CHISEL: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially-Hashed Signed Distance Fields

Matthew Klingensmith
Carnegie Mellon Robotics Institute
mklingen@andrew.cmu.edu

Ivan Dryanovski
The Graduate Center,
City University of New York
idryanovski@gc.cuny.edu

Siddhartha S. Srinivasa
Carnegie Mellon Robotics Institute
siddh@cs.cmu.edu

Jizhong Xiao
The City College of New York
jxiao@ccny.cuny.edu

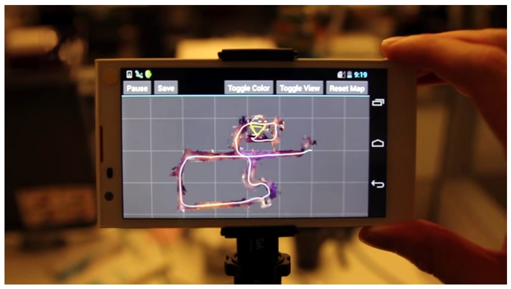
***Abstract*—We describe CHISEL: a system for real-time house-scale (300 square meter or more) dense 3D reconstruction onboard a Google *Tango* [1] mobile device by using a dynamic spatially-hashed truncated signed distance field[2] for mapping, and visual-inertial odometry for localization. By aggressively culling parts of the scene that do not contain surfaces, we avoid needless computation and wasted memory. Even under very noisy conditions, we produce high-quality reconstructions through the use of space carving. We are able to reconstruct and render very large scenes at a resolution of 2-3 cm in real time on a mobile device without the use of GPU computing. The user is able to view and interact with the reconstruction in real-time through an intuitive interface. We provide both qualitative and quantitative results on publicly available RGB-D datasets [3], and on datasets collected in real-time from two devices.**

I. INTRODUCTION

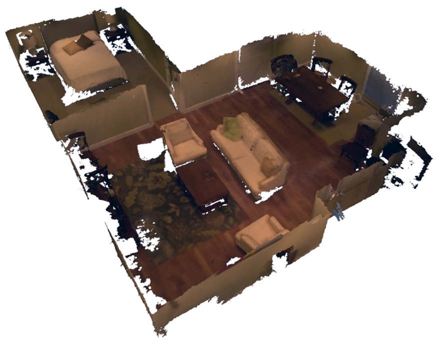
Recently, mobile phone manufacturers have started adding high-quality depth and inertial sensors to mobile phones and tablets. The devices we use in this work, Google's *Tango* [1] phone and tablet have very small active infrared projection depth sensors combined with high-performance IMUs and wide field of view cameras (Section IV-A). Other devices, such as the Occiptal Inc. *Structure Sensor* [4] have similar capabilities. These devices offer an onboard, fully integrated sensing platform for 3D mapping and localization, with applications ranging from mobile robots to handheld, wireless augmented reality.

Real-time 3D reconstruction is a well-known problem in computer vision and robotics [5]. The task is to extract the true 3D geometry of a real scene from a sequence of noisy sensor readings online. Solutions to this problem are useful for navigation, mapping, object scanning, and more. The problem can be broken down into two components: localization (*i.e.* estimating the sensor's pose and trajectory), and mapping (*i.e.* reconstructing the scene geometry and texture).

Consider house-scale (300 square meter) real-time 3D mapping and localization on a *Tango*-like device. A user (or robot) moves around a building, scanning the scene. At house-scale, we are only concerned with features with a resolution of about 2-3 cm (walls, floors, furniture, appliances, etc.). To facilitate scanning, real-time feedback is given to the user on the device's screen. The user can export the resulting 3D scan without losing any data. Fig.1 shows an example of this use



(a) CHISEL creating a map of an entire office building floor on a mobile device in real-time.



(b) Reconstructed apartment scene at a voxel resolution of 2cm.

Fig. 1: CHISEL running on Google's *Tango* [1] device.

case (Section III) in progress.

House-scale mapping requires that the 3D reconstruction algorithm run entirely onboard; and fast enough to allow real-time interaction. Importantly, the entire dense 3D reconstruction must fit inside the device's limited (2-4GB) memory (Section IV-A). Because some mobile devices lack sufficiently powerful discrete graphics processing units (GPU), we choose not to rely on general purpose GPU computing to make the problem tractable in either creating or rendering the 3D reconstruction.

3D mapping algorithms involving occupancy grids [6], keypoint mapping [7] or point clouds [8–10] already exist for mobile phones at small scale. But most existing approaches either require offline post-processing or cloud computing to create high-quality 3D reconstructions at the scale we are interested in.

Many state-of-the-art real-time 3D reconstruction algorithms [2, 11–14] compute a truncated signed distance field (TSDF) [15] of the scene. The TSDF stores a discretized estimate of the distance to the nearest surface in the scene. While allowing for very high-quality reconstructions, the TSDF is very memory-intensive. Previous works [12, 13] have extended the TSDF to larger scenes by storing a *moving* voxelization and throwing away data incrementally – but we are interested in preserving the volumetric data for later use. The size of the TSDF needed to reconstruct an entire house may be on the order of several gigabytes (Section IV-D), and the resulting reconstructed mesh has on the order of 1 million vertices.

In this work, we detail a realtime house-scale TSDF 3D reconstruction system (called CHISEL) for mobile devices. Experimentally, we found that most ($\sim 93\%$) of the space in a typical scene is in fact empty (Table I). Iterating over empty space for the purposes of reconstruction and rendering wastes computation time; and storing it wastes memory. Because of this, CHISEL makes use of a data structure introduced by Nießner *et al.* [2], the dynamic spatially-hashed [16] TSDF (Section III-F), which stores the distance field data as a two-level structure in which static 3D voxel *chunks* are dynamically allocated according to observations of the scene. Because the data structure has $\mathcal{O}(1)$ access performance, we are able to quickly identify which parts of the scene should be rendered, updated, or deleted. This allows us to minimize the amount of time we spend on each depth scan and keep the reconstruction running in real-time on the device (Table 9e).

For localization, CHISEL uses a mix of visual-inertial odometry [17, 18] and sparse 2D keypoint-based mapping [19] as a black box input (Section III-J), combined with incremental scan-to-model ICP [20] to correct for drift.

We provide qualitative and quantitative results on publicly available RGB-D datasets [3], and on datasets collected in real-time from two devices (Section IV). We compare different approaches for creating (Section IV-C) and storing (Section IV-D) the TSDF in terms of memory efficiency, speed, and reconstruction quality. CHISEL, is able to produce high-quality, large scale 3D reconstructions of outdoor and indoor scenes in real-time onboard a mobile device.

II. RELATED WORK

Mapping paradigms generally fall into one of two categories: landmark-based (or *sparse*) mapping, and high-resolution *dense* mapping [19]. While sparse mapping generates a metrically consistent map of landmarks based on key features in the environment, dense mapping globally registers all sensor data into a high-resolution data structure. In this work, we are concerned primarily with dense mapping, which is essential for high quality 3D reconstruction.

Because mobile phones typically do not have depth sensors, previous works [9, 21, 22] on dense reconstruction for mobile phones have gone to great lengths to extract depth from a series of registered monocular camera images. Since our work focuses on mobile devices with integrated depth sensors, such as the Google *Tango* devices [1], we do not need to perform costly monocular stereo as a pre-requisite to dense reconstruction. This allows us to save our memory and CPU budget for the 3D reconstruction itself.

One of the simplest means of dense 3D mapping is storing multiple registered point clouds. These point-based methods [8–10, 22] naturally convert depth data into projected 3D points. While simple, point clouds fail to capture local scene structure, are noisy, and fail to capture *negative* (non-surface) information about the scene. This information is crucial to scene reconstruction under high levels of noise [23].

Elfes [6] introduced Occupancy Grid Mapping, which divides the world into a voxel grid containing occupancy probabilities. Occupancy grids preserve local structure, and gracefully handle redundant and missing data. While more robust than point clouds, occupancy grids suffer from aliasing, and lack information about surface normals and the interior/exterior of obstacles. Occupancy grid mapping has already been shown to perform in real-time on *Tango* [1] devices using an Octree representation [24].

Curless and Levoy [15] created an alternative to occupancy grids called the Truncated Signed Distance Field (TSDF), which stores a voxelization of the signed distance field of the scene. The TSDF is negative inside obstacles, and positive outside obstacles. The surface is given implicitly as the zero isocontour of the TSDF. While using more memory than occupancy grids, the TSDF creates much higher quality surface reconstructions by preserving local structure.

In robotics, distance fields are used in motion planning [25], mapping [26], and scene understanding. Distance fields provide useful information to robots: the distance to the nearest obstacle, and the gradient direction to take them away from obstacles. A recent work by Wagner *et al.* [27] explores the direct use of a TSDF for robot arm motion planning; making use of the gradient information implicitly stored in the TSDF to locally optimize robot trajectories. We are interested in similarly planning trajectories for autonomous flying robots by directly using TSDF data. Real-time onboard 3D mapping, navigation and odometry has already been achieved on flying robots [28, 29]using occupancy grids. CHISEL is complimentary to this work.

Kinect Fusion [11] uses a TSDF to simultaneously extract the pose of a moving depth camera and scene geometry in real-time. Making heavy use of the GPU for scan fusion and rendering, *Fusion* is capable of creating extremely high-quality, high-resolution surface reconstructions within a small area. However, like occupancy grid mapping, the algorithm relies on a single fixed-size 3D voxel grid, and thus is not suitable for reconstructing very large scenes due to memory constraints. This limitation has generated interest in extending TSDF fusion to larger scenes.

Moving window approaches, such as *Kintinuous* [12] extend Kinect Fusion to larger scenes by storing a moving voxel grid in the GPU. As the camera moves outside of the grid, areas which are no longer visible are turned into a surface representation. Hence, distance field data is prematurely thrown away to save memory. As we want to save distance field data so it can be used later for post-processing, motion planning, and

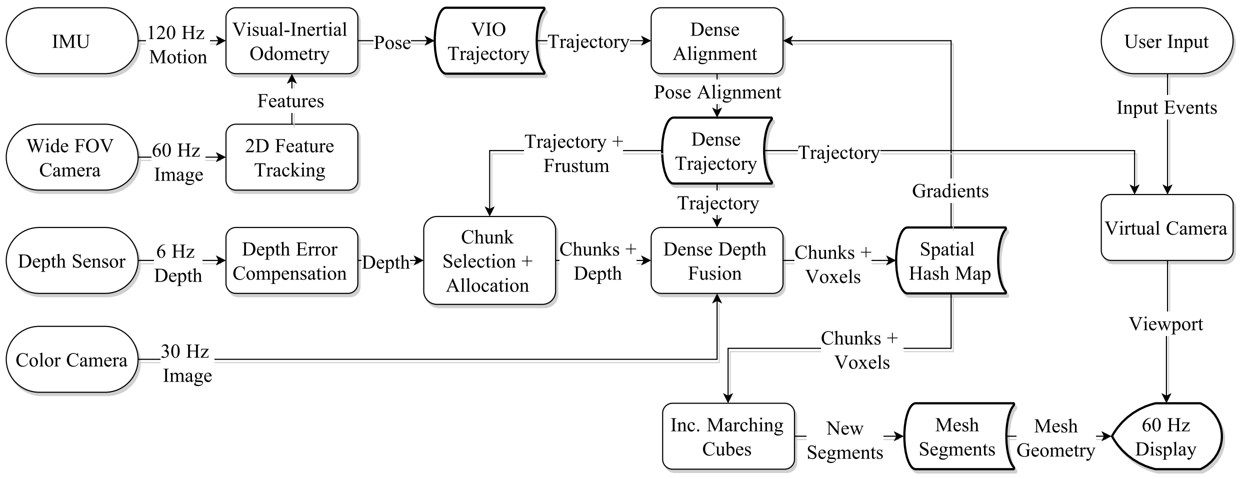


Fig. 2: CHISEL system diagram

other applications, a moving window approach is not suitable.

Recent works have focused on extending TSDF fusion to larger scenes by compressing the distance field to avoid storing and iterating over empty space. Many have used *hierarchal* data structures such as octrees or KD-trees to store the TSDF [30, 31]. However, these structures suffer from high complexity and complications with parallelism.

An approach by Nießner *et al.* [2] uses a two-layer hierarchal data structure that uses spatial hashing [16] to store the TSDF data. This approach avoids the needless complexity of other hierarchical data structures, boasting $\mathcal{O}(1)$ queries, and avoids storing or updating empty space far away from surfaces.

CHISEL adapts the spatially-hashed data structure of Nießner *et al.* [2] to *Tango* devices. By carefully considering what parts of the space should be turned into distance fields at each timestep, we avoid needless computation (Table II) and memory allocation (Fig.5a) in areas far away from the sensor. Unlike [2], we do not make use of any general purpose GPU computing. All TSDF fusion is performed on the mobile processor, and the volumetric data structure is stored on the CPU. Instead of rendering the scene via raycasting, we generate polygonal meshes incrementally for only the parts of the scene that need to be rendered. Since the depth sensor found on the *Tango* device is significantly more noisy than other commercial depth sensors, we reintroduce space carving [6] from occupancy grid mapping (Section III-E) and dynamic truncation (Section III-D) into the TSDF fusion algorithm to improve reconstruction quality under conditions of high noise. The space carving and truncation algorithms are informed by a parametric noise model trained for the sensor using the method of Nguyen *et al.* [32], which is also used by [2].

III. SYSTEM IMPLEMENTATION

A. Preliminaries

Consider the geometry of the ideal pinhole depth sensor. Rays emanate from the sensor origin to the scene. Fig.3a is a diagram of a ray hitting a surface from the camera, and a voxel that the ray passes through. Call the origin of the ray o, and the endpoint of the ray $\mathbf{x}$. The length of the ray is given by $z = \|\mathbf{o} - \mathbf{x}\|$. The direction of the ray is given by $\hat{\mathbf{r}} = \frac{\mathbf{o}-\mathbf{x}}{z}$. The endpoint of each ray represents a point on the surface. We can also parameterize the ray by its direction and endpoint, using an interpolating parameter u:

$$\mathbf{v}(u) = \mathbf{x} - u\hat{\mathbf{r}} \tag{1}$$

At each timestep t we have a set of rays Z_t. In practice, the rays are corrupted by noise. Call d the true distance from the origin of the sensor to the surface along the ray. Then z is actually a random variable drawn from a distribution dependent on d called the *hit probability* (Fig.3b). Assuming the hit probability is Gaussian, we have:

$$z \sim \mathcal{N}(d, \sigma_d) \tag{2}$$

where σ_d is the standard deviation of the depth noise for a true depth d. We train this model using a method from *Nguyen et al.* [32].

Since depth sensors are not ideal, an actual depth reading corresponds to many possible rays through the scene. Therefore, each depth reading represents a *cone*, rather than a ray. The ray passing through the center of the cone closely approximates it near the sensor, but the approximation gets worse further away.

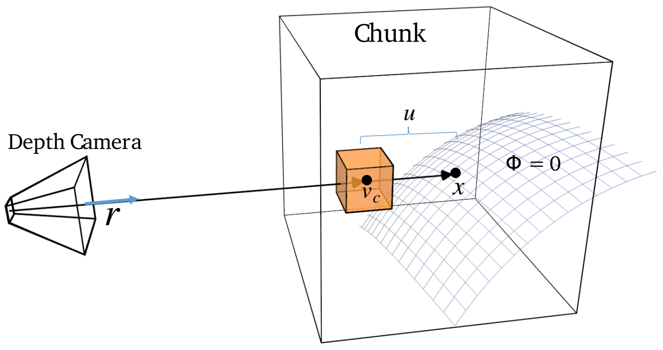
B. The Truncated Signed Distance Field

We model the world [15] as a volumetric signed distance field $\Phi : \mathbf{R}^3 \to \mathbf{R}$. For any point in the world x, $\Phi(x)$ is the distance to the nearest surface, *signed* positive if the point is outside of obstacles and negative otherwise. Thus, the zero isocontour ($\Phi = 0$) encodes the surfaces of the scene.

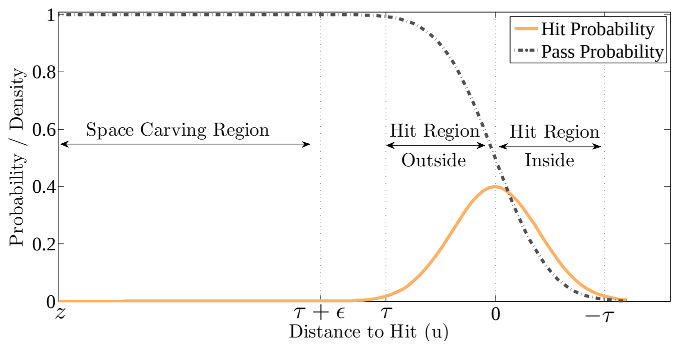
Since we are mainly interested in reconstructing surfaces, we use the Truncated Signed Distance Field (TSDF) [15]:

$$\Phi_\tau(\mathbf{x}) = \begin{cases} \Phi(\mathbf{x}) & \text{if} |\Phi(\mathbf{x})| < \tau \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3}$$

where $\tau \in \mathbf{R}$ is the *truncation distance*. Curless and Levoy [15] note that very near the endpoints of depth rays, the TSDF is closely approximated by the distance along the ray to the nearest observed point.



(a) Raycasting through a voxel.



(b) Noise model and the space carving region.

Fig. 3: Computing the Truncated Signed Distance Field (TSDF).

Algorithm 1 Truncated Signed Distance Field

```
1:  for $t \in [1 \ldots T]$ do                              ▷ For each timestep
2:      for $\{\mathbf{o}_t, \mathbf{x}_t\} \in Z_t$ do               ▷ For each ray in scan
3:          $z \leftarrow \|\mathbf{o}_t - \mathbf{x}_t\|$
4:          $\mathbf{r} \leftarrow \frac{\mathbf{x}_t - \mathbf{o}_t}{z}$
5:          $\tau \leftarrow \mathcal{T}(z)$                  ▷ Dynamic truncation distance
6:          $\mathbf{v}_c \leftarrow \mathbf{x}_t - u\mathbf{r}$
7:          for $u \in [\tau + \epsilon, z]$ do               ▷ Space carving region
8:              if $\Phi_\tau(\mathbf{v}_c) \le 0$ then
9:                  $\Phi_\tau(\mathbf{v}_c) \leftarrow$ undefined
10:                 $W(\mathbf{v}_c) \leftarrow 0$
11:         for $u \in [-\tau, \tau]$ do                      ▷ Hit region
12:             $\Phi_\tau(\mathbf{v}_c) \leftarrow \frac{W(\mathbf{v}_c)\Phi_\tau(\mathbf{v}_c) + \alpha_\tau(u)u}{W(\mathbf{v}_c) + \alpha_\tau(u)}$
13:             $W(\mathbf{v}_c) \leftarrow W(\mathbf{v}_c) + \alpha_\tau(u)$
```

The algorithm for updating the TSDF given a depth scan used in [15] is outlined in Alg.1. For each voxel in the scene, we store a signed distance value, and a weight $W : \mathbf{R}^3 \to \mathbf{R}^+$ representing the confidence in the distance measurement. Curless and Levoy show that by taking a weighted running average of the distance measurements over time, the resulting zero-isosurface of the TSDF minimizes the sum-squared distances to all the ray endpoints.

We initialize the TSDF to an undefined value with a weight of 0, then for each depth scan, we update the weight and TSDF value for all points along each ray within the truncation distance τ. The weight is updated according to the scale-invariant weighting function $\alpha_\tau(u) : [-\tau, \tau] \to \mathbf{R}^+$.

It is possible [32] to directly compute the weighting function α_τ from the hit probability (2) of the sensor; but in favor of better performance, linear, exponential, and constant approximations of α_τ can be used [11, 12, 14, 15]. We use the constant approximation $\alpha_\tau(u) = \frac{1}{2\tau}$. This results in poorer surface reconstruction quality in areas of high noise than methods which more closely approximate the hit probability of the sensor.

C. Colorization

As in [12, 14], we create colored surface reconstructions by directly storing color as volumetric data. Color is updated in exactly the same manner as the TSDF. We assume that each depth ray also corresponds to a color in the RGB space. We step along the ray from $u \in [-\tau, \tau]$, and update the color of each voxel and its weight:

$$C(\mathbf{v}) \leftarrow \frac{W_c(\mathbf{v})C(\mathbf{v}) + \alpha_c(u)\mathbf{c}}{W_c(\mathbf{v})} \tag{4}$$

$$W_c(\mathbf{v}) \leftarrow W_c(\mathbf{v}) + \alpha_c(u) \tag{5}$$

where $\mathbf{v} = \mathbf{x} - u\hat{\mathbf{r}}$, $\mathbf{c} \in \mathbf{R}^{3+}$ is the color of the ray in color space, and α_c is a color weighting function. As in [14], we have chosen RGB color space for the sake of simplicity, at the expense of color consistency with changes in illumination.

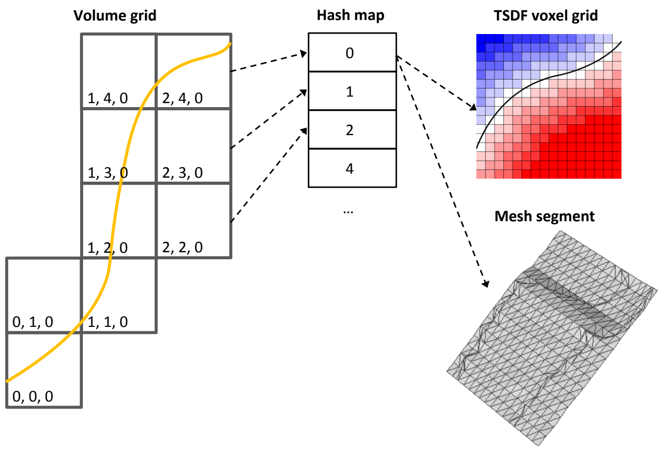
D. Dynamic Truncation Distance

As in[2, 32], we use a dynamic truncation distance based on the noise model of the sensor rather than a fixed truncation distance to account for noisy data far from the sensor. The truncation distance is given as a function of depth, $\mathcal{T}(z) = \beta\sigma_z$, where σ_z is the standard deviation of the noise for a depth reading of z (2), and β is a scaling parameter which represents the number of standard deviations of the noise we are willing to consider. Algorithm 1, line 5 shows how this is used.

E. Space Carving

When depth data is very noisy and sparse, the relative importance of negative data (that is, information about what parts of the scene do not contain surfaces) increases over positive data [6, 23]. Rays can be viewed as *constraints* on possible values of the distance field. Rays passing through empty space constrain the distance field to positive values all along the ray. The distance field is likely to be nonpositive only very near the endpoints of rays.

We augment our TSDF algorithm with a *space carving* [6] constraint. Fig.3b shows a hypothetical plot of the hit probability of a ray $P(z - d = u)$, the pass probability $P(z - d < u)$, vs. u, where z is the measurement from the sensor, and d is the true distance from the sensor to the surface. When the pass probability is much higher than the hit probability in a particular voxel, it is very likely to be unoccupied. The hit integration region is inside $[-\tau, \tau]$, whereas regions closer to the camera than $\tau + \epsilon$ are in the space carving region. Along each ray within the space carving region, we CHISEL away data that has a nonpositive stored SDF. Algorithm 1, line 8 shows how this is accomplished.



(a) spatial hashing.

Fig. 4: Fig.4a: chunks are spatially-hashed [16] into a dynamic hash map.

Voxel Class	Voxel Count	% of Bounding Box
Unknown Culled	62,105,216	77.0
Unknown	12,538,559	15.5
Outside	3,303,795	4.1
Inside	2,708,430	3.4

TABLE I: Voxel statistics for the Freiburg 5m dataset.

Space carving gives us two advantages: first, it dramatically improves the surface reconstruction in areas of very high noise (especially around the edges of objects: see Fig.9), and second, it removes some inconsistencies caused by moving objects and localization errors. For instance, a person walking in front of the sensor will only briefly effect the TSDF before being removed by space carving.

F. The Dynamic Spatially-Hashed TSDF

Each voxel contains an estimate of the signed distance field and an associated weight. In our implementation, these are packed into a single 32-bit integer. The first 16 bits are a fixed-point signed distance value, and the last 16 bits are an unsigned integer weight. Color is similarly stored as a 32 bit integer, with 8 bits per color channel, and an 8 bit weight. A similar method is used in [2, 11, 12, 14] to store the TSDF. As a baseline, we could consider simply storing all the required voxels in a monolithic block of memory. Unfortunately, the amount of memory storage required for a fixed grid of this type grows as $\mathcal{O}(N^3)$, where N is the number of voxels per side of the 3D voxel array. Additionally, if the size of the scene isn't known beforehand, the memory block must be resized.

For large-scale reconstructions, a less memory-intensive and more dynamic approach is needed. Some works have either used octrees [24, 30, 31], or use a moving volume[12]. Neither of these approaches is desirable for our application. Octrees, while maximally memory efficient, have significant drawbacks when it comes to accessing and iterating over the volumetric data [2, 33]. Like [2], we found that using an octree to store the TSDF data to reduce iteration performance by an order of magnitude when compared to a fixed grid.

Instead of using an Octree, moving volume, or a fixed grid, CHISEL uses a hybrid data structure introduced by Nießner *et al.* [2]. We divide the world into a two-level tree. In the first level we have *chunks* of voxels (Fig.4a). Chunks are spatially-hashed [16] into a dynamic hash map. Each chunk consists of a fixed grid of ${N_v}^3$ voxels, which are stored in a monolithic memory block. Chunks are allocated dynamically from a growing pool of heap memory as data is added, and are indexed in a spatial 3D hash map based on their integer coordinates. As in [2, 16] we use the hash function: $\textbf{hash}(x, y, z) = p_1 x \oplus p_2 y \oplus p_3 z \bmod n$, where x, y, z are the 3D integer coordinates of the chunk, p_1, p_2, p_3 are arbitrary large primes, $\oplus$ is the xor operator, and n is the maximum size of the hash map.

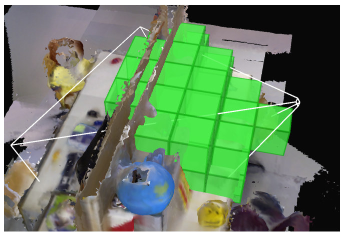


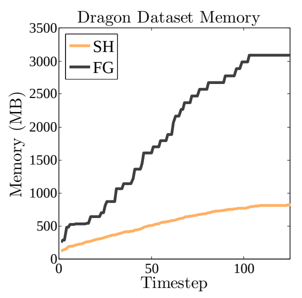
Fig. 6: Only chunks which intersect both intersect the camera frustum and contain sensor data (green) are updated/allocated.

Since chunks are a fixed size, querying data from the chunked TSDF involves rounding (or bit-shifting, if N_v is a power of two) a world coordinate to a chunk and voxel coordinate, and then doing a hash-map lookup followed by an array lookup. Hence, querying is $\mathcal{O}(1)$ [2]. Further, since voxels within chunks are stored adjacent to one another in memory, cache performance is improved while iterating through them. By carefully selecting the size of chunks so that they corresponding to τ, we only allocate volumetric data near the zero isosurface, and do not waste as much memory on empty or unknown voxels.

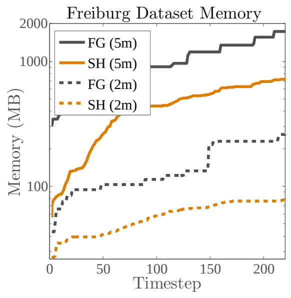
G. Frustum Culling and Garbage Collection

To determine which chunks should be allocated, updated by a depth scan and drawn, we use frustum culling (a well known technique in computer graphics). We create a camera frustum with a far plane at the maximum depth reading of the camera, and the near plane at the minimum depth of the camera. We then take the axis-aligned bounding box of the camera frustum, and check each chunk inside the axis-aligned bounding box for intersection with the camera frustum. Only those which intersect the frustum are updated. Chunks are allocated when they first interesect a camera frustum.

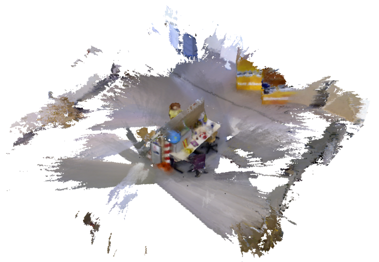
Since the frustum is a conservative approximation of the space that could be updated by a depth scan, some of the chunks that are visible to the frustum will have no depth data associated with them. Fig.6 shows all the TSDF chunks which intersect the depth camera frustum and are within τ of a hit as green cubes. Only these chunks are updated when new data is received. Chunks which do not get updated during a depth scan are garbage collected (deleted from the hash map). Since the size of the camera frustum is in a fixed range, the garbage collection process has $\mathcal{O}(1)$ performance. Niessner *et al.* [2] use a similar approach for deciding which voxel chunks should be updated using raycasting rather than geometric frustum culling. We found frustum culling faster than raycasting on



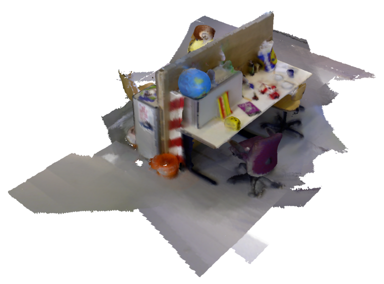
(a) Memory usage efficiency.



(b) Varying depth frustum.



(c) 5m depth frustum.



(d) 2m depth frustum.

Fig. 5: Spatial Hashing (SH) is more memory efficient than Fixed Grid (FG) and provides control over depth frustum.

the mobile CPU.

H. Depth Scan Fusion

To fuse a depth scan into the TSDF (Alg.1), it is necessary to consider each depth reading in each scan and upate each voxel intersecting its visual cone. Most works approximate the depth cone with a ray through the center of the cone [2, 11], and use a fast rasterization algorithm to determine which voxels intersect with the rays [34]. We will call this the *raycasting* approach. Its performance is bouned by $\mathcal{O}(N_{\text{ray}} \times l_{\text{ray}})$, where N_{ray} is the number of rays in a scan, and l_{ray} is proportional to the length of a ray being integrated. If we use a fixed-size truncation distance and do not perform space carving, $l_{\text{ray}} = \tau$. However, with space-carving (Section III-E), the length of a ray being integrated is potentially unbounded.

A useful alternative to raycasting is *projection mapping*. Used in [11, 14, 23, 32], projection mapping works by projecting the visual hull of each voxel onto the depth image and comparing the depth value there with the geometric range from the voxel to the camera plane. Alg.2 describes projection mapping. Instead of iterating over each ray, projection mapping iterates over each voxel. Like most other works, we approximate each voxel's visual hull by a single point at the center of the voxel. Projection mapping's performance is bounded by $\mathcal{O}(N_v)$, where N_v is the number of voxels affected by a depth scan. This value is nearly constant over time, and depending on the resolution of the TSDF, may be significantly less than the number of rays in a scan. However, projection mapping suffers from resolution-dependent aliasing errors, because the distance to the center of a voxel may differ from the true length of the ray passing through the voxel by up to half the length of the diagonal of the voxel. Further, by ignoring the visual hull of the voxel and only projecting its center, we ignore the fact that multiple depth cones may intersect a voxel during each scan. Section IV-C compares the performance and quality of these methods.

I. Rendering

Most other TSDF fusion methods [2, 11] render the scene by directly raytracing it on the GPU. This has the advantage of having rendering time independent of the reconstruction size, but requires TSDf data to be stored on the GPU. Since all of our computation is performed on the CPU, we instead use an approach from computer graphics for rendering large terrains [35] using incremental Marching Cubes [36]. For each chunk, we store a triangle mesh segment (Fig.4a). Meshing is done asynchronously with the depth scan fusion, and is performed lazily (*i.e.*, only when chunks need to be rendered).

Triangle meshes are generated at the zero isosurface of the TSDF whenever the chunk has been updated by a depth scan. As in [12, 14], colors are computed for each vertex by trilinear interpolation. At the borders of chunks, vertices are duplicated. Only those meshes which are visible to the virtual camera frustum are rendered. Meshes that are very far away from the camera are rendered as a colored bounding box. Meshes far away from the user are destroyed once the number of allocated meshes exceeds some threshold. In this way, only a very small number of mesh segments are being updated and/or sent to the GPU in each frame.

Algorithm 2 Projection Mapping

```
1:  for $\mathbf{v}_c \in V$ do                                  ▷ For each voxel
2:      $z \leftarrow \|\mathbf{o}_t - \mathbf{v}_c\|$
3:      $z_p \leftarrow \mathbf{InterpolateDepth}(\mathbf{Project}(\mathbf{v}_c))$
4:      $u \leftarrow z_p - z$
5:      $\tau \leftarrow \mathcal{T}(z_p)$                          ▷ Dynamic truncation
6:      if $u \in [\tau + \epsilon, z_p]$ then                  ▷ Space carving region
7:          if $\Phi_\tau(\mathbf{v}_c) \leq 0$ then
8:              $\Phi_\tau(\mathbf{v}_c) \leftarrow$ undefined
9:              $W(\mathbf{v}_c) \leftarrow 0$
10:     if $u \in [-\tau, \tau]$ then                           ▷ Hit region
11:         $\Phi_\tau(\mathbf{v}_c) \leftarrow \frac{W(\mathbf{v}_c)\Phi_\tau(\mathbf{v}_c)+\alpha(u)u}{W(\mathbf{v}_c)+\alpha(u)}$
12:         $W(\mathbf{v}_c) \leftarrow W(\mathbf{v}_c) + \alpha(u)$
```

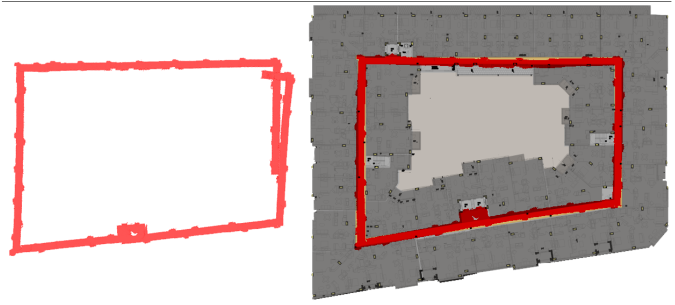


Fig. 7: System pose drift in a long corridor.

J. Online Pose Estimation

Pose data is estimated first from an onboard visual-inertial odometry (VIO) system that fuses data from a wide-angle camera, an inertial measurement unit, and a gyroscope at 60 Hz using 2D feature tracking and an Extended Kalman Filter. A more detailed discussion of this system can be found in [17, 18]. We treat the VIO system as a black box, and do not feed any data back into it. Unlike most other TSDF fusion methods [2, 11, 12, 14] , we are unable to directly estimate the pose of the sensor using the depth data and scan to model iterative closest point (ICP)[20], since the depth data is hardware-limited to only 3-6Hz, which is too slow to directly estimate the pose. Instead, we incrementally adjust the VIO pose estimate by registering each new depth scan with to model built up over time. Given a pose estimate of the sensor at time t, H_t, we can define a cost function of the pose and scan Z_t:

$$c(H_t, Z_t) = \sum_{\mathbf{x}_i \in Z_t} \Phi_\tau(H_t \mathbf{x}_i)^2 \tag{6}$$

this cost function implies that each of the data points in the scan are associated with its nearest point on the surface of the reconstruction Φ_τ (which has distance 0). This is similar to the ICP cost function, where the corresponding point is implicitly described by the signed distance function. Using a first order Taylor approximation of the signed distance field, we can estimate th nearest point on the surface to z_i by looking at the gradient of the signed distance function, and stepping along the direction of the gradient once to directly minimize the distance:

$$m_i \approx -\frac{\nabla\Phi_\tau(H_t z_i)}{\|\nabla\Phi_\tau(H_t z_i)\|}\Phi_\tau(H_t z_i) \tag{7}$$

then, to align the current depth scan with the global model, we use this first-order approximation as the model to register the point cloud against. The gradient of the TSDF is recomputed at each step of ICP using central finite differencing, which has $\mathcal{O}(1)$ performance. We then iterate until c is minimized. Corrective transforms are accumulated as each depth scans arrives.

By registering each new depth scan to the global model, we can correct for small frame-to-frame errors in system drift in a small area. However, as the trajectory becomes longer , the system can still encounter drift. Fig.7 shows a top down view of a $\sim$ 175 meter long corridor reconstructed by a user holding the device. The left figure shows the reconstruction achieved by visual odometry and dense alignment alone, with $\sim$ 5 meter drift present at the end of the corridor. The right figure shows the reconstruction after the trajectory has been globally optimized offline using bundle adjustment, overlaid on the floorplan of the building.

IV. Experiments and Results

A. Hardware

We implemented Chisel on two devices: a *Tango* "Yellowstone" tablet device, and a *Tango* "Peanut" mobile phone device. The phone device has 2GB of RAM, a quadcore CPU, a six-axis gyroscope and accelerometer, a wide-angle

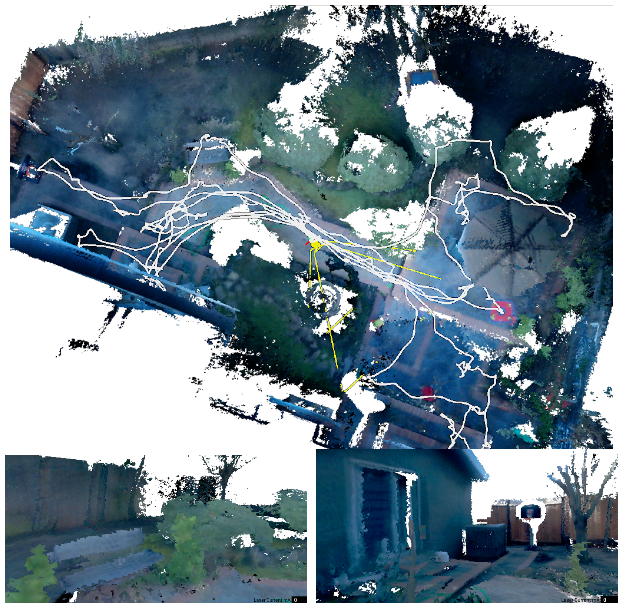
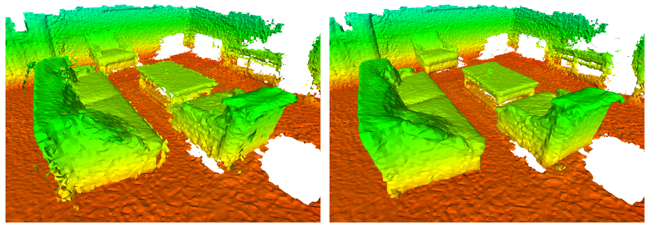
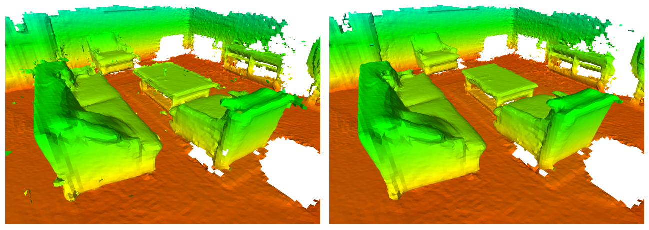


Fig. 8: Outdoor scene reconstruction.



(a) Projection, No Carving (b) Projection, Carving



(c) Raycast, No Carving (d) Raycast, Carving

Method	Color	Carving	Desktop (ms.)	Tablet (ms.)
Raycast	×	×	**14 ± 02**	**62 ± 13**
	✓	×	**20 ± 05**	**80 ± 16**
	×	✓	53 ± 10	184 ± 40
	✓	✓	58 ± 16	200 ± 37
Project	×	×	33 ± 05	106 ± 22
	✓	×	39 ± 05	125 ± 23
	×	✓	**34 ± 04**	**116 ± 19**
	✓	✓	**40 ± 05**	**128 ± 24**

(e) Single scan fusion time.

Fig. 9: Scan fusion methods are compared in the apartment Fig.1b dataset. Fig.9e reports the amount of time required to fuse a single frame for each scan fusion method.

120° field of view tracking camera which refreshes at 60Hz, a projective depth sensor which refreshes at 6Hz, and a 4 megapixel color sensor which refreshes at 30Hz. The tablet device has 4GB of ram, a quadcore CPU, an Nvidia Tegra K1 graphics card, an identical tracking camera to the phone device, a projective depth sensor which refreshes at 3Hz, and a 4 megapixel color sensor which refreshes at 30Hz (Fig.2).

TSDF Method	Meshing Time (ms.)	Update Time (ms.)
256^3 Fixed Grid	2067 ± 679	3769 ± 1279
16^3 Spatial Hashing	102 ± 25	128 ± 24

TABLE II: The time taken per frame in milliseconds to generate meshes (Section III-I), and update the TSDF using colorization, space carving, and projection mapping are shown for the "Room" (Fig.1b) dataset.

B. Use Case: House Scale Online Mapping

Using CHISEL we are able to create and display large scale maps at a resolution as small as 2cm in real-time on board the device. Fig.1a shows a map of an office building floor being reconstructed in real-time using the phone device. This scenario is also shown in a video here [37]. Fig.7 shows a similar reconstruction of a $\sim 175m$ office corridor. Using the tablet device, we have reconstructed (night time) outdoor scenes in real-time.Fig.8 shows an outdoor scene captured at night with the yellowstone device at a 3cm resolution. The yellow pyramid represents the depth camera frustum. The white lines show the trajectory of the device. While mapping, the user has immediate feedback on model completeness, and can pause mapping for live inspection. The system continues localizing the device even while the mapping is paused. After mapping is complete, the user can save the map to disk.

C. Comparing Depth Scan Fusion Algorithms

We implemented both the raycasting and voxel projection modes of depth scan fusion (Section III-H), and compared them in terms of speed and quality. Table 9e shows timing data for different scan insertion methods is shown for the "Room" dataset (Fig.1b) in milliseconds. Raycasting (Alg.1) is compared with projection mapping (Alg.2) on both a desktop machine and a *Tango* tablet. Results are shown with and without space carving (Section III-E) and colorization (Section III-C).The fastest method in each category is shown in bold. We found that projection mapping was slightly more efficient than raycasting when space carving was used, but raycasting was nearly twice as fast when space carving was not used. At a 3cm resolution, projection mapping results undesirable aliasing artifacts; especially on surfaces nearly parallel with the camera's visual axis. The use of space carving drastically reduces noise artifacts, especially around the silhouettes of objects (Fig.9).

D. Memory Usage

Table I shows voxel statistics for the Freiburg 5m (Fig.5c) dataset. Culled voxels are not stored in memory. Unknown voxels have a weight of 0. Inside and Outside voxels have a weight > 0 and an SDF than is ≤ 0 and > 0 respectively. Measuring the amount of space in the bounding box that is culled, stored in chunks as unknown, and stored as known, we found that the vast majority (77%) of space is culled, and of the space that is actually stored in chunks, 67.6% is unknown. This fact drives the memory savings we get from using the spatial hashmap technique from Niessner *et al.* [2].

We compared memory usage statistics of the dynamic spatial hashmap (SH) to a baseline fixed-grid data structure (FG) which allocates a single block of memory to tightly fit the entire volume explored (Fig.5a). As the size of the space explored increases, spatial hashing with $16 \times 16 \times 16$ chunks uses about a tenth as much memory as the fixed grid algorithm. Notice that in Fig.5a, the baseline data structure uses nearly 300MB of RAM whereas the spatial hashing data structure never allocates more than 47MB of RAM for the entire scene, which is a 15 meter long hallway.

We tested the spatial hashing data structure (SH) on the Freiburg RGB-D dataset [3], which contains ground truth pose information from a motion capture system (Fig.5d). In this dataset, a *Kinect* sensor makes a loop around a central desk scene. The room is roughly 12 by 12 meters in area. Memory usage statistics (Fig. 5b) reveal that when all of the depth data is used (including very far away data from the surrounding walls), a baseline fixed grid data structure (FG) would use nearly 2GB of memory at a 2cm resolution, whereas spatial hashing with $16 \times 16 \times 16$ chunks uses only around 700MB. When the depth frustum is cut off at 2 meters (mapping only the desk structure without the surrounding room), spatial hashing uses only 50MB of memory, whereas the baseline data structure would use nearly 300MB. We also found that running marching cubes on a fixed grid rather than incrementally on spatially-hashed chunks (Section III-I) to be prohibitively slow (Table II).

V. LIMITATIONS AND FUTURE WORK

Admittedly, CHISEL's reconstructions are much lower resolution than state-of-the-art TSDF mapping techniques, which typically push for sub-centimeter resolution. In particular, Nießner *et al.* [2] produce $4mm$ resolution maps of comparable or larger size than our own through the use of commodity GPU hardware and a dynamic spatial hash map. Ultimately, as more powerful mobile GPUs become available, reconstructions at these resolutions will become feasible on mobile devices.

CHISEL cannot guarantee global map consistency, and drifts over time. Many previous works [38, 39] have combined sparse keypoint mapping, visual odometry and dense reconstruction to reduce pose drift. Future research must adapt SLAM techniques combining visual inertial odometry, sparse landmark localization and dense 3D reconstruction in a way that is efficient enough to allow real-time relocalization and loop closure on a mobile device.

We have implemented an open-source ROS-based reference implementation of CHISEL intended for desktop platforms (link omitted for double-blind review).

ACKNOWLEDGEMENTS

This work was done as part of Google's Advanced Technologies and Projects division (ATAP) for *Tango*. Thanks to Johnny Lee, Joel Hesch, Esha Nerurkar, and Simon Lynen and other ATAP members for their essential collaboration and support on this project. Thanks to Ryan Hickman for collecting outdoor data.

References

[1] Google. (2014) Project Tango. [Online]. Available: https://www.google.com/atap/projecttango/#project 1, 2

[2] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, 2013. 1, 2, 3, 4, 5, 6, 8

[3] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *IROS*, 2012. 1, 2, 8

[4] Occipital. (2014) Structure Sensor http://structure.io/. [Online]. Available: http://structure.io/ 1

[5] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., 2004. 1

[6] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, 1989. 1, 2, 3, 4

[7] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *ISMAR*, 2007. 1

[8] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy, "Real-time 3D model acquisition," *SIGGRAPH*, 2002. 1, 2

[9] P. Tanskanen and K. Kolev, "Live metric 3d reconstruction on mobile phones," *ICCV*, 2013. 2

[10] T. Weise, T. Wismer, B. Leibe, and L. V. Gool, "Accurate and robust registration for in-hand modeling," in *3DIM*, October 2009. 1, 2

[11] R. Newcombe and A. Davison, "KinectFusion: Real-time dense surface mapping and tracking," *ISMAR*, 2011. 1, 2, 4, 5, 6

[12] T. Whelan and H. Johannsson, "Robust real-time visual odometry for dense RGB-D mapping," *ICRA*, 2013. 1, 2, 4, 5, 6

[13] T. Whelan, M. Kaess, J. Leonard, and J. McDonald, "Deformation-based loop closure for large scale dense RGB-D SLAM," 2013. 1

[14] E. Bylow, J. Sturm, and C. Kerl, "Real-time camera tracking and 3D reconstruction using signed distance functions," *RSS*, 2013. 1, 4, 5, 6

[15] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *SIGGRAPH*, 1996. 1, 2, 3, 4

[16] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," *Proceedings of Vision, Modeling, Visualization*, 2003. 2, 5

[17] D. G. Kottas, J. A. Hesch, S. L. Bowman, and S. I. Roumeliotis, "On the consistency of vision-aided inertial navigation," in *Experimental Robotics*. Springer, 2013. 2, 6

[18] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *ICRA*, 2007. 2, 6

[19] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," 2002. 2

[20] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1992. 2, 7

[21] R. Newcombe, "DTAM: Dense tracking and mapping in real-time," *ICCCV*, 2011. 2

[22] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, 2014. 2

[23] M. Klingensmith, M. Herrmann, and S. S. Srinivasa, "Object Modeling and Recognition from Sparse , Noisy Data via Voxel Depth Carving," 2014. 2, 4, 6

[24] K. Wurm, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," *ICRA*, 2010. 2, 5

[25] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *ICRA*, 2009. 2

[26] N. Vandapel, J. Kuffner, and O. Amidi, "Planning 3-d path networks in unstructured environments." in *ICRA*, 2005. 2

[27] R. Wagner, U. Frese, and B. Bäuml, "3d modeling, distance and gradient computation for motion planning: A direct gpgpu approach," in *ICRA*, 2013. 2

[28] R. Valenti, I. Dryanovski, C. Jaramillo, D. Perea Strom, and J. Xiao, "Autonomous quadrotor flight using onboard rgb-d visual odometry," in *ICRA*, 2014. 2

[29] I. Dryanovski, R. G. Valenti, and J. Xiao, "An open-source navigation system for micro aerial vehicles," *Auton. Robots*, 2013. 2

[30] M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based fusion for realtime 3d reconstruction," *Graph. Models*, 2013. 2, 5

[31] J. Chen, D. Bautembach, and S. Izadi, "Scalable real-time volumetric surface reconstruction," *ACM Trans. Graph.*, 2013. 2, 5

[32] C. Nguyen, S. Izadi, and D. Lovell, *3D Imaging, Modeling*, 2012. 3, 4, 6

[33] T. M. Chilimbi, M. D. Hill, and J. R. Larus, "Making pointer-based data structures cache conscious," *Computer*, pp. 67–74, 2000. 5

[34] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *In Eurographics*, 1987. 5

[35] H. Nguyen, *Gpu Gems 3*, 1st ed. Addison-Wesley Professional, 2007. 6

[36] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM Siggraph*, 1987. 6

[37] http://vimeo.com/117544631. 7

[38] J. Stueckler, A. Gutt, and S. Behnke, "Combining the strengths of sparse interest point and dense image registration for rgb-d odometry," in *ISR/Robotik*, 2014. 8

[39] M. Nießner, A. Dai, and M. Fisher, "Combining inertial navigation and icp for real-time 3d surface reconstruction," 2014. 8