GLFW OPENGL/ES を使った GUI フレームワーク

HIRA_KUNI_45 の略歴

- バイク乗り、整備、エンジンOH
- 主にゲーム関係の仕事に従事
- ゲームボーイ、ファミコン、セガサターン、プレイステーションなど、コンシューマープラットホームなど
- アナログ、デジタルなどハードウェアーの設計、製作、論理合成、基板のアートワークなど
- 金属加工（旋盤、フライス）、溶接
- 基本何でもやるけど化学はあんましやった事無い
- C++はごく普通の練度
- 未だにQuadra950を所有してるけど、Appleには関心が殆ど無い

C++11やBOOSTなど

- 石橋を叩いて渡る傾向で、新しい物には直ぐに飛びつかない
- 新しい物を吸収したいが、難しい部分も・・・
- 当たり障りの無い部分のみ使っている
- 多くは、コレ何に使うんだ？ポカーンな感じ
- 面倒そうなので、従来の方法を使ったり・・
- まぁボチボチ

OPENGL/ES

- SGIのワークステーション時代から使っている
- OpenGL　love！
- Viva　赤本
- 高度なレンダリングには、練度が足りないので何とかしたくて勉強中

※シェーダーなど

- Renderman のシェーダーなら多少書ける
- DirectXは？だと思ってる

※ms 社のOpenGLエバンジェリストに聞いた事

ＧＬＦＷ３とは？

- マルチプラットホームのフレームワーク

※Windows、Linux、OS-X

※ライセンスは unmodified zlib/libpng license

- 必要最小限の機能、シンプルな構造
- サンプルもあり、理解しやすい
- glut の貧弱な部分を強化した感じ

※glutはメンテナンスされていない

GUIフレームワークの開発

- 他のオープンソースのGUIとはちょっと違う
- リアルタイム性を重視
- OpenGL/ESとマッチ
- GUIの構造や構成がC++に向いている
- かなり前から少しづつ構築している為、最近の流行とは逆行する部分も・・・
- 実装は、C++11やboostの機能を的確に利用出来ていない
- 組み込みでも使える構成

※組み込みでは iostream には注意を要す

利用しているオープンソース

- ＧＬＦＷ３
- ＧＬＥＷ
- Freetype2
- OpenAL
- libz、libpng、libjpeg、openjpeg
- Mad(mp3)、AAC
- Mupdf、jbig2dec
- bullet（physics）

GITHUB/E-MAIL/TWITTER

https://github.com/hirakuni45/glfw3_app

hira@rvf-rc45.net

@hira_kuni_45

全体の構成（COMMON）

- core
- gl_fw
- img_io
- snd_io
- widgets
- utils
- minizip
- 開発中（mdf、collada）

CORE

- 機種依存性が高いコード
- GLFW３をラップしたインターフェース
- Freetype2とのインターフェース

※他のフォントライブラリーとインターフェース（VITA、PS3）

- 漢字のFEP関係のインターフェース（開発中）
- 入力デバイスのラッパー（ＧＬＦＷ３）

GL_FW

- OpenGL/ESを使った描画クラスなど
- OpenGL　C++　ラッパー
- フォントの描画、管理
- 画像の管理（モーションオブジェクト）
- ライト（開発中）
- シーングラフ（開発中）

※以前に作ったけど、イマイチなので捨てた

- シェーダー（開発中）

IMG_IO

- 色、画像を扱う基本クラス
- フルカラー、グレースケール、インデックスドカラー、アルファチャンネル
- 描画（ペイント機能）、拡大縮小など
- 画像フォーマットの出し入れ

※png、bmp、jpeg、openjpeg、ｔｇａ、dds

- 他、小物など色々
- 他のライブラリーに移りたい欲求もあるけど、修正範囲が大きいので当面利用する

SND_IO

- 単音、ストリームなどを扱うクラス
- 定位（OpenAL）
- 音楽フォーマットの出し入れ

※mp3、aac、wav

- サンプリング周波数、チャネル、解像度
- 他、小物など色々
- OpenALとの橋渡しなど

WIDGETS

- GUI の部品を扱うクラス
- Widget管理
- ボタン、スライダー、フレーム、チェックボックス、ラジオボタン、etc
- 基本的には２次元オブジェクトの描画で行い、シェーダーなどは使わない方針
- 組み込み機器にも移植が可能な軽量コンパクト設計

※モーションオブジェクトクラス

UTILS

- 頂点、行列など線形代数関係
- ファイルI/Oクラス
- 文字列操作
- ファイル情報、ディレクトリー情報取得関係

※ｂｏｏｓｔを使うべき？

- プリファレンス（XMLは嫌い）
- シーン制御、シーン管理関係
- 他、小物

GLFW３の改造

- ドラッグ・アンド・ドロップ

※Ｗｉｎｄｏｗｓのみ

- FEP関係のメッセージ（予定）

開発環境

- MinGW（MSYS）

※以前はcygwin64を使ってた

- make
- CMake
- emacs
- VisualStudioでは試していないけど動くだろう
- gcc が好き！
- 将来的にはclang、LLVMに移行予定

GUIフレームワークに期待される事

- 日常と非日常な処理
- ありがちな処理は、簡単な設定だけで可能
- 複雑で特殊な処理を実装することが可能
- 拡張が容易
- 予想できるスマートな挙動
- シンプルな構造（※）
- マウス、ジョイステック、タッチパッドなどでも操作が可能（今後の課題）

次世代のGUI？

- マウスでも、タッチ操作でも可能なポリシー
- Windows、OS-X、X11(Qt)の操作はある程度継承
- 最適な操作ポリシーは、アプリケーションによる・・
- 「ユーザーインターフェースガイドラインを守れば何もかも上手く行く」幻想にすぎない
- どうすれば最適なのか解答は無い、ケースバイケース
- CPUもグラフィックスも十分高速なので、もっとヘビーなGUIを考えても良いかもしれない（見た目だけの問題ではなく）

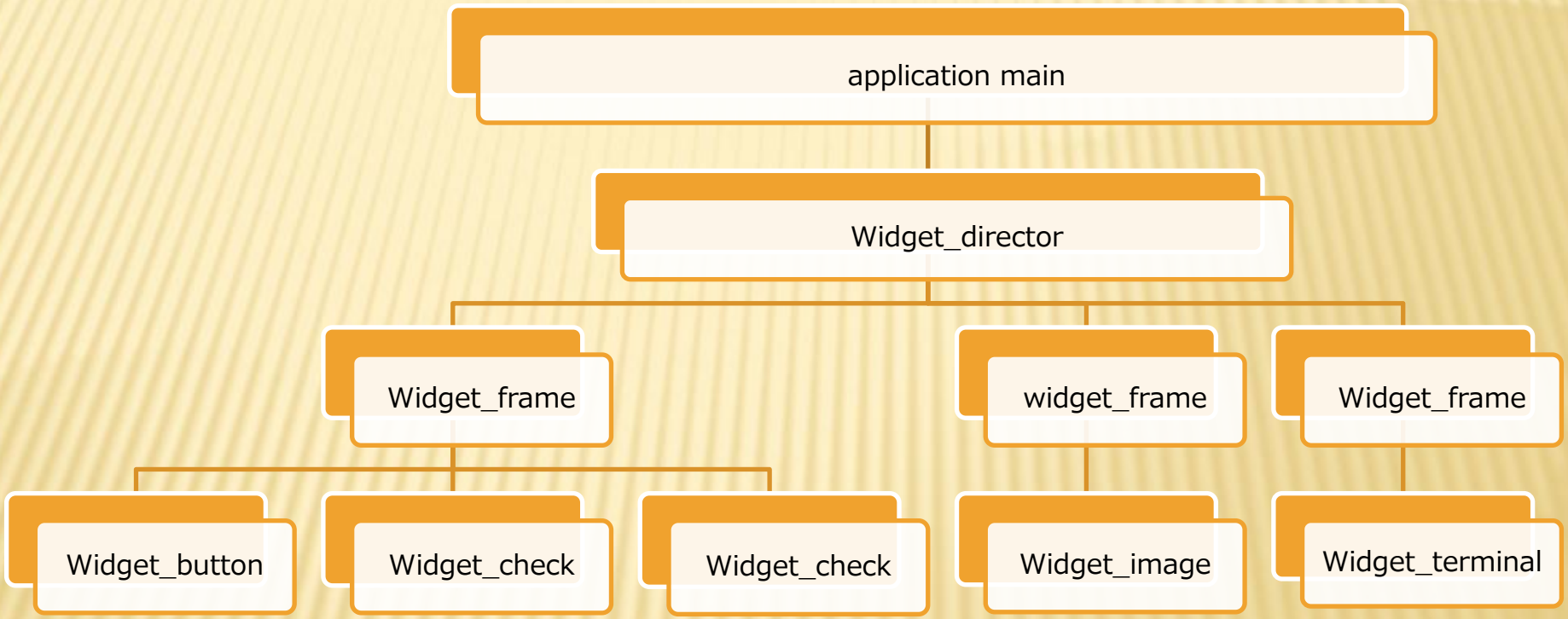
アプリケーションの構造

- メッセージを使わない
- 毎フレーム直列同期処理
- ゲーム向きの構造
- シーン毎の処理
- initialize
- update
- render
- destroy

実際のアプリケーション

- Spinv　スペースインベーダーエミュレーター
- Player　音楽プレイヤー
- Image　画像ビューアー
- Bmc　ビットマップ変換ツール
- Pmdv　PMDビューアー

IMAGE アプリケーション



WIDGET の制御

- 選択
  - 押した瞬間（positive）
  - 押している（level）
  - 離した瞬間（negative）
- 移動
  - 押しながら移動（ドラッグ）
- リサイズ
  - 押しながらリサイズ（右ボタン）

基本制御プロパティ

- 表示・非表示(enable)
- 停止(stall)
- 位置固定
- 水平リサイズ
- 垂直リサイズ
- サイズ固定

実際のソースコード

アプリケーション・コンテキスト

```
namespace app {

    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    /*!
            @brief  アプリケーション・共有リソース
    */
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
    struct core {

            al::sound   sound_;

            gui::widget_director    widget_director_;

            sys::preference         preference_;

            core() { }
    };
}
```

メイン

```
#include "main.hpp"
#include “img_main.hpp"

typedef app::img_main      start_app;

static const char* window_key_ = { "application/window" };
static const char* app_title_ = { "Image" };
static const vtx::spos start_size_(1024, 768);
static const vtx::spos limit_size_(800, 600);
```

```
int main(int argc, char** argv)
{
    gl::create_glcore();

    gl::IGLcore* igl = gl::get_glcore();

    // カレントパスを生成
    std::string tmp;
    utils::convert_delimiter(argv[0], '¥¥', '/', tmp);
    std::string pref;
    utils::get_file_base(tmp, pref);
    pref += ".pre";
    std::string path;
    utils::get_file_path(tmp, path);
```

```
    if(!igl->initialize(path)) {
        return -1;
    }

    utils::director<app::core> director;
    director.at().preference_.load(pref);
    vtx::srect rect(vtx::spos(10, 40), start_size_);
    if(!director.at().preference_.load_rect(window_key_, rect)) {
//      std::cout << "Load rect error..." << std::endl;
    }
    if(!igl->setup(rect, app_title_, false)) {
        return -1;
    }
    igl->set_limit_size(limit_size_);

    director.at().sound_.initialize(16);
    director.at().widget_director_.initialize();
    director.install_scene<start_app>();
```

```
while(!igl->get_exit_signal()) {
	igl->service();

	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
	gl::glColor(img::rgbaf(1.0f));

	glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);
	glEnable(GL_TEXTURE_2D);
	glEnable(GL_BLEND);
	glDisable(GL_DEPTH_TEST);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

	director.render();

	igl->flip_frame();

	director.at().sound_.service();
}
```

```
    // プログラム終了の廃棄
    director.erase_scene();
    director.render();

    {
        vtx::srect rect(igl->get_locate(), igl->get_size());
        director.at().preference_.save_rect(window_key_, rect);
    }

    director.at().preference_.save(pref);

    igl->destroy();

    gl::destroy_glcore();
}
```

アプリケーション・定義

```
namespace app {

    class img_main : public utils::i_scene {

        utils::director<core>&      director_;

    public:
        img_main(utils::director<core>& d) : director_(d)   { }
        virtual ~img_main() { }

        void initialize();

        void update();

        void render();

        void destroy();
    };
}
```

アプリケーション初期化

```
void img_main::initialize()
{
	using namespace gui;
	widget_director& wd = director_.at().widget_director_;

	{ // 画像ファイル表示用フレーム
		widget::param wp(vtx::srect(30, 30, 256, 256));
		widget_frame::param wp_;
		wp_.plate_param_.set_caption(24);
		frame_ = wd.add_widget<widget_frame>(wp, wp_);
	}
	{ // 画像ファイル表示イメージ
		widget::param wp(vtx::srect(0, 0, 256, 256), frame_);
		widget_image::param wp_;
		image_ = wd.add_widget<widget_image>(wp, wp_);
		image_->set_state(widget::state::CLIP_PARENTS);
		image_->set_state(widget::state::RESIZE_ROOT);
		image_->set_state(widget::state::MOVE_ROOT, false);
	}
```

アプリケーション初期化

```
    { // 機能ツールパレット
        widget::param wp(vtx::srect(10, 10, 130, 300));
        widget_frame::param wp_;
        tools_ = wd.add_widget<widget_frame>(wp, wp_);
        tools_->set_state(widget::state::SIZE_LOCK);
    }
    { // ファイラー起動ボタン
        widget::param wp(vtx::srect(10, 10, 100, 40), tools_);
        widget_button::param wp_("file");
        open_ = wd.add_widget<widget_button>(wp, wp_);
    }

    // プリファレンスの取得
    sys::preference& pre = director_.at().preference_;
    if(filer_) {
        frame_->load(pre);
    }
}
```

アプリケーション UPDATE

```
void amg_main::update()
{
        gui::widget_director& wd = director_.at().widget_director_;

        if(open_) {
            if(open_->get_selected()) {
                if(filer_) {
                    bool f = filer_->get_state(gui::widget::state::ENABLE);
                    filer_->enable(!f);
                }
            }
        }

        std::string imfn;
        int id = igl->get_recv_file_id();
        if(dd_id_ != id) {
            dd_id_ = id;
            const utils::strings& ss = igl->get_recv_file_path();
            if(!ss.empty()) {
                imfn = ss.back();
            }
        }
```

アプリケーション UPDATE

```
if(filer_) {
    wd.top_widget(filer_);

    if(filer_id_ != filer_->get_select_file_id()) {
        filer_id_ = filer_->get_select_file_id();
        imfn = filer_->get_file();
    }
}
```

アプリケーション UPDATE

```
if(!imfn.empty()) {
    img::img_files& imf = wd.at_img_files();
    if(!imf.load(imfn)) {
        dialog_->set_text("Can't decode image file:¥n '"
         + filer_->get_file() + "'");
        dialog_->enable();
    } else {
        image_offset_.set(0.0f);
        frame_->at_local_param().text_param_.text_ = imfn;
        mobj_.destroy();
        mobj_.initialize();
        img_handle_ = mobj_.install(imf.get_image_if());
        image_->at_local_param().mobj_ = mobj_;
        image_->at_local_param().mobj_handle_ = img_handle_;
    }
}

wd.update();
```

アプリケーション・レンダリング

```
void img_main::render()
{
	director_.at().widget_director_.service();
	director_.at().widget_director_.render();
}
```

アプリケーション廃棄

```
void img_main::destroy()
{
	sys::preference& pre = director_.at().preference_;
	if(filer_) filer_->save(pre);
	if(frame_) frame_->save(pre);
	if(tools_) tools_->save(pre);
	if(scale_) scale_->save(pre);
}
```

質疑、応答

ありがとうございました