Fill and Transfer:
A Simple Physics-based Approach for Containability Reasoning

Lap-Fai Yu[1] Noah Duncan[2] Sai-Kit Yeung[3]
[1]University of Massachusetts Boston, [2]University of California, Los Angeles,
[3]Singapore University of Technology and Design

Abstract

The visual perception of object affordances has emerged as a useful ingredient for building powerful computer vision and robotic applications [31]. In this paper we introduce a novel approach to reason about liquid containability - the affordance of containing liquid. Our approach analyzes container objects based on two simple physical processes: the Fill and Transfer of liquid. First, it reasons about whether a given 3D object is a liquid container and its best filling direction. Second, it proposes directions to transfer its contained liquid to the outside while avoiding spillage. We compare our simplified model with a common fluid dynamics simulation and demonstrate that our algorithm makes human-like choices about the best directions to fill containers and transfer liquid from them. We apply our approach to reason about the containability of several real-world objects acquired using a consumer-grade depth camera.

1. Introduction

One of the fundamental metrics to categorize man-made objects is functionality, also called object affordance [11, 13], which Gibson defined as the opportunities for action provided by a particular object. Object affordance is an important concept in the field of visual perception. The human cognition system possesses subtle yet powerful reasoning ability when it comes to analyzing the affordances of everyday objects in a scene. This ability enables humans to correctly manipulate objects to finish daily tasks.

Most of the man-made objects in our living environment are designed to possess certain object affordances. Pointed out by Sullivan [34] as a famous design postulate, "form ever follows function". Tversky [35] further noted, "function can often be inferred from form, in part because form can determine function". This indeed explains why the pitcher in Figure 1(a) has a narrow mouth (so that liquid can be poured out steadily via its mouth), and why the hole-ridden tray in Figure 1(b) is not designed for holding liquid (as liquid can leak through its holes).

Figure 1(c) shows a collection of different types of objects, whose visual appearance varies significantly in terms of geometry and surface reflectance. However, all these objects share the same object affordance of containability. Figure 1(d) shows a plate and a cup, each able to contain liquid. However, for carrying liquid, one usually prefers to use the cup instead of the plate, because the plate will spill the liquid if perturbed even slightly but the cup will not. Our goal is to devise a simple approach that enables computers to make these choices without expensive computation.

Figure 1. Some non-trivial questions that our simple approach can answer given a 3D object. (a) Which direction shall we tilt the pitcher to pour the water out? (b) Shall we use this container to hold water? (c) Which object is a good liquid container? (d) Shall we use a plate or a cup to carry water?

Recent advancement of consumer-grade 3D depth sensors (*e.g.* Occipital Structure Sensor[1]) allows even general users to easily acquire full-view 3D data of everyday objects, using a mobile device. The growing availability of good-quality 3D data increases the level of visual analysis that computers can perform to understand objects and scenes, for example, by physics-based reasoning [15, 40, 41]. Current vision research aims to use this data to enable computers to understand and manipulate everyday objects as naturally as humans do.

We share the same stance as in *qualitative physics* [9]: simple physics-based models can be more suitable than precise numerical simulations for reasoning about many practical daily situations. This stance can be justified by the fact that humans can interact seamlessly with the objects in their everyday surroundings, without knowing the equations and numerical parameters required by traditional physics to describe the physical interactions happening.

To this end, we propose a simplified physics-based approach to perform containability analysis of objects. We demonstrate that our approach can answer non-trivial questions about containability (Figure 1) and hence make the right manipulation decisions. We experimentally verify our approach by comparing its outputs with human preferences and with a common fluid dynamics simulation.

[1] http://structure.io

2. Related Work

Containability. Containability has been categorized as one of the common functional affordance features. The affordance network [36] defines containability as the ability of objects to hold solids or liquids transferred to it[2]. Geometrically, containability is defined using the criteria of "high convexity". Examples include bowls, cups and pots. However, this simple definition does not provide sufficient reasoning power to distinguish between different types of containers nor provide information about how the container should be manipulated to accomplish various related tasks. Our novelty lies in devising a computational approach to perform such containability analysis, such that the fill and transfer direction of a possible container can be automatically deduced. To the best of our knowledge, no existing work has tackled this exact problem.

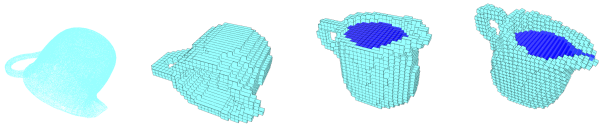
Physics-based Reasoning. There are recent successes [2, 15, 40, 41] in using physics to reason about object properties and scenes in computer vision and robotics communities. For example, physics-based stability reasoning can effectively improve scene segmentation and support detection. Battaglia et al. [2] used simulation as a reasoning engine to perform physical scene understanding. Their experiments verified that humans do rough physical simulation when trying to understand or make predictions about the physical world, which resonates with the observations by McCloskey on *intuitive physics* [27]. Our work is inspired by these, in which we do simplified physics-based reasoning to analyse containability. Our work also coincides with the research direction of *qualitative physics* [9], which is about developing computational representations and models that capture the ways by which people reason about properties in the physical world, in our case, the containability of objects. Our work is also motivated by psychological research [3, 28] which finds that, humans, even as infants, categorize objects according to their affordance.

Affordance Prior. There are also recent breakthroughs in computer vision about using affordance prior [1, 12, 14, 16, 17, 19, 22] to analyze the functionality of objects or scenes. Grabner et al. [12] demonstrated that by using human sitting poses as priors, one can effectively distinguish between chairs and non-chairs in a 3D object database. Gupta et al. [14] introduced an approach for understanding a scene in terms of what affordances it supports. The method builds a joint space which can effectively predict a set of possible human poses from a single image. Kim et al. [19] used human pose priors to perform human-centric shape analysis on 3D models of man-made objects. Zhao and Zhu [39] proposed an integrated framework to perform 3D scene parsing by jointly reasoning about functionality, geometry and appearance of the 3D scene. Yu et al. [38] synthesized feasible indoor scenes by incorporating common functionality priors. There are also recent successes in using affordance priors to detect or reason about functionality regions of objects in 2D images [37, 42]. Refer to [7] for a comprehensive review.

Object Manipulation. An important goal in computer vision is to enable computers to automatically understand and reason about daily objects and scenes, even for the intrinsic object properties and physical relations, like humans do on a daily basis [4, 15]. Such reasoning ability has direct applications, for example, it can enable robots to manipulate with living environments like humans do, such as to open new doors [20], to grasp novel objects [29, 30], and to manipulate objects according to their perceived affordances in general [18, 21]. Our approach also progresses towards this goal, by automatically detecting container objects and reasoning about their manipulation.

With regard to container manipulation, Brandl et al. [5] used warped parameters to transfer pouring actions from a source object to a target object. The source object is manually annotated with semantic labels, which are transferred to the target object by warping the source object to match the target. The fill and transfer directions are predefined specific to the semantic labels (e.g. handle, rim) of the mug-like containers used. Kunze et al. [23] proposed to use physical simulation to reason about object manipulation. In their example of pouring liquids, the container is prefilled and the transfer direction is predefined. Compared to these works, our approach does not require any annotation and focuses on automatically deducing both the filling and transfer directions for general containers. Recently, Liang et al. [26] also found that physical simulation is a good approximation for evaluating human cognition of containing relations, through a series of interesting simulation experiments

[2]In this work, we do not consider *containability closure* as defined in [36]. A tetrapak is an example of such.



Point cloud Voxelized Fill Transfer

Figure 2. Overview of our approach. Cyan voxels refer to the container, blue voxels refer to the containee. Our approach automatically identifies containers and determines its fill and transfer directions.

3. Overview

Figure 2 shows an overview of our approach. Our containability analysis is motivated by the following high-level tasks which are commonly performed by humans on liquid containers. *Fill* refers to filling a container with the containee substance. *Transfer* refers to manipulating a container filled with the containee substance so that some or all of the containee substance is transferred elsewhere, either to another container or to the external environment. Although these tasks are simple to describe, several non-trivial questions must be answered in order to carry them out satisfactorily: How to identify a container? How to fill up a container? And how to manipulate the filled-up container?

In our approach, we simplify our problem by only considering the gravitational force, and the normal forces exerted by the container. The former prevents a containee from going upward, and the latter prevents a containee from passing through the container. Without violating these restrictions, a containee will try to go as downwards as possible to lower its gravitational potential energy.

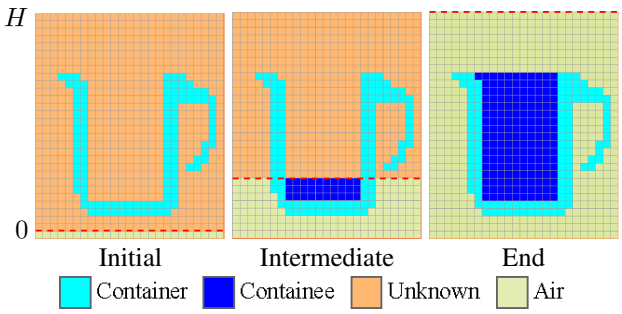


Figure 3. Computing the containee volume. Initial: all the voxels at height level $h = 0$ are preset to have state *Air*. Intermediate: our algorithm propagates state *Air* by flood-fill at the current height level (voxel layer just below the red dashed line). All the voxels not flood-filled have state *Containee*. End: After flood-filling all the height level, the containee volume is found by counting all the *Containee* voxels.

Algorithm 1 Compute Containee Volume

Input:Voxel state grid $\mathbf{W}$.

for $h = 1$ **to** H **do**
  Set empty queue $Queue$;
  foreach $W_{x,h,z}$ **do**
    if $W_{x,h,z}$ is *Unknown and* $W_{x,h-1,z}$ is *Air* **then**
      Push $W_{x,h,z}$ to $Queue$;
  1) Push corner voxel $W_{0,h,0}$ to $Queue$;
  2) Do flood-fill on height level h using $Queue$, set each flood-filled voxel to have state *Air*;
  3) Set each $W_{x,h,z}$ with state *Unknown* to *Containee*;
  4) Compute Containee volume $V(\mathbf{W})$ as the total number of *Containee*;
  5) Update $h^* = h$ if $V(\mathbf{W})$ is changed

Output:$V(\mathbf{W})$ and h^*.

Figure 4. Pseudocode for computing the containee volume.

4. Approach

4.1. Voxel Representation

Our approach works in the voxel domain. The input can be a point cloud of a 3D model, or a 3D CAD model in mesh format which we convert to a point cloud by running a ray-casting style virtual scanner that samples points on the model's surface. We voxelize the world into a 3D grid of equal-sized, cubic voxels. The size of the world is defined such that it encloses the entire bounding box of the container under all possible self-rotations. To facilitate subsequent computations, we enlarge the tight bounding box defined above by padding three voxels at its sides.

We use a simple voxelization scheme. Let $\mathbf{W}$ be the volume containing all the voxels in the defined world. Each voxel $W_{x,y,z}$ is indexed by its location along the x, y and z axes. At the beginning, each voxel has two states: *Container* and *Unknown*. *Container* indicates that the voxel is part of the container. *Unknown* indicates that the voxel is not part of the container and its state is yet to be determined. Our goal is to set all the *Unknown* states into *Containee* or *Air*. *Containee* indicates that the voxel is filled with the containee substance, e.g., liquid. *Air* indicates that the voxel is not filled with the containee substance. Initially, we set all the voxels at the lowest level to have state *Air*, as this will facilitate our algorithm's execution.

4.2. Fill

In this part, our algorithm searches for the direction to fill up the container that results in the maximum containee volume, which we refer to as the best filling axis. To achieve this, our algorithm makes various attempts to fill up the container along different filling axes (poses), guided by a smoothing-based optimization. We first detail our algorithm for computing containee volume under a particular pose. After that we can search for the best filling axis (pose) that gives the maximum containee volume.

Computing Containee Volume. Given a container in its current pose, and suppose the gravitational force is acting downwards ($[0, -1, 0]^T$), we want to compute the volume of containees that can fill up the container without leakage and the corresponding height level h^*. We refer to this volume as the containee volume $V(\mathbf{W})$ of the container.

We propose a remarkably simple algorithm without any fluid simulation to perform this task. Instead of pouring *Containee* (liquid) from the top, we flood fill the *Air* from the bottom. The key insight is that, although we do not know which voxels belong to the "inside" of the container to contain the containee, we know that the voxel at the lower-left corner at the current height level h must belong to *Air*, because we have padded the world such that it is larger than the bounding box of the container under any self-rotation (Section 4.1). Therefore, when we flood-fill from the lower-left corner to propagate the *Air* state, any adjacent *Unknown* voxel will become *Air*. However, the *Container* voxels will block the *Air* from flood-filling into its "inside", preserving the *Unknown* voxels.

Therefore, when the flood-fill is done, any voxel at the current height level h either has state *Air*, *Container* or *Unknown*. All the *Unknown* voxels are turned into *Containee*. Our algorithm starts from height level $h = 1$ and runs incrementally at each height level until $h = H$ where H is the maximum height in $\mathbf{W}$. Notice that the previous height level will not be affected by the current height level. To speed up, our algorithm re-uses the states determined at the previous height level when computing for the current height level. This is done by pushing all the *Unknown* voxels with an underneath *Air* voxel to the flood-fill queue, before the flood-fill begins. Refer to Figure 3 for an illustration and Figure 4 (**Algorithm 1**) for the pseudocode. The complexity of **Algorithm 1** is $\mathcal{O}(|\mathbf{W}|)$, where $|\mathbf{W}|$ denotes the number of voxels used, as each voxel is processed at most once.

In short, our algorithm goes through each height level to determine the *Containee* voxels in $\mathbf{W}$. Finally it outputs the containee volume $V(\mathbf{W})$ as the total number of *Containee* voxels, and the corresponding height level h^*.

Best Filling Axis Search. In general, a container object does not carry any notion of how it should be filled. Given a filling axis $\mathbf{f}$, we want to find out the corresponding containee volume assuming the gravitational force is acting towards $[0, -1, 0]^T$. In practice, this is equivalent to rotating

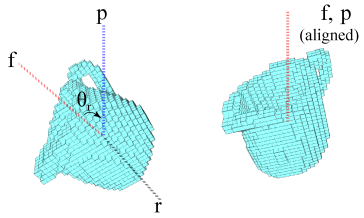


Figure 5. Left: before alignment. Right: after alignment. Rotating the container object by $R_{\mathbf{f}\to\mathbf{p}}$ aligns $\mathbf{f}$ with $\mathbf{p}$. The container object can then be filled up assuming the gravitational force is acting towards $[0,-1,0]^T$.

the object such that $\mathbf{f}$ aligns with $[0,1,0]^T$ and then performing the filling.

Define $\mathbf{p} = [0,1,0]^T$. Since both $\mathbf{f}$ and $\mathbf{p}$ are known, computing the rotation function $\mathbf{R}_{\mathbf{f}\to\mathbf{p}}$ is trivial. $\mathbf{R}_{\mathbf{f}\to\mathbf{p}}$ is parametrized by only one rotation angle $\theta_\mathbf{r}$ along the rotational axis $\mathbf{r}$ perpendicular to the plane spanned by $\mathbf{f}$ and $\mathbf{p}$, i.e., $\mathbf{r} = \frac{\mathbf{p}\times\mathbf{f}}{||\mathbf{p}\times\mathbf{f}||}$ and $\theta_\mathbf{r} = -2\cos^{-1}(\mathbf{p}^T\mathbf{f})$. Figure 5 denotes their relationships.

Once we rotate the container by $\mathbf{R}_{\mathbf{f}\to\mathbf{p}}$, we can compute the containee volume $V(\mathbf{W})$ by **Algorithm 1**. The best filling axis $\mathbf{f}^*$ is defined as the filling axis $\mathbf{f}$ which results in the maximum containee volume:

$$\mathbf{f}^* = \arg\max_{\mathbf{f}} V(\mathbf{R}_{\mathbf{f}\to\mathbf{p}}(\mathbf{W})) \tag{1}$$

We use a smoothing-based optimization [25] to guide the search for the best filling axis $\mathbf{f}^*$, as follows:

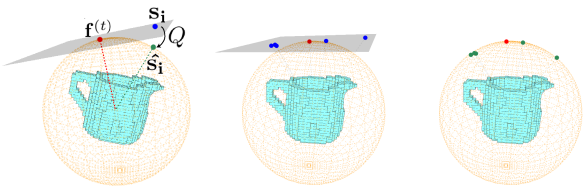
Initialization. We uniformly sample 12 filling axes over the surface of a unit sphere and compute their corresponding containee volumes. We choose the filling axis corresponding to the largest containee volume as our initialization.

Optimization. Starting with the initial filling axis, the smoothing-based optimization proceeds iteratively. At the current iteration t, it performs two updates: 1) the current filling axis estimate, $\mathbf{f}^{(t)}$; 2) the variance of the current estimate, $(\sigma^{(t)})^2$. The optimization stops when the variance is smaller than a threshold, that is, when we are very certain that the current filling axis estimate is close to an optimum.

Both practically and theoretically [25], the optimization process represents the estimate using a Gaussian distribution from which samples are drawn to update the estimate and variance. This is non-trivial in our scenario where our estimates are constrained to be the directions over a unit sphere. In other words, given the current filling axis estimate $\mathbf{f}^{(t)}$, which is taken as the mean of a Gaussian distribution, and the variance of the current estimate, we need to sample directions over the unit sphere from this Gaussian distribution.

While Gaussian sampling over the unit sphere is non-trivial, we can instead perform Gaussian sampling over the tangent plane at the intersection of the current filling axis estimate $\mathbf{f}^{(t)}$ and the unit sphere, and project the samples back from the tangent plane to the unit sphere. This is motivated by the recent work of Straub et al. [33] which proposed an elegant representation to encode scene normals as clusters of distributions over a unit sphere.

Specifically, at iteration t of the optimization process, given the current filling axis estimate, $\mathbf{f}^{(t)}$, and



Notations Before Projection After Projection

Figure 6. Left: notations. Right: projecting the Gaussian samples (blue) from the tangent plane (gray) about the current filling axis estimate (red) to the unit sphere surface, via the Riemannian logarithm map Q. The projected samples (green) are shown.

the variance of the current estimate, $(\sigma^{(t)})^2$, we sample points $\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_k}$ from the 2D Gaussian distribution $\mathcal{N}(\mathbf{f}^{(t)}, (\sigma^{(t)})^2\mathbf{I})$ defined over the tangent plane about $\mathbf{f}^{(t)}$ ($\mathbf{I}$ is the identity matrix). Denote $\mathbf{u_i} = \mathbf{s_i} - \mathbf{f}^{(t)}$. Each of these sample points $\mathbf{s_i}$ are then projected back to a point $\hat{\mathbf{s}}_\mathbf{i}$ on the unit sphere via the Riemannian logarithm map[3] Q [6, 8] (Figure 6):

$$\hat{\mathbf{s}}_\mathbf{i} = Q(\mathbf{f}^{(t)}, \mathbf{u_i}) = \mathbf{f}^{(t)}\cos(||\mathbf{u_i}||) + \frac{\mathbf{u_i}}{||\mathbf{u_i}||}\sin(||\mathbf{u_i}||) \tag{2}$$

Gaussian samples can hence be generated on the tangent plane and used for the smoothing-based optimization. At each iteration during the optimization, we sample points $\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_k}$ to perform updates.

Current Filling Axis Update. We update the current filling axis by:

$$\mathbf{u_i} = \mathbf{s_i} - \mathbf{f}^{(\mathbf{t})} \tag{3}$$

$$\hat{\mathbf{s}}_\mathbf{i} = Q(\mathbf{f}^{(t)}, \mathbf{u_i}) \tag{4}$$

$$\bar{\mathbf{u}} = \frac{\sum_{i=1}^{k} V(\mathbf{R}_{\hat{\mathbf{s}}_\mathbf{i}\to\mathbf{p}}(\mathbf{W}))\mathbf{u_i}}{\sum_{i=1}^{k} V(\mathbf{R}_{\hat{\mathbf{s}}_\mathbf{i}\to\mathbf{p}}(\mathbf{W}))} \tag{5}$$

$$\mathbf{f}^{(t+1)} = Q(\mathbf{f}^{(t)}, \bar{\mathbf{u}}) \tag{6}$$

In our experiments we used $k = 5$ sample axis directions drawn from the Gaussian distribution on the tangent plane about axis $\mathbf{f}^{(t)}$. The above equations essentially compute the new axis from the projection of the weighted sum of the Gaussian samples, and we use the containee volumes (computed by **Algorithm 1**) corresponding to each sample axis direction as the weights.

Current Variance Update. we update the variance of the current estimate by:

$$(\sigma^{(t+1)})^2 = \frac{1}{2}\frac{\sum_{i=1}^{k} V(\mathbf{R}_{\hat{\mathbf{s}}_\mathbf{i}\to\mathbf{p}}(\mathbf{W}))\mathbf{u_i}^T\mathbf{u_i}}{\sum_{i=1}^{k} V(\mathbf{R}_{\hat{\mathbf{s}}_\mathbf{i}\to\mathbf{p}}(\mathbf{W}))} \tag{7}$$

At each iteration, the variance tells us how certain our current estimate is and this certainty is automatically updated based on the neighborhood, a very useful property of smoothing-based optimization. The optimization stops when the variance is smaller than a threshold (0.1 in our experiments). It usually stops in about 4-5 iterations in our experiments. Figure 7 shows an example.

[3] Refer to the supplementary material for a proof.

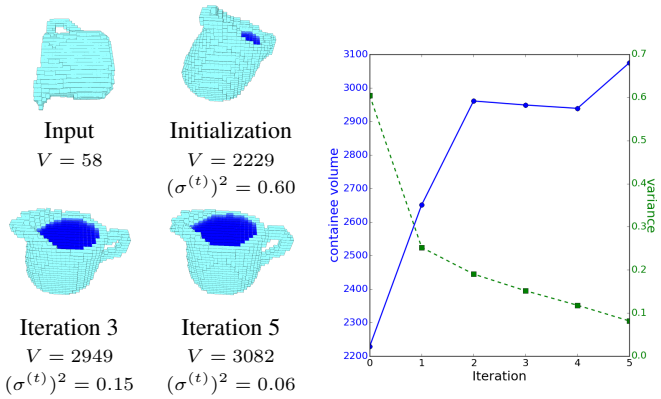


Figure 7. Containee volume and variance over iterations.

Identifying Containers. Our algorithm makes use of the best filling axis and the corresponding maximum containee volume, to determine whether an object is a container. This is done by computing the percentage of the maximum containee volume out of the total volume of the container plus containee. If the percentage is smaller than a threshold (10% in our experiments), we consider that the object is not a container. Otherwise it proceeds to the next step to deduce the transfer directions. Figure 10 shows some examples.

4.3. Transfer

Next we reason about the *Transfer* part, that is, the manipulation of the filled-up container such that some or all of its containees are transferred to a destination. We suppose that a desirable transfer should avoid spillage, *i.e.* all of the transferred containees should end up at the destination rather than elsewhere. To reduce the chance of spillage, one possible measure is that the cross-sectional area of the flow of liquid from the container to the destination should be minimized, so that the overlap with the destination is maximized. Refer to our supplementary material for assumptions and justification. Our approach approximately calculates the cross-sectional area in the voxel domain by counting the number of voxels that the "leaking" containee voxels will pass through as they transfer from the container.

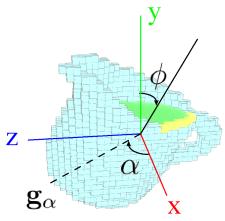


Figure 8. Notations.

Assume the container has already been rotated to stand upright according to the best filling axis as described in Section 4.2. Formally each transfer direction is described as a tilt axis. We want to determine the tendency of containees leaking out as the container is being tilted about different tilt axes. It is intuitive to again use an axis-angle representation to represent the tilt motion. Specifically, each tilt motion can be described by a tilt axis $\mathbf{g}_\alpha = [\cos(\alpha), 0, \sin(\alpha)]^T$ lying on the x-z plane and a tilt angle ϕ. Our goal is to compute a transfer metric for each tilt axis $\mathbf{g}_\alpha$, and make use of the transfer metrics to deduce the best tilt axis(s) for *Transfer*.

Transfer Metrics Computation. We propose to use a simple metric, by counting the number of voxels N at the leakage boundary (Figure 9 Right) if we add one more layer of containee under the tilted status.

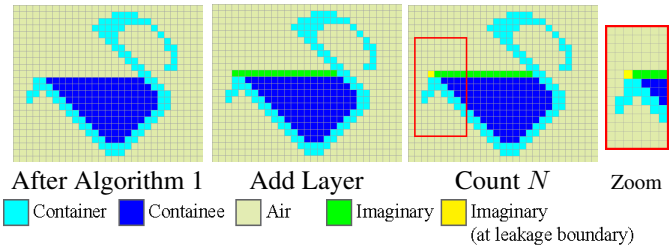


Figure 9. Computing N (**Algorithm 2**). Left: our algorithm first applies **Algorithm 1** to fill the container up to height level h^*. Middle: it then grows a layer of *Imaginary* voxels at height level h^*+1. Right: it computes N by counting the number of *Imaginary* voxels at the leakage boundary (*i.e.Imaginary* voxels adjacent to an *Air* voxel at height level $h^* + 1$).

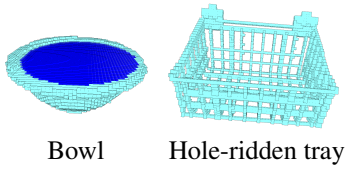


Figure 10. Identifying containers. Left: the bowl is identified as a container. Right: the hole-ridden tray which cannot be filled up by containee is identified as a non-container.

Algorithm 2 Compute N

Input:Voxel state grid $\mathbf{W}$ of the container tilted about tilt axis $\mathbf{g}_\alpha$ by angle ϕ.
Initialize:
Run **Algorithm 1** to fill up the container and set states in $\mathbf{W}$. Get height level h^*;
Set empty queue $Queue$;
foreach $W_{x,h^*+1,z}$ **do**
  if $W_{x,h^*+1,z}$ is *Air and* $W_{x,h^*,z}$ is *Containee* **then**
    Push $W_{x,h^*+1,z}$ to $Queue$;

Do flood-fill on height level $h^* + 1$ using $Queue$, as follow:

- set each flood-filled voxel to have state *Imaginary*.
- push neighbor voxel to $Queue$ if neighbor voxel is supported by a $Container$ underneath.

Output:N as the number of *Imaginary* voxels at the leakage boundary.

Figure 11. Pseudocode for computing N.

Suppose the container is tilted about axis $\mathbf{g}_\alpha$ by angle ϕ. We fill up the container to the maximum containee volume following **Algorithm 1**, and get the corresponding height level h^*. Now, consider an imaginary layer of containee voxels added at height level $h^* + 1$, i.e., we assign a new state *Imaginary* to the *Air* voxels as long as their underneath voxels are *Containee*, and this is followed by a flood-fill (Figure 9 Middle). State *Imaginary* refers to those imaginary voxels that had just leaked under the current tilting motion.

Now, we compute N by counting the number of *Imaginary* voxels that have an adjacent *Air* voxel at the same height level $h^* + 1$. See Figure 9 for illustration and Figure 11 (**Algorithm 2**) for the pseudocode. The complexity of **Algorithm 2** is $\mathcal{O}(|\mathbf{W}|)$, where $|\mathbf{W}|$ is the number of

voxels used, since each voxel is processed at most once. Figure 12 visualizes the leakage voxels under different tilt axes with $\phi = 25°$. Figure 13 shows N along different tilt axes at different tilt angles.

We will use N to define our transfer metric $Z_{\mathbf{g}}$. For notational convenience, we drop the subscript α of $\mathbf{g}_\alpha$ in the following, as we are referring to a fixed tilt axis. For each tilt axis $\mathbf{g}$, we compute the sum $Z_{\mathbf{g}} = \sum_\phi N_{\mathbf{g},\phi}$, where $N_{\mathbf{g},\phi}$ refers to the N obtained as the container is tilted about axis $\mathbf{g}$ by angle ϕ. Intuitively, a larger $Z_{\mathbf{g}}$ means that there will be more leakage voxels, i.e., a larger surface area to leak as the container is tilted about tilt axis $\mathbf{g}$ at different tilt angles.

Determining Transfer Directions. In our experiments, we sample the tilt axis $\mathbf{g}_\alpha = [\cos(\alpha), 0, \sin(\alpha)]$, where $\alpha = 0°, 10°, ..., 360°$, and the tilt angle $\phi = 0°, 1°, ..., 45°$, unless otherwise specified. We compute $Z_{\mathbf{g}}$ for each tilt axis. To alleviate the quantization problem caused by discrete sampling as can be observed in Figure 13, we smooth the sums $Z_{\mathbf{g}}$ by a Hann window of size 10. To find out if the $Z_{\mathbf{g}}$ of all the tilt axes are similar, we compute the normalized standard deviation of all the $Z_{\mathbf{g}}$. If the normalized standard deviation is smaller than 0.1, we conclude that there is no particular preference and hence all the tilt axes (transfer directions) are equally-likely. Otherwise, we locate the local minimum(s) of $Z_{\mathbf{g}}$, and propose the set of tilt axis(s) $\{\mathbf{g}_\alpha\}$ corresponding to the local minimum(s) as the tilt axis(s) that the container should be tilted about to transfer. Figure 14 shows an illustration.

5. Experiments

5.1. Data

We first perform our experiments on a dataset of 3D models obtained from the Princeton Shape Benchmark [32] and the Trimble 3D Warehouse. This dataset consists of 72 objects, including 44 common household container objects (from keywords *e.g.*'pitcher', 'cup') and 28 random objects.

User Annotations. As there is no notion of containability defined on these 3D models, we resort to human users to provide the ground-truth annotations. We recruit 50 human users to annotate the dataset. Each human user annotates half of the dataset, via a 3D user interface. The user is shown the objects one by one. For each object, the user first determines whether it is a container or not[4]. If it is a container, s(he) annotates its best filling axis. Then s(he) is asked to transfer liquid from the filled-up container to another container, while avoiding spillage. S(he) is asked whether all the transfer directions are equally-likely, or s(he) has some preferred transfer directions. In the latter case, s(he) annotates all the preferred transfer directions. Figure 16 shows some objects and results computed by our approach. Please refer to the supplementary material for the full list of the annotated objects and results.

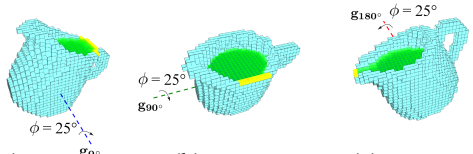
5.2. Results

Implementation. We implemented our algorithms using C++ and Python. We performed our experiments on a laptop running Intel Core i7-4800MQ @ 2.7GHz CPU with 8GB RAM. Overall, our unoptimized implementation takes about 2 minutes to analyse each object.

Fluid Simulation. Our approach relies on numerous evaluations of **Algorithm 1** to obtain the containee volumes. To verify its correctness, we compare **Algorithm 1** with a physics-based fluid simulation algorithm, the Lattice Boltzmann Method (LBM), commonly used in computational fluid dynamics. We use the LBM solver[5] provided by

[4]We asked users to fill up the object as possible by only considering the object's geometry, as our approach only considers geometry.

[5]http://wiki.blender.org/index.php/Doc:2.6/



(a) $\mathbf{g}_{0°}$ (b) $\mathbf{g}_{90°}$ (c) $\mathbf{g}_{180°}$

Figure 12. Tilting a container. Green refers to the imaginary layer of leakage voxels. Yellow refers to the voxels at the leakage boundary. (a–c) Tilting about tilt axes $\mathbf{g}_{0°}$, $\mathbf{g}_{90°}$ and $\mathbf{g}_{180°}$ respectively by tilt angle $\phi = 25°$.

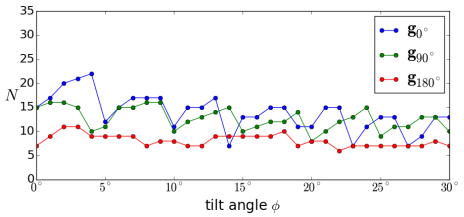
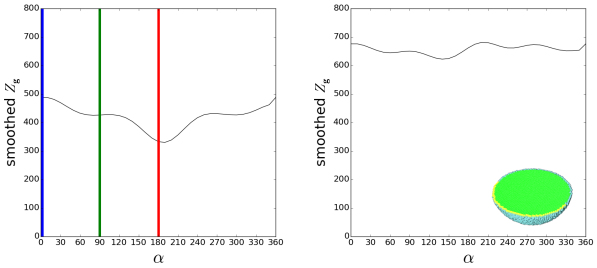


Figure 13. N vs. tilt angle ϕ about different tilt axes. Blue, green, red curves correspond to Figure 12 (a), (b) and (c).



Pitcher Bowl

Figure 14. Determining the preferred tilt axis. The plots show the smoothed $Z_{\mathbf{g}}$ (*i.e.* N sum) of different tilt axes. Left: plot of the pitcher. The blue, green, and red lines corresponds to Figure 12 (a), (b) and (c) respectively. The tilt axis $\mathbf{g}_{180°}$ is chosen, which corresponds to the local minimum (red). Right: plot of the bowl. All tilt axes are equally-likely, as indicated by the small normalized SD (< 0.1) of its smoothed $Z_{\mathbf{g}}$.

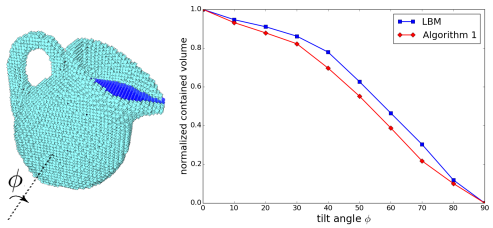


Figure 15. A comparison with fluid simulation (LBM). Left: a pitcher being tilted about a fixed tilt axis. Right: the corresponding normalized containee volumes found by **Algorithm 1** and LBM. The curves show similar trends.

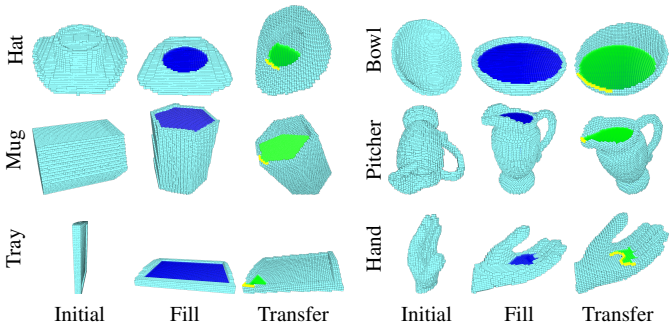


Figure 16. Containability analysis of some objects in our dataset.

Blender. For each object in our dataset, we tilt it about a fixed tilt axis and run LBM to obtain the containee volumes. Figure 15 shows the results of tilting a pitcher. The curves obtained from **Algorithm 1** and LBM show similar trends. To compare, we normalize each curve and calculate the absolute difference between the normalized volumes at each tilt angle. The average difference per tilt angle is 0.083 over all the objects. On average, one simulation by LBM takes 543 seconds (9 minutes) while **Algorithm 1** takes 0.06 seconds. **Algorithm 1** requires significantly less time and hence is a more favorable choice for our containability analysis which requires numerous evaluations.

User Annotations Comparison. We compare the outputs of our approach with the user annotations. Table 1 shows the accuracy of identifying containers, using our simple heuristic of computing the ratio of the maximum containee volume to the object size (Sec 4.2 and Figure 10). The overall accuracy is 94.44%.

Next we compare the best filling direction found by our approach to those annotated by the users. The user annotations on the best filling directions agree quite well in general, with a standard deviation of about $21.14°$. We use the average of the user-annotated best filling directions as the ground-truth. Our average angular error drops from $30.52°$ at initialization to $16.64°$ after optimization.

Table 2 shows the accuracy of our approach in choosing whether all transfer directions are equally likely or if one or more are preferred. For objects annotated with preference(s), Figure 17 (Left) shows the accuracy of our approach via precision and recall. For each of our proposed tilt axes, we check if its deviation from any user-annotated tilt axis is less than the angular error tolerance. If so we count it as a match. Our approach achieves a precision of about 0.75 when the tolerance is $10°$, and the precision rises to 0.82 when the tolerance is $20°$. Please refer to the supplementary material for details of the user study settings, the results of each object and some observations.

Adding Noise. We analyze the robustness of our approach by adding varying levels of Gaussian noise to the input point clouds. Refer to Figure 17 for the effects of noise in the accuracy of identifying containers and estimating the tilt axis. Our approach performs reasonably well. For a standard deviation of 0.5 voxel length, the overall accuracy of identifying containers falls to 84.72% and the precision in choosing transfer direction falls to 0.70 (using angular error tolerance of $20°$). Please refer to the supplementary material for details of the analyses.

`Manual/Physics/Fluid/Technical_details`

Overall Accuracy	False-Positive	False Negative
94.44%	2.78%	2.78%

Table 1. Accuracy of identifying containers.

	Equally-likely (users)	Preference (users)
Equally-likely (ours)	100.00%	20.00%
Preference (ours)	0.00%	80.00%

Table 2. Comparison between user choices and our automatic output on whether all the transfer directions are equally-likely or if there is(are) preference(s).

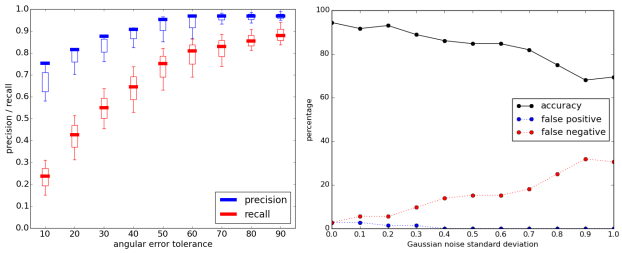


Figure 17. Left: Precision and recall versus angular error tolerance in tilt axis estimation. Markers refer to the results from noise-free data. Box plots refer to the results from data contaminated with Gaussian noises of SD in $[0.0, 1.0]$. Right: Accuracy of identifying containers versus Gaussian noise SD.

Real-world Scenes. We apply our approach to analyse objects in real-world scenes captured by a consumer-grade depth camera. Figure 18(a) shows one indoor scene from the UW RGB-D Scene Dataset [24] captured by Kinect. Figure 18(b)-(g) shows other scenes and objects captured by a Structure Sensor attached on an iPad, which can acquire full-view 3D data. We use the segmentation functionality provided by the Structure SDK to extract segmented objects from the scenes. Our approach can identify the containers within the clutter of objects and deduce the filling and tilt axes.

Choosing between Containers. Our simplified physics-based approach can help computers answer other containability-related questions. For example, by reasoning about simple tilting motions, computers can recognize an important difference between containers. Figure 18(b) shows an example to choose between a cup and a plate for transferring liquid. The cup is a better choice, because as it is tilted, the rate of decrease of its containee volume is smaller compared to that of the plate. Therefore the cup's ability to contain liquid is more robust to perturbation which usually happens during transfer.

6. Discussion

Assumptions and Limitations. We discuss the limitations of our approach resulting from the assumptions we make to scope our problem.

In the *Fill* part, our approach assumes that a container can always be filled by liquid from the outside. In other words, it does not handle containability closure [36], hence it classifies the sealed can in Figure 18(a) as a non-container. Our approach also assumes that a container has only one filling direction, hence it may not work on a container with multiple regions for holding liquid, as the filling directions for different regions may be conflicting. Further, depending on the resolution of the 3D data acquisition device, small holes may not be captured and considered. Figure 19 (Left)

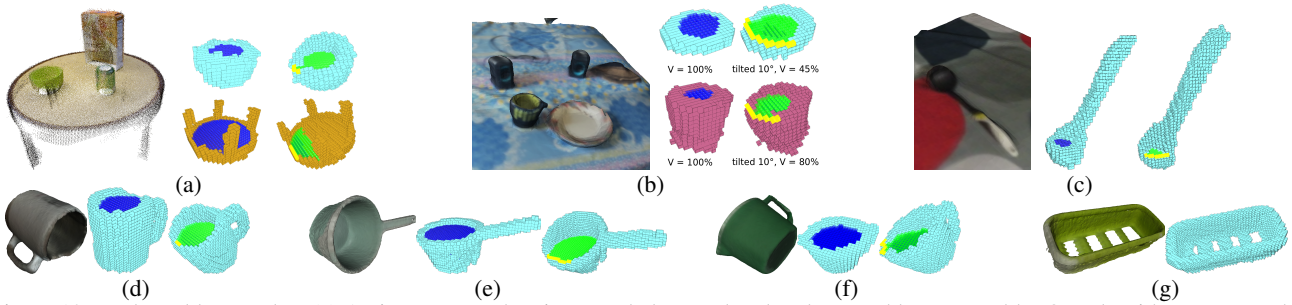


Figure 18. Real-world examples. (a) An input scene showing a sealed can, a bowl and a cereal box on a table. Our algorithm suggests that the bowl is a container and all its transfer directions are equally-likely, and that the table after being inverted is also a possible container. (b) An input scene showing a cup, a plate, two speakers and a piece of cloth. The cup and plate are identified as possible containers. The cup is found to be more robust to slight perturbation, as it has a smaller rate of decrease of containee volume (V) when slightly tilted. (c)-(g) Results on other real-world objects captured by a Structure Sensor. The hole-ridden tray in (g) is identified as a non-container.

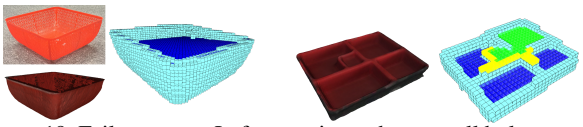


Figure 19. Failure cases. Left: a strainer whose small holes are not captured by the sensor is being filled up. Right: the internal flow of liquid between the cells in a bento box is treated as a transfer.

shows a failure case where a strainer is wrongly filled up as if it was a container.

In the *Transfer* part, our approach only analyzes simple tilting motion, and it assumes that the liquid transfer process should avoid spillage. While we believe this is a reasonable and useful assumption, it may not hold in some situations, particularly when the destination is large compared to the container. For example, when pouring water out of a container to put out a fire, one may just pour the water out as quickly as possible without caring about the spillage. We believe that a voxel-based simulation approach can still be adopted to reason about such situations. On the other hand, liquid may flow from one region to another region within a container, such as for the bento box shown in Figure 19 (Right). This may confuse our approach in deducing the direction to transfer liquid to the outside.

Our approach does not consider human priors, which can be useful cues for deducing containability: a container should be graspable by a human hand; when pouring liquid out of a handled cup, one would probably grasp the handle and pour liquid in a direction opposite of the handle's location. A container should also have a reasonable size and weight that a human can manipulate. Without considering these, our approach classifies the inverted table in Figure 18(a) as a possible container.

Our approach assumes that the container has a static geometry which is structurally strong enough to hold the liquid. Our approach does not consider the container's material, which can be important in affordance reasoning. For example, one would probably not fill up a paper bag with water, though its shape suggests that it can be a container, because paper absorbs water and the paper bag's geometry would deform. Finally, our approach does not handle partial 3D data (obtained from a RGB-D image) in general due to the difficulty of deciding between real holes and missing data. We believe a learning-based approach may tackle this limitation better and we leave it for future work.

Other Features. Other physical or geometric features may be helpful in deducing containability. For example, one may deduce the upright orientation (filling direction) of a container by assuming that it should stand stably on a flat surface [10]. Our approach instead uses the maximum containee volume to deduce the filling direction, as this measure is more directly aligned with our goal and can be used to distinguish between containers and non-containers. Additionally, geometric symmetry may be helpful for deducing the transfer direction, since the transfer motion is often a rotation about the plane of bilateral symmetry. Our approach partially considers symmetry by comparing the volume of liquid poured out in each direction (Figure 14). To speed up the approach, one may perform stability and symmetry analysis on the object in a preprocessing step, to rule out unstable orientations and redundant transfer directions to save computational time.

7. Future Work and Conclusion

We presented a novel, simplified physics-based approach to analyse containability, to automatically identify containers and deduce the fill and transfer directions. We are interested in devising physics-based approaches to reason about other common affordances too. We believe the ultimate goal of this research direction is to grant visual systems the cognitive power that human beings have in reasoning about and interacting with everyday objects and scenes.

8. Acknowledgements

We thank Michael Brown for narrating the video and William Koh for scanning the data. Sai-Kit Yeung is supported by SUTD-ZJU Collaboration Research Grant 2012 SUTDZJU/RES/03/2012, SUTD-MIT International Design Center Grant IDG31300106, and Singapore MOE Academic Research Fund MOE2013-T2-1-159. Part of the work was done when Noah was visiting SUTD and supported by Singapore MOE Academic Research Fund MOE2013-T2-1-159. We acknowledge the support of the SUTD Digital Manufacturing and Design (DManD) Centre which is supported by the Singapore National Research Foundation. Lap-Fai Yu is supported by the University of Massachusetts Boston StartUp Grant P20150000029280 and the Joseph P. Healey Research Grant Program provided by the Office of the Vice Provost for Research and Strategic Initiatives & Dean of Graduate Studies of the University of Massachusetts Boston.

References

[1] E. Bar-Aviv and E. Rivlin. Functional 3d object classification using simulation of embodied agent. In *BMVC*, 2006. 2

[2] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110:18327–18332, 2013. 2

[3] A. E. Booth and S. Waxman. Object names and object functions serve as cues to categories for infants. *Developmental Psychology*, 38(6):948–957, 2002. 2

[4] K. L. Bouman, B. Xiao, P. Battaglia, and W. T. Freeman. Estimating the material properties of fabric from video. In *ICCV*, 2013. 2

[5] S. Brandl, O. Kroemer, and J. Peters. Generalizing pouring actions between objects using warped parameters. In *Humanoids*, 2014. 2

[6] G. Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*. Springer, 2011. 4

[7] C. Desai and D. Ramanan. Predicting functional regions on objects. In *CVPR Workshops*, June 2013. 2

[8] M. do Carmo. *Riemannian Geometry*. Birkhäuser Verlag, 1992. 4

[9] K. D. Forbus. Qualitative physics: Past, present, and future. In H. E. Shrobe and AAAI, editors, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, pages 239–296. Kaufmann, 1988. 1, 2

[10] H. Fu, D. Cohen-or, G. Dror, and A. Sheffer. Upright orientation of man-made objects. *ACM Trans. Graph.*, 2008. 8

[11] J. J. Gibson. The theory of affordances. *Lawrence Erlbaum*, 1977. 1

[12] H. Grabner, J. Gall, and L. V. Gool. What makes a chair a chair? In *CVPR*, 2011. 2

[13] J. G. Greeno. Gibson's affordances. *Psychological Review*, pages 336–342, 1994. 1

[14] A. Gupta, S. Satkin, A. A. Efros, and M. Hebert. From 3d scene geometry to human workspace. In *CVPR*, 2011. 2

[15] Z. Jia, A. Gallagher, A. Saxena, and T. Chen. 3d-based reasoning with blocks, support, and stability. In *CVPR*, 2013. 1, 2

[16] Y. Jiang, H. S. Koppula, and A. Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *CVPR*, 2013. 2

[17] Y. Jiang, M. Lim, and A. Saxena. Learning object arrangements in 3d scenes using human context. In *ICML*, 2012. 2

[18] D. Katz, A. Venkatraman, M. Kazemi, J. A. D. Bagnell, and A. T. Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. In *RSS*, 2013. 2

[19] V. G. Kim, S. Chaudhuri, L. Guibas, and T. Funkhouser. Shape2pose: Human-centric shape analysis. *ACM Trans. Graph.*, 2014. 2

[20] E. Klingbeil, A. Saxena, and A. Y. Ng. Learning to open new doors. In *RSS Workshop on Robot Manipulation*, 2008. 2

[21] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013. 2

[22] H. S. Koppula and A. Saxena. Physically grounded spatio-temporal object affordances. In *ECCV*, 2014. 2

[23] L. Kunze, M. E. Dolha, E. Guzman, and M. Beetz. Simulation-based temporal projection of everyday robot object manipulation. In *AAMAS*, 2011. 2

[24] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *ICRA*, 2011. 7

[25] M. Leordeanu and M. Hebert. Smoothing-based optimization. In *CVPR*, 2008. 4

[26] W. Liang, Y. Zhao, Y. Zhu, and S.-C. Zhu. Evaluating human cognition of containing relations with physical simulation. In *CogSci*, 2015. 2

[27] M. McCloskey. Intuitive physics. *Scientific American*, 248(4):114–122, 1983. 2

[28] L. M. Oakes and K. L. Madole. Function revisited: How infants construe functional features in their representation of objects. *Adv. in Child Development and Behavior*, 36:135–185, 2008. 2

[29] A. Saxena, J. Driemeyer, and A. Ng. Robotic grasping of novel objects using vision. *IJRR*, 27(2):157, 2008. 2

[30] A. Saxena, L. Wong, and A. Y. Ng. Learning grasp strategies with partial shape information. In *AAAI*, 2008. 2

[31] Second Workshop on Affordances: Visual Perception of Affordances and Functional Visual Primitives for Scene Analysis, ECCV 2014. http://theaffordances.appspot.com/workshops/ECCV. Accessed: 4-12-2014. 1

[32] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The princeton shape benchmark. In *In Shape Modeling International*, pages 167–178, 2004. 6

[33] J. Straub, G. Rosman, O. Freifeld, J. J. Leonard, and J. W. Fisher III. A mixture of manhattan frames: Beyond the manhattan world. In *CVPR*, 2014. 4

[34] L. H. Sullivan. The tall office building artistically considered. *Lippincott's Magazine*, 1896. 1

[35] B. Tverskyi. Form and function. In L. A. Carlson and E. van der Zee, editors, *Functional features in language and space: Insights from perception, categorization and development*, pages 331–347. Oxford University Press, 2004. 1

[36] K. M. Varadarajan and M. Vincze. Afnet: The affordance network. In *ACCV*, 2012. 2, 7

[37] B. Yao, J. Ma, and L. Fei-Fei. Discovering object functionality. In *ICCV*, 2013. 2

[38] L.-F. Yu, S. K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. Osher. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30(4), 2011. 2

[39] Y. Zhao and S.-C. Zhu. Scene parsing by integrating function, geometry and appearance models. In *CVPR*, 2013. 2

[40] B. Zheng, Y. Zhao, J. C. Yu, K. Ikeuchi, and S.-C. Zhu. Beyond point clouds: Scene understanding by reasoning geometry and physics. In *CVPR*, 2013. 1, 2

[41] B. Zheng, Y. Zhao, J. C. Yu, K. Ikeuchi, and S.-C. Zhu. Detecting potential falling objects by inferring human action and natural disturbance. In *ICRA*, 2014. 1, 2

[42] Y. Zhu, A. Fathi, and L. Fei-Fei. Reasoning about object affordances in a knowledge base representation. In *ECCV*, 2014. 2