GLSL を使った
クリエイティブコーディング入門

@MachiaWorx/ _machia_

自己紹介

- 趣味でゲーム・曲・メガデモ・シンセサイザー等色々作ってます。
- 2018 年当時の書き方なので色々直したいのですがこのまま公開しておきます。
- 趣味 100% の内容ですが、何か得るものがあれば幸いです

事前説明

- クリエイティブコーディング
  アート + 技術による創作の世界で使われる言葉
  今回はグラフィック分野にフォーカスを当てて説明します
- GLSL とは、OpenGL （という名前のグラフィック API ）におけるシェーダ言語
  シェーダとは、3D オブジェクトに対し最終的に出力される色を定義する機能のこと

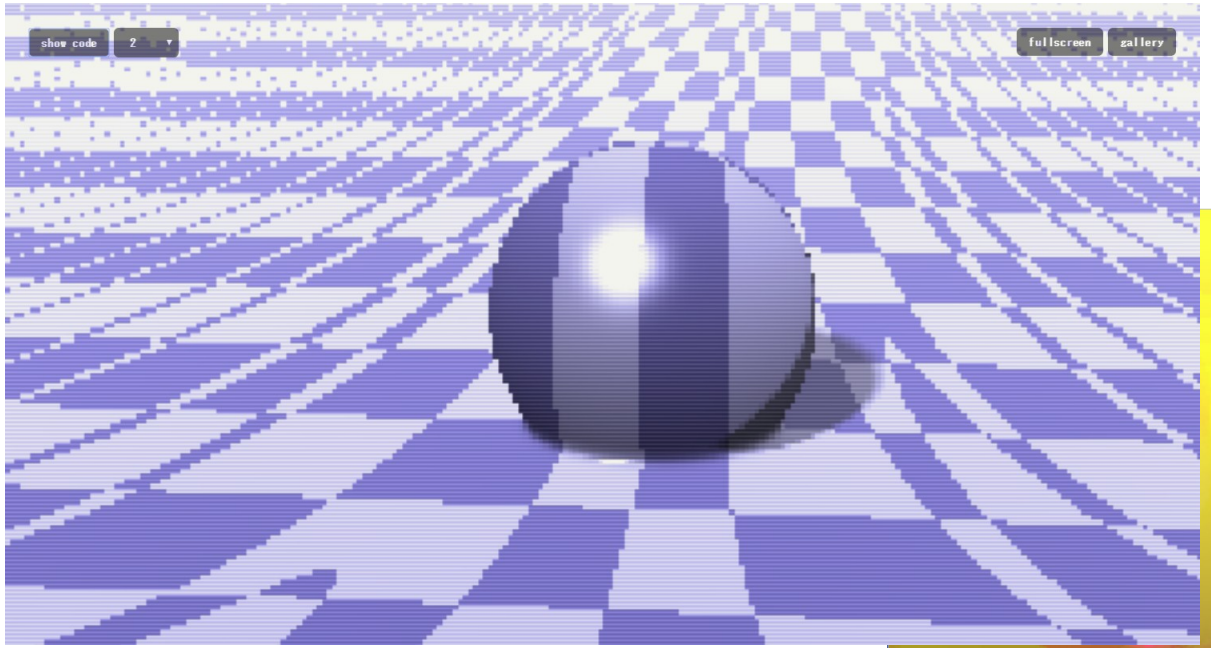
海外の凄そうな映像

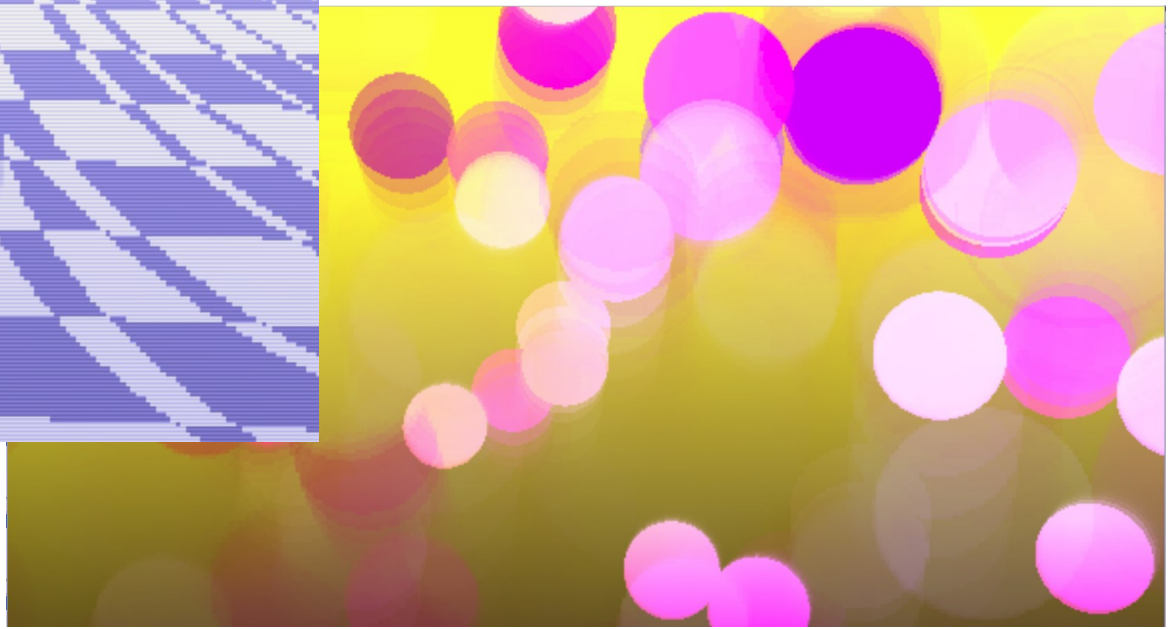
elevated by RGBa and TBC

10 年近く前に海外の有名なチームが作成したプログラムで、以下特徴あり。
・実行ファイルのサイズが 4kb
・映像と音楽をプログラムで生成してる

今回説明する内容は、使用 API は違いますがこの映像で使われてる技術と同様です。
（余談ですが、「デモシーン」と呼ばれるサブカルチャーが存在しており、この映像も該当します）

自作絵の一例





今回の説明内容

- コーディングだけで絵を作るアプローチの説明
- レイマーチングについて説明
- コーディングでグラフィックを作成する利点

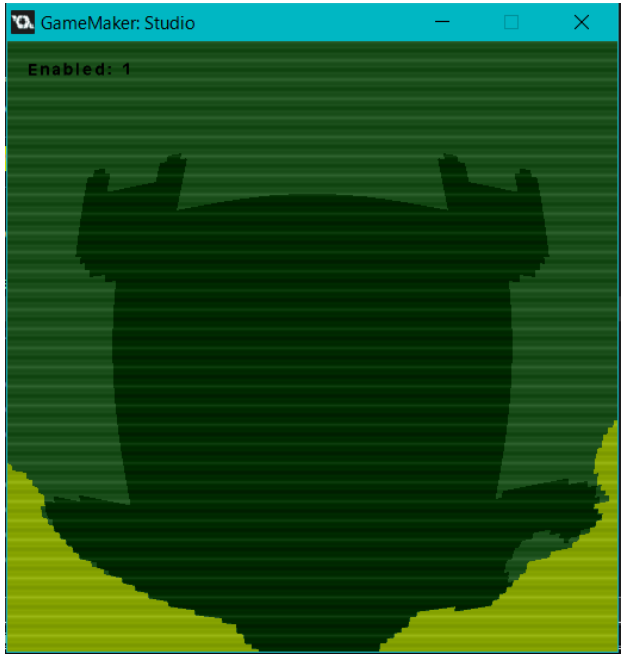
1. コーディングだけでグラフィックを作成する

コーディングだけでグラフィック作成してみるよ

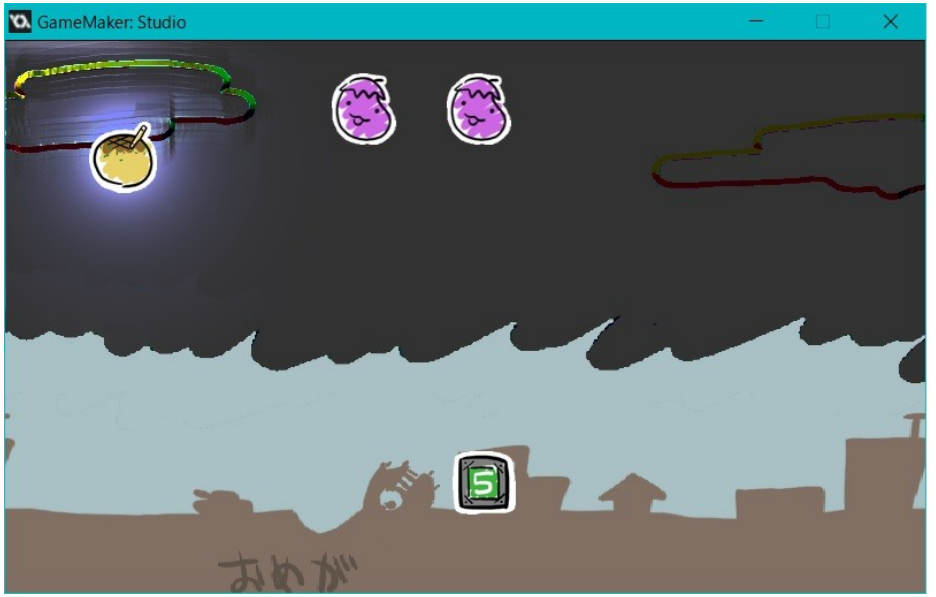
- 今回は「ポリゴンを配置し、その上にシェーダで色を定義して絵を書く」アプローチで描画する
- シェーダ→ 3D オブジェクトにおいて最終的に出力される色等の情報を定義する機能（前述）
  OpenGL/DirectX 等グラフィック API に実装されている
- 複数あるシェーダのうち、「フラグメントシェーダ」を利用

本来想定されるシェーダの使い方

例：テクスチャの色変更・フィルタ

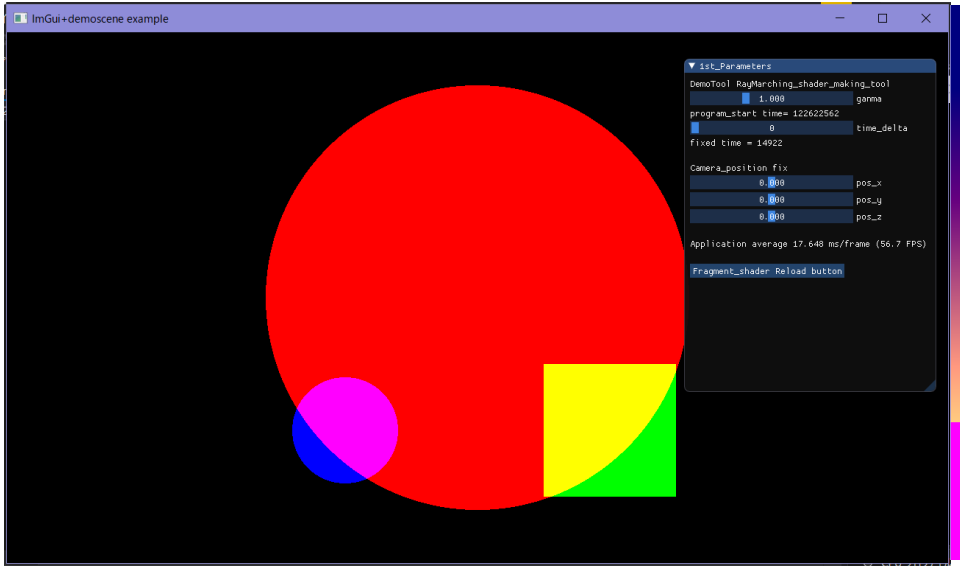


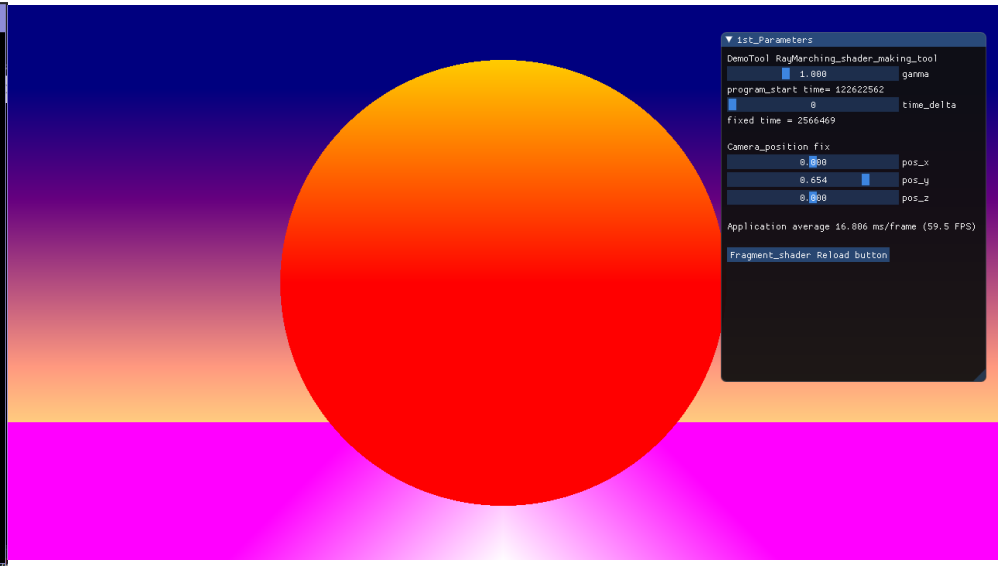
例：スペキュラマッピング
（テクスチャに光源を反射する凹凸つける）



お絵かきした一例

- 手法：テクスチャ上にシェーダでお絵かき
- 目標： 2D の絵（円・四角形等）





同様の手法を手軽に試せる環境

- WebGL(OpenGL をブラウザで駆動する技術) を使った GLSL 実行環境→ GLSLSandbox/Shadertoy
- 自作

```
void CreateShaderProgram()
{
    g_pProgram = ((PFNGLCREATEPROGRAMPROC)wglGetProcAddress("glCreateProgram"))();

    g_pTemp = ((PFNGLCREATESHADERPROC)(wglGetProcAddress("glCreateShader")))(GL_VERTEX_SHADER);
    ((PFNGLSHADERSOURCEPROC)wglGetProcAddress("glShaderSource"))(g_pTemp, 1, &vertexShaderSource, 0);
    ((PFNGLCOMPILESHADERPROC)wglGetProcAddress("glCompileShader"))(g_pTemp);
    ((PFNGLATTACHSHADERPROC)wglGetProcAddress("glAttachShader"))(g_pProgram, g_pTemp);

    g_pTemp = ((PFNGLCREATESHADERPROC)wglGetProcAddress("glCreateShader"))(GL_FRAGMENT_SHADER);
```

2. レイマーチングで立体を描画してみる

レイマーチングで描画する

- レイマーチングって何
  レイトレーシングの手法の一種（レイキャスト法ともいう）
- レイトレーシングって何
  立体を描画するための手法のひとつ

  詳細を説明しようとするとかなり難しいので、プログラム書くのに必要な知識だけ説明します。

レイトレーシングについて
大雑把レベルで説明

- 光の流れをシミュレートする
- 例えばどこから光があたって、どのオブジェクトに反射するか、回折するか等
- 複数のオブジェクトによる干渉がある、オブジェクトの材質が光を反射しにくいケース等、計算する要素が多い

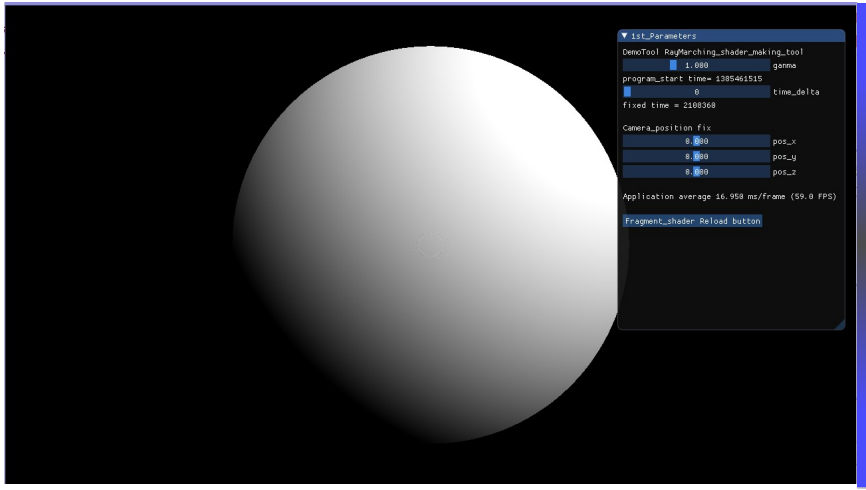
レイマーチングとは

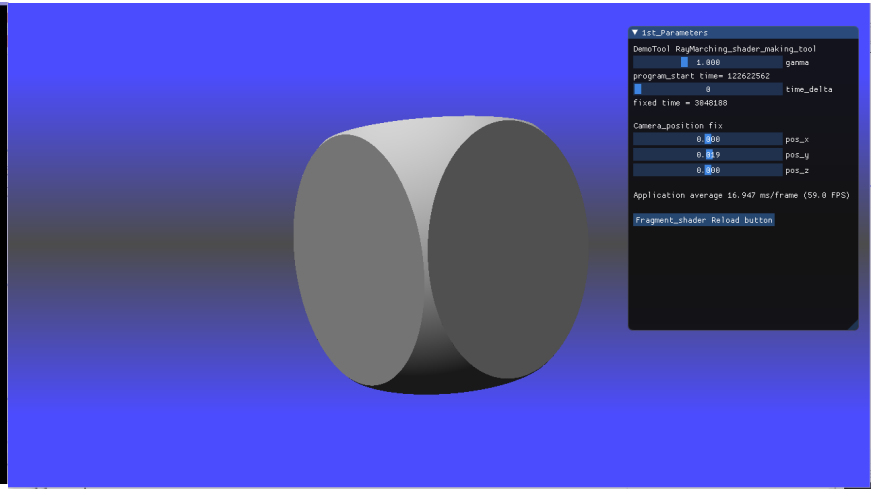
- 距離関数と呼ばれる関数を定義して、その関数が返す値を見て「レイが衝突しているか」を確認する手法
  例： float f = length(ray) – radian;// 円の式

- 調査位置からオブジェクトが存在する距離を計測、その後調査位置を少しずつ変えて同様の計測を実施
  →最終的に細かい形状を割り出す

お絵かきの一例

- 手法：さっきの GLSL 使う方法
- 目標：立体を描く（円、立方体等）





3. クリエイティブコーディングの利点

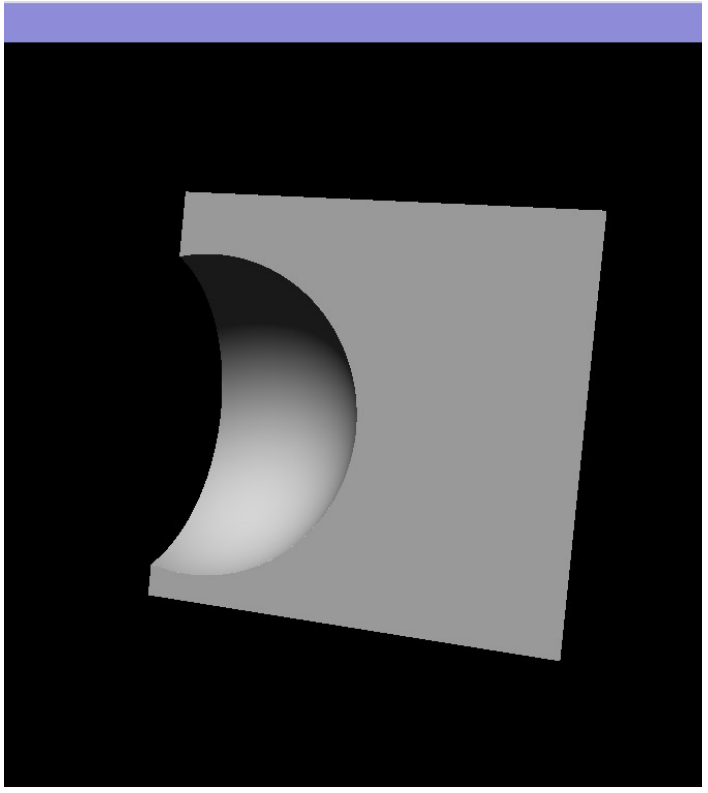
今回のプレゼン内容が役立つ状況とは・・・

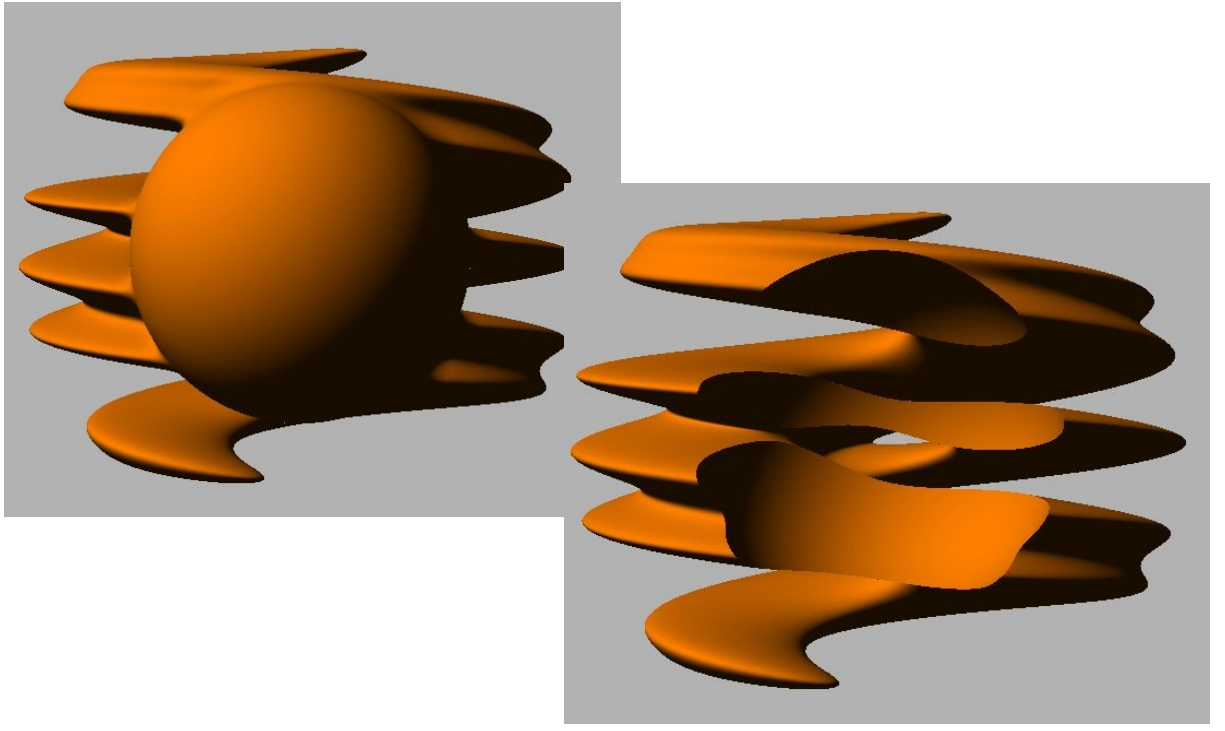
- 表現活動という見地からすればアリとは思うが・・・
- ゲーム・研究等に導入する際の優位点ってあるかな？
- メリット・デメリットを踏まえつつ判断は適時行うのでも問題ないとは思う

メリット

- 通常作成困難な物体が描画可能
  （ブーリアン演算による穴あけ等）
- アーキテクチャの限界を超えることが可能
  2D アーキテクチャで 3D 描画が可能等
- コンピュータ上で綺麗な絵が出せる
  （曲線、光源を駆使した風景画等）

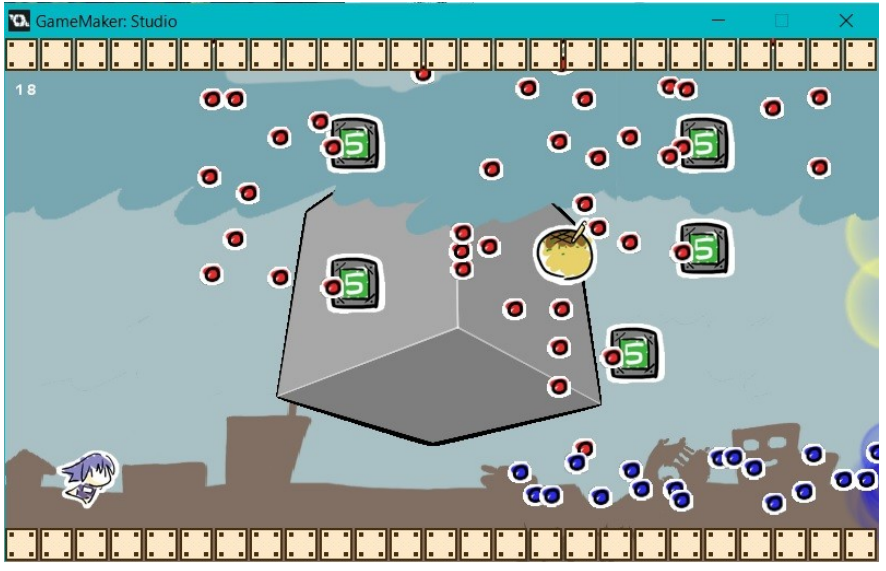
穴あけ事例

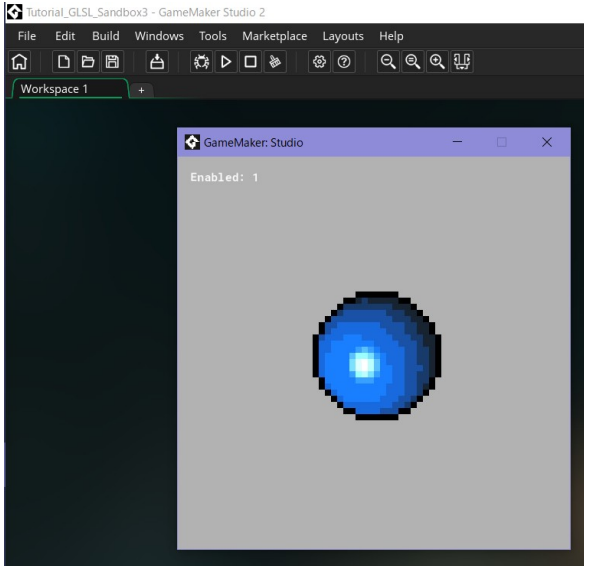




2D アーキテクチャで 3D 描画する事例

- GameMaker:Studio ・・・ 2D メインのゲームエンジン
  ここにレイマーチングで 3D モデルを表示
  2D テクスチャに描画しているためエンジン上の調整が容易





デメリット

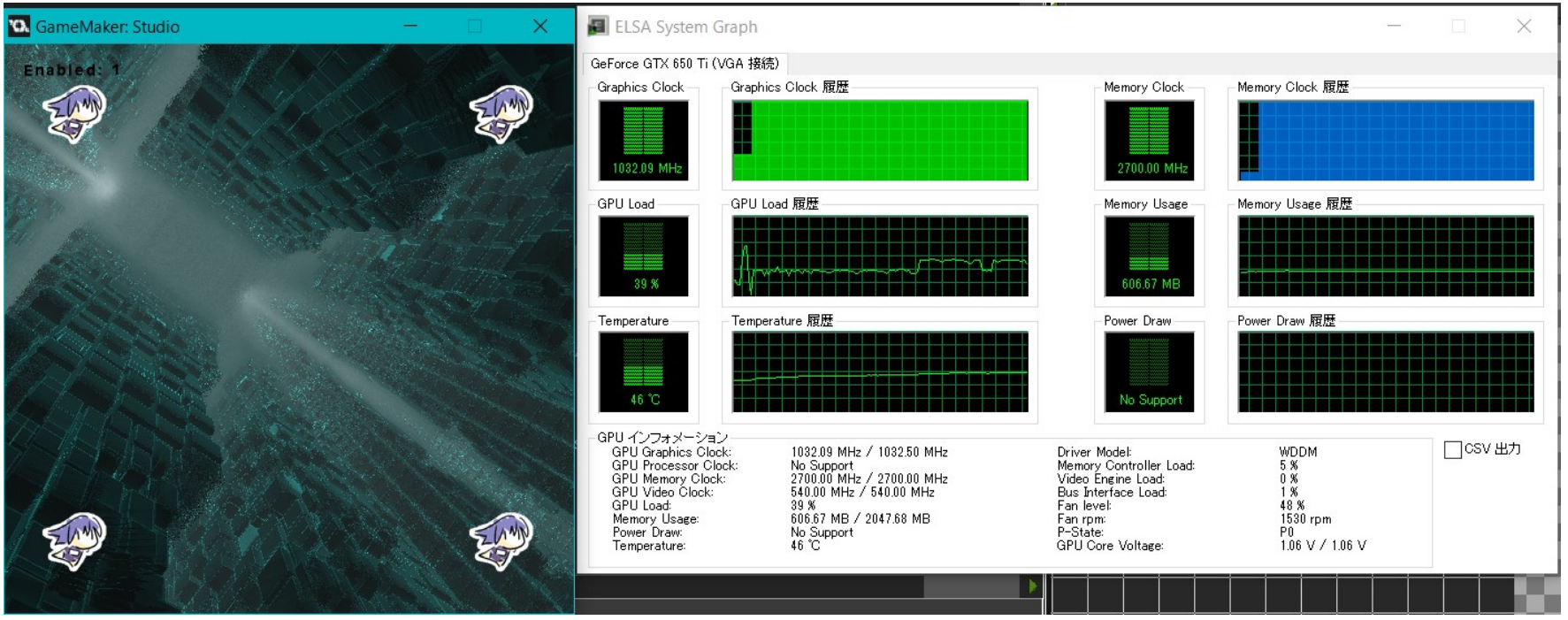
- 絵を一枚出すのに必要な手間が多い
- 現状実装可能なプログラム方法が特殊すぎる
- 場合によるが、計算量が多すぎるケースあり

プログラム方法

- C 言語 / アセンブラ /Rust 等
  ノンリアルタイムなら問題ないと思う
  リアルタイム出力が可能か要確認
- GLSL/HLSL 等
  プログラム方法が特殊
  手続き型に慣れてる人には苦行
- DirectX Raytracing
  Microsoft 公式 API 、出たばかりでノウハウが少ない

計算量が増大する事例

右のダイアログに GPU のリソース使用率を表示
GPU 使用率（左上・緑のゲージ）、メモリクロック（右上・青のゲージ）が常に高い



その他、個人的に良さそうな点

- 自分が新世界の神にでもなった気分が味わえる
- 適当な模様が必要になる場合、式から自作できる
  （ Perlin ノイズ等）
- シェーダの仕組みを小さい単位で知ることができる

これから冬も近づいてくるし、
インドア活動として
クリエイティブコーディングをオススメします

付録１：オススメの環境（下にいくほど茨の道）

- Processing/OpenFramework/TouchDesigner/VVVV
  コーディングから動作確認まで一個のツールで可能
- GLSL Sandbox/Shadertoy(Web サイト)
  コーディングから動作確認まで可能、シェーダ (GLSL) の知識が必要
- Unity/UnrealEngine/GameMaker:Studio 等ゲームエンジン
  変数の作法が違ったり直で絵を書き込めないケースあるので要確認
- 自力で構築 (OpenGL/DirectX を叩く / ソフトウェアレンダリング等)

付録２：参考資料

- GLSL Sandbox
  http://glslsandbox.com/
- Shadertoy
  https://www.shadertoy.com/

付録２：参考資料

- はじめようレイトレーシング
  http://raytracing.xyz/
- The Book Of Shaders
  https://thebookofshaders.com/
- 趣味としてのクリエイティブ・コーディング
  https://note.mu/deconbatch/m/m6968df38e53d
- WebGL 開発支援サイト wgld.org
  https://wgld.org/