# NRealエミュレータハンズオン

@yusuke_ota

# 環境

- OS: Windows 10(MacやLinuxでもよいはず)
- Unity: 2019.3.7f1
- NrealSDK: 1.2.1 ダウンロードページ(要ユーザー登録)

# 予定

- Nrealって何？(5分)
- SDKインポート(5分)
- Nreal仮想コントローラの操作習熟(10分)
- 画像認識Onエミュレータの構築(15分)
- 画像ライブラリの作成(5分)
- ARマーカーの話(20分)
- 何か作る(60分)

# 最初に

本ハンズオンは少人数で行っている(はず)ので、随時質疑応答承ります。
**どんどん発言してください。**

発表者は、職業プログラマではないので、マサカリ投げ放題です。
**どんどん投げてください。**

発表者は、発表時間超過の常習犯です。
予定の時間配分を**超過しても許して**ください。

# Nrealって何

## 概要

軽量(88g)、低価格(Consumer Kit $499)のARグラス
視野角は52°
見た目もいい意味で普通
https://www.nreal.ai/

※Consumer Kitは2020年前半発売予定(早く出て)

# ソフトウェア

## SDK

Unity 2018.2.x以降対応
Unreal Engine, Android Native リリース予定

## 開発

Android SDK 8.0(≒Android OS 8)以上が必要
Android向けにビルド(apkで出力)

# 対応する機能1

| 機能 | NReal SDK | AR Foundation(ARCore) |
| --- | --- | --- |
| 6DoFトラッキング | デュアルカメラとIMUによるSLAM | シングルカメラIMUによるSLAM |
| ポイントクラウド | リアルタイムマッピング 3D点群作成(アクセス可能) | 3D点群作成(アクセス可能) |
| 平面認識 | 床、壁 | 床、壁 |
| 画像追跡 | 同時追跡数1 | 同時追跡数自由 |

※ IMU: 加速度センサー、ジャイロセンサーをまとめたもの

# 対応する機能2

| 機能 | NReal SDK | AR Foundation(ARCore) |
| --- | --- | --- |
| Android OS | 8以上 | 7以上 |
| レイキャスト | 視線レイキャスト、通常レイキャスト(画面タッチ) | AR レイキャスト(画面タッチ) |
| 共有 | Android相手のみ | ×(ARCoreを直接使う必要あり) |
| テスト | Unity Editor | 実機 |

# SDKインポート

## インポート

Nreal SDKはUnity Packageとして配布されているので、
ダウンロードして、Unityのプロジェクトにドロップすれば、OKです。

# Unsafe許可

Nreal SDKは内部でUnsafeコードを使っています。
今の状態では動かないので、Unsafeを許可します。

## アセンブリ定義

このSDKのためだけに、プロジェクト全体をUnsafe許可にするのは嫌なので、
**アセンブリ定義を用いて、Nreal SDK内部でだけUnsafeを許可**します。

**Editor拡張はビルドに含まれるとビルドエラーが出る**そうなので、
NRSDK/Editor下にも別で生成します。

アセンブリ定義については、以下の記事がおすすめ
https://qiita.com/toRisouP/items/d206af3029c7d80326ed

# Nreal仮想コントローラの操作習熟

まずUnity EditorのPlay Modeでの操作に慣れます。

## 移動

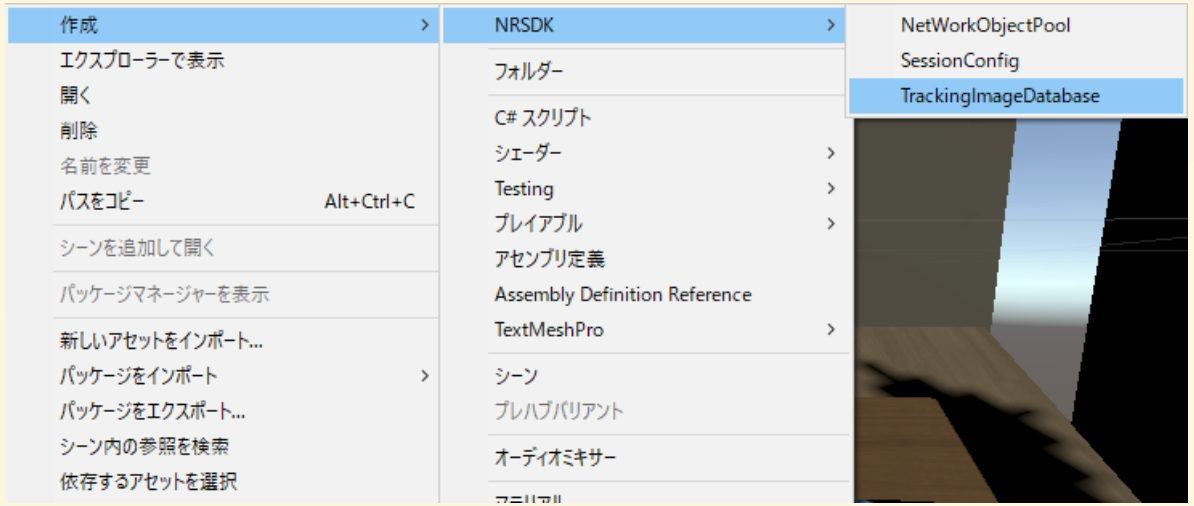
| キー | 動作 |
| --- | --- |
| w | (グラス)奥へ進む |
| s | (グラス)手前に戻る |
| a | (グラス)左へ移動 |
| d | (グラス)右へ移動 |
| space + マウス移動 | (グラス)視点を動かす(3DoF) |

# コントローラ操作

| キー | 動作 |
| --- | --- |
| ↑ | (コントローラー)上方向にスワイプ |
| ↓ | (コントローラー)下方向にスワイプ |
| ← | (コントローラー)左方向にスワイプ |
| → | (コントローラー)右方向にスワイプ |
| 左クリック | (コントローラー)決定 |
| 右クリック | (コントローラー)キャンセル |

# 画像ライブラリの作成(5分)

エミュレータに含めたい画像を**すべて**選択してから、右クリックして、TrackingImageDatabaseを生成します。




画像を選択していない状態では、TrackingImageDatabaseが選択できません。(ハマった)
あとからTrackingImageDatabaseの登録画像数を増やすことはできません。入れ替えはできます。

# 使用する画像ライブラリの設定

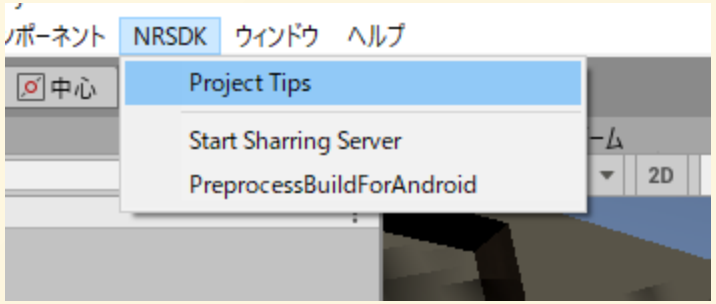
右クリック -> NRSDK -> SessionConfig
でSessionConfigを生成して、

- ImageTrackingModeをEnableに設定
- TrackingImageDatabaseの項目に使いたい物を設定

# 画像認識Onエミュレータの構築(15分)

## ビルド設定

ビルド設定は画像のProject Tipsを開いて、すべて適用すればOKです。




詳細なビルド設定は以下のリンク先を読んでください。
https://developer.nreal.ai/develop/unity/android-quickstart

# Image Tracking Emulate

画像を置きたい場所に、Emulator/Prefab内のNRTrackableImageTargetを置きます。

画像認識用エミュレート機能なので、実際にビルドするときは、Disableにしておくとよい(はず)です。

## 注意点

エミュレータ上で動かす場合、ARマーカーがどんなに低品質でも必ず認識します。

ARマーカーを登録するときは、品質に注意しましょう。

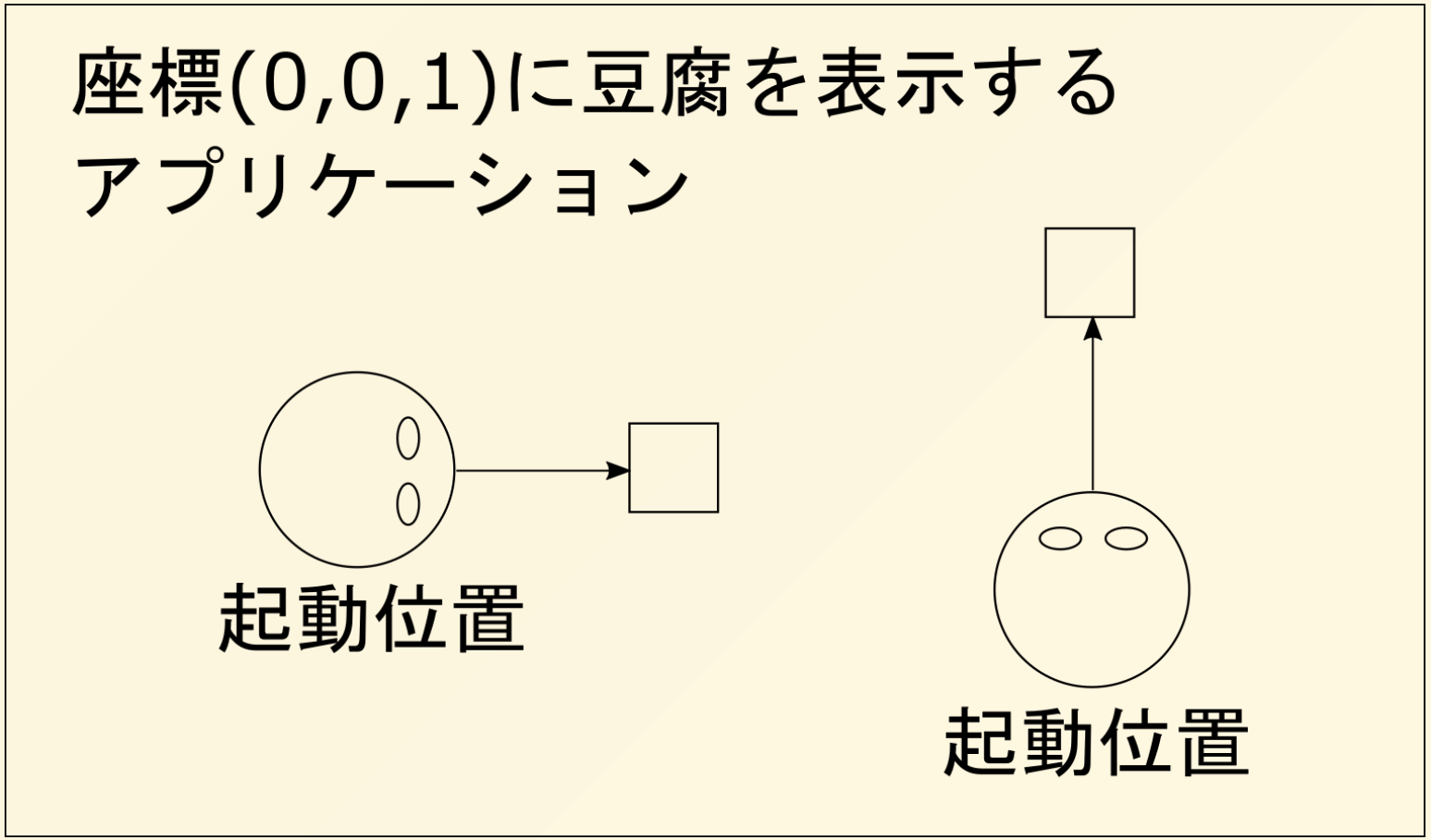
# ARマーカーの話

## ARマーカーがなぜいるのか(ざっくり)

ARアプリケーションは起動したときのスマートフォンの位置を原点(0,0,0)に設定する
-> 起動した位置でオブジェクトと実空間の位置関係が変わる
-> 実空間に基点(ARマーカー)を設定すれば、オブジェクトと実空間の位置関係が変わらない

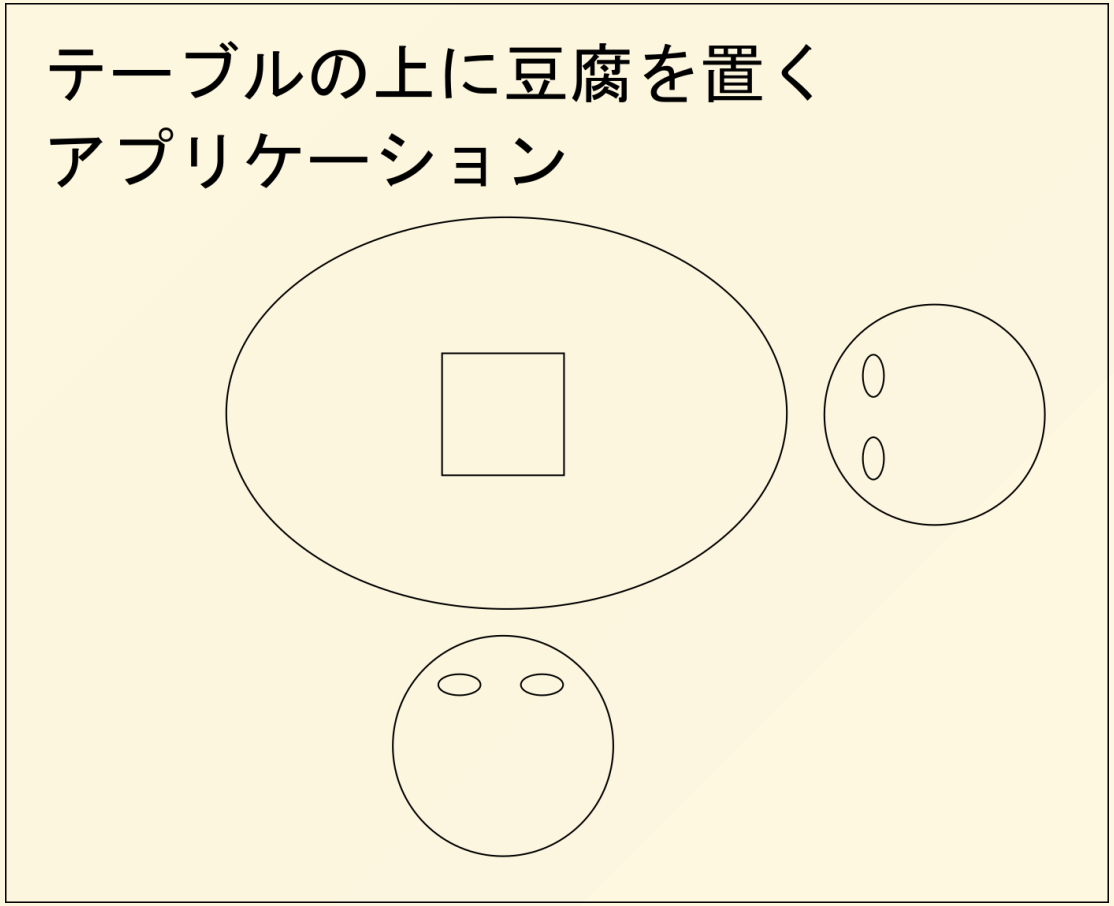
# ARマーカーを使わない場合

表示するオブジェクトの位置が現実空間と関係ない時



# ARマーカーを使う場合

表示するオブジェクトの位置が現実空間と関係ある時




**常に実空間の同じ場所に表示したいとき**基点としてARマーカーを使う

# ARマーカーを使う上での注意点

ARマーカーを登録したからと言って、必ず認識されるものではない

認識しづらいマーカーだと、認識位置にノイズが混ざりやすくなる

TODO:動画再生

# 基本的な画像認識の仕組み

## 特徴点が集中する場所

特徴点は

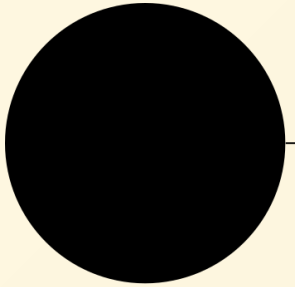
- 輪郭(こっちが主)
- 何も検出できない点

に生成される

この記事がおすすめ
https://qiita.com/icoxfog417/items/adbbf445d357c924b8fc

# 輪郭検出

照度の急激な変化を輪郭として認識する
-> コントラスト超重要



照度の変化量が大きい
≒ここが輪郭



照度の変化量が小さい
≒ここは輪郭？

## 色はそんなに重要じゃない

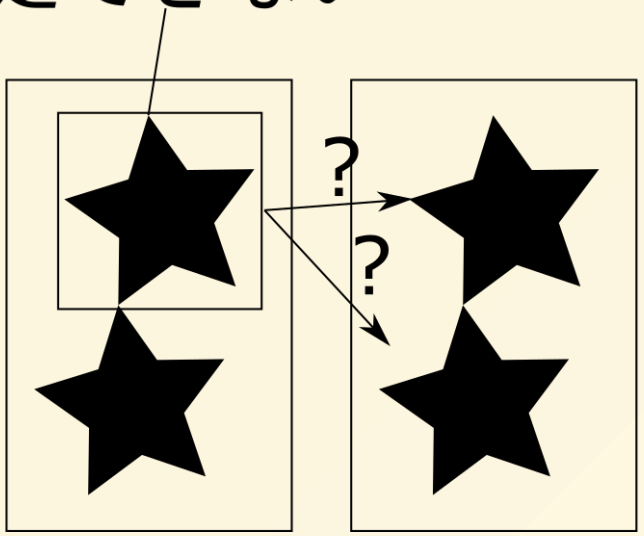
画像処理において、照度だけ使いたいとき、
グレースケール変換するのはよくある手法 [**独自研究**] [**要出典**]

例:
既定の画像を新しい画像から探すOpenCVチュートリアル
カラーでも動作するが、グレースケール化されている
https://docs.opencv.org/4.3.0/d7/dff/tutorial_feature_homography.html
https://docs.opencv.org/4.3.0/db/d70/tutorial_akaze_matching.html

# 類似点が多いと特定しづらい1

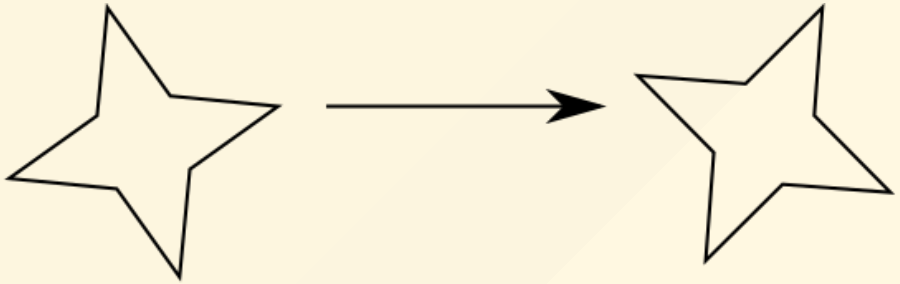
自己類似性の高い画像(パターン柄など)だと、部位の特定ができず、
画像認識に必要な情報量が増える

この部分の特徴点だけだと
特定できない



## 類似点が多いと特定しづらい2

対象性が高い(正4角形など)と、向きがわからず、
画像認識に必要な情報量が増える



何°回転しているのかわからない
≒向きがわからない

# 認識しやすいARマーカー(データ)

- ハイコントラスト(輪郭が検出しやすい)
- パターン柄でない(同じものを繰り返すと部分の特定できない)
- 対称性が低い(向きがわかりやすい)
- 他のARマーカーとの違いが明確(別の画像として認識しないよう)

# 認識しやすいARマーカー(物理)

- 変形しない(ゆがむと認識しづらい)
- 正面から認識する
- 光を反射しにくい紙質(反射光で画像が潰れる)
- 印刷サイズが大きい(サイズ指定されている場合を除く)
- 移動しない(移動物は位置情報の更新がかかる)

# 認識しにくいARマーカー

(認識しやすいARマーカーの逆は省略)

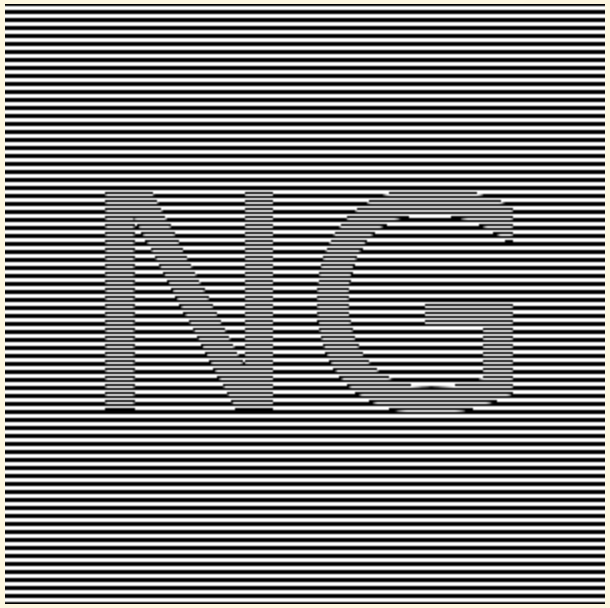
- すかすかな画像(特徴点が少ない)
- 高圧縮のjpgファイル(画像の輪郭がぼやける)

# ARマーカーの認識にあまり影響しない点

- 大きい画像サイズ(必要な情報量以上のものは、あまりパフォーマンスに影響しない)
- 色数の多さ(特徴点検出は主に輪郭に依存する ≒ コントラスト超重要)

# ダメなARマーカー集

# 最低得点(せっかく作ったから)




| ARCore | Nreal |
| --- | --- |
| 特徴点不足 | 31~35 |

パターン柄と、輪郭の取りにくさが敗北の決め手

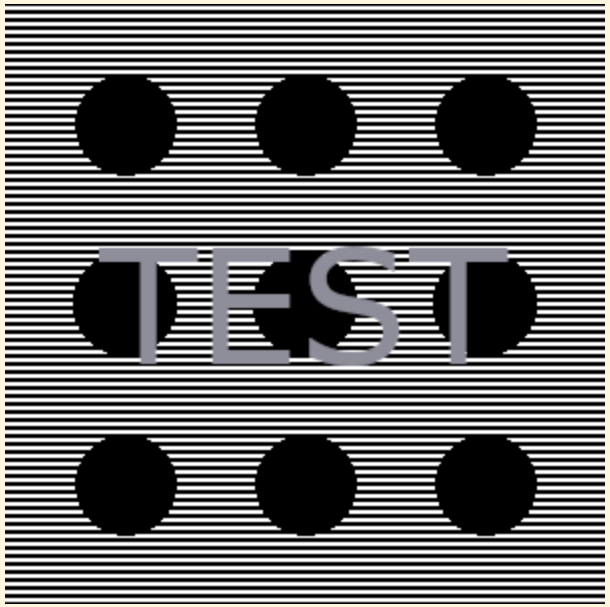
# 白紙



| ARCore | Nreal |
| --- | --- |
| 特徴点不足 | 40 |

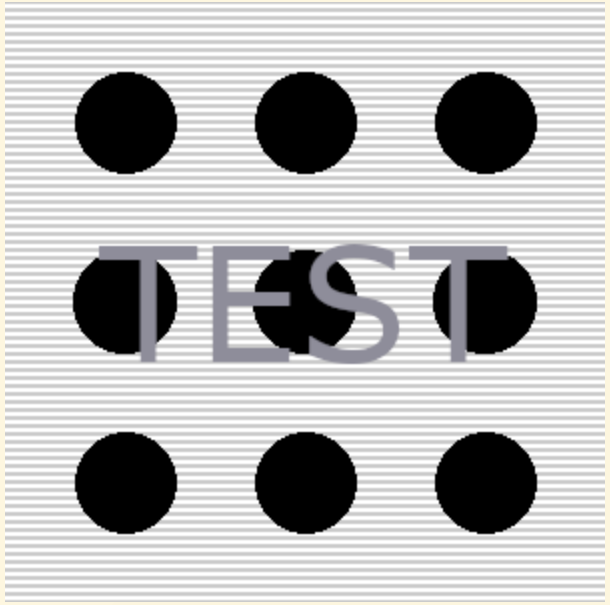
白紙回答 論外

# パターンの濃さ



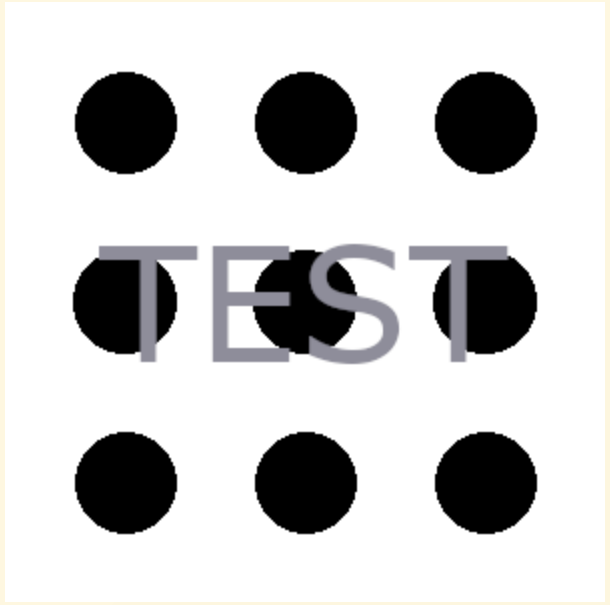

| ARCore | Nreal |
| --- | --- |
| 0 | 49 |

文字が入っている分加点




| ARCore | Nreal |
| --- | --- |
| 20 | 57 |

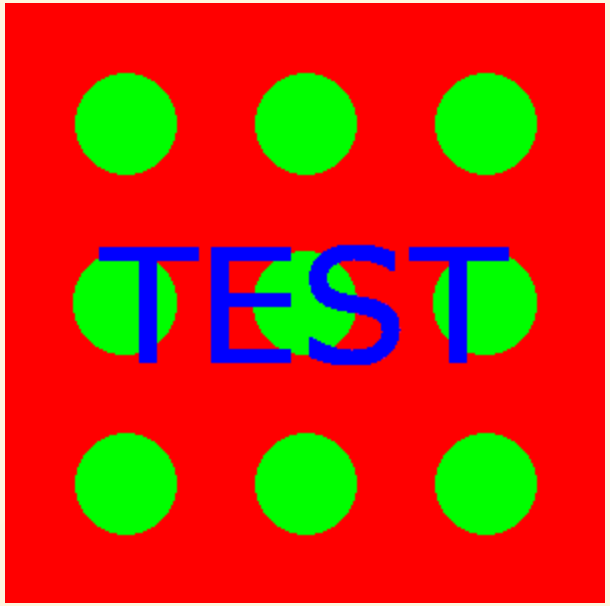
パターン柄を薄くしたもの




| ARCore | Nreal |
| --- | --- |
| 80 | 47 |

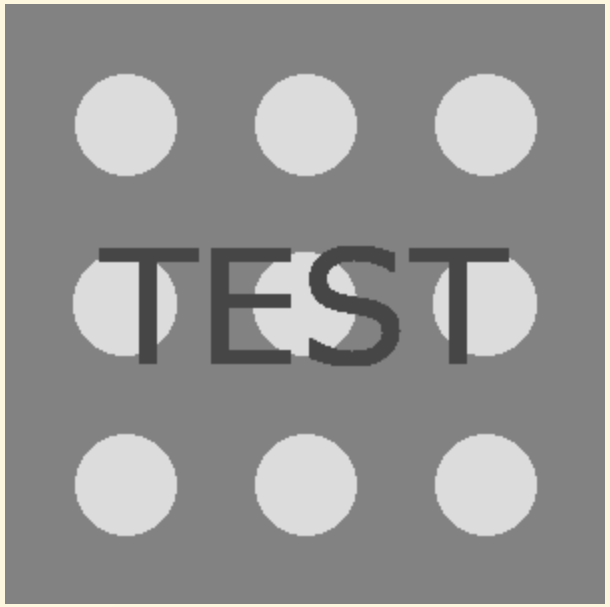
ARCore的には余分なパターン柄の背景は低評価の模様

# 着色




| ARCore | Nreal |
|---|---|
| 0 | 48 |

目に悪い原色系。グレースケールにすると輪郭が取りにくくなる




| ARCore | Nreal |
| --- | --- |
| 0 | 48 |

カラー版と変化なし

# 何か作る(60分)