卒 業 研 究

サーモグラフィからの３次元物体抽出及び利用方法の研究

指導教員　蒔苗　耕司

宮城大学事業構想学部　デザイン情報学科

メディアデザインコース　蒔苗研究室

21522100 栁沼　洋詞

2019 年 2 月

目次

第1章　序論

1.1　研究の背景

近年，わが国ではSociety5.0の一環として建設現場にICT技術を全面的に導入することで生産性向上を１つの目標とするi-Constructionという政策を掲げており，その中で建設現場における測量から設計、施工、維持管理に至る建設プロセス全体を３次元データで繋ぎ合わせることが生産性向上の要となっている．建設プロセスのうちの測量段階では現在写真測量から得たデータセットからSfM技術(Structure from Motion)により対象の３次元モデルを生成する．さらに，維持管理段階における点検作業では異常な熱から発生する不具合等の検査のために赤外線サーモグラフィを用いることがある．最近，SfM技術の向上により通常の写真との位置情報の重ね合わせ等をせず赤外線サーモグラフィから直接的に物体表面の温度情報を表す３次元モデルを生成することができるようになった．しかし，赤外線サーモグラフィの問題として対象物体と地面等との温度が近く識別が困難な場合があり，サーモグラフィから生じた３次元モデルも同様の視認性の問題が起こる．

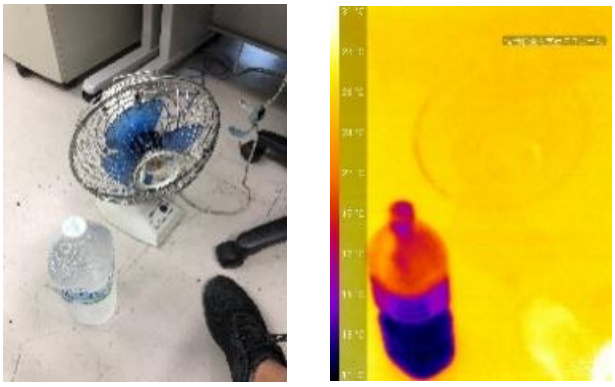


Fig. 1　左：通常の写真，右：サーモ

1.2　研究の目的

1.1で述べた背景を基に，本研究は，サーモグラフィから生成した３次元モデルで起こることがある視認性の問題に対する解決案と生産性向上に関する方法を研究する．具体的には，サーモグラフィにおける対象物体の抽出，リアルの３次元モデルへのサーモグラフィ（IRT３次元モデルと呼ぶ）からの３次元モデルの重畳，重畳されたIRT３次元モデルを利活用するシステムの開発の３つを本研究の方向性として進める．

第2章　SfM 技術の概要

2.1　Structure from Motion

Structure from Motion (SfM) は，一連の 2 次元イメージから 3 次元シーンの構造を推定するプロセスである．SfM は，3 次元スキャンや拡張現実など，多くのアプリケーションで使用されている．

2.2　写真測量ソフト「3DF Zephyr」における SfM アルゴリズム

既往の研究では，写真測量ソフト「3DF Zephyr」を用いてサーモグラフィから３次元モデルを復元することができることを明らかにした．本研究では，Zephyr を用いて３次元サーモグラフィモデルを復元する．Zephyr で使用されている最新の SfM 技術を述べる．

2.2.1　SAMANTHA

Samantha は，追加情報を入力することなく，自動的に写真が撮られた位置と方向を再現できる 3Dflow 社の SfM 技術である．技術コミュニティでは，最も効率的で進んでいるうちの 1 つとして認知されている．業績の一部が一流国際コンピューター画像認識学会にて発表されている．

2.2.2　STASIA

3DF Stasia は，3Dflow 社独自のアルゴリズムで，2D 画像セットから非常に高精度な高密度点群を作成する．コンピューター画像認識では，Stasia のプロセスはマルチビューステレオとして最もよく知られている．可能な限り最も高精度なモデル化を行うため，Samantha と密なデータ交信を行う．精度の良さが特徴であり，高密度点群作成のために，入力画像の全ピクセルを利用している．

2.2.3　SASHA

3df Sasha は，詳細部を有する高密度点群からメッシュを作成する 3Dflow 社独自のアルゴリズムである．サーフェスを作成する時，できるだけ詳細部を復元できることは重要である．Sasha は，3D モデルからシャープなエッジを取り出すことが出来る．それ故，建造物，測量，文化財の計測，市街地モニターなどの用途において，Sasha がより適している．

2.2.4 TEXTURING

このアルゴリズムでは，各ピクセルに最適なカラーを自動的に選択，設定する．これにより、異なる照明コンディションで撮られた写真からでも，貼り付けるテクスチャのカラーバランスを整えることができるようになった．

2.2.5 3DF MASQUERADE

このツールは,本ソフトのインストールパッケージに含まれた,外部実行可能ファイルとして開発された．Masquerade では，マスク操作時，なるべく操作時間を節約できるよう，スマートかつ手軽な方法で，画像エリアをマスクすることが可能である．

3DF Masquerade は，バックグラウンドノイズが多い時や，モデルの対象がバックグラウンドに対して不自然に動かされた時に役に立つ．最もありそうなシナリオは対象がターンテーブル上にある場合である．

第3章　実験環境

3.1　外部構成

本研究で復元した３次元モデルは，サーモカメラ（FLIR 社　FLIR One Pro），RGB デジタルカメラ（Apple 社　iPhoneX），HP pavilion 500，サーモカメラとして FLIR one Pro，サーモグラフィ表示媒体（HUAWEI 社　HUAWEI P20 lite），写真測量ソフトウェア（3DF 社　3DF Zephyr Pro 学生トライアル版），PC（HP Pavilion 500，OS Microsoft 10 64bit）により復元した．後述する温度範囲表示システムはゲーム開発エンジン Unity Pro により開発した．Fig. 2 はハードウェア製品の画像，ソフトウェア製品のロゴを示す．

Fig.3.1 上から左右順に Flir One Pro，HUAWEI P20 lite，iPhone X，Zephyr，Unity pro

3.2　　内部構成

Fig. ３は複数のデジタル写真（RGB 画像）から復元した RGB３次元モデルと複数のサーモグラフィ（IRT 画像）から復元した IRT３次元モデルの復元に至るデータフローと，IRT３次元モデルの抽出とそれの RGB３次元モデルへの重畳のデータフローを示す.

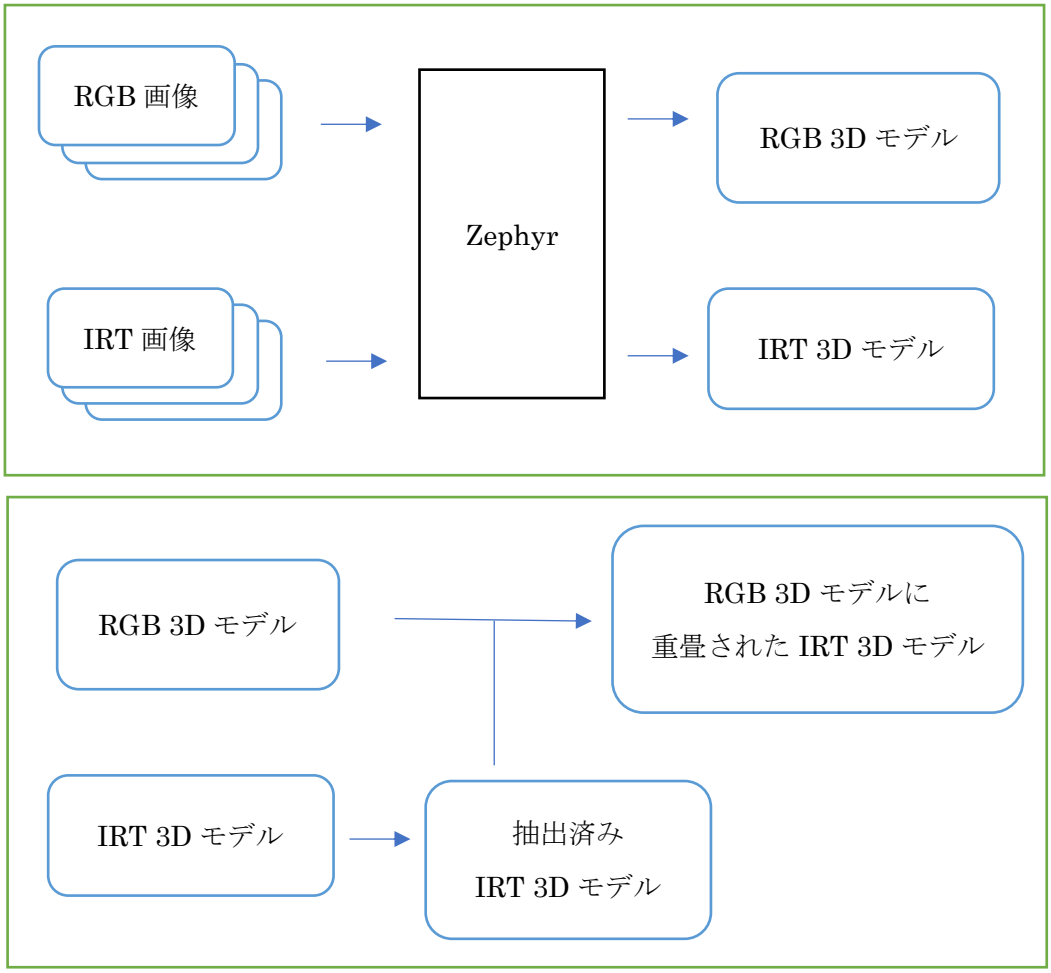


Fig. 3.2 データフロー

第4章　複数の画像から３次元モデルを復元する方法

本章では，SfM 技術により複数の画像から３次元モデルを復元する方法を詳細に述べる．本研究では，同研究室の友人から温度分布が豊かな人体を対象に３次元モデルで実験を行う．

4.1　RGB 画像からの RGB３次元モデルの復元方法

Zephyr には，動画入力方法があり，動画を入れると設定に従って画像に分割して PC のフォルダに入れる．生成された複数の画像を入力できるようになる．動画での入力の方がより効率的であるため，本実験では動画での入力を採用している．本節では，以下の手順から RGB３次元モデルの復元を行った．

1. iPhoneX カメラの動画機能を用いて人体の周りを一周して撮影する．
2. 本ソフトを立ち上げ，最初の画面から新規プロジェクトを開く．
3. 「カメラ方向付け後に高密度点群を計算」「高密度点群抽出後サーフェースを計算」「サーフェース抽出後にテクスチャを計算」にチェックを入れて，次へ．

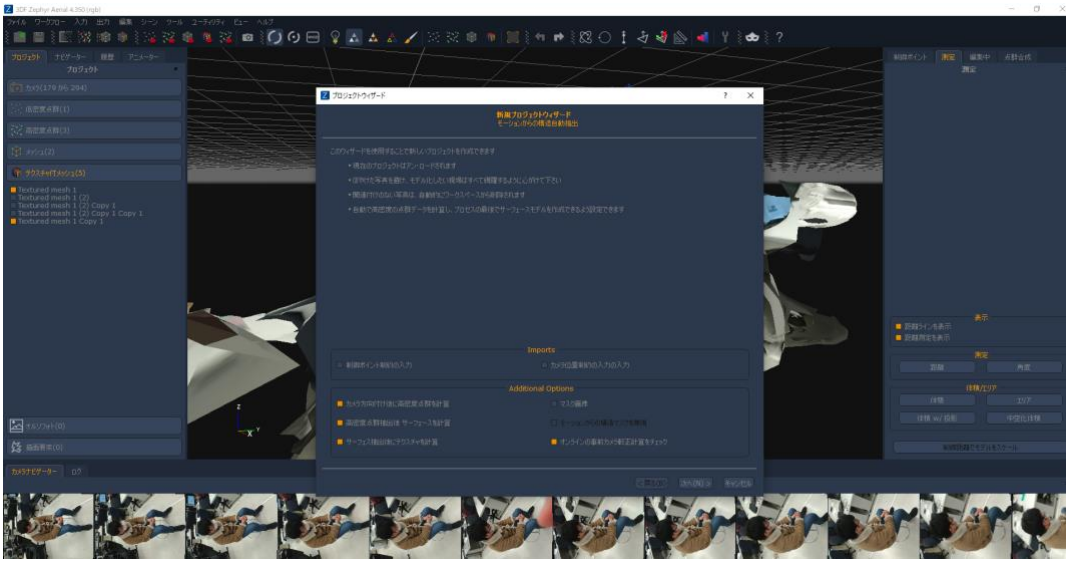


Fig.4.1-1 プロジェクト設定画面

4. 動画写真入力を行い，任意に設定し「フレームの抽出とワークスペースの入力」を行う．

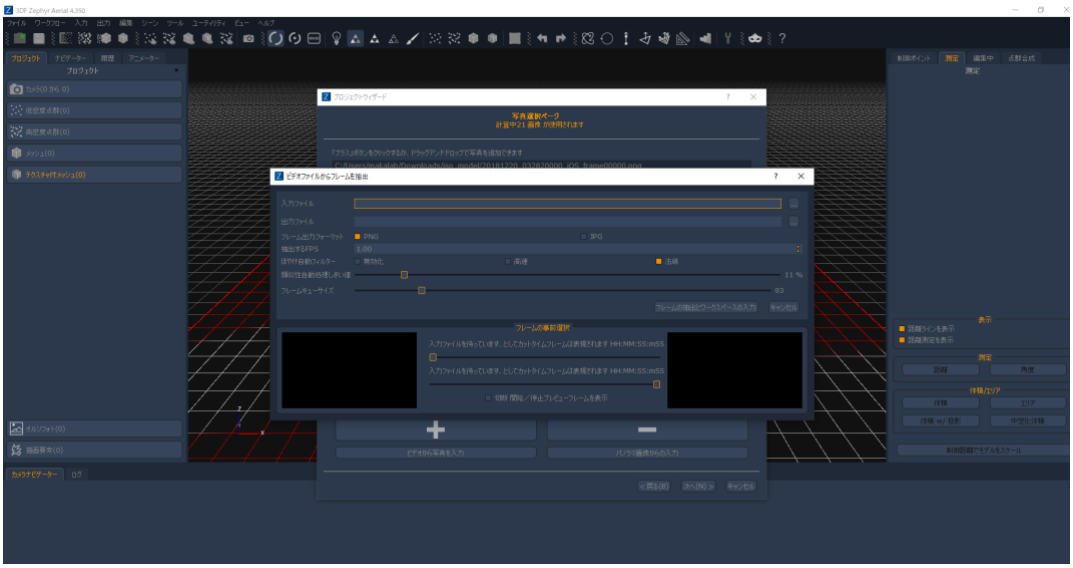


Fig. 4.1-2 動画画像入力画面

5. 生成された複数の画像を入力した状態で次へ．

6. 深い，高精細等の任意の設定をして，実行．

7. RGB３次元モデルが生成される．

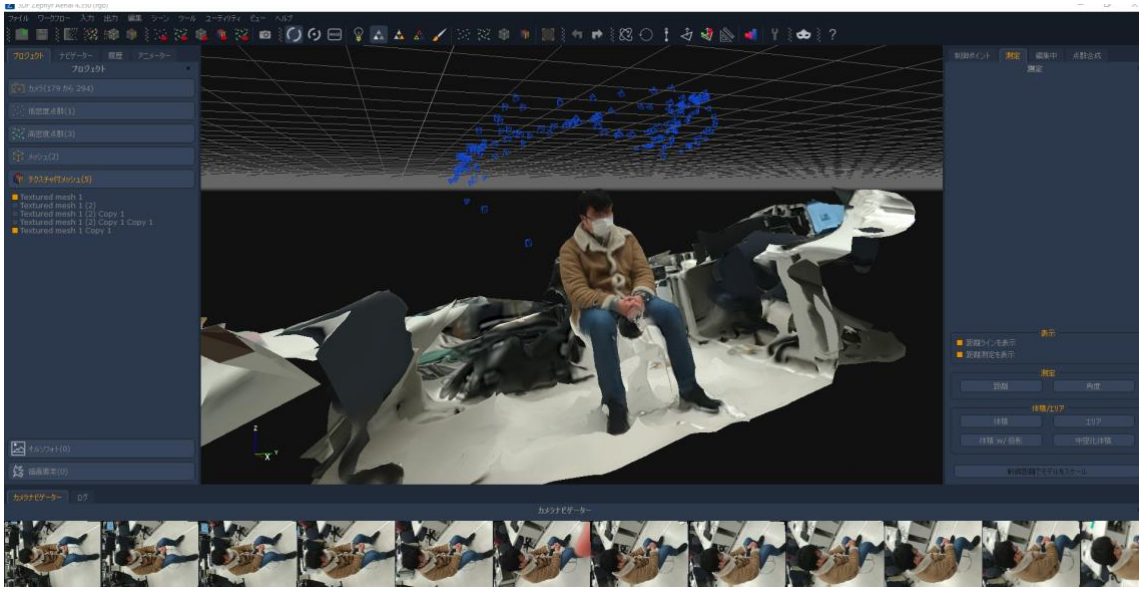


Fig. 4.1-3 RGB３次元モデル復元結果

4.2　IRT 画像からの IRT３次元モデルの復元方法

本節では，サーモカメラ Flir one pro を使用して同様の対象物体の動画を撮影した．また，後に行う３次元モデルの重畳のため，デジタルカメラでの撮影時の同様の姿勢のまま，サーモカメラでの撮影を行った．本節では，以下の手順から IRT３次元モデルの復元を行った．

1. Flir one pro を Flir one アプリ搭載済みのアンドロイドスマートフォンに接続する．
2. Flir one アプリの画面を開き，サーモカメラが作動する．
3. 設定をサーマル画像と温度固定，動画撮影モードにする．
4. 前節同様，人体を一周しながら撮影する．
5. 前節同様，本ソフトを立ち上げ，最初の画面から新規プロジェクトを開く．
6. 前節同様，諸々の設定を行う，テクスチャ付き３次元モデルの生成までの設定する．
7. 前節同様，動画入力を行う．
8. 前節同様，生成された複数の画像を入力する．
9. 前節同様，深い，高精度等の任意の設定をして，実行する．
10. IRT３次元モデルが生成される．
11. メニューから任意の階層に名前をつけて保存する．

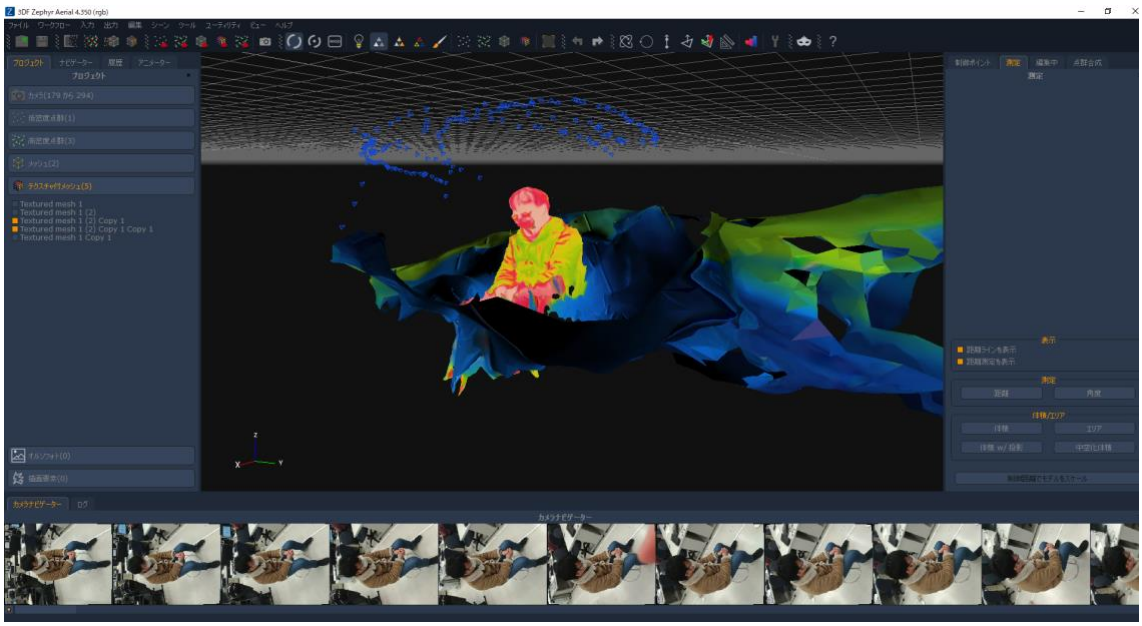


Fig. 4.2 IRT３次元モデル復元結果

第5章　対象物体の抽出方法の比較

本研究では，実際の現場で行われる対象物体の抽出効率も，現場での作業効率において重要な要因となるため，Zephyr で行うことができる対象物体の２つの抽出方法を比較・評価を行い，現場に対しての有用性を考察する．本章では，ノイズや不必要な背景等の排除，つまり対象物体の抽出を２つの方法により行ない，同時にそれぞれの全体所要時間，処理時間，実質作業時間を計測した．

5.1　クリーニングツールによる物体抽出

本節では，クリーニングツールでの人体対象の物体抽出を行う．クリーニングツールによる抽出とは，３次元的なマウス操作により選択範囲を定め，削除していく機能と操作の繰り返しである．この方法では，すでに３次元化された物体から抽出を行なっていくため，前章で復元した人体の IRT 画像からの３次元モデルをそのまま使用する．以下の手順によりクリーニングツールによる物体抽出を行なう．ただし，実験を行う筆者自身はチュートリアルを一通り終えているとする．

1. Zephyr の起動し，画面を開く．
2. 保存した先ほどのサーモグラフィからの３次元モデルを開く．
3. 「手動選択」選択する．選択範囲を指定できるようになる．

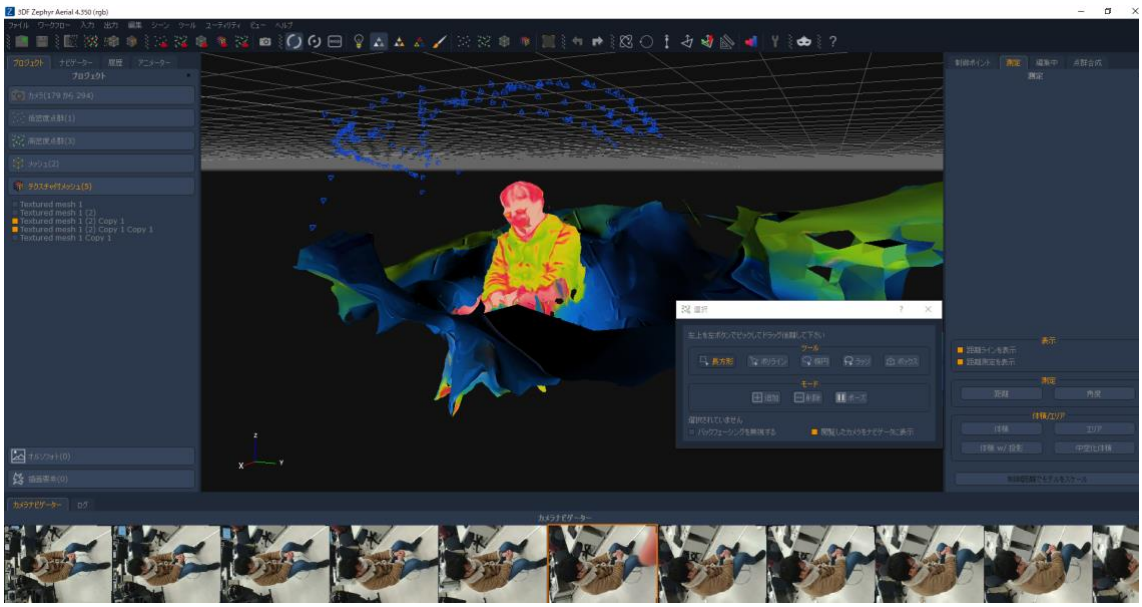


Fig.5.1-1 手動選択画面

4. ノイズや背景等を，長方形選択等，任意の方法により範囲選択をする．
5. Delete キーを押して，削除する
6. ノイズや不要な背景等がすべてなくなるまで４と５の手順を繰り返す．
7. 保存する．

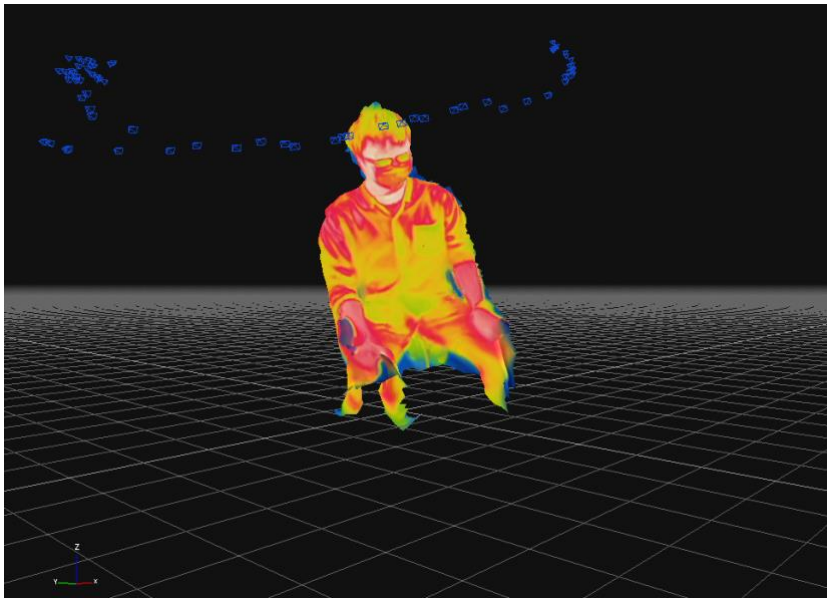


Fig.5.1-2 抽出結果

5.2　マスキングツールによる物体抽出

本節では，マスキングツールによる対象物体の抽出を行う．マスキングツールは，クリーニングツールとは異なり，IRT 画像からマスキングを行なっていく２次元的なマウス操作をする．前章で動画から画像に分割し作成した IRT 画像を用いる．以下の手順によりマスキングツールによる物体抽出を行う．

1. Zephyr の起動し，画面を開く．
2. 新規プロジェクトを開く．
3. 「マスク画像」にチェックを入れる．
4. IRT 画像を入力し，次へ．
5. Masquerade を起動する．
6. 各画像をマスキングする．

6.1. 青マーカーで対象物体を外側囲う．

6.2.　赤マーカーで対象物体を境界線の内側をなぞる.

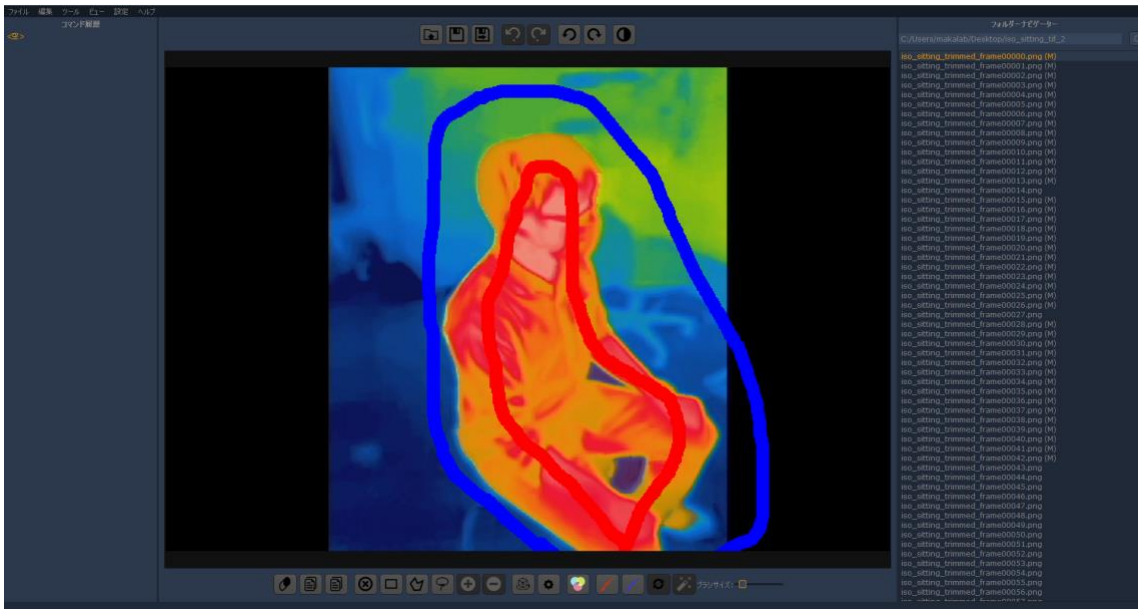


Fig.5.2-1 マスキング画面

7.　本画面に戻り，マスキングを施した IRT 画像を入力した状態で「次へ」を選択する.

8.　諸々の設定に変更する.

9.　実行する.

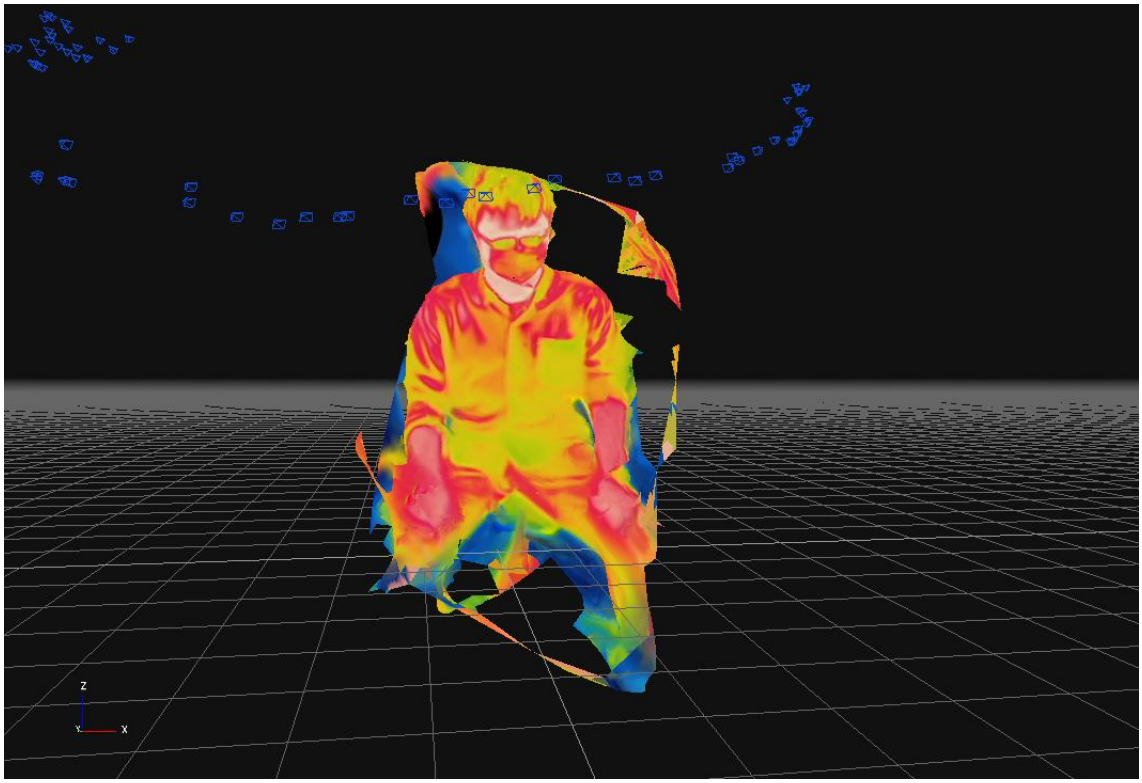


Fig.5.2-2 抽出結果

5.3　比較結果

本節では，２つの方法を計測結果や特徴をまとめ，考察した．結果を受けて，使用したデータセットが１００枚近くあるということもあるが，明らかにクリーニングツールの方が対象物体を速く抽出できる．しかし，マスキングツールを用いた抽出が効率的となる場合がある．例えば，データセットの枚数が少ない，背景が変動し対象物体が変化しない，３次元的な操作が苦手，確実にきれいに不要な部分を削除したい，といった場合である．したがって，手動による抽出方法は適宜決定することが考えられる．

項目	クリーニング	マスキング
全体所要時間	１１分	３３分
処理時間	６分	９分
実質作業時間	１分１０秒	２４分
特徴	・３次元的な操作 ・比較的慣れが必要 ・必要な部分を削除しやすい ・取り残しがしやすい ・修正がしやすい	・２次元的な操作 ・操作が簡単 ・対象の背景が変動する時に便利 ・取り残しがあまりない

Table.5.3 （IRT 画像約１００枚）

第6章　サーモグラフィからの３次元モデルの利活用

本研究では，IRT 画像から復元し対象物体を抽出した３次元モデルを RGB 画像から復元した３次元モデルへ重畳することによって IRT３次元モデルの利活用を提案し，実験した．

6.1　IRT３次元モデルの RGB３次元モデルへの重畳

本節では，RGB３次元モデルに対して対象物体が抽出された IRT３次元モデルを重畳させる実験を行う．以下の手順により IRT３次元モデルを重畳させる．ただし，先ほど復元した RGB３次元モデルと抽出した IRT３次元モデルを用いる．

1. 最初の画面から保存した IRT３次元モデルのファイルを開く．
2. 制御ポイントバーを開く．
3. 識別しやすい各部分に４点以上の制御点を与える．

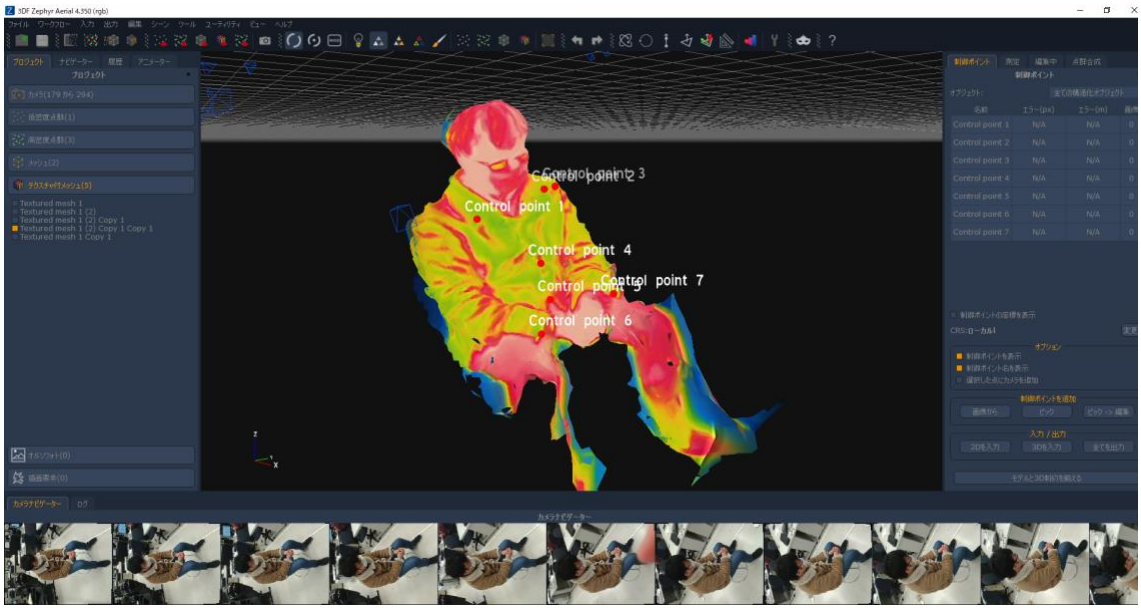


Fig.6.1-1 IRT３次元モデルの制御ポイント画面

4. 保存する．
5. RGB３次元モデルを開く．
6. IRT３次元モデルに与えた制御点となるたけ同じ位置に制御点を与える．

Fig. 6.1-2 RGB３次元モデルの制御ポイント画面

7. 保存する.
8. PC 上から IRT３次元モデルのファイルがあるフォルダを開く.
9. Zephyr の画面上に IRT３次元モデルのファイルをドラッグする.
10. マージ（統合する）を選択する.
11. 保存する.

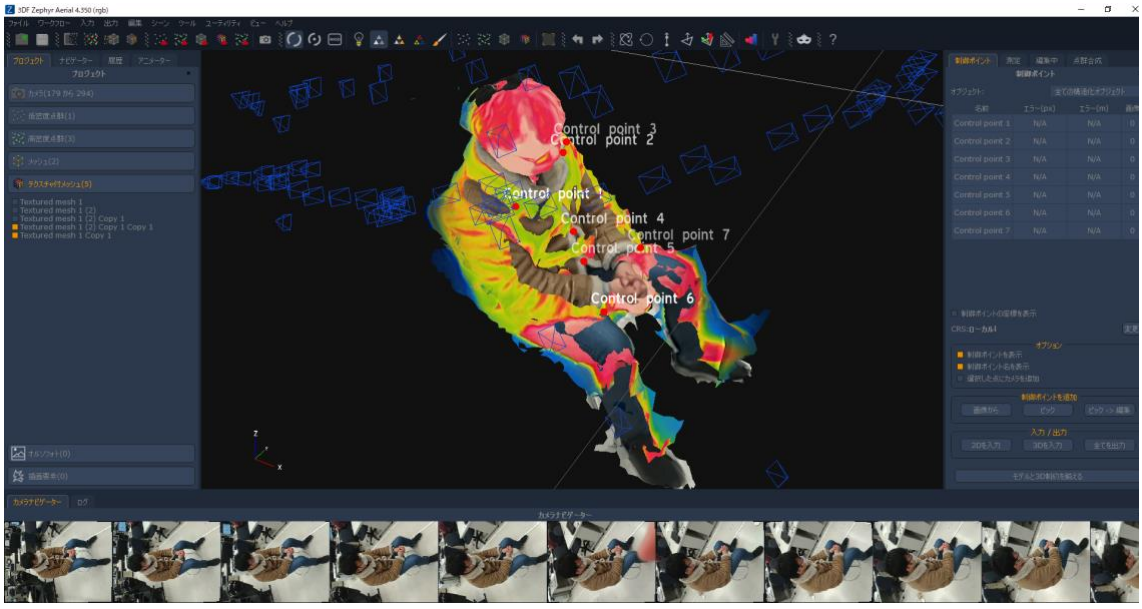


Fig. 6.1-3 統合

6.2　　重畳された IRT３次元モデルの利活用の考察

前節では，Zephyr 上で２つの３次元モデルを統合した．本節では，重畳された IRT３次元モデルの利活用の考察を行う．

Zephyr 上で表示される重畳された IRT３次元モデルは表示のオンオフを切り替えできる．これによって，１章で述べた，IRT３次元モデルにおける境界線に対する視認性の悪さという問題点に対して，一つの解決案になるのではないかと考えられる．何故なら，IRT３次元モデル上では曖昧な境界線があるとしても，IRT３次元モデルの表示をオフにすることで即座に覆われている RGB３次元モデルを確認できるので，境界線がどの部分になるのかがわかるのではないかと考えられるからである．

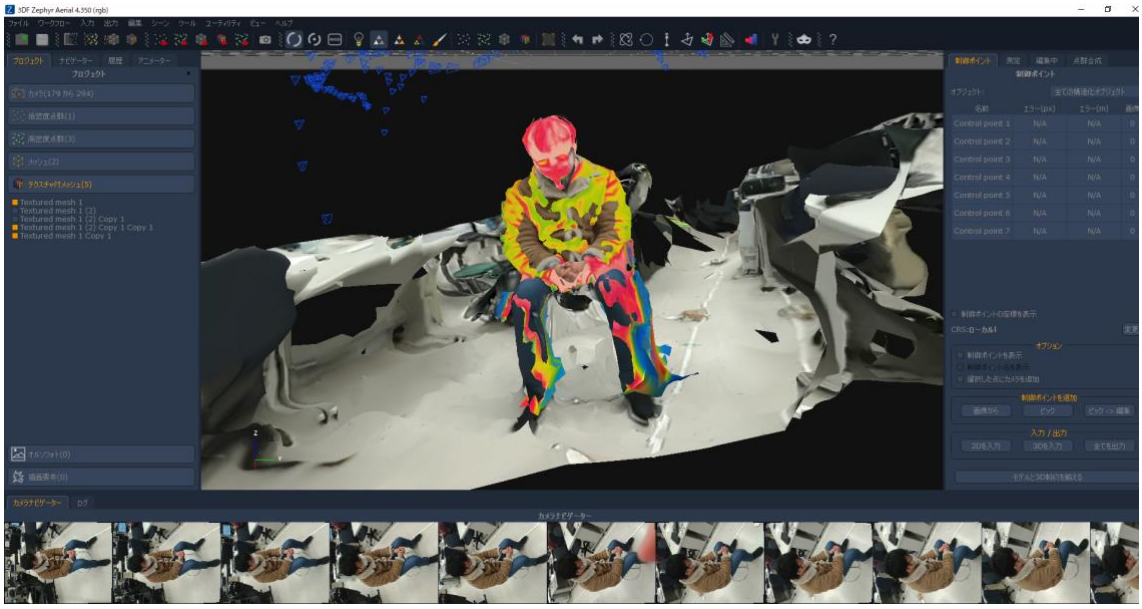


Fig.6.2 重畳

第7章　温度範囲表示システムの開発

本研究では，前章で作成した重畳された IRT３次元モデルのさらなる利活用のための提案として，温度範囲表示システムを開発した．本システムは，予め作成し重畳された IRT３次元モデルで表される温度情報の範囲を絞って表示することができる．また，IRT３次元モデルの表示切り替えを行うことにより覆われた RGB３次元モデルを交互に確認することができる．そして，対象物体を中心にカメラ操作ができるので四方八方から確認することができる．

7.1　温度範囲表示システムの使用方法

本システムの使用方法をここに述べる．ただし，本研究で作成した重畳された３次元モデルを使用する．

1. いずれかの温度範囲のチェックボックスをチェックする．
2. 「表示範囲を変更する」をクリックする．
3. 再び異なる温度範囲を表示した場合，「リセット」をクリックする．
4. ２．と３．同様の手順を追う．
5. 「表示/非表示」により IRT３次元モデルの表示切り替えができる．
6. 「F」キーを押しながらマウスを上下左右に動かすとカメラを上下左右操作できる．
7. 「？」をクリックすると，ヘルプを開くことができる．

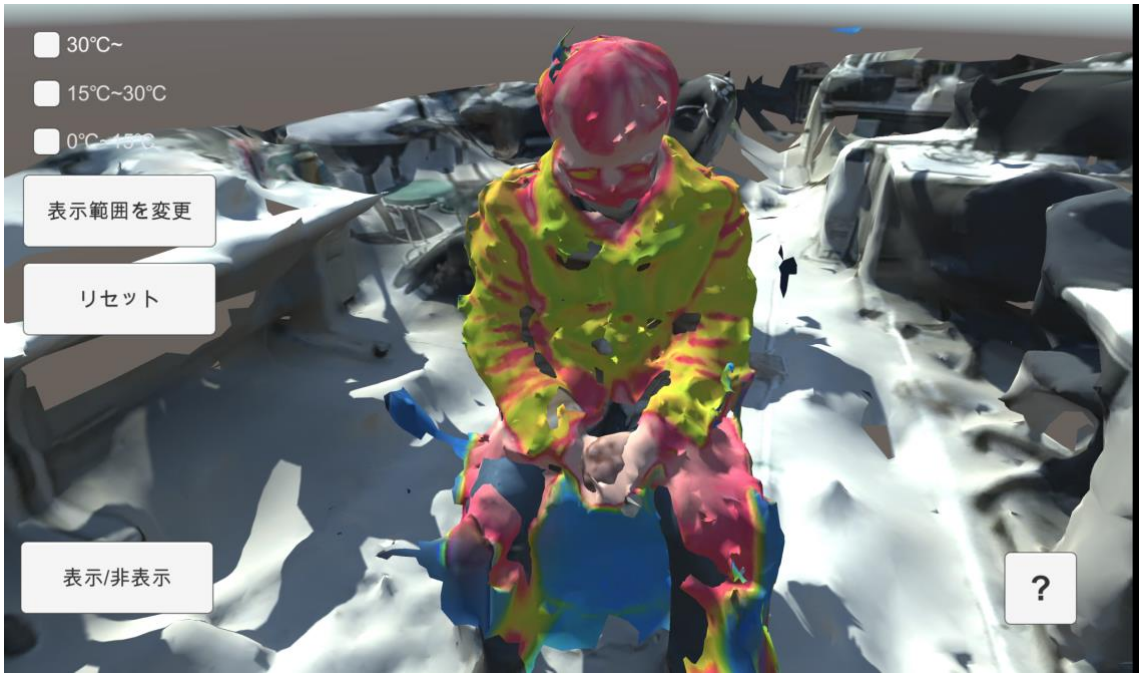


Fig.7.1-1 システム初期画面

Fig.7.1-2 中温度の範囲表示結果，上左斜め視点

7.2　温度範囲表示システムに対する考察

本システムは，RGB３次元モデルに重畳した IRT３次元モデルの温度情報を指定した温度範囲から表示させることができた.

本システムを用いることにより以下の利点が考えられる.

A)　確認したい温度範囲のみを表示させることで、どの領域が問題となっていそうか，より速く点検できる.

B)　表示切り替え機能により，RGB３次元モデルと交互に確認できるのでサーモグラフィで起こり得る境界線の曖昧さの問題を回避することができる.

第8章　結論と課題

8.1　結論

本研究では，デジタルカメラとサーモカメラから対象の物体のそれぞれ RGB 画像と IRT 画像を撮影し，Zephyr によりそれぞれの３次元モデルを復元した．そして，クリーニングツールとマスキングツールの二つの方法により復元した IRT３次元モデルにおける対象物体のみを抽出し，同時に二つの方法を計測し，比較と評価を行なった結果，クリーニングツールによる方法の方が速いものの，適宜方法を決めるべきだとした．また，制御点を与えることにより抽出済みの IRT３次元モデルを RGB３次元モデルへ重畳させ，考察を行なった．最後に，重畳させた IRT３次元モデルを利活用するための温度範囲表示システムを開発し，本研究で作成した IRT３次元モデルで実験と考察を行ない，３次元化されたサーモグラフィの視認性の悪さに対する解決のアプローチの一つになるのではないかと考えた．

8.2　課題

本研究では，二つの方法により対象物体のみの抽出を行なったが，実際の現場では大規模なデータセットを用いられることが考えられるので，いずれの方法にしてもかなりの時間を要するだろう．しかし，手動で行ったところを自動化すれば作業効率の向上を実現することができると考えられる．したがって，IRT３次元モデルにおける自動物体抽出システム及びアルゴリズムの開発を展望として行いたい．

本研究で開発した本システムは数値による詳細な温度範囲の指定ができず，改善していきたい．また，対象とする３次元モデルをカメラから見えるようにする配置を現状手動で行っているため，自動配置ができるようにしたい．さらに，エディター上にドラッグすることによって３次元モデルを入力するのではなく，アプリとしてビルドした後もアプリから３次元モデルのファイルを入力できるように行いたい．最後に，本システムの完成度を実用レベルまでに引き上げた後，実際のユーザーに対する有用性の実証を行いたい．

参考文献

1) i-Construction の推進状況
http://www.mlit.go.jp/tec/i-construction/pdf/01.3_kikaku_siryou1.pdf

2) 国土交通省の新たな取り組み「i-Construction」とは？
https://www.kentem.jp/product-info/key-icon/

3) PHOTOGRAMMETRIC 3D BUILDING RECONSTRUCTION FROM THERMAL IMAGES
https://air.uniud.it/retrieve/handle/11390/1117425/175716/isprs-annals-IV-2-W3-25-2017.pdf

4) Thermal 3D Modelling
http://eprints.whiterose.ac.uk/120381/1/ISARC2017-Paper068.pdf

5) 3D Thermal Modeling of Built Environments Using Visual and Infrared Sensing
https://spectrum.library.concordia.ca/982193/1/Al%20Lafi_MASc_S2017.pdf

6) 3DF Zephyr Tutorial
https://www.3dflow.net/technology/documents/3df-zephyr-tutorials/

7) 3DF Zephyr Technology
https://www.3dflow.net/technology/

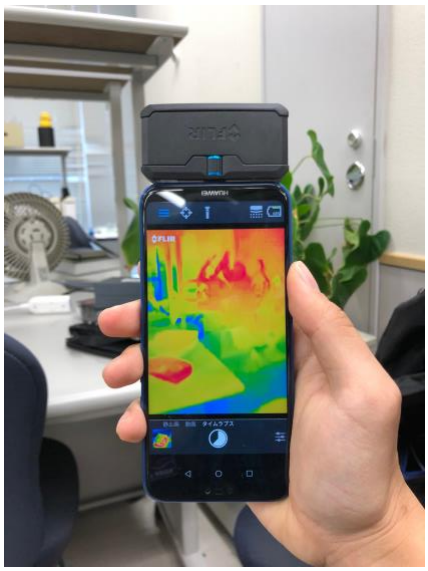
謝辞

本研究を進めるにあたり，終始ご指導賜りました宮城大学事業構想学部デザイン情報学科，蒔苗耕司教授に深く感謝の意を表し，厚く御礼申し上げます．また，研究を進めるに当たり，実験に協力し，共に研究に励んだ蒔苗研究室の同士の皆様に，深く御礼申し上げます．

付録

A) Flir one pro の使用方法

1. アンドロイド端末の Play ストアで FLIR ONE アプリをダウンロード.
2. FLIR ONE アプリを開く.
3. FLIR ONE PRO をアンドロイド端末に取り付ける.
4. 上下を逆にして持つ.
5. 右下のアイコンを開く.
6. 「サーマル画像」にする.
7. 「温度範囲固定」にする.
8. 画面を左右にスワイプして,「静止画」または「動画」で撮影する.

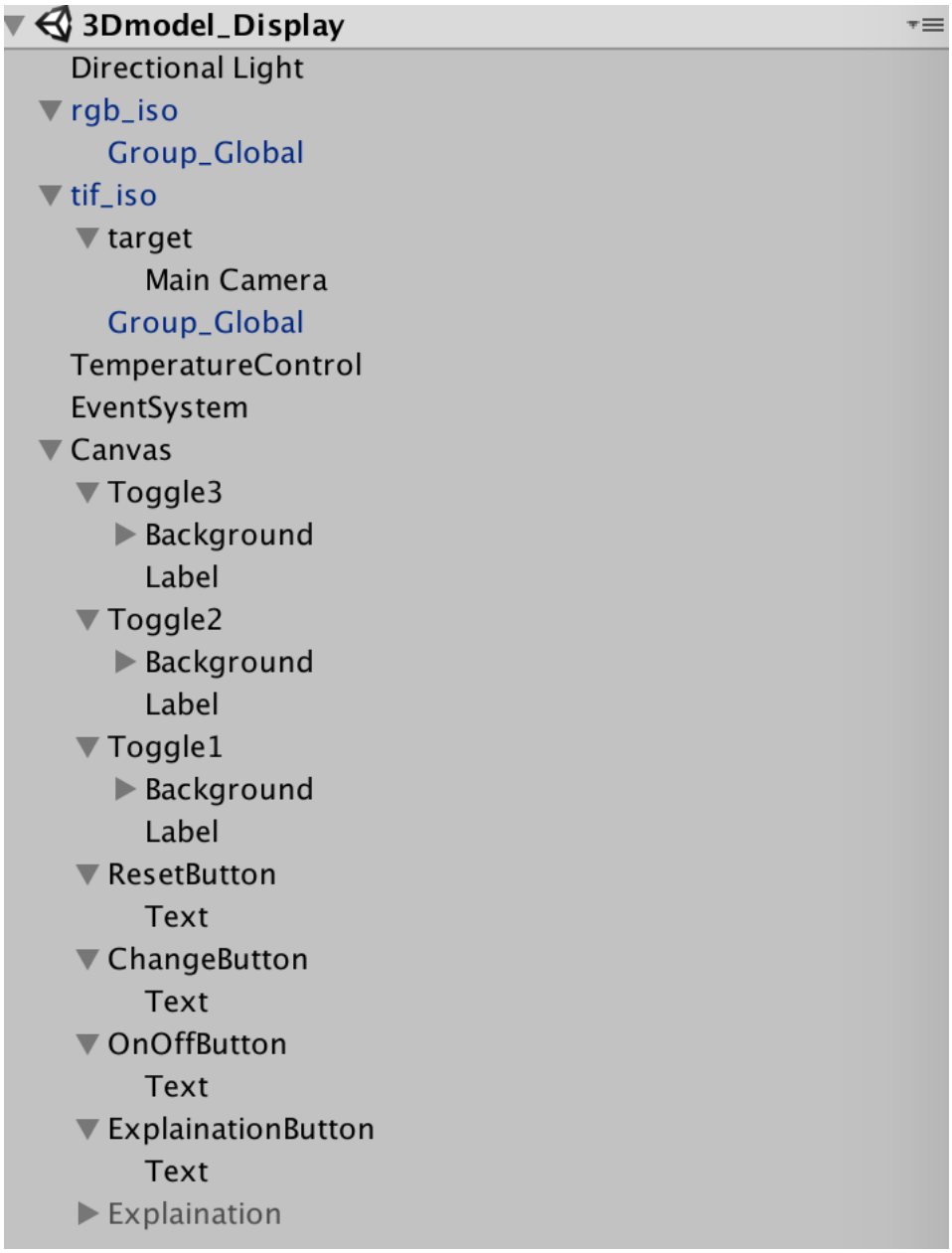


B) 3DF Zephyr Education 版

https://www.3dflow.net/3df-zephyr-education/

C) 温度範囲表示システムのソースコード

・ゲームオブジョジェクヒエラルキー



・カメラコントロール

・ターゲットオブジェクトに追加

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraController : MonoBehaviour {
    public GameObject player;
    public GameObject mainCamera;
    void Start() {
        mainCamera = Camera.main.gameObject;
        player = GameObject.FindGameObjectWithTag("Player");
    }
    void Update() {

        transform.position = player.transform.position;
        if (Input.GetKey (KeyCode.F)) {
                rotateCameraAngle ();
        }
    }

    private void rotateCameraAngle() {
        float rotate_speed = 8.5f;
        Vector3 angle = new Vector3(
            Input.GetAxis("Mouse X") * rotate_speed,
            Input.GetAxis("Mouse Y") * rotate_speed,
            0
        );
        transform.eulerAngles += new Vector3 (angle.y, angle.x);
    }
}
```

・対象外画素の透明化

・Group_Golabal オブジェクトに追加

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TemperatureChanger : MonoBehaviour {
    public static Material[] sMaterials;
    public static Color[] p0, p1, p2, p3;
    public static Texture2D sTexture0, sTexture1, sTexture2, sTexture3;

    // Use this for initialization
    void Start () {
        sMaterials = GetComponent<Renderer> ().materials;

        sTexture0 = (Texture2D) sMaterials[0].mainTexture;
        p0 = sTexture0.GetPixels();

        sTexture1 = (Texture2D) sMaterials[1].mainTexture;
        p1 = sTexture1.GetPixels();

        sTexture2 = (Texture2D) sMaterials[2].mainTexture;
        p2 = sTexture2.GetPixels();

        sTexture3 = (Texture2D) sMaterials[3].mainTexture;
        p3 = sTexture3.GetPixels();
    }

    // Update is called once per frame
    void Update () {

    }
```

```
float GetMaxMinTemp(int choose){

    GameObject ValueRange = GameObject.FindGameObjectWithTag ("GameController");
    float MaxD = ValueRange.GetComponent<MaxMinValue> ().MaxD;
    float MinD = ValueRange.GetComponent<MaxMinValue> ().MinD;
    float Max = ValueRange.GetComponent<MaxMinValue> ().Max;
    float Min = ValueRange.GetComponent<MaxMinValue> ().Min;

    switch(choose)
    {
    case 0:
        return MaxD;
    case 1:
        return MinD;
    case 2:
        return Max;
    case 3:
        return Min;
    default:
        return MaxD;
    }
}
bool FilteringTemtperature(Color pixel,float MaxD, float MinD){

    //範囲パーセンテージの作成 0 から index までの範囲が表示される
    float ShowRangeLow = MinD;
    float ShowRangeHigh = MaxD;

    float H, S, V;
    Color.RGBToHSV (pixel,out H, out S, out V );
```

```
    float pixelFloat = H;


    //フィルタリング
    if (pixelFloat <= ShowRangeHigh && pixelFloat > ShowRangeLow) {
        return true;
    } else if (pixelFloat < ShowRangeLow) {
        return false;
    } else if (pixelFloat > ShowRangeHigh) {
        return false;
    } else
        return true;
}
public void ChangeColorofMaterial(float Max, float Min){
    Material[] materials = GetComponent<Renderer> ().materials;
    //マテリアル0----------------------------------------------------------------------------
    Texture2D mainTexture0 = (Texture2D) materials[0].mainTexture;
    Color[] pixels0 = mainTexture0.GetPixels();//ピクセル取得
    // 書き換え用テクスチャ用配列の作成
    Color[] change_pixels0 = new Color[pixels0.Length];
    for (int i = 0; i < pixels0.Length; i++) {
        Color pixel = pixels0[i];
        //にこで各画素のフィルタリングを行う
        bool pixelFiltering = FilteringTemtperature(pixel, Max, Min);
        //Debug.Log (pixelFiltering);
        if (pixelFiltering == true) {
            // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
            Color change_pixel = new Color (pixel.r, pixel.g, pixel.b, pixel.a);
            change_pixels0.SetValue (change_pixel, i);
        } else {//透明にする
            Color change_pixel = new Color (1.0f, 1.0f, 1.0f, 0.0f);
            change_pixels0.SetValue (change_pixel, i);
        }
```

```
}
// 書き換え用テクスチャの生成
Texture2D change_texture0 = new Texture2D (mainTexture0.width, mainTexture0.height, TextureFormat.RGBA32, false);
change_texture0.filterMode = FilterMode.Point;
change_texture0.SetPixels (change_pixels0);
change_texture0.Apply();
// テクスチャを貼り替える
materials[0].mainTexture = change_texture0;

//マテリアル1----------------------
Texture2D mainTexture1 = (Texture2D) materials[1].mainTexture;
Color[] pixels1 = mainTexture1.GetPixels();

// 書き換え用テクスチャ用配列の作成
Color[] change_pixels1 = new Color[pixels1.Length];
for (int i = 0; i < pixels1.Length; i++) {
    Color pixel = pixels1[i];

    //ここで各画素のフィルタリングを行う
    bool pixelFiltering = FilteringTemtperature(pixel,Max, Min);
    //Debug.Log (pixelFiltering);
    if(pixelFiltering == true){
        // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
        Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
        change_pixels1.SetValue(change_pixel, i);
    }else {//透明にする
        Color change_pixel = new Color(1.0f, 1.0f, 1.0f, 0.0f);
        change_pixels1.SetValue(change_pixel, i);
    }
}
// 書き換え用テクスチャの生成
Texture2D change_texture1 = new Texture2D (mainTexture1.width, mainTexture1.height, TextureFormat.RGBA32, false);
change_texture1.filterMode = FilterMode.Point;
```

```
change_texture1.SetPixels (change_pixels1);
change_texture1.Apply();

// テクスチャを貼り替える
materials[1].mainTexture = change_texture1;
//マテリアル2------------------------------------------------------------
Texture2D mainTexture2 = (Texture2D) materials[2].mainTexture;
Color[] pixels2 = mainTexture2.GetPixels();
// 書き換え用テクスチャ用配列の作成
Color[] change_pixels2 = new Color[pixels2.Length];
for (int i = 0; i < pixels2.Length; i++) {
    Color pixel = pixels2[i];
    //ここで各画素のフィルタリングを行う

    bool pixelFiltering = FilteringTemtperature(pixel, Max, Min);
    //Debug.Log (pixelFiltering);
    if(pixelFiltering == true){
        // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
        Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
        change_pixels2.SetValue(change_pixel, i);
    }else {//透明にする
        Color change_pixel = new Color(1.0f, 1.0f, 1.0f, 0.0f);
        change_pixels2.SetValue(change_pixel, i);
    }
}
// 書き換え用テクスチャの生成
Texture2D change_texture2 = new Texture2D (mainTexture2.width, mainTexture2.height, TextureFormat.RGBA32, false);
change_texture2.filterMode = FilterMode.Point;
change_texture2.SetPixels (change_pixels2);
change_texture2.Apply();
// テクスチャを貼り替える
materials[2].mainTexture = change_texture2;
//マテリアル3------------
```

```
    Texture2D mainTexture3 = (Texture2D) materials[3].mainTexture;
    Color[] pixels3 = mainTexture3.GetPixels();
    // 書き換え用テクスチャ用配列の作成
    Color[] change_pixels3 = new Color[pixels3.Length];
    for (int i = 0; i < pixels3.Length; i++) {
        Color pixel = pixels3[i];

        //ここで各画素のフィルタリングを行う
        bool pixelFiltering = FilteringTemtperature(pixel, Max, Min);
        //Debug.Log (pixelFiltering);
        if(pixelFiltering == true){
            // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
            Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
            change_pixels3.SetValue(change_pixel, i);
        }else {//透明にする
            Color change_pixel = new Color(1.0f, 1.0f, 1.0f, 0.0f);
            change_pixels3.SetValue(change_pixel, i);
        }
    }
    // 書き換え用テクスチャの生成
    Texture2D change_texture3 = new Texture2D (mainTexture3.width, mainTexture3.height, TextureFormat.RGBA32, false);
    change_texture3.filterMode = FilterMode.Point;
    change_texture3.SetPixels (change_pixels3);
    change_texture3.Apply();

    // テクスチャを貼り替える
    materials[3].mainTexture = change_texture3;
}


public void ResetColorMaterial2(){


}
```

```
public void ResetColorMaterial(){

    // 書き換え用テクスチャ用配列の作成
    Color[] change_pixels0 = new Color[p0.Length];
    for (int i = 0; i < p0.Length; i++) {
        Color pixel = p0[i];
            // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
            Color change_pixel = new Color (pixel.r, pixel.g, pixel.b, pixel.a);
            change_pixels0.SetValue (change_pixel, i);
    }
    // 書き換え用テクスチャの生成
    Texture2D change_texture0 = new Texture2D (sTexture0.width, sTexture0.height, TextureFormat.RGBA32, false);
    change_texture0.filterMode = FilterMode.Point;
    change_texture0.SetPixels (change_pixels0);
    change_texture0.Apply();
    // テクスチャを貼り替える
    sMaterials[0].mainTexture = change_texture0;
    //マテリアル 1----------------------

    // 書き換え用テクスチャ用配列の作成
    Color[] change_pixels1 = new Color[p1.Length];
    for (int i = 0; i < p1.Length; i++) {
        Color pixel = p1[i];
        // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
        Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
        change_pixels1.SetValue(change_pixel, i);
    }
    // 書き換え用テクスチャの生成
    Texture2D change_texture1 = new Texture2D (sTexture1.width, sTexture1.height, TextureFormat.RGBA32, false);
    change_texture1.filterMode = FilterMode.Point;
    change_texture1.SetPixels (change_pixels1);
    change_texture1.Apply();
```

```
// テクスチャを貼り替える
sMaterials[1].mainTexture = change_texture1;
//マテリアル2------------------------------------
// 書き換え用テクスチャ用配列の作成
Color[] change_pixels2 = new Color[p2.Length];
for (int i = 0; i < p2.Length; i++) {
    Color pixel = p2[i];

    // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
    Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
    change_pixels2.SetValue(change_pixel, i);

}
// 書き換え用テクスチャの生成
Texture2D change_texture2 = new Texture2D (sTexture2.width, sTexture2.height, TextureFormat.RGBA32, false);
change_texture2.filterMode = FilterMode.Point;
change_texture2.SetPixels (change_pixels2);
change_texture2.Apply();

// テクスチャを貼り替える
sMaterials[2].mainTexture = change_texture2;
//マテリアル3---------
// 書き換え用テクスチャ用配列の作成
Color[] change_pixels3 = new Color[p3.Length];
for (int i = 0; i < p3.Length; i++) {
    Color pixel = p3[i];

    // 書き換え用テクスチャのピクセル色を指定，RGB をそのまま表示する
    Color change_pixel = new Color(pixel.r, pixel.g, pixel.b, pixel.a);
    change_pixels3.SetValue(change_pixel, i);
}
// 書き換え用テクスチャの生成
```

```
        Texture2D change_texture3 = new Texture2D (sTexture3.width, sTexture3.height, TextureFormat.RGBA32, false);
        change_texture3.filterMode = FilterMode.Point;
        change_texture3.SetPixels (change_pixels3);
        change_texture3.Apply();

        // テクスチャを貼り替える
        sMaterials[3].mainTexture = change_texture3;
    }

}
```

・最大最小値設定

```
Using System.Collections;

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.Linq;
public class SubmitButton : MonoBehaviour {

    //連携する GameObject
    public ToggleGroup toggleGroup;
    public GameObject changeTemp;

    // Use this for initialization
    void Start(){
    }
    // Update is called once per frame
    void Update()
    {
    }
```

```
    static public string low = "0℃~15℃";
    static public string mid = "15℃~30℃";
    static public string high = "30℃~";
    public void onClick()
    {
        //Get the label in activated toggles
        string selectedLabel = toggleGroup.ActiveToggles().First().GetComponentsInChildren<Text>().First(t => t.name == "Label")
.text;
        if (selectedLabel == low) {
            float max = 0.6f;
            float min = 0.3f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);


        } else if (selectedLabel == mid) {
            float max = 0.3f;
            float min = 0.1f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);
        } else if (selectedLabel == high) {
            float max = 1f;
            float min = 0.6f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);
        }
        Debug.Log("selected " + selectedLabel);
    }
}
```

・温度範囲選択

・ChangeButton オブジェクトに追加

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.UI;
using System.Linq;

public class SubmitButton : MonoBehaviour {

    //連携する GameObject
    public ToggleGroup toggleGroup;
    public GameObject changeTemp;
    static public string low = "0℃~15℃";
    static public string mid = "15℃~30℃";
    static public string high = "30℃~";
    public void onClick()
    {
        //Get the label in activated toggles
        string selectedLabel = toggleGroup.ActiveToggles().First().GetComponentsInChildren<Text>().First(t => t.name == "Label")
.text;
        if (selectedLabel == low) {
            float max = 0.6f;
            float min = 0.3f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);



        } else if (selectedLabel == mid) {
            float max = 0.3f;
            float min = 0.1f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);
        } else if (selectedLabel == high) {
            float max = 1f;
            float min = 0.6f;
            changeTemp.GetComponent<TemperatureChanger> ().ChangeColorofMaterial (max, min);
        }
        Debug.Log("selected " + selectedLabel);
```

```
    }
}
```

・表示非表示機能

・OnOffButton オブジェクトに追加

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class OnOff : MonoBehaviour {
    public GameObject TIFmaterial;
    void OnTIF(){
        TIFmaterial.SetActive (true);
    }
    void OffTIF(){
        TIFmaterial.SetActive (false);
    }
    public void switchTIF(){
        if (TIFmaterial.activeSelf == true) {
            OffTIF ();
        } else if (TIFmaterial.activeSelf == false) {
            OnTIF ();
        } else {
            Debug.Log ("Failure of OnOffTIF");
        }
    }
}
```