

SSP in a Nutshell #1

SERIKO

<http://ssp.shillest.net/>

C.Ponapalt (ぽな@ぼぐとら)



今回の講演内容

- SSPの実装内容から考察する、ゴースト開発のギモンへの回答
- 実装レベルからの話なので少し難解なところもあるかもしれません
- レベル:3クマー (by わんくま同盟)



いろいろの～がき

- 講演途中でのツッコミ・質問を歓迎します
- SERIKOは正確にはアニメーション定義のみを指しますが、ここではシェル作成に使う全般について解説します
- 一部着せ替え定義はMAYUNAだろ！とか、collisionはどーすんねんとかいうツッコミは勘弁！



疑問1: surfaces.txtの 書き方について

けっこう難解なSERIKO定義、その全貌を徹底
解明！（ウソ）

そもそもこんなややこしいのを手で書くのが間
違いの元なので、できるだけツール類を使って
ください！（例:ころぺた）



合成順の制御は？

SSPでは「書いた順に」合成していきます。

写真屋で言うところのレイヤー1、レイヤー2
＝最初に書いたアニメーション、次に書いた…

element定義やは「背景」にあたるもの

注意：着せ替え定義は実装上の都合で別扱い
です。

ToDo: ID順にソートしたほうがいいかな？



当たり判定の優先順位

アニメーションと一見逆で、書いた順に上から重なっていきます

前で書いた分が後で書いた分を上書き
→実はどちらも単純にループをまわしてるだけ

当たり判定の中に別の当たり判定があるような定義を書く時に便利

ToDo: これもID順にソート?



タイミング制御の怪

ウエイトの値は、描画「前」=表示する時間を
書くわけではありません！

実質的には、1つ後の定義のウエイト値が「表
示時間」になります

なぜこんな仕様？→これでないと最初のアニメ
を描画するまでの待ち時間が設定できない
…単に手前にSleep書いちゃったままひっこみ
がつかなくなっただけかもしれないけど…



descript.txtのこと… 忘れないで下さい

定義はsurfaces.txtだけではありません！

着せ替えメニュー定義や全体のバルーン位置
制御とか、メニュー背景などなどはこっち

特に着せ替え関係のトラブルの相談で見落と
しがちな部分

メニュー定義なんかはかなり難解

→改善の余地あり



それでも まともに動きません！

エラーログ機能の活用

設定-開発/その他-エラーや警告がある場合に通知する、をON

- 警告(黄色いアイコン): SSPではまともに解釈できるけど他を考えて修正してね
- エラー(赤いアイコン): どうにもこうにもならなかったなので捨てました！



疑問2: タイミング制御

アニメーションと喋りを同時に実行していろいろな効果を狙う

歌詞ファイルを読み込んで歌わせる

その他…

どの程度、厳密な時間制御が可能か？



(1) 各スレッドの動作状況

SSPのスレッド構造

- 全ゴーストを統括するためのメインスレッド
- 機能実行用にいくつか
- アニメーションは1キャラクター1スレッド
(通常¥0¥1で2スレッド)
- SakuraScript制御用に1スレッド



(1) 各スレッドの動作状況

- メインスレッド - 優先度 Normal (標準)
 - その他 - 優先度 Below Normal (ちょっと下)
- メインスレッドから制御する構造のため、このように調整 (別に全部 Normal でも影響はない…)
- アニメーションやスクリプト実行は、OS のその他のアプリの影響を受けて遅くなる可能性があります



(2) 各スレッド自体の制御方法による遅れ

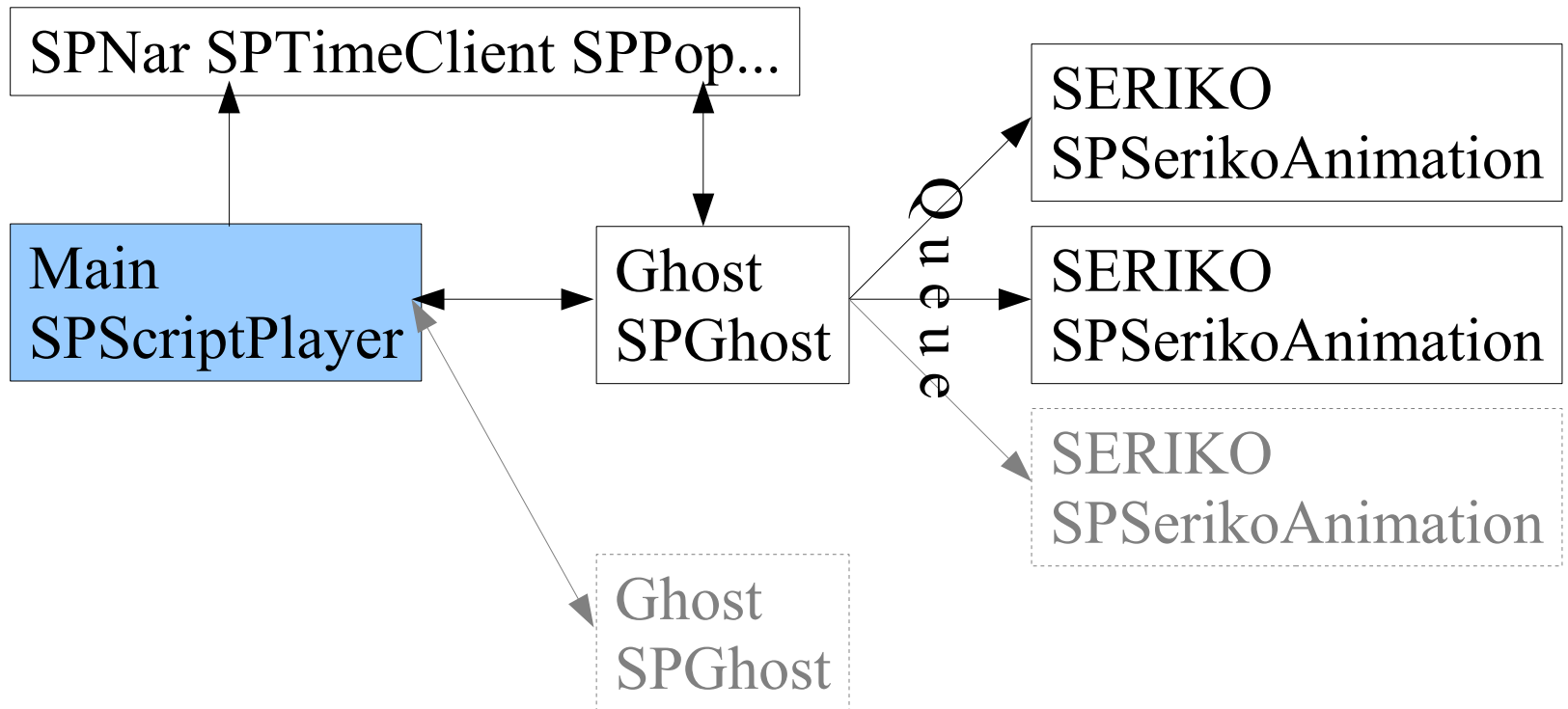
SakuraScript制御用スレッド、アニメーションスレッドは、全てProducer-Consumerパターンで作られています

要するにコマンドをキュー(ToDoリストみたいなもの)につっこんで順番に処理していく方法

実行を指示してから実際に実行されるまで微妙な遅れ



スレッド関係のまとめ



(3) 画像合成による遅れ

毎回全アニメーション部品を合成処理して表示なので、遅くなる原因は以下の条件。

- element合成がやたら多い
- 着せ替え機能がやたら複雑
- 同時に実行しているアニメーションが多い

仕様上合成したものをキャッシュ…とかできませんから！



どの程度影響するの？

(1)(3)については、処理時間も考慮して適当に調整しているのでそれなりに吸収

↑ SSPのアニメーション動作が他に比べて「ちょっとはやい？」と言われる原因

↑ SakuraScriptの解釈も同じ方法のためそれなりに正確な結果が得られる

(2)については、むちゃくちゃに遅いマシン以外はほとんど無視できます



さらに正確に！

¥_w[animation,ID]

で指定したアニメーションが終了するまでの待機が可能

¥i[ID,wait]

でもOK、処理内容に応じてお好きなのを使って下さい



疑問3: どの程度メモリを食うの？

常駐アプリの基本、メモリ消費量の削減
ゴーストにそこまで求める人はいないでしょう
が…

どうやったらメモリを食いつぶす量を節約できるのか？

(SHIORIとか別にして、とりあえずシェルでケチる方法)



もともと少ないです

起動時に全部読み込む

→けっこうメモリ食うんじゃないの？

→正確なアニメーションの実現のため

実は内部でRLE圧縮もどきを利用してできるだけ圧縮した形で保持しています

赤赤赤赤赤赤→「赤6個」

色の変化が少ないシンプルな画像ほど得意
(グラデーションとかは勘弁！)



さらにケチる 涙ぐましい実装…

「抜き色」だけの行(X方向)があった場合、そこは一切メモリ割り当てない(NULL)

リージョンモードの時は α 値を全部つぶして(ゼロ初期化)圧縮されやすく

逆にULWモードの時はリージョンを生成していません→このへんが切替に再起動必要な理由
定義保持用オブジェクトをキャッシュ、まとめて割り当て new□



アニメーションや着せ替え 定義を減らす →あまり効果なし

座標と動作オプション、ウエイト量、その他管理用にちよこつと程度

アニメーション定義がやたら多い程度ではほとんど影響はありません

画像の量による影響がほとんどを占める

いらない画像ファイルは即delete.txtやらに書いて消してしまいましょう！



surface?.png をできるだけ置かない

仕様互換のため、surface1000.pngなどの
surface数字.pngと書かれたものは全て読込
surfaces.txtに書いていなくても全部読んで
elementなど、ファイル名指定できるものは
できるだけsurface?.pngという形式のファイル
名以外を使おう！

逆にアニメーション用の部品などはこの形式で
ないとともに動かない……



256色に制限する

今時256色パレットなんて先祖がえりもアレすぎるのですが…

256色とフルカラー、別の形式で内部保持

256色なら単純計算で食いつぶす量は1/4

実際には透明度(アルファ)も持ってるので、PA(2) / RGBA(4)で1/2程度



elementまみれにしてケチる

element合成はバラバラのまま、
表示する時に合成

表示の共通部分を切り出してelementだらけに
するとかなりケチれます

例：体部分をベースに顔の表情だけを入れ替
える福笑い式？（Emily・the “MobileMaster”）

むちゃくちゃバラバラにすると逆にCPUを食う
ハメになることに注意！



まとめ

というわけで、今回の講演内容を一言で総括してみました。



聞いてくれてありがとうございます！

ろくな講演じゃないので
次はあなたが
しゃべってください！

