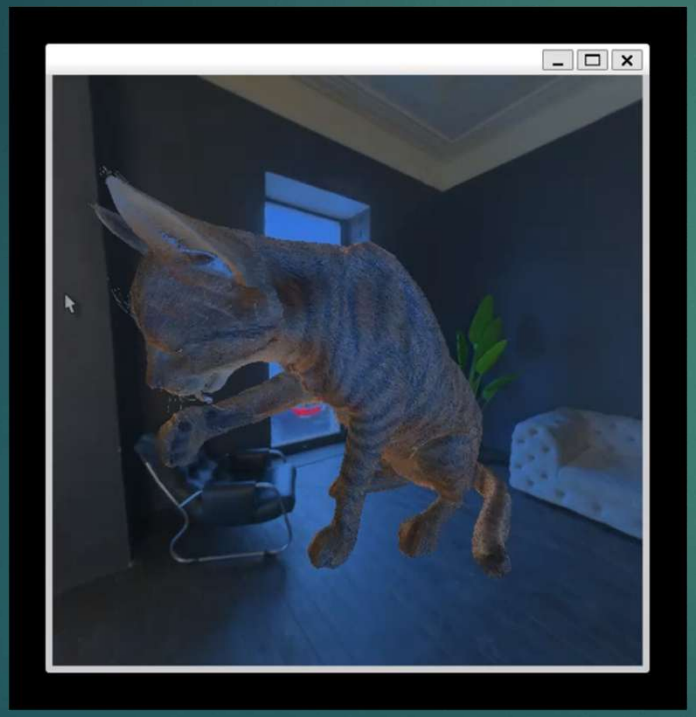
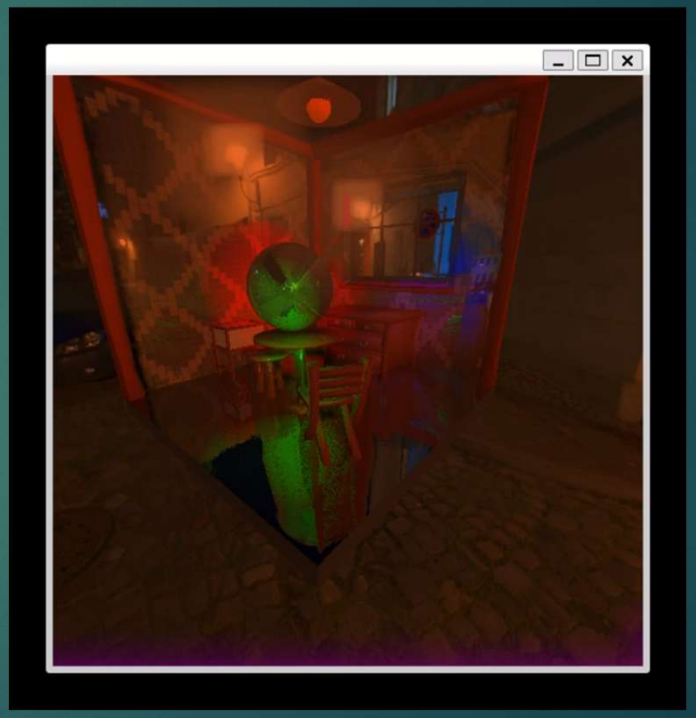
Obj Renderer
~3Dモデル描画エンジン開発~

福井大学 工学部 電気電子情報工学科 上田渉夢

リポジトリ+デモ動画：
https://github.com/regusan/ObjRenderer



デモ：VATによる頂点アニメーション



デモ：ライティング

プロジェクト概要

本プロジェクトを通じて学びたいこと
- ゲーム開発を通じて生まれたレンダリング・エンジンに関する興味を昇華させたいため
- C++に関するさらなる理解を深める

目的
- ビルド手順を記したり、扱いやすいプログラムにすることで第三者が使えるような整備を行う
- 最近のリッチなゲームに実装してあるレンダリング手法の実装を行う
- OpenGLやDirectXなどを使わずに実装してみる

概要	
ジャンル	リアルタイム ソフトウェアラスタライザ
言語	C++
バックエンド	X11
開発環境	VSCode,Ubuntu(WSL)
ライブラリ	Eigen:線形代数 stb_image.h:画像IO nlohmann-json3:JsonIO
方式	遅延レンダリング、 マルチレンダーターゲット、 イメージベースドライティング

```
/// @brief レンダーターゲットクラス
class RenderTarget : public RAsset
{
private:
    Vector2i screenSize = Vector2i(0, 0);
    vector<Vector3f> array; // 3次元ベクトルで色を保持
```

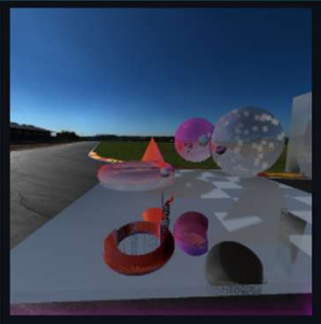
RenderTargetから作成

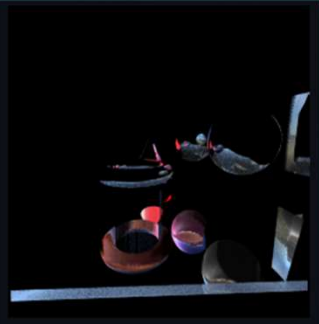
処理の流れ

エンジン機能を用い スポーンされたメッシュを収集 → 各メッシュに対して 頂点シェーダー実行 → ピクセルシェーダー実行 → ポストエフェクト実行 → 最終的な絵を合成 → X11ウィンドウに画像を転送

レンダリング系機能紹介

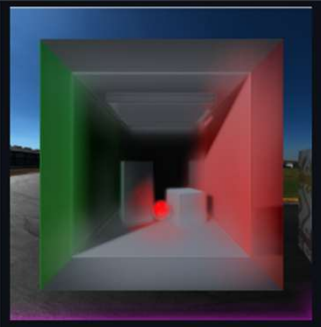
Screen Space Reflection





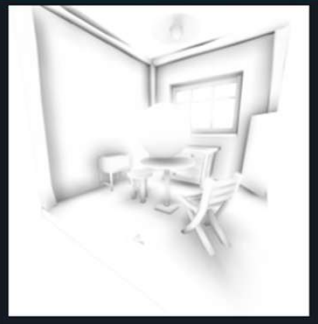
Screen Space Contact Shadow

Screen Space Global Illumination

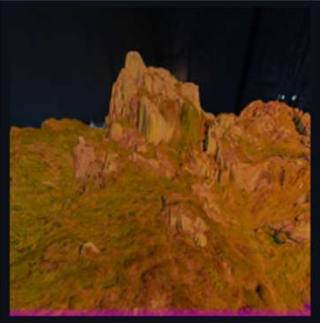


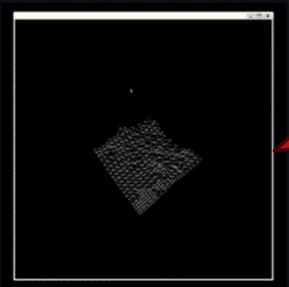
Screen Space Ambient Occlusion

None



Tessllation + Displacement Map





テッセレーション量を
LODのように距離で
変化させ負荷低減！

Lights

Point Light	Point Projector Light	Torus Light	Box Light

エンジン系機能紹介

リソースマネージャー

```
currentMaterial.diffuseMap = AssetSubSystem::getInstance().textureManager.LoadAsset(dir / texturePath);
```

マテリアルのテクスチャロード処理の例

- メッシュとテクスチャで使用している
- テクスチャやメッシュなど大容量のデータを一元管理し重複によるメモリの無駄遣いを防ぐ
- 参照カウンタ式の共有ポインタで管理
- テンプレートと継承を用い多様なオブジェクトを管理可能にしている

Jsonファイルによるシーン定義

```
{
    "_comment": "シーンの定義ファイル",
    "GameObjects": {
        "Mesh1": {
            "_comment": "メッシュの定義例",
            "class": "MeshActor",
            "_classExplain":"GCLASS()マクロで登録したクラス名で
            "args": {
            "_comment":"引数はargs内に記述する。args内はプログラ
                "position": [
                    0,
                    0,
                    0
                ],
                "rotation": [
                    0,
                    0,
                    0
                ],
                "scale": [
```

スポーンさせたいオブジェクトや
その引数を記述

シリアライズ

ゲーム上オブジェクトのクラス例

```
#pragma once

#include "../Actor.hpp"
using namespace Transform;

/// @brief Actorの例
class HogeActor : public Actor
{
protected:
    Vector3f color = Vector3f(1, 0, 0);

public:
    HogeActor() {}
    /// @brief Sceneファイル読み取り時にJsonから渡される
    /// @param args
    HogeActor(json args) : Actor(args)
    {
        this->color.x() = args["color"][0];
        this->color.y() = args["color"][1];
        this->color.z() = args["color"][2];
    };
    ~HogeActor() {};
};

// GCLASSマクロでSceneファイルから生成するための登録
GCLASS(HogeActor)
```

- エンジンが認識するためのデータ生成をUEのUCLASSマクロのようなGCLASSマクロで補助
- 特殊なコンストラクタを定義すればJsonファイルの引数をパースできる
- 毎フレーム呼ばれるTickやスポーン時に呼ばれるStart関数などを整備

ユーザーが極力エンジンを意識せずに
クラスを作成できるよう注力

学んだことと今後の展望

学んだこと

- ゼロからの実装力
  - 同じことをやってる人は少ないため、UEやUnityを参考にして自分のコードに落とし込む必要がある
    - 真に理解してるか問われる？
- 使いやすさを担保することは難しい
  - 一般化するのが大変
  - いかにエンジンの実装をユーザーに意識させないか
  - ただしきれいにまとまった時はすごくうれしい！
- 変更が入る前提での作成が大事
  - 機能を分割して作成し、それを組み合わせて目的のものを作る
  - 機能を所有させるか継承させるか

今後の展望

- レンダラをCUDAに移行
  - CPUでの並列処理など頑張ったが、やっぱり遅い…
  - 現在CUDAを勉強中…
- 気になる技術を適宜実装してみる
  - 最近はVATによるアニメーションとサブサーフェススキャッタリングを実装中…
- UI機能など
- エディタを充実させたい
  - X11バックエンド以外をサポートさせる…

ご清聴ありがとうございました!