# 1/60秒でRuby

## Rubyでゲームを作ったら

kumaryu (Ryuichi Sakamoto)

# 自己紹介

- kumaryu (本名: 坂本龍一)
- ただのRuby好き
- 職業もプログラマ
- プログラミング配信とかやってます

# 概要

1. Rubyでゲームを作る
2. 作ってみた
3. 最適化をする
4. ゲームを面白くする
5. 配布
6. まとめと今後

# 1. Rubyでゲームを作る

# Rubyでゲームを作る

ここでは特にビデオゲーム

- ゲームロジック
  - 入力応答
  - 衝突検出/応答
  - ゲームAI
- 音声処理
- 描画処理

リアルタイムシステム

これで1フレーム
毎秒60回行う
(16ms/フレーム)

# Rubyでゲームを作る

ゲームでスクリプトが使われている
Lua、Scheme、Unreal Script...

- 柔軟性
- 変更の容易さ
- 試行錯誤の速さ

でもやっぱり...

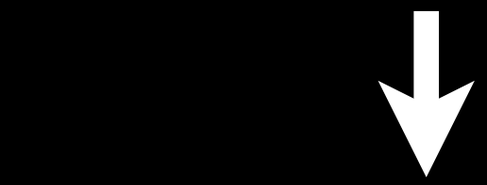
# Ruby だろ？

**作ってみた！**

# Rubyでゲームを作る

ただし

- ノウハウの蓄積が少ない



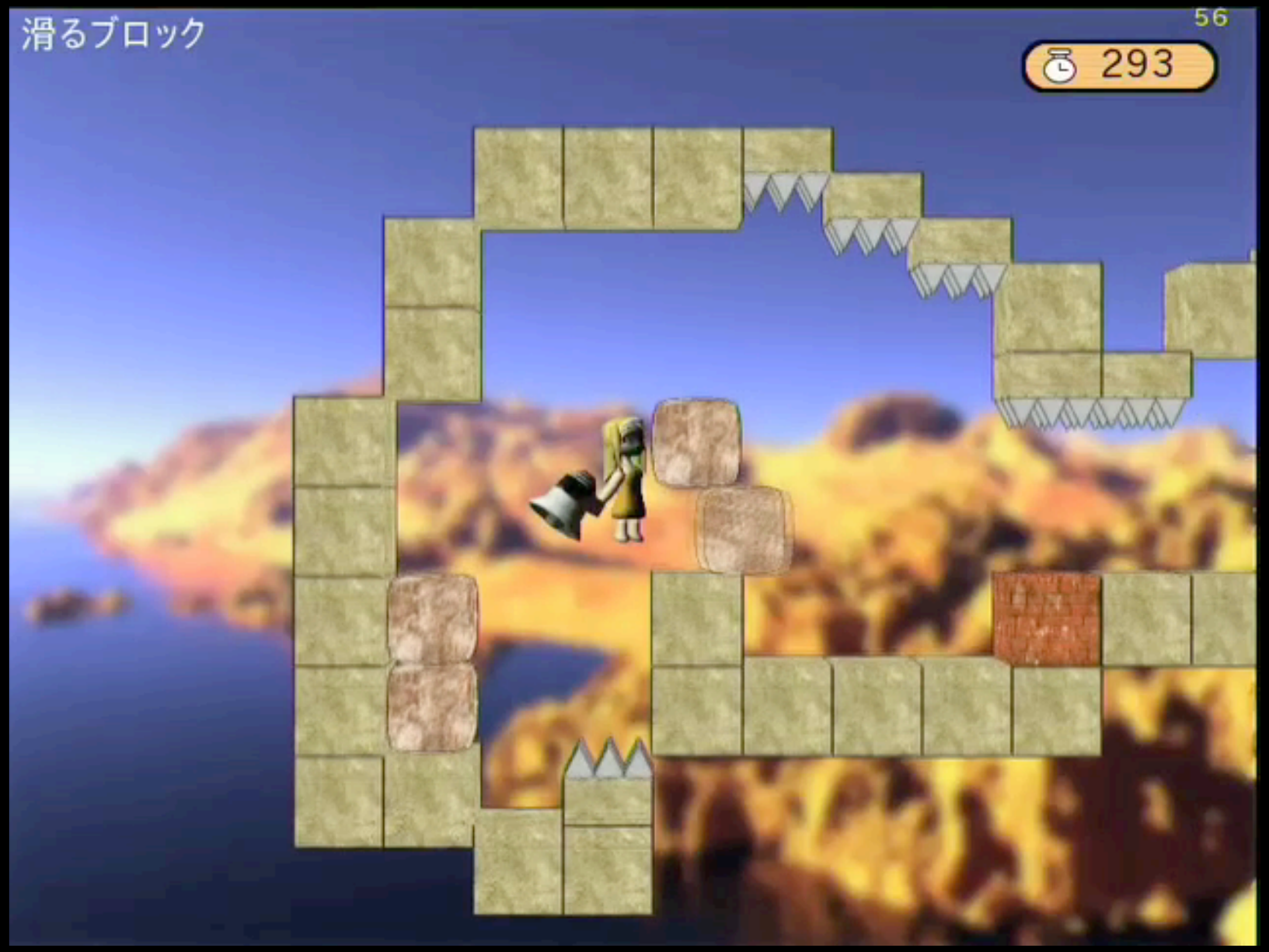
この発表が情報蓄積の助けになるといいなぁ

# 2. 作ってみた

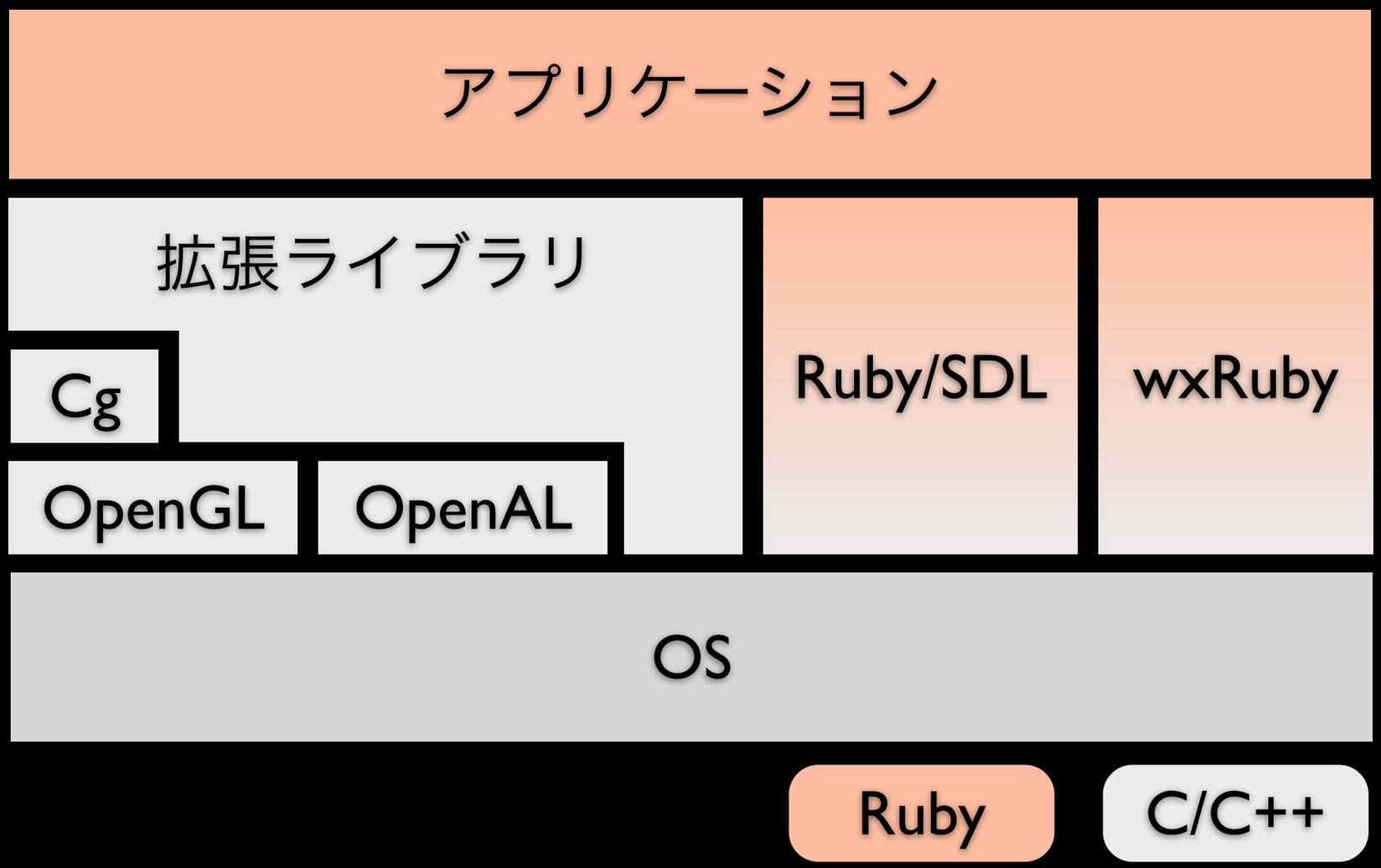
# 作ってみた

- 横スクロールアクションパズル
- 3Dなのは描画だけ
  - 時間・ネタ・経験不足
- あまり厳しい操作は要求しないゲーム

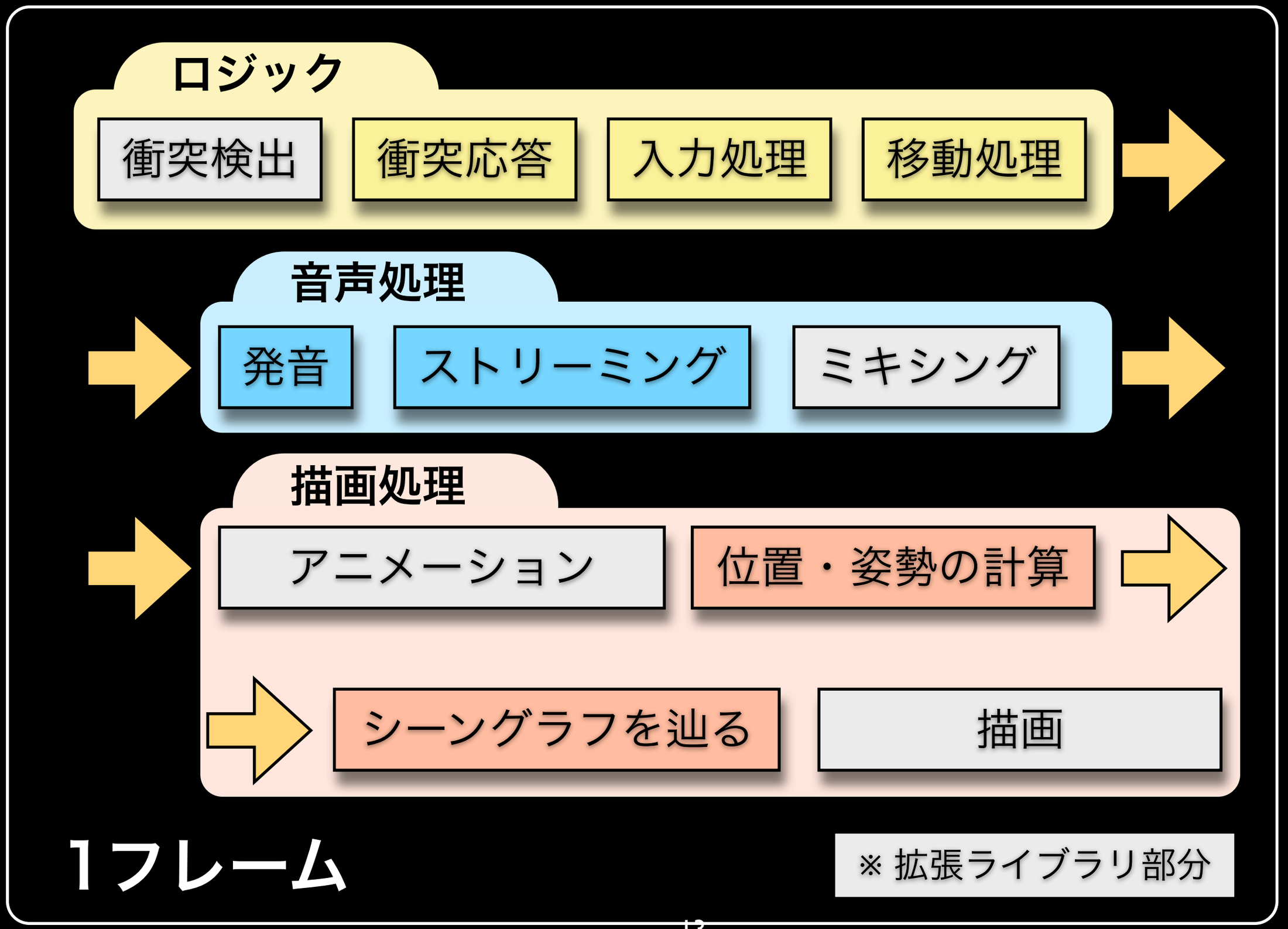
# 作ってみた
# デモ



# 作ってみた構成図

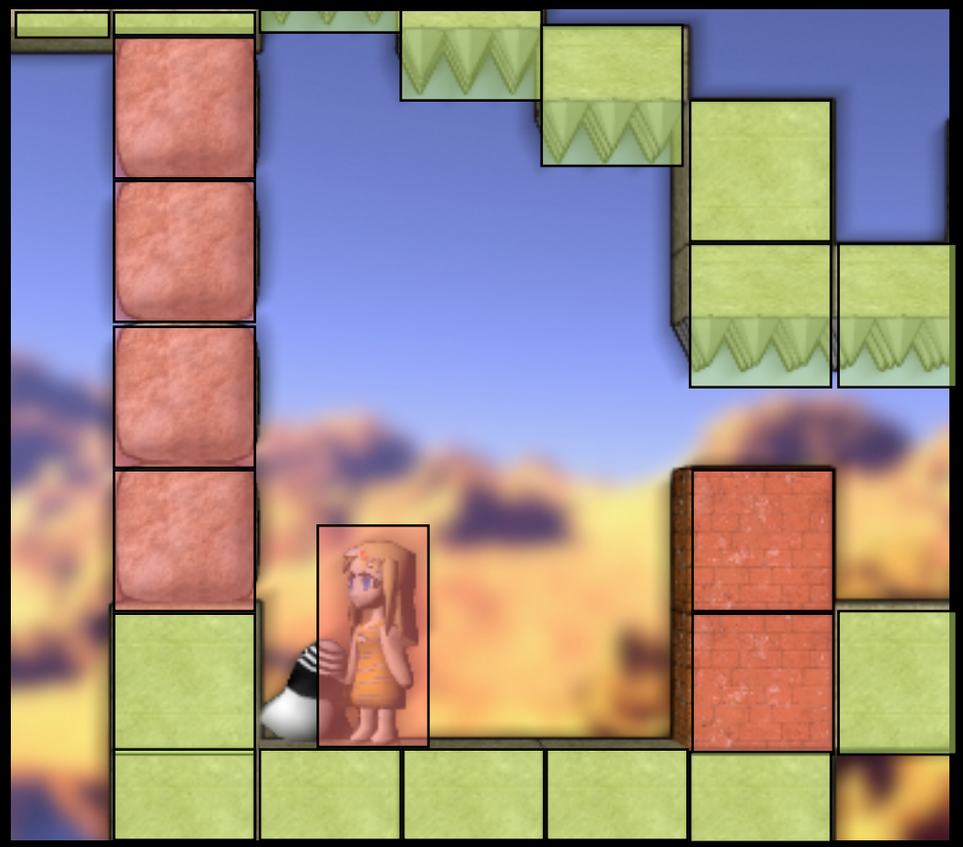

アプリケーション
拡張ライブラリ
Cg
OpenGL
OpenAL
Ruby/SDL
wxRuby
OS
Ruby
C/C++

# 処理の流れ


ロジック
衝突検出
衝突応答
入力処理
移動処理
音声処理
発音
ストリーミング
ミキシング
描画処理
アニメーション
位置・姿勢の計算
シーングラフを辿る
描画
1フレーム
※ 拡張ライブラリ部分

# ゲームロジック
## 衝突検出

- リアルタイムゲームの大事なところ
- 2DのBOX
- 拡張ライブラリで実装



静止物

動く物

※判定位置はイメージです

# ゲームロジック
# 衝突応答

- 衝突に対する結果を処理する
- ブロックが壊れるとか
- 壁に当たったら反発力を受ける
- 移動の実際の解決はあとで

# ゲームロジック
# 入力処理

- ゲームパッドやキーボードからデータを取得する
- 入力に応じて移動ベクトルやキャラクターの状態を設定する

# ゲームロジック
## 移動処理

- 移動や反発力による移動ベクトルを合計して実際に位置を変更する
- 重力とか入れると非常に面倒
- ブロックの隙間につっかかったり
- すり抜けたり
- 適当にごまかした

# 音声処理

- OpenALにつっこむだけ
- Ogg/Vorbis libvorbisfile
- SoundEffect
  - OpenALのバッファに展開しておく
- BGM
  - ストリーミング
  - キューが空いたらバッファに必要分展開して突っ込む

# 描画処理

- ゲームロジックでシーングラフを構成
- Screenから辿ってrenderメソッドを呼ぶ
- 描画とロジックを切り離す

# 描画処理
## 実際の描画

- 頂点データなどを前もってファイルから読み込んでおく
- 位置や姿勢を計算
- OpenGLでGPUに流し込む

# 描画処理

## GPU処理

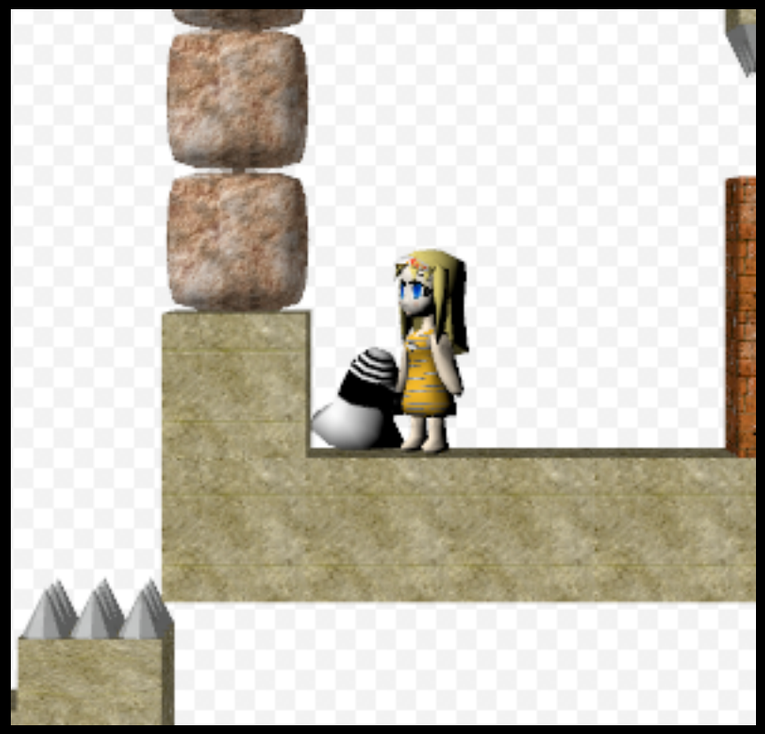
- シェーダ → GPUで動くプログラム
- 頂点毎の変形、座標系変換、透視変換
- ピクセル毎の色計算(陰影付け等)
- シェーダ言語(外部DSL)で書く
  - GLSLとCg
- リソースと速度の制限により汎用言語をGPUで動かすのは難しい

  ➡ Rubyでは書けない
- 実行時コンパイルなので相性はいい

# 描画処理
## アニメーション

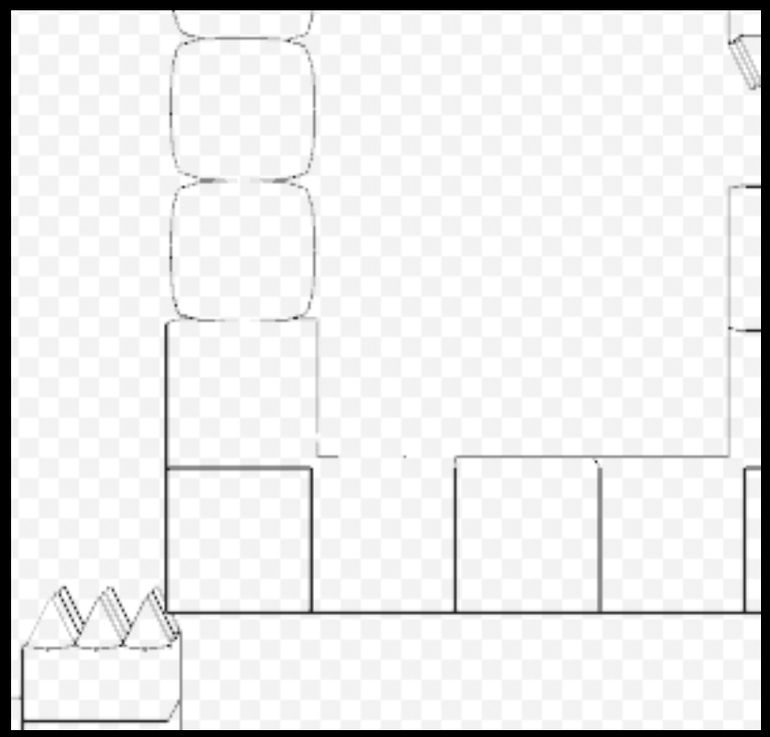

- 頂点単位で行なうので処理量多い
- 拡張ライブラリで実装
- 本当はGPUでやりたい

# 描画処理
## ポストエフェクト

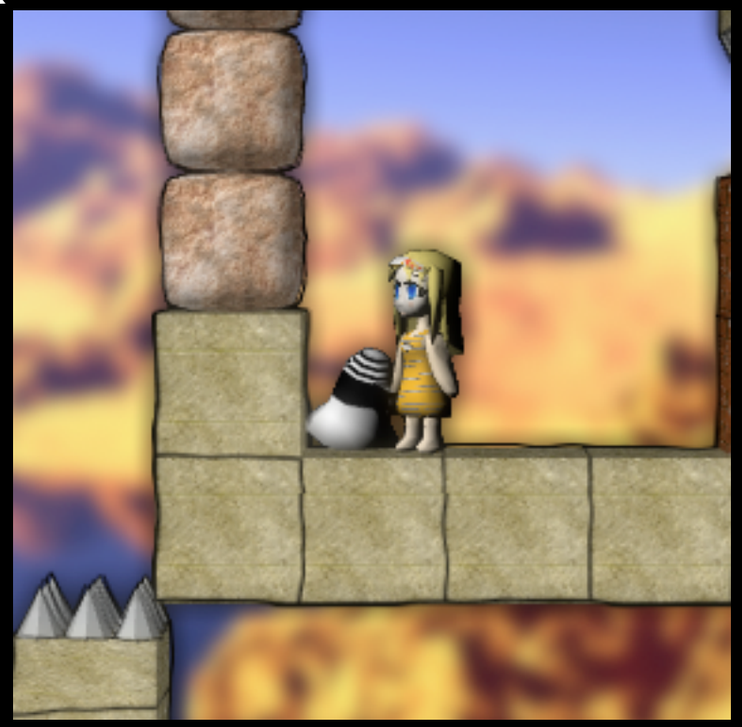


縮小・暈かし

合成



歪ませて合成

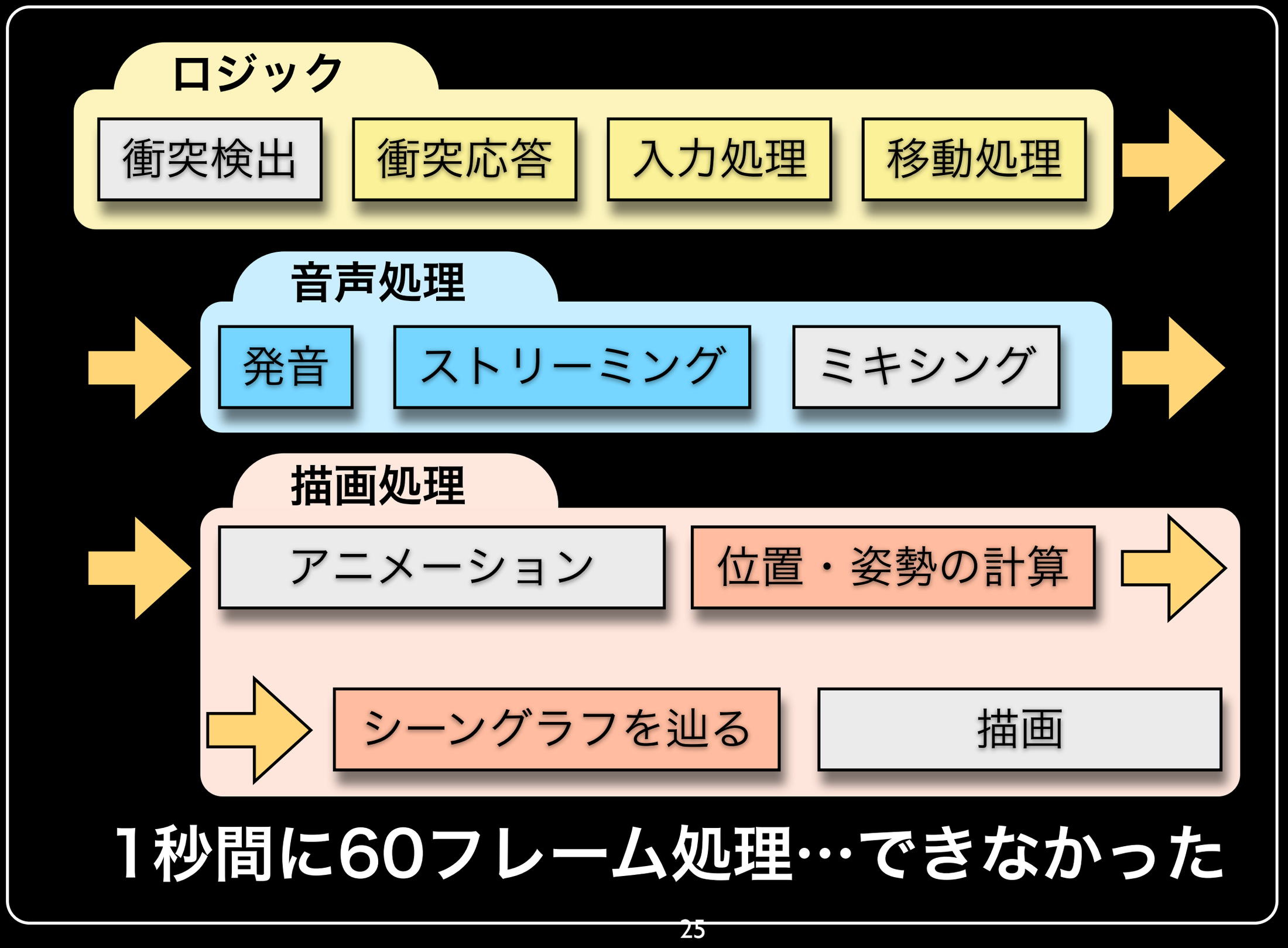


# その他の処理

フレームスキップ

- 16ms突破してたら次のフレームの描画処理を飛ばす
- 滑らかさを犠牲にゲームスピード維持
- ロジックと描画の分離が有利

# 処理の流れ




１秒間に60フレーム処理…できなかった

# 3. 最適化する

# 最適化する

「最適化」しよう

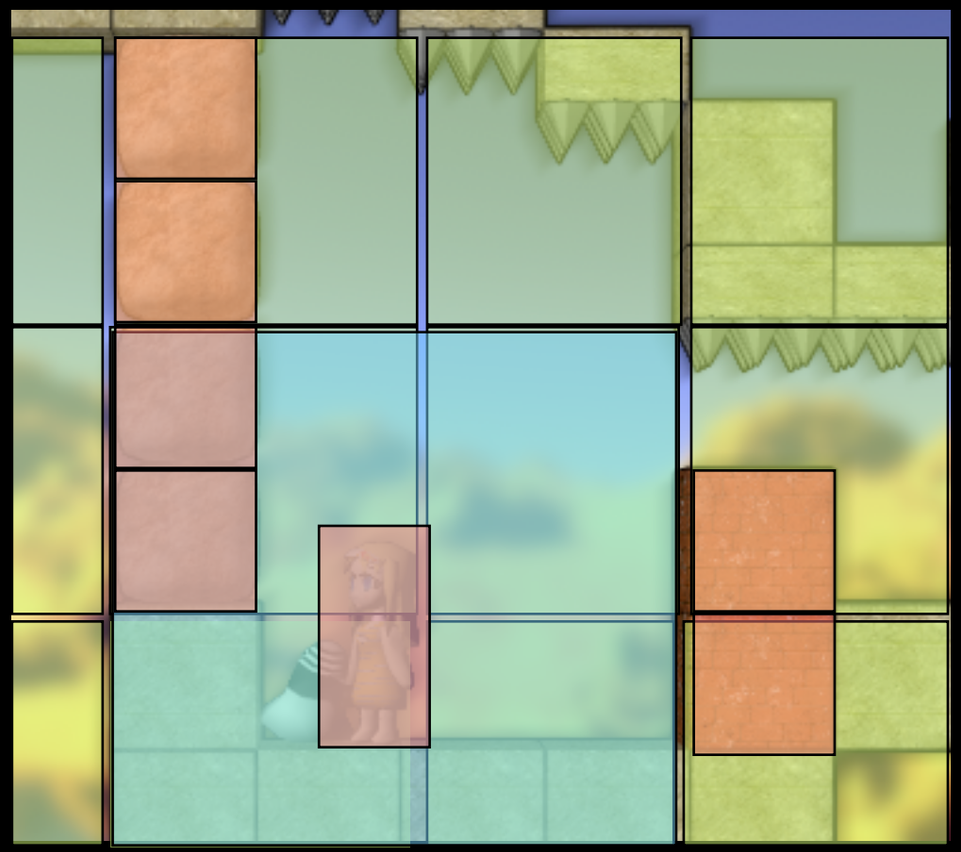
- 早すぎる最適化をしない
- アルゴリズム、データ構造を見直す
- データを取って客観的に見る
  - プロファイル
  - ベンチマーク
  - GCプロファイル

# 最適化する
# 早すぎる最適化

- むやみに大量のブロックを置いたらやはり重い
- Cで書いてもちょっと速くなっただけ
  ➡ Cで書く労力に見合わなかった
- そんなに大量のブロック置かないし
- アルゴリズムの改良を先に考える

# 最適化する
# アルゴリズムの見直し

- 必要以上に遠くのオブジェクトとも判定してしまう問題
- マップを四分木で分割
  近くの領域とだけ判定
- オブジェクトが移動したらマップも更新



# 最適化する
# プロファイルを取る

- ボトルネックが調べられる
- 手動プレイすると安定したデータが取れないのでゲームリプレイ等を流す
- 標準のprofilerではとても重すぎる
- ruby-profを使った
  - ただし1.9.1ではプロファイルが取りづらい

# 最適化する
# プロファイル結果

描画だけで
全体の87%

| %Total | %Self | Total | Self | Wait | Child | Calls | Name |
|---|---|---|---|---|---|---|---|
| **100.00%** | **0.03%** | **762579.09** | **223.13** | **0.00** | **762355.96** | **1** | **GRiko::Game#main** |
| | | 663542.62 | 31.31 | 0.00 | 663511.31 | 1357/1357 | <Class::GRiko::Screen>#render |
| | | 64943.13 | 88.79 | 0.00 | 64854.34 | 1357/1357 | GRiko::Game#preprocess |
| | | 32774.71 | 72.60 | 0.00 | 32702.11 | 1357/1358 | <Class::GRiko::Scene>#update |
| | | 913.36 | 34.08 | 0.00 | 879.28 | 1357/1357 | GRiko::Game#postrender |
| | | 79.55 | 30.84 | 0.00 | 48.71 | 1357/1357 | <Class::GRiko::Sound>#update |
| | | 46.95 | 31.32 | 0.00 | 15.63 | 1357/1357 | <Class::GRiko::Screen>#quit? |
| | | 18.44 | 18.44 | 0.00 | 0.00 | 1357/17992 | Proc#call |
| | | 11.87 | 11.87 | 0.00 | 0.00 | 1357/1357 | GRiko::Game#postprocess |
| | | 9.86 | 9.86 | 0.00 | 0.00 | 1357/1357 | Riko::FrameBasedTime#delayed |
| | | 9.18 | 9.18 | 0.00 | 0.00 | 1357/1357 | Riko::FrameBasedTime#frame |
| | | 6.29 | 6.29 | 0.00 | 0.00 | 1357/1357 | GRiko::Game#prerender |

描画処理は単純に処理量が多い
ゲームロジックはRubyで余裕

# 最適化する
# ベンチマークを取る

- 速くなったかどうかを比較する
- 全力でリプレイを流して終了までの時間を取る
- 描画待ち、時間待ちは外す

# 最適化する
# ベンチマーク結果

- 改良したデータは取れなかったので
- 1.8.7-p173 vs 1.9.1-p129

| 1.8.7-p173 | user(s) | system(s) | real(s) |
|---|---|---|---|
| | 31.37 | 1.34 | 37.87 |
| | 32.31 | 1.28 | 39.16 |
| | 32.73 | 1.17 | 38.98 |
| avg. | 32.14 | 1.26 | 38.67 |

| 1.9.1-p129 | user(s) | system(s) | real(s) |
|---|---|---|---|
| | 21.69 | 1.11 | 27.57 |
| | 22.39 | 1.0 | 28.39 |
| | 21.58 | 1.12 | 28.15 |
| avg. | 21.89 | 1.08 | 28.04 |

**約3割減！**

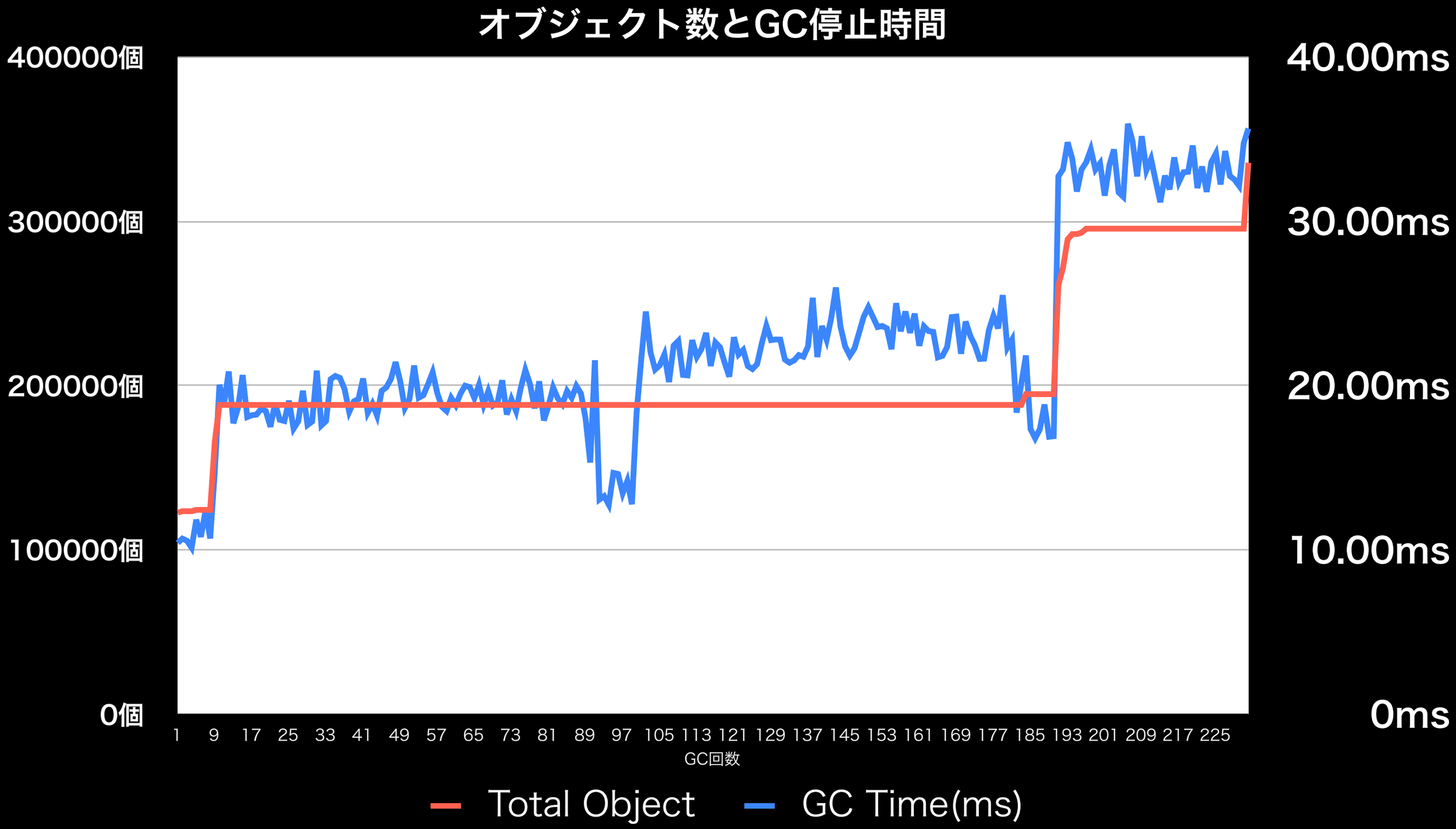
**※当社比**

# 最適化する
## GCの時間を計測

- GCの停止時間が痛そう
- プロファイルには出てこない
- 1.9 新機能 GC::Profilerを使う

| Index | Invoke Time(sec) | Use Size(byte) | Total Size(byte) | Total Object | GC Time(ms) |
|---|---|---|---|---|---|
| 1 | 0.778 | 1950300 | 2457600 | 122850 | 10.44 |
| 2 | 0.810 | 1951680 | 2473984 | 123669 | 10.70 |
| 3 | 0.838 | 1951720 | 2473984 | 123669 | 10.57 |
| 4 | 0.871 | 1953040 | 2473984 | 123669 | 10.14 |
| 5 | 0.904 | 1957120 | 2490368 | 124488 | 11.85 |
| 6 | 0.940 | 1961220 | 2490368 | 124488 | 10.81 |
| 7 | 0.972 | 1962580 | 2490368 | 124488 | 12.44 |
| 8 | 1.006 | 1963920 | 2490368 | 124488 | 10.71 |
| 9 | 1.526 | 1995940 | 3325952 | 166257 | 14.77 |
| 10 | 1.793 | 2001500 | 3768320 | 188370 | 20.07 |

# 最適化する


オブジェクト数とGC停止時間
400000個
300000個
200000個
100000個
0個
40.00ms
30.00ms
20.00ms
10.00ms
0ms
1 9 17 25 33 41 49 57 65 73 81 89 97 105 113 121 129 137 145 153 161 169 177 185 193 201 209 217 225
GC回数
Total Object
GC Time(ms)

# 最適化する
# GCの時間を計測

- 30ms以上も止まってる
- オブジェクトが増えると停止時間も増える
- メモリリークしてる気がする
- メモリリークを簡単に調べる方法があるといいんだけど
- GCがあってもメモリリークに気をつけよう
- 停止時間の短いGCが欲しい

# 最適化する
# まとめ

- 今回の結果
  - 描画が重い
  - 1.9が速い
  - オブジェクト数に気をつける
- 一般的に成り立つとは限らない
- データを取って客観的に見よう

# 4. ゲームを面白くする

# 問題点

そんなことより

ゲームが面白くない

# ゲームを面白くする

- 試行錯誤する
- バランス調整をする

➡ 開発サイクルの短縮が必須！

# ゲームを面白くする
# ツールを作る

- 直感的に調整したい！
- ステージエディタ
  - GUI: wxRuby
  - ゲームのコードをそのまま流用
  - MarshalやYAMLで簡単シリアライズ

# ゲームを面白くする
# ツールデモ

# ゲームを面白くする
# 動的読み込み

- 試行錯誤をリアルタイムにやりたい
  - 試す
  - 書き換える
  - 読み込み
  - 試す…
- Rubyのコードとデータ両方やりたい
- 今回はやってない

# ゲームを面白くする テストプレイ

- テストプレイしてもらう
  - 現状うまくできてない
  - テストプレイ結果を自動で収集できるといいかも
- 今後の課題

# 5. 配布

# 配布

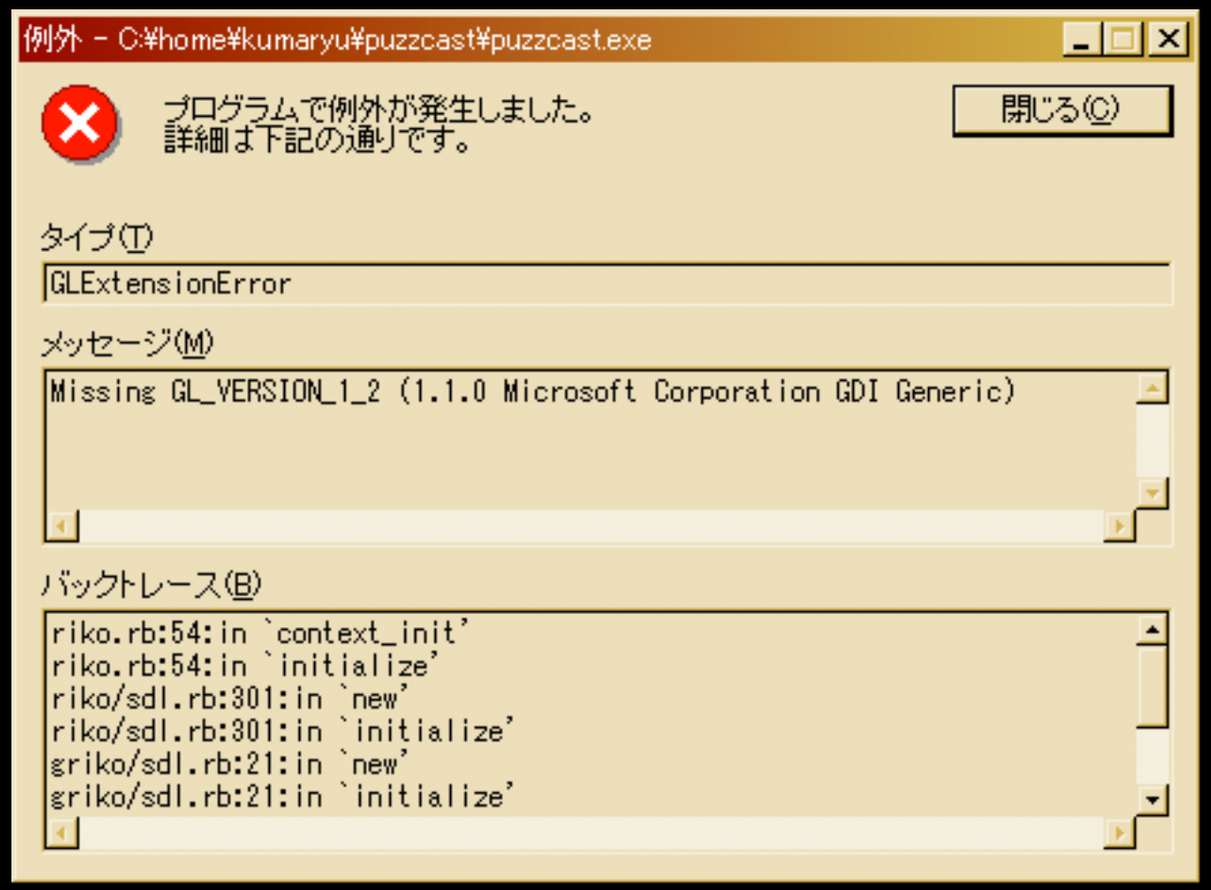
- Windows用とMac用を配布
- 設定用アプリ
  - wxRubyとRuby/SDLで
- WindowsはExerbでexe化
- Macは適当にappにまとめた

# 配布
## 情報の収集

- Exerbの例外ウィンドウが便利だった
- エラーメッセージ、バックトレース含めてコピペで報告してもらえる
- 本当は親切なエラーメッセージも出せればいいんだけど
- 環境の情報も集めたい
  - GPU、OS、ドライババージョンなど

# 配布
## 情報の収集

- 例外ウィンドウ例(Exerb)



# 5. まとめと今後

# 苦労した所

- ゲームを面白くするのが大変
- 素材が作るのが大変
- ハードウェア毎の動作確認
- やっぱり60フレームでない

# Rubyでゲームを作る

Rubyにゲームを作れる能力は十分ある

ノウハウをためていこう！

# 今後

- まだまだ足りない
  - 実装したい機能
  - 素材
  - 経験が足りない
  - 沢山ゲームを作りたい

# おしまい

- 質問(聞くな)
- バイナリ配布
  - http://arekuma.s300.xrea.com/puzzcast/index.html
- ソースはこちら
  - http://miru.kumaryu.net/head/net.kumaryu.puzzcast
  - http://miru.kumaryu.net/head/net.kumaryu.reriko