Embedded System Software

Project - Mini Game

20171664
이상윤

Part 1

Motivation

Part 1

Motivation

Part 2

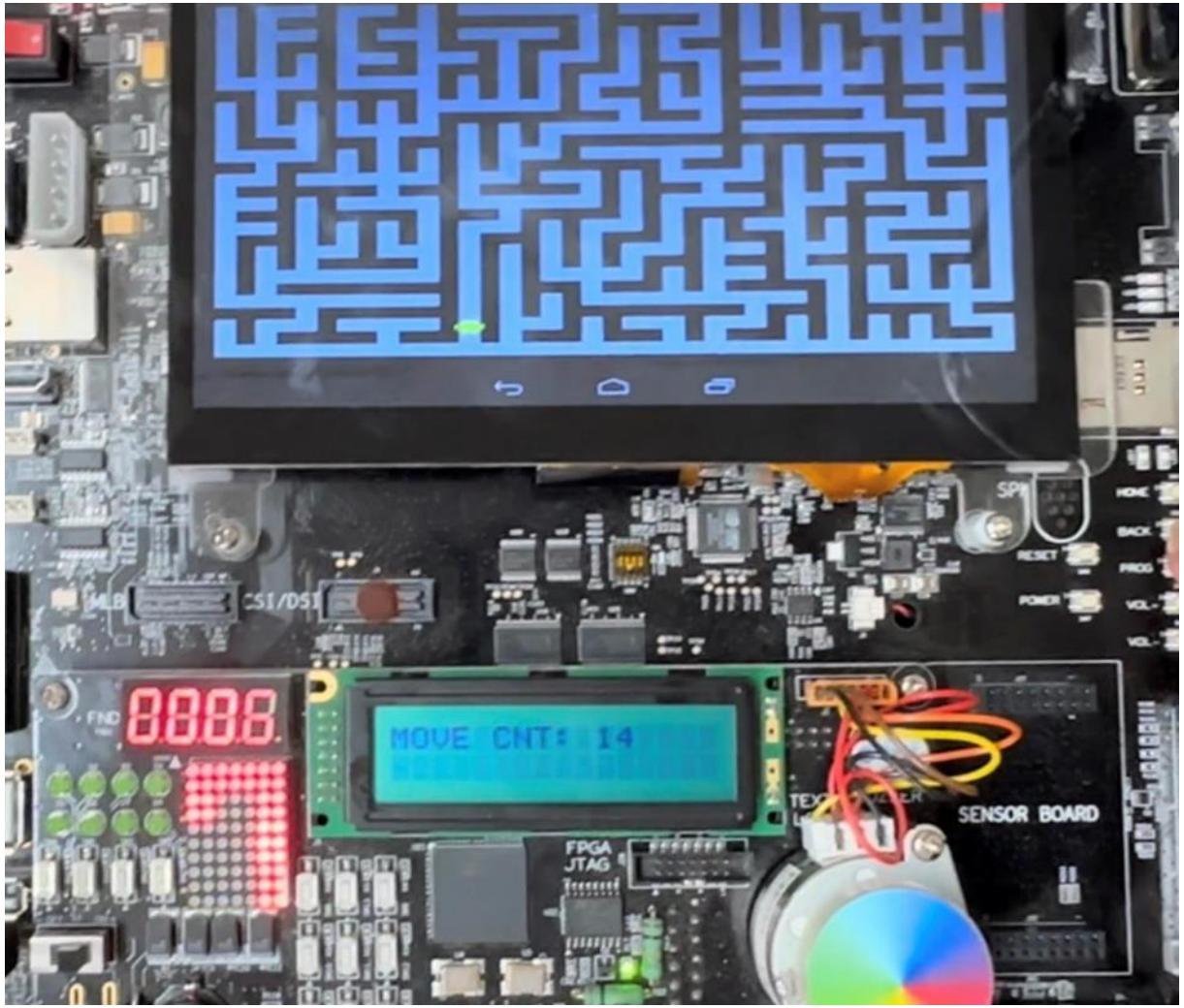
Game1 - Car Avoidance Game

- Use the dip switch to move up, down, left, and right.
- Press the back button (interrupt) to open the game settings screen.
- Timer is displayed on the FND and DOT during the game.
- Start with 3 lives, displayed on the LED and text LCD.
- The game ends when Androboy collects 3 cars, followed by an end motion.

Part 2

Game 2 – Maze Game

- Use the dip switch to move up, down, left, and right.
- Press the back button (interrupt) to open the game settings screen.
- Timer is displayed on the FND and DOT during the game.
- The number of moves is displayed on the text LCD.
- The game ends when Androman reach the goal point.



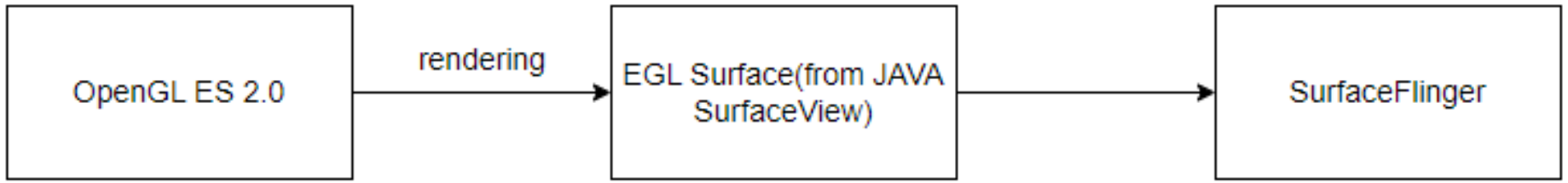
Part 3

Details

- External Library: GLM 0.9.5.4 (Header-Only library version, 2014-06-21, located at /work/mydroid/android-ndk-r10e/platforms/android-19/arch-arm/usr/include)

Part 4

Architecture - Rendering



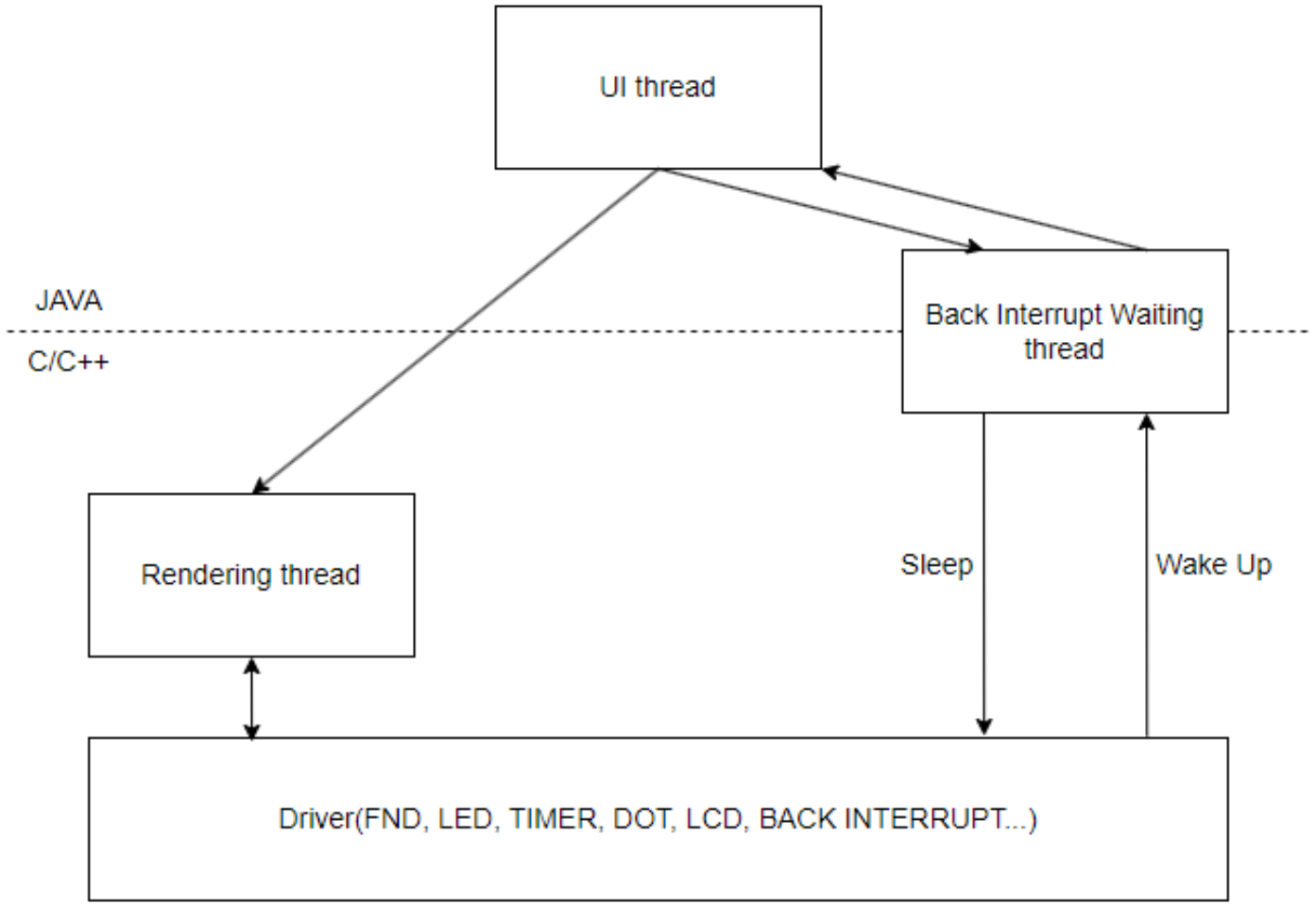

```
struct egl_status{
        ANativeWindow* window;
        EGLDisplay display;
        EGLSurface surface;
        EGLContext context;
        EGLConfig config;
        EGLint numConfigs;
        EGLint format;
        EGLint width;
        EGLint height;

        bool exists_window;
};
static struct egl_status egl;
```

Rendering thread is needed(OpenGL ES context requires its own thread).

Part 4

Architecture - Interrupt



Part 4

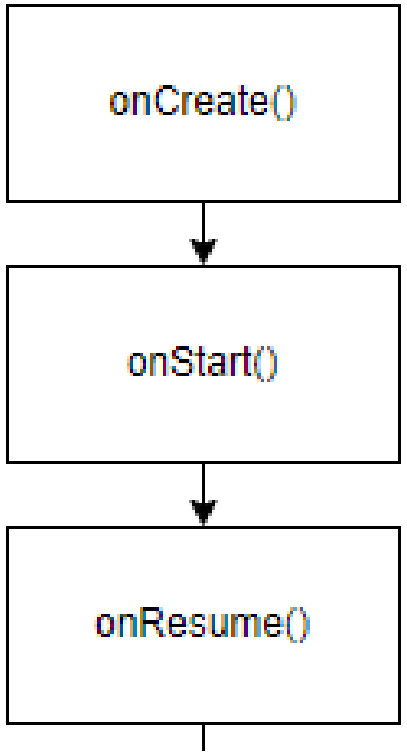
Lifecycle Consideration

- Considerations: C/C++ thread, JAVA thread, Surface lifecycle, onDestroy() not always called, screen off state(PROG button), forced termination state, etc.

- Key Facts:
  - close() is called unless threads remain(This seems to be due to Garbage Collector).
  - onCreate() is always called when the activity is created.
  - surfaceChanged() is called at least once when surface is created.
  - onResume() -> surfaceChanged() -> onPause() -> surfaceDestroyed() is guaranteed. If the created Surface is not destroyed, surfaceCreated(), surfaceChanged() will be ignored.

Part 4

Lifecycle Consideration



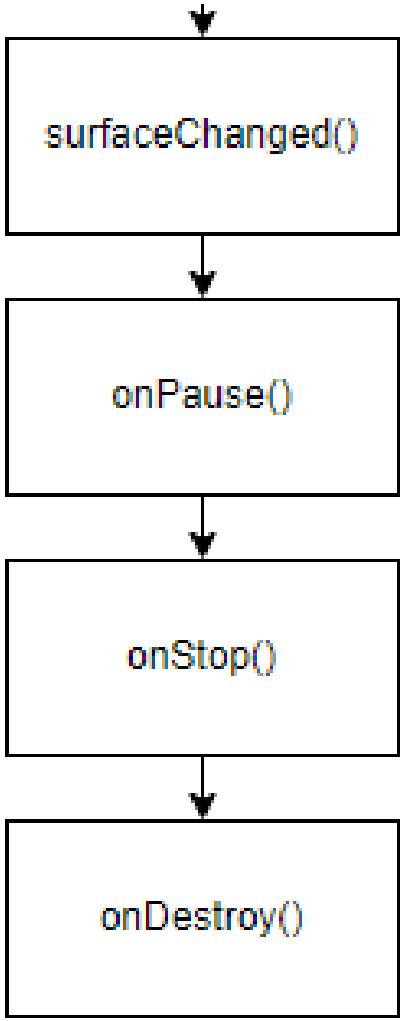
- open dev file.
- Initialize fpga and game attributes.

- Handle resource allocation like pthread_mutex_init().

- Create rendering thread(if not created initially) and back interrupt waiting thread.
- Set fpga and egl attributes.

Part 4

Lifecycle Consideration



- Create rendering thread.
- Set egl.

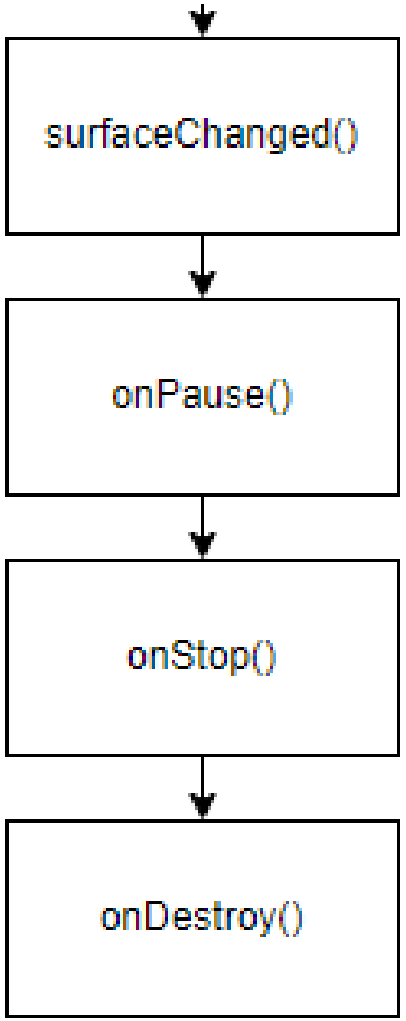
- **Destroy rendering thread and back interrupt waiting thread.**

- Deallocate

- close

Part 4

Lifecycle Consideration



- Create rendering thread.
- Set egl.

- **Destroy rendering thread and back interrupt waiting thread.**

- Deallocate

- close

Part 5

Code

```
▸ Documentation/
▸ images/
▾ MiniGameController/
  ▾ jni/
    ▾ game1/
      ▾ models/
          androboy.cpp
          androboy.h
          car.cpp
          car.h
          house.cpp
          house.h
          road.cpp
          road.h
          sword.cpp
          sword.h
      ▾ shaders/
          loader.cpp
          loader.h
        jniapi.cpp
        jniapi.h
        renderer.cpp
        renderer.h
    ▸ game2/
      Android.mk
      Application.mk
      logger.h
  ▸ libs/
  ▸ res/
  ▾ src/com/example/minigamecontroller/
      BackPopupActivity.java
      Game1Activity.java
      Game2Activity.java
      MainActivity.java
    AndroidManifest.xml
    ic_launcher-web.png
    proguard-project.txt
    project.properties
▾ modules/
    driver.c
    interrupt_ctrl.c
    interrupt_ctrl.h
    led_ctrl.c
    led_ctrl.h
    logging.h
    Makefile
    prepare.sh
    switch_ctrl.c
    switch_ctrl.h
    text_lcd_ctrl.c
    text_lcd_ctrl.h
    timer_ctrl.c
    timer_ctrl.h
▸ work/mydroid/android-ndk-r10e/platforms/android-19/arch-arm/usr/include/glm/
  Readme.md
```

Part 5 Code

```
void game1_resume(void){
        pthread_mutex_lock(&gstate.mutex);
        gstate.is_paused = false;
        pthread_mutex_unlock(&gstate.mutex);

        // At most onResume() -> surfaceChanged() -> onPause() -> surfaceDestroyed().
        // However, if the created surface is not destroyed, from the second onResume onwards,
        // surfaceCreated and surfaceChanged will be ignored.
        // In other words, the rendering thread is initially started from surfaceChanged
        // and subsequently from onResume.
        if(egl.exists_window)
                pthread_create(&gstate.tid, NULL, render_loop, NULL);
        else
                LOG_INFO("First onResume()");

        ioctl(gpad.fd, IOCTL_RUN_TIMER_NONBLOCK);
}
```

```
static void* render_loop(void* nouse){
        acquire_context();

        // change screeen vertical to horizontal and normalize coords.
        vp_matrix = glm::ortho(-egl.width / 2.0, egl.width / 2.0,
                -egl.height / 1.35, egl.height / 1.35, -1000.0, 1000.0);

        while(true){
                // consider onPause()
                pthread_mutex_lock(&gstate.mutex);
                if(gstate.is_paused){
                        release_context();
                        pthread_mutex_unlock(&gstate.mutex);
                        pthread_exit(0);
                }
                read_gpad();
                process_gameover();
                draw_frame();
                if (!eglSwapBuffers(egl.display, egl.surface)) {
                        LOG_ERROR("eglSwapBuffers() returned error %d", eglGetError());
                }
                ++gstate.cur_time;
                pthread_mutex_unlock(&gstate.mutex);
                usleep(10000); // 10ms
        }
}
```

```
/*
 * In a blocking manner, the back interrupt detector thread, which is passed in Java,
 * detects the back button interrupt.
 */
bool game1_wait_back_interrupt(void){
        int waked_by_intr;
        ioctl(gpad.fd, IOCTL_WAIT_BACK_INTERRUPT, &waked_by_intr);
        if(waked_by_intr) {
                LOG_INFO("waked up by interrupt");
                return true;
        }
        LOG_INFO("waked up by pause");
        return false;
}
```

```
/*
 * Block JAVA's thread.
 */
int interrupt_wait_back_intr(void){
        init_completion(&over);
        wait_for_completion(&over);
        return waked_by_back_handler;
}
void interrupt_wake_back_waiting_thread(void){
        spin_lock(&sl);
        waked_by_back_handler = 0;
        complete(&over);
        spin_unlock(&sl);
}
```

```
class BackInterruptDetector extends Thread{
        BackInterruptDetector(){}

        public void run(){
                Log.i(TAG, "Interrupt Dectector started");
                // Blocking manner
                if(nativeWaitBackInterrupt()){
                        Log.i(TAG, "Waked up by interrupt");
                        // wake up
                        Intent intent = new Intent(Game1Activity.this, BackPopupActivity.class);
                        intent.putExtra("CALLING_ACTIVITY", Game1Activity.class);
                        startActivity(intent);
                        overridePendingTransition(0, 0);
                }
                else{
                        Log.i(TAG, "Waked up by pause");
                }
        }
}
private BackInterruptDetector backInterruptDetector;
```

Part 5

Code(game2)

```cpp
static std::vector<std::vector<int> > generate_maze(JNIEnv* env, jobject obj){
        jclass cls = env->GetObjectClass(obj);
        jmethodID mid = env->GetMethodID(cls, "generateMaze","()[[I");
        if(mid == NULL)
                LOG_ERROR("method id not found");

        std::vector<std::vector<int> > ret;
        jobjectArray res = (jobjectArray)env->CallObjectMethod(obj, mid);
        jsize row = env->GetArrayLength(res);
        ret.resize(row);
        for(int i = 0; i < row; ++i){
                jintArray jarr = (jintArray)env->GetObjectArrayElement(res, i);
                jsize col = env->GetArrayLength(jarr);
                ret[i].resize(col);
                jint* carr = env->GetIntArrayElements(jarr, NULL);
                for(int j = 0; j < col; ++j)
                        ret[i][j] = carr[j];
                env->ReleaseIntArrayElements(jarr, carr, 0);
        }
        return ret;
}
```

```cpp
private int[][] generateMaze() {
        // initialize
        int[][] ret = new int[2*ROW + 1][2*COL + 1];
        int[][] profile = new int[ROW][COL];
        for(int i = 0; i < 2*ROW + 1; ++i){
                for(int j = 0; j < 2*COL + 1; ++j){
                        if((i & 1) != 0 && (j & 1) != 0)
                                ret[i][j] = 0;
                        else
                                ret[i][j] = 1;
                }
        }
        for(int i = 0; i < ROW; ++i)
                for(int j = 0; j < COL; ++j)
                        profile[i][j] = i*COL + j;

        // Eller's Algorithm
```