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

    #include <X11/Xlib.h>

    /*
        If you get linking errors when using C++, you need
        to add extern "C" here or in X11-xcb.h, unless
        this bug is already fixed in your version:
        http://bugs.freedesktop.org/show_bug.cgi?id=22252
    */
    #include <X11/Xlib-xcb.h> /* for XGetXCBConnection, link with libX11-xcb */

    #include <xcb/xcb.h>

    #include <GL/glx.h>
    #include <GL/gl.h>

    /*
        Attribs filter the list of FBConfigs returned by glXChooseFBConfig().
        Visual attribs further described in glXGetFBConfigAttrib(3)
    */
    static int visual_attribs[] =
    {
        GLX_X_RENDERABLE, True,
        GLX_DRAWABLE_TYPE, GLX_WINDOW_BIT,
        GLX_RENDER_TYPE, GLX_RGBA_BIT,
        GLX_X_VISUAL_TYPE, GLX_TRUE_COLOR,
        GLX_RED_SIZE, 8,
        GLX_GREEN_SIZE, 8,
        GLX_BLUE_SIZE, 8,
        GLX_ALPHA_SIZE, 8,
        GLX_DEPTH_SIZE, 24,
        GLX_STENCIL_SIZE, 8,
        GLX_DOUBLEBUFFER, True,
        //GLX_SAMPLE_BUFFERS  , 1,
        //GLX_SAMPLES         , 4,
        None
    };

    void draw()
    {
        glClearColor(0.2, 0.4, 0.9, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
    }

    int main_loop(Display *display, xcb_connection_t *connection, xcb_window_t window, GLXDraw
    {
        int running = 1;
        while(running)
        {
            /* Wait for event */
            xcb_generic_event_t *event = xcb_wait_for_event(connection);
            if(!event)
            {
                fprintf(stderr, "i/o error in xcb_wait_for_event");
                return -1;
            }

            switch(event->response_type & ~0x80)
            {
                case XCB_KEY_PRESS:
                    /* Quit on key press */
                    running = 0;
                    break;
                case XCB_EXPOSE:
                    /* Handle expose event, draw and swap buffers */
                    draw();
```

```
                    glXSwapBuffers(display, drawable);
                    break;
                default:
                    break;
            }

            free(event);
        }

        return 0;
    }

    int setup_and_run(Display* display, xcb_connection_t *connection, int default_screen, xcb_
    {
        int visualID = 0;

        /* Query framebuffer configurations that match visual_attribs */
        GLXFBConfig *fb_configs = 0;
        int num_fb_configs = 0;
        fb_configs = glXChooseFBConfig(display, default_screen, visual_attribs, &num_fb_config
        if(!fb_configs || num_fb_configs == 0)
        {
            fprintf(stderr, "glXGetFBConfigs failed\n");
            return -1;
        }

        printf("Found %d matching FB configs", num_fb_configs);

        /* Select first framebuffer config and query visualID */
        GLXFBConfig fb_config = fb_configs[0];
        glXGetFBConfigAttrib(display, fb_config, GLX_VISUAL_ID , &visualID);

        GLXContext context;

        /* Create OpenGL context */
        context = glXCreateNewContext(display, fb_config, GLX_RGBA_TYPE, 0, True);
        if(!context)
        {
            fprintf(stderr, "glXCreateNewContext failed\n");
            return -1;
        }

        /* Create XID's for colormap and window */
        xcb_colormap_t colormap = xcb_generate_id(connection);
        xcb_window_t window = xcb_generate_id(connection);

        /* Create colormap */
        xcb_create_colormap(
            connection,
            XCB_COLORMAP_ALLOC_NONE,
            colormap,
            screen->root,
            visualID
            );

        /* Create window */
        uint32_t eventmask = XCB_EVENT_MASK_EXPOSURE | XCB_EVENT_MASK_KEY_PRESS;
        uint32_t valuelist[] = { eventmask, colormap, 0 };
        uint32_t valuemask = XCB_CW_EVENT_MASK | XCB_CW_COLORMAP;

        xcb_create_window(
            connection,
            XCB_COPY_FROM_PARENT,
            window,
            screen->root,
            0, 0,
            150, 150,
            0,
            XCB_WINDOW_CLASS_INPUT_OUTPUT,
```

```
            visualID,
            valuemask,
            valuelist
            );


        // NOTE: window must be mapped before glXMakeContextCurrent
        xcb_map_window(connection, window);

        /* Create GLX Window */
        GLXDrawable drawable = 0;

        GLXWindow glxwindow =
            glXCreateWindow(
                display,
                fb_config,
                window,
                0
                );

        if(!window)
        {
            xcb_destroy_window(connection, window);
            glXDestroyContext(display, context);

            fprintf(stderr, "glXDestroyContext failed\n");
            return -1;
        }

        drawable = glxwindow;

        /* make OpenGL context current */
        if(!glXMakeContextCurrent(display, drawable, drawable, context))
        {
            xcb_destroy_window(connection, window);
            glXDestroyContext(display, context);

            fprintf(stderr, "glXMakeContextCurrent failed\n");
            return -1;
        }

        /* run main loop */
        int retval = main_loop(display, connection, window, drawable);

        /* Cleanup */
        glXDestroyWindow(display, glxwindow);

        xcb_destroy_window(connection, window);

        glXDestroyContext(display, context);

        return retval;
    }

    int main(int argc, char* argv[])
    {
        Display *display;
        int default_screen;

        /* Open Xlib Display */
        display = XOpenDisplay(0);
        if(!display)
        {
            fprintf(stderr, "Can't open display\n");
            return -1;
        }

        default_screen = DefaultScreen(display);
```

```
        /* Get the XCB connection from the display */
        xcb_connection_t *connection =
            XGetXCBConnection(display);
        if(!connection)
        {
            XCloseDisplay(display);
            fprintf(stderr, "Can't get xcb connection from display\n");
            return -1;
        }

        /* Acquire event queue ownership */
        XSetEventQueueOwner(display, XCBOwnsEventQueue);

        /* Find XCB screen */
        xcb_screen_t *screen = 0;
        xcb_screen_iterator_t screen_iter =
            xcb_setup_roots_iterator(xcb_get_setup(connection));
        for(int screen_num = default_screen;
            screen_iter.rem && screen_num > 0;
            --screen_num, xcb_screen_next(&screen_iter));
        screen = screen_iter.data;

        /* Initialize window and OpenGL context, run main loop and deinitialize */
        int retval = setup_and_run(display, connection, default_screen, screen);

        /* Cleanup */
        XCloseDisplay(display);

        return retval;
    }
```