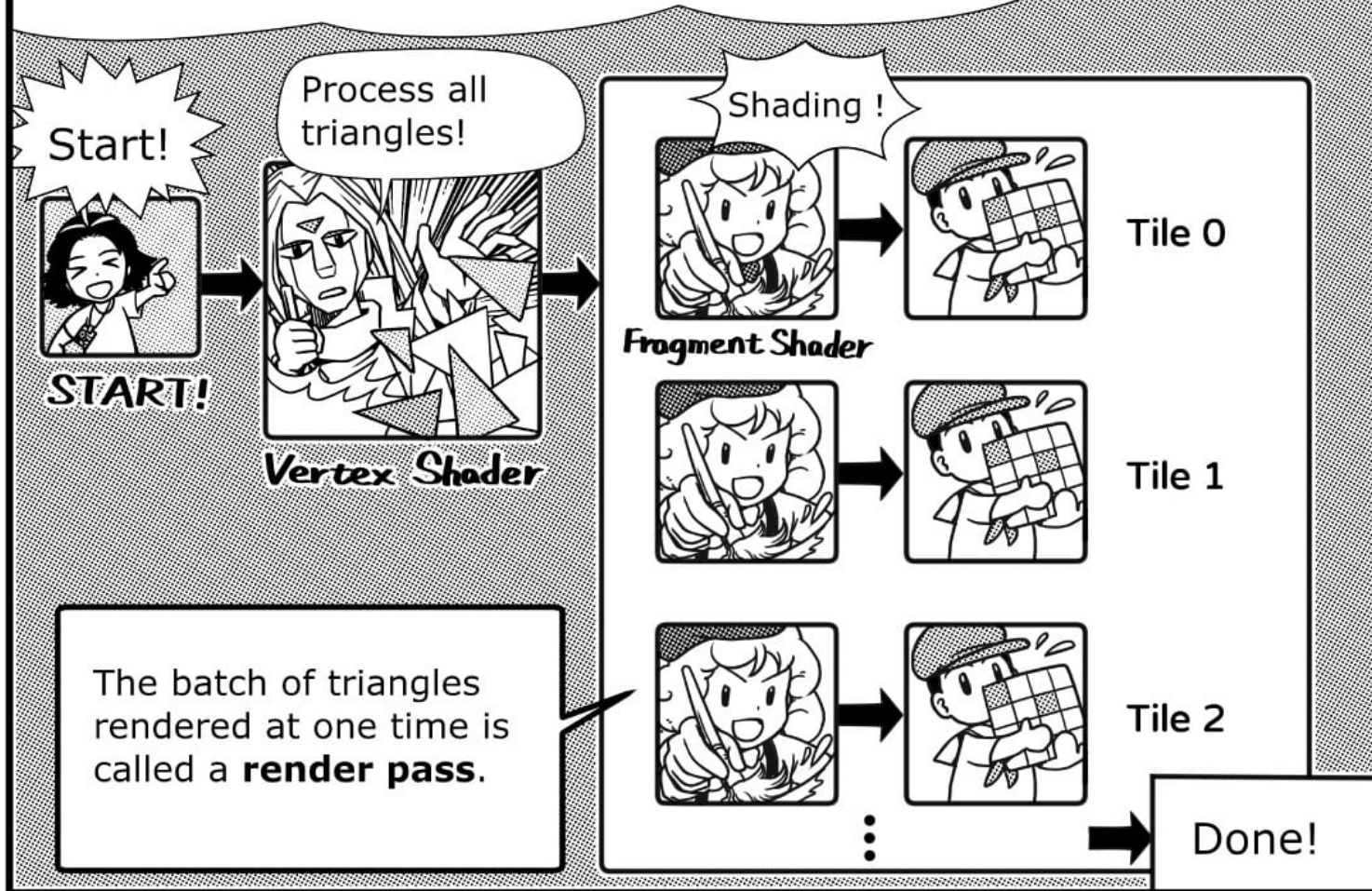
Comic: Ikaridon Yu

I have learned the difference between PC and mobile GPUs. So what should I do to take advantage of the mobile GPU architecture?





On PC GPUs, triangles are rendered one by one. But on mobile GPUs, multiple triangles are rendered at one time.



Tiles are processed one by one in a **render pass** until the frame is finished.



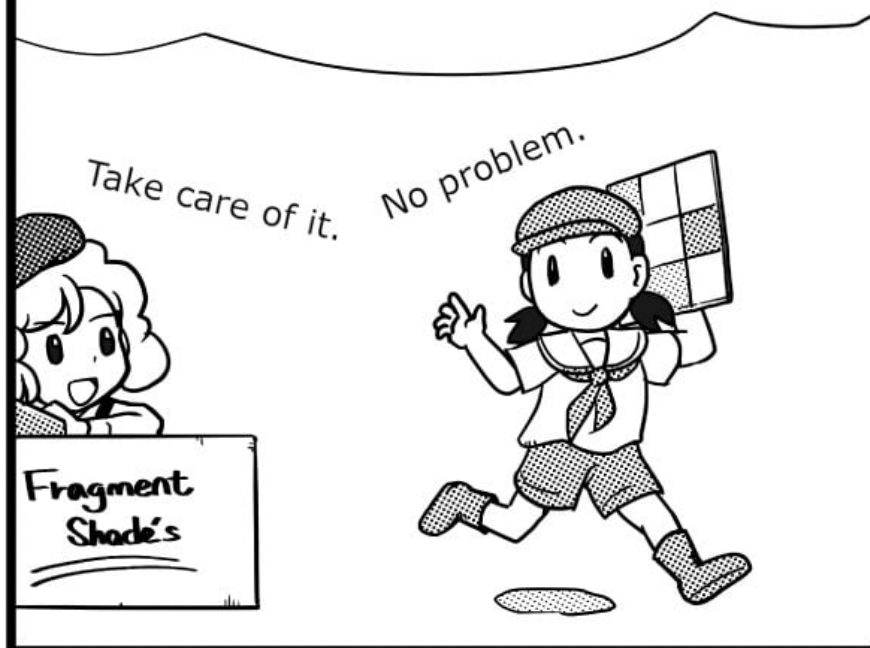
One thing that needs to be mentioned is that there are two tasks performed at the beginning and end of a **render pass**...



First, the data of the last frame needs to be moved from the video memory to tile memory at the beginning of a **render pass**.



Second, the data in the tile memory needs to be moved back to video memory at the end of a render pass. So the content of frame can be preserved.



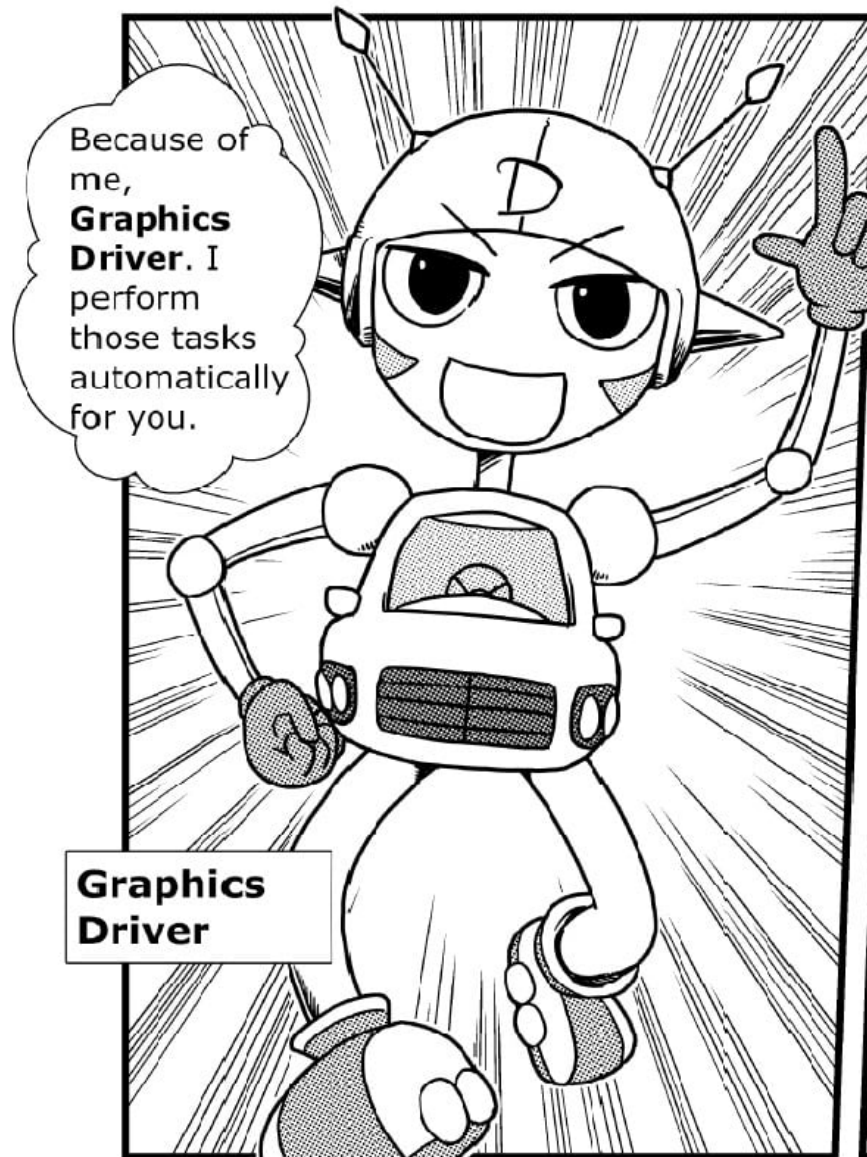
WHAT!?

I didn't do those things on a mobile GPU! Why was the game still running correctly?



Hahaha, that's because of...

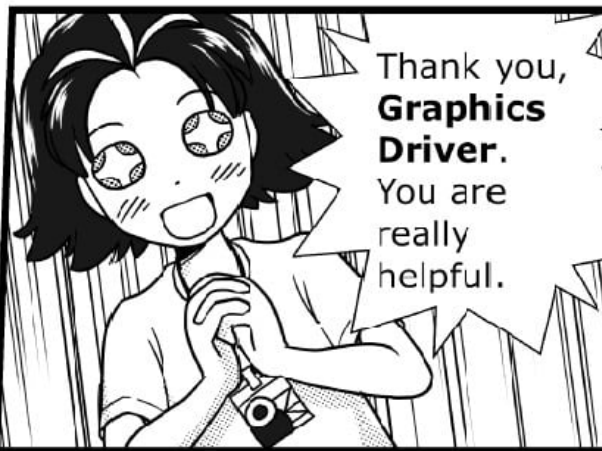




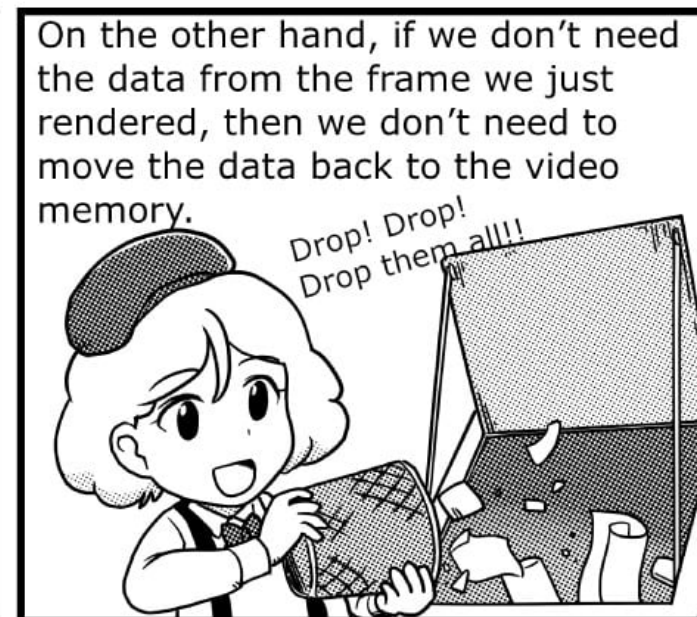
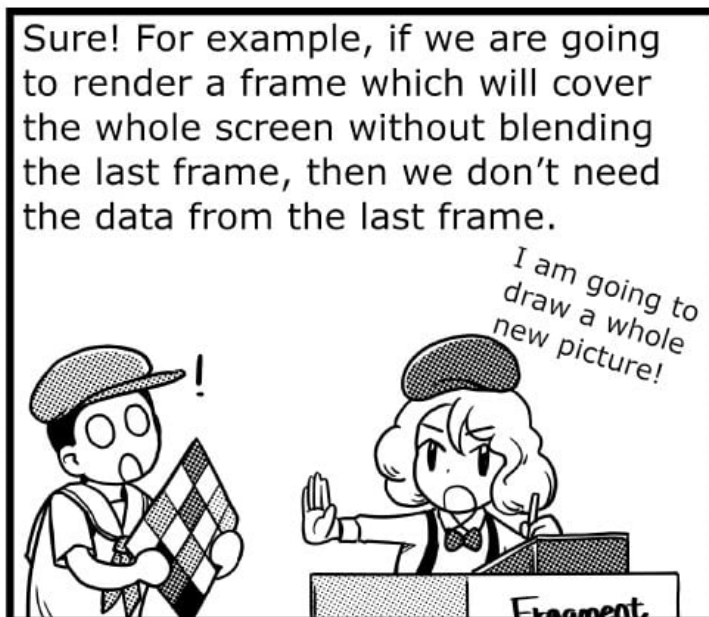
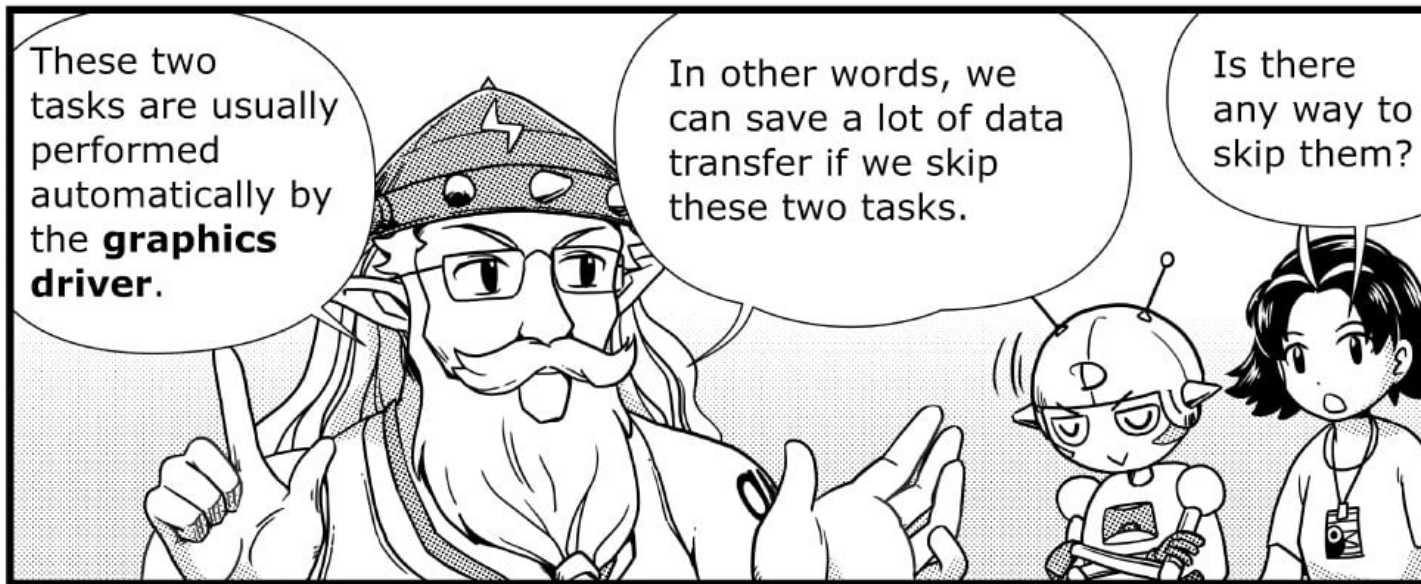
Graphics
Driver

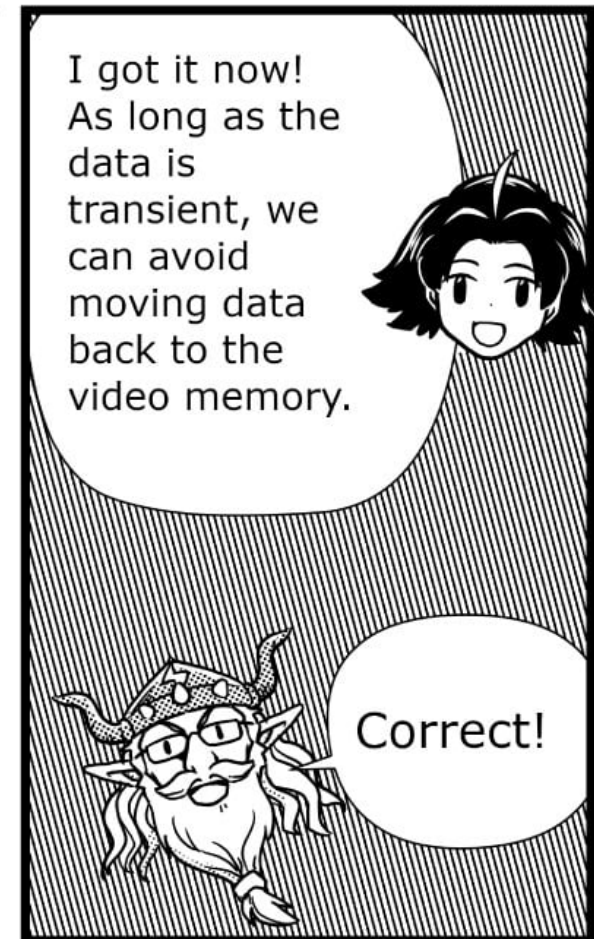
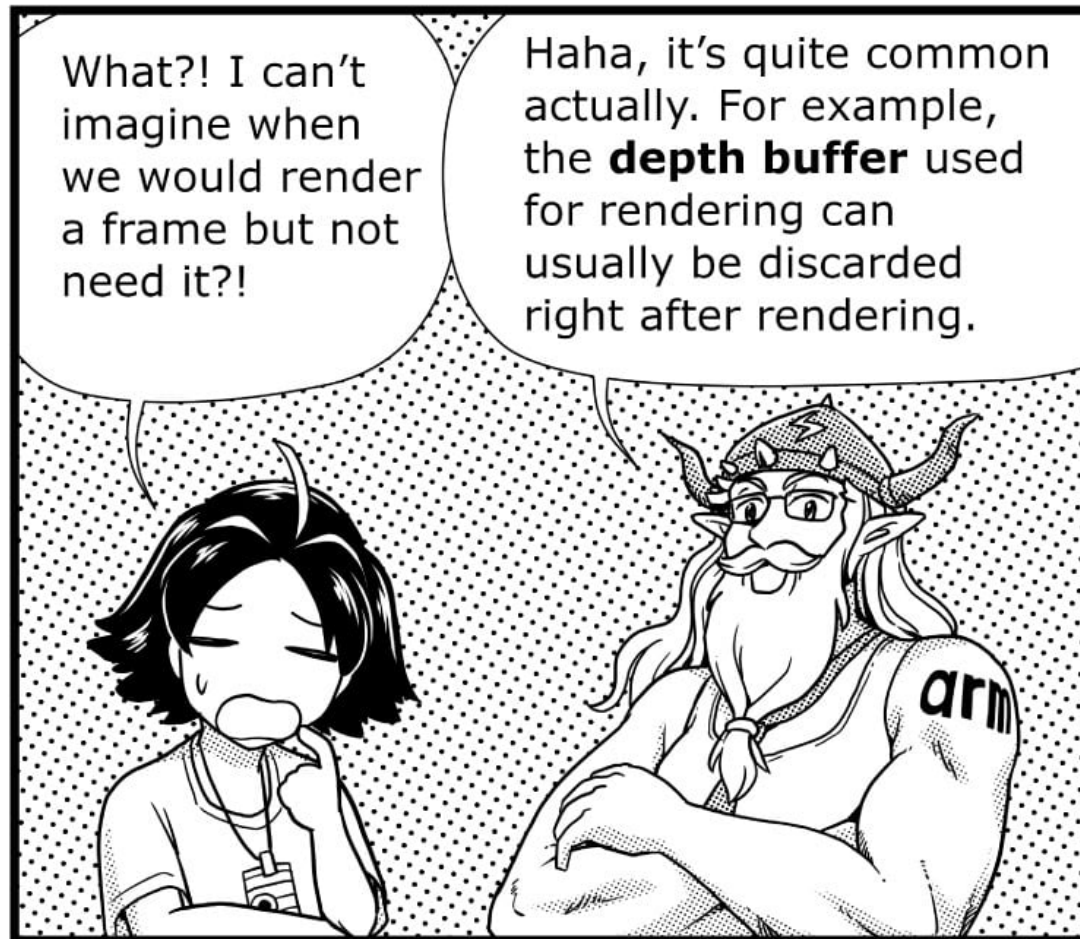
arrr

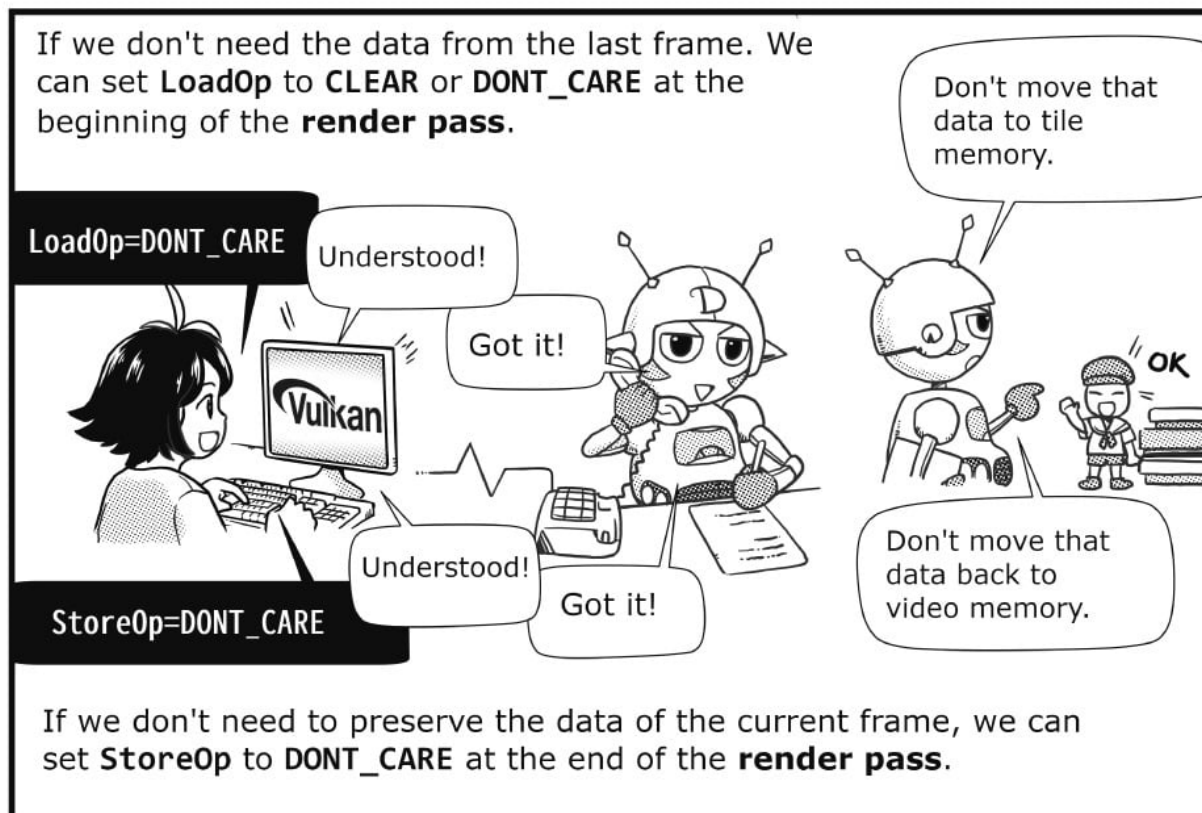
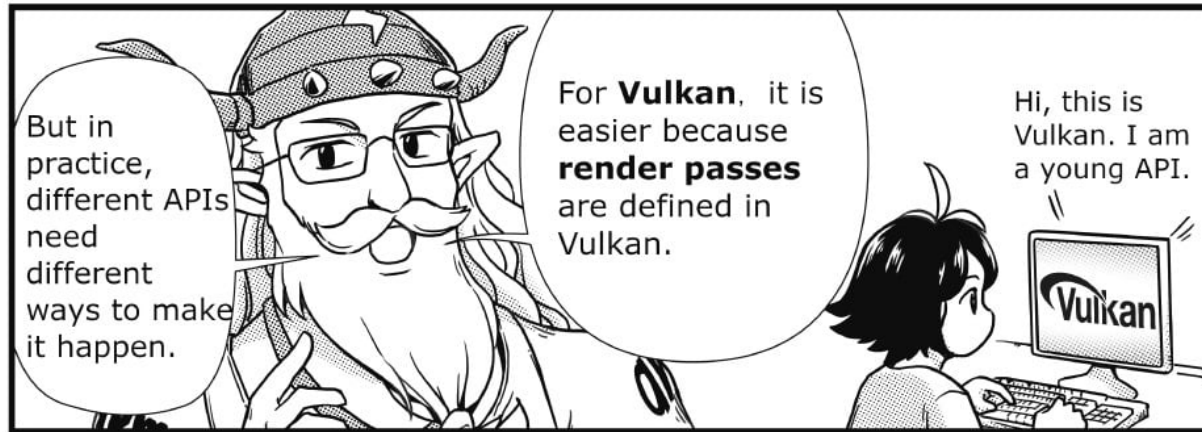
I also collect all triangles that need to be rendered to form a **render pass** for you.



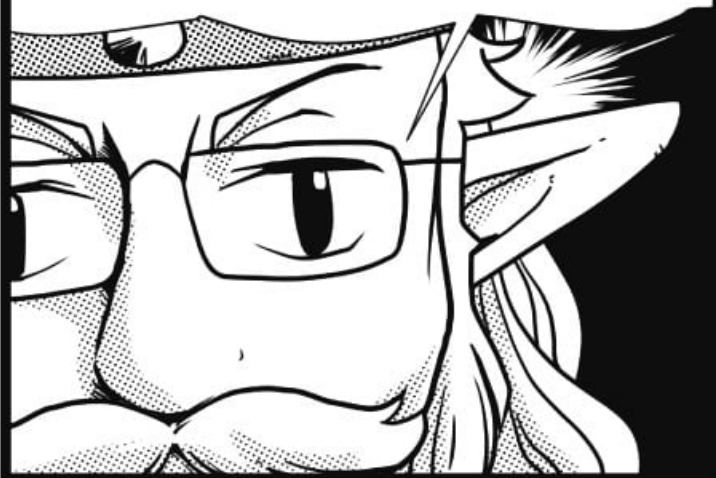
Thank you,
**Graphics
Driver.**
You are
really
helpful.







But since OpenGL ES is an old API, there is no **render pass** defined.



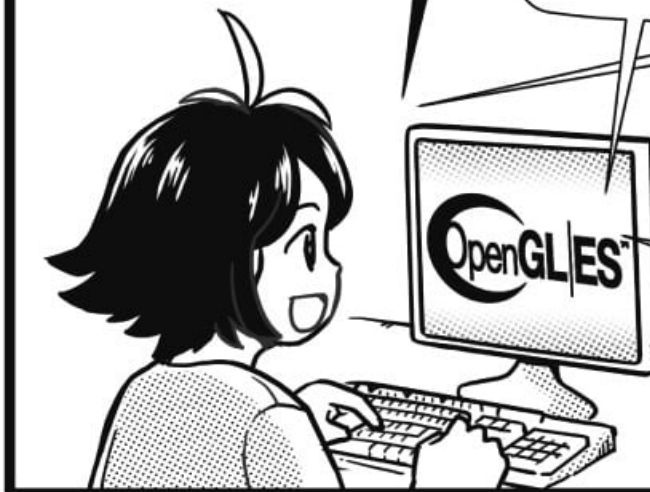
`LoadOp=DONT_CARE`

What the heck?

I don't need the data from the last frame, so...

English please...

How should I do it?



?



We can call particular APIs to notify the graphics driver.

If we call `glClear()` or `glInvalidateFramebuffer()` before rendering, then we can notify the graphics driver that we don't need the data from the last frame.

Let me talk to **Graphics Driver** directly!

`glClear()`

No problem, I will process it per your request.

What should I do when rendering is done then?

You can call `glInvalidateFramebuffer()` to notify the driver to discard the data in the tile memory.

`glInvalidateFramebuffer()`

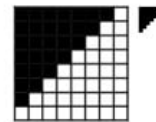
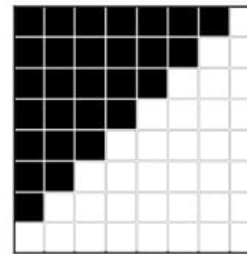
Cool! Following what you have taught me, I will now start porting my PC game to mobile.

Wait a second, there is one more useful optimization tip.

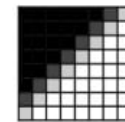
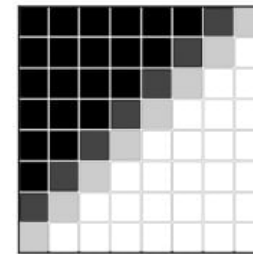
Tip?



Modern games usually use **MSAA** to reduce the anti-aliasing, but there is one big problem when using **MSAA**...



Without Antialiasing



With Antialiasing

The problem is that **MSAA** will increase the data transfer. The data transfer will become 4 times bigger when using **4X MSAA**.

GO!
GO!

4 times
bigger
power !

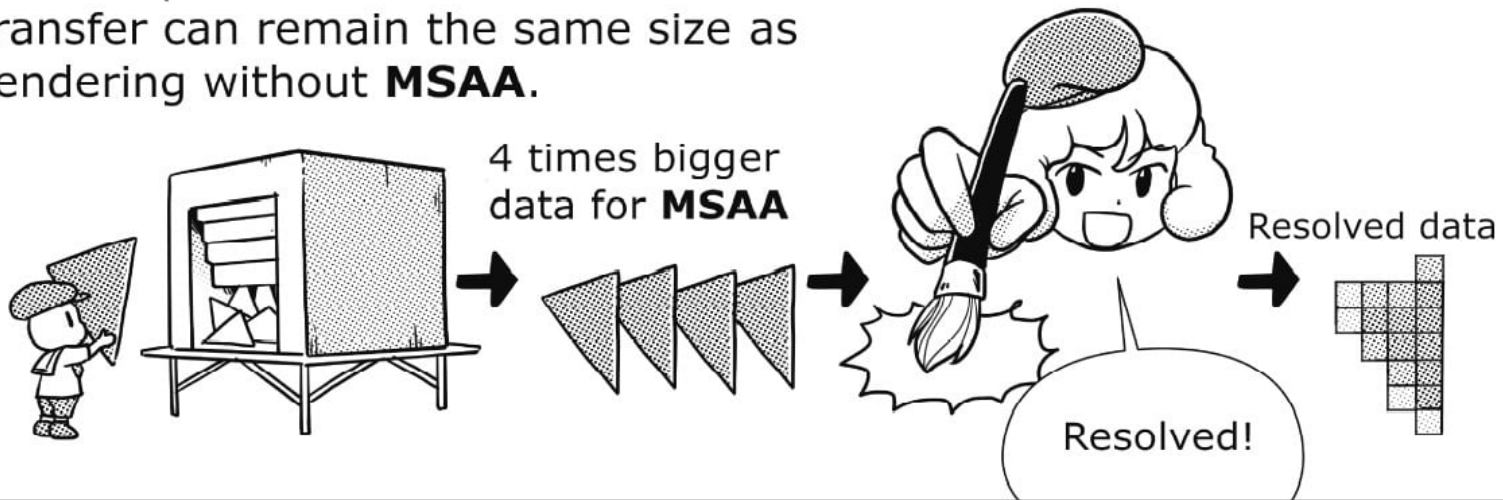
But we can take advantage of **tile memory** again!
We can put all **MSAA** data into tile memory.

Tile

Fragment
Shade's

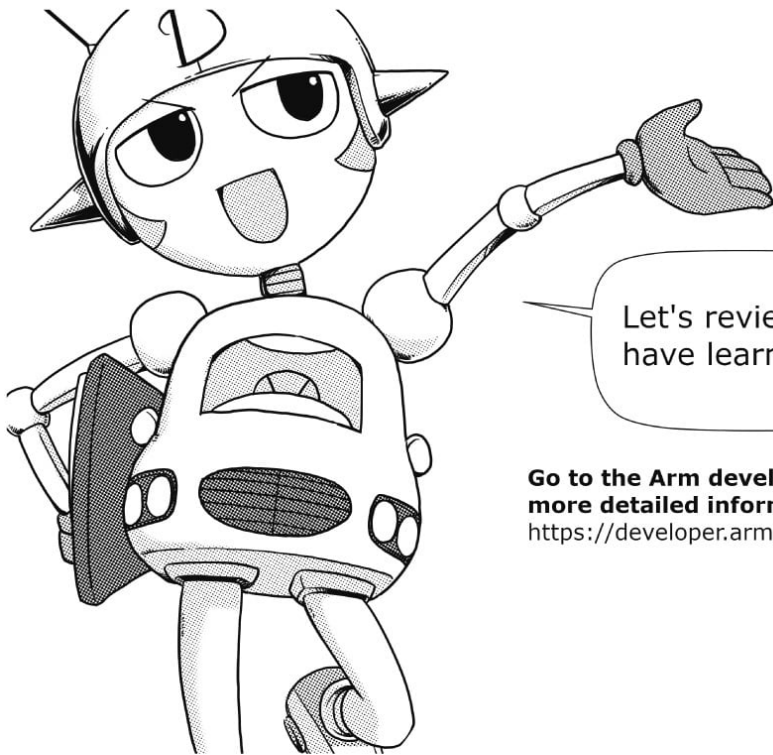
Let me
handle this.

We can resolve **MSAA** inside **tile memory** then output resolved data. So the data transfer can remain the same size as rendering without **MSAA**.



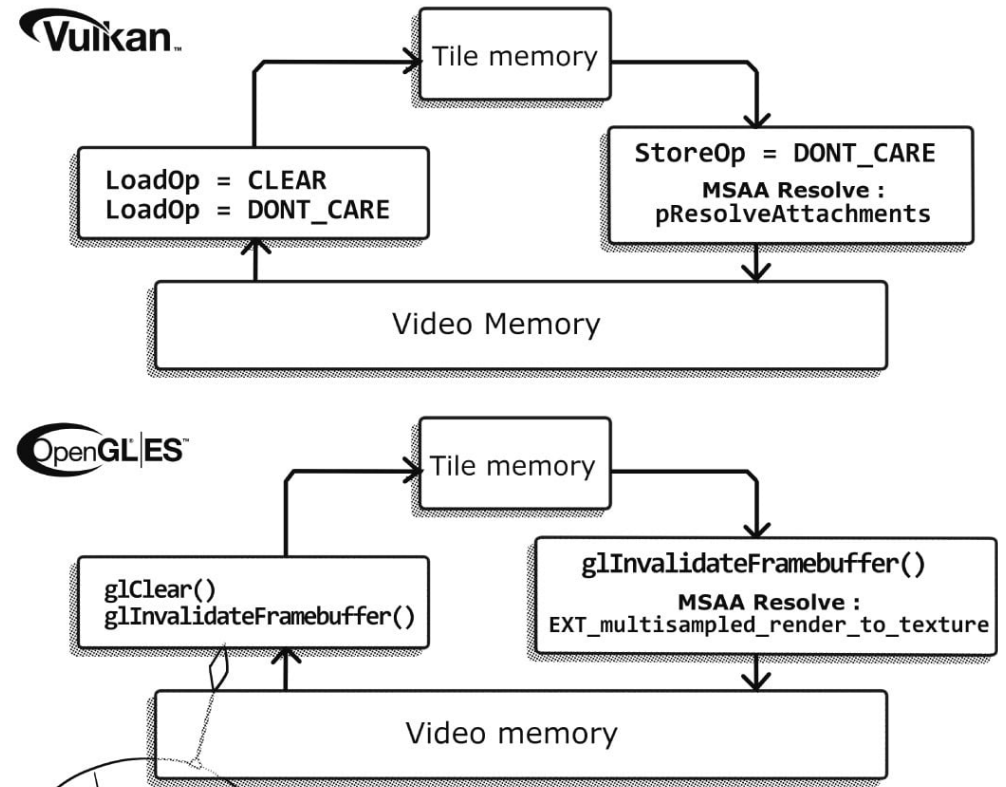
That's all for today. You can port your game to mobile now!

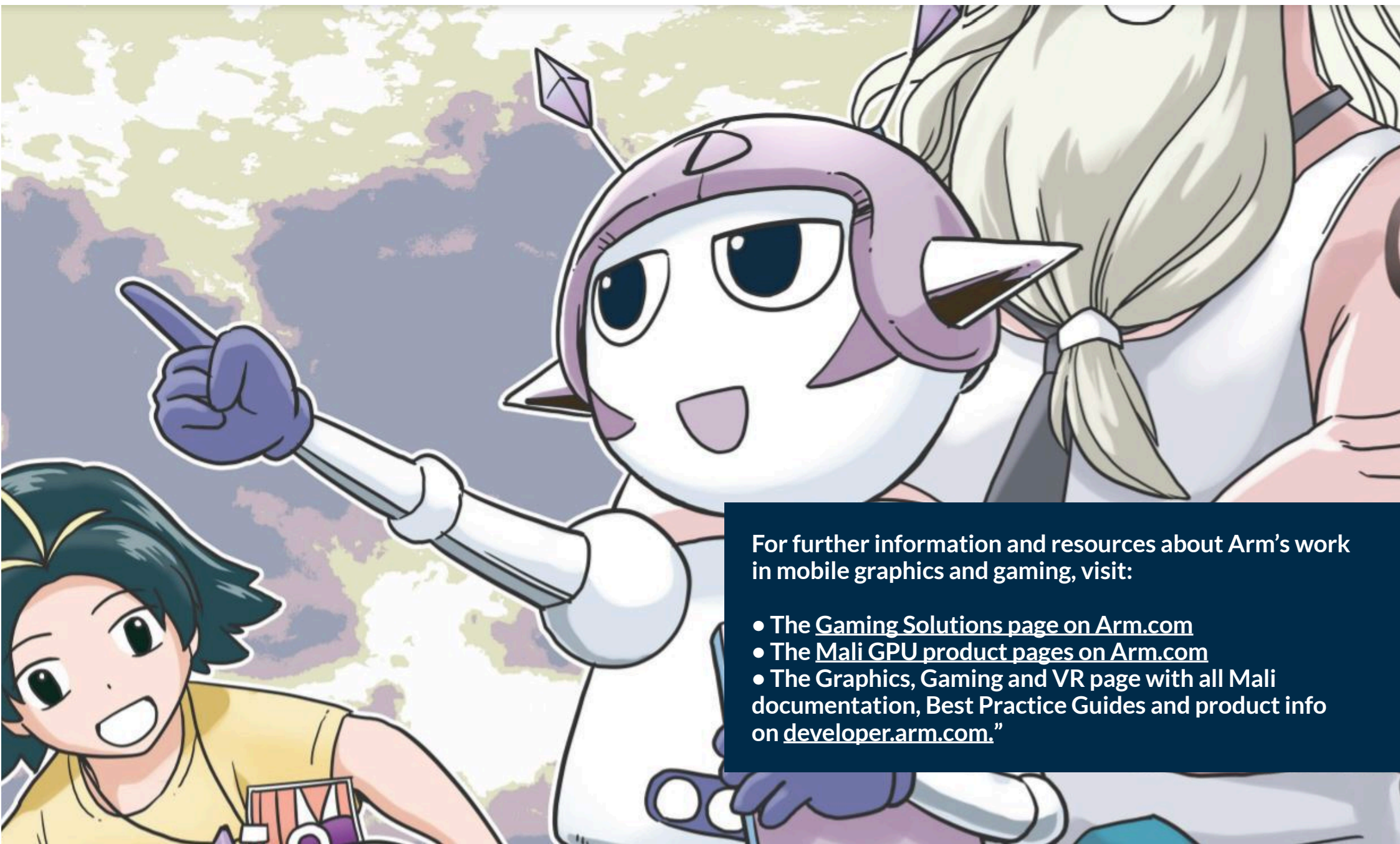




Let's review what we have learned today.

Go to the Arm developer site for more detailed information :
<https://developer.arm.com/renderpass>





For further information and resources about Arm's work in mobile graphics and gaming, visit:

- The [Gaming Solutions page on Arm.com](#)
- The [Mali GPU product pages on Arm.com](#)
- The Graphics, Gaming and VR page with all Mali documentation, Best Practice Guides and product info on [developer.arm.com](#)."

Thanks for Reading the Arm Manga Guide to the
Mali GPU

To Learn More, Please
Contact Us