OpenGL's texture DMA

DMA stands for Direct Memory Access

by Rostyslav Lesovyi

GlobalLogic

Problems

- **Async texture upload**
- **Rapid updates of texture or its parts**
- **Texture data download**

Solutions

- **Simple download/upload**
- **Shared OpenGL contexts**
- **PBO (Pixel Buffer Objects)**
- **EGLImage and GraphicBuffer**

Simple download/upload

- glTexImage2D/glTexSubImage2D for upload
- glReadPixels for download

Pros:

- very easy to use and understand

Cons:

- super slow!!!
- operates in a single thread

Shared OpenGL contexts

What to do

- create background thread
- create shared OpenGL context
- attach context to background thread
- use **glTexImage2D**, **glTexSubImage2D** or **glReadPixels**

Shared OpenGL contexts

Shared context creation:

```
EGLContext context =
egl.eglCreateContext(display, config, parentContext, null);

EGLSurface surface =
egl.eglCreatePbufferSurface(display, config, surfaceAttributes);
```

Shared OpenGL contexts

```
EGLContext context =
egl.eglCreateContext(display, config, parentContext, null);

EGLSurface surface =
egl.eglCreatePbufferSurface(display, config, surfaceAttributes);
```

Shared OpenGL contexts

Pros:

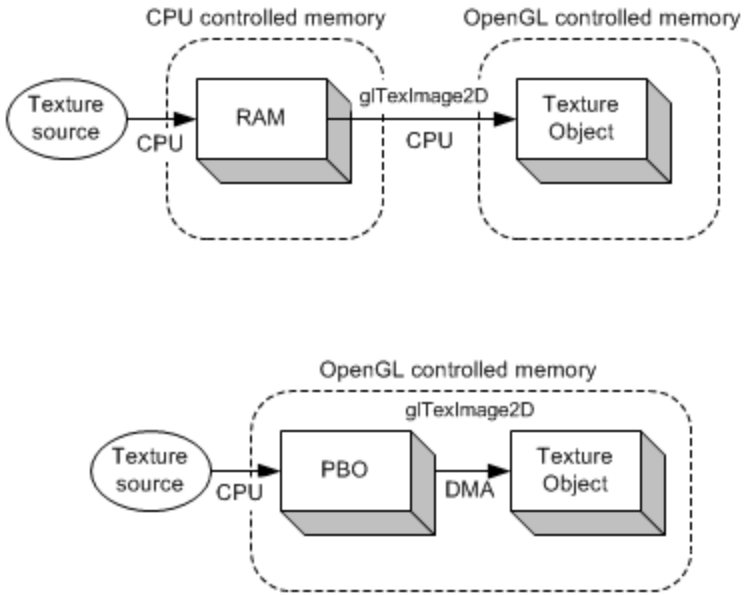
- relatively easy to use
- full threading support

Cons:

- relying on slow functions

More info:
https://github.com/pcwalton/test-async-texture-upload

Pixel Buffer Objects



Pixel Buffer Objects

Pros:

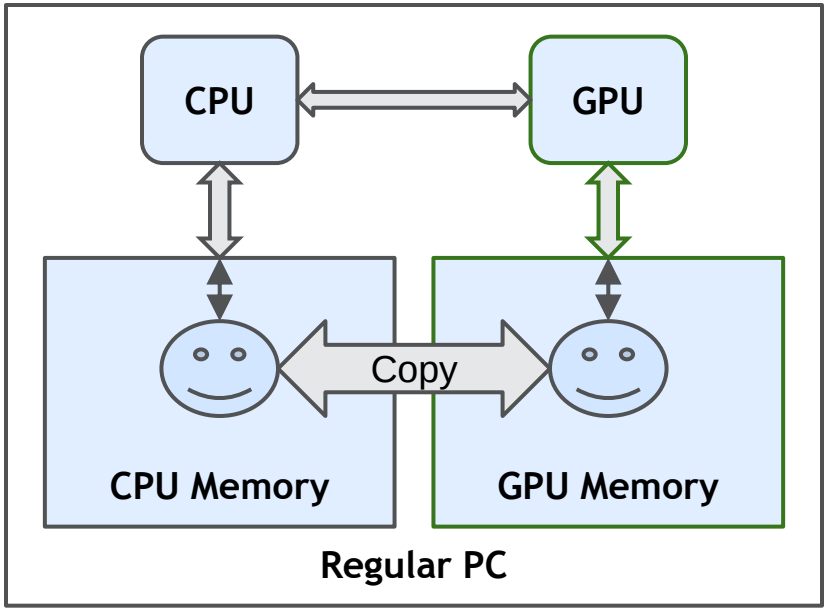
- relatively easy to use
- supports asynchronous upload/download

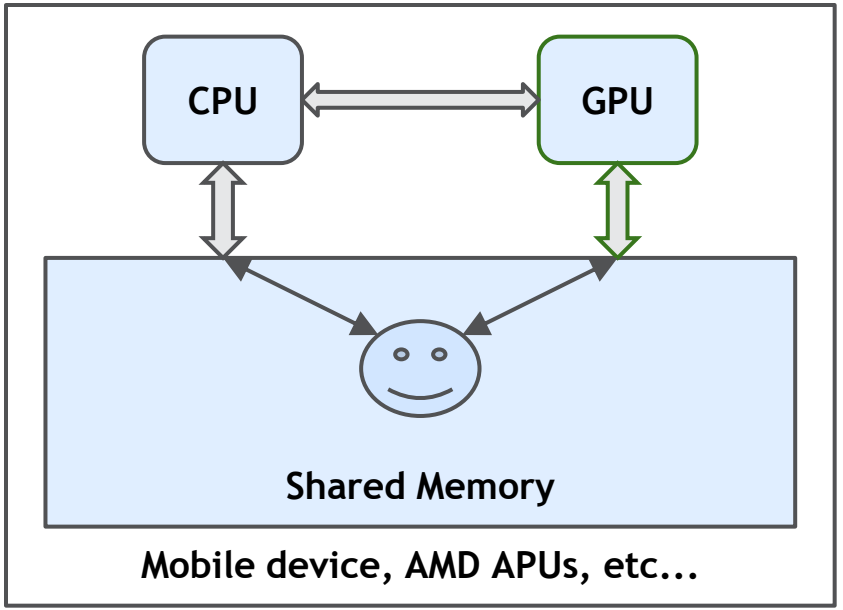
Cons:

- Android 4.3+ / OpenGL ES 3.0
- hard to use for two-ways texture updates
- still copies from PBO to CPU and vice versa

More info:
http://www.songho.ca/opengl/gl_pbo.html

EGLImage and GraphicBuffer





 = Texture image data (actual pixels)

EGLImage and GraphicBuffer

Pros:

- direct access to texture's pixel data
- can be accessed by any thread
- supported by Android 2.0.1+

Cons:

- requires NDK
- requires usage of ARM library

More info:
https://wiki.mozilla.org/Platform/GFX/Gralloc

Q&A

Thank you