OpenGL ES

Pyglet has experimental support for OpenGL ES 3.2, with some limitations. Devices like Raspberry Pi 4 and 5 supports OpenGL ES 3.1 with extensions covering most of the missing 3.2 features (with the exception of tessellation shaders). There are likely other devices with ES support that can run Pyglet.

Creating a window / context

In order to successfully create a context and window with OpenGL ES we first need to:

- Likely disable the shadow window because very few environments supports this
- Pass in a custom config at window creation specifying OpenGL ES and version

Example:

```
import pyglet

# Disable shadow window
pyglet.options.shadow_window = False

# Query the best config for the screen
display = pyglet.display.get_display()
screen = display.get_default_screen()
config = screen.get_best_config()

# Specify that we want to use OpenGL ES 3.1
# This is very specific to the Raspberry Pi 4 and 5. Use 3.2 if you can.
config.opengl_api = "gles"
config.major_version = 3
config.minor_version = 1

# Create the window
window = pyglet.window.Window(config=config)
```

Limitations

Multisampling

Be careful with enabling multisampling on the window. Likely `get_best_config()` will not have multisampling enabled.

Textures

Pixel format for textures and images are limited to `RGBA`. If you try to load any other pixel format through pyglet's image loading functions you will get an error when they are turned into OpenGL textures later.

The reason for this limitation is that OpenGL ES does not support pixel format conversion during pixel transfer, meaning when calling `glTexImage` to upload pixel data to the GPU, the pixel format must match the internalformat. In desktop OpenGL an RGB image will be automatically converted to RGBA during this transfer. This is not supported in OpenGL ES.

You are, however, free to create your own textures with any pixel format using the gl bindings directly.

Compute Shader

If you are planning to bind textures to image units with the intention of using them in a compute shader, you need to create these textures yourself using the gl bindings and allocate space using `glTexStorage`. Pyglet is using `glTexImage` to allocate space and upload the pixel data for textures.

The difference here is that `glTexStorage` creates immutable storage. You can only call it once and then fill the allocated space with pixel data using `glTexSubImage`. `glTexImage` on the other hand can be called multiple times reallocating the texture storage each time. Likely OpenGL ES doesn't want to deal with the extra complexity of validating the texture storage of images every time a compute shader is dispatched.

In short: Create your own textures with immutable storage if you are planning to use them in a compute shader.

Shaders

Pyglet's shader system supports basic conversion between GLSL 1.5/3.3 shaders to GLES 3.2 shaders when running in OpenGL ES mode. This is to ensure that pyglet's built-in shaders also works with GL ES. This system is not perfect and are likely to have some flaws.

Currently the shaders are converted to ES 3.1 shaders as a careful approach manually enabling extensions needed for ES 3.2. Precisions qualifiers are injected using `mediump` by default.