

河原電子ビジネス専門学校

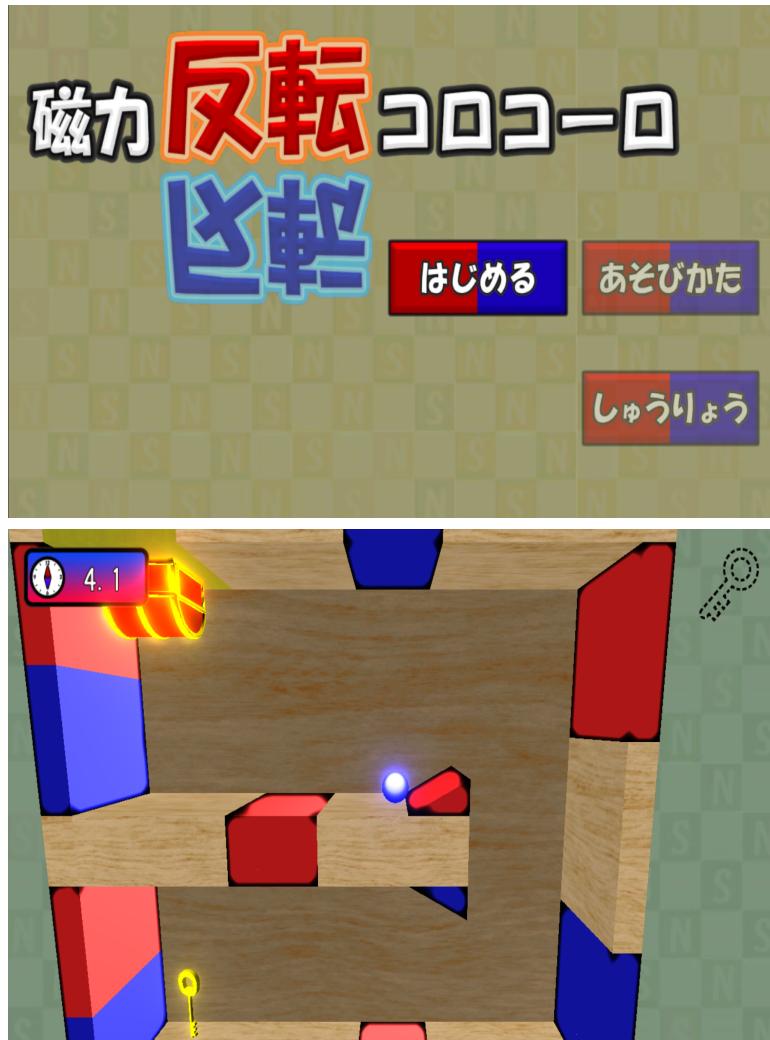
ゲームクリエイター科 2年 東 浩樹

目次

- 1.概要
- 2.操作説明
- 3.担当ソースコード
- 4.エンジン側の追加仕様
- 5.技術要素
 - 1.PBR (物理ベースレンダリング)
 - 2.デプスシャドウ
 - 3.川瀬式ブルーム
 - 4.シンプルクスノイズを利用したノイズ表現
 - 5.輪郭線の実現
- 6.こだわりと苦労した点

概要

作品名：磁力反転コロコロ



使用ゲームエンジン：自作エンジン

使用言語：C++

使用ツール：Visual Studio 2019、3ds Max

使用ライブラリ：BulletPhysics、Effekseer、DirectXTK12

環境：Windows10、DirectX12

制作人数：3人

制作期間：4カ月半

GitHub：<https://github.com/Azuki2828/MyGame>

操作説明

Jキー or Aボタン：自機の磁極を変換



担当ソースコード

BackGround.h
BackGround.cpp
ConstTriggerBoxValue.h
ConstValue.h
DeathBlock.h
DeathBlock.cpp
DirectionLight.h
DirectionLight.cpp
FontRender.h
FontRender.cpp
HUD.h
HUD.cpp
Key.h
Key.cpp
LightBase.h
LightBase.cpp
LightCamera.h
LightManager.h
LightManager.cpp
Magnet.h
Magnet.cpp
main.h
MainCamera.h
MainCamera.cpp
Player.h
Player.cpp
RuleSceneConstData.h
SaveData.h
SaveData.cpp

Seesaw.h
Seesaw.cpp
SkinModelRender.h
SkinModelRender.cpp
SoundManager.h
SoundManager.cpp
SpriteRender.h
SpriteRender.cpp
TreasureBox.h
TreasureBox.cpp

また、上記含め、全てのコードのリファクタリングを担当。

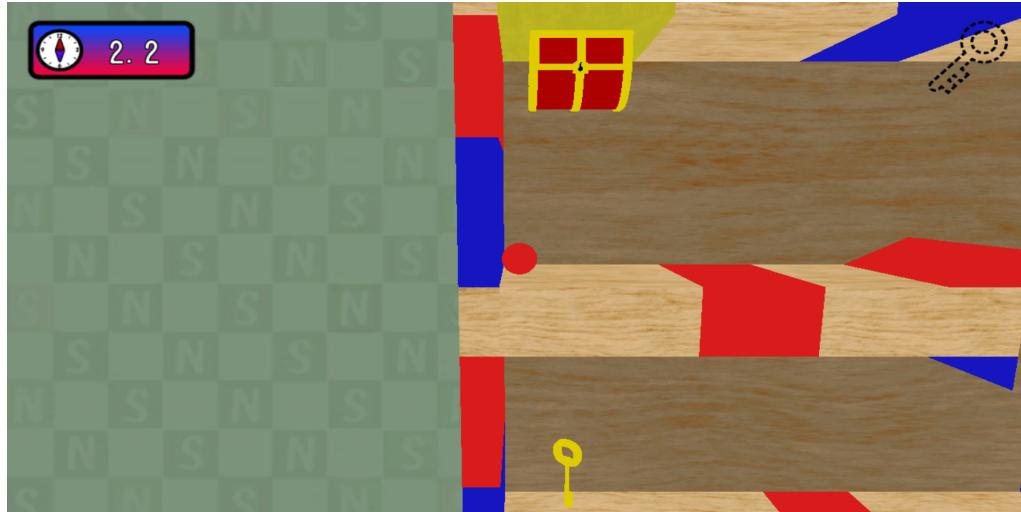
エンジン側の追加仕様

- ・PBRを実装
- ・川瀬式ブルームを実装
- ・デプスシャドウを実装
- ・ノイズ表現を実装
- ・輪郭線の表現を実装
- ・SkinModelRenderクラスを実装
- ・SpriteRenderクラスを実装
- ・SoundManagerクラスを実装
- ・FontRenderクラスを実装
- ・HUDクラスを実装

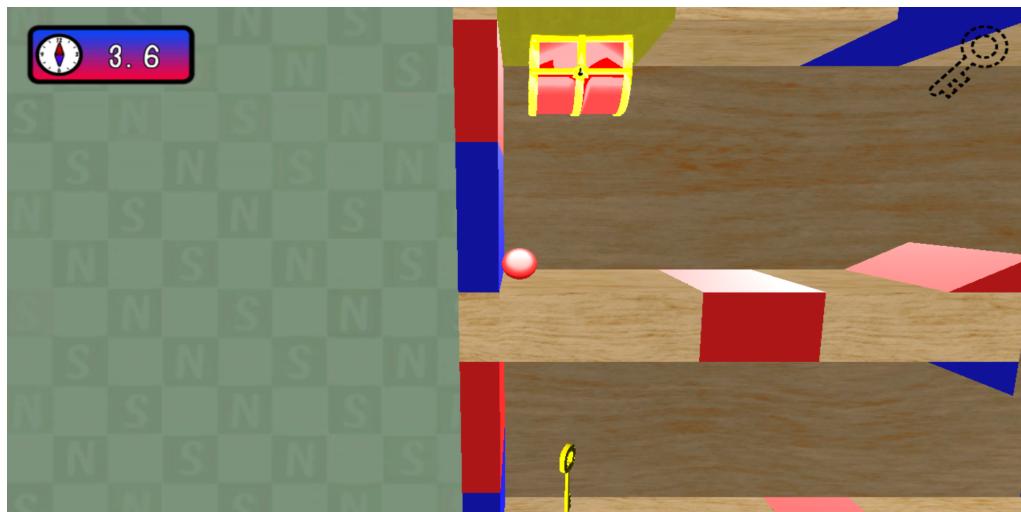
技術要素

1.PBR（物理ベースレンダリング）

本作品ではディズニーベースのPBRを実装しており、**フレネル反射を考慮した拡散反射、クックトランスマルモデルを利用したスペキュラ反射**を用いています。ランバート拡散反射では、入射してきた光よりも強い光を返すため、物理的に正しくはありませんでした。その値は π 倍であることが知られています。そのため、本作品ではその値を π で割った**正規化ランバート拡散反射**を採用しています。鏡面反射では、物体の金属度、なめらかさのパラメータを元に鏡面反射率を求めていました。オン鏡面反射ではこのパラメータが無かったため、視点からサーフェイスに伸びるベクトルと反射ベクトルしか鏡面反射に使うネタがありませんでした（絞り率は定数とする）。今回は各物体ごとにパラメータを設定し、不自然な金属感を出さないようにしています。これにより、エネルギー保存則に従ったライティング処理を実装しました。



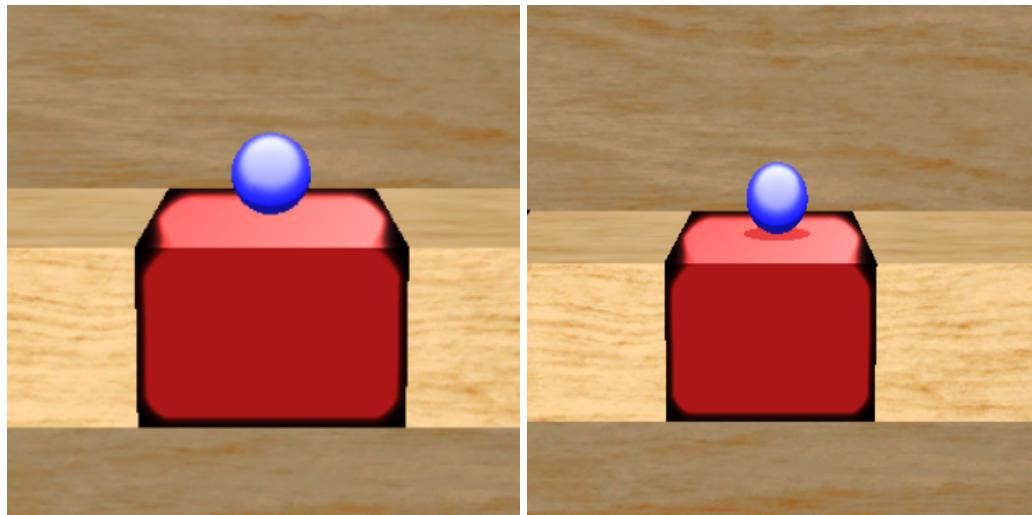
↑テクスチャカラーによるゲーム画面



↑PBR実装後のゲーム画面

2. デプスシャドウ

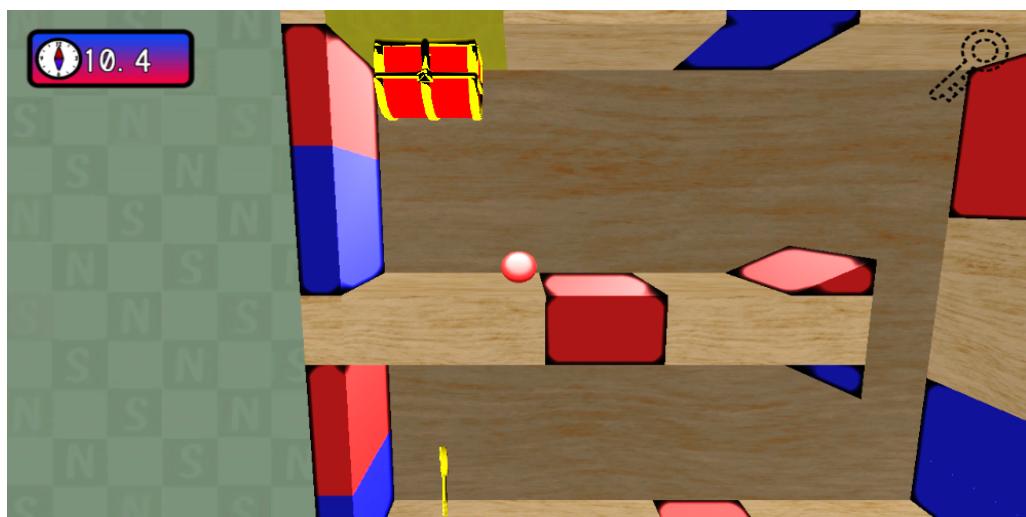
影を生成する手法として、デプスシャドウを採用しています。投影シャドウではグレースケールをシャドウマップに描き込んでいましたが、ライトスクリーン空間でのZ値（深度値）をシャドウマップに描き込むように切り替えることにより、投影シャドウの欠点であった、「影が落ちないはずの場所に影が落ちてしまう」現象を克服しました。



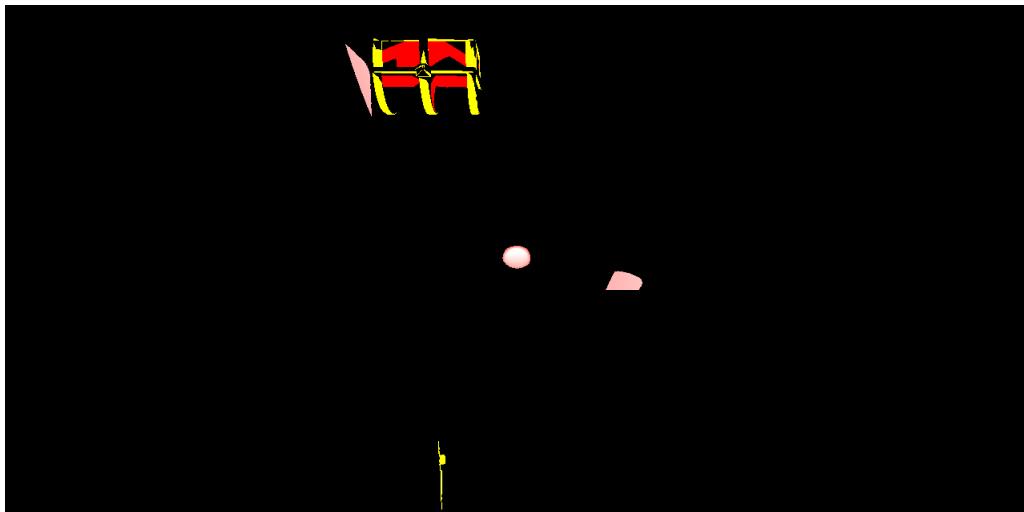
<左>デプスシャドウなし <右>デプスシャドウあり

3. 川瀬式ブルーム

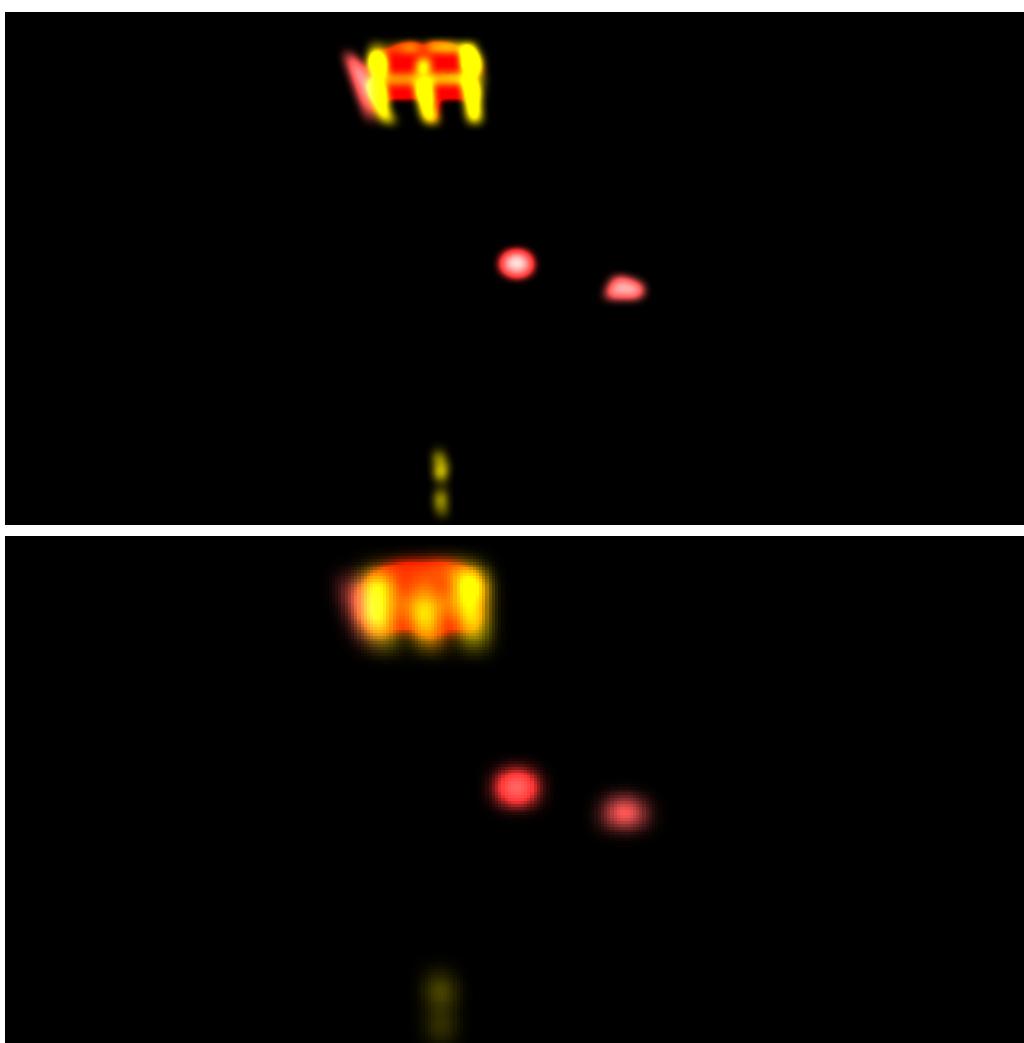
ぼかし表現にはガウシアンブラーを使用していますが、このブラーを一度だけでなく、複数回使用しています。その際に、ダウンサンプリング用のレンダリングターゲットを用意して、ガウスフィルターをかけて縮小します。それらを同解像度に拡大合成することであふれテクスチャを生成し、絶妙なブルームに仕上げています。

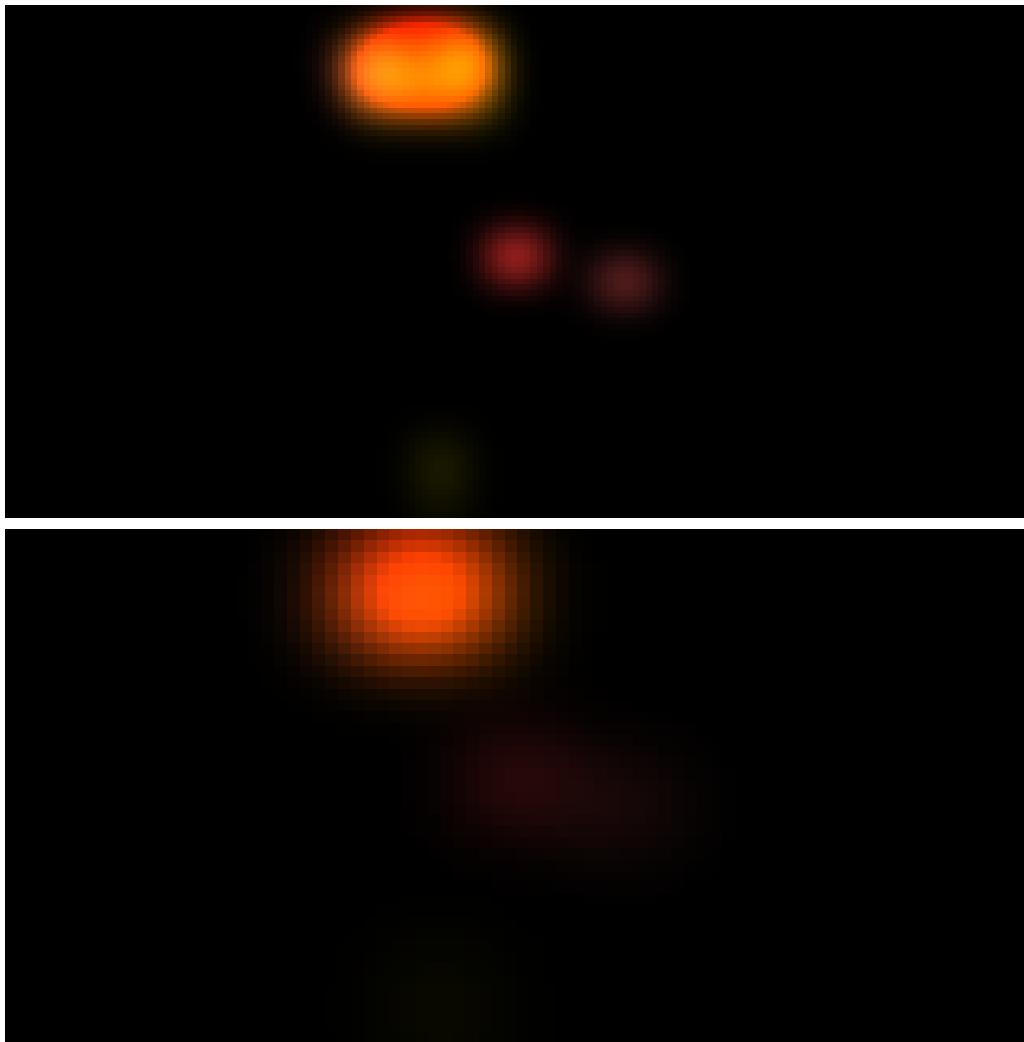


↑ブルームをかける前の画面

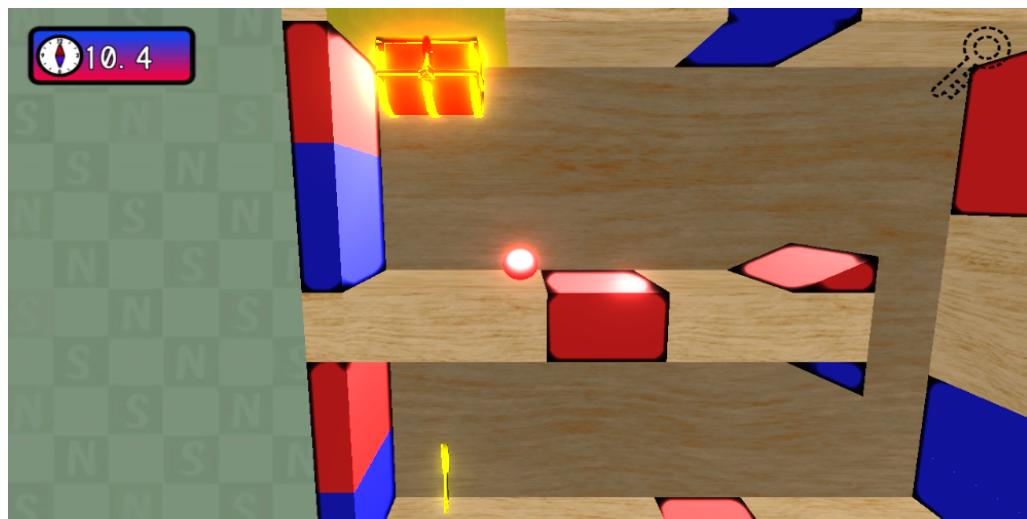


↑輝度テクスチャ





↑輝度テクスチャを使用して4回ガウシアンブラーをかける。



↑ブルームをかけた後のゲーム画面

4. シンプレックスノイズを利用したノイズ表現

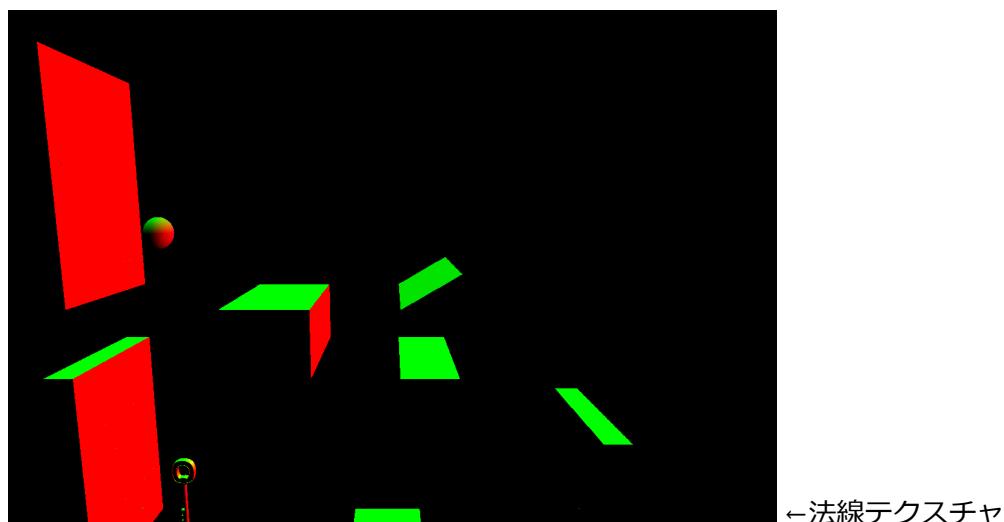
シンプレックスノイズは連続性のある乱数を生み出し、テクスチャの加工に用いることで様々な表現ができる

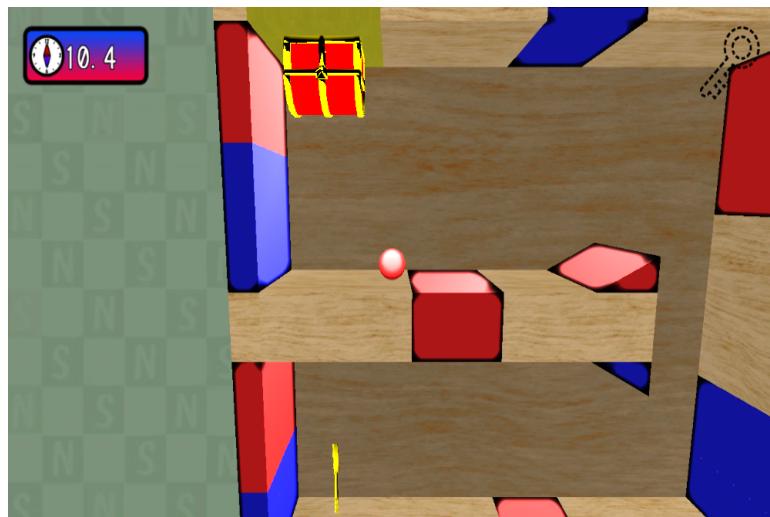
ます。パーリンノイズでは、N次元において 2^N 個の頂点が必要でしたが、シンプレックスノイズはN+1個の頂点数で済むため、次元数が増えるほど計算量も少なくなり、高速に計算することができます。このゲームのとあるブロックにもこのシンプレックスノイズを使用し、1フレームごとにテクスチャを歪ませ、気味の悪いノイズ表現に成功しています。



5.輪郭線の実現

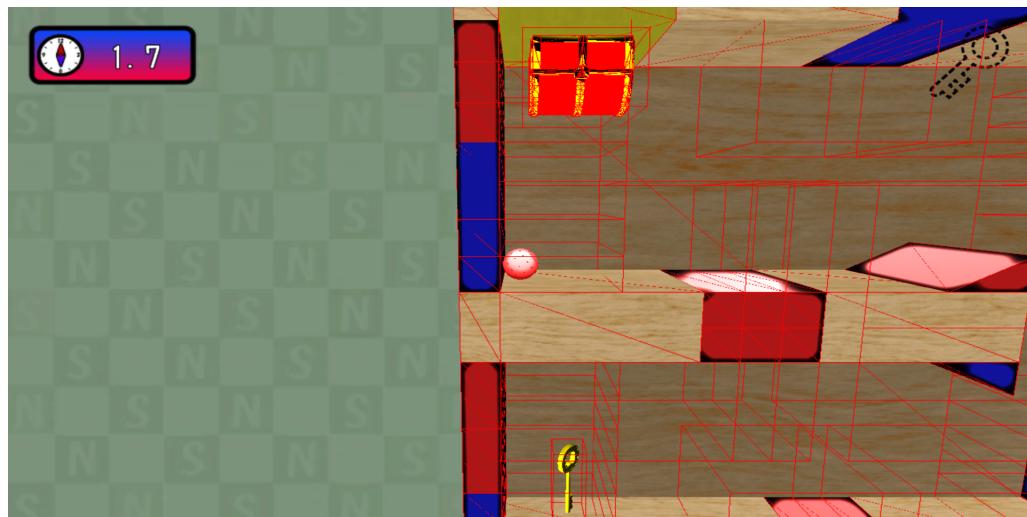
輪郭線の実現のために、法線テクスチャ (G-Buffer) を生成しています。この法線テクスチャでは、あるテクセルの法線情報と周囲 8 テクセルの法線情報を比較し、その内積で、オブジェクトの角に近いか判定しています。





こだわりと苦労した点

このゲームは、磁石から受ける磁力で自機を進めていきます。そのため、ゴールするためには磁力の強さや自機の質量、摩擦などを調整する必要がありました。しかしそれ以上に問題だったのが磁力の影響範囲です。かつては自機と磁石の距離で磁力計算をしていました。そのため、ステージ全体に磁石が設置されていることもあり、想定していない磁石からも磁力を受けてしまい、まともに進めないということが起きました。そこで、**磁力の影響範囲を、距離ではなくトリガーボックスを用いて行う**ことにしました。



↑赤線が影響範囲を示している

各磁石ごとに影響範囲を決めることで、ステージギミックも作りやすくなり、不自然な動きも抑えることができました。