第6回　クリエイティブ・コーディング入門

外部ライブラリの利用（画像認識による生体計測）

今日の内容

- 外部ライブラリの利用
- MediaPipeを使った手の位置情報を取得
- 応用例：手の位置情報を用いた表現
  - パーティクルの生成
  - 距離を使った描画の制御
- その他の生体情報の検出例

外部ライブラリの利用

CDNで外部ライブラリを利用する

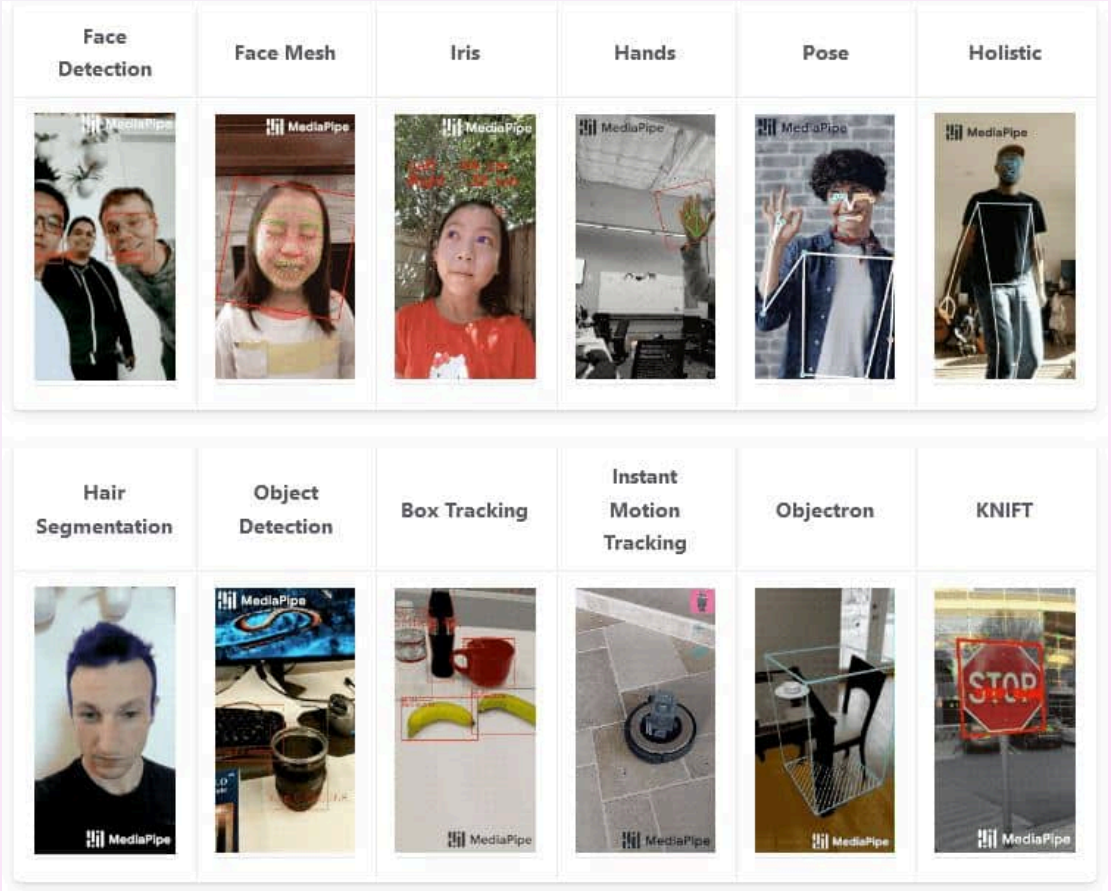
- htmlのscriptタグでCDNから読み込むことが可能
- p5.jsもCDNで読み込まれている

```
<head>

<script src="https://cdn.jsdelivr.net/npm/・・・/p5.js"></script>
<script src="https://cdn.jsdelivr.net/npm/・・・/p5.sound.min.js"></script>

</head>
```

MediaPipe

- Google提供の機械学習ライブラリ
- CDNで比較的簡単に利用できる
- 映像から生体情報を取得
- 音声認識などもできる

参考：MediaPipeソリューションズ



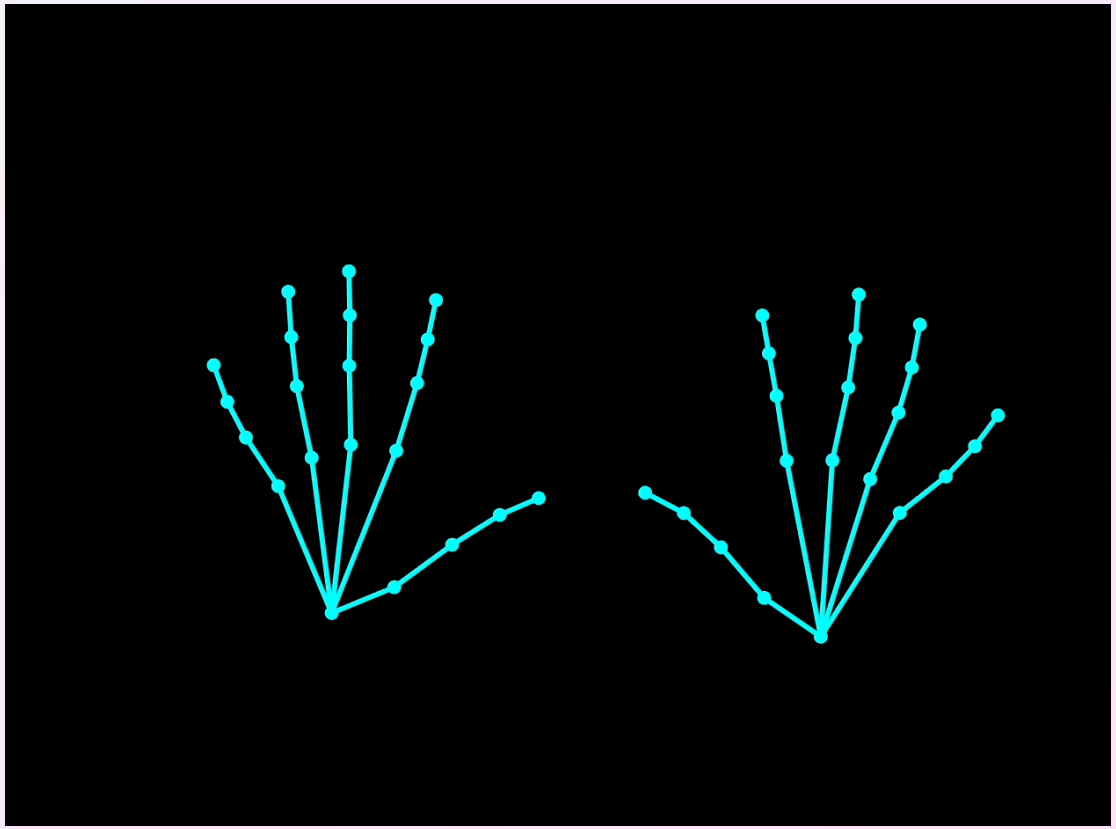
MediaPipeを使った生体計測

- 手や顔、姿勢などを複数の点（点群）として取得
- 座標情報（landmarks）が主な出力
- 計測や可視化表現に向いている
- 今回は主に「手の計測」を扱う
  - 姿勢や顔の計測もほぼ同じようにできる

MediaPipeを使った手の検出

手の位置情報を取得

サンプルコード



MediaPipeを使う手順

- `createCapture()` でカメラ映像を取得する
- `Hands` クラスを使って手の検出モデルを初期化する
- `onResults()` で認識結果を `handsRes` に保存する
- `async/await` による非同期処理で `hands.send()` を繰り返し実行する
- `draw()` 関数で手の位置情報を使った描画を行う

index.htmlでライブラリを読み込む

- Hands用ライブラリを追加すると `Hands` クラスが使える
- p5.js → MediaPipe → `sketch.js` の順で読み込む
- 読み込み順を間違えるとクラスが未定義になる

```
<script src="https://cdn.jsdelivr.net/npm/p5@1.11.11/lib/p5.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@mediapipe/hands@0.4.1675469240/hands.js"></script>
<script src="sketch.js"></script>
```

グローバル変数を用意

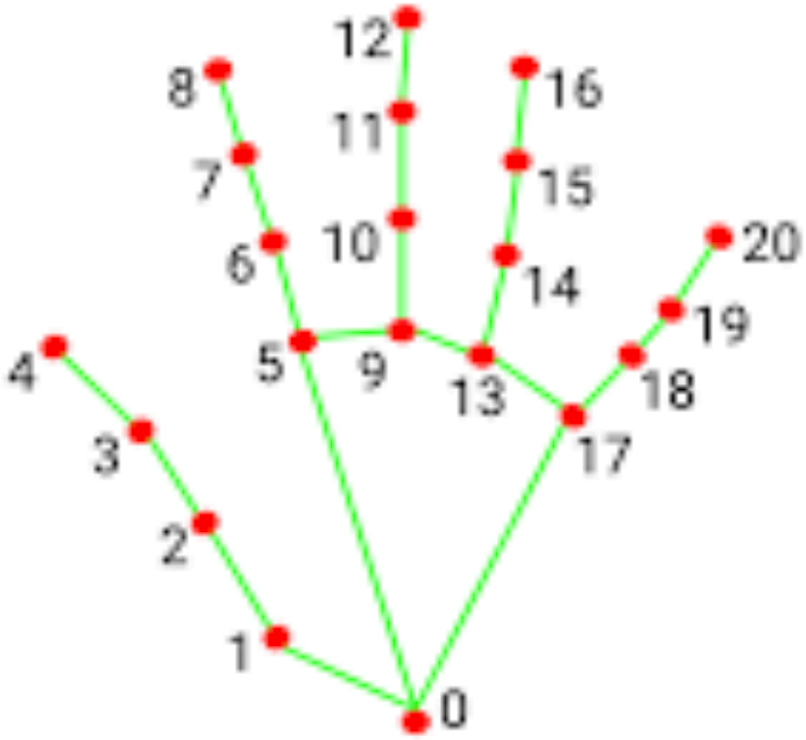
- インスタンス用の変数や配列を用意
- `processing` は非同期処理の重複実行を防ぐために用意

```
let cam; //カメラ用のインスタンス変数
let hands; //MediaPipe Hands用のインスタンス変数
let handsRes = null; // 毎フレーム更新される認識結果を保存する
let processing = false;
```

手の関節の番号

- `MULTI_HAND_LANDMARKS` に配列番号で手の関節の位置情報が格納される
- 関節を `line` で繋げるための配列を用意

```
//手の形状をlineでつなげる配列を用意
const HAND_CONNECTIONS = [
  [0,1],[1,2],[2,3],[3,4],
  [0,5],[5,6],[6,7],[7,8],
  [0,9],[9,10],[10,11],[11,12],
  [0,13],[13,14],[14,15],[15,16],
  [0,17],[17,18],[18,19],[19,20],
];
```



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

setup関数でのカメラの準備

- `setup()` 内で `createCapture()` を呼びカメラを起動する
  - `{ flipped: true }` で鏡のような左右反転映像にする
- `cam.size()` でカメラ映像と `canvas` サイズを揃える
- `cam.hide()` でHTMLのvideo表示を非表示にする

```
cam = createCapture(VIDEO, { flipped: true });
cam.size(640, 480);
cam.hide();
```

setup関数でのHandsのセットアップ

- インスタンスにモデルファイルを読み込む
- `setOptions()` で最大手数や検出精度を設定する

```
hands = new Hands({ locateFile: (file) => base + file });
hands.setOptions({
  maxNumHands: 2, //検出可能な最大手数
  modelComplexity: 1, //モデル精度（0:軽量〜2:高精度）
  minDetectionConfidence: 0.5, // 手を検出する最低信頼度
  minTrackingConfidence: 0.5, // 検出後の手を追跡する最低信頼度
  selfieMode: true, //セルフィーモードを設定（左右反転）
});
```

setup関数でイベント処理を登録する

- 推論完了時に結果を受け取る処理を登録する
- カメラ準備完了後に推論ループを開始する処理を登録する

```
hands.onResults(function (res) {          // 推論完了時に呼ばれる処理を登録
  handsRes = res;                         // 最新の推論結果を保存
});

cam.elt.onloadedmetadata = function () { // カメラ準備完了後の処理を登録
  requestAnimationFrame(processFrame);   // 推論ループを開始
};
```

asyncを使って非同期処理にする

- `hands.send()` の推論に時間がかかるため、`await` で完了まで待機させる
- 描画ループ（p5.js）と推論処理を安全に同期させるための処理
- 重い処理でも画面更新が止まらないようにする

```
async function processFrame() {
  if (!cam?.elt || processing) {          // カメラ未準備 or 推論中なら
    requestAnimationFrame(processFrame);  // 次フレームだけ予約して
    return;                               // 今回の処理は行わない
  }
  processing = true;                      // 推論中フラグを立てる
  await hands.send({ image: cam.elt });   // 推論完了まで待機（非同期）
  processing = false;                     // 推論終了 → フラグ解除
  requestAnimationFrame(processFrame);    // 次のフレームを処理
}
```

draw関数で背景とカメラの映像を用意

- `background()` で毎フレーム背景をクリアする
- `image(cam, ...)` でカメラ映像をキャンバス全体に描画する
  - `image(img, x, y, width, height);`
  - 表示位置（x, y）とサイズ（width, height）を指定できる

```
background(0);
//カメラの映像を表示（コメントアウトでオフに）
image(cam, 0, 0, width, height);
```

draw関数で推論結果の有無をチェック

- 推論結果がまだ届いていない場合は以降の描画を行わない
- ランドマーク情報がない場合も処理をスキップする

```
// 推論結果が未取得、または手が検出されていない場合は処理をスキップ
if (
  handsRes === null ||
  handsRes.multiHandLandmarks === null
) {
  return;
}
```

draw関数で手のランドマークを描画

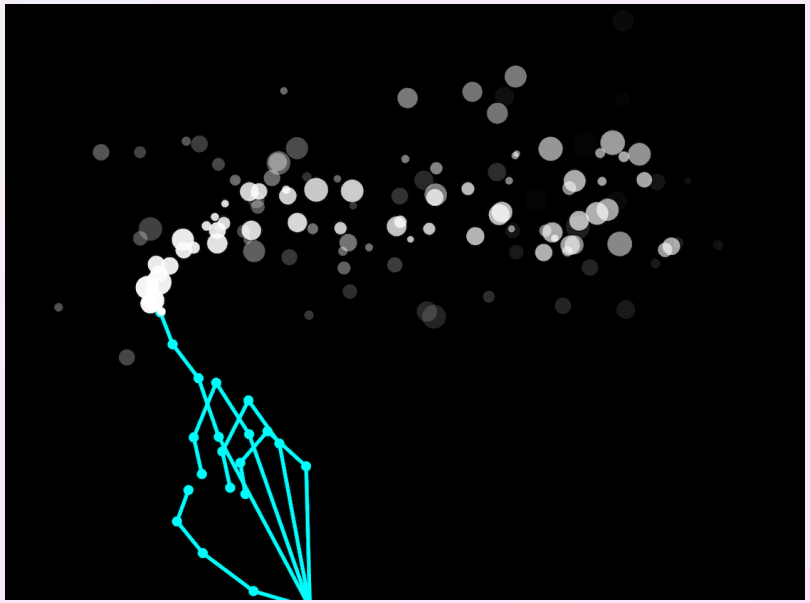
- `handsRes.multiHandLandmarks` に手のランドマーク（関節点）が配列で格納されている（例：`lm[0].x` = 手首の x 座標）
- 値は0〜1に正規化されているため，`width` と `height` を掛けてcanvas座標に変換

```
for (let i = 0; i < handsRes.multiHandLandmarks.length; i++) {
  let lm = handsRes.multiHandLandmarks[i];
  stroke(0, 255, 255);
  strokeWeight(3);
  for (const [a, b] of HAND_CONNECTIONS) {
    line(lm[a].x * width, lm[a].y * height, lm[b].x * width, lm[b].y * height);
  }
  noStroke();
  fill(0, 255, 255);
  for (const p of lm) {
    circle(p.x * width, p.y * height, 8);
  }
}
```

応用：手の動きを使って描画を制御

応用①：手の位置情報を使ってパーティクルを描画

サンプルコード



指の位置からパーティクルを生成する

- Particleクラスをコピーして、手の位置情報を使ってパーティクルを生成

```
for (let i = 0; i < handsRes.multiHandLandmarks.length; i++) {
  let lm = handsRes.multiHandLandmarks[i];
  /* 中略 */
  //人差し指(lm[8])の位置を使ってパーティクルを生成
  particles.push(new Particle(lm[8].x*width, lm[8].y*height));
}
```

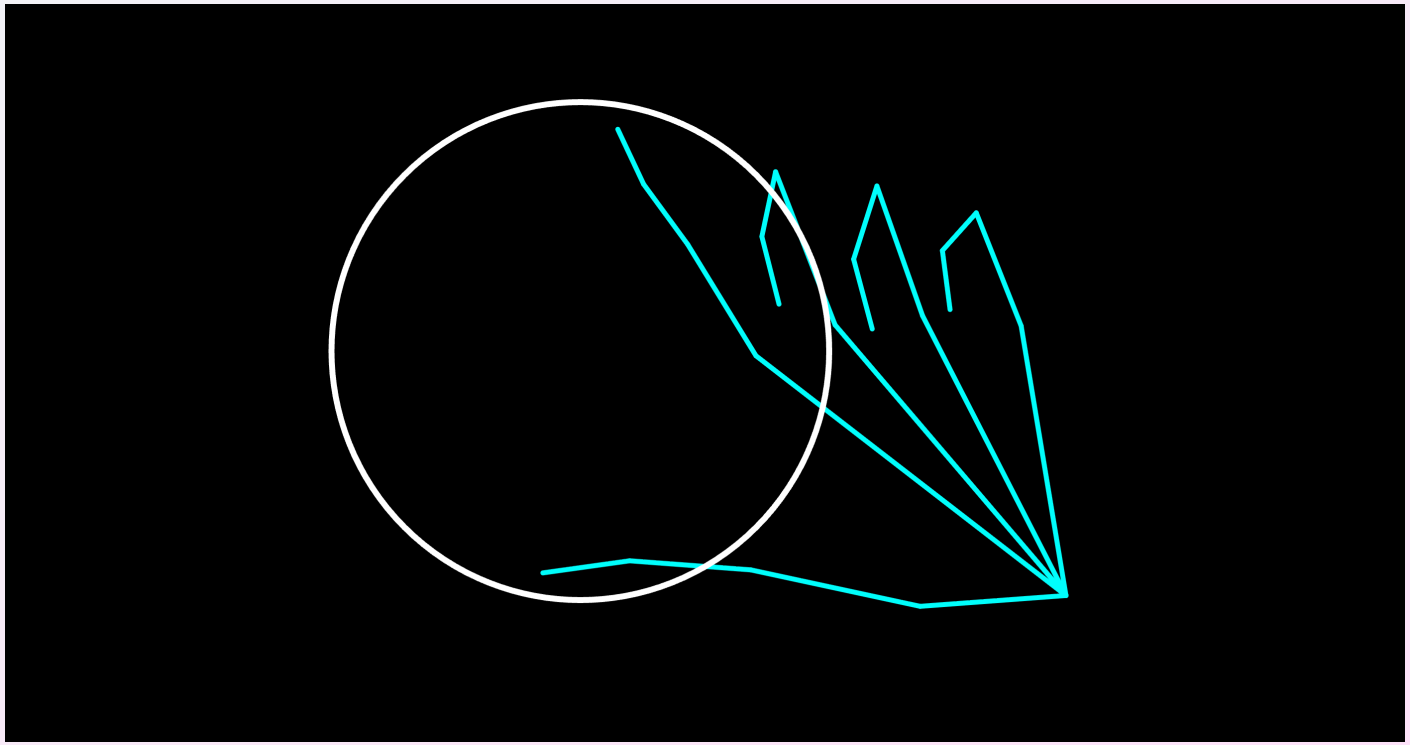
フルスクリーンモードに設定（サンプルコード）

- `canvas(windowWidth, windowHeight)` でキャンバスをウィンドウサイズに
- 描画位置は `lm[a].x * width` などで自動的にキャンバスサイズに調整される

```
function windowResized(){
  resizeCanvas(windowWidth, windowHeight);
}
// fキーでフルスクリーンの切り替えをできるようにする
function keyPressed(){
  if(key == "f"){
    let fs = fullscreen();
    fullscreen(!fs);
  }
}
```

応用②：dist関数による距離測定による描画

サンプルコード



人差し指と親指の距離を測って円のサイズを

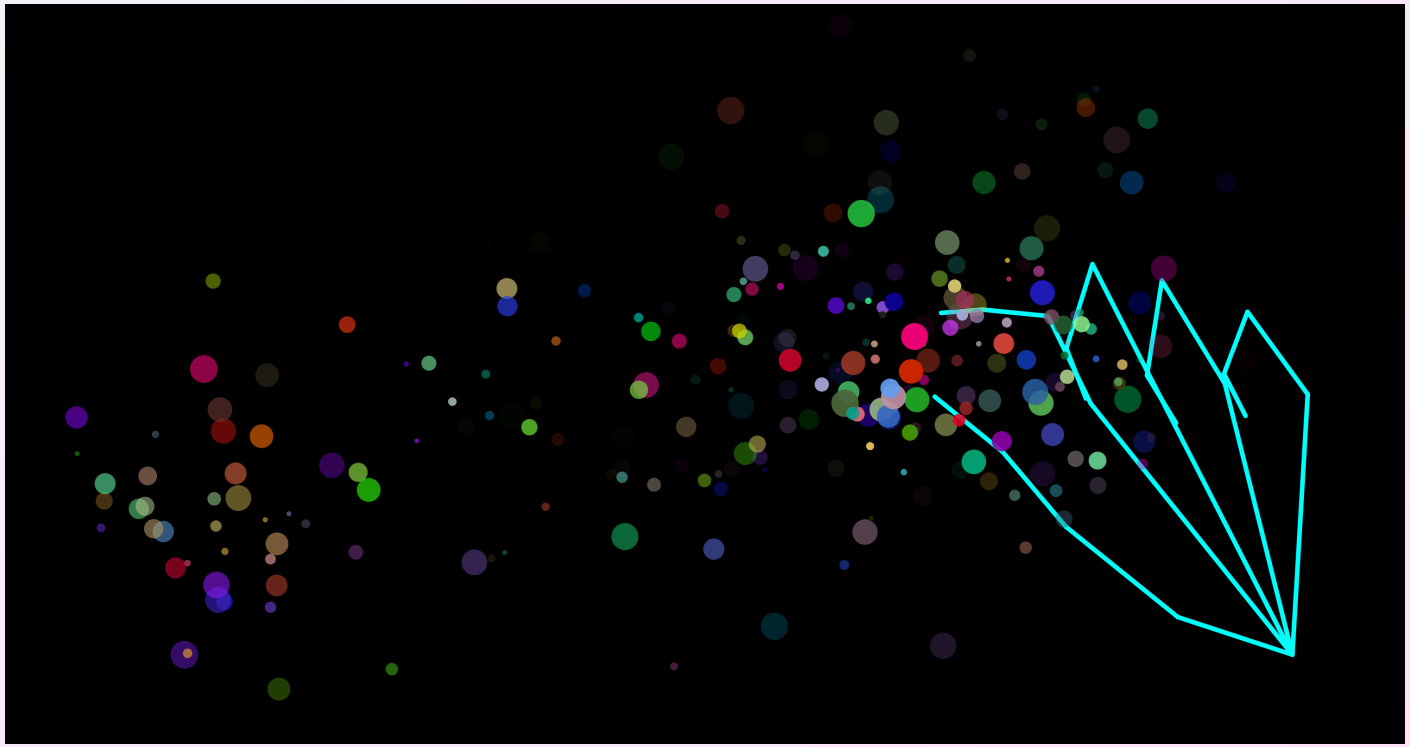
- 二点間の距離を `dist関数` で取得して、 `map関数` で円のサイズに反映

```
const a = lm[4]; // 親指の先端（THUMB_TIP）
const b = lm[8]; //人差し指の先端（INDEX_FINGER_TIP）
const ax = a.x * width;
const ay = a.y * height;
const bx = b.x * width;
const by = b.y * height;
const d = dist(ax, ay, bx, by);
let targetSize = map(d, 20, 180, 10, 500);
targetSize = constrain(targetSize, 10, 500);
smoothSize[i] = lerp(smoothSize[i], targetSize, 0.25);
const cx = (ax + bx) * 0.5;
const cy = (ay + by) * 0.5;
ellipse(cx, cy, smoothSize[i]);
```

指を摘んでいるときだけパーティクルを生成する（コード)

- `dist` で指の間隔が狭い状態の時だけパーティクルを生成するようにする

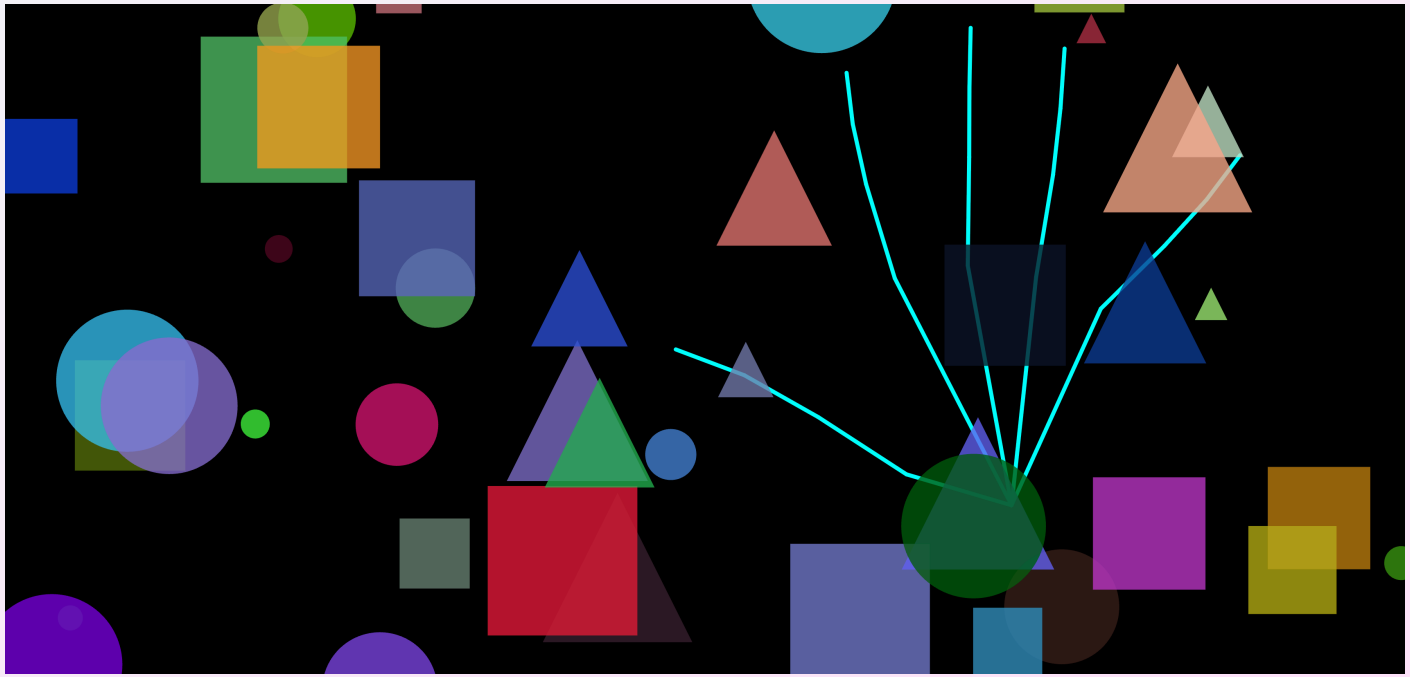
```
if(d < 150){particles.push(new Particle(cx, cy, d));}
```



指の開き具合でパーティクルのサイズを調整する（コード)

- `dist` を `map` でオブジェクトの倍率として利用

```
targetScale = map(d, 20, 180, 0.2, 10.0);
```



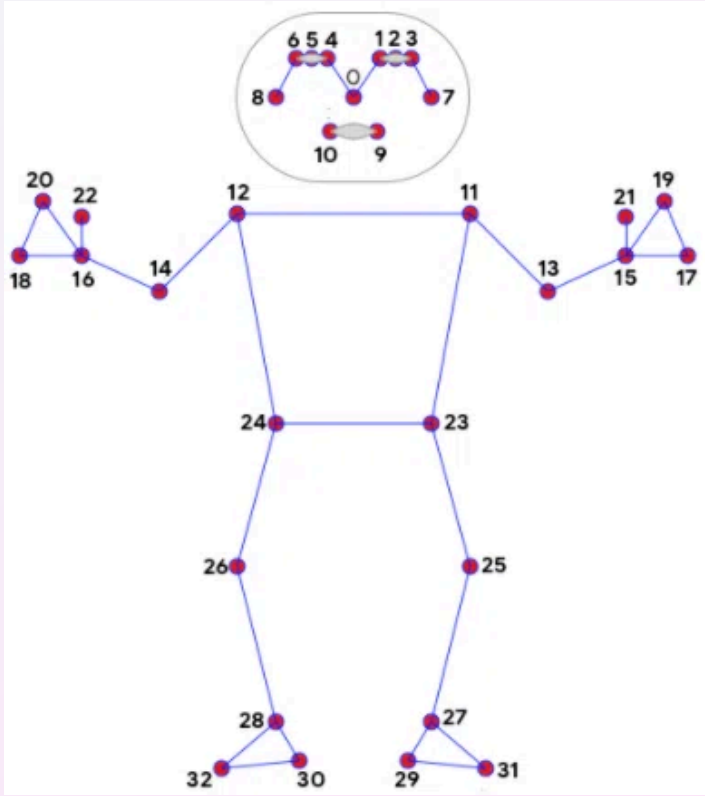
その他の生体情報の検出

姿勢と顔の検出

- `hand` 以外にも `pose` と `face` のモデルが利用できる
- 基本的には `hand` のものと同じ
- 姿勢情報の検出 (サンプルコード)
- 顔の検出(サンプルコード)

姿勢(Pose)のランドマーク(サンプルコード)

- `hands` と同じようにランドマークと配列番号が対応している
- 33個のランドマークが検出される



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

顔(Face)のランドマーク(サンプルコード)

- これも同様にランドマークと配列番号が対応
- 478個のランドマークが検出される
- 画像はランドマークを番号にしたもの
  - サンプルコード