ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

GameDev и ОС Аврора

Портирование игры

Андреев Ярослав
омп

Mobius 2023 Autumn

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

О себе

- c/c++ разработчик (10+ лет)
- работаю в ОМП
- увлечен компьютерными играми и геймдевом (всю жизнь)
- Open Source разработчик под ОС Аврора / SailfishOS (6 лет)
- портирую игры на ОС Аврора, just for fun

Github

Boosty

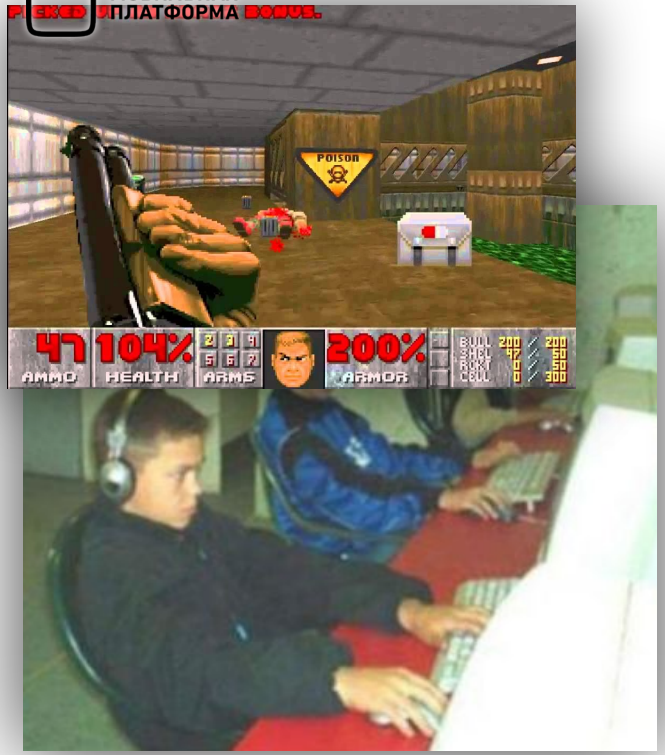
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

История

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней



История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней
- Doom

История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней
- Doom -> Quake

История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней
- Doom -> Quake -> Quake 2

История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней
- Doom -> Quake -> Quake 2
- в 2017 купил SailfishOS RUS смартфон

История

- конец 90-х, начало 2000, когда компьютерные игры становятся все круче и технологичней
- Doom -> Quake -> Quake 2
- в 2017 купил SailfishOS RUS смартфон
- просто скомпилить и готово!

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Оказалось все не так просто …

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Оказалось все не так просто …

О чем пойдет речь

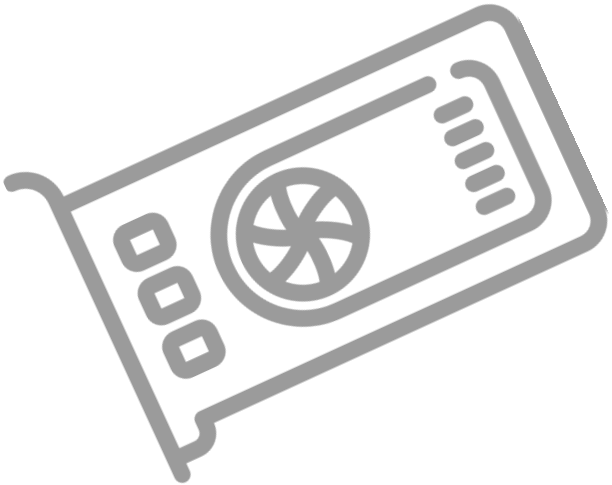
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

О чем пойдет речь

- графический стандарт
- композитор
- поворот экрана
- отклик на события системы
- воспроизведение аудио
- внешние устройства
- запрет гашения экрана
- чтение датчиков (акселерометр, гироскоп)

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Графический стандарт



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Графический стандарт

- OpenGL
+ GLESv2 (OpenGL ES v2)
+ GLESv3 (OpenGL ES v3)
+ vulkan

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Графический стандарт

- OpenGL
+ GLESv2 (OpenGL ES v2)
+ GLESv3 (OpenGL ES v3)
- ~~vulkan~~ (пока что нет в ОС Аврора)

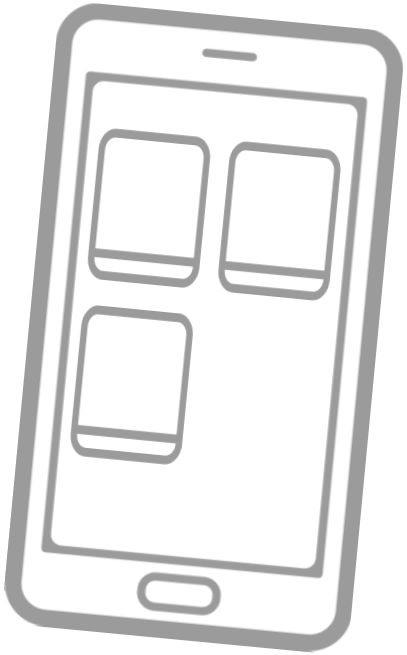
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Графический стандарт

- OpenGL
+ GLESv2 (OpenGL ES v2)
+ GLESv3 (OpenGL ES v3)
- ~~vulkan~~ (пока что нет в ОС Аврора)

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

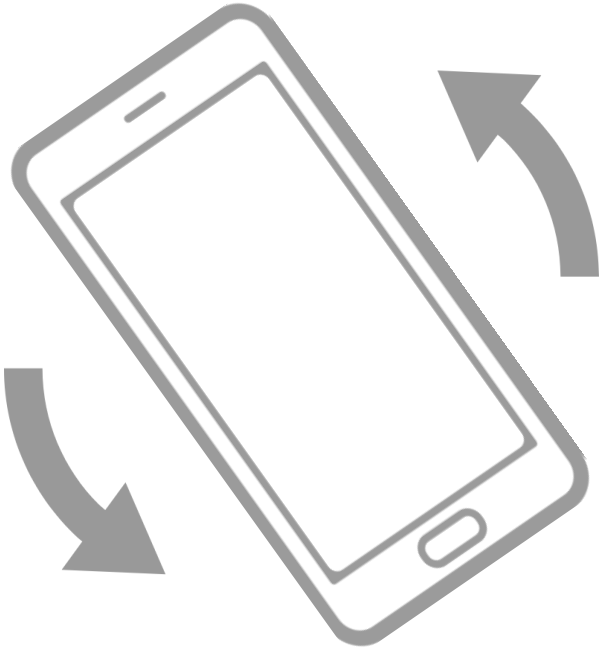
Композитор



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Композитор

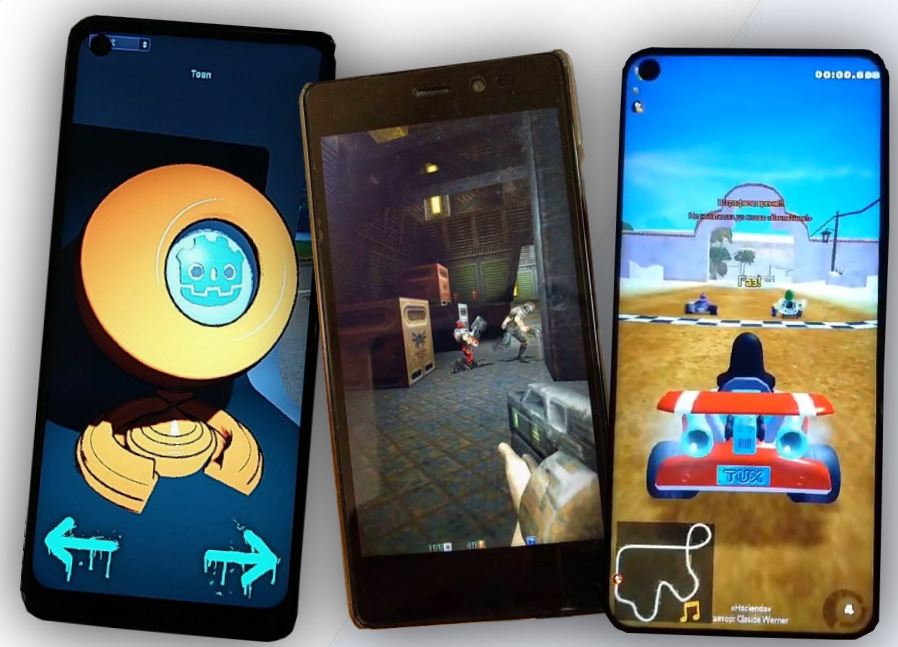
Lipstick и поворот экрана



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

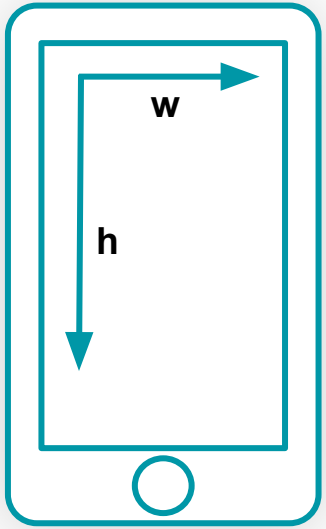
- Qt приложения с использованием Sailfish Silica и Aurora Controls
- Все остальные приложения (Wayland EGL, SDL2, GLFW и даже Qt, и др)



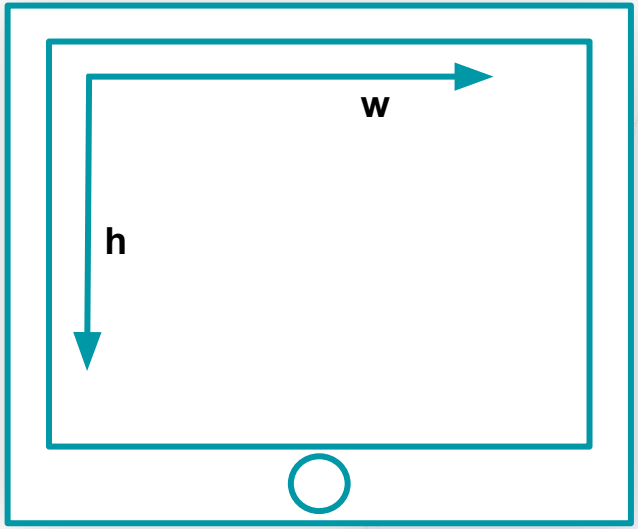
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

- нативная развертка



Native Portrait



Native Landscape

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

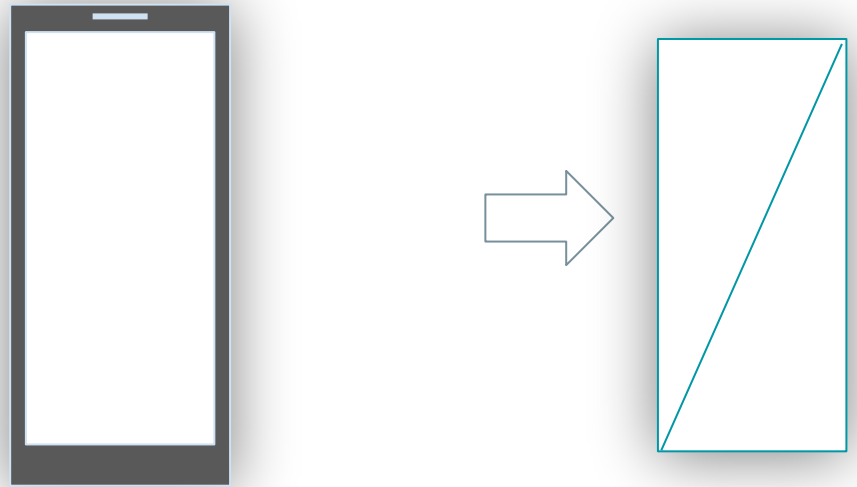
Используем FBO (Frame Buffer Object)

Native
Portrait

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

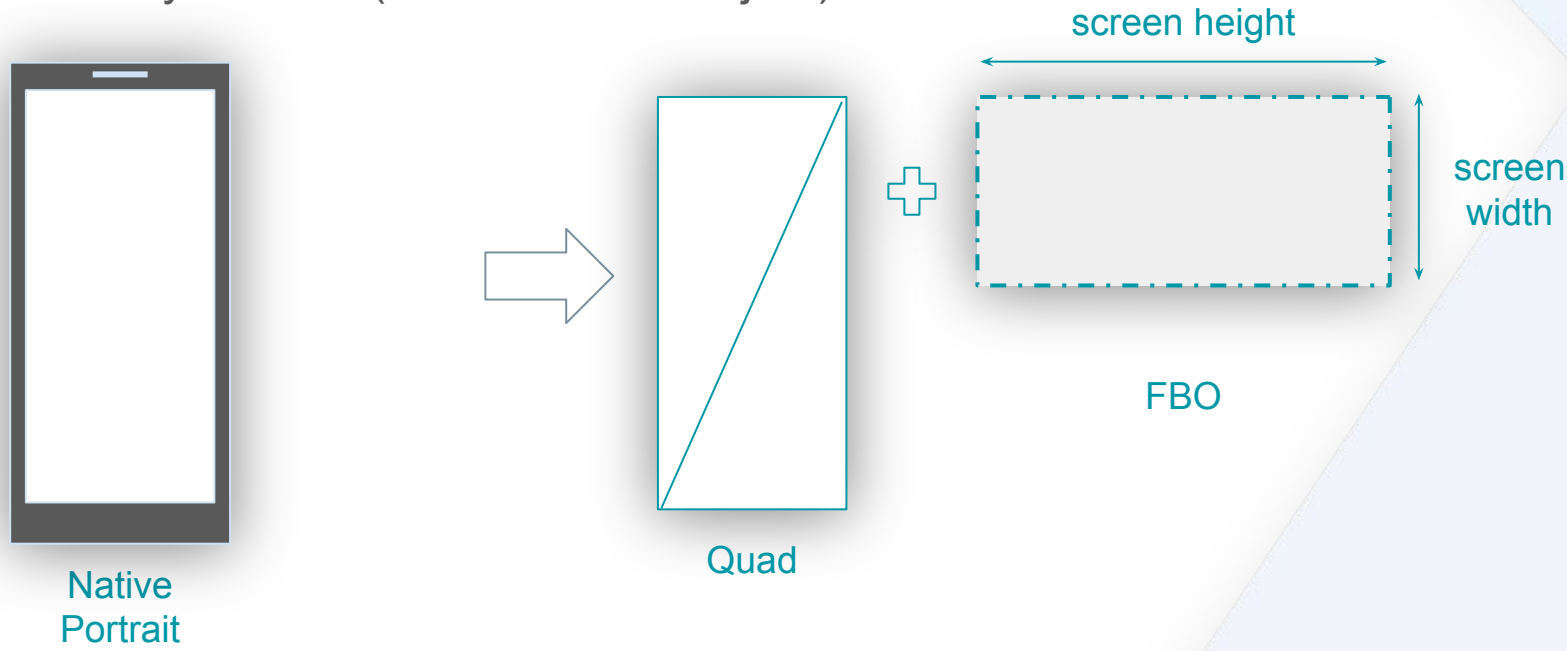
Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)



Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)

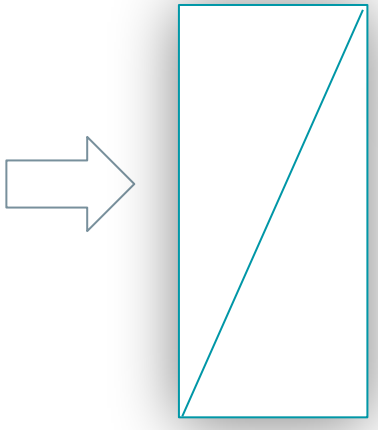


ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

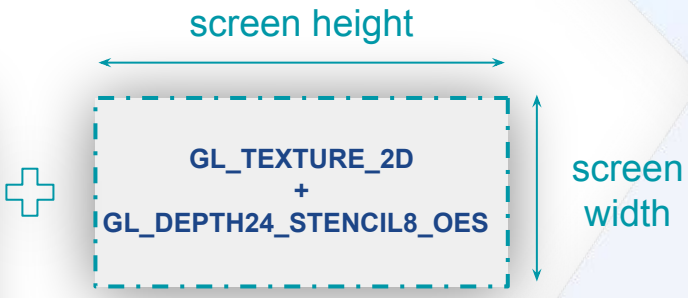
Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)

Native Portrait



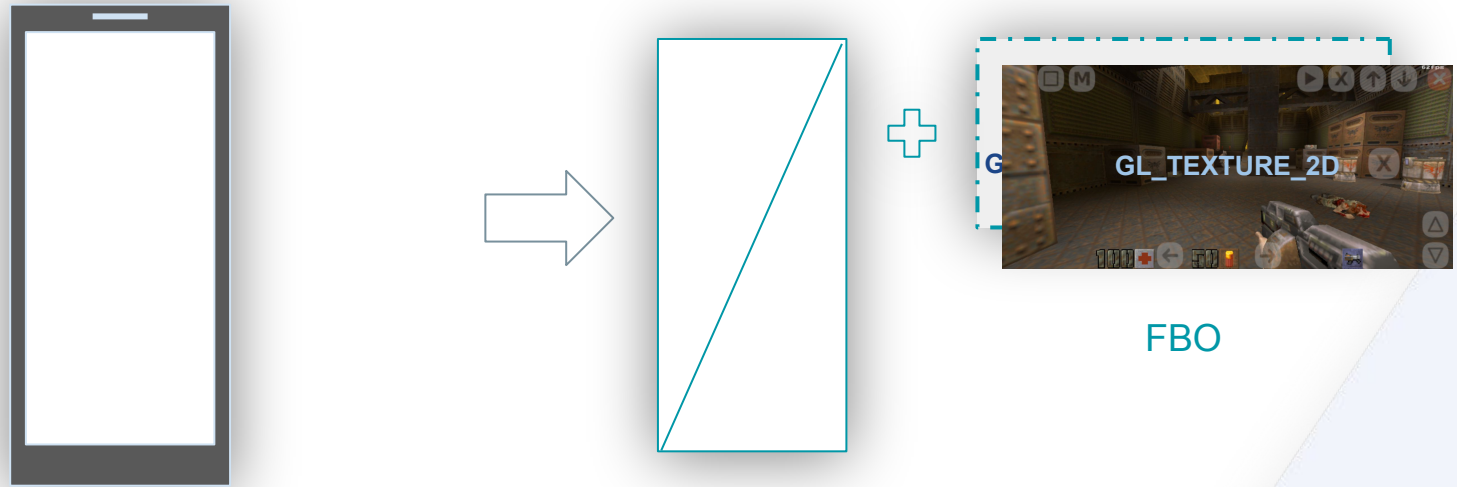
Quad



FBO

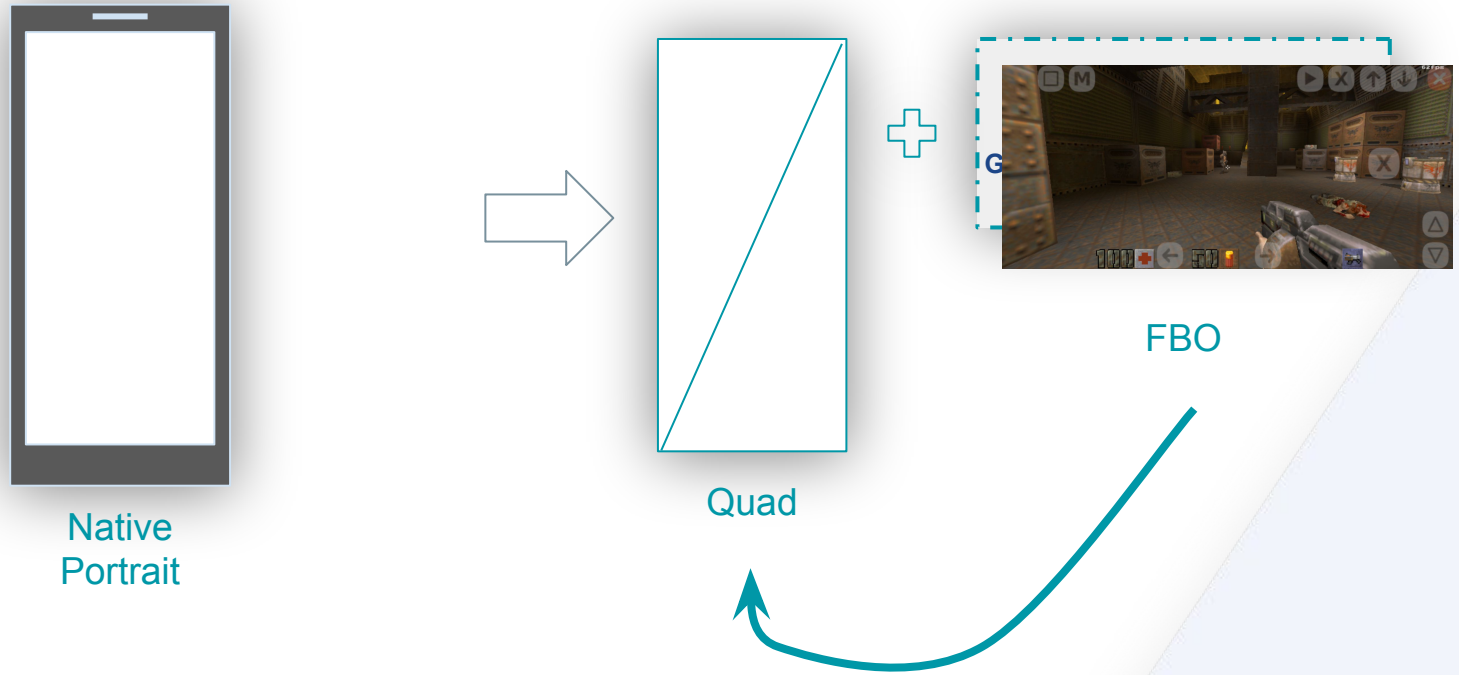
Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)



Lipstick и поворот экрана

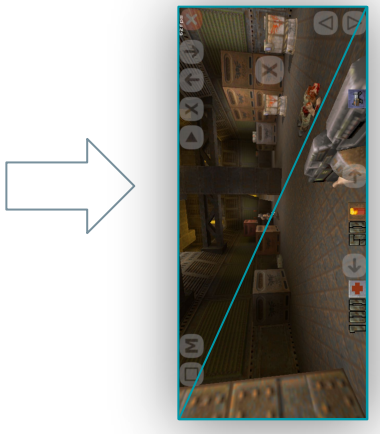
Используем FBO (Frame Buffer Object)



Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)

Native Portrait



Quad

Lipstick и поворот экрана

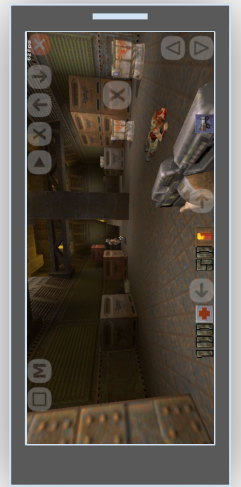
Используем FBO (Frame Buffer Object)

Native Portrait

Quad

Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)



Native Portrait

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)

Lipstick и поворот экрана

Используем FBO (Frame Buffer Object)

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

Сообщаем в Wayland поворот контента нашего приложения. В случае SDL2:

```
#include <SDL_video.h>
#include <SDL_syswm.h>
#include <wayland-client-protocol.h>

void setOrientation( SDL_Window *window, SDL_DisplayOrientation orientation) {
   struct SDL_SysWMinfo wmInfo;
   SDL_VERSION (&wmInfo.version);
   SDL_GetWindowWMInfo( window, &wmInfo);

   switch (orientation) {
   case SDL_ORIENTATION_LANDSCAPE :
       wl_surface_set_buffer_transform(wmInfo.info.wl.surface,  WL_OUTPUT_TRANSFORM_270 );
       break;
   case SDL_ORIENTATION_LANDSCAPE_FLIPPED :
       wl_surface_set_buffer_transform(wmInfo.info.wl.surface,  WL_OUTPUT_TRANSFORM_90 );
       break;
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

Сообщаем в Wayland поворот контента нашего приложения. В случае SDL2:

```
#include <SDL_video.h>
#include <SDL_syswm.h>
#include <wayland-client-protocol.h>

void setOrientation(SDL_Window *window, SDL_DisplayOrientation orientation) {
    struct SDL_SysWMinfo wmInfo;
    SDL_VERSION(&wmInfo.version);
    SDL_GetWindowWMInfo(window, &wmInfo);

    switch (orientation) {
    case SDL_ORIENTATION_LANDSCAPE:
        wl_surface_set_buffer_transform(wmInfo.info.wl.surface, WL_OUTPUT_TRANSFORM_270);
        break;
    case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
        wl_surface_set_buffer_transform(wmInfo.info.wl.surface, WL_OUTPUT_TRANSFORM_90);
        break;
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

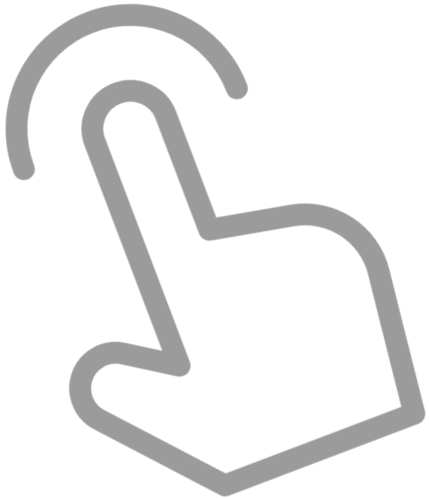
Сообщаем в Wayland поворот контента нашего приложения. В случае SDL2:

```
#include <SDL_video.h>
#include <SDL_syswm.h>
#include <wayland-client-protocol.h>

void setOrientation(SDL_Window *window, SDL_DisplayOrientation orientation) {
   struct SDL_SysWMinfo wmInfo;
   SDL_VERSION(&wmInfo.version);
   SDL_GetWindowWMInfo(window, &wmInfo);

   switch (orientation) {
   case SDL_ORIENTATION_LANDSCAPE:
       wl_surface_set_buffer_transform(wmInfo.info.wl.surface, WL_OUTPUT_TRANSFORM_270);
       break;
   case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
       wl_surface_set_buffer_transform(wmInfo.info.wl.surface, WL_OUTPUT_TRANSFORM_90);
       break;
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Lipstick и поворот экрана

Отклик на события



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- координаты сенсорного экрана нужно поворачивать так же как и рендер

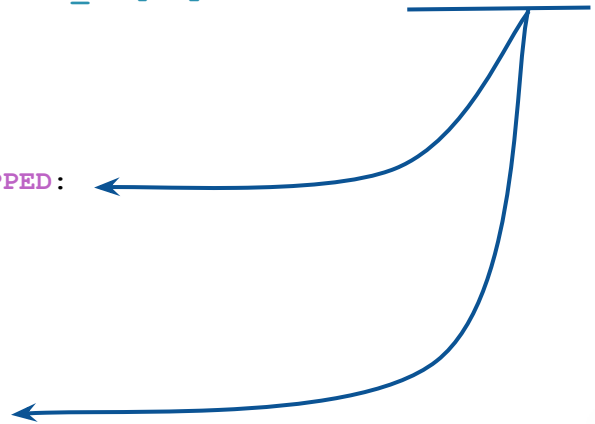
```
void transformTouch(SDL_TouchFingerEvent &tfinger,
                    SDL_Window *window, SDL_DisplayOrientation orientation) {
  int w,h;
  float tmp;
  SDL_GetWindowSize(window, &w, &h);
  switch (orientation) {
  case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
      tmp = tfinger.y;
      tfinger.y = tfinger.x;
      tfinger.x = w - tmp;
      break;
  default:
  case SDL_ORIENTATION_LANDSCAPE:
      tmp = tfinger.y;
      tfinger.y = h - tfinger.x;
      tfinger.x = tmp;
      break;
  }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- координаты сенсорного экрана нужно поворачивать так же как и рендер

```
void transformTouch(SDL_TouchFingerEvent &tfinger,
                    SDL_Window *window, SDL_DisplayOrientation orientation) {
  int w,h;
  float tmp;
  SDL_GetWindowSize(window, &w, &h);
  switch (orientation) {
  case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
      tmp = tfinger.y;
      tfinger.y = tfinger.x;
      tfinger.x = w - tmp;
      break;
  default:
  case SDL_ORIENTATION_LANDSCAPE:
      tmp = tfinger.y;
      tfinger.y = h - tfinger.x;
      tfinger.x = tmp;
      break;
  }
}
```



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- координаты сенсорного экрана нужно поворачивать так же как и рендер

```
void transformTouch(SDL_TouchFingerEvent &tfinger,
                    SDL_Window *window, SDL_DisplayOrientation orientation) {
  int w,h;
  float tmp;
  SDL_GetWindowSize(window, &w, &h);
  switch (orientation) {
  case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
      tmp = tfinger.y;
      tfinger.y = tfinger.x;
      tfinger.x = w - tmp;
      break;
  default:
  case SDL_ORIENTATION_LANDSCAPE:
      tmp = tfinger.y;
      tfinger.y = h - tfinger.x;
      tfinger.x = tmp;
      break;
  }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- координаты сенсорного экрана нужно поворачивать так же как и рендер

```
void transformTouch(SDL_TouchFingerEvent &tfinger,
                    SDL_Window *window, SDL_DisplayOrientation orientation) {
  int w,h;
  float tmp;
  SDL_GetWindowSize(window, &w, &h);
  switch (orientation) {
  case SDL_ORIENTATION_LANDSCAPE_FLIPPED:
      tmp = tfinger.y;
      tfinger.y = tfinger.x;
      tfinger.x = w - tmp;
      break;
  default:
  case SDL_ORIENTATION_LANDSCAPE:
      tmp = tfinger.y;
      tfinger.y = h - tfinger.x;
      tfinger.x = tmp;
      break;
  }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)
  - постановка на паузу

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)
  - постановка на паузу
  - возобновление при возврате фокуса

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)
  - постановка на паузу
  - возобновление при возврате фокуса
- закрытие приложения

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

Отклик на события

- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)
  - постановка на паузу
  - возобновление при возврате фокуса
- закрытие приложения
  - штатное завершение приложения

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

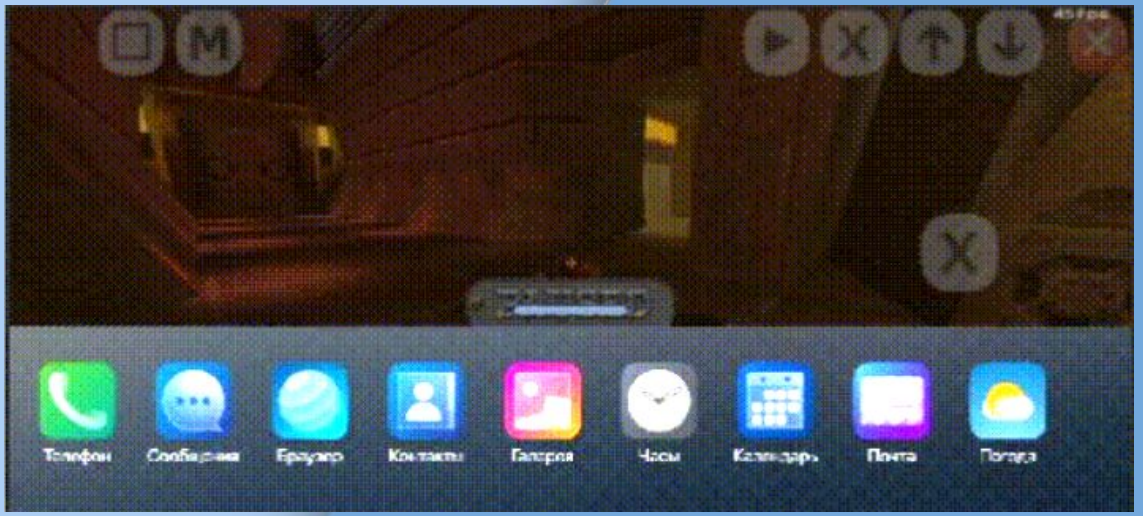
- потеря фокуса (сворачивание приложения, поступление звонка и т.п.)
  - постановка на паузу
  - возобновление при возврате фокуса
- закрытие приложения
  - штатное завершение приложения
  - и прочее

```
void processEvent(SDL_Event *event) {
    switch (event->type) {
    case SDL_WINDOWEVENT:
        switch (event->window.event) {
        case SDL_WINDOWEVENT_CLOSE:
            game->exit();
            break;
        case SDL_WINDOWEVENT_FOCUS_LOST:
            game->pause();
            break;
        case SDL_WINDOWEVENT_FOCUS_GAINED:
            game->continue();
        }
        break;
    case SDL_QUIT:
        game->exit();
    }
}
```

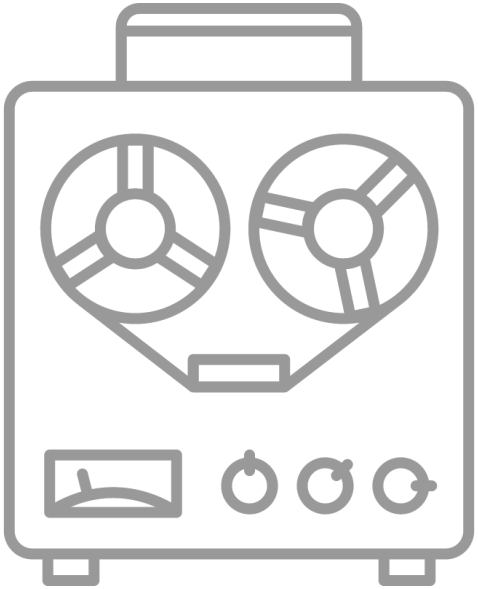
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Отклик на события

ставим на паузу при потере фокуса



Воспроизведение звука



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- ОС Аврора использует PulseAudio
- Чем проигрывать звук:
  - SDL2, OpenAL и совместимое с PulseAudio
- Могут возникнуть проблемы:
  - звука может вообще не быть
  - задержки при воспроизведении звука

- Используем **libaudioresource**
  - помечаем источник звука как **GAME**
  - получаем наименьшие задержки звука

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
    ge->audio_resource = audioresource_init(
        AUDIO_RESOURCE_GAME,
        on_audio_resource_acquired,
        this);
    audioresource_acquire(ge->audio_resource);

    while (!ge->is_audio_resource_acquired) {
        g_main_context_iteration(NULL, false);
    }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
    ge->audio_resource = audioresource_init(
        AUDIO_RESOURCE_GAME,
        on_audio_resource_acquired,
        this);
    audioresource_acquire(ge->audio_resource);

    while (!ge->is_audio_resource_acquired) {
        g_main_context_iteration(NULL, false);
    }
}
```

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- инициализация **libaudioresource**

```
#include <audioresource.h>
#include <glib.h>

static void on_audio_resource_acquired(audioresource_t *, bool, void *user_data);

void init_audio(GameEngine *ge)
{
   ge->audio_resource = audioresource_init(
       AUDIO_RESOURCE_GAME,
       on_audio_resource_acquired,
       this);
   audioresource_acquire(ge->audio_resource);

   while (!ge->is_audio_resource_acquired) {
       g_main_context_iteration(NULL, false);
   }
}
```

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

Воспроизведение звука

- callback функция при получении ресурса

```
static void on_audio_resource_acquired(audioresource_t *audio_resource,
                                       bool acquired, void *user_data)
{
   GameEngine *ge = (GameEngine *)user_data;
   if (acquired)
   {
       ge->is_audio_resource_acquired = true;
       ge->start_audio_driver();
   }
   else
       ge->stop_audio_driver();
}
```

Внешние устройства

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Внешние устройства

- нет программного задания позиции курсора мыши (old Wayland)
- геймпады, клавиатуры - просто работают по BT и OTG

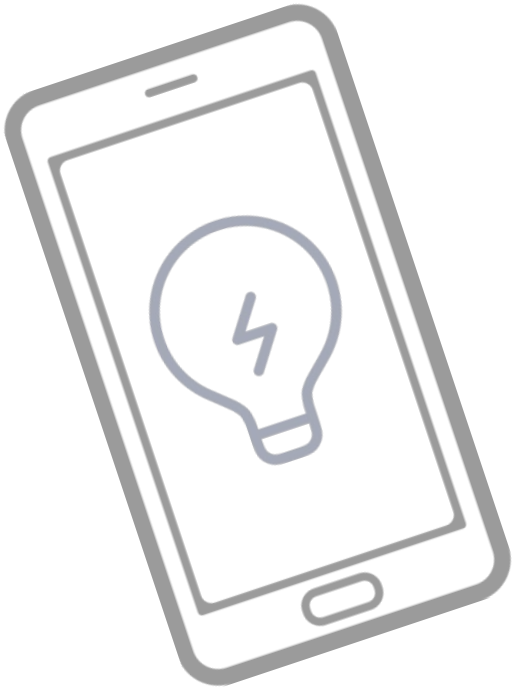
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Внешние устройства

- нюанс - девайс переходит в режим сна, после таймаута гашения экрана

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана

- используется служба MCE (Mode Control Entity) по DBus шине (IPC в ОС Аврора)

Параметры для обращения к сервису по DBus

```
#define MCE_SERVICE                 "com.nokia.mce"
#define MCE_REQUEST_PATH            "/com/nokia/mce/request"
#define MCE_REQUEST_IF              "com.nokia.mce.request"
```

Названия методов которые хотим использовать

```
#define MCE_PREVENT_BLANK_REQ       "req_display_blanking_pause"
#define MCE_CANCEL_PREVENT_BLANK_REQ "req_display_cancel_blanking_pause"
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана

- подключаемся к DBus (пример SDL2)

```
#include <SDL2/src/core/linux/SDL_dbus.h>
...
SDL_DBusContext *context = SDL_DBus_GetContext();
```

- каждые 60 секунд MCE_PREVENT_BLANK_REQ

```
SDL_DBus_CallVoidMethodOnConnection(context->system_conn,
                                    MCE_SERVICE,
                                    MCE_REQUEST_PATH,
                                    MCE_REQUEST_IF,
                                    MCE_PREVENT_BLANK_REQ,
                                    DBUS_TYPE_INVALID);
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана

- подключаемся к DBus (пример SDL2)

```
#include <SDL2/src/core/linux/SDL_dbus.h>
...
SDL_DBusContext *context = SDL_DBus_GetContext();\
```

- каждые 60 секунд **MCE_PREVENT_BLANK_REQ**

```
SDL_DBus_CallVoidMethodOnConnection(context->system_conn,
                                    MCE_SERVICE,
                                    MCE_REQUEST_PATH,
                                    MCE_REQUEST_IF,
                                    MCE_PREVENT_BLANK_REQ,
                                    DBUS_TYPE_INVALID);
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана

- при закрытии приложения (паузе в игре или сворачивании окна) прекращаем держать экран включенным MCE_CANCEL_PREVENT_BLANK_REQ

```
SDL_DBus_CallVoidMethodOnConnection(context->system_conn,
                                    MCE_SERVICE,
                                    MCE_REQUEST_PATH,
                                    MCE_REQUEST_IF,
                                    MCE_CANCEL_PREVENT_BLANK_REQ,
                                    DBUS_TYPE_INVALID);
```

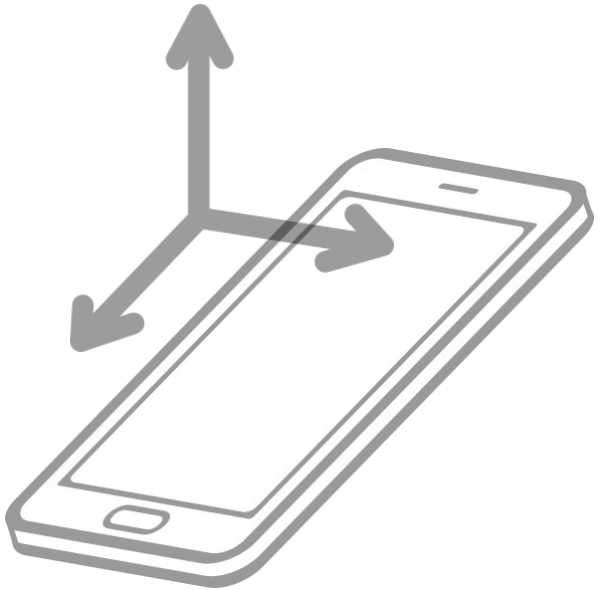
ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Запрет гашения экрана

- при закрытии приложения (паузе в игре или сворачивании окна) прекращаем держать экран включенным **MCE_CANCEL_PREVENT_BLANK_REQ**

```
SDL_DBus_CallVoidMethodOnConnection(context->system_conn,
                                    MCE_SERVICE,
                                    MCE_REQUEST_PATH,
                                    MCE_REQUEST_IF,
                                    MCE_CANCEL_PREVENT_BLANK_REQ,
                                    DBUS_TYPE_INVALID);
```

Чтение датчиков (акселерометр, гироскоп)



ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Чтение датчиков

- основной framework для ОС Аврора это Qt
- через Qt поддерживается весь набор возможностей
- используем хак для работы с датчиками

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Чтение датчиков

- подключим **QCoreApplication** к нашему SDL2 приложению
- инициализируем Qt обертки **QAccelerometer** и **QGyroscope**

```
#include <QCoreApplication>
#include <QAccelerometer>
#include <QGyroscope>
...
QCoreApplication app(0, nullptr);
QAccelerometer accelerometer(&app);
accelerometer.start();

QGyroscope gyroscope(&app);
gyroscope.start();
```

Чтение датчиков

- в нашем основном цикле приложения читаем данные с датчиков

```
while (true) {
    // делаем опрос событий
    QCoreApplication::processEvents(QEventLoop::AllEvents, 1);
    // читаем данные по акселерометру
    game_log("Accelerometer X: %.3f", accelerometer.reading()->x());
    game_log("Accelerometer Y: %.3f", accelerometer.reading()->y());
    // читаем данные по гироскопу
    game_log("Gyroscope X: %.3f", gyroscope.reading()->x());
    game_log("Gyroscope Y: %.3f", gyroscope.reading()->y());
    game_log("Gyroscope X: %.3f", gyroscope.reading()->x());
    // стандартный опрос и обработка событий SDL2
    SDL_Event event;
    while (SDL_PollEvent(&event)) { ... }
}
```

ОТКРЫТАЯ МОБИЛЬНАЯ ПЛАТФОРМА

Спасибо за внимание

Github

Boosty

Aurora Developers