影を用いたインタラクティブスクリーンの実装

NITIC Shunta Hayashi

October 27, 2025

Figure 1: 影で遊ぶ子供たち

1　概要

本プロジェクトでは、ユーザーが自身の影を利用してブロック崩しを操作するインタラクティブスクリーンを Processing で開発した。本システムは、以下の三つの主要機能から構成される。

1. **キャリブレーション**: カメラ映像からスクリーン領域のみを抽出するため、ユーザーが指定した四点に基づきカメラの補正を行う。
2. **影の検出**: 補正済み映像から影を抽出し、それを物理演算用の点群データに変換する。
3. **物理シミュレーション**: 抽出された影の点群と画面上のオブジェクト（`Cannonball`）との衝突や反発を計算する。

本稿では、上記各機能のアルゴリズム及び実装手法について概説する。

2　簡易的な射影変換手法

本システムにおける座標変換は、厳密なホモグラフィ行列による射影変換ではなく、**四辺形補間（Quadrilateral Interpolation）**と呼ばれる手法を採用している。本手法は、カメラ座標系における任意の四辺形をスクリーン座標系の四辺形にマッピングする処理である。

2.1　キャリブレーション（四点取得）

プログラム実行時、ユーザーはスクリーン投影面の四隅を選択する。この四点は、カメラ映像からスクリーン領域のみを取得するための補正計算に使用される。

2.2　座標マッピングの事前計算

主要な処理は、別スレッドで実行される `calculateIntersections()` 関数にて行われる。本関数の目的は、スクリーン座標 (i, j) に対応するカメラ座標 (x, y) を計算し、その結果をルックアップテーブル（`intersection[height][width]`）に格納することである。

スクリーン座標を $[0, 1]$ に正規化したパラメータ u（水平）および v（垂直）を用いて、四隅 P_0（左上）、P_1（右上）、P_2（右下）、P_3（左下）間を補間する。

$$
\begin{aligned}
P_{\text{top}}(u) &= (1-u)P_0 + uP_1 \\
P_{\text{bottom}}(u) &= (1-u)P_3 + uP_2 \\
P(u,v) &= (1-v)P_{\text{top}}(u) + vP_{\text{bottom}}(u)
\end{aligned}
$$

この $P(u, v)$ がスクリーン座標 (i, j) に対応するカメラ座標 (x, y) である。

2.3　歪み補正画像の生成

`runMainInteraction()` 内で `renderWarpedImage()` 関数が呼ばれ、前述の LUT を用いて歪み補正画像 `warpedImage` を生成する。画面の各ピクセル (i, j) について、対応するカメラ座標 (x, y) の色情報をコピーすることで補正を行う。

3　影を当たり判定に変換する手法

影の検出及び物理演算への適用は二段階で行われる。

3.1　影の検出（画像処理）

`detectDarkPoints()` 関数により、補正済み画像 `warpedImage` から影を検出する。処理負荷軽減のため、`PIXEL_SKIP` ピクセル間隔で走査し、輝度が `BRIGHTNESS_THRESHOLD` 以下の点を `motionPoints` に追加する。また、1 フレームあたりの最大点数は `MAX_POINTS_PER_FRAME` により制限される。

3.2　衝突判定（物理演算）

`Cannonball` クラスの `checkCollision()` 関数により、影の点群に対する衝突判定を行う。ボール中心から各点までの距離を計算し、ボール半径と影点半径の和より短ければ衝突と判定する。衝突した点群の平均座標 $\vec{p}_{\text{avg}}$ を計算し、ボール中心 $\vec{p}_{\text{pos}}$ とのベクトル差を反発方向として速度に加算する。

$$
\vec{v}_{\text{ball}} \leftarrow \vec{v}_{\text{ball}} + \text{repulsionStrength} \cdot \frac{\vec{p}_{\text{pos}} - \vec{p}_{\text{avg}}}{\|\vec{p}_{\text{pos}} - \vec{p}_{\text{avg}}\|}
$$

これにより、ボールは影の塊から外向きに反発する運動を示す。

4　物理演算の安定化

高速移動時にオブジェクトや影を通過してしまうトンネリング現象を防ぐため、1 フレーム内で物理演算を複数回に分割して計算する。具体的にはサブステップ数 `subSteps` を設定し、$dt = 1/\text{subSteps}$ で時間積分を行う。

$$
\text{int subSteps} = 5; \quad \text{float dt} = 1.0/(\text{float})\text{subSteps};
$$

これにより、衝突判定の精度を向上させ、薄いオブジェクトや小規模な影の点も正確に検出可能となる。