2016/06/08　9:45-11:15

第22回画像センシングシンポジウム　チュートリアル講演会

Kinect等の色距離センサを用いた点群処理と3D物体認識

ーベーシックな手法と最新動向・ソフトウェアの紹介ー

産業技術総合研究所　人工知能研究センター

金崎 朝子

本日の資料について

最新版

https://goo.gl/sUr6mC

紙の資料にはないチュートリアル資料

https://goo.gl/HxH8cG

Ubuntu PCをお持ちの方は

本チュートリアルをより楽しむために…

1. ROSをインストール
   http://wiki.ros.org/jade/Installation/Ubuntu ※Desktop-Full推奨

2. 必要なファイルをダウンロード

mydesk.bag（453MB）
https://www.dropbox.com/s/sn0w59sg81bhzm9/mydesk.bag?dl=0
save_pcd.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/save_pcd.cpp
convertpcd2ply.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/convertpcd2ply.cpp
milk.pcd
https://github.com/PointCloudLibrary/pcl/blob/master/test/milk.pcd?raw=true
milk_cartoon_all_small_clorox.pcd
https://github.com/PointCloudLibrary/pcl/blob/master/test/milk_cartoon_all_small_clorox.pcd?raw=true
correspondence_grouping.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/correspondence_grouping.cpp

自己紹介

- 2008年3月　東京大学　工学部機械情報工学科　卒業
- 2008年4月　東京大学　大学院情報理工学系研究科　修士課程進学
- 2010年3月　同学　修士課程修了（情報理工学）
- 2010年4月　同学　博士課程進学
- 2013年3月　同学　博士課程修了（情報理工学）
- 2013年4月　（株）東芝研究開発センター　正規職員
- 2013年12月　東京大学　大学院情報理工学系研究科　助教
- 2016年4月　産業技術総合研究所　人工知能研究センター　研究職員

TOSHIBA
Leading Innovation

東京大学
THE UNIVERSITY OF TOKYO

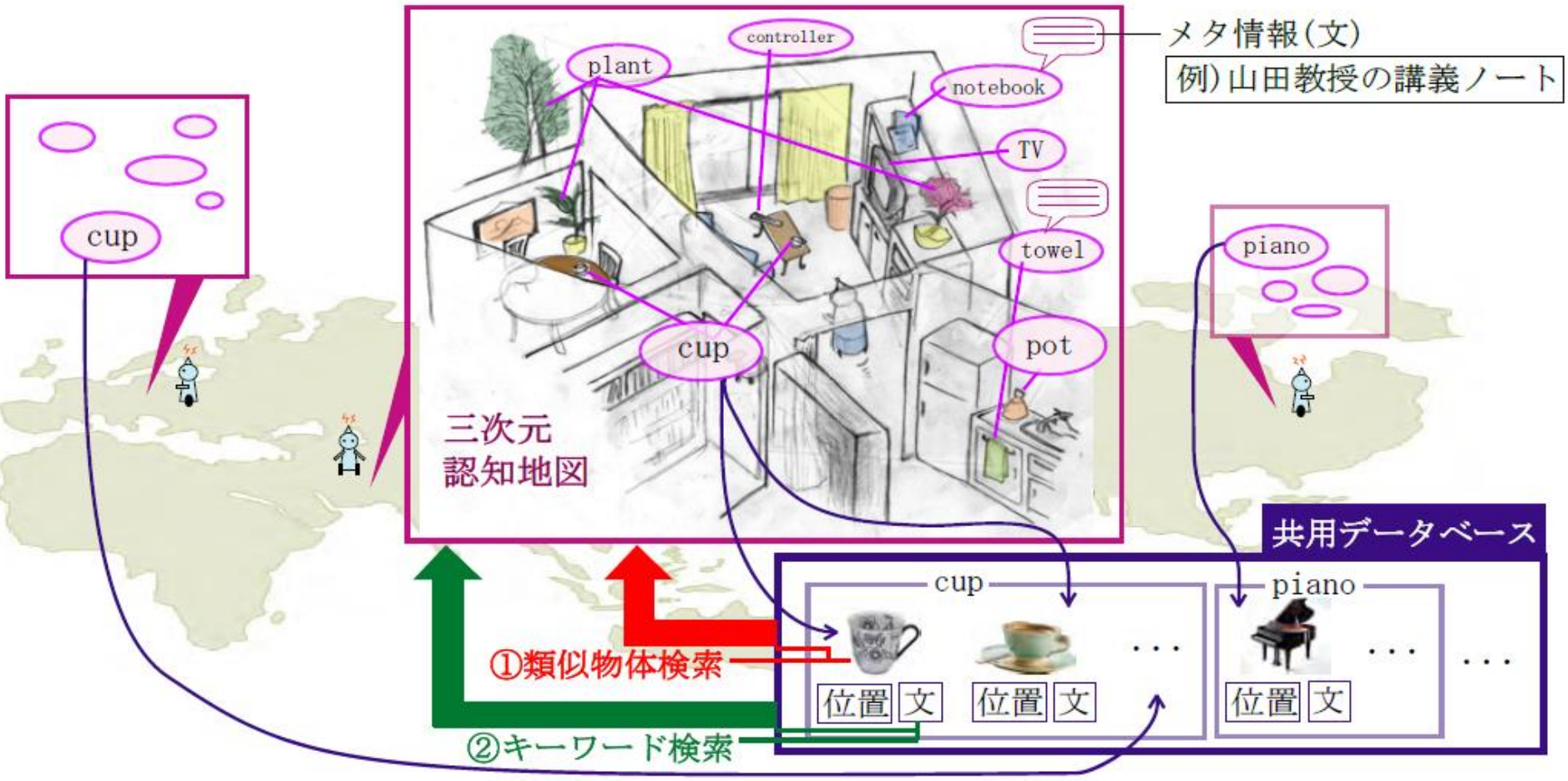
AIST
NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST)

（最近の研究）自律移動ロボットの物体認識

- person, chair等の20種類の物体を検出
- 物体候補抽出法：
  3D Selective Search [Kanezaki+, 2015]
  https://github.com/kanezaki/selective_search_3d

目的とするロボットアプリケーションシステム

大目標：実物体インターネット≒実世界G○○gle？



RGBD画像と3D点群

RGB画像
（色画像）

Depth画像
（距離画像、深度画像）

3D点群（ポイントクラウド）

Kinectセンサ活用事例

研究

医療

https://i.ytimg.com/vi/wUEEsYH0zUE/maxresdefault.jpg

農業

http://agrifood.jp/2015/12/362/

通販

https://i.ytimg.com/vi/Mr71jrkzWq8/maxresdefault.jpg

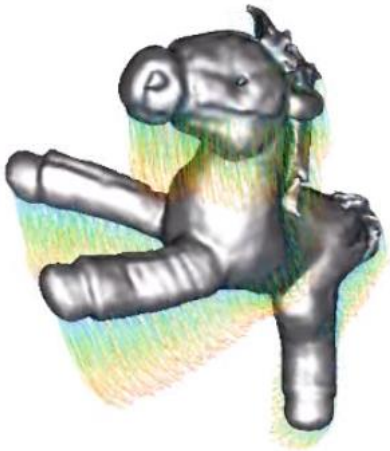
RGBDデータ活用事例
―研究―

活用事例(1)　3Dモデリング

DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time.
R. Newcombe, D. Fox, and S. Seitz, **CVPR 2015 Best Paper**

動画

Live RGBD Video



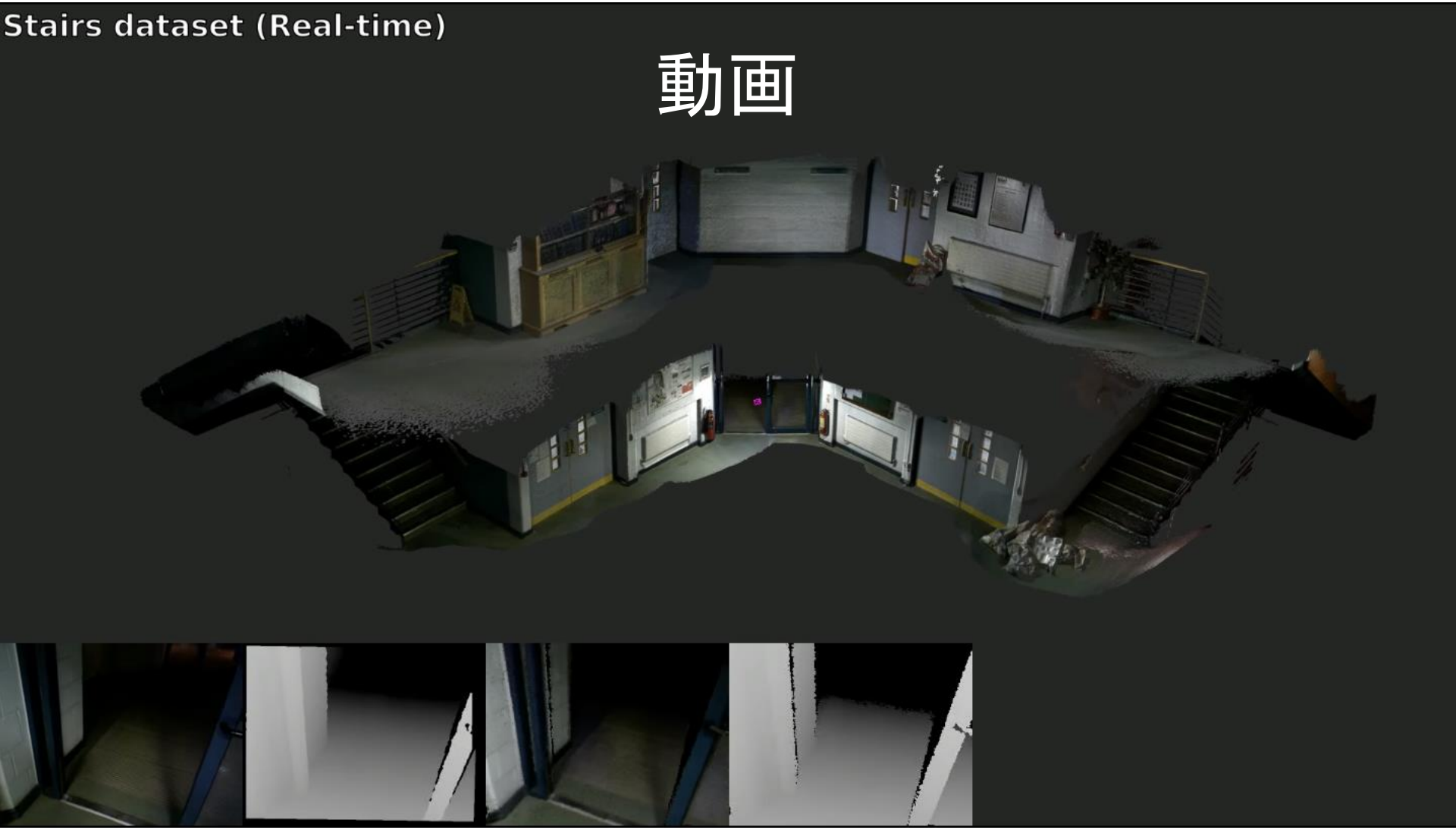
Warped Model with Motion Field Trails
(Correspondence trails from last 15 frames)

Live Model Output

https://www.youtube.com/watch?v=i1eZekcc_lM

活用事例(2) SLAM

ElasticFusion: Dense SLAM Without A Pose Graph. T. Whelan, S. Leutenegger, B. Glocker, R. F. Salas-Moreno, and A. Davison, Robotics: Science and Systems (RSS), 2015.

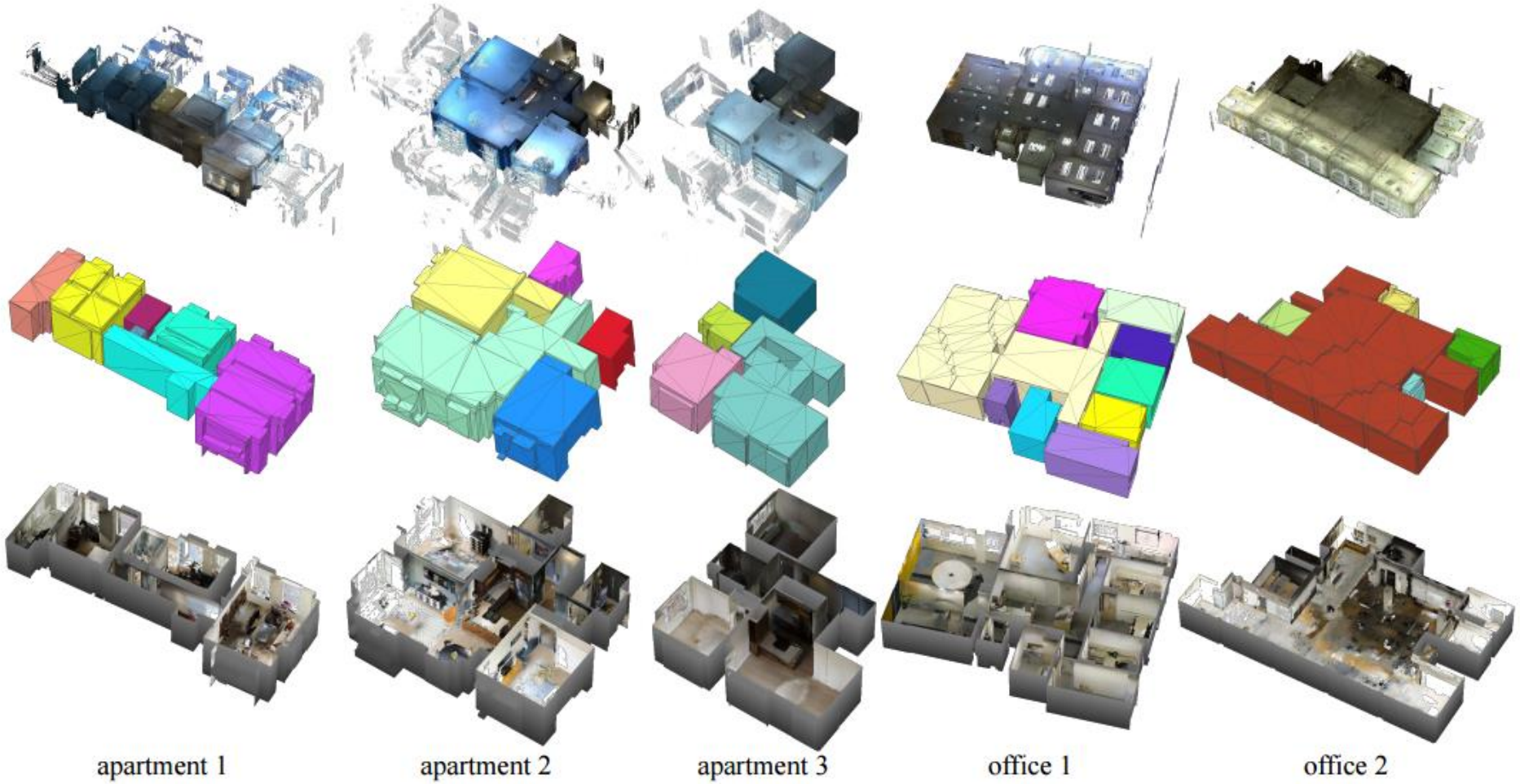


https://www.youtube.com/watch?v=-dz_VauPjEU

活用事例(3)　室内環境モデリング

Structured Indoor Modeling. S. Ikehata, H. Yan, and Y. Furukawa, **ICCV 2015 (oral)**

動画



https://www.youtube.com/watch?v=RZU8w3uvenU

活用事例(4)　遠隔コミュニケーション

holoportation: virtual 3D teleportation in real-time (Microsoft Research)

https://www.youtube.com/watch?v=7d59O6cfaM0

お品書き

1. RGBDの歴史
   - 3D特徴量の紹介
   - RGBD研究の分類と研究例
2. チュートリアル
   - 3Dデータの読み込みと表示
   - Point Cloud Library (PCL)の使い方
   - その他のオープンソースライブラリの紹介
3. 3D物体認識の最新動向
   - ディープラーニングと大規模データセット
4. まとめ

3D特徴量の紹介

3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

↑

主にPCLに入っているものを紹介する

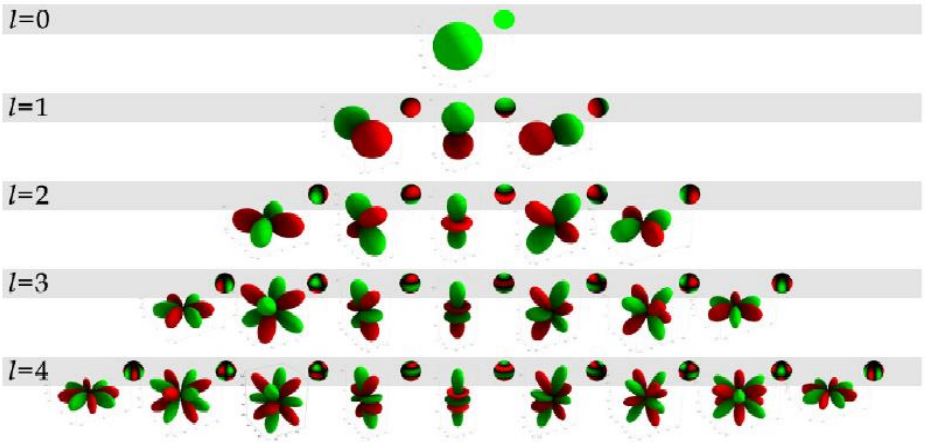
参考:

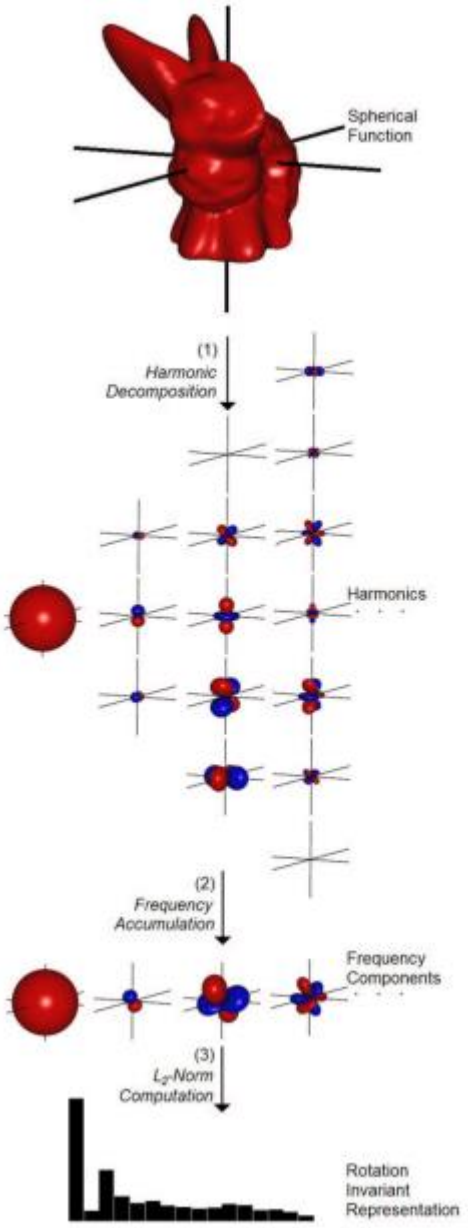
3D Model Retrieval Using Probability Density-Based Shape Descriptors. C. B. Akgül, B. Sankur, Y. Yemez, and F. Schmitt, PAMI, 2009

3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

$$f(\theta,\phi)=\sum_{l=0}^{\infty}\sum_{m=-l}^{+l}c_l^m Y_l^m(\theta,\phi)$$





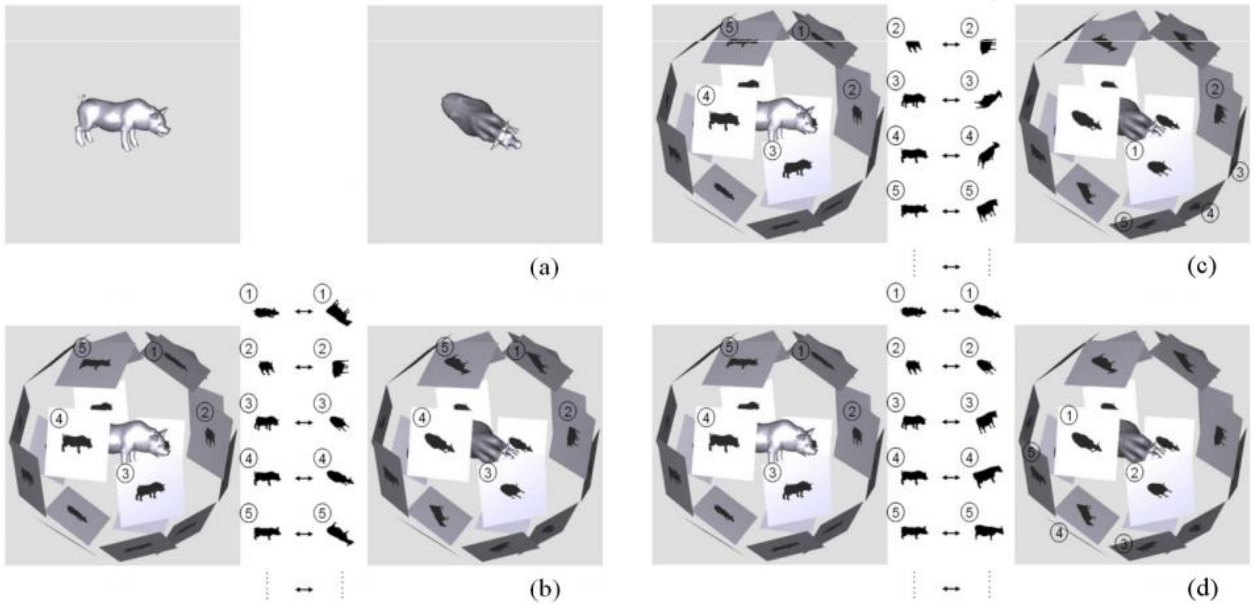
Spherical Harmonic Representations

Kazhdan, M., Funkhouser, T., & Rusinkiewicz, S. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (pp. 156-164), 2003.

3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

3Dモデル検索→ http://3d.csie.ntu.edu.tw/~dynamic/

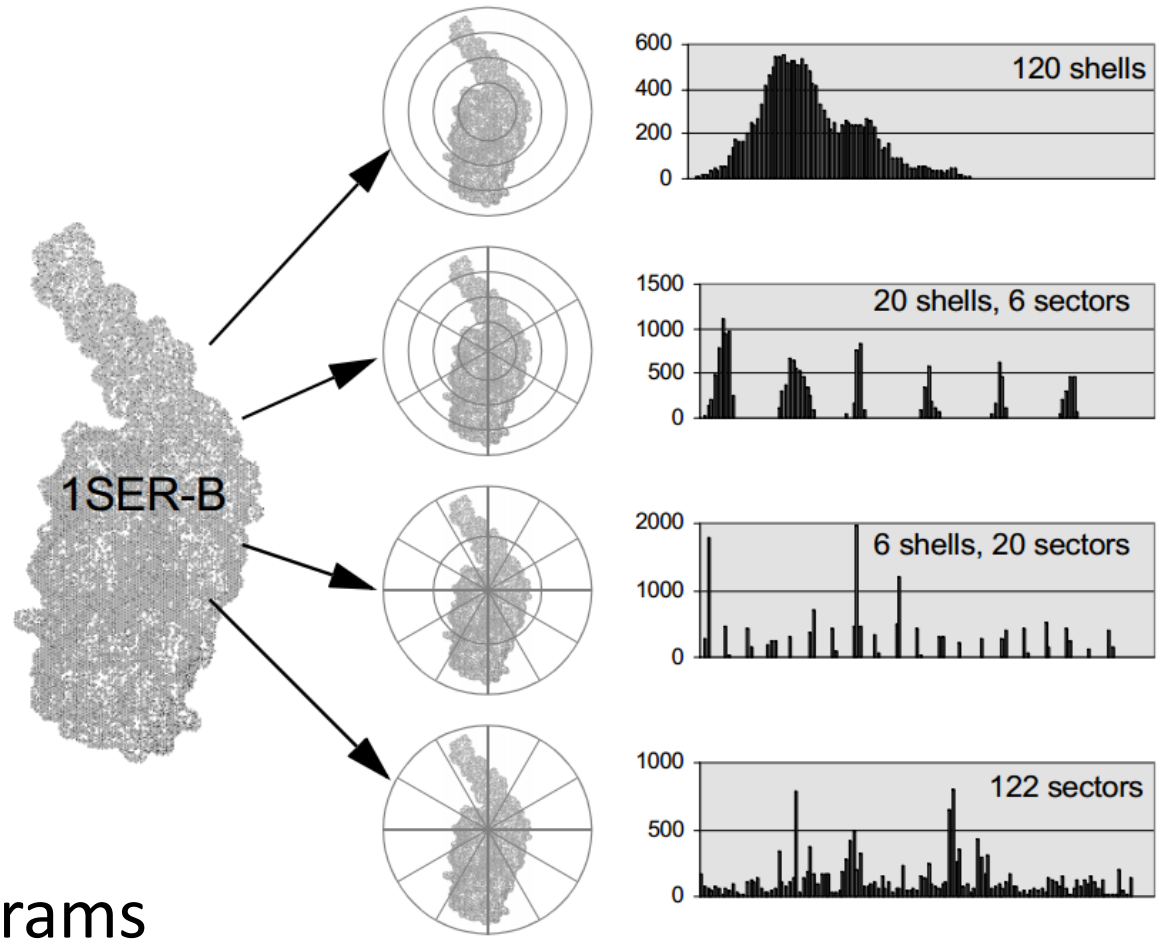


Light Field Descriptor

Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen and Ming Ouhyoung, "On Visual Similarity Based 3D Model Retrieval", *Computer Graphics Forum (EUROGRAPHICS'03)*, Vol. 22, No. 3, pp. 223-232, Sept. 2003.

3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

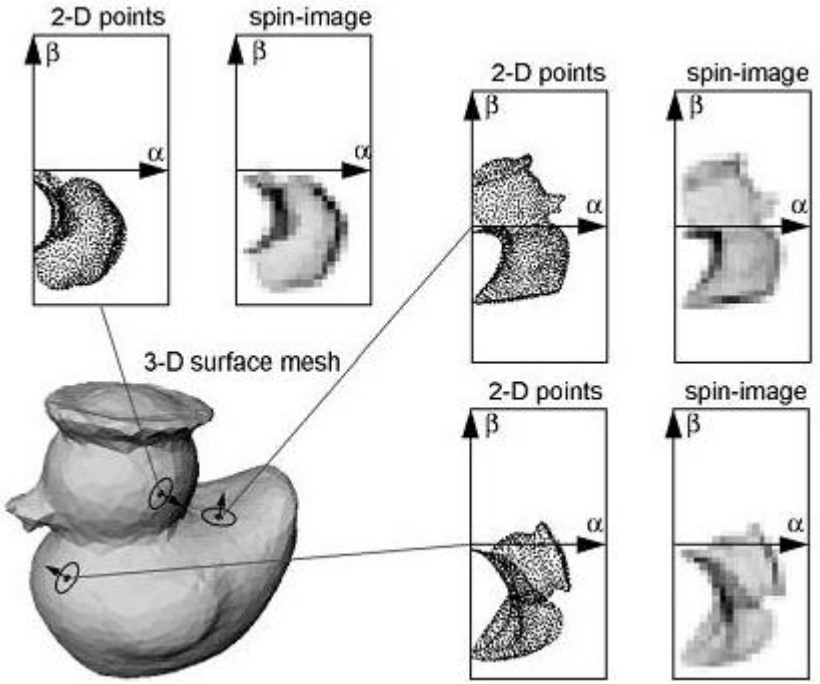


3D Shape Histograms

Ankerst, M., Kastenmüller, G., Kriegel, H. P., & Seidl, T. (1999, January). 3D shape histograms for similarity search and classification in spatial databases. In *Advances in Spatial Databases* (pp. 207-226), 1999.

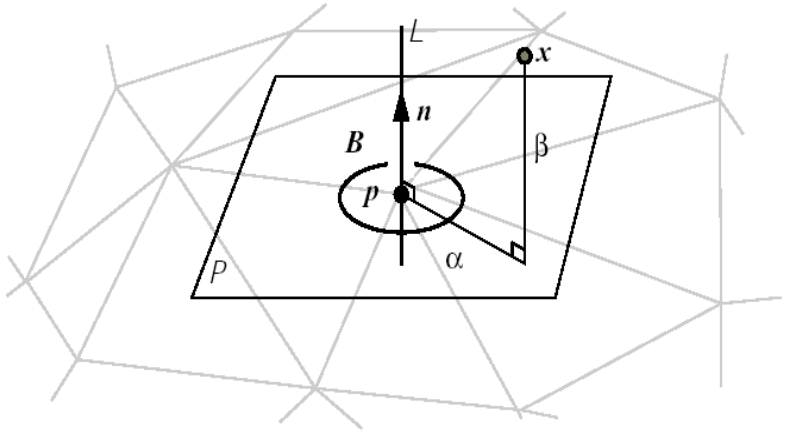
3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース



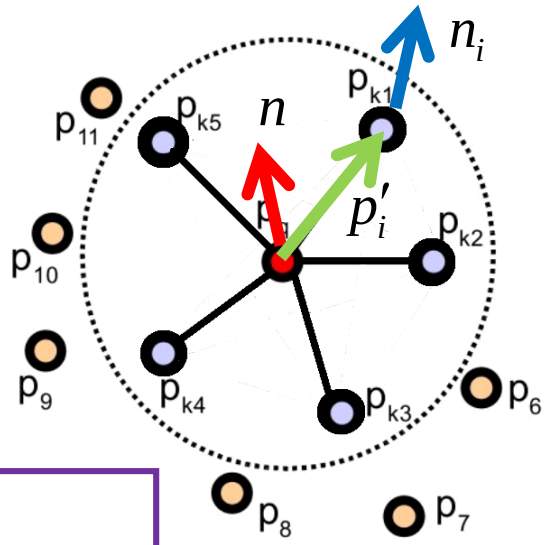
Spin Image

Johnson, Andrew E., and Martial Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.5 (1999): 433-449.



3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース



各点に対して、小球領域に含まれるk近傍点を求め、

全ペアの二点間の

パラメータα, ϕ, θを求めて

ヒストグラムを作る。

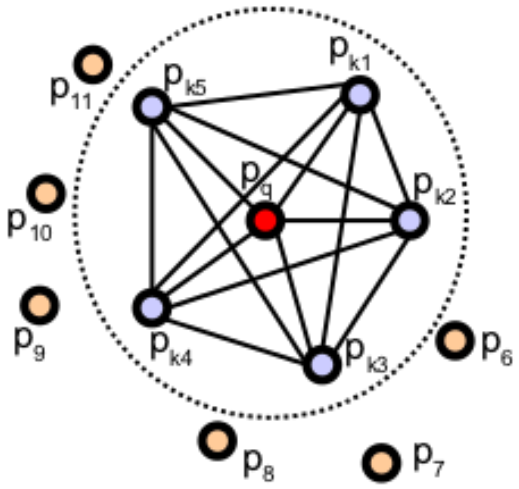
$$\alpha = (p_i' \times n) \cdot n_i$$

$$\phi = (n \cdot p_i') / \|p_i'\|$$

$$\theta = \arctan((n \times (p_i' \times n) \cdot n_i), n \cdot n_i)$$

計算量は$O(Nk^2)$

N: 点の数



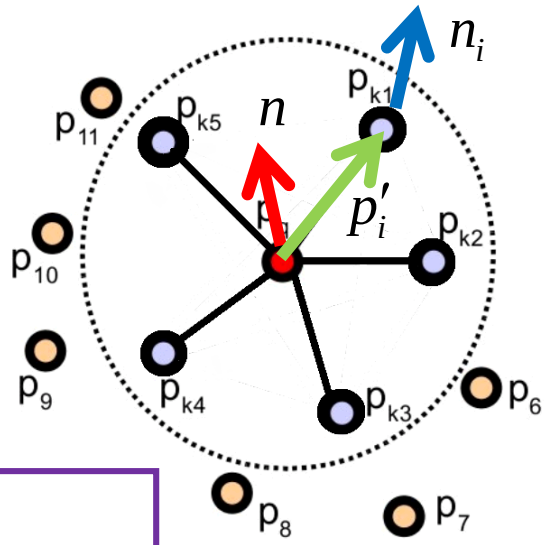
PFH

Point Feature Histogram

R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, “Aligning point cloud views using persistent feature histograms,” in Proc. Int. Conf. Intelligent Robots and Systems (IROS), 2008.

3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース



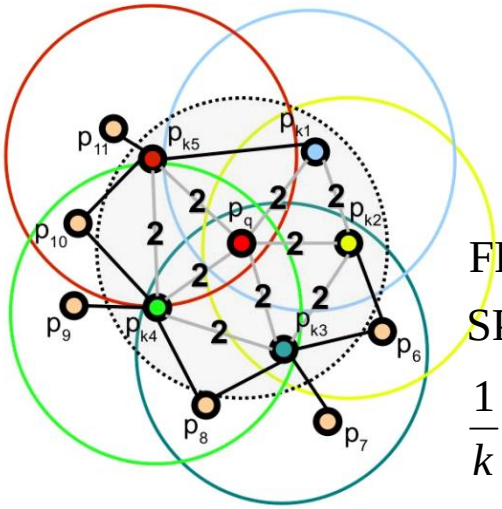
各点pに対して、小球領域に含まれるk近傍点を求め、**点pと近傍点**の二点間のパラメータα, ϕ, θを求めてヒストグラムを作る。

$\mathrm{SPFH}(p)$

$$\alpha = (p_i' \times n) \cdot n_i$$
$$\phi = (n \cdot p_i') / \|p_i'\|$$
$$\theta = \arctan((n \times (p_i' \times n) \cdot n_i), n \cdot n_i)$$

計算量は$O(Nk)$

N: 点の数



$$\mathrm{FPFH}(p) = \mathrm{SPFH}(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_i}\cdot\mathrm{SPFH}(p_i)$$

FPFH Fast Point Feature Histogram

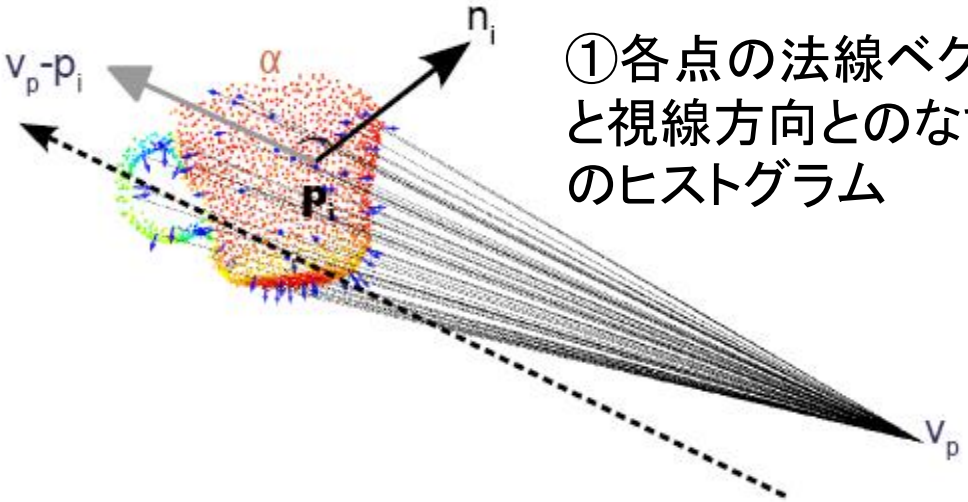
R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (FPFH) for 3-D registration,” in Proc. Int. Conf. Robotics and Automation (ICRA), 2009.

3D特徴量の紹介

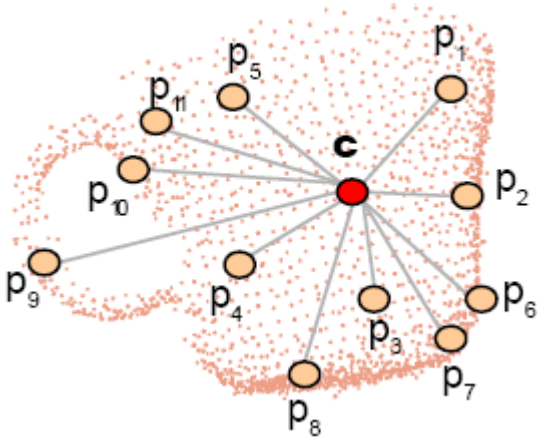
- 変換ベース
- 2Dベース
- ヒストグラムベース

視点に**依る**特徴量⇒姿勢推定に利用

②GlobalなFPFH（Extended FPFH）

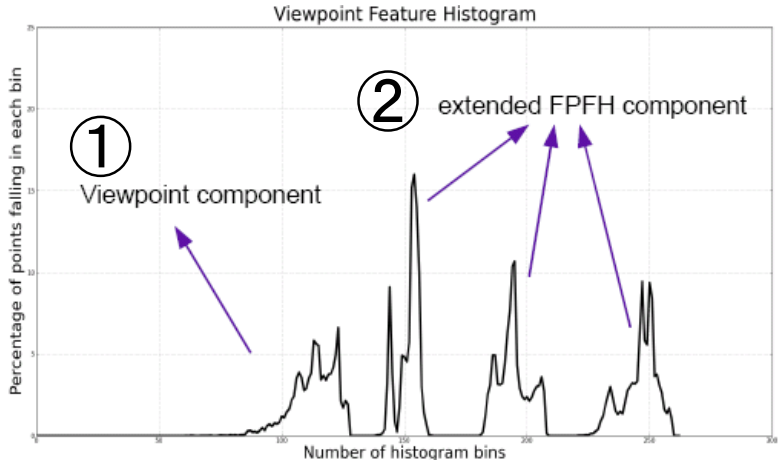


①各点の法線ベクトルと視線方向とのなす角のヒストグラム



VFH　Viewpoint Feature Histogram

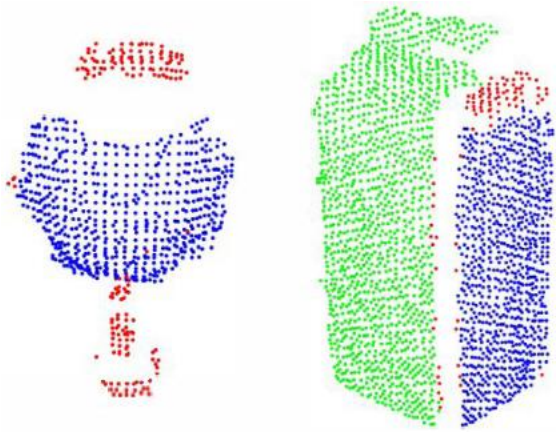
R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, “Fast 3-D recognition and pose using the viewpoint feature histogram,” in Proc. Int. Conf. Intelligent Robots and Systems (IROS), 2010.

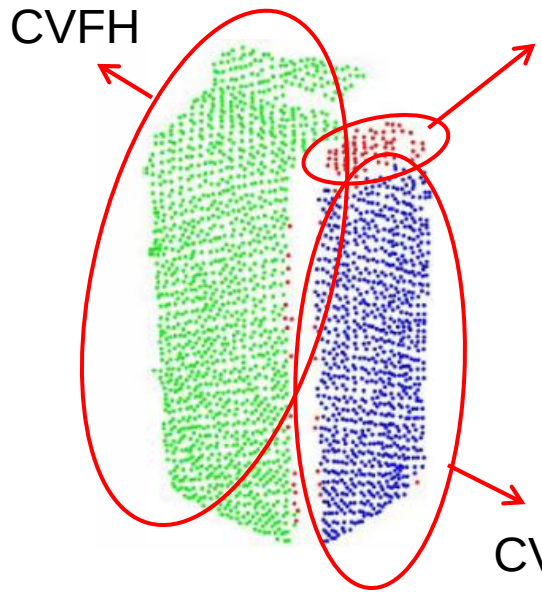


3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

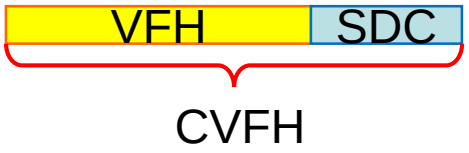
(1) 物体をパーツ領域に分け、





(2) 各パーツ領域から
VFHと同様の
ヒストグラム
（ただし中心点と
中心法線ベクトルは
各パーツ領域のもの）
を計算し、

(3) Shape Distribution
Component (SDC) を計算
してヒストグラム化し、
VFHと連結する



CVFH Clustered Viewpoint Feature Histogram

A. Aldoma, N. Blodow, D. Gossow, S. Gedikli, R. Rusu, M. Vincze, and G. Bradski, “CAD-model recognition and 6 DOF pose estimation using 3D cues,” in Proc. ICCV workshop on 3dRR, 2011

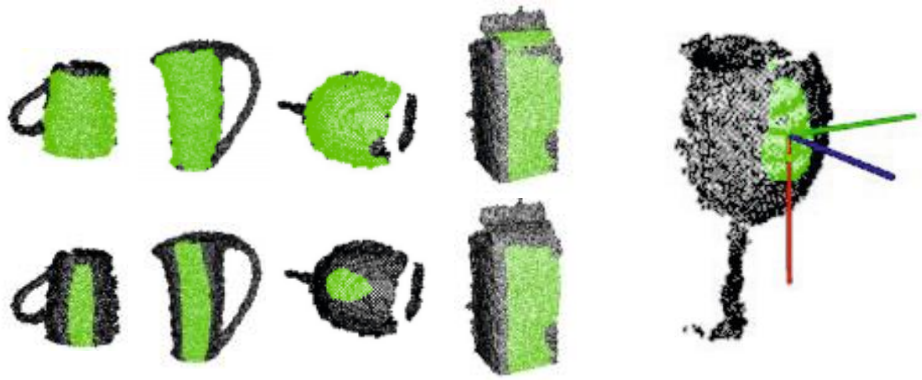
3D特徴量の紹介

- 変換ベース
- 2Dベース
- ヒストグラムベース

SGURFを求める:
パーツ領域の中心法線ベクトルから遠い点を削除

CVFH

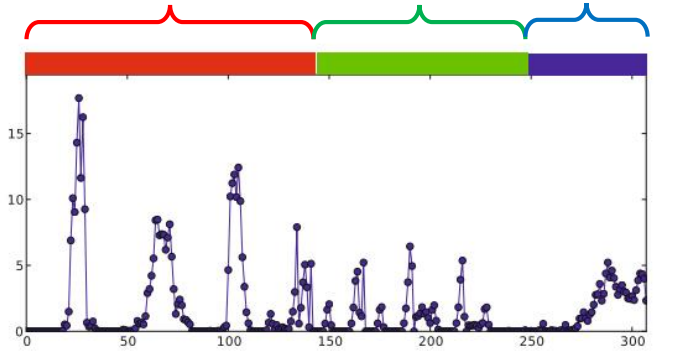
SGURF



OUR-CVFH

SGURFから

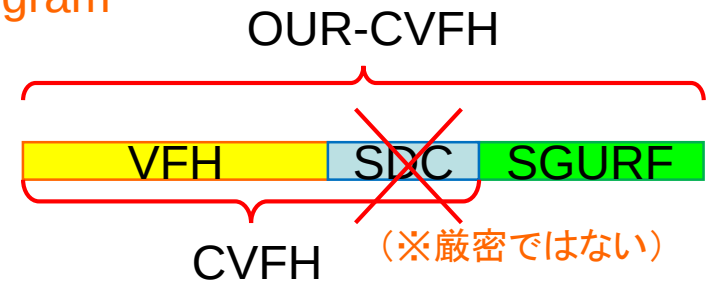
法線ヒストグラム　空間ヒストグラム　viewpoint component



OUR-CVFH

Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram

A. Aldoma, F. Tombari, R. Rusu, and M. Vincze, “OUR-CVFH – Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation”, in Joint DAGM-OAGM Pattern Recognition Symposium, 2012.

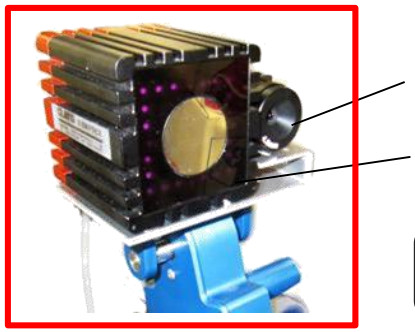


RGBD研究の分類と研究例

RGBD研究の分類と研究例

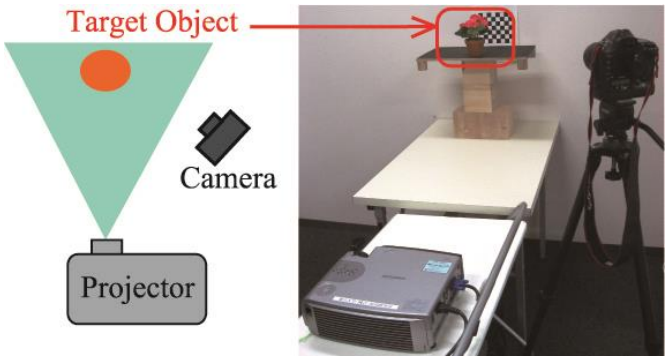
修士研究（ロボット学会2008～ICRA2011）はデータ取得に苦労した。

※ICRA2011の論文投稿〆切は2010年9月



MESA SR-4000 TOF sensor
PointGray Flea2 camera

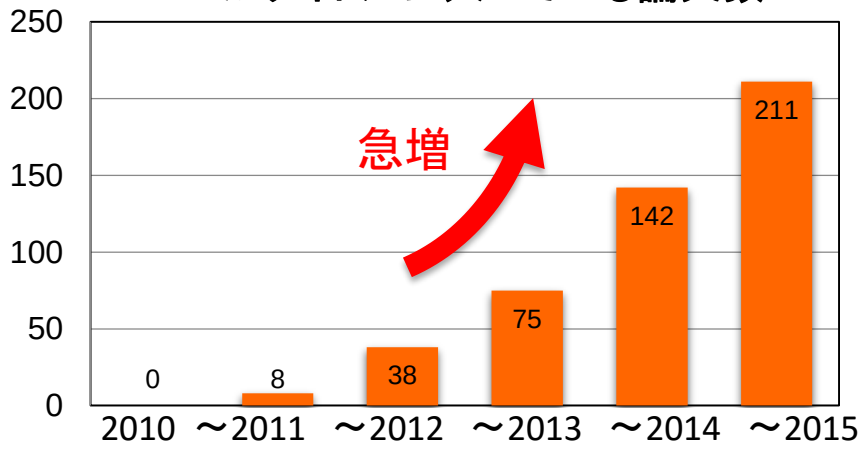
176×144 pixel、約100万円



2010年11月 Kinect登場

640×480 pixel **14,800円**

RGBDがタイトルに入っている論文数



RGBD研究の分類と研究例

- ROS 3D Contest

http://www.ros.org/news/2011/02/ros-3d-contest-the-results.html
http://wiki.ros.org/openni/Contests/ROS%203D

Overall:
1st Place ($3000): Customizable Buttons
2nd Place ($2000): Quadrotor Altitude and Obstacle Avoidance
3rd Place ($1000): Humanoid Teleoperation
4th Place ($500): Person Tracking and Reconstruction from a Mobile Base with a 7 DOF Manipulator

Most Useful:
1st Place ($2000): RGBD-6D-SLAM
2nd Place ($1000): Automatic Calibration of Extrinsic Parameters

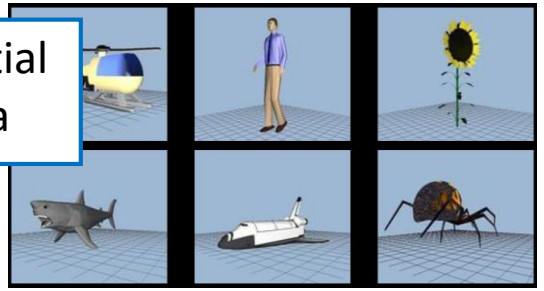
- ICRA2011

Best Vision Paper: Sparse Distance Learning for Object Recognition Combining RGB and Depth Information

Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox

初のRGBDデータセット

RGBD研究の分類と研究例

	2次元	2.5次元	3次元
色なし	 線画 *Vision*	 距離データ *Robot*	 形状データ *Graphics*
色あり	 RGB画像	 RGBD画像	 テクスチャ付形状データ

色がついた

D次元が増えた

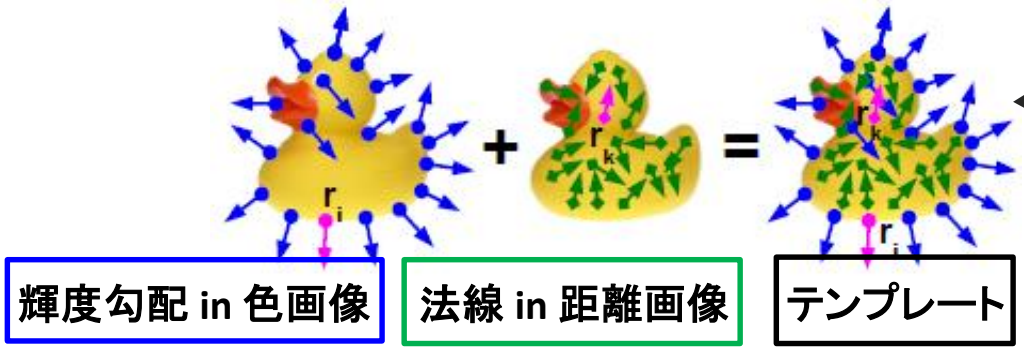
Partial data

RGBD研究の分類と研究例

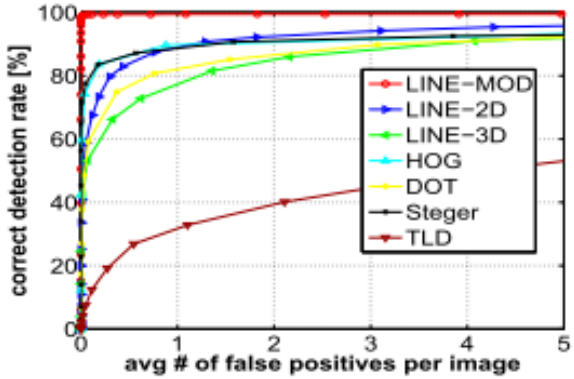
マルチモーダルフュージョン（1/2）

Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes

Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. *IEEE ICCV*, 2011.



テクスチャレスな物体を表現するのに
表面形状の情報（法線）で補おう



can parse a VGA image with over **3000 templates** with about **10 fps** on the **CPU**

RGBD研究の分類と研究例

マルチモーダルフュージョン（2/2）

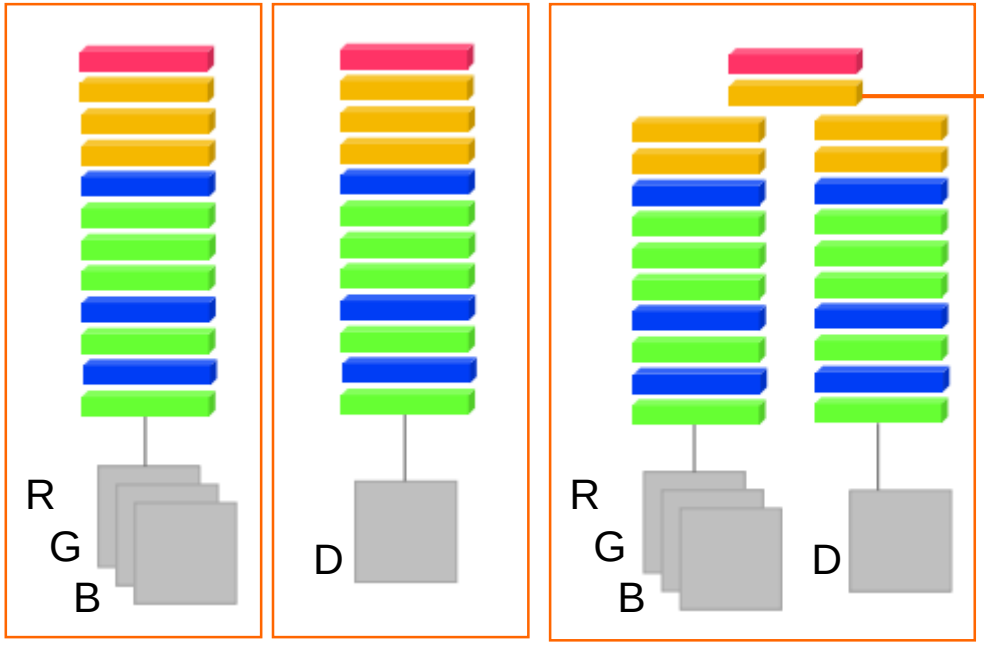
MMSS: Multi-modal Sharable and Specific Feature Learning for RGB-D Object Recognition

Anran Wang, Jianfei Cai, Jiwen Lu, and Tat-Jen Cham.

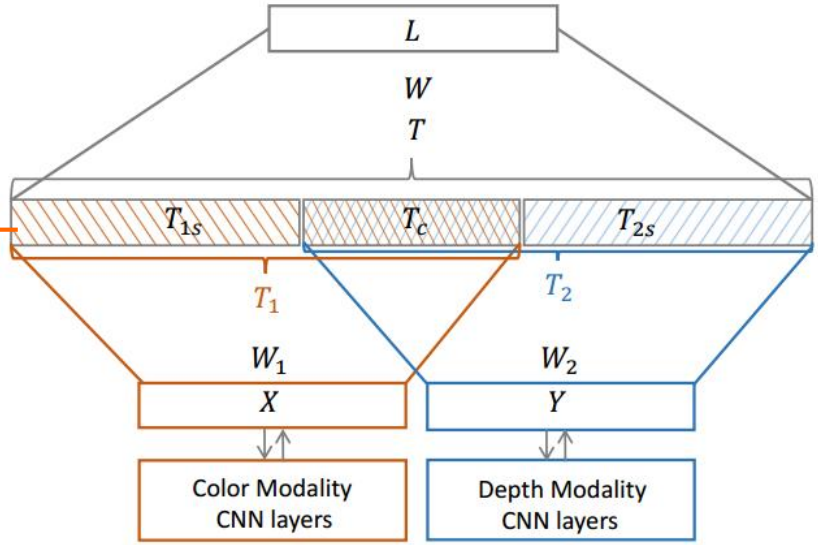
IEEE ICCV, 2015.

pre-training

multi-modal feature learning



最後のfully-connected層で
RGBとDepthが共通部分を持つよう
Deep CNNを学習する



RGBD研究の分類と研究例

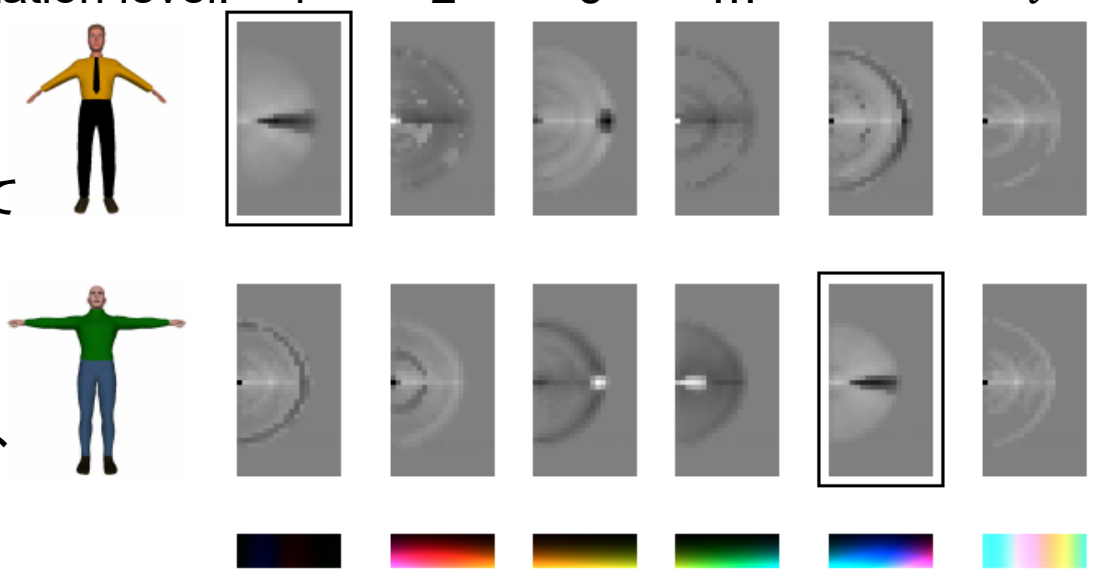
Combining color and shape descriptors for 3D model retrieval. Pasqualotto et al., Signal Processing: Image Communication 28.6, 2013.

Quantization level	luminance (L)	hue angle (h_{ab})
1	$L < 10$	$\forall$
2	$10 < L < 98$	$-0,35 < \theta < 1,4$
3	$10 < L < 98$	$1,4 < \theta < 1,92$
4	$10 < L < 98$	$1,92 < \theta < 3,42$
5	$10 < L < 98$	$3,42 < \theta < 5,93$
6	$L > 98$	$\forall$

点群特徴量
＋カラー（1/2）

Color Spin Image

カラースペースをl段階に分割し
各段階の色を持つ点だけを抽出して
Spin Imageを作る。
色分けしない点群から抽出する
Spin Imageとあわせて
計$(l+1)$個のSpin Imageを抽出し、
（各々PCAをかけて）連結する。

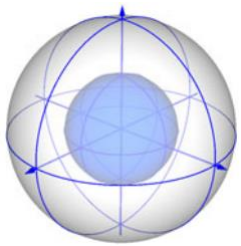


RGBD研究の分類と研究例

点群特徴量
＋カラー（2/2）

Unique signatures of histograms for local surface description [Tombari et al., ECCV2010]

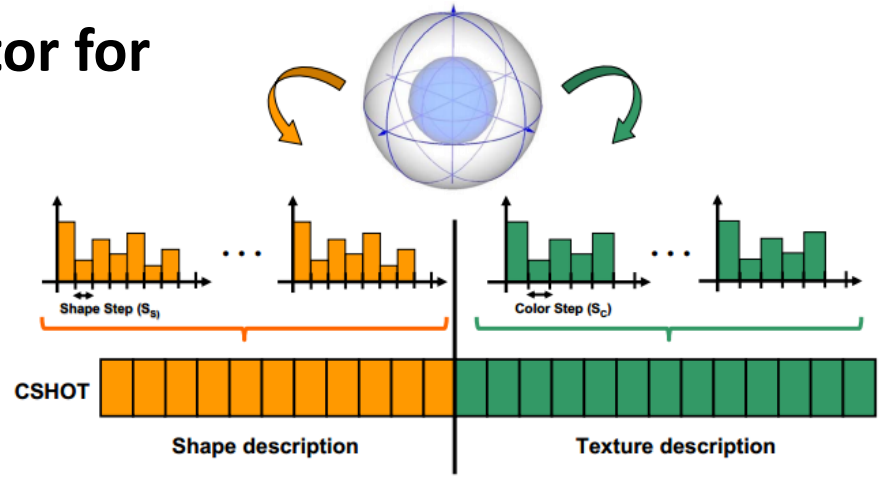
- SHOT記述子
  - デファクトスタンダードな3D点群記述子
  - PCLにも実装されている



各点まわりの局所領域を 8 x 2 x 2 に分割
各領域の法線ベクトル$\boldsymbol{n}_{v_i}$と
点の法線ベクトル$\boldsymbol{n}_u$の内積$cos\theta_i = \boldsymbol{n}_{v_i} \cdot \boldsymbol{n}_u$
のヒストグラム

A combined texture-shape descriptor for enhanced 3D feature matching
[Tombari et al., ICIP2011]

- CSHOT記述子
  - SHOTのカラー版



RGBD研究の分類と研究例

The Partial View Heat Kernel Descriptor for 3D Object Representation [Brandao et al., ICRA2014]

- Heat Kernel Signature (HKS) 記述子を、
  Partial Viewなデータの記述向けに拡張した。

＋テクスチャも考慮

cf.) Heat Kernel Signature (HKS)

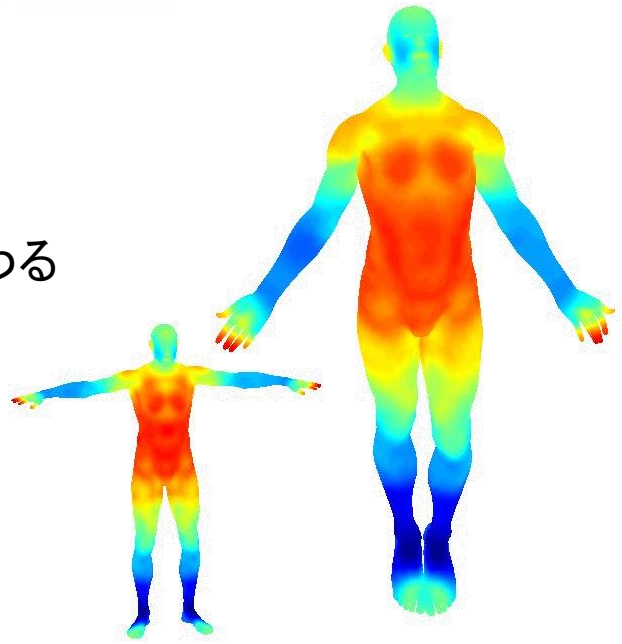
全周モデル向け。(non-rigidにもrigidにも使える)

各点の記述子は、物体全体の表面形状から計算される。

⇒ 視点が変わると見えてる部分が変わるので、HKSも変わる

$$k(v_j, v_s, t) = \sum_{i=1}^{N} e^{-\lambda_i t} \phi_{i,j} \phi_{i,s}$$

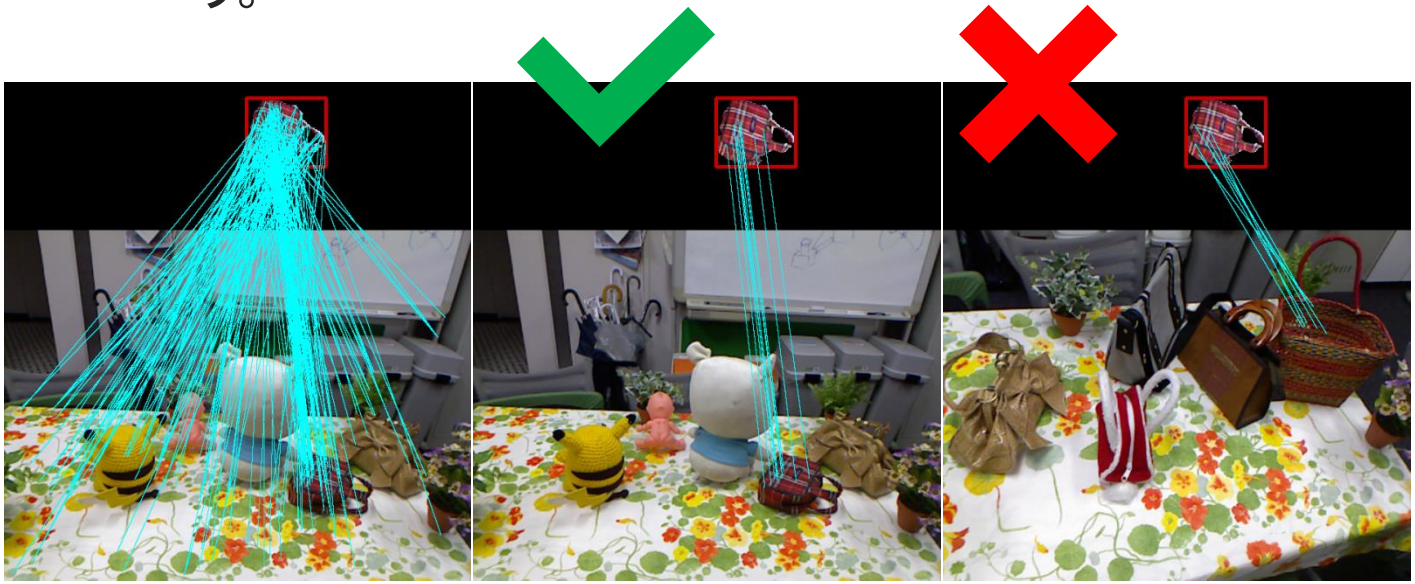
物体全体表面のLaplace-Beltrami作用素の
固有値と固有ベクトル←partial viewになると変化。



RGBD研究の分類と研究例

Learning Similarities for Rigid and Non-Rigid Object Detection [Kanezaki et al., 3DV2014]

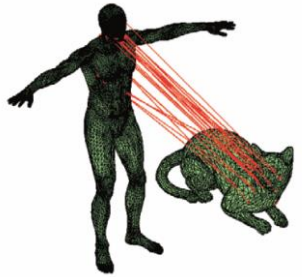
- QAPによるグラフマッチングに用いる類似度行列の学習手法を提案した。
- RGBD画像を用いた剛体物体検出と人エモデルの非剛体物体検出を統一的なフレームワークで扱う。



(a) Initial correspondences (b) Selected correspondences (c) Selected correspondences

true positive　　false positive

true positive



false positive

チュートリアル

https://goo.gl/HxH8cG

Ubuntu PCをお持ちの方は

本チュートリアルをより楽しむために…

1. ROSをインストール
   http://wiki.ros.org/jade/Installation/Ubuntu ※Desktop-Full推奨

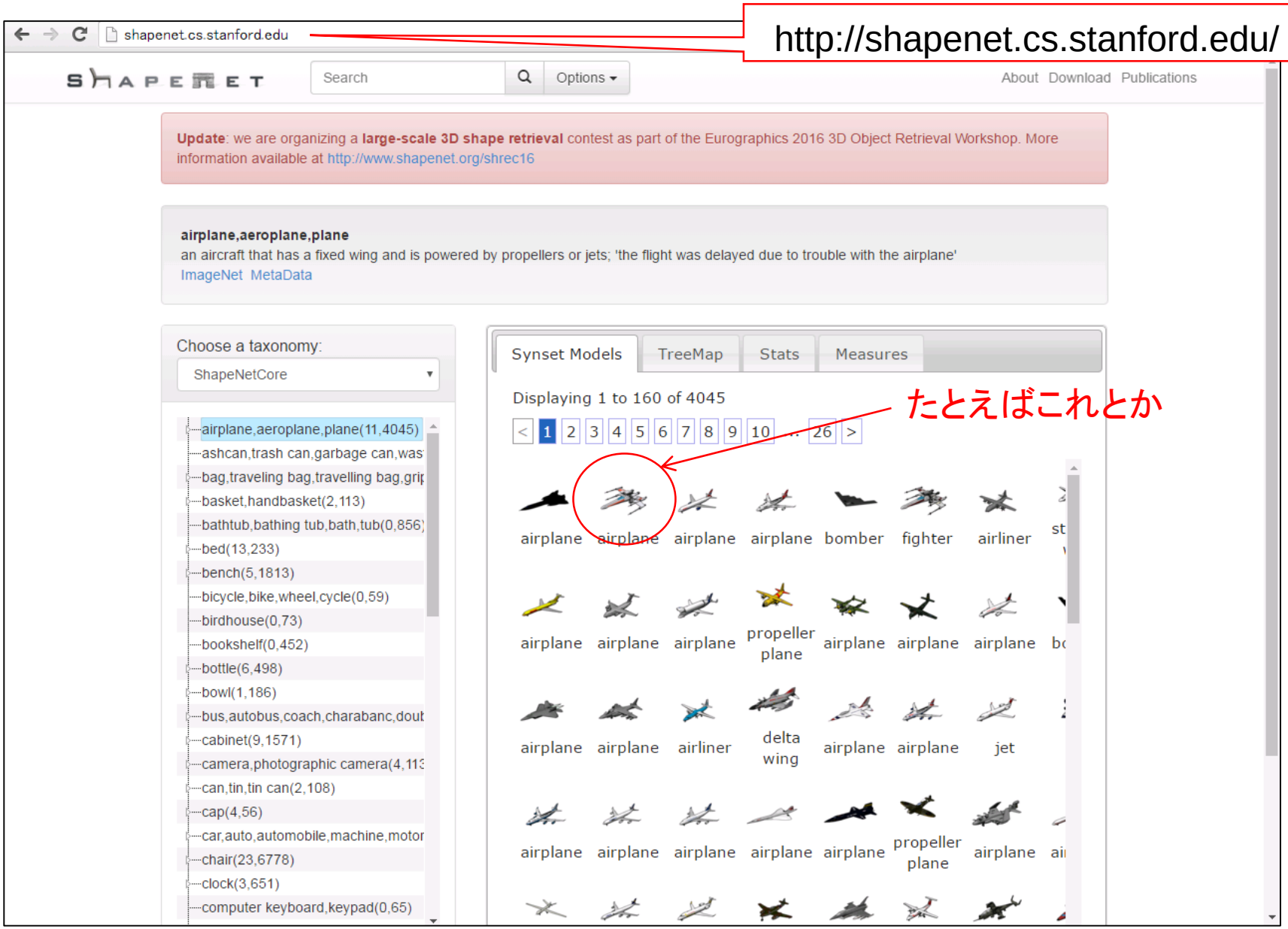
2. 必要なファイルをダウンロード

mydesk.bag（453MB）
https://www.dropbox.com/s/sn0w59sg81bhzm9/mydesk.bag?dl=0
save_pcd.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/save_pcd.cpp
convertpcd2ply.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/convertpcd2ply.cpp
milk.pcd
https://github.com/PointCloudLibrary/pcl/blob/master/test/milk.pcd?raw=true
milk_cartoon_all_small_clorox.pcd
https://github.com/PointCloudLibrary/pcl/blob/master/test/milk_cartoon_all_small_clorox.pcd?raw=true
correspondence_grouping.cpp
https://github.com/kanezaki/ssii2016_tutorial/blob/master/correspondence_grouping.cpp

真のチュートリアル

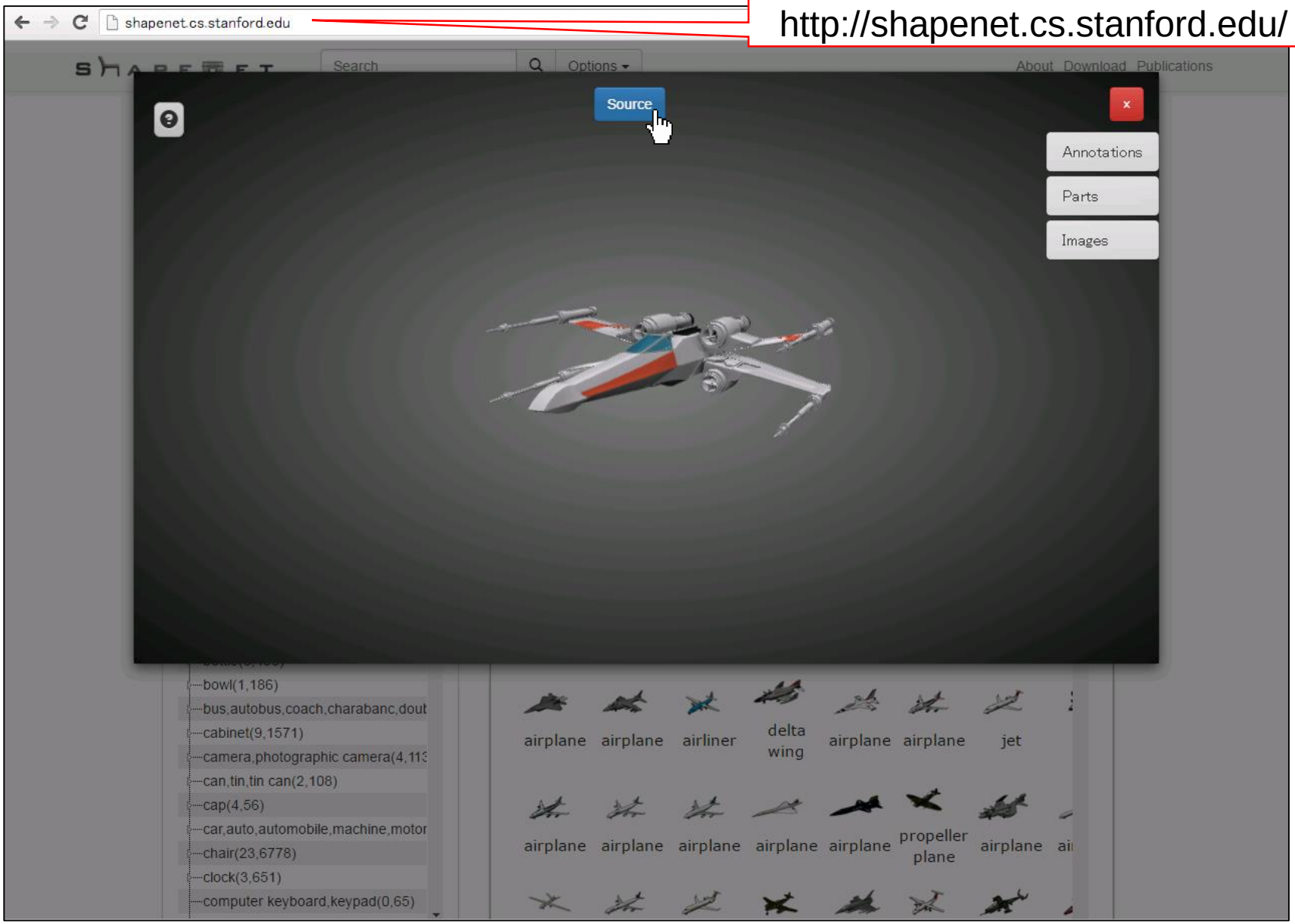
1. 3Dデータの読み込みと表示

3Dモデルをダウンロードしてみる

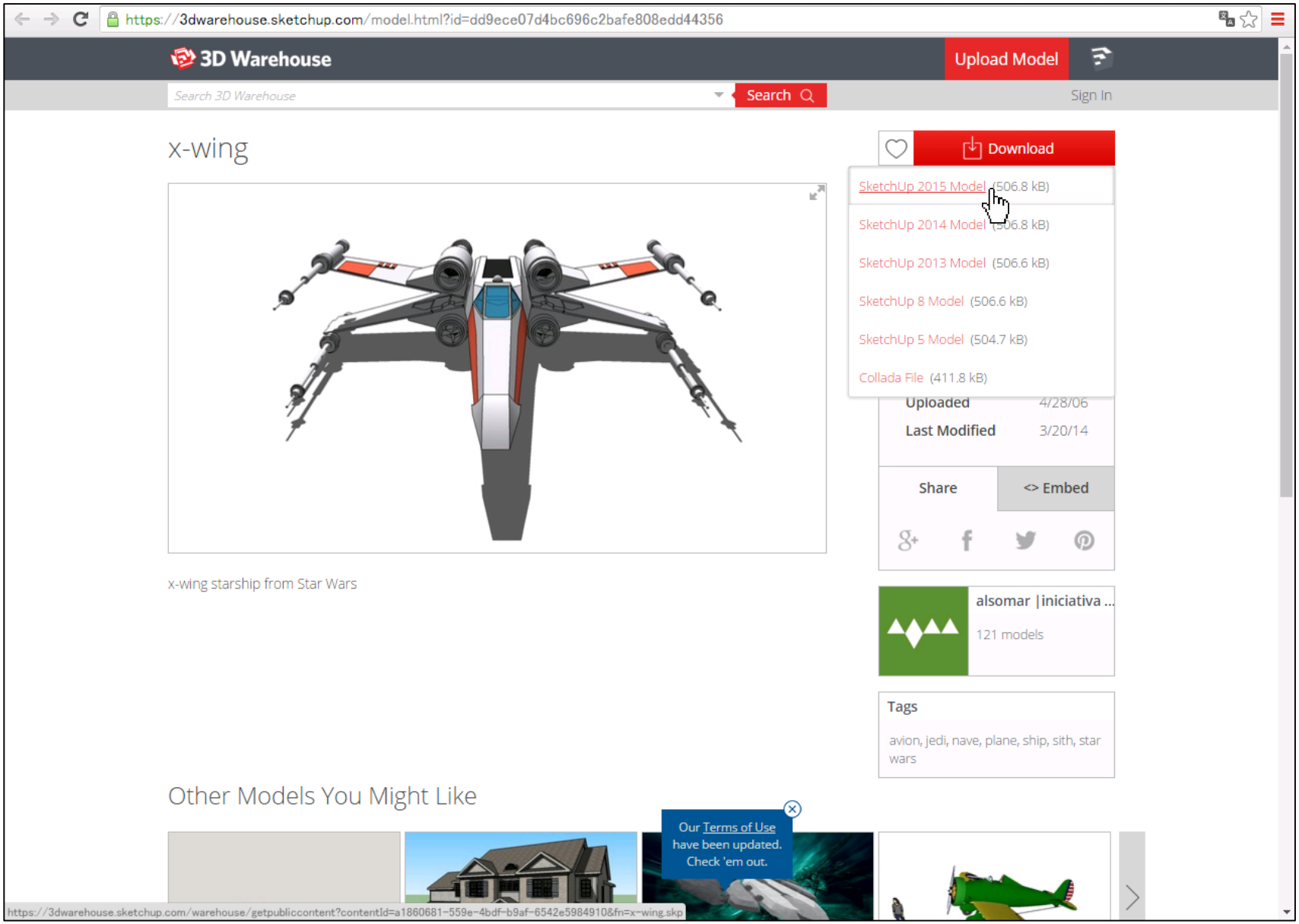


3Dモデルをダウンロードしてみる

http://shapenet.cs.stanford.edu/

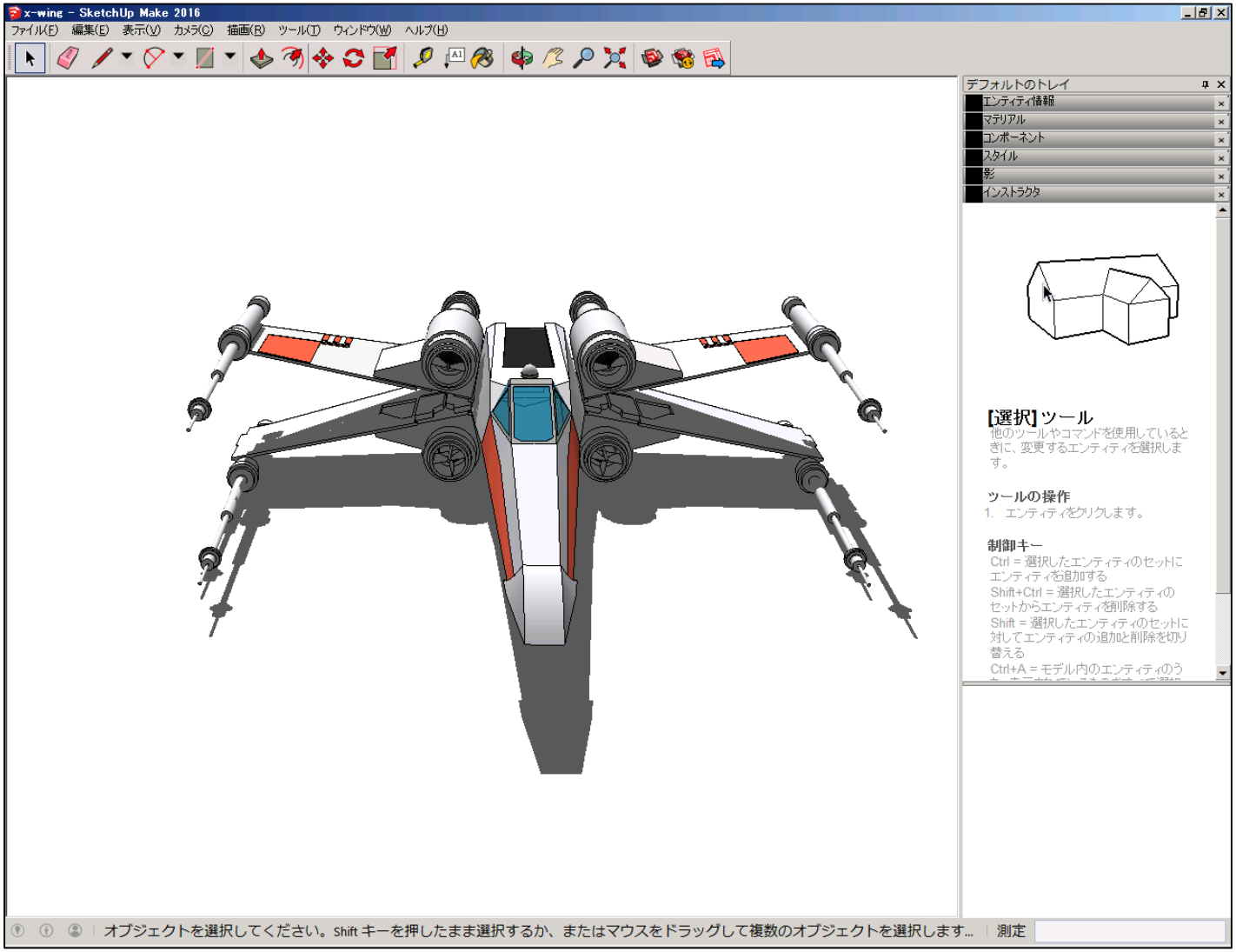


3Dモデルをダウンロードしてみる



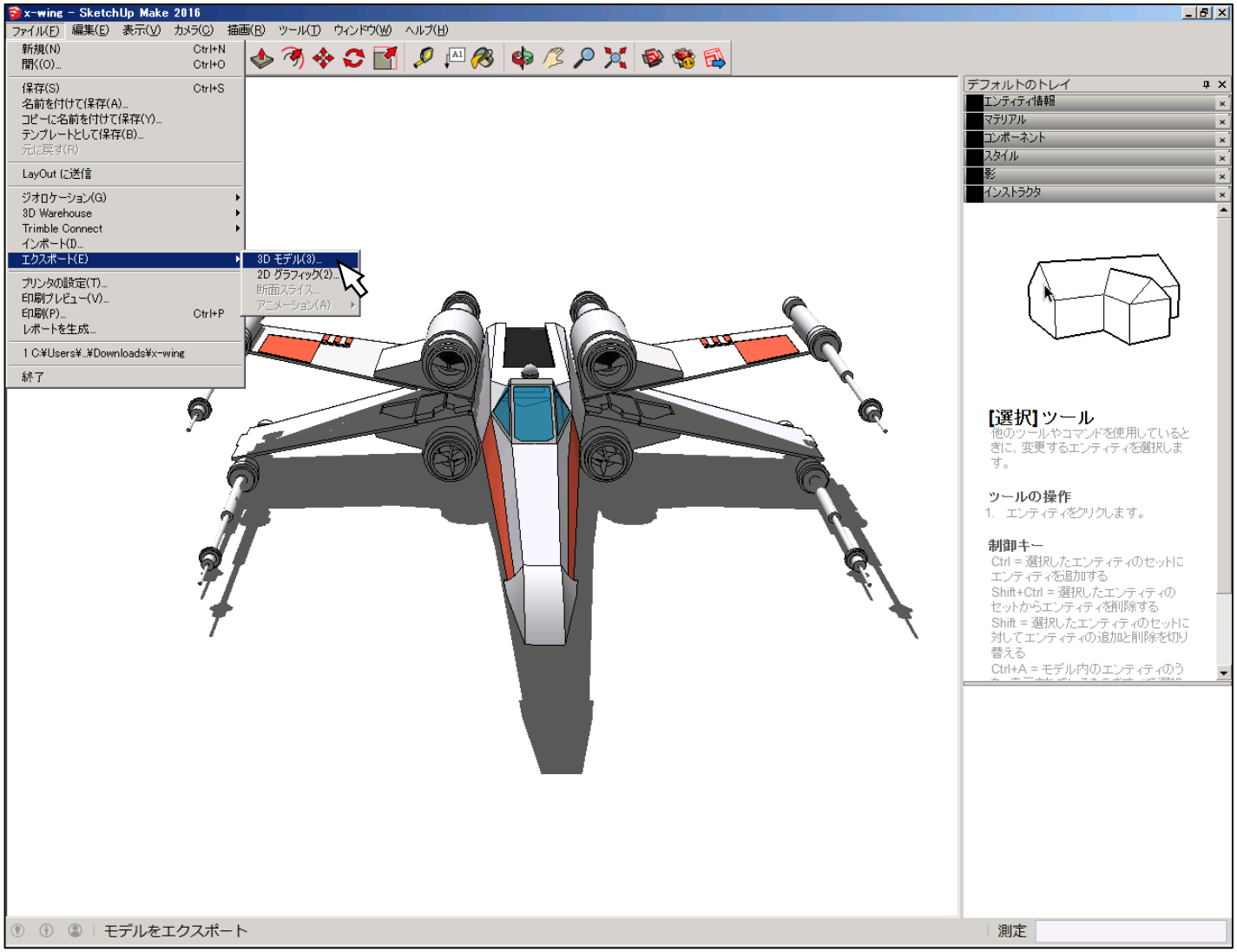
3Dモデルをダウンロードして見る

- SketchUpをダウンロードして https://www.sketchup.com/ja/download
- 先ほどダウンロードした3Dモデル（x-wing.skp）を開く



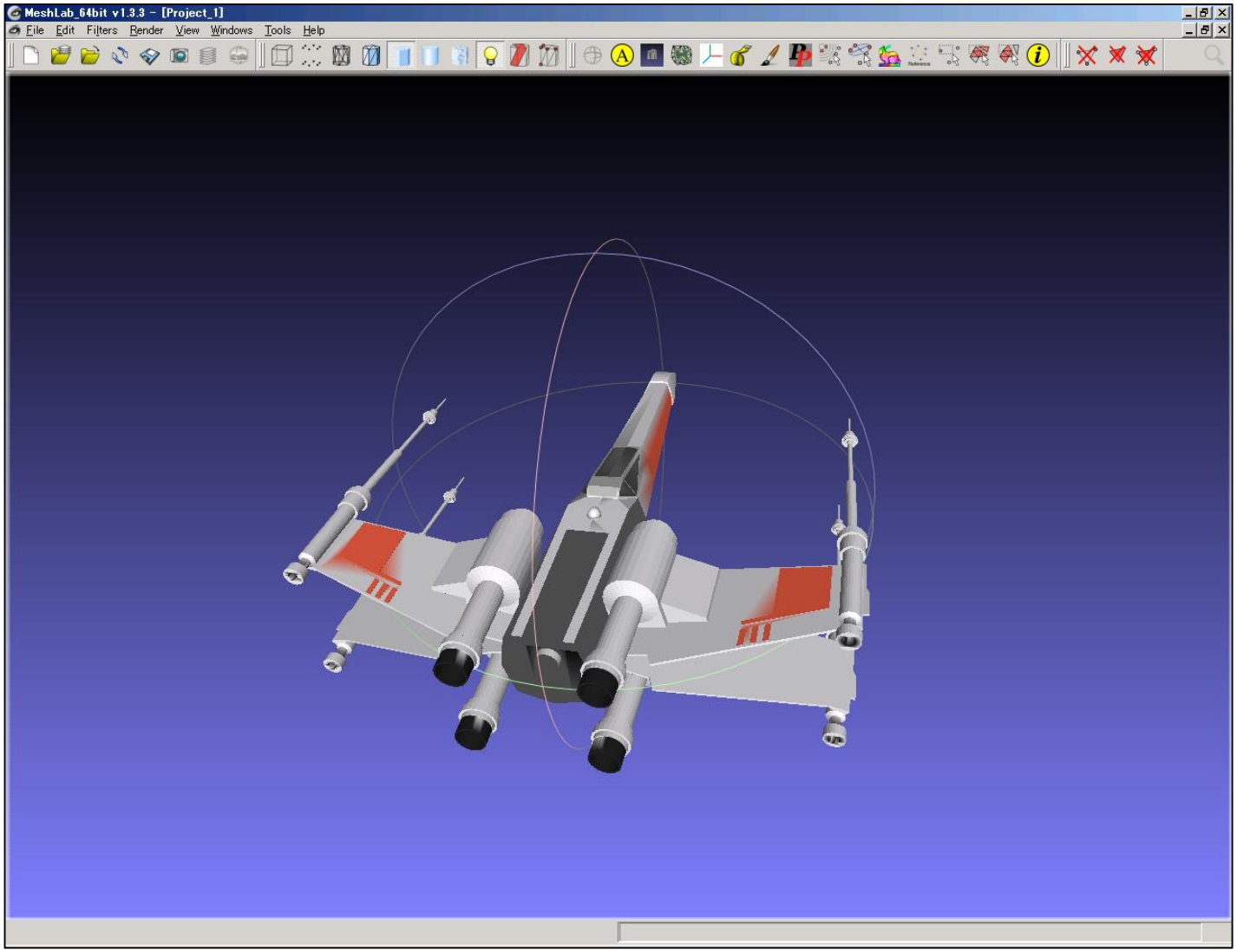
3Dモデルをダウンロードして見る

- ファイル＞エクスポート＞3Dモデル　を選択し
- ファイルの種類は「OBJファイル(*.obj)」を選んでエクスポートする（x-wing.obj）



3Dモデルをダウンロードして見る

- MeshLabをダウンロードして　http://meshlab.sourceforge.net/
- 先ほどエクスポートしたモデル（x-wing.obj）をドラッグ&ドロップする



3Dモデルをダウンロードして見る

- OBJファイル（x-wing.obj）をワードパッド等で開いて見る

```
# Alias OBJ Model File
# Exported from SketchUp, (c) 2000-2012 Trimble Navigation Limited
# File units = meters

mtllib x-wing.mtl   マテリアルファイル名

g Mesh1 x_wing1 Model

usemtl _Chalk_
v 5.56702 2.48864 -4.62216
vt 3.06437 65.0011
vn 0.891 0.454004 -5.00661e-014
v 5.56537 2.48864 -4.64307
vt 3.89005 65.0011
vn 0.88003 0.454004 -0.139383
v 5.55322 2.50871 -4.64115
vt 3.85206 65.927
vn 0.799046 0.5878 -0.126556
v 5.55472 2.50871 -4.62216
vt 3.10236 65.927
vn 0.809006 0.5878 -3.62843e-014
f 1/1/1 2/2/2 3/3/3 4/4/4

v 5.57602 2.
vt 3.91787 70.4035
vn 0.939344 0.309028 -0.148777
vt 3.89005 71.3297
vt 3.06437 71.3297
f 5/5/5 6/6/6 2/7/2 1/8/1

v 5.58152 2.44401 -4.62216
vt 3.01957 74.0644
```

頂点（vertex）

頂点座標 →
テクスチャ座標 →
法線ベクトル →

面（face）

頂点情報 →

頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号

x-wing.mtl

```
#
## Alias OBJ Material File
# Exported from SketchUp, (c) 2000-2012 Trimble Navigation Limited

newmtl _Chalk_
Ka 0.000000 0.000000 0.000000
Kd 0.909804 0.905882 0.925490
Ks 0.330000 0.330000 0.330000

newmtl _Concrete_rough_
Ka 0.000000 0.000000 0.000000
Kd 0.658824 0.662745 0.647059
Ks 0.330000 0.330000 0.330000

newmtl _Tomato_
Ka 0.000000 0.000000 0.000000
Kd 1.000000 0.388235 0.278431
Ks 0.330000 0.330000 0.330000

newmtl _Charcoal_
Ka 0.000000 0.000000 0.000000
Kd 0.137255 0.137255 0.137255
Ks 0.330000 0.330000 0.330000

newmtl _Glass_Blue_Tint_
Ka 0.000000 0.000000 0.000000
Kd 0.035294 0.686275 0.945098
Ks 0.330000 0.330000 0.330000
d 0.480000
```

3Dモデルをダウンロードして見る

- OBJファイル（x-wing.obj）をワードパッド等で開いて見る

メッシュ（ポリゴン）：
面にテクスチャが貼られる

点群（Point Cloud）：
点に色情報がついている

```
# Alias OBJ Model File
# Exported from SketchUp, (c) 2000-2012 Trimble Navigation Limited
# File units = meters

mtllib x-wing.mtl   マテリアルファイル名

g Mesh1 x_wing1 Model

usemtl _Chalk_
v 5.56702 2.48864 -4.62216
vt 3.06437 65.0011
vn 0.891 0.454004 -5.00661e-014
v 5.56537 2.48864 -4.64307
vt 3.89005 65.0011
vn 0.88003 0.454004 -0.139383
v 5.55322 2.50871 -4.64115
vt 3.85206 65.927
vn 0.799046 0.5878 -0.126556
v 5.55472 2.50871 -4.62216
vt 3.10236 65.927
vn 0.809006 0.5878 -3.62843e-014
f 1/1/1 2/2/2 3/3/3 4/4/4

v 5.57602 2.
vt 3.91787 70.4035
vn 0.939344 0.309028 -0.148777
vt 3.89005 71.3297
vt 3.06437 71.3297
f 5/5/5 6/6/6 2/7/2 1/8/1

v 5.58152 2.44401 -4.62216
vt 3.01957 74.0644
```

頂点（vertex）

頂点座標 →
テクスチャ座標 →
法線ベクトル →

面（face）

頂点情報 →

頂点座標値番号/テクスチャ座標値番号/頂点法線ベクトル番号

x-wing.mtl

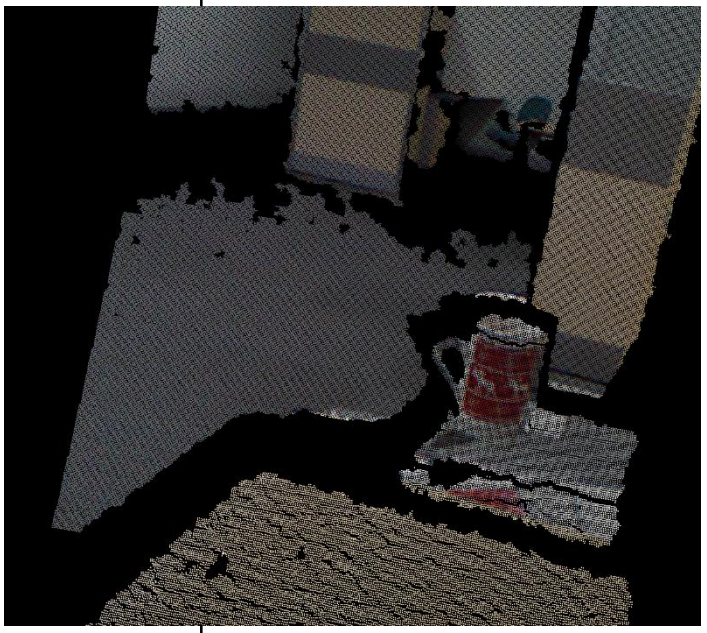
```
#
## Alias OBJ Material File
# Exported from SketchUp, (c) 2000-2012 Trimble Navigation Limited

newmtl _Chalk_
Ka 0.000000 0.000000 0.000000
Kd 0.909804 0.905882 0.925490
Ks 0.330000 0.330000 0.330000

newmtl _Concrete_rough_
Ka 0.000000 0.000000 0.000000
Kd 0.658824 0.662745 0.647059
Ks 0.330000 0.330000 0.330000

newmtl _Tomato_
Ka 0.000000 0.000000 0.000000
Kd 1.000000 0.388235 0.278431
Ks 0.330000 0.330000 0.330000

newmtl _Charcoal_
Ka 0.000000 0.000000 0.000000
Kd 0.137255 0.137255 0.137255
Ks 0.330000 0.330000 0.330000

newmtl _Glass_Blue_Tint_
Ka 0.000000 0.000000 0.000000
Kd 0.035294 0.686275 0.945098
Ks 0.330000 0.330000 0.330000
d 0.480000
```

3Dモデルをダウンロードして見る

- 点群（Point Cloud）フォーマットはPCLで用いられる.pcdファイル等がメジャー

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 640
HEIGHT 480
VIEWPOINT 0 0 0 1 0 0 0
POINTS 307200
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
0.81915 0.32 0 4.2108e+06
0.97192 0.278 0 4.2108e+06
0.944 0.29474 0 4.2108e+06
0.98111 0.24247 0 4.2108e+06
0.93655 0.26143 0 4.2108e+06
0.91631 0.27442 0 4.2108e+06
0.81921 0.29315 0 4.2108e+06
0.90701 0.24109 0 4.2108e+06
0.83239 0.23398 0 4.2108e+06
0.99185 0.2116 0 4.2108e+06
0.89264 0.21174 0 4.2108e+06
0.85082 0.21212 0 4.2108e+06
0.81044 0.32222 0 4.2108e+06
0.74459 0.32192 0 4.2108e+06
```

注：ASCII版はバグがあるので、BINARY版を使ってください



KinectからのRGBD画像・点群を扱う

ここからはROSを使います！

リアルタイム処理に便利だから

環境

- OS: Ubuntu 14.04
- センサ: Kinect v1
- 言語: C++

ROSは基本的にUbuntuしか
サポートしてないから

キャリブレーションなくてもそこ
そこ綺麗だから
Kinect v2は境界がボソボソに
なるから
（v2も使えるけどちょっと面倒です）

ASUS XtionでもOKです

Pythonも使えるので興味のある
人はどうぞ。

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

KinectからのRGBD画像・点群を扱う

手順

1. **ROSをインストールする**
2. Kinect画像を表示する
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

http://wiki.ros.org/jade/Installation/Ubuntu　に書いてあるとおりにやればよい。

jadeはROSのバージョンの名前。
自分の知る限りcturtle, diamondback, electric, fuerte, groovy, hydro, indigo, jadeが存在する

```
$ sudo sh –c ‘echo “deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main” > /etc/apt/sources.list.d/ros-latest.list‘
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
$ sudo apt-get update
$ sudo apt-get install ros-jade-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/jade/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

.bashrcに妙なものを書きたくない人は、ターミナル起動するたびに
$ source /opt/ros/jade/setup.bash　を実行すればよい。

KinectからのRGBD画像・点群を扱う

手順

1. **ROSをインストールする**
2. Kinect画像を表示する
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

Kinectドライバをラップ**した**ROSパッケージをインストールする。

```
$ sudo apt-get install ros-jade-openni-launch
```

※最近はOpenNIが提供終了したせい？か、普通には動かない模様。

```
$ sudo apt-get install ros-indigo-freenect-launch
```

残念ながらjadeでfreenect_launchパッケージがなかったのでindigoを使う
ROSのバージョンが違うパッケージも共存して使えるので無問題

豆知識
Ubuntuパッケージ名は-（ハイフン）、ROSパッケージ名は_（アンダーバー）

Xtionをお使いの方はこちら

```
$ sudo apt-get install ros-jade-openni2-launch
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. **Kinect画像を表示する**
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

Kinectを挿して、データの取り込みを開始する。

```
$ source /opt/ros/indigo/setup.sh
$ roslaunch freenect_launch freenect.launch
```

Xtionをお使いの方はこちら

```
$ roslaunch openni2_launch openni2.launch
```

これはこのまま放置して、別のターミナル（ウィンドウ or タブ）を開く。

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. **Kinect画像を表示する**
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

Kinec

$ sc
$ ro

このチュートリアルでは既に録ってあるデータを再生します。

$ roscore

放置して、別のターミナル（ウィンドウ or タブ）を開いて、

Xtion

$ rosbag play mydesk.bag -l

$ ro

詳しくは$ rosbag -h

これはこのまま放置して、別のターミナル（ウィンドウ or タブ）を開く。

mydesk.bag（453MB）→ https://www.dropbox.com/s/sn0w59sg81bhzm9/mydesk.bag?dl=0

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. **Kinect画像を表示する**
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

新しいターミナルで、ROSトピックを確認する。

```
$ rostopic list
```

たとえばカラー画像のROSトピックがpublishされていることを確認する。

```
$ rostopic hz /camera/rgb/image_color
```

```
% rostopic list
/camera/debayer/parameter_descriptions
/camera/debayer/parameter_updates
/camera/depth/camera_info
/camera/depth/disparity
/camera/depth/image
/camera/depth/image_raw
/camera/depth/image_rect
/camera/depth/image_rect_raw
/camera/depth/points
/camera/depth_rectify_depth/parameter_descriptions
/camera/depth_rectify_depth/parameter_updates
/camera/depth_registered/camera_info
```

```
% rostopic hz /camera/rgb/image_color
subscribed to [/camera/rgb/image_color]
average rate: 30.038
        min: 0.029s max: 0.037s std dev: 0.00191s window: 18
average rate: 30.056
        min: 0.027s max: 0.038s std dev: 0.00240s window: 49
average rate: 30.034
        min: 0.027s max: 0.038s std dev: 0.00242s window: 79
average rate: 30.009
        min: 0.027s max: 0.040s std dev: 0.00238s window: 109
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. **Kinect画像を表示する**
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

注：Xtion＋openni2.launchをお使いの方は、カラー画像のROSトピックが
/camera/rgb/image_raw
なので、以下、image_colorをすべてimage_rawに読み替えてください。

新しいターミナルで、ROSトピックを確認する。

```
$ rostopic list
```

たとえばカラー画像のROSトピックがpublishされていることを確認する。

```
$ rostopic hz /camera/rgb/image_color
```

```
% rostopic list
/camera/debayer/parameter_descriptions
/camera/debayer/parameter_updates
/camera/depth/camera_info
/camera/depth/disparity
/camera/depth/image
/camera/depth/image_raw
/camera/depth/image_rect
/camera/depth/image_rect_raw
/camera/depth/points
/camera/depth_rectify_depth/parameter_descriptions
/camera/depth_rectify_depth/parameter_updates
/camera/depth_registered/camera_info
```

```
% rostopic hz /camera/rgb/image_color
subscribed to [/camera/rgb/image_color]
average rate: 30.038
        min: 0.029s max: 0.037s std dev: 0.00191s window: 18
average rate: 30.056
        min: 0.027s max: 0.038s std dev: 0.00240s window: 49
average rate: 30.034
        min: 0.027s max: 0.038s std dev: 0.00242s window: 79
average rate: 30.009
        min: 0.027s max: 0.040s std dev: 0.00238s window: 109
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. **Kinect画像を表示する**
3. Kinect点群を描画する
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

カラー画像を表示する。

```
$ rosrun image_view image_view image:=/camera/rgb/image_color
```

デプス画像を表示する。

```
$ rosrun image_view image_view image:=/camera/depth/image_raw
```

rosrun構文

rosrun ＜パッケージ名＞ ＜実行ファイル名＞ ＜コマンドライン引数＞

（指定したパッケージの中の実行ファイルを実行しているだけ。）

KinectからのRGBD画像・点群を扱う

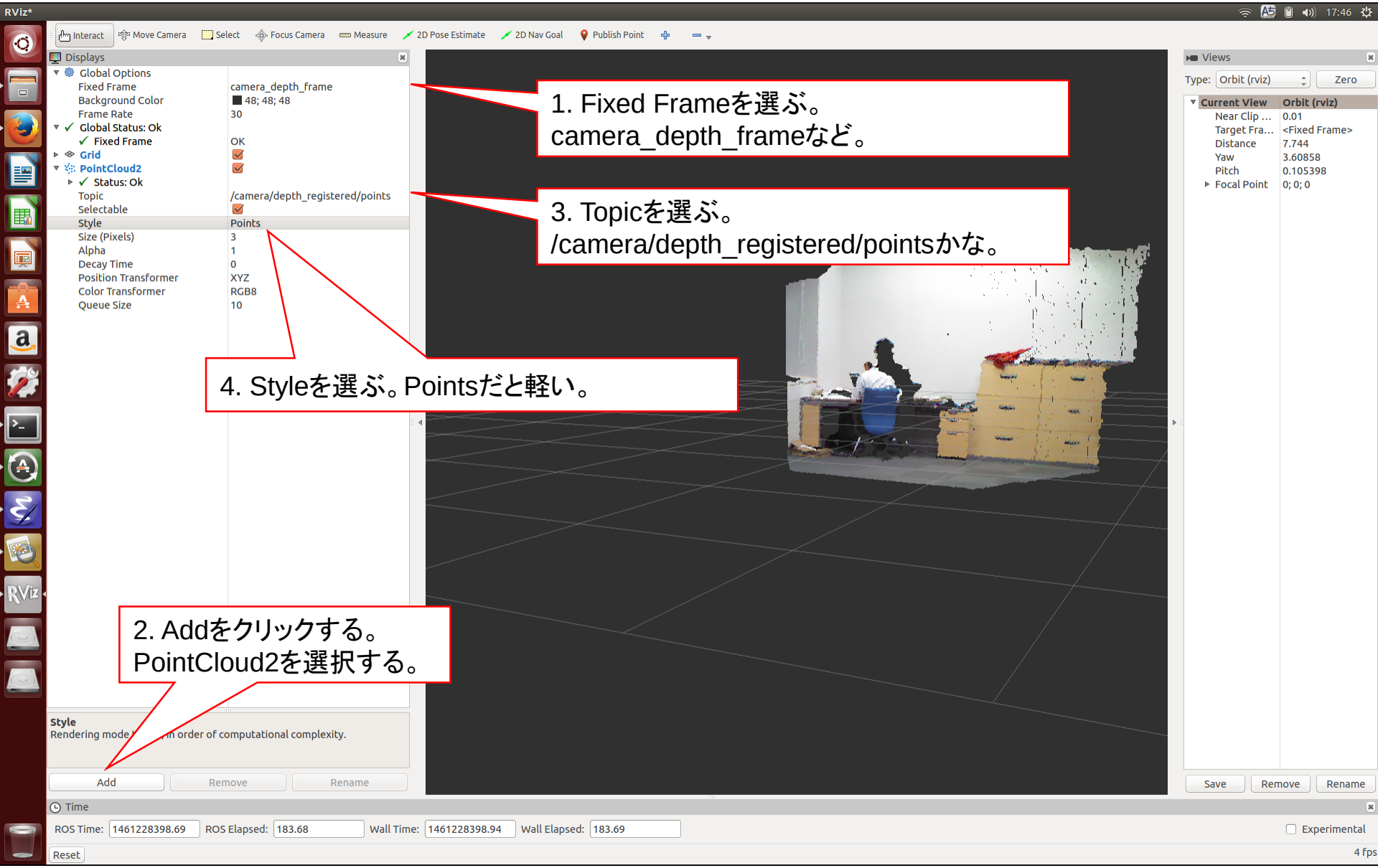
手順

1. ROSをインストールする
2. Kinect画像を表示する
3. **Kinect点群を描画する**
4. Kinect点群を保存する
5. 保存したKinect点群を描画する

ビジュアライゼーションツールのrvizを起動する。

```
$ rosrun rviz rviz
```

```
$ rosrun rviz rviz
```



KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. **Kinect点群を保存する**
5. 保存したKinect点群を描画する

自分でROSパッケージを作る。

```
$ mkdir ~/ros
$ export ROS_PACKAGE_PATH=~/ros:$ROS_PACKAGE_PATH
```

こうすることで~/rosディレクトリ以下のディレクトリがROSのパスに加わる。

```
$ cd ~/ros
$ roscreate-pkg save_pcd pcl_ros cv_bridge
$ cd save_pcd
```

~/ros/save_pcdというディレクトリができる。これがsave_pcdパッケージの雛形。

```
# roscreate-pkg構文
      roscreate-pkg  ＜今作るパッケージ名＞  ＜依存するパッケージ名＞
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. **Kinect点群を保存する**
5. 保存したKinect点群を描画する

CMakeLists.txtに下記の一文を加える。

rosbuild_add_executable(save_pcd save_pcd.cpp)

こんなかんじで。

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)

# Set the build type.  Options are:
#  Coverage       : w/ debug symbols, w/o optimization, w/ code-coverage
#  Debug          : w/ debug symbols, w/o optimization
#  Release        : w/o debug symbols, w/ optimization
#  RelWithDebInfo : w/ debug symbols, w/ optimization
#  MinSizeRel     : w/o debug symbols, w/ optimization, stripped binaries
#set(ROS_BUILD_TYPE RelWithDebInfo)

rosbuild_init()

#set the default path for built executables to the "bin" directory
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
#set the default path for built libraries to the "lib" directory
set(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)

#uncomment if you have defined messages
#rosbuild_genmsg()
#uncomment if you have defined services
#rosbuild_gensrv()

#common commands for building c++ executables and libraries
#rosbuild_add_library(${PROJECT_NAME} src/example.cpp)
#target_link_libraries(${PROJECT_NAME} another_library)
#rosbuild_add_boost_directories()
#rosbuild_link_boost(${PROJECT_NAME} thread)
#rosbuild_add_executable(example examples/example.cpp)
#target_link_libraries(example ${PROJECT_NAME})

rosbuild_add_executable(save_pcd save_pcd.cpp)
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. **Kinect点群を保存する**
5. 保存したKinect点群を描画する

save_pcd.cppを置く。
下記からダウンロードしてください。
↓
https://github.com/kanezaki/ssii2016_tutorial/blob/master/save_pcd.cpp

```
#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <opencv2/highgui/highgui.hpp>
#include <pcl_ros/io/pcd_io.h>
#include <sensor_msgs/image_encodings.h>

class SavePCD {
private:
  ros::NodeHandle _nh;
  ros::Subscriber _sub1, _sub2;
  pcl::PointCloud<pcl::PointXYZRGB> input_cloud;
  int save_count;
public:
  SavePCD() : save_count(0) {
    //* subscribe ROS topics
    _sub1 = _nh.subscribe ("/camera/rgb/image_color", 1,  &SavePCD::image_cb, this);
    ROS_INFO ("Listening for incoming data on topic /camera/rgb/image_color ..." );
    _sub2 = _nh.subscribe ("/camera/depth_registered/points", 1,  &SavePCD::points_cb, this);
    ROS_INFO ("Listening for incoming data on topic /camera/depth_registered/points ..." );
  }
  ~SavePCD() {}

  //* get points
  void points_cb( const sensor_msgs::PointCloud2ConstPtr& cloud ){
    if ((cloud->width * cloud->height) == 0)
      return;
    pcl::fromROSMsg (*cloud, input_cloud);
  }

  //* show color img and save color img + point cloud
  void image_cb( const sensor_msgs::ImageConstPtr& msg ){
    cv_bridge::CvImagePtr cv_ptr = cv_bridge::toCvCopy(msg);

    //* show color img
    cv::Mat color_img = cv_ptr->image;
    cv::imshow( "color image", color_img );
    cv::waitKey(10);

    if ((input_cloud.width * input_cloud.height) == 0)
      return;

    //* save
    std::stringstream filename1;
    filename1 << save_count << ".png";
    cv::imwrite( filename1.str(), color_img );
    std::cout << filename1.str() << " saved." << std::endl;
    std::stringstream filename2;
    filename2 << save_count << ".pcd";
    pcl::io::savePCDFileBinary( filename2.str(), input_cloud );
    std::cout << filename2.str() << " saved." << std::endl;
    save_count++;
    usleep( 300000 );
  }
};

int main( int argc, char** argv ){
  ros::init(argc,argv,"save_pcd");
  SavePCD spcd;
  ros::spin();

  return(0);
}
```

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. **Kinect点群を保存する**
5. 保存したKinect点群を描画する

```
$ make
```

./binフォルダ以下にsave_pcdという実行ファイルができる。

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. **Kinect点群を保存する**
5. 保存したKinect点群を描画する

```
$ rosrun save_pcd save_pcd
```

あるいは

```
$ ./bin/save_pcd
```

```
% rosrun save_pcd save_pcd
[ INFO] [1461229852.848525452]: Listening for incoming data on topic /camera/rgb/image_color ...
[ INFO] [1461229852.850923396]: Listening for incoming data on topic /camera/depth_registered/points ...
0.png saved.
0.pcd saved.
1.png saved.
1.pcd saved.
2.png saved.
2.pcd saved.
```

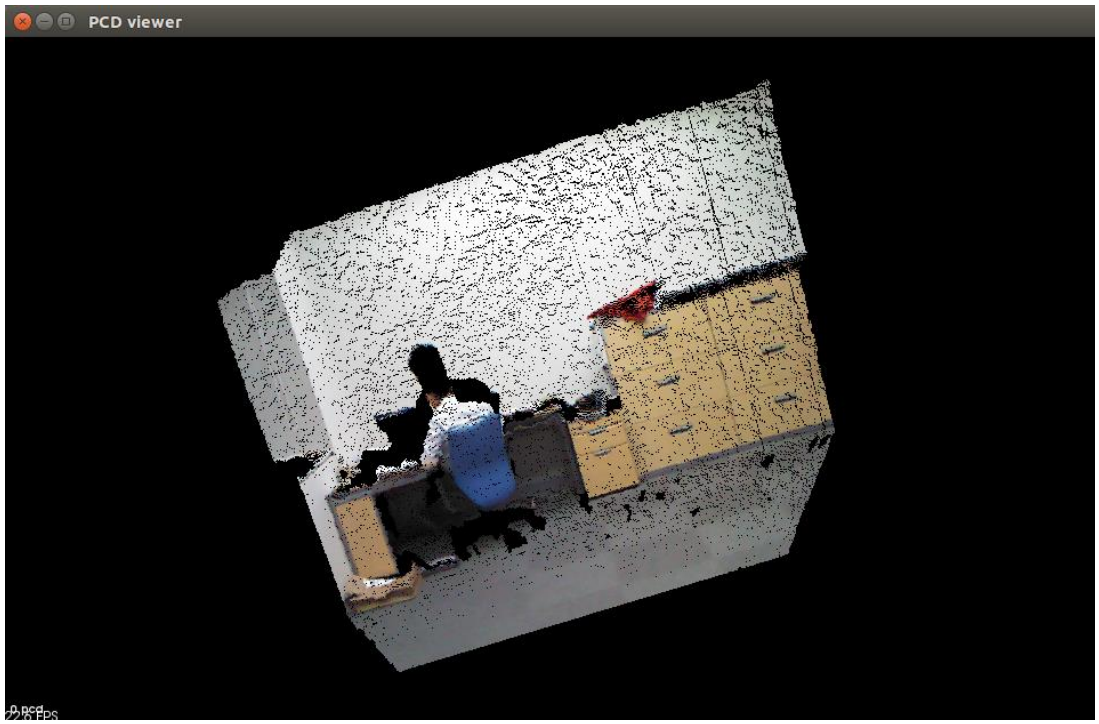
画像（.png）と点群（.pcd）がカレントディレクトリに保存され続けるので、
はやめに Ctrl+C して止めてください。

KinectからのRGBD画像・点群を扱う

手順

1. ROSをインストールする
2. Kinect画像を表示する
3. Kinect点群を描画する
4. Kinect点群を保存する
5. **保存したKinect点群を描画する**

```
$ pcl_viewer 0.pcd
```



“r”キーを押して
“5”キーを押すと
色付き点群が現れる。

詳しい使い方（ヘルプ）は
“h”キーを押す。

PCDファイルを（無理やり）PLYファイルにする

CMakeLists.txtに下記の一文を加える。

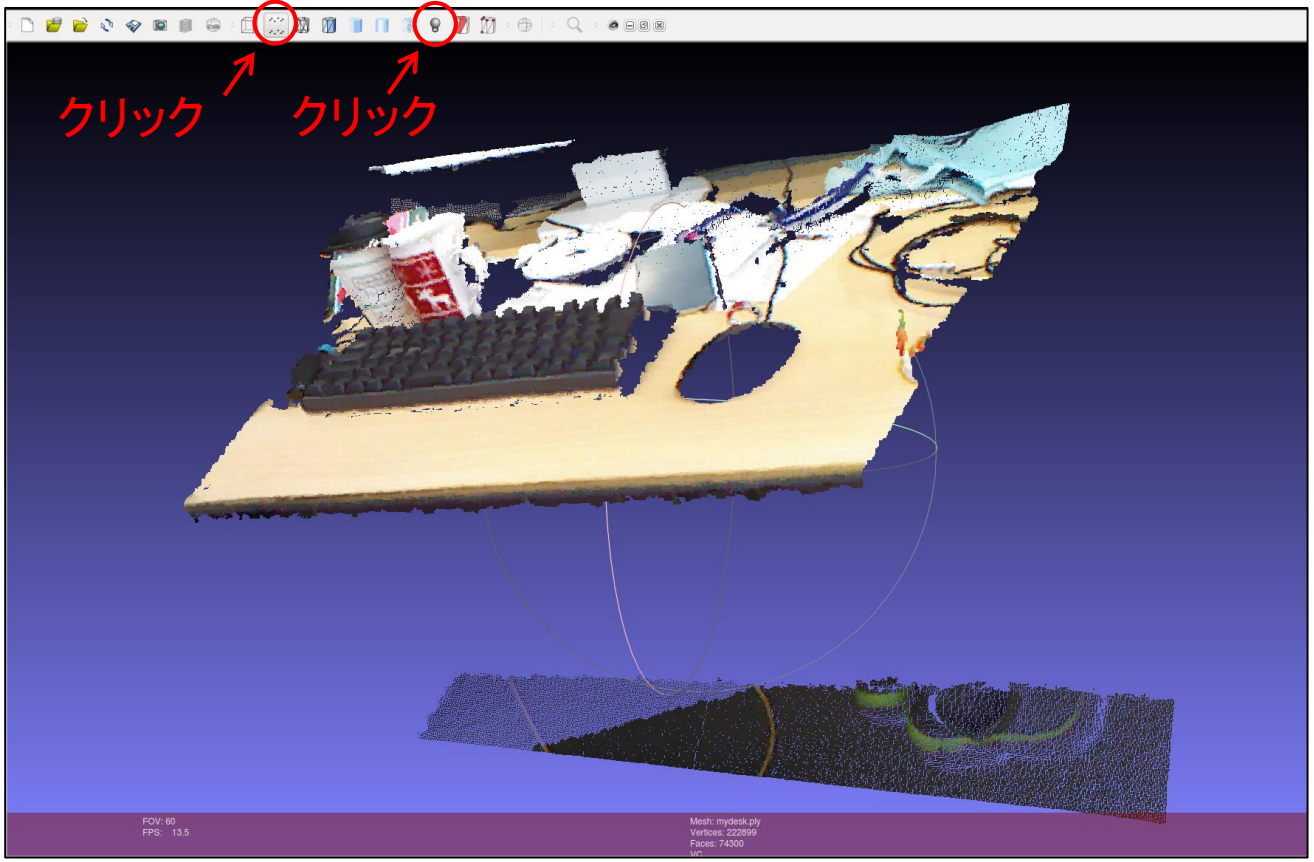
```
rosbuild_add_executable(convertpcd2ply convertpcd2ply.cpp)
```

実行→

```
$ make
$ rosrun save_pcd convertpcd2ply 0.pcd 0.ply
```

convertpcd2ply.cppを置く。
下記からダウンロードしてください。
↓
https://github.com/kanezaki/ssii2016_tutorial/blob/master/convertpcd2ply.cpp

MeshLabでインポート→点群表示→光源OFF



真のチュートリアル

2. Point Cloud Library (PCL)の使い方

UbuntuでPCLのインストール

- 既に先のチュートリアル内容でROSのDesktop-Fullをインストールした人はもう入っています。
  （/usr/bin/以下にpcl_viewer等の実行ファイルが既にあるはず。）

- ROSを使わず、PCLだけインストールして使いたい人はこちら。

```
$ sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
$ sudo apt-get update
$ sudo apt-get install libpcl-all
```

- PCLをソースからコンパイルして使いたい人はこちら。

```
$ git clone https://github.com/PointCloudLibrary/pcl
$ cd pcl; mkdir build; cd build
$ cmake ..
$ make
```

参考

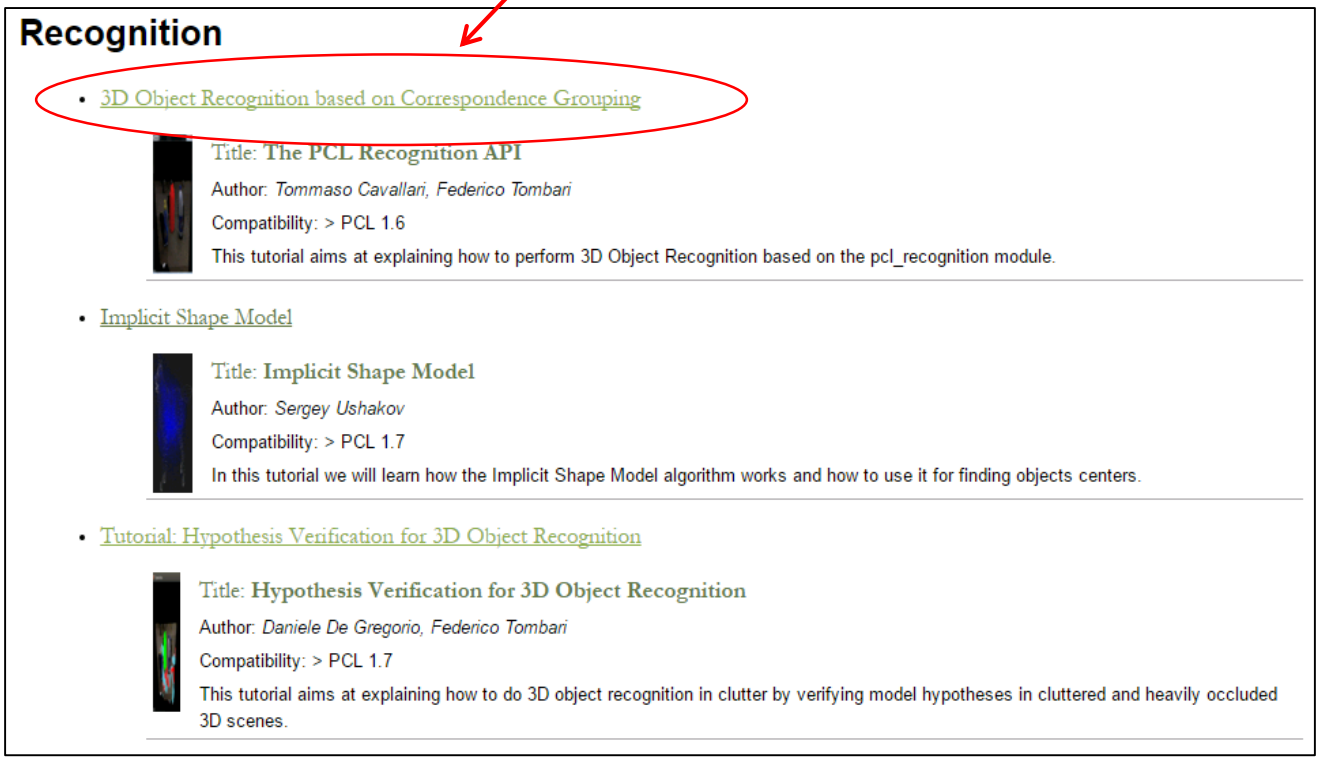
http://pointclouds.org/downloads/linux.html

PCLの公式サイトのチュートリアル

http://pointclouds.org/documentation/tutorials/

- Basic Usage
- Advanced Usage
- Applications
- Features
- Filtering
- I/O
- Keypoints
- KdTree
- Octree
- Range Images
- Recognition
- Registration
- Sample Consensus
- Segmentation
- Surface
- Visualization
- GPU

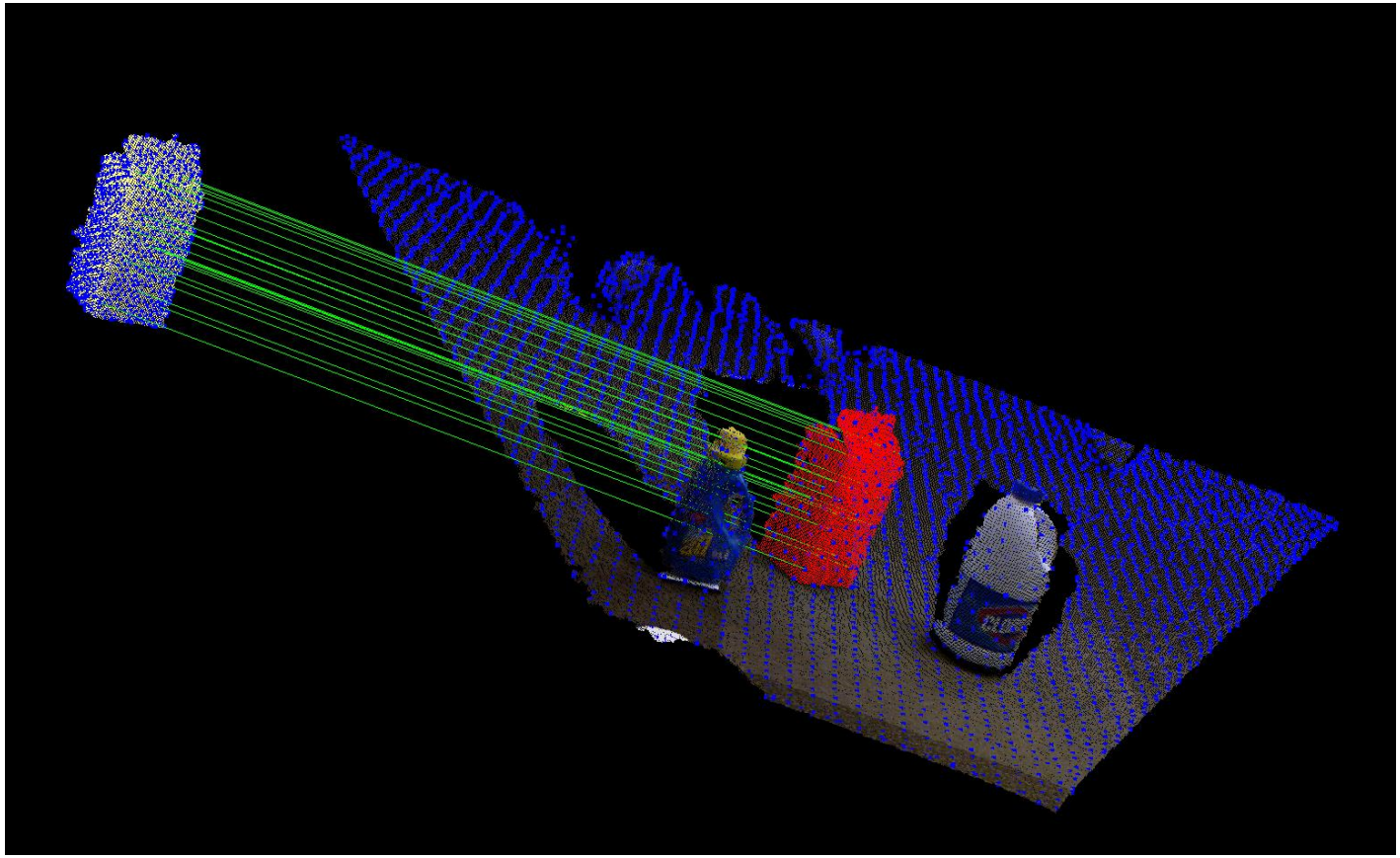
今回はこれをやってみます。



PCLの公式サイトのチュートリアル

http://pointclouds.org/documentation/tutorials/correspondence_grouping.php#correspondence-grouping

3D Object Recognition based on Correspondence Grouping



シーンから所望の物体を、対応点探索により発見する。

3D Object Recognition based on Correspondence Grouping

手順

1. **自分のROSパッケージを作る**
2. 必要なファイルをダウンロードする
3. ちょっとコードを修正してコンパイル
4. 実行

correspondence_groupingという名のパッケージを作る。

```
$ cd ~/ros
$ roscreate-pkg correspondence_grouping pcl_ros
$ cd correspondence_grouping
```

CMakeLists.txtに下記の文を加える。

```
find_package(PCL 1.7 REQUIRED)
rosbuild_add_executable(correspondence_grouping correspondence_grouping.cpp)
target_link_libraries(correspondence_grouping ${PCL_LIBRARIES})
```

注） 別にROSを使わなくてもOKです。
その場合はPCL公式チュートリアルページを参考にしてください。

3D Object Recognition based on Correspondence Grouping

手順

1. 自分のROSパッケージを作る
2. **必要なファイルをダウンロードする**
3. ちょっとコードを修正してコンパイル
4. 実行

点群ファイル

- milk.pcd
  https://github.com/PointCloudLibrary/pcl/blob/master/test/milk.pcd?raw=true
- milk_cartoon_all_small_clorox.pcd
  https://github.com/PointCloudLibrary/pcl/blob/master/test/milk_cartoon_all_small_clorox.pcd?raw=true

ソースコード

- correspondence_grouping.cpp
  http://pointclouds.org/documentation/tutorials/_downloads/correspondence_grouping.cpp

3D Object Recognition based on Correspondence Grouping

手順

1. 自分のROSパッケージを作る
2. 必要なファイルをダウンロードする
3. **ちょっとコードを修正してコンパイル**
4. 実行

```
 #include <pcl/features/board.h>
-#include <pcl/filters/uniform_sampling.h>
+#include <pcl/filters/voxel_grid.h>
 #include <pcl/recognition/cg/hough_3d.h>
 #include <pcl/recognition/cg/geometric_consistency.h>

-  pcl::UniformSampling<PointType> uniform_sampling;
+  pcl::VoxelGrid<PointType> uniform_sampling;
   uniform_sampling.setInputCloud (model);
-  uniform_sampling.setRadiusSearch (model_ss_);
+  uniform_sampling.setLeafSize (model_ss_,model_ss_,model_ss_);
   uniform_sampling.filter (*model_keypoints);
   std::cout << "Model total points: " << model->size () << "; Selected Keypoints: " << model_keypoints->size () << std::endl;

   uniform_sampling.setInputCloud (scene);
-  uniform_sampling.setRadiusSearch (scene_ss_);
+  uniform_sampling.setLeafSize (scene_ss_,scene_ss_,scene_ss_);
   uniform_sampling.filter (*scene_keypoints);
   std::cout << "Scene total points: " << scene->size () << "; Selected Keypoints: " << scene_keypoints->size () << std::endl;
```

修正済みのcorrespondence_grouping.cppはこちら→
https://github.com/kanezaki/ssii2016_tutorial/blob/master/correspondence_grouping.cpp

3D Object Recognition based on Correspondence Grouping

手順

1. 自分のROSパッケージを作る
2. 必要なファイルをダウンロードする
3. ちょっとコードを修正してコンパイル
4. **実行**

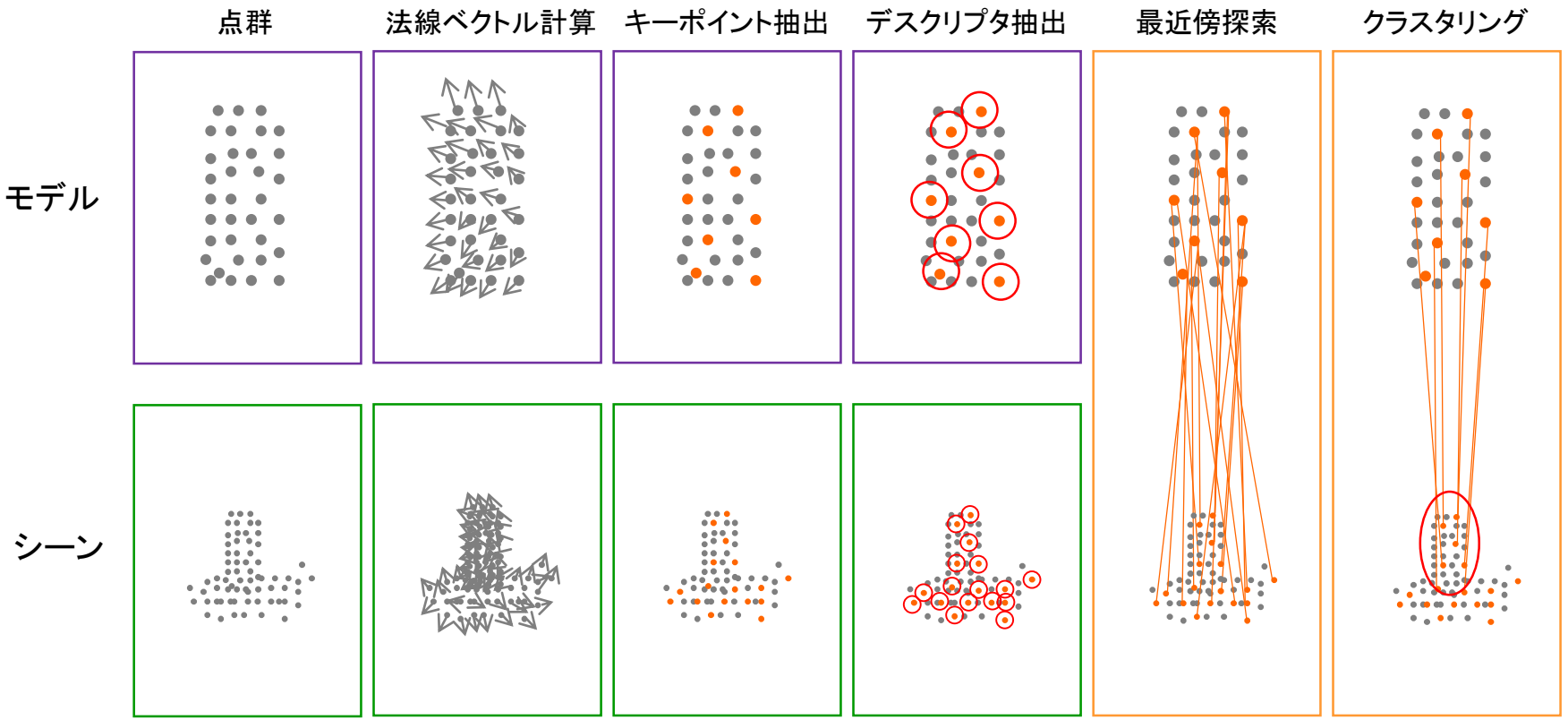
```
$ rosrun correspondence_grouping correspondence_grouping milk.pcd ¥
milk_cartoon_all_small_clorox.pcd -k -c
```

あるいは

```
$ ./bin/correspondence_grouping milk.pcd milk_cartoon_all_small_clorox.pcd -k -c
```

3D Object Recognition based on Correspondence Grouping

処理内容の概要



サンプルプログラムの実装

Uniform Sampling

SHOT

FLANN

Hough 3D
or
Geometric Consistency (GC)

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

162行目～

```
pcl::PointCloud<PointType>::Ptr model (new pcl::PointCloud<PointType> ());
pcl::PointCloud<PointType>::Ptr model_keypoints (new pcl::PointCloud<PointType> ());
pcl::PointCloud<PointType>::Ptr scene (new pcl::PointCloud<PointType> ());
pcl::PointCloud<PointType>::Ptr scene_keypoints (new pcl::PointCloud<PointType> ());
pcl::PointCloud<NormalType>::Ptr model_normals (new pcl::PointCloud<NormalType> ());
pcl::PointCloud<NormalType>::Ptr scene_normals (new pcl::PointCloud<NormalType> ());
pcl::PointCloud<DescriptorType>::Ptr model_descriptors (new pcl::PointCloud<DescriptorType> ());
pcl::PointCloud<DescriptorType>::Ptr scene_descriptors (new pcl::PointCloud<DescriptorType> ());
```

モデル（検出対象物体）とシーン（環境）の点群いろいろ。
色付き点群全体、キーポイント点群、法線ベクトル、デスクリプタの
すべてのデータを点群形式で持っている。

点群タイプはpcl::PointXYZRGBA等、さまざまある。
このソースコード内での定義は16行目～（下記）

```
typedef pcl::PointXYZRGBA PointType;
typedef pcl::Normal NormalType;
typedef pcl::ReferenceFrame RFType;
typedef pcl::SHOT352 DescriptorType;
```

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

213行目～

```
//
//  Compute Normals
//
pcl::NormalEstimationOMP<PointType, NormalType> norm_est;
norm_est.setKSearch (10);
norm_est.setInputCloud (model);
norm_est.compute (*model_normals);

norm_est.setInputCloud (scene);
norm_est.compute (*scene_normals);
```

法線ベクトルの計算。
なにか三次元点群処理しようと思ったら大抵はこれが必要。
今回はデスクリプタの計算のために必要。

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

225行目～

```
  //
  //  Downsample Clouds to Extract keypoints
  //
  pcl::VoxelGrid<PointType> uniform_sampling;
  uniform_sampling.setInputCloud (model);
  uniform_sampling.setLeafSize (model_ss_,model_ss_,model_ss_);
  uniform_sampling.filter (*model_keypoints);
  std::cout << "Model total points: " << model->size () << "; Selected Keypoints: " << model_keypoints-
>size () << std::endl;

  uniform_sampling.setInputCloud (scene);
  uniform_sampling.setLeafSize (scene_ss_,scene_ss_,scene_ss_);
  uniform_sampling.filter (*scene_keypoints);
  std::cout << "Scene total points: " << scene->size () << "; Selected Keypoints: " << scene_keypoints-
>size () << std::endl;
```

キーポイントの抽出だが、
今回は単純に等間隔にダウンサンプリングした点群をキーポイント点群とする。
（もっとちゃんとキーポイント抽出するなら、たとえばISS等がPCLに実装されている。）

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

240行目～

```
//
//  Compute Descriptor for keypoints
//
pcl::SHOTEstimationOMP<PointType, NormalType, DescriptorType> descr_est;
descr_est.setRadiusSearch (descr_rad_);

descr_est.setInputCloud (model_keypoints);
descr_est.setInputNormals (model_normals);
descr_est.setSearchSurface (model);
descr_est.compute (*model_descriptors);

descr_est.setInputCloud (scene_keypoints);
descr_est.setInputNormals (scene_normals);
descr_est.setSearchSurface (scene);
descr_est.compute (*scene_descriptors);
```

各キーポイントまわりのデスクリプタの抽出。
今回はSHOT記述子を抽出する。

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

256行目～

```
  //
  //  Find Model-Scene Correspondences with KdTree
  //
  pcl::CorrespondencesPtr model_scene_corrs (new pcl::Correspondences ());

  pcl::KdTreeFLANN<DescriptorType> match_search;
  match_search.setInputCloud (model_descriptors);

  //  For each scene keypoint descriptor, find nearest neighbor into the model keypoints descriptor cloud
and add it to the correspondences vector.
  for (size_t i = 0; i < scene_descriptors->size (); ++i)
  {
    std::vector<int> neigh_indices (1);
    std::vector<float> neigh_sqr_dists (1);
    ….
    (省略)
```

FLANNによる最近傍探索により、対応点集合を得る。

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

282行目～

```
 //
 //  Actual Clustering
 //
 std::vector<Eigen::Matrix4f, Eigen::aligned_allocator<Eigen::Matrix4f> > rototranslations;
 std::vector<pcl::Correspondences> clustered_corrs;

 //  Using Hough3D
 if (use_hough_)
 {
   //
   //  Compute (Keypoints) Reference Frames only for Hough
   //
…
   (省略)
```

クラスタリングにより正解の物体の点集合を得る。（2種類の手法が選べる。）

Hough: F. Tombari and L. Di Stefano: “Object recognition in 3D scenes with occlusions and clutter by Hough voting”, 4th Pacific-Rim Symposium on Image and Video Technology, 2010.

GC: H. Chen and B. Bhanu: “3D free-form object recognition in range images using local surface patches”, Pattern Recognition Letters, vol. 28, no. 10, pp. 1252-1262, 2007.

3D Object Recognition based on Correspondence Grouping

ソースコードの説明（コアなところだけ。）

最後にPCLVisualizerを使って結果を描画する。

```
(省略)
```

余談：点群を表示させたいだけならCloudViewerクラスで数行で書ける。

The CloudViewer
http://pointclouds.org/documentation/tutorials/cloud_viewer.php

真のチュートリアル

3.その他のオープンソースライブラリの紹介

本日紹介するオープンソース

1. LSD-SLAM

   http://vision.in.tum.de/research/vslam/lsdslam
   https://github.com/tum-vision/lsd_slam
   J. Engel, T. Schöps, D. Cremers
   LSD-SLAM: Large-Scale Direct Monocular SLAM
   European Conference on Computer Vision (ECCV), 2014

   RGB-Dではないが、単眼カメラでvisual SLAMをするコード
   ROSで動く。

2. ProjectInSeg

   http://campar.in.tum.de/Chair/ProjectInSeg
   K. Tateno , F. Tombari, N. Navab
   When 2.5D is not enough: Simultaneous Reconstruction, Segmentation and Recognition on dense SLAM
   IEEE International Conference on Robotics and Automation (ICRA), 2016

   RGB-D画像を入力とし、dense SLAMをしながらセグメンテーションと認識を行う。
   ※公開されてるコードではセグメンテーションまで

(1) LSD-SLAM

J. Engel, T. Schöps, D. Cremers
LSD-SLAM: Large-Scale Direct Monocular SLAM
European Conference on Computer Vision (ECCV), 2014
https://github.com/tum-vision/lsd_slam

1. ソースのダウンロード（クローン）

```
$ cd ~/ros
$ git clone https://github.com/tum-vision/lsd_slam.git lsd_slam
```

2. コンパイル

```
$ rosmake lsd_slam
```

※rosmakeは複数のパッケージをまとめたスタックを、パッケージ依存関係を見ながらmakeします

3. 以下、4つのターミナルウィンドウ（タブ）を立ち上げて各々実行。

```
$ roscore
```

```
$ rosrun lsd_slam_viewer viewer
```

```
$ rosbag play ~/LSD_room.bag
```

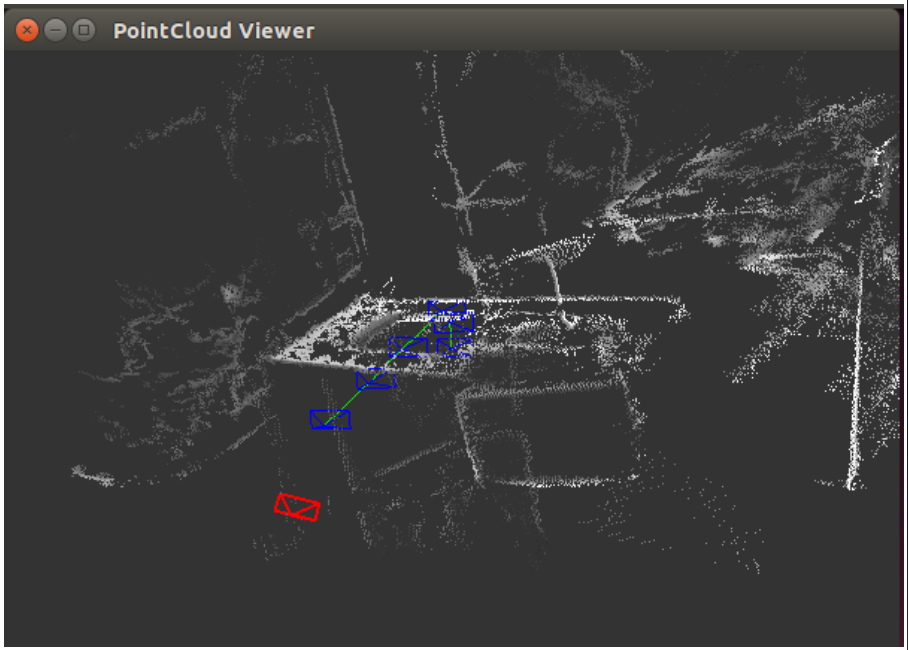
```
$ rosrun lsd_slam_core live_slam image:=/image_raw camera_info:=/camera_info
```

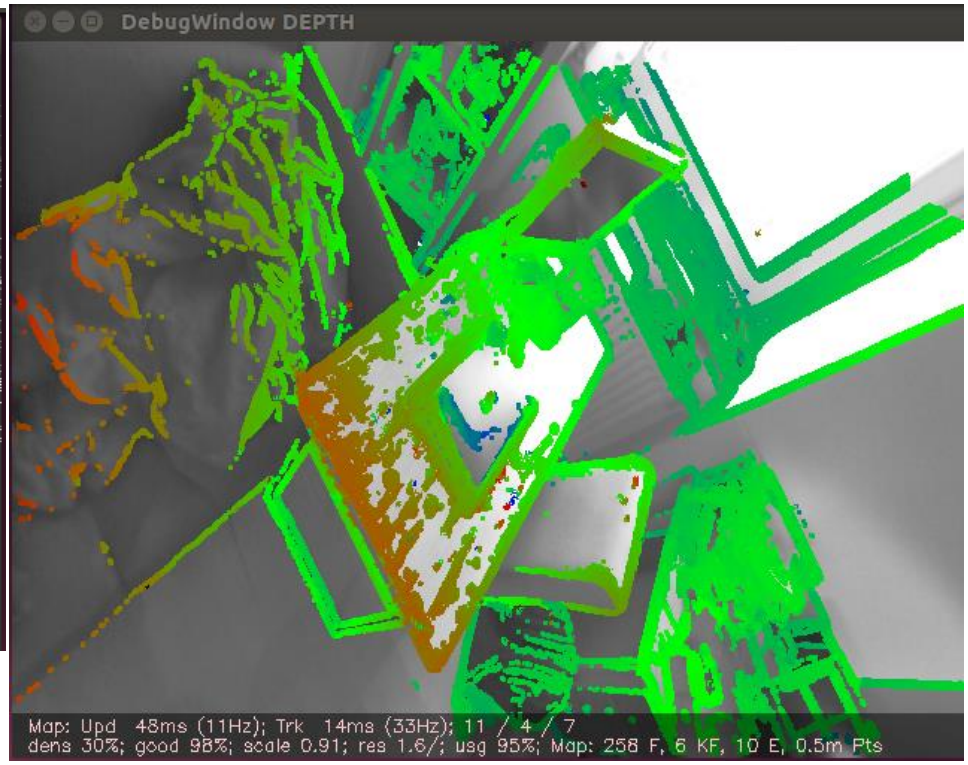
データセット。ダウンロードはこちら http://vmcremers8.informatik.tu-muenchen.de/lsd/LSD_room.bag.zip

(1) LSD-SLAM

J. Engel, T. Schöps, D. Cremers
LSD-SLAM: Large-Scale Direct Monocular SLAM
European Conference on Computer Vision (ECCV), 2014
https://github.com/tum-vision/lsd_slam

こんな感じになります





(1) LSD-SLAM

J. Engel, T. Schöps, D. Cremers
LSD-SLAM: Large-Scale Direct Monocular SLAM
European Conference on Computer Vision (ECCV), 2014
https://github.com/tum-vision/lsd_slam

Kinectからも可能（もちろん普通のウェブカメラからも）。

以下、3つのターミナルウィンドウ（タブ）を立ち上げて各々実行。

```
$ source /opt/ros/indigo/setup.sh; roslaunch freenect_launch freenect.launch
```

```
$ rosrun lsd_slam_viewer viewer
```

```
$ rosrun lsd_slam_core live_slam image:=/camera/rgb/image_color camera_info:=/camera/rgb/camera_info
```

(2) ProjectInSeg

K. Tateno , F. Tombari, N. Navab
When 2.5D is not enough: Simultaneous Reconstruction, Segmentation and Recognition on dense SLAM
IEEE International Conference on Robotics and Automation (ICRA), 2016
http://campar.in.tum.de/Chair/ProjectInSeg

1. ソースのダウンロード

```
$ wget http://campar.in.tum.de/personal/tateno/IROS2015/InSeg.zip
$ unzip InSeg.zip
```

2. コンパイル

```
$ mkdir InSeg/buildLinux/
$ cd InSeg/buildLinux/
$ cmake ..
$ make
```

3. 実行

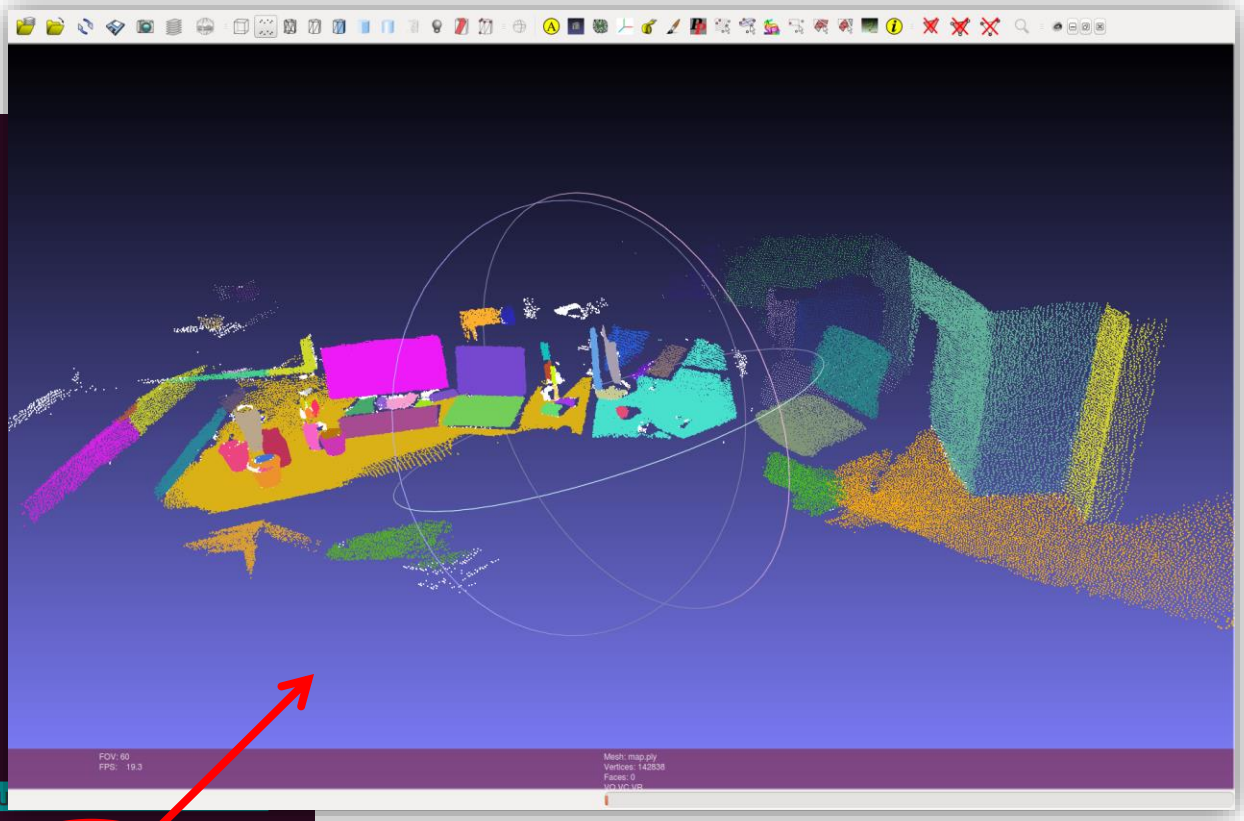
```
$ cd InSegTest
$ ../../bin/InSegTest
```

(2) ProjectInSeg

K. Tateno , F. Tombari, N. Navab
When 2.5D is not enough: Simultaneous Reconstruction, Segmentation and Recognition on dense SLAM
IEEE International Conference on Robotics and Automation (ICRA), 2016
http://campar.in.tum.de/Chair/ProjectInSeg

こんな感じになります

```
 0.80889  0.519232 -0.275853  -1613.12
       0         0         0         1
FrameIndex 356:
 -0.49934  0.300009  -0.81281  -282.534
-0.335199  0.798189  0.500543   788.827
 0.798942  0.522386 -0.298001  -1669.02
       0         0         0         1
FrameIndex 357:
-0.512205   0.30198 -0.804027   -260.41
-0.326624  0.797324  0.507542   807.146
 0.794335  0.522573 -0.309755  -1713.36
       0         0         0         1
FrameIndex 358:
-0.520242  0.300435 -0.799433   -246.26
-0.329472  0.793012  0.512434   836.416
  0.78791  0.529974 -0.313569   -1753.4
       0         0         0         1
FrameIndex 359:
-0.531494  0.291246 -0.795422  -221.807
-0.336121  0.789427  0.513649   861.878
 0.777523  0.540352 -0.321678  -1776.39
       0         0         0         1
FrameIndex 360:
[kanezaki@kanezaki-CFSZ5-2] ~/lib/InSeg/bu
% ls
CMakeFiles/  Makefile  cmake_install.cmake  map.ply
```



その他のオープンソース情報

Kinect Fusion

https://msdn.microsoft.com/en-us/library/dn188670.aspx
3Dスキャン（三次元再構成）するKinect for Windowsの公式アプリ。

Ubuntu+PCLでの動作状況（2016年5月 金崎調べ）
　OpenNIサポート終了の影響か、Kinect v1では動かなかったが、
　Xtionでは動いた。

CUDAをインストール後、下記のとおりPCLをソースからコンパイルして使用。

コンパイル

```
$ git clone https://github.com/PointCloudLibrary/pcl
$ cd pcl; mkdir build; cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. -DWITH_CUDA=ON -DBUILD_GPU=ON
$ make
```

実行

```
$ ./bin/pcl_kinfu_app
```

トラッキングコンペティション2016

http://sigmr.vrsj.org/tc2016/

- SLAMの精度をオンサイトで競う大会
  - 9月14-16日のVR学会年次大会@筑波で開催
  - 8月31日登録締切(アルゴリズム公開不要)

1. 開始地点で競技用座標系を獲得
2. SLAMを用いて自己位置推定しながら移動
3. 与えられた座標にマーキング

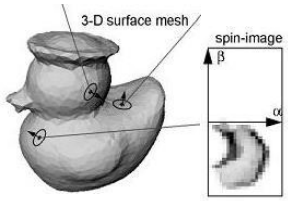
真のチュートリアル

おしまい

3D物体認識の最新動向
ーディープラーニングと大規模データセットー

3D物体認識の最新動向

1999



Spin Imageによる
3D物体認識

2010.11

Kinect登場

2012.10

IMAGENET

大規模画像認識で
Deep Learning登場

RGBDの流行

Deepの流行

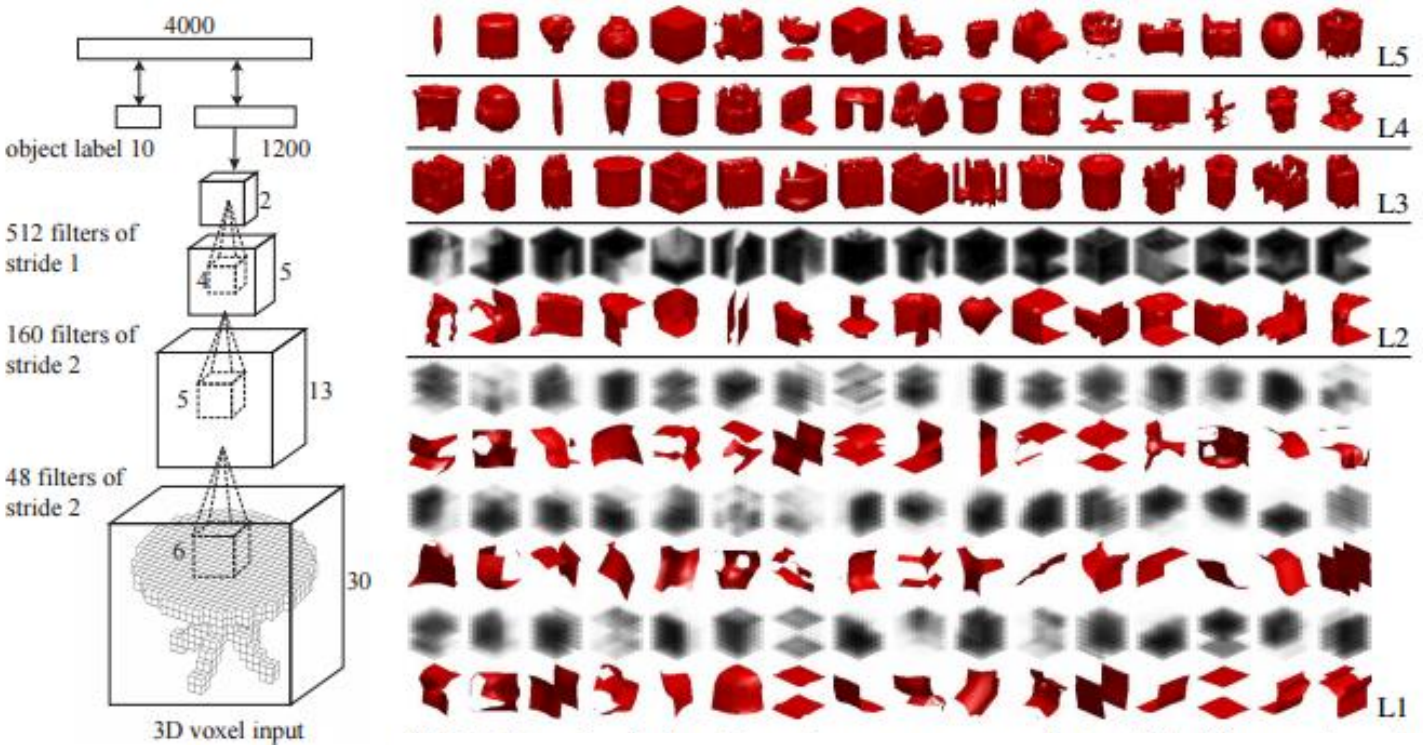
2015.6

大規模RGBDデータセット＋
3D物体のDeep Learning
@CVPR2015

3D物体認識の最新動向

3D ShapeNets: A Deep Representation for Volumetric Shapes

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. In CVPR 2015.



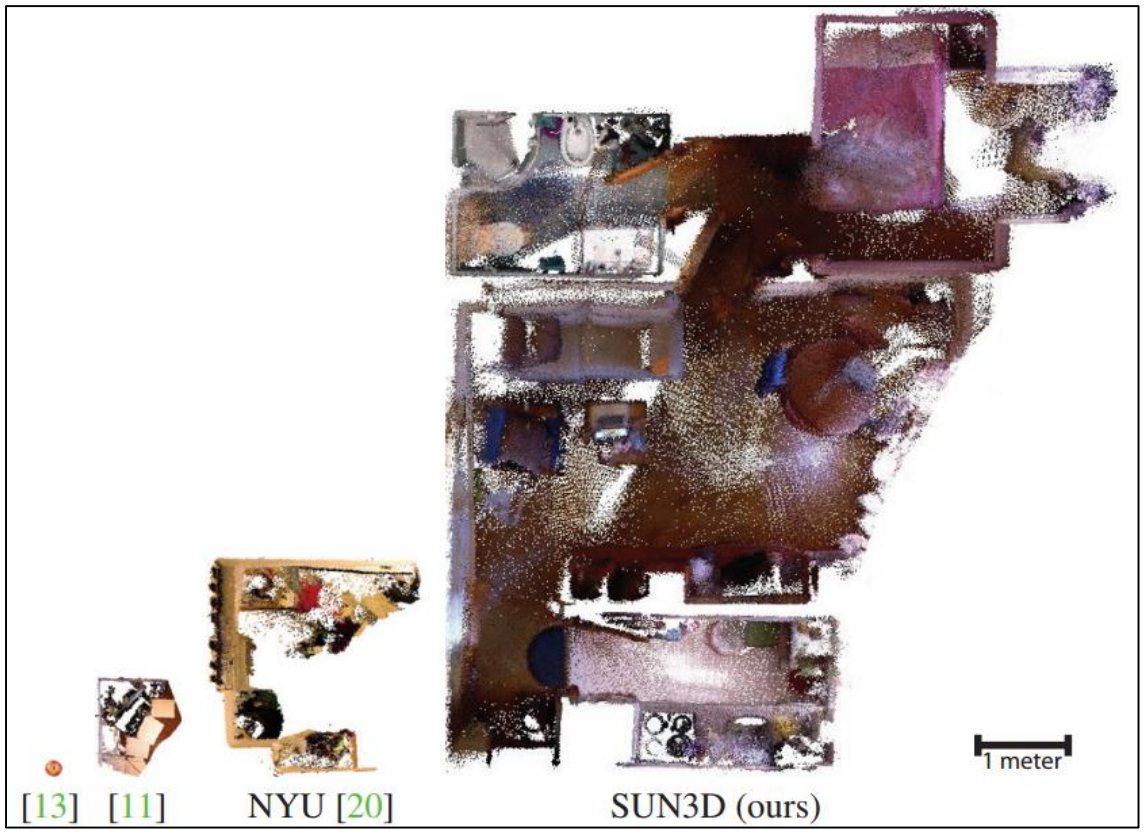
- 151,128 3D CAD models belonging to 660 unique object categories を
- 30 x 30 x 30のボクセルデータに変換して、Deep Learningで学習。
- Light Field descriptor [Chen et al. 2003], Spherical Harmonic descriptor [Kazhdan et al. 2003] と比較して高性能。

3D物体認識の最新動向

- トレンド＝大規模化

  ex.) **SUN 3D** [J. Xiao et al. 2013], **SUN RGB-D** [S. Song et al. 2015]

  ディープラーニングを行うには大規模データセットが必要。



部屋を撮影してから物体のラベルをつける流れなので、物体のカテゴリレベルで整理されていない

Figure 3. **Space coverage.** Top view of the reconstruction results to illustrate the typical 3D space covered by four different datasets at the same physical scale (meters). (a) RGB-D Object dataset [13]: multiple views of a small object; (b) Berkeley 3-D Object [11]: one view of a scene; (c) NYU Depth [20]: one corner of a room; (d) Ours: a full apartment including bedroom, living room, kitchen, and bathroom.

J. Xiao, A. Owens, and A. Torralba. SUN3D: A Database of Big Spaces Reconstructed using SfM and Object Labels. ICCV 2013.

3D物体認識の最新動向

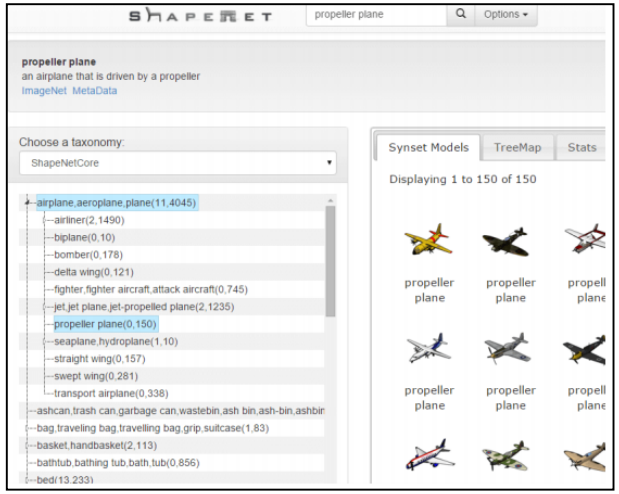
- トレンド＝大規模化

  ex.) **SUN 3D** [J. Xiao et al. 2013], **SUN RGB-D** [S. Song et al. 2015]

  ディープラーニングを行うには大規模データセットが必要。

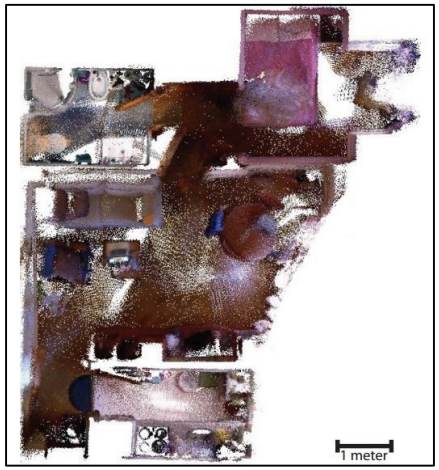
CADモデル等の人工データ

RGBDセンサで撮影した実データ



Shapenet [Chang+, 2015]

http://shapenet.cs.stanford.edu/



SUN3D [Xiao+, 2013]

Large Dataset of Object Scans [Choi+, 2015]

recruited 70 operators

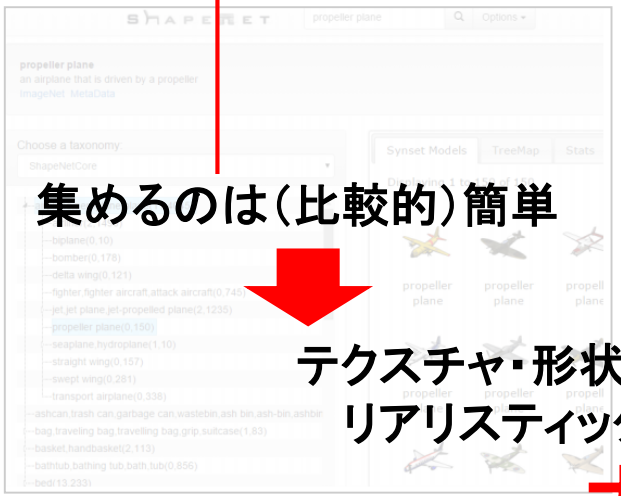
3D物体認識の最新動向

- トレンド＝大規模化

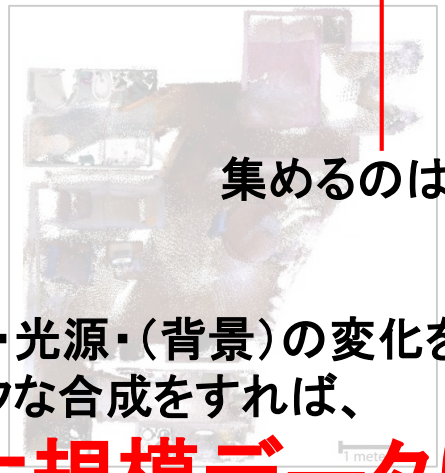
  ex.) **SUN 3D** [J. Xiao et al. 2013], **SUN RGB-D** [S. Song et al. 2015]

  ディープラーニングを行うには大規模データセットが必要。

CADモデル等の人工データ　vs.　RGBDセンサで撮影した実データ



集めるのは（比較的）簡単



集めるのは大変

テクスチャ・形状・光源・（背景）の変化を加え、
リアリスティックな合成をすれば、

大規模データは作れる

Shapenet [Chang+, 2015]
http://shapenet.cs.stanford.edu/

SUN3D [Xiao+, 2013]

Large Dataset of Object Scans
[Choi+, 2015]

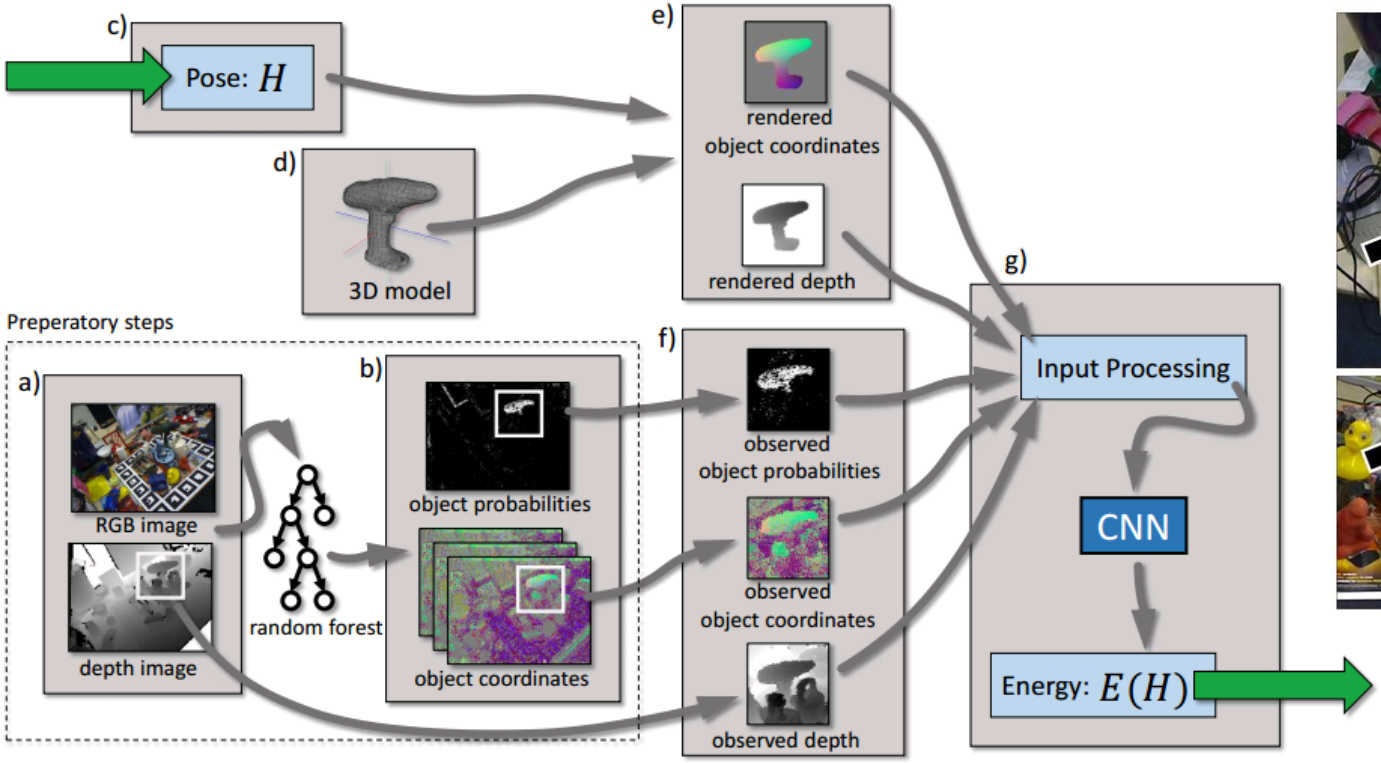
recruited

大規模データは正義

3D物体認識の最新動向

Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images

A. Krull, E. Brachmann, F. Michel, M. Y. Yang, S. Gumhold, and C. Rother, in ICCV, 2015.



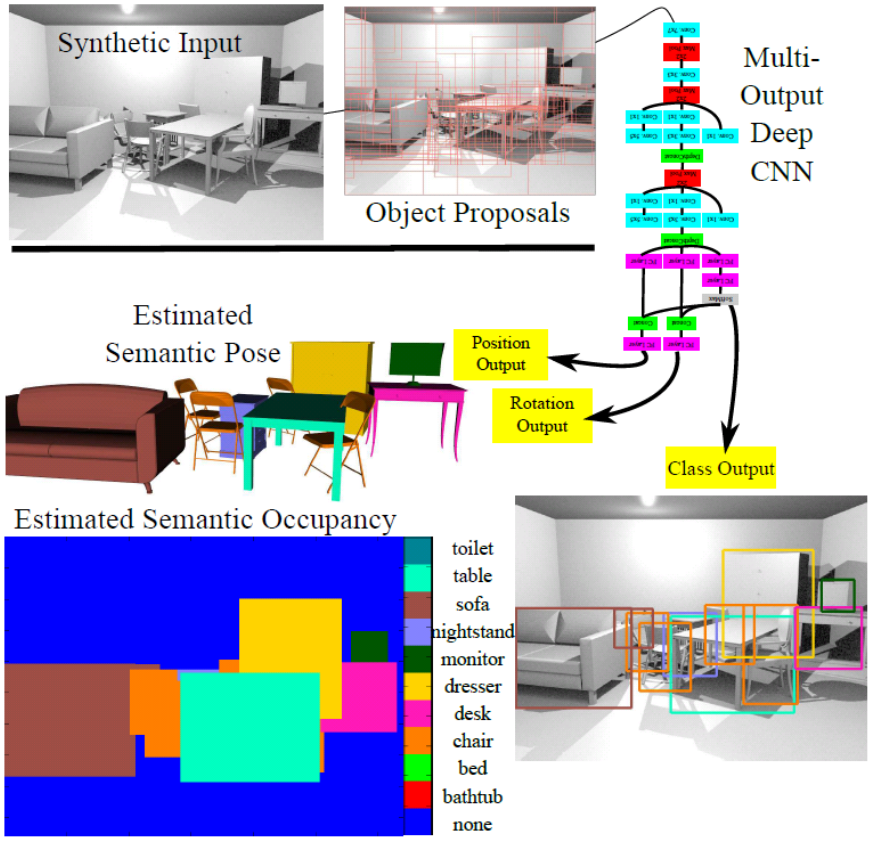
- 3Dモデルを姿勢Hでレンダリングして、実際の観測との誤差を計算
- その誤差を入力としエネルギー関数$E(H)$を出力するCNNを学習
- 学習したCNNを使って姿勢の事後分布$p(H|\boldsymbol{x};\boldsymbol{\theta})$を計算
- オクルージョンの激しい環境下でも物体の姿勢推定が高精度に可能

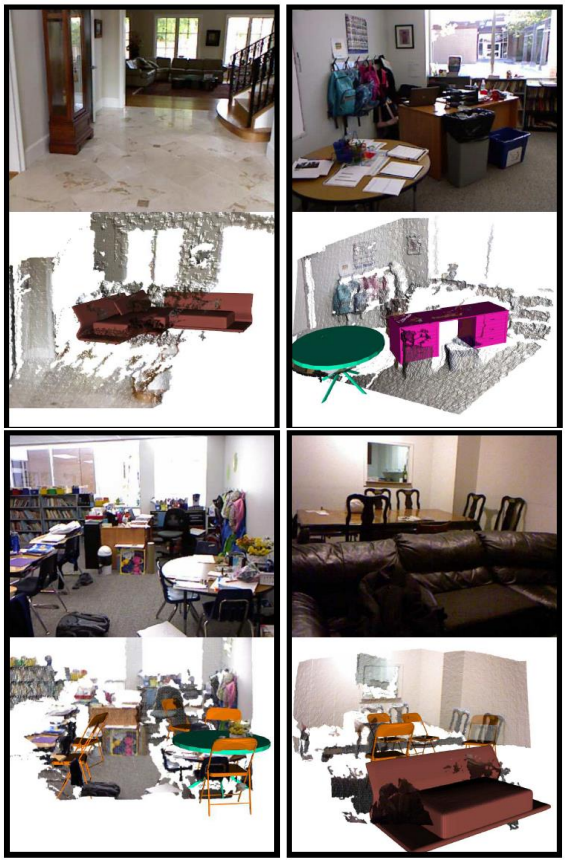
$$p(H|\mathbf{x};\boldsymbol{\theta}) = \frac{\exp\big(-E(H,\mathbf{x};\boldsymbol{\theta})\big)}{\int \exp\big(-E(\hat{H},\mathbf{x};\boldsymbol{\theta})\big)d\hat{H}}$$

3D物体認識の最新動向

Semantic Pose using Deep Networks Trained on Synthetic RGB-D

Jeremie Papon and Markus Schoeler, in ICCV, 2015.



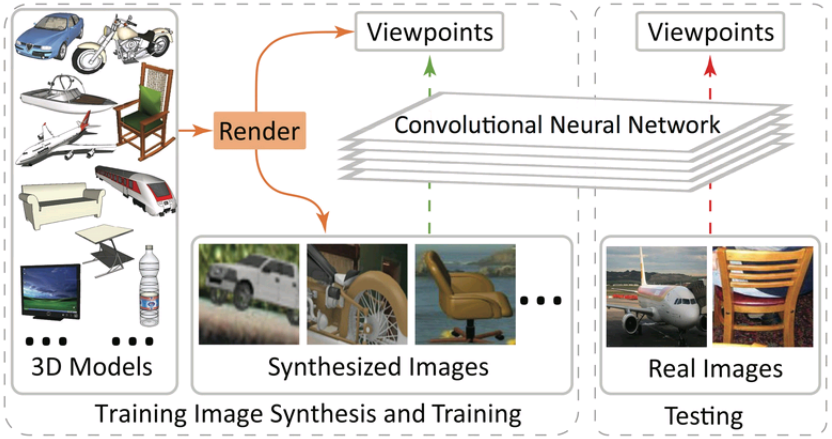


- ModelNet10の3Dモデルを使って7,000のRGBDシーンをランダムに生成
- シーン中の物体のクラスラベル、位置、姿勢を出力するDeep CNNを学習
- 多物体の識別と姿勢推定を一度にやってしまうので、1シーンの処理時間はGPUで数秒程度

3D物体認識の最新動向

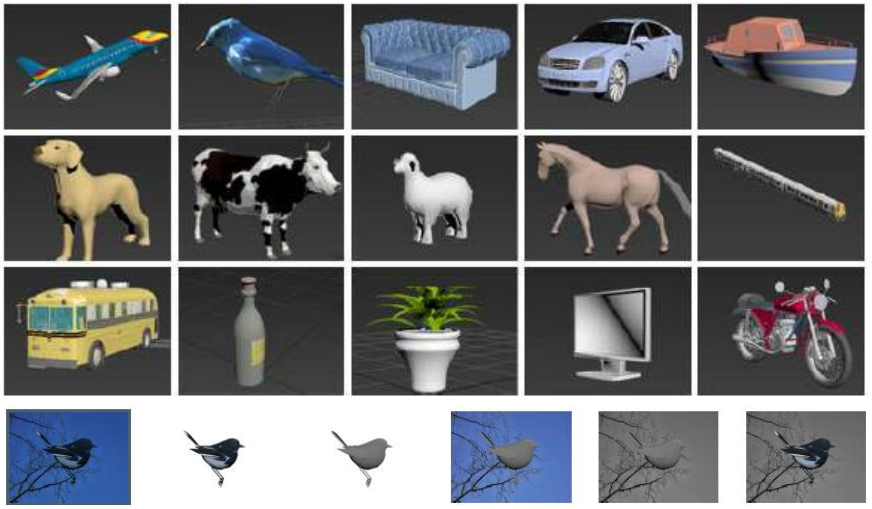
他にも…

Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views. H. Su, C. R. Qi, Y. Li, and L. J. Guibas, in ICCV, 2015.



- スタンフォード大のshapenetを使用 http://shapenet.cs.stanford.edu/
- 1度刻みのカメラ姿勢(θ, ϕ, ψ)を出力するCNNを学習
- PASCAL 3D+で物体検出＆姿勢推定精度を評価

Learning Deep Object Detectors from 3D Models. X. Peng, B. Sun, K. Ali, and K. Saenko, in ICCV, 2015.



- 3D Warehouseの3Dモデルを使用 https://3dwarehouse.sketchup.com/
- object texture, color, 3D pose, 3D shape, background scene texture and colorを合成
- synthetic varianceによってinvarianceを獲得

etc.

まとめ

- 3D特徴量の紹介
- RGBD研究の分類と紹介
- 3D点群等の扱い方（チュートリアル）
- 3D物体認識の最新動向

Take-Home Message

(1) Robotics, Computer Vision, Computer Graphics等
さまざまな分野の知識がRGBDに活かされる

(2) 大規模データは作れる（人工データ vs. 実データ）

参考文献

1. Kanezaki, Asako, and Tatsuya Harada. "3D Selective Search for obtaining object candidates." Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, 2015.
2. Newcombe, Richard A., Dieter Fox, and Steven M. Seitz. "DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.
3. Whelan, Thomas, Stefan Leutenegger, Renato F. Salas-Moreno, Ben Glocker, and Andrew J. Davison. "ElasticFusion: Dense SLAM without a pose graph." In Proceedings of Robotics: Science and Systems (RSS), 2015.
4. Ikehata, Satoshi, Hang Yang, and Yasutaka Furukawa. "Structured Indoor Modeling." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1323-1331. 2015.
5. Akgul, Ceyhun Burak, Bülent Sankur, Yücel Yemez, and Francis Schmitt. "3D model retrieval using probability density-based shape descriptors." Pattern Analysis and Machine Intelligence, IEEE Transactions on 31, no. 6 (2009): 1117-1133.
6. Kazhdan, Michael, Thomas Funkhouser, and Szymon Rusinkiewicz. "Rotation invariant spherical harmonic representation of 3 d shape descriptors." In Symposium on geometry processing, vol. 6, pp. 156-164. 2003.
7. Chen, Ding - Yun, Xiao - Pei Tian, Yu - Te Shen, and Ming Ouhyoung. "On visual similarity based 3D model retrieval." In Computer graphics forum, vol. 22, no. 3, pp. 223-232, 2003.
8. Ankerst, Mihael, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. "3D shape histograms for similarity search and classification in spatial databases." In Advances in Spatial Databases, pp. 207-226. Springer Berlin Heidelberg, 1999.
9. Johnson, Andrew E., and Martial Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes." Pattern Analysis and Machine Intelligence, IEEE Transactions on 21.5 (1999): 433-449.

参考文献

10. Rusu, Radu Bogdan, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. "Aligning point cloud views using persistent feature histograms." In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pp. 3384-3391. IEEE, 2008.
11. Rusu, Radu Bogdan, Nico Blodow, and Michael Beetz. "Fast point feature histograms (FPFH) for 3D registration." In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pp. 3212-3217. IEEE, 2009.
12. Rusu, Radu Bogdan, Gary Bradski, Romain Thibaux, and John Hsu. "Fast 3d recognition and pose using the viewpoint feature histogram." In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 2155-2162. IEEE, 2010.
13. Aldoma, Aitor, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu, and Gary Bradski. "CAD-model recognition and 6DOF pose estimation using 3D cues." In IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011.
14. Aldoma, Aitor, Federico Tombari, Radu Bogdan Rusu, and Markus Vincze. “OUR-CVFH–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation.” In Joint DAGM-OAGM Pattern Recognition Symposium, 2012.
15. Lai, Kevin, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. "Sparse distance learning for object recognition combining rgb and depth information." In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pp. 4007-4013. IEEE, 2011.
16. Hinterstoisser, Stefan, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes." In IEEE ICCV, pp. 858-865, 2011.
17. Wang, Anran, Jianfei Cai, Jiwen Lu, and Tat-Jen Cham. "MMSS: Multi-modal Sharable and Specific Feature Learning for RGB-D Object Recognition." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1125-1133. 2015.

参考文献

18. Pasqualotto, Giuliano, Pietro Zanuttigh, and Guido M. Cortelazzo. "Combining color and shape descriptors for 3D model retrieval." Signal Processing: Image Communication 28, no. 6 (2013): 608-623.
19. Tombari, Federico, Samuele Salti, and Luigi Di Stefano. "Unique signatures of histograms for local surface description." In Computer Vision–ECCV 2010, pp. 356-369, 2010.
20. Brandao, Simao, Joao P. Costeira, and Marco Veloso. "The partial view heat kernel descriptor for 3d object representation." In Robotics and Automation (ICRA), 2014 IEEE International Conference on, pp. 1054-1059. IEEE, 2014.
21. Kanezaki, Asako, Emanuele Rodola, Daniel Cremers, and Tatsuya Harada. "Learning similarities for rigid and non-rigid object detection." In 3D Vision (3DV), 2014 2nd International Conference on, vol. 1, pp. 720-727. IEEE, 2014.
22. Wu, Zhirong, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3d shapenets: A deep representation for volumetric shapes." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1912-1920. 2015.
23. Xiao, Jianxiong, Andrew Owens, and Antonio Torralba. "SUN3D: A database of big spaces reconstructed using sfm and object labels." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1625-1632. 2013.
24. Song, Shuran, Samuel P. Lichtenberg, and Jianxiong Xiao. "Sun rgb-d: A rgb-d scene understanding benchmark suite." In Proceedings of the IEEE CVPR, pp. 567-576. 2015.
25. Chang, Angel X., Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese et al. "ShapeNet: An Information-Rich 3D Model Repository." arXiv preprint arXiv:1512.03012 (2015).
26. Choi, Sungjoon, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. "A Large Dataset of Object Scans." arXiv preprint arXiv:1602.02481 (2016).

参考文献

27. Krull, Alexander, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. "Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images." In Proceedings of the IEEE International Conference on Computer Vision, pp. 954-962. 2015.
28. Papon, Jeremie, and Markus Schoeler. "Semantic Pose using Deep Networks Trained on Synthetic RGB-D." In Proceedings of the IEEE International Conference on Computer Vision, pp. 774-782. 2015.
29. Su, Hao, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views." In Proceedings of the IEEE International Conference on Computer Vision, pp. 2686-2694. 2015.
30. Peng, Xingchao, Baochen Sun, Karim Ali, and Kate Saenko. "Learning Deep Object Detectors from 3D Models." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1278-1286. 2015.