内容

河原電子ビジネス専門学校 ゲームクリエイター科
小村　篤

１．作品概要

・タイトル
　カタタママリシイ

・対応ハード
　PC　Windows10

・製作人数
　１人

・製作期間
　2019 年 5 月～ 2020 年 2 月　2020 年 7 月～

・製作環境
　エンジン
　　学校内製のエンジン(DirectX11)
　ツール
　　Microsoft VisualStudio2019
　　3DSMAX2020
　　FireAlpaca
　　GitHub
　　Effekseer
　使用言語
　　C++　Python3.7

・担当箇所
　ゲーム部分全般、このポートフォリオに記載しているエンジン部分

２．オブジェクトのデータ管理

・参照
PythonApplication1　Game/Object/ObjectData.h,cpp

ゲーム内には様々なオブジェクトがありますが、それらのデータは Excel で管理しています。そのデータを Python でテキストドキュメントに変換して、そのテキストドキュメントをゲームで読み込んでいます。

Excel

名前	図鑑とかで表示する名前	幅(x)	高さ(y)	奥行(z)	体積(cm3)	球体か？(1でtrue、0でfalse)
othumbtack	ガビョウ	1.3	0.7	1.3	9.464	0
othumbtack_blue	ガビョウ	1.3	0.7	1.3	9.464	0
othumbtack_red	ガビョウ	1.3	0.7	1.3	9.464	0
oeraser_blue	ケシゴム	7	2.5	13.3	1862	0
oeraser_red	ケシゴム	7	2.5	13.3	1862	0
opencil_blue	エンピツ	2.9	2.5	22.8	1322.4	0
opencil_yellow	エンピツ	2.9	2.5	22.8	1322.4	0
odomino_blue	ドミノ	4.6	5.7	1.7	356.592	0
odomino_red	ドミノ	4.6	5.7	1.7	356.592	0
odomino_green	ドミノ	4.6	5.7	1.7	356.592	0
omatchstick	マッチボウ	6.6	0.4	0.24	5.0688	0
osaikoro	サイコロ	1.8	1.8	1.8	46.656	0
omagnet	ジシャク	8.3	0.7	2.3	106.904	0
omouse	ネズミ	4.6	9.9	18	6557.76	0
ocockroach	ゴキブリ	8	1.7	9.1	990.08	0
oglass_yellow	グラス	11.1	19.6	11.1	19319.328	0
oglass_blue	グラス	11.1	19.6	11.1	19319.328	0

Python

テキストドキュメント

test.cag - メモ帳

ファイル(F)　編集(E)　書式(O)　表示(V)　ヘルプ(H)

```
othumbtack ガビョウ 1.3 0.7 1.3 9.463999999999999 0 0 0 0
othumbtack_blue ガビョウ 1.3 0.7 1.3 9.463999999999999 0 0 0 0
othumbtack_red ガビョウ 1.3 0.7 1.3 9.463999999999999 0 0 0 0
oeraser_blue ケシゴム 7.0 2.5 13.3 1862.0 0 1 0 0
oeraser_red ケシゴム 7.0 2.5 13.3 1862.0 0 1 0 0
opencil_blue エンピツ 2.9 2.5 22.8 1322.4 0 1 0 0
opencil_yellow エンピツ 2.9 2.5 22.8 1322.4 0 1 0 0
odomino_blue ドミノ 4.6 5.7 1.7 356.592 0 0 0 0
odomino_red ドミノ 4.6 5.7 1.7 356.592 0 0 0 0
odomino_green ドミノ 4.6 5.7 1.7 356.592 0 0 0 0
omatchstick マッチボウ 6.6 0.4 0.24 5.0688 0 0 0 0
osaikoro サイコロ 1.8 1.8 1.8 46.656000000000006 0 0 0 0
omagnet ジシャク 8.3 0.7 2.3 106.904 0 1 0 0
omouse ネズミ 4.6 9.9 18.0 6557.76 0 0 1 0
ocockroach ゴキブリ 8.0 1.7 9.1 990.0799999999999 0 0 1 0
oglass_yellow グラス 11.1 19.6 11.1 19319.328 0 0 0 0
oglass_blue グラス 11.1 19.6 11.1 19319.328 0 0 0 0
opetbottle ボトル 9.5 32.5 9.5 23465.0 0 1 0 0
osmartphone スマートフォン 20.1 1.9 34.5 10540.439999999999 0 0 0 0
ospoon スプーン 3.3 1.1 21.5 624.36 0 1 0 0
ofork フォーク 3.3 1.0 19.9 525.3599999999999 0 1 0 0
oclock トケイ 15.0 19.0 19.9 45372.0 0 0 0 0
oapple リンゴ 25.0 25.0 25.0 125000.0 0 0 0 0
```

ゲームで読み込む

３．オフスクリーンレンダリング

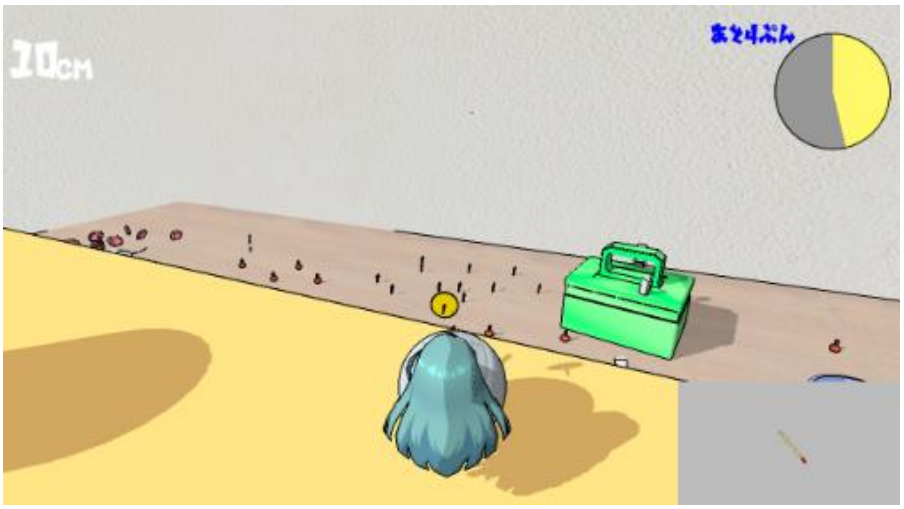
・参照

Game/OffScreen.h .cpp　Game/Object/Collection.h .cpp

ゲーム中に右下に巻き込んだものを表示していますが、これはフレームバッファとは別のレンダリングターゲットにモデルを描画したのを、フレームバッファに合成しています。

フレームバッファ　　オフスクリーン

フレームバッファ

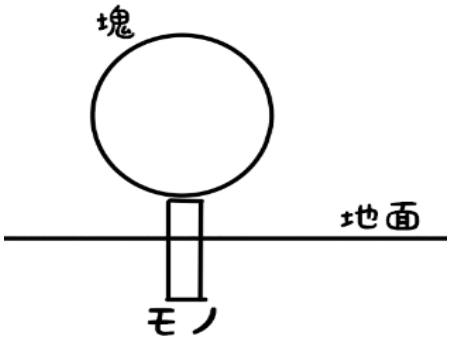


４．デコボコ処理

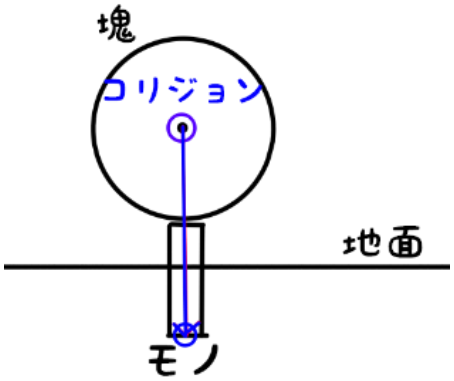
あるオブジェクトを巻き込むと、オブジェクトの形に合わせて塊が動く

・参照

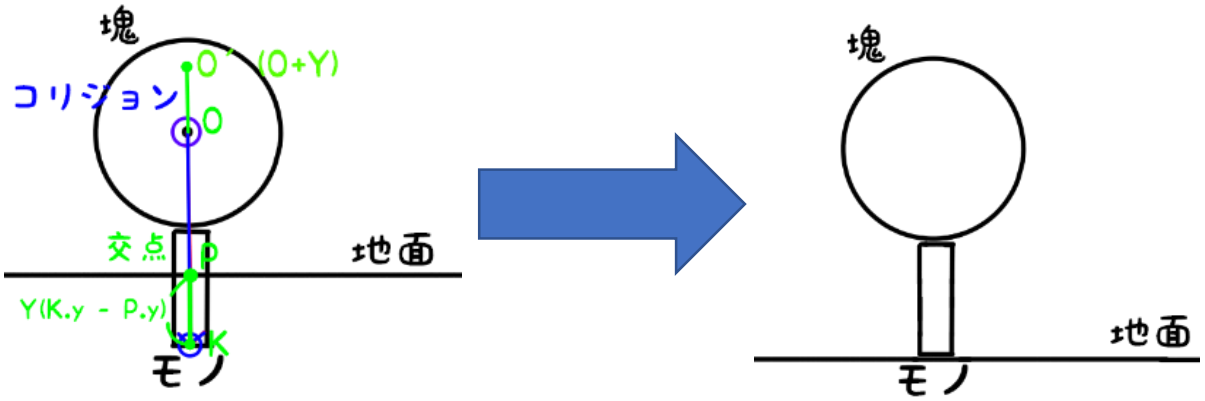
Game/LineSegment.h .cpp



オブジェクトがこのように塊にくっついている場合、



コリジョンを塊の中央からオブジェクトの先端まで移動させます。



この時コリジョンが地面と衝突したら、オブジェクトの先端から交点までの高さを求めて、その高さを塊の座標に加算します。

５．塊にくっついた時のオブジェクトの挙動

・参照
Game/Object/Obj.h .cpp

スキニングとよく似ているのですが、オブジェクトが塊にくっついた時、まず１フレーム目に

オブジェクトのワールド行列×塊の逆ワールド行列

＝塊座標系のオブジェクトのローカル行列

をして塊を中心としたオブジェクトのローカル行列を求めます。

そして毎フレーム

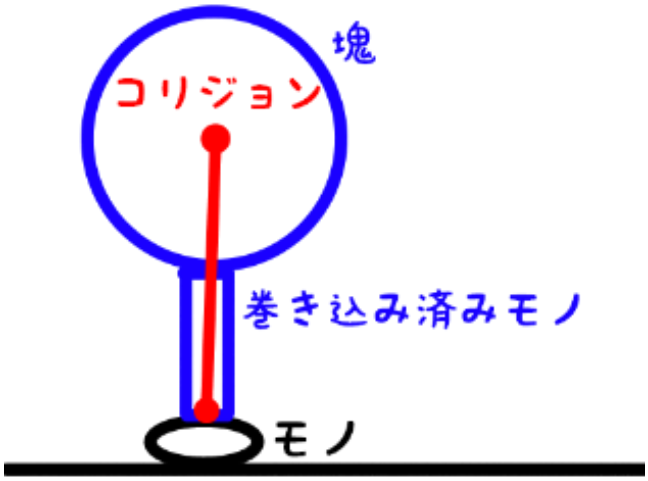
塊座標系のオブジェクトのローカル行列×塊のワールド行列＝オブジェクトのワールド行列

をしてオブジェクトのワールド行列を求めています。

６．塊とオブジェクトの巻き込み判定

```
//衝突したときに呼ばれる関数オブジェクト(地面用)
struct SweepResultGround : public btCollisionWorld::ConvexResultCallback
{
    bool isHit = false;                                 //衝突フラグ。
    CVector3 hitPos = CVector3(0.0f, -FLT_MAX, 0.0f);   //衝突点。
    CVector3 startPos = CVector3::Zero();                   //レイの始点。
    CVector3 hitNormal = CVector3::Zero();              //衝突点の法線。
    btCollisionObject* me = nullptr;                    //自分自身。自分自身との衝突を除外するためのメン
    float dist = FLT_MAX;                               //衝突点までの距離。一番近い衝突点を求めるため。
    CVector3 GroundNormalVector = CVector3::AxisY();
    float radius = 0.0f;                                //塊の半径
    //衝突したときに呼ばれるコールバック関数。
    virtual btScalar    addSingleResult(btCollisionWorld::LocalConvexResult& convexResult, bool normalI
    {
        if (convexResult.m_hitCollisionObject == me
            || convexResult.m_hitCollisionObject->getUserIndex() == enCollisionAttr_Character
            || convexResult.m_hitCollisionObject->getUserIndex() == enCollisionAttr_User
            || convexResult.m_hitCollisionObject->getUserIndex() == enCollisionAttr_LineSegment) {
            //自分に衝突した。or キャラクタ属性のコリジョンと衝突した。
            return 0.0f;
        }
        //コリジョンの属性がモノであるかつサイズが塊がモノより十分に大きければ巻き込んだことにする
        if (convexResult.m_hitCollisionObject->getUserIndex() == enCollisionAttr_Object
            && GetCompareSize(radius, convexResult.m_hitCollisionObject->GetSize())) {
            convexResult.m_hitCollisionObject->SetisHit();
```

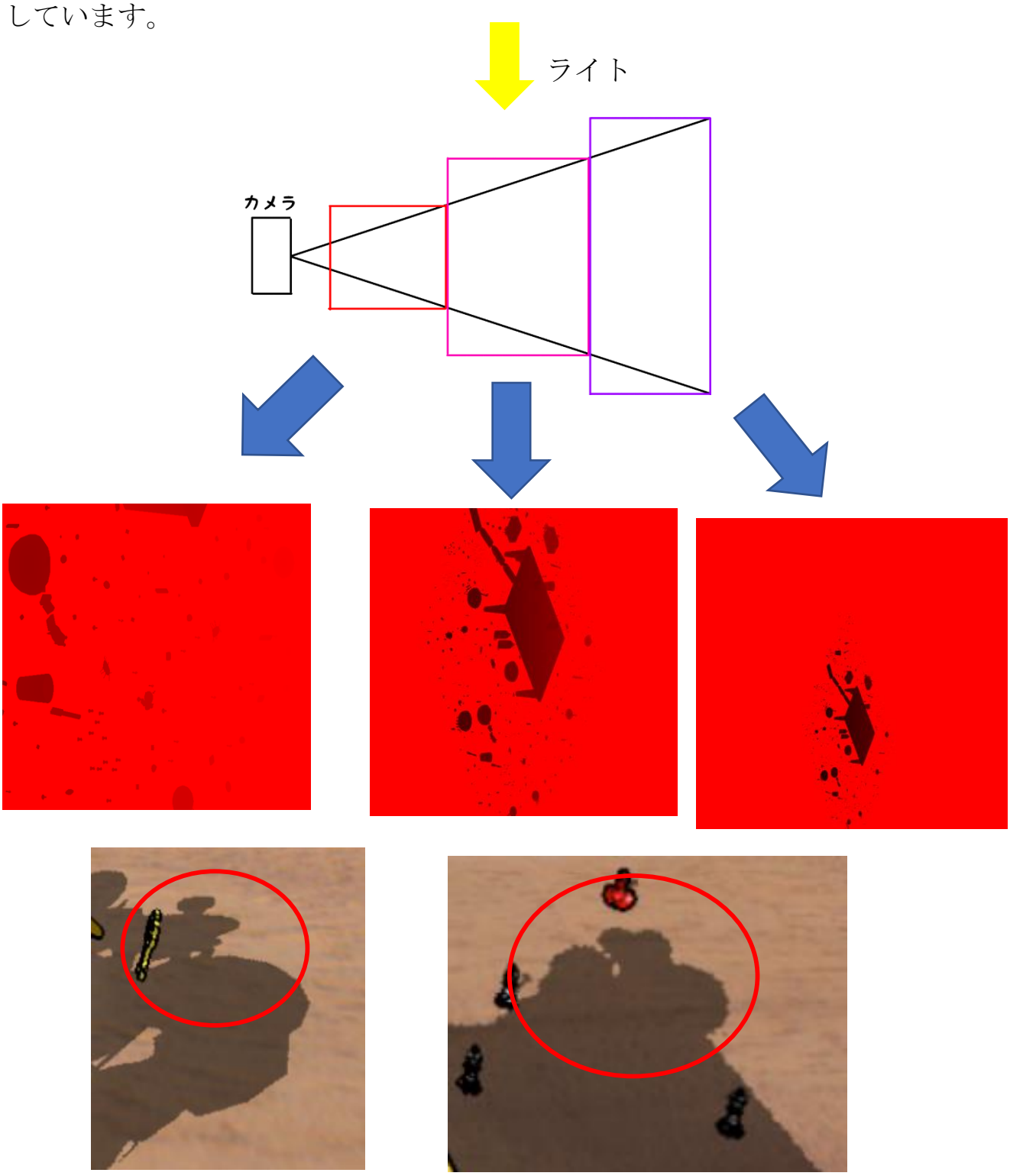
これが塊のコリジョンが他のコリジョンと衝突した際のコールバック関数の一部なのですが、オブジェクトのコリジョンと衝突した時に、塊のサイズがオブジェクトより十分に大きければ、巻き込んだことにしています。



またこのように尖っているオブジェクトを巻き込んでいる場合には、ガタガタ処理のためにコリジョンをオブジェクトの先端まで移動させるのですが、その際も同様の処理をしています。これにより、尖ったオブジェクトを巻き込んでいてもオブジェクトを巻き込みやすくしています。

７． Cascade Shadow Map

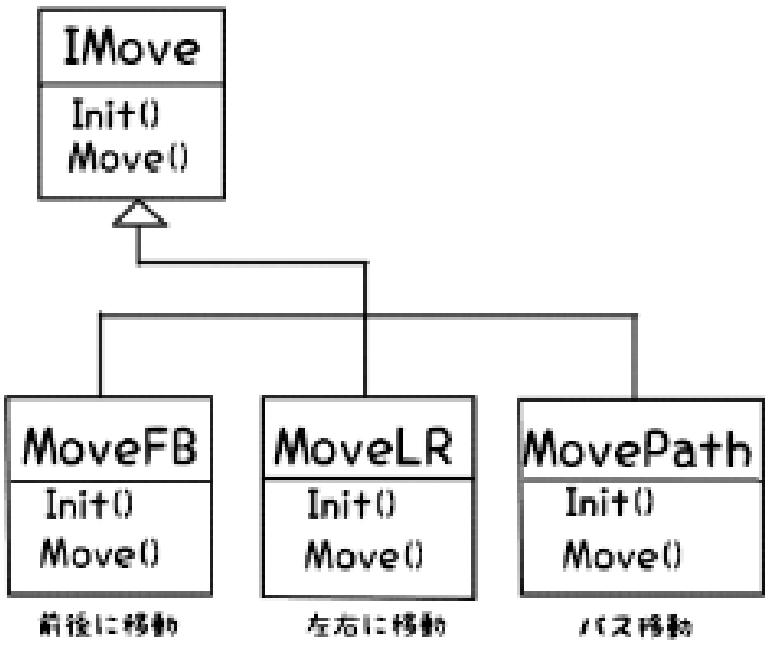
視錘台をカメラに近い場所から３つに分割して、シャドウマップを生成しています。

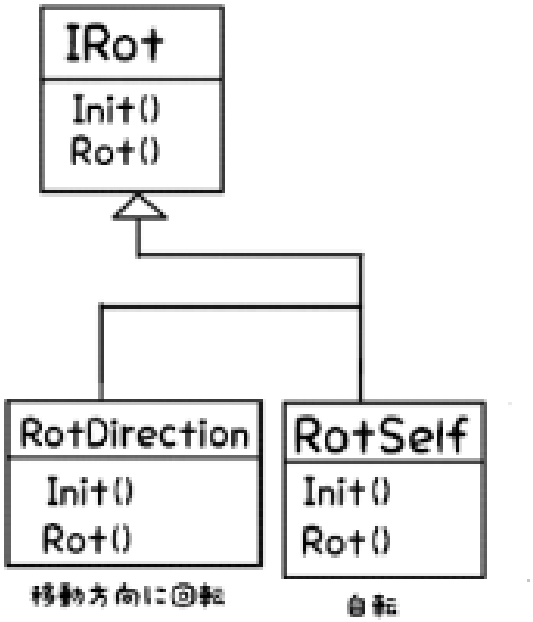


メモリの使用量を抑えるために、カメラから距離があるシャドウマップは解像度を落としています。

８．Strategy パターン

オブジェクトの移動や回転の挙動を Strategy パターンで実装しています。





９．Qiita の記事

以下 Qiita で私が書いた記事の URL です。

・エッジ検出

・参照
kgEngine/graphics/normal/EdgeDetection.h .cpp
Game/Assets/shader/edgedelection.fx

https://qiita.com/akurobit/items/b5231ffae738810b63e7

・円形ゲージ

・参照
kgEngine/graphics/2D/Sprite.h .cpp
Game/Assets/shader/sprite.fx

https://qiita.com/akurobit/items/ab90b88088678f706e3a

・トゥーンシェーダー

・参照
Game/Assets/shader/model.fx

https://qiita.com/akurobit/items/a016ef4a022eed15c268

・被写体が全部移るためのカメラ設定

・参照
Game/Collection.cpp

https://qiita.com/akurobit/items/a6dd03baef6c05d7eae8

・画面分割

・参照
kgEngine/graphics/GraphicsEngine.h .cpp

https://qiita.com/akurobit/items/1619bc26010441b8008c

１０．URL

GitHub の URL です。このゲームのリポジトリです。

https://github.com/komura-athushi/kgEngine.git