How to develop OpenGL ES (GLES) 2.0 applications on Linux?

I would like to develop OpenGL ES 2.0 apps on my Ubuntu machine. I could not find any libraries/emulators that support GLES 2.0 yet. Any suggestions?

linux ubuntu opengl-es

edited Sep 6 '16 at 20:04
genpfault
37.8k 8 41 83

asked Sep 28 '10 at 2:15
l.thee.a
1,098 3 16 25

5 Answers

You can use POWERVR SDK to emulate Opengl es on your PC. You can download the SDK here. The archive provides the necessary steps to install the emulation libraries as a documentation file and includes tutorials and demo applications with source codes.

answered Dec 16 '10 at 16:46
Bertan Gundogdu
140 1 6

2 This requires a registration. The ARM Mali version is an open download: malideveloper.com/opengl-es-20-emulator.php – ahcox Mar 15 '12 at 21:56

1 @ahcox Not to mention it's cca 300kb compared to 300mb+, includes both ES1 and ES2 libs, and the hosting seems to support larger speeds than 30kbps. I'd really love to get the POWERVR SDK (I suppose I would get PVRTC support), but this is silly. Admittedly, Imagination Technologies may be only temporarily experiencing difficulties, yet, had you, ahcox, not posted this, I'd really give up. You should have posted this as an alternative answer! :-) – Ivan Vučica Jun 18 '12 at 17:20

Mesa supports it. If you want to restrict yourself to OpenGL ES *only* then you'll need to build it into a separate directory and then add the appropriate include and library directories.

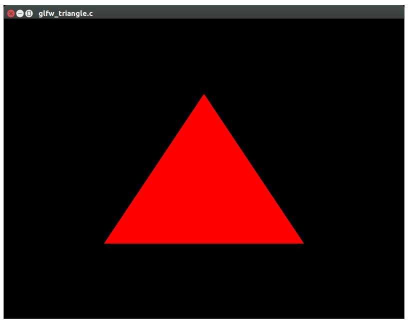
answered Nov 29 '10 at 9:20
Ignacio Vazquez-Abrams
490k 75 887 1024

GLFW, Mesa, Ubuntu 16.04 AMD64

This was not easy to setup on Ubuntu 14.04, but now it just works.

```
sudo apt-get install libglfw3-dev libgles2-mesa-dev
gcc glfw_triangle.c -lGLESv2 -lglfw
```

Output:



Source:

```
#include <stdio.h>
#include <stdlib.h>

#define GLFW_INCLUDE_ES2
#include <GLFW/glfw3.h>

static const GLuint WIDTH = 800;
static const GLuint HEIGHT = 600;
static const GLchar* vertex_shader_source =
    "#version 100\n"
    "attribute vec3 position;\n"
    "void main() {\n"
    "   gl_Position = vec4(position, 1.0);\n"
    "}\n";
static const GLchar* fragment_shader_source =
    "#version 100\n"
    "void main() {\n"
    "   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n"
    "}\n";
static const GLfloat vertices[] = {
     0.0f,  0.5f, 0.0f,
     0.5f, -0.5f, 0.0f,
    -0.5f, -0.5f, 0.0f,
};

GLint common_get_shader_program(const char *vertex_shader_source, const char
*fragment_shader_source) {
    enum Consts {INFOLOG_LEN = 512};
    GLchar infoLog[INFOLOG_LEN];
    GLint fragment_shader;
    GLint shader_program;
    GLint success;
    GLint vertex_shader;

    /* Vertex shader */
    vertex_shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertex_shader, 1, &vertex_shader_source, NULL);
    glCompileShader(vertex_shader);
    glGetShaderiv(vertex_shader, GL_COMPILE_STATUS, &success);
    if (!success) {
        glGetShaderInfoLog(vertex_shader, INFOLOG_LEN, NULL, infoLog);
        printf("ERROR::SHADER::VERTEX::COMPILATION_FAILED\n%s\n", infoLog);
    }

    /* Fragment shader */
    fragment_shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragment_shader, 1, &fragment_shader_source, NULL);
    glCompileShader(fragment_shader);
    glGetShaderiv(fragment_shader, GL_COMPILE_STATUS, &success);
    if (!success) {
        glGetShaderInfoLog(fragment_shader, INFOLOG_LEN, NULL, infoLog);
        printf("ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n%s\n", infoLog);
    }

    /* Link shaders */
    shader_program = glCreateProgram();
    glAttachShader(shader_program, vertex_shader);
    glAttachShader(shader_program, fragment_shader);
    glLinkProgram(shader_program);
    glGetProgramiv(shader_program, GL_LINK_STATUS, &success);
    if (!success) {
        glGetProgramInfoLog(shader_program, INFOLOG_LEN, NULL, infoLog);
        printf("ERROR::SHADER::PROGRAM::LINKING_FAILED\n%s\n", infoLog);
    }

    glDeleteShader(vertex_shader);
    glDeleteShader(fragment_shader);
    return shader_program;
}

int main(void) {
    GLuint shader_program, vbo;
    GLint pos;
    GLFWwindow* window;

    glfwInit();
    glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 0);
    window = glfwCreateWindow(WIDTH, HEIGHT, __FILE__, NULL, NULL);
    glfwMakeContextCurrent(window);

    printf("GL_VERSION  : %s\n", glGetString(GL_VERSION) );
    printf("GL_RENDERER : %s\n", glGetString(GL_RENDERER) );
    shader_program = common_get_shader_program(vertex_shader_source,
fragment_shader_source);
    pos = glGetAttribLocation(shader_program, "position");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glViewport(0, 0, WIDTH, HEIGHT);

    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
    glVertexAttribPointer(pos, 3, GL_FLOAT, GL_FALSE, 0, (GLvoid*)0);
    glEnableVertexAttribArray(pos);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    while (!glfwWindowShouldClose(window)) {
        glfwPollEvents();
        glClear(GL_COLOR_BUFFER_BIT);
        glUseProgram(shader_program);
        glDrawArrays(GL_TRIANGLES, 0, 3);
        glfwSwapBuffers(window);
    }
    glDeleteBuffers(1, &vbo);
    glfwTerminate();
    return EXIT_SUCCESS;
}
```

The key line lines of code are:

```
#define GLFW_INCLUDE_ES2
#include <GLFW/glfw3.h>
```

`GLFW_INCLUDE_ES2` is documented at: http://www.glfw.org/docs/latest/build_guide.html#build_macros and a quick look at the source shows that it forwards to GLES:

```
 #elif defined(GLFW_INCLUDE_ES2)
  #include <GLES2/gl2.h>
  #if defined(GLFW_INCLUDE_GLEXT)
   #include <GLES2/gl2ext.h>
  #endif
```

This source seems to be written in the common subset of GLES and OpenGL (like much of GLES), and also compiles with `-lGL` if we remove the `#define GLFW_INCLUDE_ES2`.

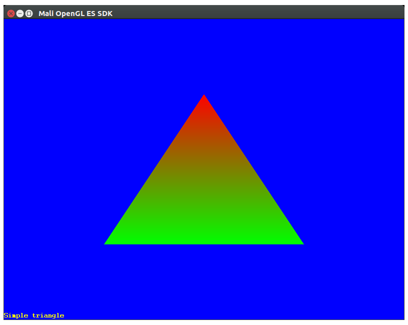
If we add things which are not in GLES like immediate rendering `glBegin`, link fails as expected.

See also: https://askubuntu.com/questions/244133/how-do-i-get-egl-and-opengles-libraries-for-ubuntu-running-on-virtualbox

Credits: genpfult made the code much more correct.

ARM Mali OpenGL ES SDK

- download from: http://malideveloper.arm.com/resources/sdks/opengl-es-sdk-for-linux/
- open the documentation HTML on a browser
- follow the "Quick Start Guide", it's simple



Contains several interesting open source examples + windowing system boilerplate (X11 + EGL).

The build system supports easy cross compilation for ARM / Mali SoCs, but I haven't tested that yet.

The key component included seems to be the "OpenGL ES Emulator" http://malideveloper.arm.com/resources/tools/opengl-es-emulator/ which "maps OpenGL ES 3.2 API calls to the OpenGL API". But that does not ship with source, only precompiled.

Uses a custom enterprisey EULA that appears to be permissive, but yeah, ask your lawyer.

Tested on SDK v2.4.4.

edited May 23 at 12:32
Community♦
1 1

answered Sep 6 '16 at 19:13
Ciro Santilli 709大抓捕六四事件 法轮功
80.1k 14 321 245

`#version 330 core` will probably choke non-Mesa ES 2.0 implementations. Should probably drop back to `#version 100` & `attrib` / `varying` instead of `in` / `out` / `layout`. Don't think VAOs are in un-extended ES 2.0 either. – genpfault Sep 6 '16 at 20:11

@genpfault if you can, post the GLES 2 compliant version on a Gist or just edit it in or link to a correct example, and I will test it on my Ubuntu again and update. I obviously know little about shaders yet, but needed a GLES hello world for platform integration testing. – Ciro Santilli 709大抓捕 六四事件 法轮功 Sep 6 '16 at 21:27

Edited, works on my Debian `testing` laptop with Skylake graphics & Mesa 11.2.2. – genpfault Sep 7 '16 at 5:30

1 @genpfault awesome! Continues to work perfectly for me. – Ciro Santilli 709大抓捕 六四事件 法轮功 Sep 7 '16 at 7:04

Develop to the OpenGL 2.0 standard, and don't use immediate mode or fixed function mode. Essentially your program will be ES 2.0 compliant.

answered Sep 28 '10 at 2:30
Justice
8,061 11 51 85

4 This is my worst case solution. Unfortunately I am not experienced enough to tell when I am stepping into the (non es) gl territory. Which is why I am looking for hard boundaries. – l.thee.a Sep 28 '10 at 23:03

I can only discourage from this solution. My advice is to create at least an openglES context (glfw or other tools might help you) or you can include an opengl ES header that does not define functions that are not used at all. – Arne Feb 15 '14 at 18:07

---

you can generate a header that has only those functions that you really need. And with glfw you can create an opengl es context. So you can't accidently use functions that you do not want to use, because they won't be defined in this way. I found this that might help you here.

gl load from the unofficial opengl sdk

answered Feb 15 '14 at 14:40
Arne
3,250 2 24 45