PCL Toyota Code Sprint 2.0 - Superquadrics

Alexandru - Eugen Ichim

Open Perception Inc.

Supervised by:

Dr. Joseph Djugash
Dr. Radu B. Rusu

2013

Contents

1 Introduction

1.1 Code Sprint

A code sprint is an accelerated research and development program, inspired by the Google Summer of Code model, and organised within the Point Cloud Library project by the host foundation, Open Perception Inc. PCL developers are paired with senior researchers and engineers from various institutions for extended R&D work with financial backup.

The first code sprint in which Toyota got involved was done in 2012, and one of the projects was that of implementing novel smoothing algorithms [8].

1.2 Target

The target of the current code sprint is that of analysing the possibilities of using superquadric object representation in robotics-oriented computer vision tasks such as object detection.

The structure of this report is as follows. In this section we continue by introducing the concept of superquadrics and some mathematical background, as well as sources of datasets we have used for development. Next, in Section 2, we look into techniques for sampling the superquadric equation in order to get point clouds and meshes. Section 3 shows the approaches we implemented for fitting superquadric objects into point clouds, with the extension of fully describing a point set as a collection of non-overlapping superquadrics. We then look into how to efficiently find objects for which we now the parametric representation in a new point cloud - Section 4. Finally, we show a complete demo using an RGB-D sensor in Section 5 and finalise with concluding remarks in Section 6.

1.3 Superquadrics

History

The concept of superquadrics was first introduced in the Computer Graphics community in [3], as a means of versatile parametric object modelling. The mathematical details have been extended in the work of Jaklič et al. [9]. Just like quadrics, superquadrics can be split into superellipsoids, superhyperboloids of one piece, superhyperboloids of two pieces and supertoroids, but for the purpose of our work, we only focused on superellipsoids (see Figure 1).

Mathematical Definition

They are created from two two-dimensional curves (a horizontal and a vertical curve) that form a three-dimensional surface under spherical product. For example, a unit sphere is created as the spherical product of a half circle in the plane orthogonal to (x, y) crossed with the full circle in the (x, y) plane.

$$m(\eta) = \begin{pmatrix} \cos\eta \\ \sin\eta \end{pmatrix}, \text{ with: } \eta \in [-\pi/2, \pi/2] \tag{1}$$

$$h(\omega) = \begin{pmatrix} \cos\omega \\ \sin\omega \end{pmatrix}, \text{ with: } \omega \in [-\pi, \pi) \tag{2}$$

The sphere becomes:

$$r(\eta, \omega) = m(\eta) \oplus h(\omega) = \begin{pmatrix} \cos\eta\cos\omega \\ \cos\eta\sin\omega \\ \sin\omega \end{pmatrix} \tag{3}$$

A superellipsoid can be obtained by the spherical product of two superellipses of the form:

$$\left(\frac{x}{a}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{b}\right)^{\frac{2}{\epsilon}} = 1 \tag{4}$$

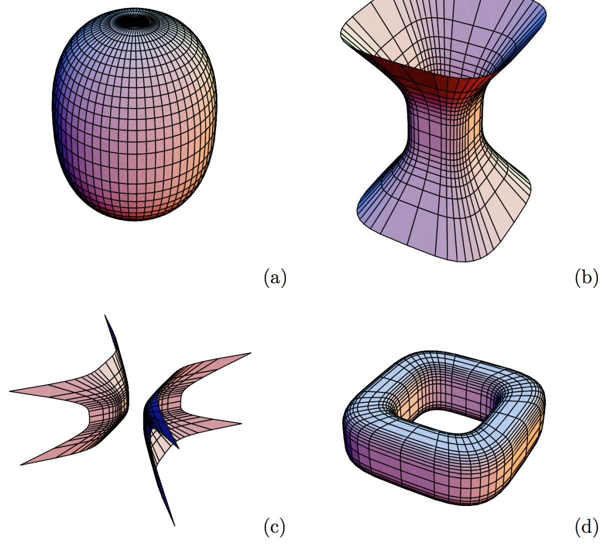


Figure 1: Family of superquadrics: (a) superellipsoid, (b) superhyperboloid of one sheet, (c) superhyperboloid of two sheets, (d) supertoroid

$$s(\theta) = \begin{pmatrix} a\cos^{\epsilon}\theta \\ b\sin^{\epsilon}\theta \end{pmatrix}, \text{ with } \theta \in [-\pi, \pi] \tag{5}$$

$$r(\eta,\omega) = s_1(\eta) \oplus s_2(\omega) = \begin{pmatrix} \cos^{\epsilon_1}\eta \\ a_3\sin^{\epsilon_1}\eta \end{pmatrix} \oplus \begin{pmatrix} a_1\cos^{\epsilon_2}\omega \\ a_2\sin^{\epsilon_2}\omega \end{pmatrix} = \begin{pmatrix} a_1\cos^{\epsilon_1}\eta\cos^{\epsilon_2}\omega \\ a_2\cos^{\epsilon_1}\eta\sin^{\epsilon_2}\omega \\ a_3\sin^{\epsilon_1}\eta \end{pmatrix}, \text{ with } \eta \in [-\pi/2, \pi/2], \omega \in [-\pi, \pi) \tag{6}$$

The five parameters of a superellipsoid are as follows:

- a_1, a_2, a_3 - scaling factors along the three axes
- ϵ_1 - the shape of the superellipsoid cross section in a plane orthogonal to the (x, y) plane and containing the z-axis
- ϵ_2 - the shape of the superellipsoid cross section parallel to the (x, y) plane

The implicit equation of a superellipsoid is thus:

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\epsilon_1}} = 1 \tag{7}$$

Because of the inside-outside functions that describe superquadrics, they can be easily manipulated by means of solid boolean operations such as union, intersection and subtraction.

Numerical Issues

There are some issues we need to take into consideration when writing code that involves the usage of superquadrics. All the powers in the implicit function are of the form $2p$: $2/\epsilon_1$, $2/\epsilon_2$, and we compute them

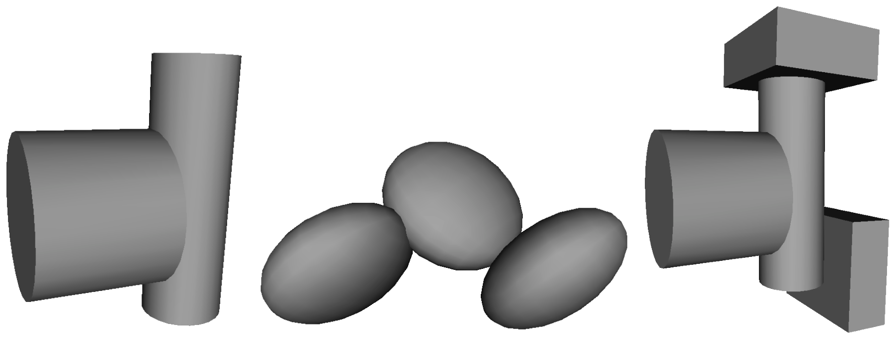


Figure 2: Example datasets created with Google Sketchup.

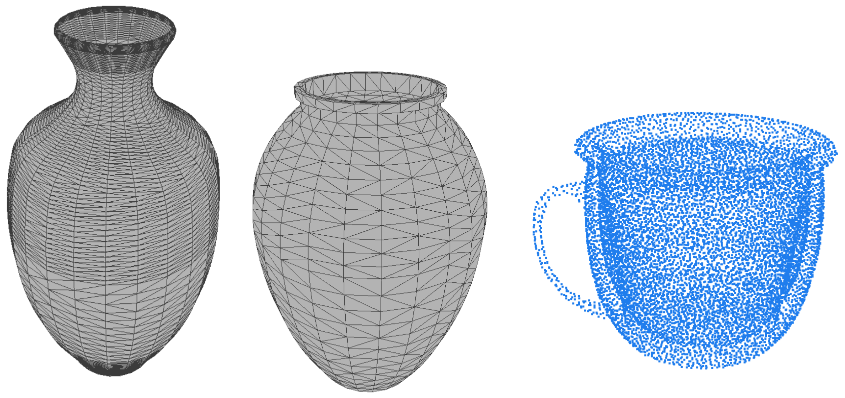


Figure 3: Example datasets downloaded from the Trimble 3D Warehouse.

in the order $x^{(2p)} = (x^2)^p$ so that they are always positive. In the case of the explicit function, all the powers are actually signed power functions, and are treated as follows:

$$x^p = sgn(x)|x|^p = \begin{cases} |x|^p & \text{if } x \geq 0 \\ -|x|^p & \text{if } x < 0 \end{cases} \tag{8}$$

Because of the limitation of the range of real numbers a 64-bit system can compute within, we need to clamp the ϵ_1 and ϵ_2 parameters in the interval $(0.1, 1.9)$. Values beyond this range do not introduce a lot more expressiveness to the objects, but create numerical problems as $\frac{\epsilon_2}{\epsilon_1}$ becomes very large.

1.4 Datasets

For the purpose of this work, we gathered data from multiple sources:

- **Created manually** - using Google Sketchup - Figure 2
- **Trimble 3D Warehouse** - `http://sketchup.google.com/3dwarehouse/` - Figure 3
- **Scanned with an RGB-D camera** - multiple frames registered with PCL KinFu - Figure 4

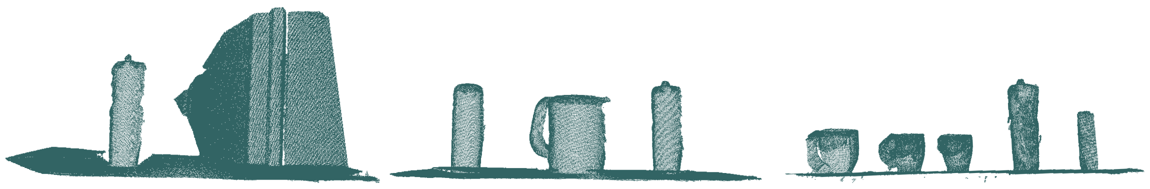


Figure 4: Example datasets created using PCL KinFu.

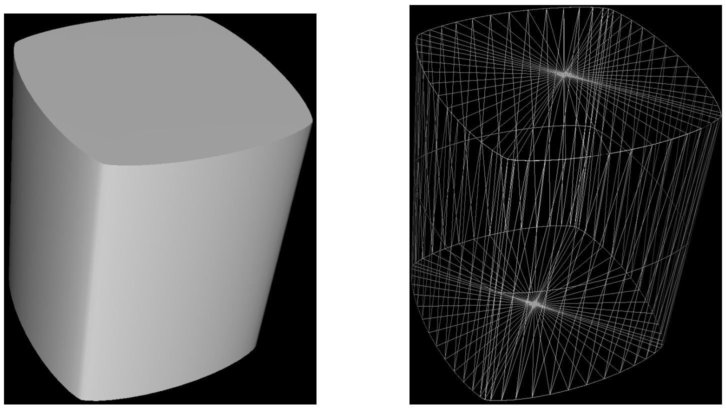


Figure 5: Superquadric sampling using the VTK library.

2 Sampling

Sampling superquadrics is important as it allows for easy visualization and for evaluating fitting or registration errors. We have tried multiple ways of doing this, as will be explained in this section.

2.1 VTK

Using the Visualization ToolKit library, the code for generating a superquadric centered at the origin is simply:

```
vtkSmartPointer<vtkSuperquadricSource> superquadricSource =
   vtkSmartPointer<vtkSuperquadricSource>::New();
superquadricSource->SetPhiRoundness (epsilon_1);
superquadricSource->SetThetaRoundness (epsilon_2);
superquadricSource->SetScale (a, b, c);
superquadricSource->SetPhiResolution (1000);
superquadricSource->SetThetaResolution (1000);
superquadricSource->SetSize (1.);
```

As the code suggests (setting ϕ and θ resolutions), this does nothing but uniformly sample the parameter space for the two angles and generate points using the explicit equation. This source can be then connected to a renderer and used to display the generated mesh - see Figure 5.

2.2 Explicit

Here, we uniformly sample the two angular parameters of the superquadric, and set the correct mesh connectivity. As with the VTK results, we see that the meshes are not spatially sampled in an uniform way - there are not enough points around the corners and the middle of the shape - see Figure 6.

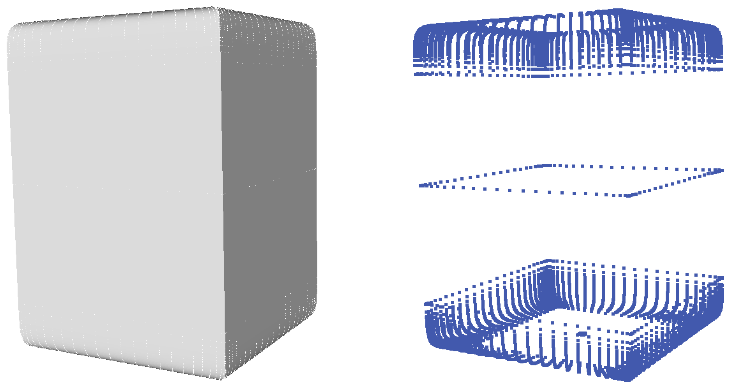


Figure 6: Superquadric sampling by uniformly sampling the angular parameter space.

2.3 Uniform Spatial Sampling

In order to obtain uniform spatial sampling, which is much more suitable for computing errors, we need to look deeper into the problem. [13] suggests a fast and mathematically sound solution. We start our derivation with the implicit and explicit equations of a superellipse:

$$\left(\frac{x}{a_1}\right)^{2/\epsilon} + \left(\frac{y}{a_2}\right)^{2/\epsilon} = 1 \tag{9}$$

$$x(\theta) = \begin{pmatrix} a_1 \cos(\theta)^\epsilon \\ a_2 \sin(\theta)^\epsilon \end{pmatrix} \tag{10}$$

The arclength between two close points $x(\theta)$ and $x(\theta + \Delta_\theta(\theta))$ on such a curve can be approximated by the euclidean distance between them:

$$D(\theta) = |x(\theta) - x(\theta + \Delta_\theta(\theta))| \tag{11}$$

Because $\Delta_\theta(\theta)$ is small, we can rely on a first order approximation:

$$D(\theta)^2 = \left(\frac{\partial}{\partial\theta}(a_1 \cos(\theta)^\epsilon)\Delta_\theta(\theta)\right)^2 + \left(\frac{\partial}{\partial\theta}(a_2 \sin(\theta)^\epsilon)\Delta_\theta(\theta)\right)^2 \tag{12}$$

which is equivalent to:

$$\Delta_\theta(\theta) = \frac{D(\theta)}{\epsilon} * \sqrt{\frac{\cos(\theta)^2 \sin(\theta)^2}{a_1^2 \cos(\theta)^{2\epsilon} \sin(\theta)^4 + a_2^2 \sin(\theta)^{2\epsilon} \cos(\theta)^4}} \tag{13}$$

By setting $D(\theta)$ to a constant spatial sampling value, the incremental updates of the angular parameter are:

$$\theta_i = \theta_{i-1} + \Delta_\theta(\theta_i) \text{ , with } \theta_0 = 0, \text{ and } i \in \{1...N\}, \text{ and } \theta_N < \pi/2 \tag{14}$$

$$\theta_i = \theta_{i-1} - \Delta_\theta(\theta_i) \text{ , with } \theta_0 = \pi/2, \text{ and } i \in \{1...N\}, \text{ and } \theta_N > 0 \tag{15}$$

There are singularities at $\theta = 0$ and $\theta = \pi/2$, so we need to treat them separately. When $\theta \to 0$, the superellipse can be approximated by:

$$x(\theta) = \begin{pmatrix} a_1 \\ a_2\theta^\epsilon \end{pmatrix} \tag{16}$$

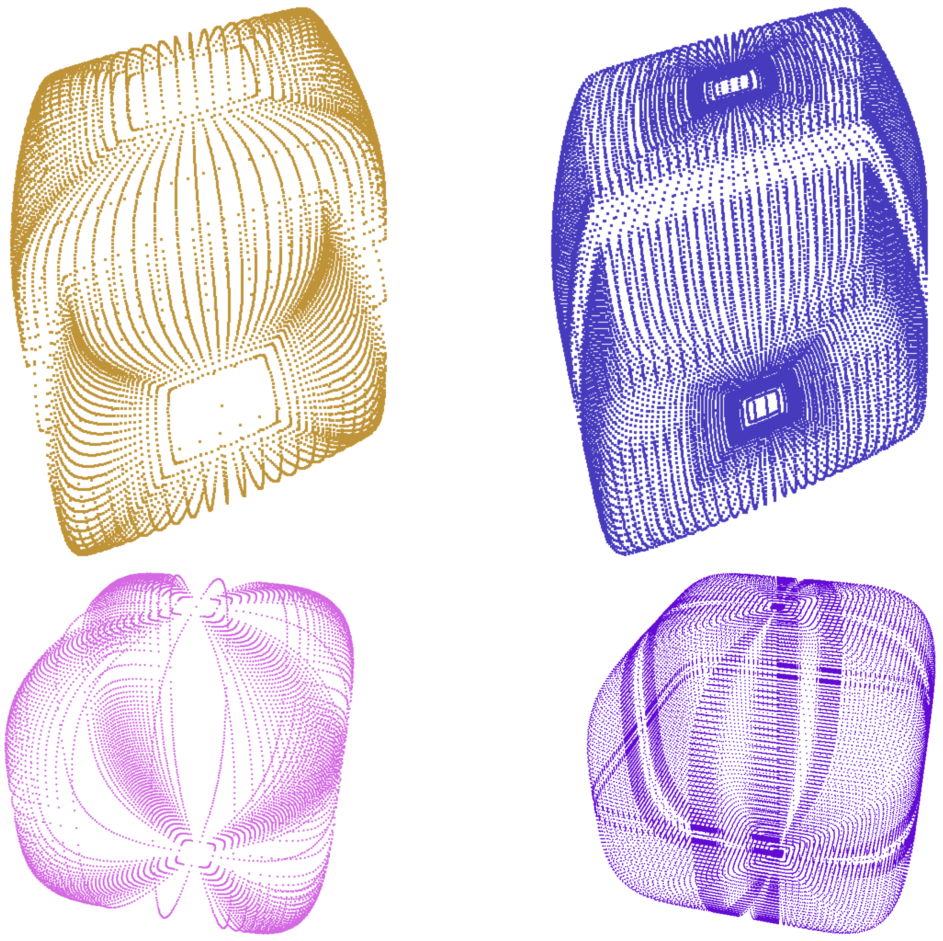


Figure 7: Superquadric sampling by adapting the angular parameter sampling so that the 3D space is uniformly sampled.

resulting in:

$$\Delta_\theta(\theta) = \left(\frac{D(\theta)}{a_2} - \theta^\epsilon\right)^{1/\epsilon} - \theta \tag{17}$$

Similarly, for $\theta \to \pi/2$:

$$\Delta_\theta(\theta) = \left(\frac{D(\theta)}{a_1} - (\pi/2 - \theta)^\epsilon\right)^{1/\epsilon} - (\pi/2 - \theta) \tag{18}$$

The final solution is to switch between the 3 branches depending on the current value of θ.

What was described until now is applied to a single superellipse. In order to spatially sample a superquadric in an uniform way, we sample one superellipse for ϵ_1 and one for ϵ_2 and then compute their spherical product (see Section 1.3 for details). Figure 7 shows the results we obtained.

3 Fitting

Error functions

One could simply use the implicit equation for fitting the superquadric models, but experiments show that the optimizer tends to converge to models that incorporate the given point set as a small part of their shape (i.e., very large solids). As such, [9] suggest to scale the error function by the square root of the product of the scales of the superquadric, in order to penalize large volumes. This technique is employed and confirmed by [10], [5]. Observe that the additional ϵ_1 exponent is used in order to have a correct distance measure independent of the shape parameter. In addition, [5] suggest to use the radial Euclidean distance instead of the normal Euclidean distance (i.e., scale the Euclidean distance by OP, the distance between the point and the center of the superquadric).

$$min\sqrt{a_1a_2a_3}\sum_{k=0}^{n}||OP|| * \left[\left(\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\epsilon_2}}\right)^{\epsilon_2/\epsilon_1} + \left(\frac{z}{a_3}\right)^{2/\epsilon_1}\right)^{\epsilon_1} - 1\right] \tag{19}$$

In our implementation, we first used the unstable branch of the Eigen library [7] which contains a rough Levenberg Marquardt algorithm implementation. Due to its lack of customizability, we looked for better options. As such, we decided to use the CERES solver library [2]. This is a large library employed for applications such as Google Street View and Google Maps. It is essentially a portable C++ library that allows for modeling and solving large complicated nonlinear least squares problems.

At each iteration of the LM algorithm, it needs to compute the Jacobian matrix of the vector of variables. CERES allows for three options:

- using the explicit formula in the code. We employed Maple to generate the C code from the symbolic partial differentiation of the error function; this yielded good results.
- using numerical differentiation. Apart from the fact that it has to compute the error function twice, there are a lot of numerical stability problems due to the very large range of values involved in the calculation, straining the floating point representation past its limits.
- using the auto differentiation module of CERES. By using a special type named Jet, CERES is able to compute the symbolic derivatives of our expressions. Jet is an implementation of n-dimensional dual numbers. We decided to use this technique in the end, as it allows for quick and easy modifications of the error functions, and it yielded good performance.

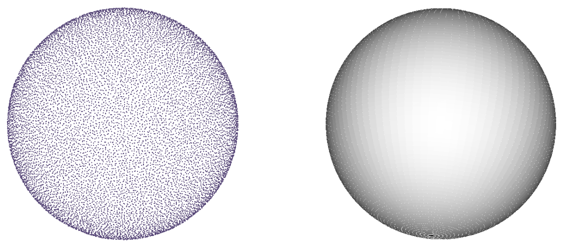
Initial fitting parameters

Due to the presence of numerous local minima in the superquadric function landscape, we need to have a good initialisation for the optimizer. One immediate idea is to set the translation to be the center of the point set we are trying to fit the model into.

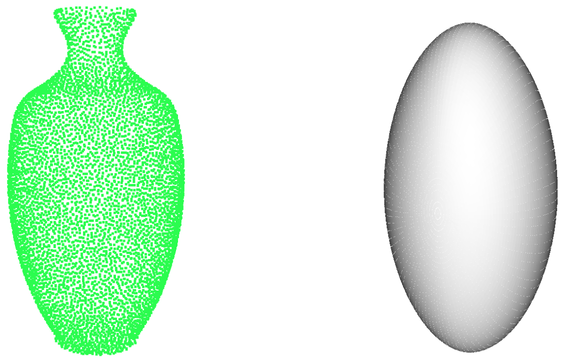
In addition, [4] suggests that the initial solution pose for the superquadric fitting optimisation is obtained by aligning the z-axis of the superquadric with each of the 3 central moments of the point set (i.e., the 3 PCA decomposition axes). We then choose the fitting with the best error out of the 3.

Furthermore, [15] propose that ϵ_1 and ϵ_2 be initialised at their midrange values of 1 (considering that the range is $[0.1, 1.9]$). Similar to the previous paper, the orientation is to be set such that the axes of the superquadric are aligned to the principal axes of the data. Moreover, we assume that the point distribution on the principal axes is Gaussian, so we can initialise the 3 scale parameters of the superquadric to be 3 times the standard deviation of the points on each of the PCA axis (the square roots of the eigenvalues of the covariance matrix of the point set).

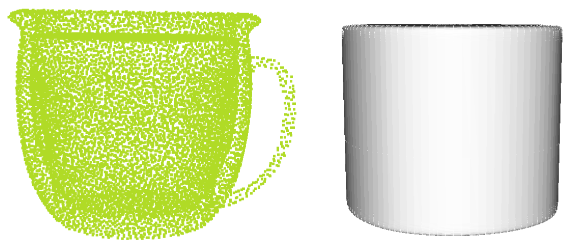
A set of results are visualised in Figure 8.



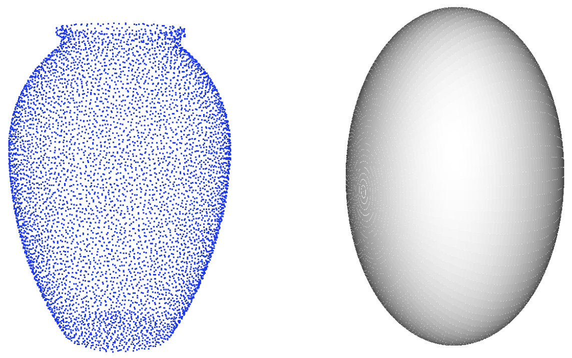
(a)



(b)



(c)



(d)

Figure 8: Results of superquadric fitting on point clouds of complete single objects.

3.1 Multipart Fitting with a Split & Merge Approach

Another interesting direction we looked into is that of explaining a set of points by multiple superquadric models. [16] propose a simple approach where a point set is segmented based on the local curvature, and then each cluster is fitted with a parametric model. Although the results presented in the paper looked reasonably good, we did not go for this technique (although we presented a similar per-cluster object fitting technique in Section 5), as we foresaw possible robustness problems.

As a result, we found the split and merge approach of Chevalier et al. [5] to be a better choice.

Splitting

- fit a superquadric to the current set of 3D points
- if the fitting error is less than a threshold T, then we stop (i.e., this cluster is well fitted to a superquadric)
- otherwise, the set of points is split in two regions, by cutting it with the plane orthogonal to the principal axis of the point set and going through the centroid of the points.
- the procedure is then called recursively on each half-subset until the number of points is too small for the cluster to be split again.

Merging

- for each subset of points, determine its list of neighboring subsets. (*)
- try to merge the subset with each of its neighbors, choosing the merge that has a fitting error below a threshold T' and the volume of the new superellipsoid is smaller than the sum of the volumes of the two superellipsoids fitted in the cluster before.
- re-iterate until no more merges are done.

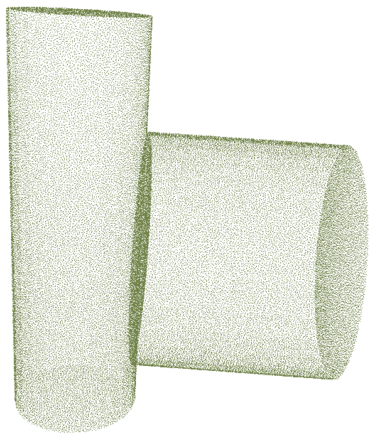
(*) - the original paper does not explain how the neighbouring subsets are found. We employ a simple technique to solve this issue: when doing the splitting, we store all the separating planes. Two subsets are considered to be neighbouring if the segment connecting the centroids intersects less than 3 splitting planes.

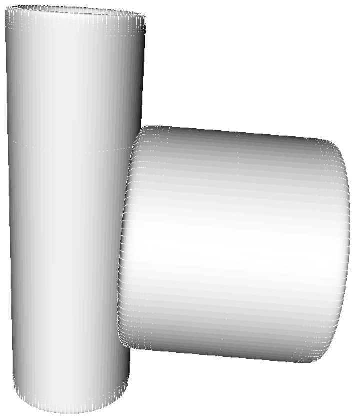
Figure 9 shows a few results we got by using this algorithm on synthetic data. We tested this technique on real datasets, but with no good results, as it is very sensitive to incomplete data. But it seems to work fairly well on complete CAD models. Further testing should be done here with full 360 degrees KinFu scans.

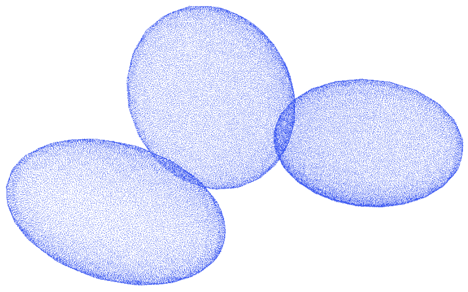
4 Detection

One of most popular techniques for detecting parametrized models in point cloud scenes is RANSAC [6]. Superquadric object detection has been tackled in [1], [11], and [4], and we considered that the approach of Biegelbauer et al. [4] had the most convincing results and we implemented it.

Figure 10 shows the general idea behind Biegelbauer's solution. The first step is to preprocess the data by removing the largest plane and possibly segmenting clusters of points that might represent separate objects in order to ease the detection process. Next, we generate m hypotheses by selecting m (~ 20) random seed points from the scene. Then, we find the points in the neighbourhood of the seed point of size equal to the minimum scaling parameter of the superquadric we are looking for ($min(a_1, a_2, a_3)$). Inside this small set of points, we randomly choose 30 points and try to rigidly align the superquadric we are looking for. We repeat this process $ransac\_restarts$ times (~ 10) and choose the best fitting pose. This pose is then used as an initialisation for a fitting procedure that includes all the points in the neighborhood. We then compute the fitting error and the number of interior and surface points to the parametric model within a neighbourhood of size $1.2 * max(a, b, c)$ around the center of the superquadric. All the possible poses are ranked by the


(a)


(b)


(c)

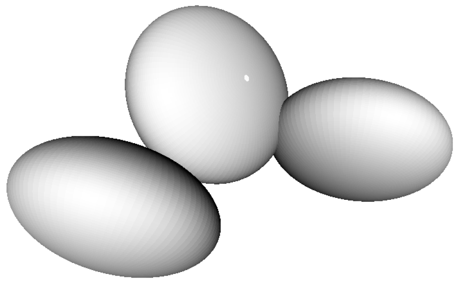
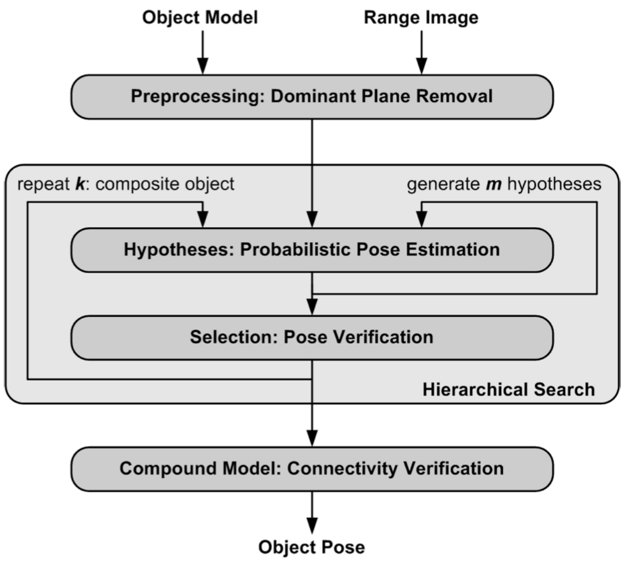

(d)

Figure 9: Split and merge algorithm results.



Object Model
Range Image
Preprocessing: Dominant Plane Removal
repeat k: composite object
generate m hypotheses
Hypotheses: Probabilistic Pose Estimation
Selection: Pose Verification
Hierarchical Search
Compound Model: Connectivity Verification
Object Pose

Figure 10: Superquadric detection pipeline as proposed by Biegelbauer et al. [4].

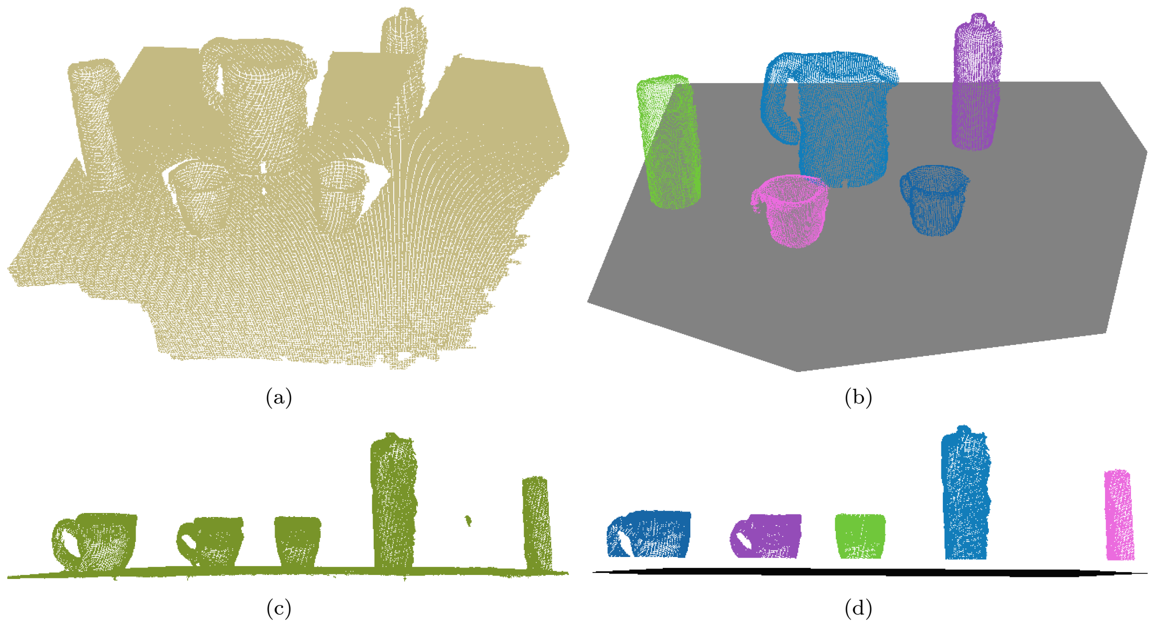


Figure 11: Our proposed tabletop segmentation: (a, c) the input cloud; (b, d) the output: the planar polygon and the separate object clusters that can be further processed.

fitting error, number of surface and interior points and the one which has the lowest sum of ranks is chosen as the output.

One needs to note that finding the pose of a given superquadric inside a set of points is done similarly to the superquadric fitting presented in Section 3, but only the 6 parameters for the pose are allowed to vary. However, we use the same initialisation techniques based on the principal axes of the point set.

5 Tabletop Demo

In order to demonstrate the superquadric algorithms we implemented on real world datasets, we have put together a pipeline for processing RGB-D data:

1. Scan a scene for a few seconds (with an Asus Xtion Pro RGB-D camera) and register the frames using PCL KinFu. The input of the next stages is the point cloud obtained from the vertices of the 3D accumulation grid of KinFu.
2. Find the largest plane using RANSAC on the whole cloud.
3. Perform Euclidean clustering on the points that were identified to lie in the plane in order to find the largest continuous surface and discard the other points, and recompute the plane equation.
4. Project the planar points onto the plane and compute the concave hull in 2D. Simplify the 2D contour to a polygon with a few vertices.
5. (Optional) Filter the non-planar points by discarding the ones that are not above/below the plane polygon (i.e., their projection onto the plane does not lie within the 2D polygon).
6. With the non-planar points, do Euclidean clustering again in order to separate different objects.

Dataset	Segmentation time [s]	Successful cluster fitting [s]
kinfu_1	22.3	6.2
kinfu_2	28.7	6.4
kinfu_3	20.8	5.8
kinfu_4	22.3	6.2
kinfu_5	28.4	5.5

Table 1: Performance timing of superquadric fitting on segmented RGB-D data.

A couple of example results of this procedure can be seen in Figure 11.

With the segmented data, we proceed in two directions: one is to fit a superquadric object to each separate cluster and one is to locate an object given its superquadric parametric description.

Superquadric fitting in real RGB-D data

Figure 12 shows the results we got and Table 1 the time performance of our implementation.

Superquadric detection in real RGB-D data

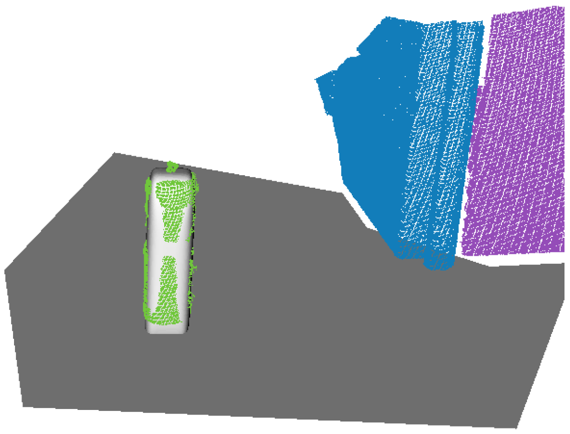
Next, we look into the object detection pipeline and apply to our tabletop context. We perform superquadric object detection with the parameters learnt for a bottle for each cluster separately. The results are shown in Figure 13.

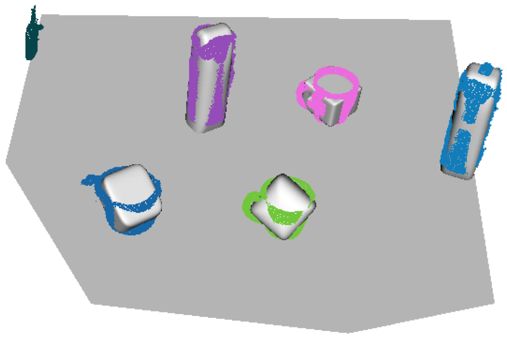
6 Conclusion and Future Work

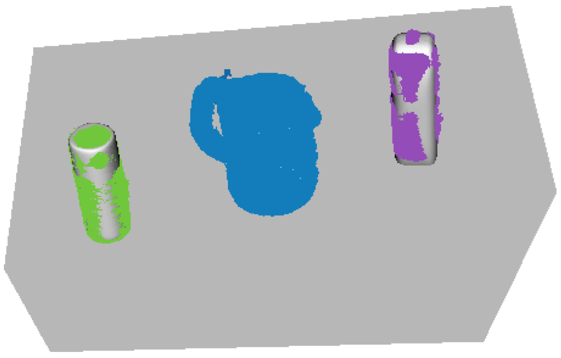
We have presented the theoretical background and the results we obtained during the second Toyota Code Sprint in collaboration with Open Perception. The outcome encourages to continue exploring the possible applications of parametric models in order to describe real world scans for robotics or commercial applications.

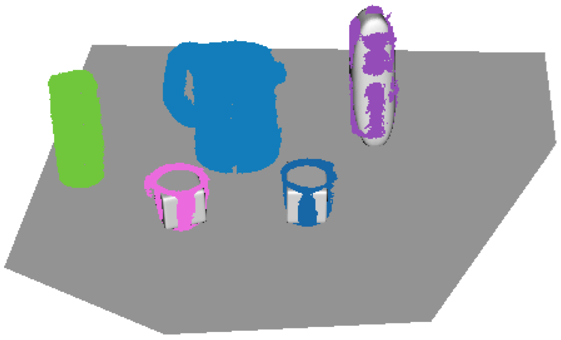
For future work, we will look into the following:

- integrate the code into PCL - this has been currently avoided due to the additional dependency to CERES and due to the highly experimental status of the implementations.
- tutorial/article on the PCL website to attract attention to the results of the project and get external feedback, or possible ideas for improvement.
- evaluate the superquadric fitting using a vfh_clustering type algorithm and compare it to other state-of-the-art object detection/fitting approaches.
- explore further possibilities of point cloud data modeling, such as [12], [14]
- superquadrics with global deformations.
- look into using different parametric solid models, other than superquadrics, that might yield more descriptive power.


(a)


(b)


(c)


(d)

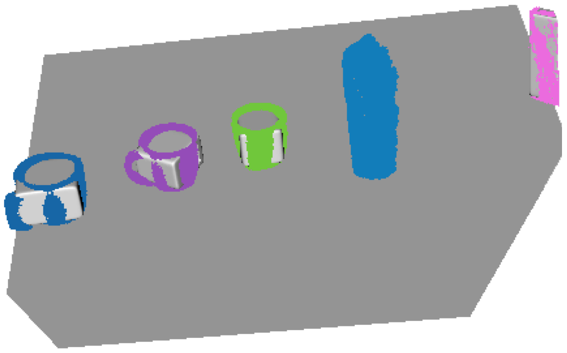
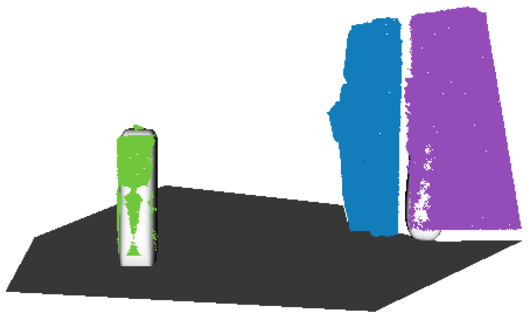
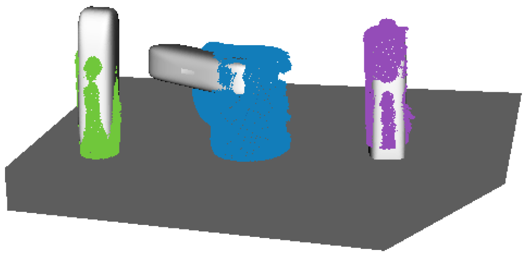

(e)

Figure 12: Superquadric fitting results of segmented RGB-D data


(a)


(b)

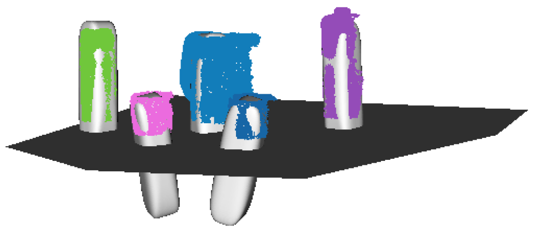

(c)

Figure 13: Superquadric detection results of segmented RGB-D data

References

[1] Ilya Afanasyev, Luca Baglivo Nicolo'Biasi, and Mariolino De Cecco. 3d object localization using superquadric models with a kinect sensor.

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `https://code.google.com/p/ceres-solver/`.

[3] A.H. Barr. Superquadrics and angle-preserving transformations. *Computer Graphics and Applications, IEEE*, 1(1):11–23, 1981.

[4] Georg Biegelbauer and Markus Vincze. Efficient 3d object detection by fitting superquadrics to range image data for robot's object manipulation. In *ICRA*, pages 1086–1091. IEEE, 2007.

[5] Laurent Chevalier, Fabrice Jaillet, and Atilla Baskurt. Segmentation and superquadric modeling of 3d objects. In *WSCG*, 2003.

[6] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[7] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[8] Alexandru-Eugen Ichim. Pcl toyota code sprint final report - smoothing. `https://github.com/PointCloudLibrary/blog/raw/master/blogweb/tocs/aichim/files/tocs_final_ichim.pdf`, 2012.

[9] Ales Jaklic, Ales Leonardis, and Franc Solina. *Segmentation and Recovery of Superquadrics*, volume 20 of *Computational imaging and vision*. Kluwer, Dordrecth, 2000. ISBN 0-7923-6601-8.

[10] Dimitrios Katsoulas, Christian Cea Bastidas, and Dimitrios I. Kosmopoulos. Superquadric segmentation in range images via fusion of region and boundary information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(5):781–795, 2008.

[11] Jaka Krivic and Franc Solina. Superquadric-based object recognition. In Wladyslaw Skarbek, editor, *CAIP*, volume 2124 of *Lecture Notes in Computer Science*, pages 134–141. Springer, 2001.

[12] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52:1–52:12, 2011.

[13] Maurizio Pilu and Robert B. Fisher. Equal-distance sampling of superellipse models, 1995.

[14] Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Trans. Graph.*, 31(6):180:1–180:11, November 2012.

[15] Yijun Xiao and J Siebert. Building superquadric men from 3d whole-body scan data. In *4th IEEE Chapter Conference on Applied Cybernetics*, pages 82–88, 2005.

[16] Yan Zhang, Andreas Koschan, and Mongi A. Abidi. Superquadric representation of automotive parts applying part decomposition. *Journal of Electronic Imaging*, 13(3):411–417, 2004.