VR × 量子
ステレオ画像から
自由視点映像を生成

Fixstars Amplifyハッカソン

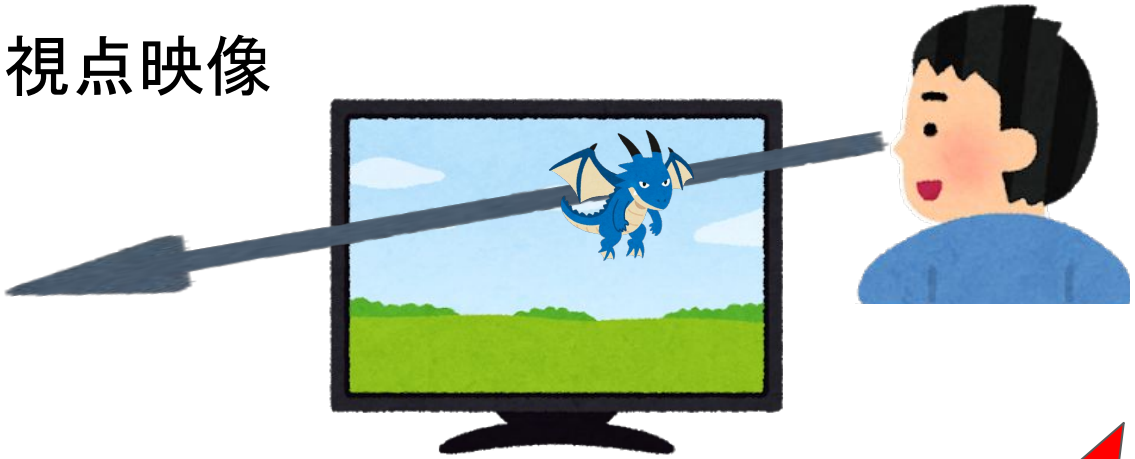
Motivation: 360°ディスプレイ, VRのコンテンツ不足

[1] “ソニー、最新透明ディスプレイを披露360度ぐるりと映像表示”, https://japanese.engadget.com/jp-2019-07-28-360.html .

[2] “Looking Glassの次期製品はホログラフィック・デジタルフォトフレーム、写真はiPhoneで撮影”, https://jp.techcrunch.com/2020/12/03/2020-12-02-looking-glasss-next-product-is-a-holographic-digital-photo-frame/

[3] “VRChat,” https://hello.vrchat.com/ .

Motivation: 自由視点映像



自動生成
したい！

視点位置により閲覧できる映像が異なる

深度（奥行き）情報がわかれば自動生成できる

Preparation: 視差解析による自動生成

ステレオカメラ

Fig 1. A left-eye image.

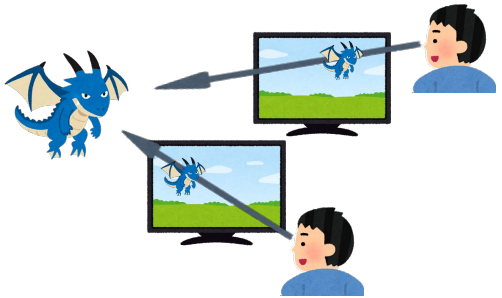
Fig 2. A right-eye image.

ステレオ画像（固定2視点）

Fig 3. An ideal depth map.

深度マップ（視差マップ）

ココの計算量が大きい



自由視点映像（多視点）

Preparation: 視差解析の方法

Fig 1. A left-eye image.

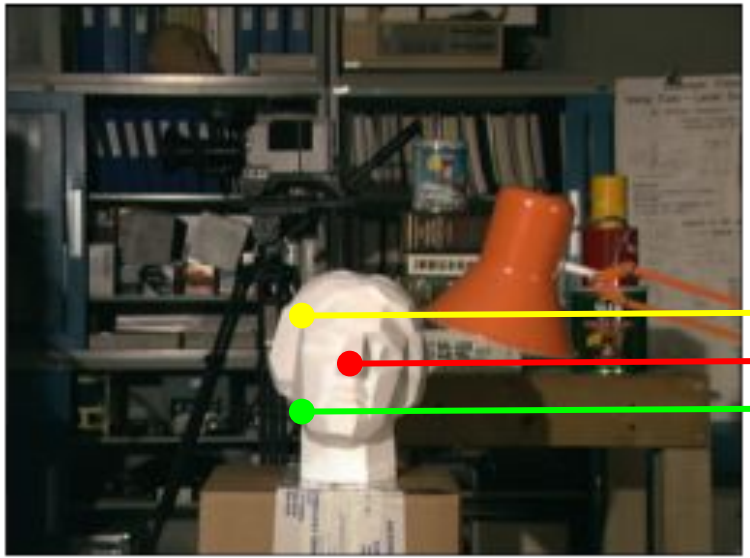
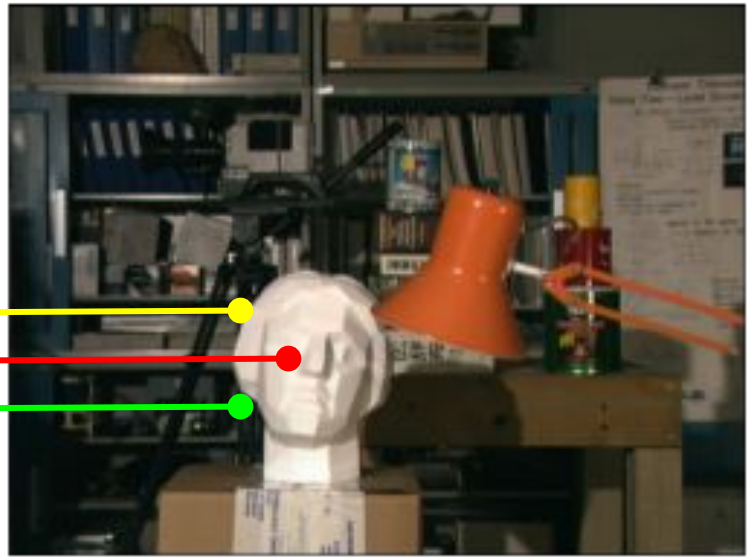


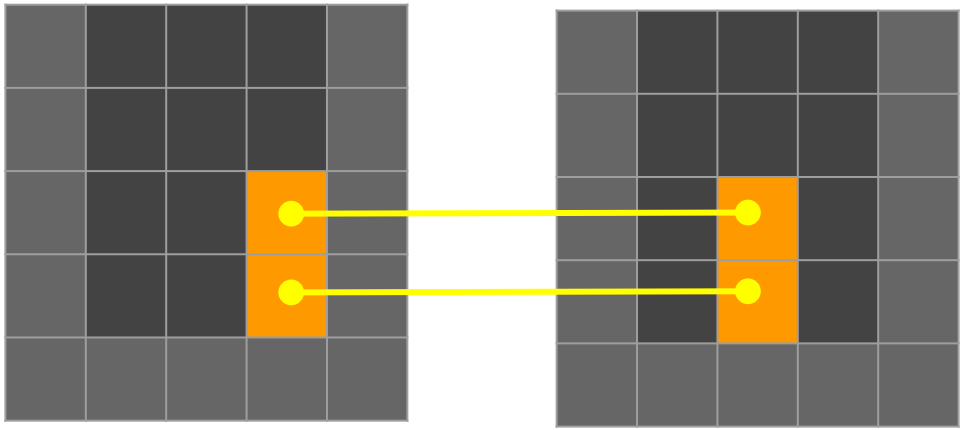
Fig 2. A right-eye image.



特徴点同士を対応付け
左右の画像のピクセルのズレ
= 視差 を求める
手前にある物体ほど視差が大きい

既存手法
SGM（Semi Global Matching）方式
SAD（Sum of Absolute Difference）方式
計算量かかるのでGPUとか

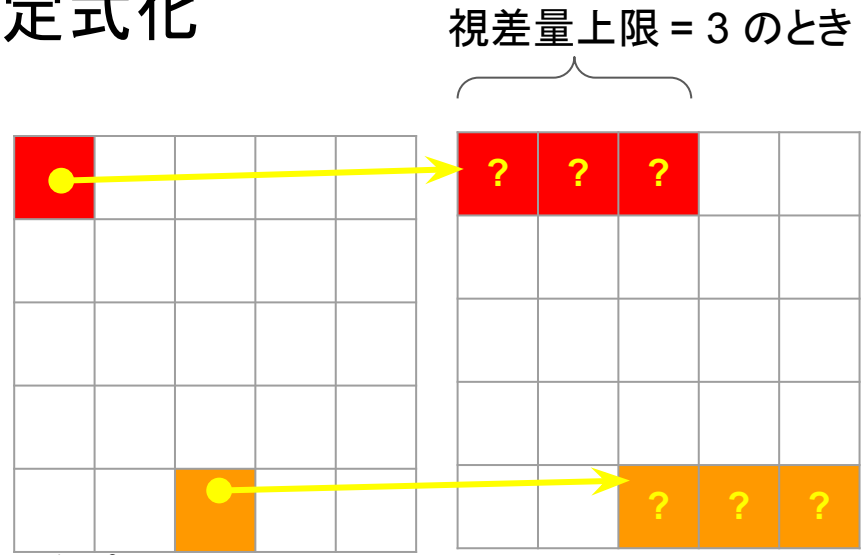
Amplify AEで視差解析



左画像の全ピクセル × 右画像の全ピクセル の対応コストを最小化すれば良い
が, そんな計算量かけたくない……

工夫：視差量に上限を定める ついでにステレオ平行を仮定しておく
対応点が同じ高さに存在

定式化



[ハードな制約条件]

1つのピクセルが対応する視差は1つ

$x(i, j, 0) + x(i, j, 1) + \ldots + x(i, j, K-1) = 1$

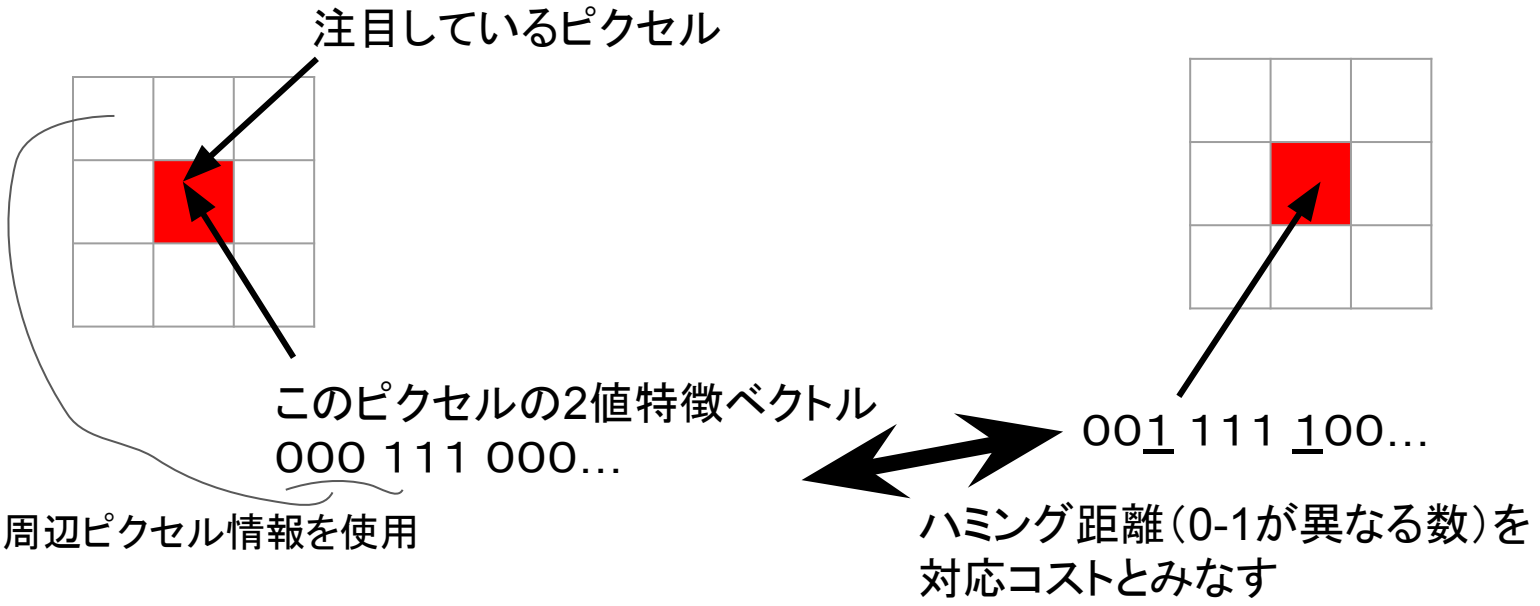
Amplifyではequal_to メソッドで
one-hot制約を実現可能

画像ピクセル × 視差量上限 の0-1指示変数として最適化
座標(j, i)のピクセルの左右の画像の視差がkのとき
　　x(i, j, k) = 1 とする

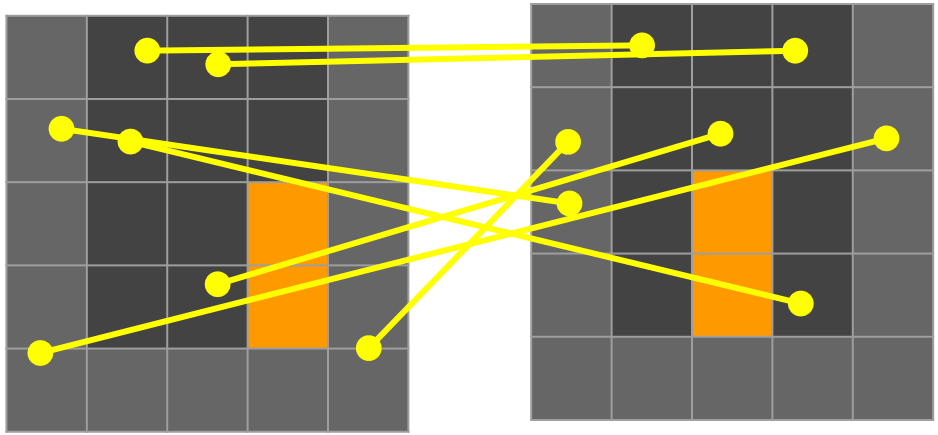
画像サイズ H × W, 視差量上限Kのとき
　　変数はHWK個 (=探索空間は2^HWK)

対応コスト

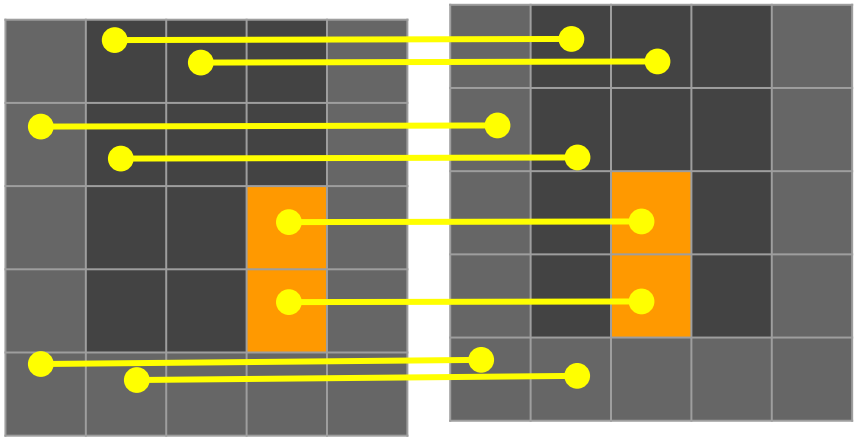
ピクセル同士の対応コストの計算に周辺のピクセル情報も使用（窓）
周辺ピクセルを閾値で2値化し，特徴ベクトルを計算
2値特徴ベクトルのハミング距離を対応コストと定義（こうすると計算が早い）



ソフトな制約条件



ただ**対応コスト**だけを指標とするとめちゃくちゃな対応が最小コストとなってしまう



ソフトな制約条件として，**隣接ピクセルの視差は0~1**を**ソフトな制約**とする
参考：SGM方式

物体の境界では視差が0~1より大きく異なるので厳密な制約ではない

実装・実行結果

jupyter stereo Last Checkpoint: 数秒前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Checkpoint created: 22:51:24 Not Trusted

Run Code

In [1]:
```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from ipywidgets import Button, IntSlider, interactive_output, HBox, Output, Box
from ipywidgets import interact, interactive, fixed, interact_manual
from amplify import Solver, decode_solution, gen_symbols, BinaryPoly, sum_poly, BinaryQuadraticModel
from amplify.client import FixstarsClient
from amplify.constraint import equal_to, penalty
from PIL import Image
```

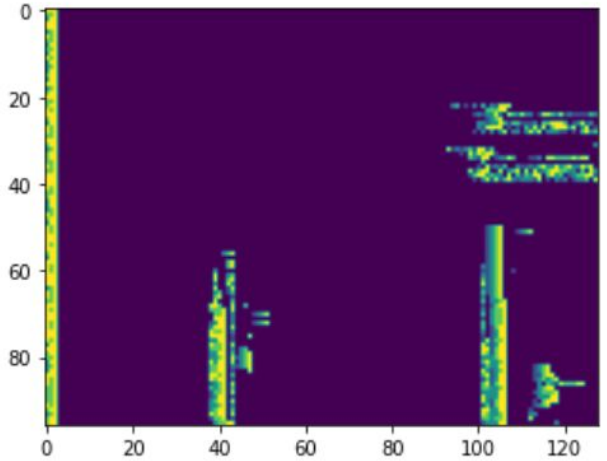
In [2]:
```
W = 384
H = 288

imgL = Image.open('img/tsukuba/scene1.row3.col1.ppm')
imgR = Image.open('img/tsukuba/scene1.row3.col3.ppm')
imgD = Image.open('img/tsukuba/truedisp.row3.col3.pgm')
```

In [3]:
```
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.imshow(imgL)
plt.title('Fig 1. A left-eye image.')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(imgR)
plt.title('Fig 2. A right-eye image.')
plt.axis('off')
plt.show()
```

Fig 1. A left-eye image. Fig 2. A right-eye image.

ipynbファイルとして実装
jupyterで実行

In [18]:
```
plt.imshow(D)
plt.show()
```



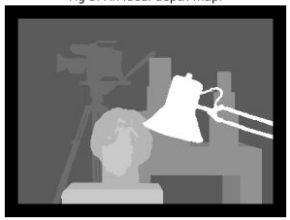
残念ながら有意な視差マップは
得られなかった

真にやりたかったこと

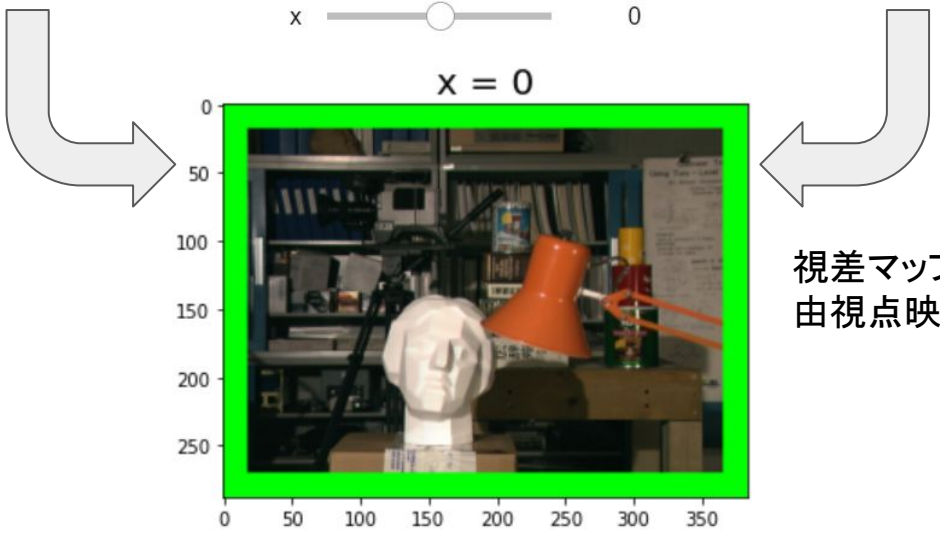
Fig 1. A left-eye image.

Fig 2. A right-eye image.

Fig 3. An ideal depth map.

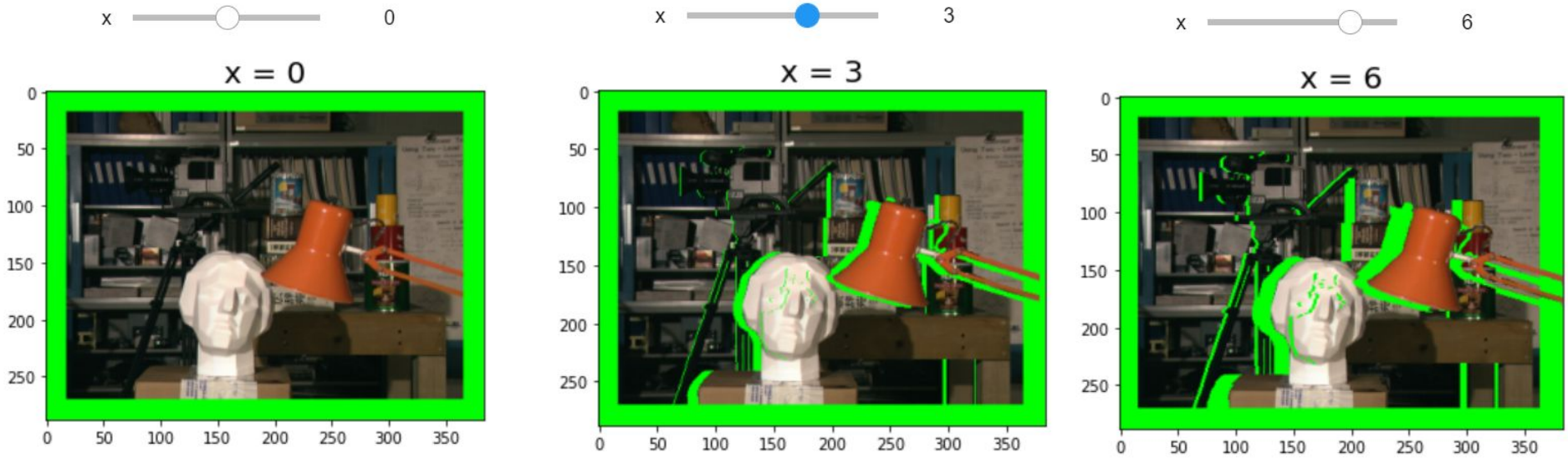


理想の視差マップを用いて
真にやりたかったことを説明
TSUKUBAステレオビジョンベンチマークを使用



視差マップからインタラクティブな自由視点映像を作成

真にやりたかったこと



バーをスライドさせるとインタラクティブに画像が変化
閲覧者の自由な視点を実現
ただしオクルージョンや視差の中間値の補間をしていないので，隙間が緑色になる
（このシステム自体はipynbに実装されているのでよければ遊んでください）

苦労した点

有意な解が得られない

- 途中の状態でうまくいっているのかわからないのでデバッグが難しい
- 定式化の問題なのか，重みの調整が悪いのか，プログラムの書き方なのか

苦労した点

型変換に苦労した

制約を作ると BinaryConstraint型

BinaryConstraint型とBinaryConstraint型を足すとBinaryConstraint**s**型

BinaryConstraint型に重みをかけるとBinaryConstraint**Term**型

BinaryConstraint**s**型にBinaryConstraint**Term**型を足すと...あれっ足せない？

定形の書き方を学ぶ必要がある