xcb / opengl

Modifier | Historique des pages | Informations sur le dépôt

Vue d'ensemble : OpenGL, Xlib et GLX

Une application typique du système X Windowing est créée à l'aide de Xlib pour communiquer avec le serveur X. Xlib est l'interface de programmation standard du système X Windowing depuis des décennies. Une application OpenGL sous X Windows doit utiliser GLX, une API standardisée, pour configurer un contexte de rendu. L'API GLX est étroitement couplée à Xlib.

Dans une configuration de bureau Linux ou Unix typique, la bibliothèque d'implémentation OpenGL est fournie par les pilotes de la carte graphique. La bibliothèque OpenGL fournit également l'implémentation de l'API GLX. Le système GLX a deux rôles : il communique avec le serveur X et initialise l'état côté client et matériel. La communication client-serveur GLX s'effectue à l'aide d'un protocole filaire GLX standardisé, qui est une extension du protocole réseau X. La bibliothèque GLX résume toutes les initialisations côté client et matérielles et les éléments internes du processus sont cachés dans la bibliothèque d'implémentation OpenGL. Les fonctions GLX peuvent également communiquer avec le serveur X à l'aide d'extensions de protocole réseau X propriétaires.

L'API GLX est spécifiée en termes de Xlib, les fonctions glX utilisent les affichages Xlib, Windows, visuels, etc. Les implémentations GLX sont également construites à l'aide de Xlib.

XCB et XCB-GLX

XCB est une bibliothèque d'interface de programmation de bas niveau pour le système X Windowing. Il se compose de liaisons en langage C aux requêtes et réponses du protocole réseau X. Les liaisons sont automatiquement générées à partir d'une description XML du protocole réseau sous-jacent. XCB-GLX est la liaison C au protocole filaire GLX et elle est également générée à partir de descriptions XML transcrites à partir des spécifications du protocole filaire GLX.

XCB-GLX communique uniquement avec le serveur X, il n'effectue aucune initialisation matérielle et ne touche pas l'état côté client OpenGL. Pour cette raison, XCB-GLX ne peut pas être utilisé en remplacement de l'API GLX. Pour utiliser OpenGL dans le système X Windowing, il faut utiliser l'API GLX, et l'API GLX est étroitement couplée à Xlib. Par conséquent, une application OpenGL sur X Windows doit utiliser Xlib et ne peut donc pas être réalisée en utilisant uniquement XCB.

Bien que l'API XCB-GLX ait peu de valeur pour les développeurs d'applications XCB utilisateurs finaux, elle peut être utilisée dans le développement de nouvelles implémentations OpenGL et GLX basées sur XCB. XCB pourrait potentiellement améliorer la vitesse et la qualité des bibliothèques OpenGL.

Utiliser OpenGL dans les applications XCB (avec Xlib)

Bien qu'il ne soit pas possible d'écrire une application OpenGL sur X Windows à l'aide de XCB pur, il est possible d'utiliser une implémentation Xlib basée sur XCB pour configurer un contexte de rendu dans une application hybride XCB/Xlib. Xlib est utilisé avec les fonctions GLX tandis que XCB peut être utilisé pour tout le reste. Vous bénéficiez de tous les avantages de l'utilisation de XCB, mais vous ne pouvez pas vous débarrasser entièrement de Xlib.

Le processus de mise en place d'une application hybride XCB/Xlib avec OpenGL est plutôt simple. XOpenDisplay est utilisé pour configurer un affichage Xlib, et XGetXCBConnection et XSetEventQueueOwner (à partir de X11/Xlib-xcb.h) sont utilisés pour obtenir la connexion XCB et configurer XCB pour gérer la file d'attente d'événements au lieu de Xlib. Le Xlib Display peut ensuite être utilisé pour la configuration du contexte GLX tandis que la connexion XCB peut être utilisée pour tout le reste. La communication entre XCB et GLX s'effectue à l'aide d'identifiants basés sur des nombres entiers (XID), qui peuvent être transmis en toute sécurité entre XCB et Xlib/GLX.

Travail futur

Pour une solution OpenGL durable basée sur XCB, une nouvelle API de remplacement pour GLX devrait être conçue et implémentée. L'API GLX pourrait, en théorie, être repensée au-dessus de XCB, en utilisant la connexion XCB et le XCB-GLX pour la communication interne du serveur X. La nouvelle API GLX XCB'ified pourrait être utilisée pour écrire une application OpenGL avec du XCB pur sans dépendance à Xlib.

Le problème évident est que GLX est une API standardisée et possède des dizaines d'implémentations différentes. Le standard GLX est régi par le groupe Khronos et c'est lui qui a pris l'initiative d'un nouveau standard API. La mise en œuvre de la nouvelle API relèverait des fournisseurs de pilotes graphiques.

Une conception et une mise en œuvre de validation de principe pourraient être réalisées à l'aide de MESA, une implémentation OpenGL open source. Pour maintenir la compatibilité ascendante et fournir un chemin de portage incrémentiel, l'API GLX actuelle pourrait être implémentée à l'aide de l'API XCB'ified en dessous, comme le fait l'implémentation Xlib basée sur XCB.

Exemple de code d'OpenGL avec XCB et Xlib

```
#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,
    {
        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_scre
    {
        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_f
        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;
```

Dernière modification dim . 10 juin 2018 14:38:52