

最終課題

・ α 拡張を用いたステレオマッチングのアルゴリズム

1. 現在のラベル β を初期化
2. グラフの総コスト E を初期化
3. 最大視差の分だけループ
 - 3-1. グラフの初期化
 - 3-2. ノードの追加
 - 3-3. 全ての画素 k に対して, ノードのソースに $D(\beta k)$ を設定
 - 3-4. 全ての画素 k に対して, ノードのソースに $D(\alpha)$ を設定
 - 3-5. 全てのエッジに対してノード a , ソースに $V(\beta p, \beta q)$, シンク側に $V(\alpha, \alpha)$ を設定
 - 3-6. 全てのエッジに対してノード p とノード a のエッジの重み (双方向) に $V(\beta p, \beta q)$ を設定
 - 3-7. 全てのエッジに対してノード p とノード a のエッジの重み (双方向) に $V(\alpha, \alpha)$ を設定
 - 3-8. 最大流・最小カットアルゴリズムを適用し, 総コスト E' を計算
 - 3-9. もし, $E > E'$ ならば現在のラベル βk を α に更新
 - 3-10. もし, $E > E'$ ならば総コストの更新 ($E = E'$)
 - 3-11. グラフの消去
4. 総コストが収束するまで 4 を繰り返す

・階層グラフカットを用いたステレオマッチングアルゴリズム

1. 現在のラベル β を初期化
2. 階層ラベル A を作成
3. グラフの総コスト E を初期化
4. 階層の深さ $depth$ だけループ
 - 4-1. グラフを初期化
 - 4-2. ノードを追加
 - 4-3. 全ての画素 k に対して現在の深さ i の階層ラベル $A[i]$ のうち最も $\beta[k]$ に最も近い値を α_k に設定
 - 4-4. 全ての画素 k に対してノードのソース側に $D(\beta_k)$, シンク側に $D(\alpha_k)$ を設定
 - 4-5. すべてのエッジに対して現在の深さ i の階層ラベル $A[i]$ のうち最も $\beta[p]$ に最も近い値を α_p に設定
 - 4-6. すべてのエッジに対して現在の深さ i の階層ラベル $A[i]$ のうち最も $\beta[q]$ に最も近い値を α_q に設定
 - 4-7. すべてのエッジに対してノード a ソース側に $V(\beta_p, \beta_q)$, シンク側に $V(\alpha_p, \alpha_q)$ を設定
 - 4-8. すべてのエッジに対してもし $V(\beta_p, \beta_q) \leq V(\alpha_p, \alpha_q)$ ならば,
 - 4-8-1. ノード a からノード p へのエッジの重みに 10000 (比較的大きい数) を設定
 - 4-8-2. ノード a からノード q へのエッジの重みに 10000 (比較的大きい数) を設定
 - 4-8-3. ノード p からノード a へのエッジの重みに, $V(\alpha_p, \beta_q) - V(\beta_p, \beta_q)$ が 0 のうち大きいほうを設定
 - 4-8-4. ノード q からノード a へのエッジの重みに, $V(\beta_p, \alpha_q) - V(\beta_p, \beta_q)$ が 0 のうち大きいほうを設定
 - 4-9. すべてのエッジに対してもし $V(\beta_p, \beta_q) \geq V(\alpha_p, \alpha_q)$ ならば,
 - 4-9-1. ノード p からノード a へのエッジの重みに 10000 (比較的大きい数) を設定
 - 4-9-2. ノード q からノード a へのエッジの重みに 10000 (比較的大きい数) を設定
 - 4-9-3. ノード a からノード p へのエッジの重みに, $V(\alpha_p, \beta_q) - V(\alpha_p, \alpha_q)$ が 0 のうち大きいほうを設定

- 4-9-4. ノード a からノード q へのエッジの重みに,
 $V(\beta p, \alpha q) - V(\alpha p, \alpha q)$ が 0 のうち大きいほうを設定
- 4-10. 最大流・最小カットアルゴリズムを適用し, 総コスト E' を計算
- 4-11. もし, $E > E'$ ならば現在のラベル β_k を求めたラベル更新
- 4-12. もし, $E > E'$ ならば総コストの更新 ($E = E'$)
- 4-13. グラフの消去
- 5. 総コストが収束するまで 4 を繰り返す

・実験

開発環境 Linux CentOS

用いた言語 C++

用いたライブラリ MAXFLOW

メールに添付したファイル 1167025-saisyukadai.zip

α 拡張を行ったフォルダ名 alpha

階層グラフカットを行ったフォルダ名 hierarchical

※注意点

1. 入力画像は ppm または ppm, かつ ascii 形式でしか実行できません.
2. また出力画像は pgm 形式のみとなっています.
3. 出力は, 最大階調を 256 に調整したものです.
4. コンパイル方法は, make コマンドで.
5. 以下のコマンドを実行

```
$ ./GraphCut [leftimagefile] [rightimagefile] [outputfile]
```

・コスト関数

・データコスト

— 出力画像 f_p が入力画像 I_p に出来るだけ近くなるようにする項

$$-D(\beta) = \text{SAD}(\beta) = \sum |I(x, y) - I(x - \beta, y)|$$

・スムーズコスト

— 隣り合った画素 f_p, f_q の明るさが出来るだけ近くなるようにする項

$$-V(f_p) = |f_p - f_q|$$

・実験結果

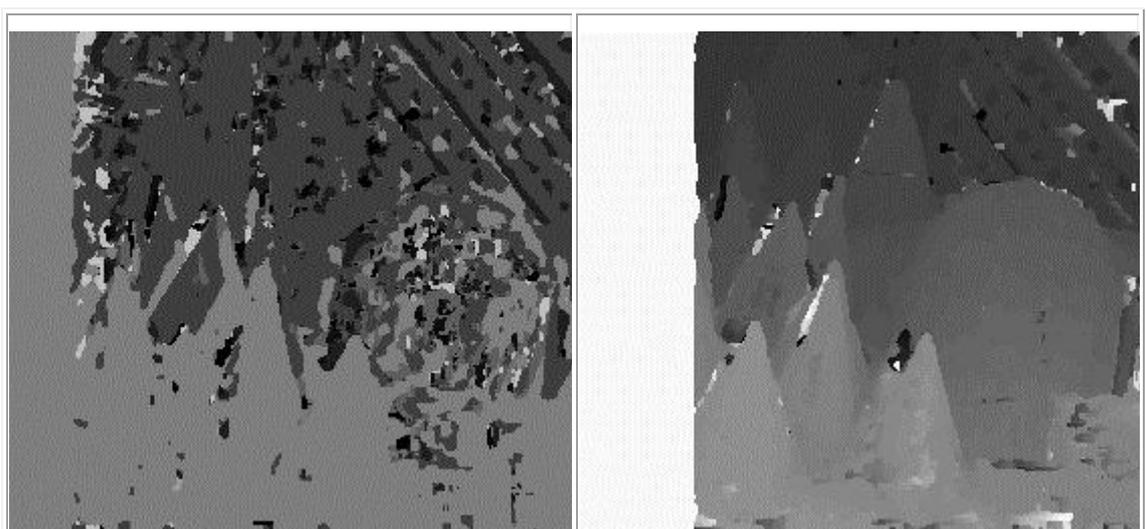
α 拡張を用いたステレオマッチング，及び階層グラフカットを用いたステレオマッチングを行った．現在のラベル β の初期値は0とした．以下に入力画像と α 拡張を用いた場合の実行結果，階層グラフカットを用いた結果を示す．なお，どちらの視差画像も，最大階調を 256 に変換した結果である．



左画像 (300×250)

右画像 (300x250)

入力画像



階層グラフカット 実行時間：13.6sec

α 拡張 実行時間：50.1sec

結果画像

・考察

α 拡張と階層グラフカットによるステレオマッチングの結果を比較すると、精度に明らかな差が生じていることがわかる。 α 拡張ではラベル α を 0~最大視差まで変化させているのに対して、階層グラフカットではラベルを段階的に変化させているため精度が悪いと考えられる。しかし、実行速度では α 拡張よりも訳 4~5 倍の速度を計測した。今回はラベルの初期値をすべて 0 にしているが、ブロックマッチングの結果を初期値とすることなどにより精度の向上が見込めると考える。