3Dブラウザゲームのボトルネック対処法

～次世代ソーシャルゲーム開発に向けて～

■Youta.Hisamichi

株式会社スクウェア・エニックス　オンライン事業部

クリエイティブ・プログラマー

簡単な自己紹介

スクウェア・エニックス・メンバーズ

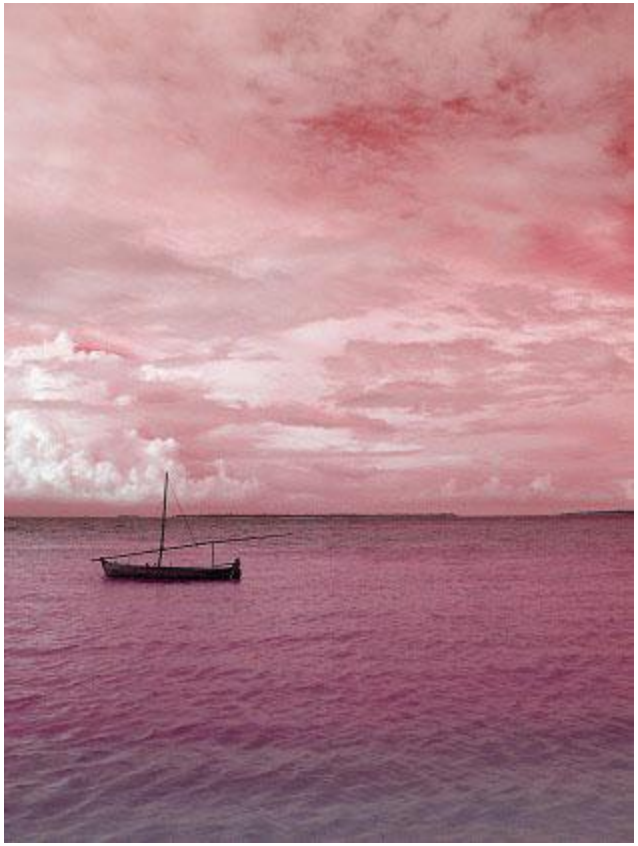
- ○ 3Dアバターエンジン開発
- ○ 3Dアバターゲームライブラリ開発　etc

セッション内に、自分の経験や勘を半ば強引に
文章化したものが含まれることをご了承ください。

ソーシャルゲームの現状

費用対効果の高いゲームだった筈が、、競合の急増やマンネリ化により

そろそろレッドオーシャン・・・！？



1つの可能性

ブラウザ3Dゲーム化

その理由（メリット）

1. 表示能力の向上が望める

→ 表現力と表示領域が格段に増す。これにより既存のブラウザゲームと一線を画すことができ、現状にマンネリ感を感じてきた、ユーザーの次の要望にも答えられうる。

2. 環境の充実

→ PC、ブラウザ、Player（プラグイン）、端末などが高速化。さすがに環境が整ってきた。

…しかし（デメリット）

とにかく重い

→ローディング（＆初期化）

→レンダリング（ふくめ描画まわり）

ブラウザは3D処理に特化している訳ではないので、
なにも考えないで表示すると、それなりのキャラを
1人表示するだけで、CPUファンがうなりだす。

そこで！

3Dブラウザゲームの
ボトルネック対処法

→ 乗り越えられれば、ブラウザゲームを
企画・開発する上で、大きなアドバンテージになる。

インデックス

1. ローディング負荷対策
   →1秒の戦い
2. レンダリング負荷対策
   →1msの戦い
3. 発想によるボトルネック対処法
   →限界を超える戦い
4. ボトルネック・マネジメント
   →真のボトルネックについて。そしてその先について

想定した言語

Flash ActionScript 3.0

ECMAScript(ECMA-262)

ないしはそれに類する言語。根本的に違い過ぎ無ければ、

他の言語にも考え方自体の応用は可能な部分も多い。

あくまで1例

→PCブラウザ、最新のアンドロイド端末など。

ローディング負荷対策

～1秒の戦い～

NowLoadingの重要性

→ 3秒で86%が遅いと感じる。

→ 仮に売り上げ月1億のゲームなら、
8600万円の機会損失にもなる恐れ。

→特にサクッっとやりたいゲームの場合、
1秒の差が致命的に。

3D化によるハードル

1. 種類の増加

2D：画像のみ

3D：ボーン、モデル、テクスチャ、モーション。さらに複数ある場合も。

→処理が膨れ上がる

2. 容量の増加

→種類と数に比例して増大。さらにアクセス回数も増える。

3. 初期化の増加

2D：画像を表示するだけ

3D：ロード後に全XMLのパース、テクスチャやボーンとの紐付け、
レンダリングの開始処理が必要・・・。やること盛りだくさん

特に重いデータは何か

テクスチャ：16KB

ボーン：23KB

モデル：35KB

モーション：250KB

※フレンドを表示する場合は、さらにこれらが人数分必要・・・！

高速データ形式について考える

1. XMLフォーマット

→容量：1MB（ロード：1秒） 初期化：90ms（1倍）

2. JSONフォーマット

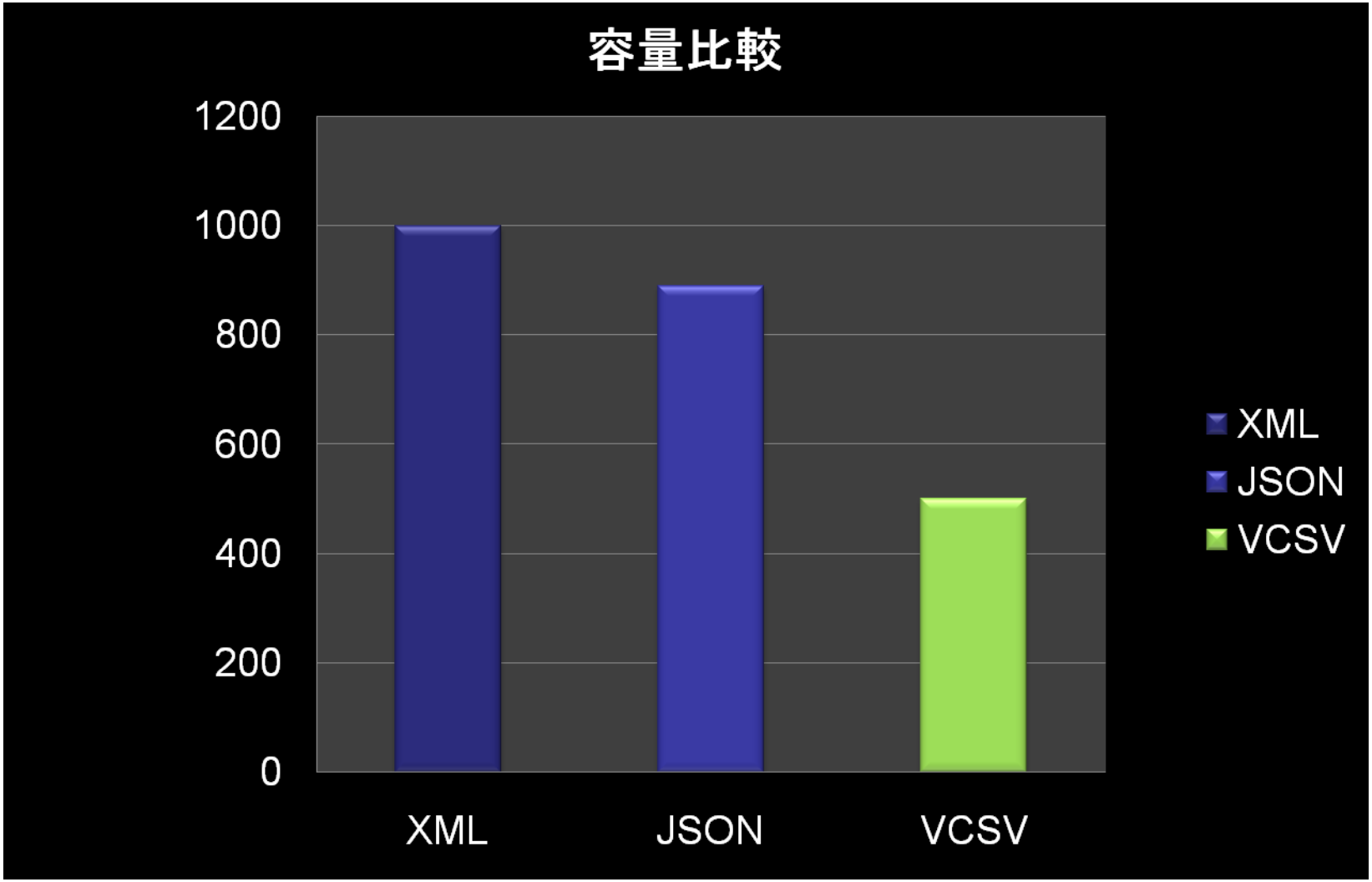
→容量：890KB（ロード：0.9秒） 初期化：580ms（6.4倍）

3. 最適化(VCSV)フォーマット

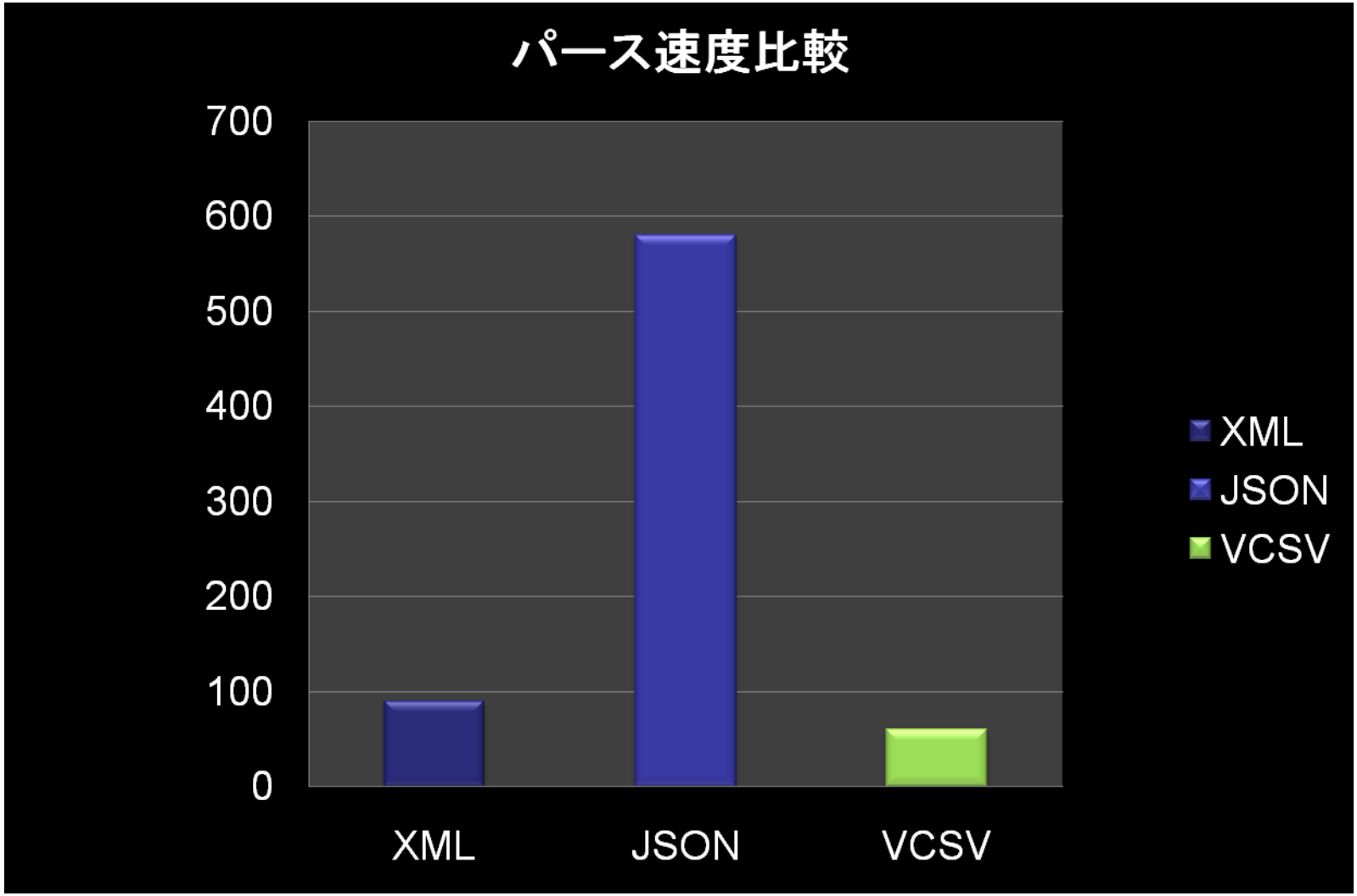
→容量：500KB（ロード：0.5秒） 初期化：60ms（0.6倍）

※ 要素数 3×10,000個のデータで計測

容量比較(グラフ)



パース速度比較(グラフ)


パース速度比較
700
600
500
400
300
200
100
0
XML
JSON
VCSV
XML
JSON
VCSV

高速データ形式について考える2

▽最適化フォーマット（VariableCSV）の例

値と変数を改行とCSVとカンマで区切った値。可読性を保ちつつ、
無駄を極力省いているので、XMLやJSONより高速なパースが望める。

name : Tarou.Satou, sex : m, comment : ABCDEEFG
name : Zirou.Suzuki, sex : f, comment : HIJKLMN
name : Ichirou.Tanaka, sex : m, comment : OPQRSTU

バグ原因になるギリギリまで無駄を省く考え方

さらに高速化しよう

3Dデータ(Collada)のパーサーを最適化する

100ms → 20ms(↑5倍高速化)

※細かいフォーマットまで決めると、さらに高速化することも可能。

さらにさらに高速化

本来デメリットであるロード時間を逆に利用する。

1キャラ500msレスポンス遅延があるとすれば、
10キャラで5000msも時間を生み出すことができる。

※キャラ生成にまったく関係ないが重い処理をここで吸収してしまうことまで可能。

さらにさらにさらに高速化

Flash内にデータを変数として内包する

ロードがゼロ秒になる

500ms → 0ms

※さらにテクスチャとColladaデータをまとめることもでき、
よりいっそう効率的。また、swfに圧縮されるので、
容量削減かつ高速化。

さらに×4高速化

Colladaをあらかじめパースしておく

ロードの次の**パースまでゼロ秒**になる

200ms → 0ms

さらに×5高速化

最終的な初期化が完了される、ギリギリ直前の状態にしておく
配列やクラスに入れて待機させておく。

パースの次の段階まで高速化される

100ms → 30ms

さらに×6高速化

変数の保持状態を工夫する

→例えばstaticで宣言すると初期化が約10倍早くなる。

例：6000行のObject型内包多重入れ子配列
　staticでない場合は初期化に70msかかったが、
　staticの場合は5ms程度だった。

70ms → 5ms

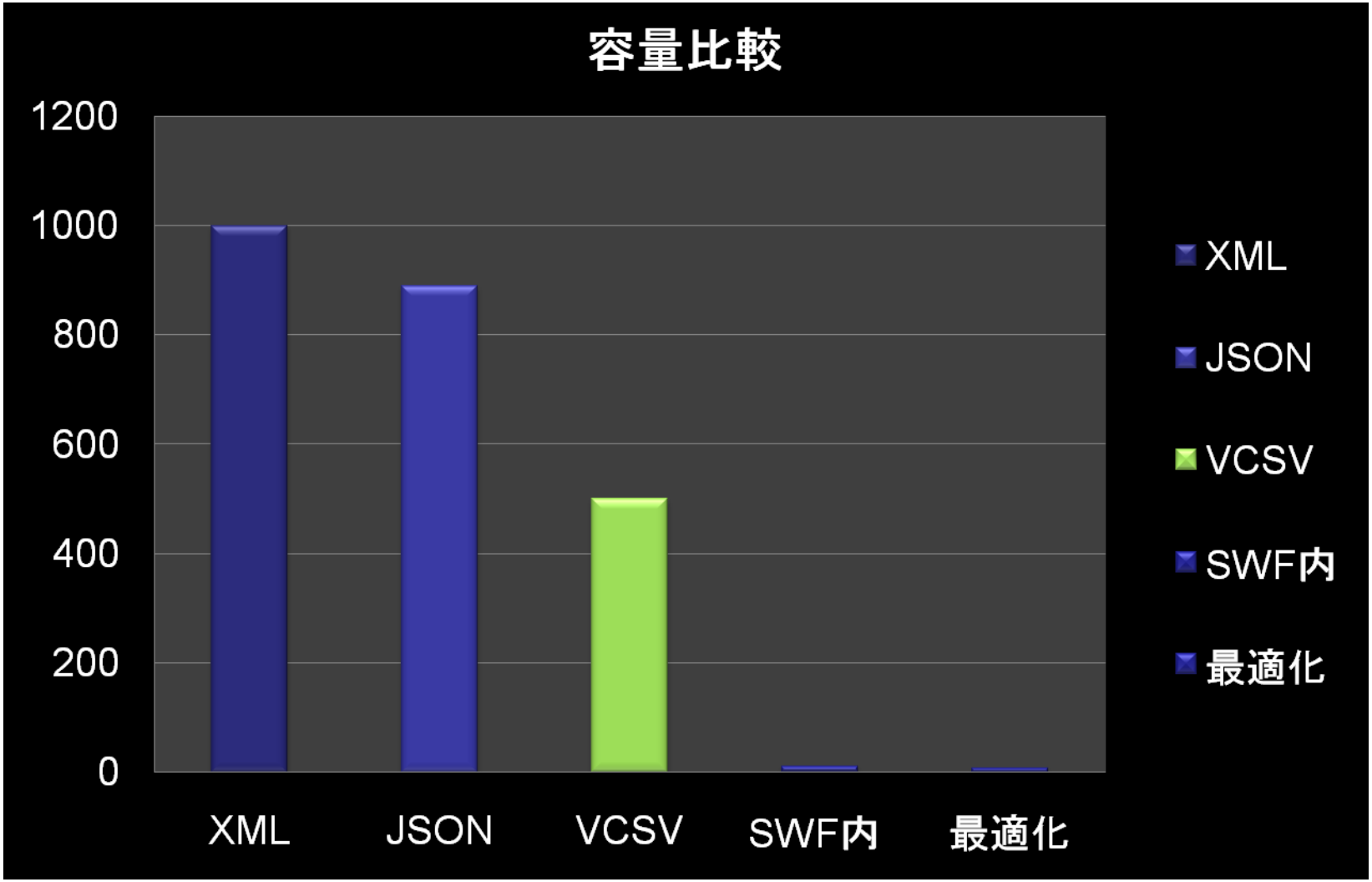
データ容量も軽くなる

速度だけでなく、

容量（≒サーバー負荷≒ロード速度）も軽くなる。

フォーマット状態	容量
XML	1MB
JSON	890KB
変数付きCSV	500KB
FLASH内に保持	10KB
FLASH内で限界まで最適化	7.6 KB 数10倍～100倍効率化

容量比較(グラフ)


容量比較
1200
1000
800
600
400
200
0
XML
JSON
VCSV
SWF内
最適化
XML
JSON
VCSV
SWF内
最適化

データ最適化の注意点

良すぎることには弊害もある。内部に埋め込みすぎると、、

- データの追加が難しくなる。
- データの変更が難しくなる。

- 開発の工数が増える。
- 運用の工数が増える。

その対策

対策案１：外部swf化までにとどめる。

→ロード以外は高速化されつつ、追加変更が可能に。

対策案２：汎用的なもののみ強めに最適化する。

→必ず使うものは内包できる。ボーンや歩行モーションなど。

対策案３：自動生成ツールを使う。

→開発と運用工数の削減

様々な局面に応じて、用いる最適化の組み合わせを工夫する。

初回以降のロードについて

2回目以降のロードは、キャッシュ専用クラスを用いて管理する

Flashさえ閉じられなければ、

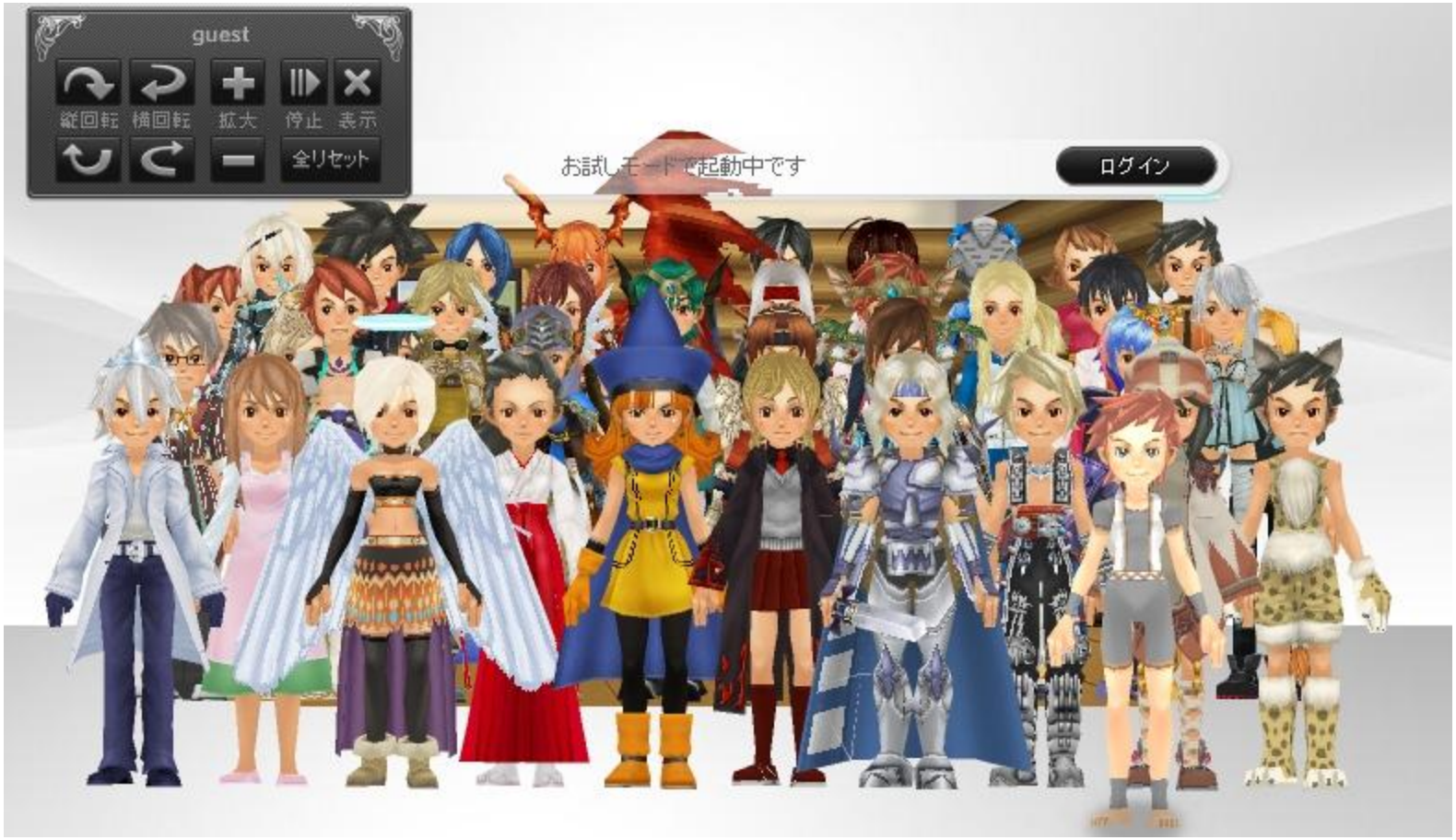
2回目の表示がゼロ秒になる

・いろんなところにバラバラにキャッシュしないで
専用のキャッシュクラスで管理すると、作業や修正が
楽＆再ロード時の場合分けなども一元処理できる

・重複ロード・重複パースが回避される。

実際のキャプチャ

約1000ポリゴンの40人表示で2秒～5秒



※サーバー負荷とPCスペックにも依存します。

レンダリング負荷対策

～2Dにはなかった、描画計算負荷対策～

一番ボトルネックになり易い箇所

毎フレーム内多重ループ内の処理が

最も計算回数が多いところのため、当然ボトルネックになりやすい。

1. ループ外 ・・・負荷 1倍
2. ループ内（100回） ・・・ 100倍負荷
3. 2重ループ内（100×100回） ・・・ 10000倍負荷
4. 2重ループ内の関数内（100×100×12回） ・・・ 12万倍負荷
5. さらに毎フレームある処理（上記に×30～60fps）

毎秒 **360万倍の負荷**

その場所ってどこ？

代表的なものに

1. **ポリゴン**の三角形を描画している個所

→100万ポリゴンなら少なくともその数×3頂点ぶんの情報処理

2. **テクスチャ**を描画している個所

→上記の100万ポリゴンそれぞれに計算しながら貼っている

3. **モーション**を設定している個所

→毎フレーム、ボーンの全頂点を高速再計算

→ 具体的にどんなアプローチがあるか。

配列の.length処理最適化

配列長を制御文の外に記述する。

```
for(var i:uint = 0; i < array.length; i++){
  ↓
var len:uint = array.length;
for(var i:uint = 0; i < len; i++){
```

前述のボトルネックの箇所で使用した場合、

7.5ms → 0.6ms

10倍以上高速化

その他の高速化についての説明

2つの視点からアプローチ

1. あまり触れられることのすくない、意外だった部分を掘り下げる。

2. 単純な1問1答的な解説ではなく、そもそもの考え方としてどんな視点を持てばよいのか。

高速化に役立つ認識1

ビルトインクラスより早い処理について

例：Math.abs()

▽ 100万回テスト

- *Math.abs(n);　　//　63ms*
- *(n < 0) ? -n : n;　//　8.8ms*

実は関数を使わない方が、7倍以上高速

→重要なのは、これ以外にも潜んでいる可能性があるということ。

高速化に役立つ認識2

最新の高速化メソッドより早い処理について1

例：drawPath と lineTo

▽ 10万回テスト

・drawPathで三角形描画 // 85ms

・lineToで三角形描画 // 79ms

ポリゴンの三角形のようなシンプルな描画形状であれば、古いメソッドであるlineToの方が逆に早い。（CPU負荷も 17%→13%）

→最新メソッドでもベンチマークしてから使うべき。

高速化に役立つ認識3

最新の高速化メソッドより早い処理について2

例：Vector型 とArray型

本来Arrayより高速であるはずのVector型が逆に遅くなる場合もある。

▽ 1万回×10回要素追加（push）テスト

- Vector // 26ms
- Array // 20ms

VectorよりArrayが早い。原因はVectorの方が初期化に時間がかかるため、
100回以下程度の少ない配列処理では、Arrayの方が早くなる。

→新しいメソッドは初期化時のコストがむしろ増えていることを疑いつつ使う。

高速化に役立つ認識4

宣言によって速度が変化する可能性について

▽おなじ配列の初期化宣言

a = new Array(); // 3ms

a = []; // 1ms → 3倍高速化

書き方1つで変化する可能性あり。さらに、再初期化の瞬間は高速化のチャンス

a.length = 0; // 0.1ms → 30倍高速化

基本インスタンスは生成せず、すべてリセットして使い回すと高速。
たとえば前述のVectorとArrayの結果も、リセットを用いると逆転する。

Vectorの初期化 26ms → Vectorのリセット 9ms（逆転する）

Arrayの初期化 20ms → Arrayのリセット 9.2ms

高速化に役立つ認識5

ビットシフトより早い処理について

a/=2; // 1秒

a>>1; // 1．2秒

→ ビットシフトは高速化のイメージが強いが、
うかつなビットシフト演算は変化がないどころか、逆に遅くなる
可能性がある。

まとめると

1. **確実に重い**と判明している処理をすべて回避すれば、確実にはやくなる。
（例：配列のlengthループ ）

2. さらにメソッドの**得手不得手**を認識することまでできると、処理速度に差が出てくる。

3. そしてさらに**見識眼（考え方）**が育つと、誰も認識していなかった高速化まで踏み込める。

発想によるボトルネック対処法

計算の高速化にはやがて限界がくる。
技術的な限界点を突破する、発想からのボトルネック解消法

2Dを3Dで大量表示したい（ビルボード処理）

・背景物（木など）を、ポリゴン１枚＋テクスチャで表示する

→ 3次元空間を維持しつつ、アイテムや
オブジェクトを大量に配置できる。

例：1万ポリゴン→1ポリゴン。
どんなデータも1ポリゴン

→ テクスチャを貼りかえるだけなので、
素材の用意、運用、アイテム販売、広告表示などのタイアップ、CGM化がスムーズになる。

3Dを3Dで大量表示したい（ハイブリッド3D）

ビルボードの板ポリゴンに、2Dではなく3Dキャラを描画する
動かしているキャラ以外はビルボード処理。

→ 3Dモデルまで大量に表示できる。
フレンドを大量表示したいソーシャルゲームに適している。

ハイブリッド3Dを、より3Dに（時間差レンダリング）

必要なときだけレンダリング

→キャラ数が増えても負荷が増えない。

・1キャラのCPU負荷10%

・10キャラのCPU負荷100%

・時間差で常に1～2人しか動かさなければ、10人表示しても
CPU負荷10～20%

原則非同期のソーシャルゲームに適している。

設計レベルから時間差レンダリングをもとに考えることもできる。

→ 例：自分のターンしかキャラが動かないゲーム

3D処理をせずに3D処理 （プリレンダリング）

初期化時にレンダリングしてアニメパターンをつくってしまう。
キャッシュした数パターンのループをBitmapDataで高速描画。

その部分は3Dで動かすより、遥かに軽くなる。
また、BitmapData回りは高速化の技法がいろいろあるので工夫し易い。

・3Dの1レンダリング　5ms

・BitmapData（copyPixels）で処理した場合　0.034ms

→150倍高速

根本的な荒技

- 描画の画質の変更
  - 画質高 ・・・ CPU負荷 30%
  - 画質中 ・・・ CPU負荷 20%
  - 画質低 ・・・ CPU負荷 10%
- フレームレートの変更
  - 60fps ・・・ CPU負荷 28%
  - 30fps ・・・ CPU負荷 30%
  - 15fps ・・・ CPU負荷 15%
  - →ただしもともとエンターフレーム処理をしていなければ、効果は少ない。
- 解像度の変更
- 処理を毎フレームおこなわない（あたり判定など）
  - →その処理のCPU負荷が半減する。

根本的な荒技の補足

賢いフレームレート調整について

例：onEnterFrameよりもTimerクラスの方が負荷が少なく、
それでいて見た目のレベルも保たれる。そんな便利な手法を模索しておく。
→37%も3D処理時のCPU負荷が軽減

賢い画質の設定について

→画質についても必要な時だけ上げるようにする。
▽高画質時：
ソーシャルゲームでよく見かける写真撮影機能は、
シャッターを切る瞬間のみ高画質にする。
▽低画質時：
高速移動中
ただし切り替えた瞬間のギャップが一番目立つため、頻繁に切り替えないようにする。

まとめると

1. ビルボード ＋
2. ハイブリット3D ＋
3. 時間差レンダリング ＋
4. プリレンダリング

（５．根本的な荒技）

→ソーシャルゲームの人数表示量に耐えうる、
　軽くてリッチなゲームの実現。

注意点

1. ビルボード処理

→カメラの上下や、回りこみが大きいと違和感の可能性。

2. ハイブリット3D処理

3. 時間差レンダリング処理

→うっかり一斉に動いて負荷が暴走しないよう、
排他制御をきちんと管理する。

4. 2D側からのアプローチ

→CPUには優しいが、メモリを食うので注意。

ボトルネック・マネジメント

～対症療法の1歩先から未来へ～

ボトルネック・マネジメント　抑えたいポイント①

▽開発前（常日頃）

1. あらゆるボトルネックの網羅

→ ≒高速化技術の蓄積

▽開発中（テコ入れ中）

2. ボトルネックの把握と対応

→ 対応によって発生するリスクも

3. 対応状況の情報共有

→ どこでどう使われたか。修正する際の注意点も。

ボトルネック・マネジメント　抑えたいポイント②

▽開発後（常日頃2）

4. 新たなボトルネックの監視

→ バージョンや端末で変化しかねないため。

5. 切り替え可能な状態

→ 仕様やバージョンによって最適な技術に変更可能な状態。

そして更なる開発ステージへ。

最大のボトルネックはどこに潜むか。

ソーシャルゲームやブラウザゲームの開発には、元コンシューマーのゲーム開発者もいれば、それとは対照的にWebデザイナーからやってきた人が開発に携わる場合がある（専門でない部分を触る可能性が大きい）。

その場合はそもそもプログラム的なボトルネック対応的な発想が存在しない場合が多い。またネイティブな技術者ではないので学習コストを払って判ったところで対応が難しい。

逆にコンソール側の人はWebサイドのボトルネックは判らない。 すなわち**人と、その構成自体が最大のボトルネック要因**といえる。

→人的ボトルネック

どうすればよいか

限界までなるべく何もさせない

開発者の技術格差を埋める必要がある。
そのために技術者は、「技術格差を埋める技術の提供」
を行う必要がある。

その際の、今までと違う部分

ただしこれは昔からあるクラスにして変数を隠蔽化ぐらいなレベルの話ではなく、可能ならプログラム自体触れないようにする。書いても1行。

Flashであれば、スクリプトやクラスをみんなでいじりまわさず、パブリッシュ（コンパイル）済みのエンジンとなるswfを1つロードすればほとんど済む状態ぐらいが理想。

究極的には、コンフィグファイル化や管理画面化が理想

それでも発生するボトルネックへの対応

たとえば3Dプログラムと全然関係ない外部swfのアイテムやエフェクトが重かったりする。いくら3D処理が軽くなっても、エフェクトが重くては当然全体的に重くなる。

▽対策案：

見た目だけでなく、CPU負荷基準を設けてQAを行う、あらかじめ基準となるドキュメントを共有した後、専用のベンチマークページで検証するようにする、など。

その先で、得られるもの

その先で、得られるもの　①

1．武器

爆発的な成長を続けるソーシャルゲーム業界で戦うための、
ブラウザ3Dという、

大きな武器が得られる

2. 魂の継承

コンシューマーが3D化した頃の、熱い時期の追体験

→ 実は15年前にも歩んできた伝説の道。

ブラウザにもそんな時代がやってきた。それは、

高い技術への挑戦と工夫の道

不可能を可能にする道

例えるなら、RPGで勇者が父親の足取りを追って旅立つような、
そんな貴重な体験。

その先で、得られるもの　③

3．業界を次のステージへ

ブラウザ3Dにより、Webとゲームの垣根がなくなり
さらにはその架け橋となって、

ゲーム業界を
次のステージに牽引できる

→ ひょっとすると最後のページのメンバーズのアバターの中には、もうすでに
あなたの見覚えのあるゲームキャラクターが、いるかもしれません。

Thank you for your listening!!

ご清聴ありがとうございます！

http://member.square-enix.com/jp/avatar/