Improved 3D Reconstruction using Combined Weighting Strategies

Patrick Stotko*
Supervised by: Tim Golla†

Institute of Computer Science II - Computer Graphics
University of Bonn
Bonn / Germany

Abstract

Due to the availability of cheap consumer depth sensors and recent advances in graphics hardware, scenes can be reconstructed in real time allowing a wide range of new applications. Current state-of-the-art approaches use a volumetric data structure and integrate recorded scans incrementally to provide complete and accurate reconstructions. However, scans usually contain noise and may be incomplete. Thus, using a simple update procedure becomes impracticable. To overcome these issues, we introduce a new weighting technique which combines different existing strategies. Typical strategies try to model such limitations, like varying visibility and depth-dependent noise, in order to estimate reasonable weights. Since the complexity of modeling grows extremely fast with respect to the number of considered limitations, development becomes complicated and prone to errors. Instead, we consider each limitation separately and construct easy-to-understand solutions for each one. Combining these small strategies leads to a more complex one and results in much higher quality reconstructions.

Keywords: Real-time Reconstruction, Voxel Hashing, GPU, Kinect, Weighting Strategies

1 Introduction

Since 3D models are extremely useful to describe the world, they are widely used in many different areas. Possible applications might be related to online stores. Buying furniture in big shops is often time-consuming and expensive. To save traveling costs and time, clients would like to use an interactive modeling tool for indoor rooms which allows detailed views of the new room from all possible perspectives. Other applications might be in the gaming industry. Modeling detailed real world objects manually is expensive and costs a lot of time and manpower. To reduce costs, already existing objects can be scanned in real time by cheap consumer hardware. In further steps, the given 3D model can be refined manually. Modeling large buildings is also quite expensive, so reconstructions captured by a drone can be very useful and increase work-flows.

For 3D reconstruction, several approaches are developed and many of them use the volumetric data structure by Curless and Levoy [5]. They partition the world into small voxels and store the reconstruction implicitly using a signed distance function (SDF). Thus, each voxel stores its signed distance from the estimated surface with respect to the camera. Since only the immediate region around the surface is actually needed, they also introduced the notion of the truncated signed distance function (TSDF). Instead of storing the exact distance to the surface, distances beyond the user-defined truncation region δ are cut off and only a relative one inside the interval $[-1,1]$ is stored. Under this implicit scheme, scans can be integrated incrementally with a cumulative moving average. However, in terms of noisy data simple averaging is not appropriate since not every data point is equally useful for the reconstruction. So scan points need to be scored by a weight function.

The general problem of developing a weight function is its growing complexity. For each new considered aspect it increases by one dimension. Hence, development becomes error-prone and very slow. To overcome this issue, a strategy to reduce this complexity is needed.

In this paper, we present a new weighting technique which precisely addresses this issue. By separating the weight function into smaller ones, each desired aspect can be solved individually. This leads to easy-to-understand solutions which can be easily compared and discussed. To achieve a complete solution of the problem, we merge them together afterwards. As a result, this technique greatly reduces the problematic complexity and gives the user a powerful tool to customize the weights for his needs.

In the following sections, we first review current state-of-the-art approaches that provide different weighting strategies to solve sensor limitations. Then, we briefly introduce the structure of the 3D reconstruction algorithm used in this paper. Based on this algorithm, we present our new weighting technique and describe how it works with current strategies. Finally, we conclude by comparing them using our benchmark system.

*stotko@cs.uni-bonn.de
†golla@cs.uni-bonn.de

2 Previous work

3D reconstruction has been a very popular research topic in the last decades. Several different approaches were developed including point-based methods, height-map based representations and implicit volumetric approaches. In case of depth maps, one popular approach is the volumetric data structure introduced by Curless and Levoy [5]. It subdivides the world into voxels and stores scans using a signed distance function (SDF). To get a reconstruction, the surface is extracted using ray casting [6] or similar techniques. Their compelling results animated researches to develop several applications on top of this scheme.

One prominent application is KinectFusion [10, 11] which can reconstruct scenes in real time using the equally named Kinect sensor. Since it originally used a regular voxel grid, the space complexity was high and reconstructions were limited to small areas. To solve this drawback, several strategies are developed including moving volume approaches [16, 19, 20], streaming between CPU and GPU [3, 13] or efficient data structures on top of the regular voxel grid [3, 13, 15, 22]. One major improvement was achieved by Nießner et al. [13] who used a hash table to allow fast access to stored data. Voxel blocks, consisting of a set of typically 8^3 voxels, are managed through the hash table and efficiently processed in parallel. This technique achieved frame rates above the 30 Hz frame rate of the Kinect. So we use this algorithm as a base and extend it with our new weighting strategy.

Other developments address the limitations of the depth sensor. Nguyen et al. [12] developed a weight function based on the measured noise of the Kinect sensor and achieved higher quality with much more details. Hemmat et al. [7, 8, 9] also presented a distance based function but only used scan points with higher weights for updating. With this approach, they achieved a similar reduction of the reconstruction error. Functions based on visibility are extensively analyzed by Sturm et al. [18]. Using them allows detailed 360° reconstructions and prevents invalid updates of voxels which causes strong artifacts.

3 Reconstruction algorithm

Since we have implemented the reconstruction algorithm of Nießner et al. [13] in CUDA [14], we start with a brief review of it. First, the given input frame is aligned to the current reconstruction to obtain the new camera pose. After estimation, we transform it into global coordinates and integrate it into the volume. In the last step, the new reconstruction is extracted and used for the next input frame.

3.1 Camera Pose Estimation

Before we can integrate the captured scan, the current pose of the camera has to be estimated. Typical approaches are based on the *Iterative Closest Point* algorithm (ICP) [2, 4]. The idea is to find correspondences between two given point clouds and compute a rigid transformation $T = [R\,|\,t]$ which minimizes the point-to-plane error between the transformed source cloud P and the target cloud Q.

$$E = \sum_{k=1}^{n} \|(T p_k - q_k) \cdot n_k\|_2^2 \tag{1}$$

Hence, finding robust correspondences is essential, so we use the efficient variant by Izadi et al. [10]. Correspondences are found using the given vertex and normal maps V_P, N_P and V_Q, N_Q by projecting each point $p_k \in P$ to image coordinates and choosing the point $q_l \in Q$ which projects to the same coordinates. If the distance between them and the angle between their normals is small, a match is found.

3.2 Integration

After the scan is transformed into global coordinates using the estimated transformation, it can be integrated into the volume. First, we determine all voxel blocks that fall into the current view frustum of the camera and integrate them into the hash table. This is performed by the GPU optimized variant of the *Digital Differential Analyzer* algorithm (DDA) [1] of Xiao et al. [21]. In the next step, we select all visible voxel blocks in the hash table and update them using the weighted cumulative moving average.

$$tsdf_{i+1} = \frac{tsdf_i \cdot w_i + tsdf \cdot w}{w_i + w} \tag{2}$$

$$w_{i+1} = w_i + w \tag{3}$$

Here, $tsdf_i$ and w_i are the old values of the voxel v, and $tsdf$ and w the TSDF value and weight estimated from the new depth map D_i. An appropriate choice for these estimated values is presented later (see section 4). After updating, outliers are removed through a garbage collection to keep the hash table sparse.

3.3 Surface Extraction

In the last step of the algorithm, the new reconstruction has to be extracted from the stored volume. For this task, we use ray casting [6]. First, we initialize the rays using the extrinsic and intrinsic parameters of our camera. Then, we compute the traversal intervals $[t_{start}(x,y), t_{end}(x,y)]$ for each output pixel $p = (x,y)$ by rasterizing all stored voxel blocks and generating two z-buffers.

If all these parameters are known, we sample the volume using *Adaptive sampling* [6]. The base step size t_0 is dynamically reduced to $t_1 = \frac{1}{8} \cdot t_0$ if the current distance to the surface is smaller than a threshold. To find immediately a point before and behind the surface during sampling, we set the step size t_0 to a multiple of the truncation region.

$$t_0 = c_{trav} \cdot \delta \tag{4}$$

This approach works best using values around $\frac{2}{3}$ since the traversal starts at the boundary of the truncation region

where the TSDF values are around ± 1. At the end, we iteratively refine the found intersection and check whether this point is really a desired surface point.

$$|tsdf_{candidate} \cdot \delta| \leq c_{quality} \tag{5}$$

If its absolute distance to the stored reconstruction is smaller than a quality threshold $c_{quality}$, it will be accepted and returned. To get high quality results, typical values of this threshold are around one-hundredth of the voxel size. In a final step, we compose the extracted isosurface to the previous ones to get a full reconstruction. We summarize multiple reconstruction points with a voxel grid filter which has the same size as the infinite regular voxel grid of our volume.

4 Weighting strategies

Choosing appropriate weights is one of the most important parts during reconstruction since some scan points may be corrupted with noise or other limitations of the camera.

4.1 TSDF functions

The TSDF function describes a distance estimation from the reconstructed surface. Because we only search for the position in the volume with value 0 (see subsection 3.3), the real distance value *sdf* of a voxel to the surface is not needed and can be truncated.

KinectFusion [10, 11] provides a very simple TSDF function. It divides the real distance by the truncation region δ and cuts off large values.

$$tsdf_{KinFu}(sdf) = \begin{cases} -1 & \text{if } sdf < -\delta \\ \frac{sdf}{\delta} & \text{if } -\delta \leq sdf \leq \delta \\ 1 & \text{if } sdf > \delta \end{cases} \tag{6}$$

Voxels that are inside the truncation region are valued with truncated distances of the desired interval $[-1,1]$, whereas those which are outside the region are valued with ± 1.

Since this function may cause problems with noisy data, Nguyen et al. [12] introduced a function based on the noise of the Kinect. They modeled the noise along the view direction as a Gaussian distribution with zero mean and standard deviation σ_z which depends on the depth d of the measurement.

$$\sigma_z(d) = 0.0012 + 0.0019 \cdot (d - 0.4)^2 \tag{7}$$

The TSDF function is now given as a cumulative distribution function of the modeled noise distribution.

$$tsdf_{NM}(sdf, d) = \text{sign}(sdf) \sqrt{1 - e^{-\frac{2}{\pi} \frac{sdf^2}{\sigma_z(d)^2}}} \tag{8}$$

In contrast to KinectFusion, voxels near the boundary of the truncation region are also considered as far away.

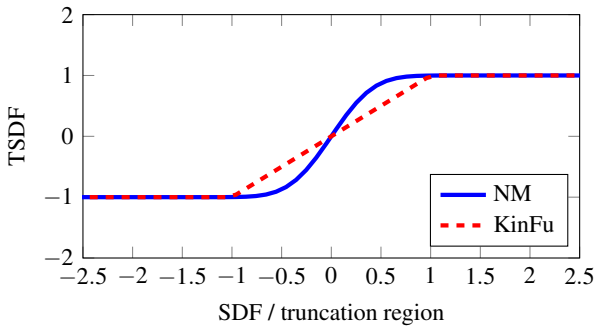


Figure 1: TSDF functions.

Therefore, finer details can be reconstructed since false zero-crossings due to noise are reduced.

For visualization in Figure 1, we fix the distance parameter d to 1.5 (which is the mean range of the Kinect) and use the relation $\delta = 3 \cdot \sigma_z(d)$ proposed by Nguyen et al. [12]. This suppresses the second dimension of the function and allows a visual comparison.

4.2 Weight functions

Weight functions are used to measure the importance of a scan point related to the currently updated voxel. In this paper, three different classes of strategies are discussed: Visibility based functions, depth based functions and angle based functions.

Visibility based functions This class of functions only uses SDF values to compute a weight. The signed distance implicitly encodes visibility information, so this class can be used in 360° reconstructions where a lot of occlusions have to be taken into account. The main drawback of them is depth-dependent noise. This effect can not be modeled here since only the relative distance of the voxel to the surface is known but not the absolute one.

In this context, the function of KinectFusion [10, 11] shows a natural behavior of scoring visibility.

$$w_{KinFu}(sdf) = \begin{cases} 1 & \text{if } sdf < 0 \\ \frac{sdf}{\delta} & \text{falls } 0 \leq sdf \leq \delta \\ 0 & \text{if } sdf > \delta \end{cases} \tag{9}$$

The function distinguishes between three different states. Voxels which lie before the measurement are scored with full weight of 1 indicating that they should be always updated. A similar scoring is performed for voxels lying behind the measurement. They might be part of the back side of the object or part of another one so updating them may cause problems. To prevent this, no update should be performed resulting in weights of value 0. The last state describes voxels inside the truncation region meaning that they lie close to the measured surface. Here, the weight decreases from 1 to 0 to ensure a smooth transition between the two other states.

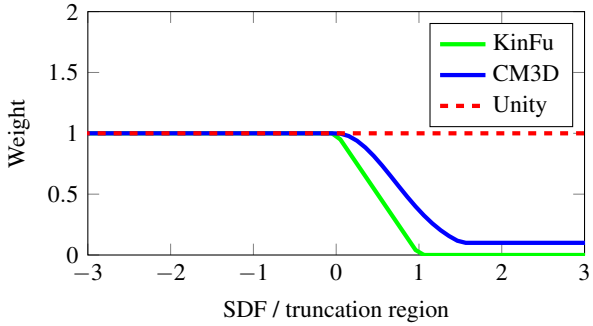


Figure 2: Visibility based weight functions.

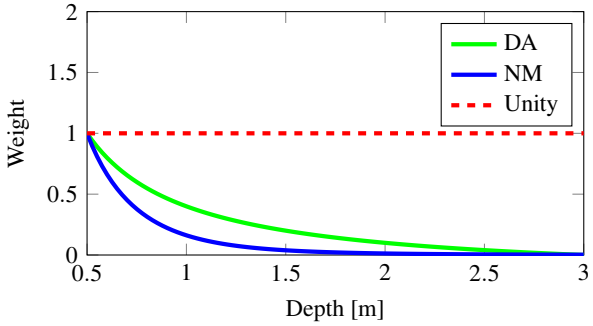


Figure 3: Depth based weight functions.

However, skipping the update of the background can lead to reconstructions having a poor density of points and small holes on the back side (see subsection 5.2). To overcome this, Sturm et al. [18] developed a similar function.

$$w_{CM3D}(sdf) = \begin{cases} 1 & \text{if } sdf < 0 \\ \max\left(w_{min}, e^{-\frac{sdf^2}{\delta^2}}\right) & \text{if } sdf \geq 0 \end{cases} \quad (10)$$

Like KinectFusion, voxels in the foreground are updated with full weight of 1, while those lying inside the truncation region or far behind the measurement are scored with decreasing weights. But instead of a minimum weight of 0, a small positive one is used. This allows the algorithm to update the back side of the reconstructed object, while also keeping the original distance information. In this way, small holes are filled and the voxels can quickly be updated if better measurements become available. A visual comparison of both function is given in Figure 2.

Depth based functions In order to compute a weight, functions of this class use the absolute distance of the measured scan point to the camera. Hence, depth-dependent noise can be modeled here. But in contrast to visibility based functions, 360° reconstructions might be problematic since no occlusion information is provided.

As in their TSDF function, Nguyen et al. [12] use the depth to estimate the noise level first and then computes a reasonable weight.

$$w_{NM}(d) = \frac{\sigma_z(d_{min})}{\sigma_z(d)} \cdot \frac{d_{min}^2}{d^2} \quad (11)$$

$$\sigma_z(d) = 0.0012 + 0.0019 \cdot (d - 0.4)^2 \quad (12)$$

Here, $\sigma_z(d)$ is again the depth-dependent standard deviation of the modeled noise distribution. The weight itself is now computed as the quotient of the minimal noise and depth and the observed noise and depth. This leads to a strong decrease of confidence for voxels being far away from the camera. Additionally, Nguyen et al. [12] also encoded implicitly a visibility based function in their update process and used a 3×3 region around the desired pixel to reduce noise and fill small holes. Since the different classes of weighting strategies are analyzed here, we drop them. The visibility based part is discussed separately and the modified update process using the region around the pixel only reduces noise perpendicular to the view direction of the camera which is not considered here.

A similar function modeling this kind of noise is developed by Hemmat et al. [7, 8, 9].

$$w_{DA}(d) = \frac{\frac{1}{d^2} - \frac{1}{d_{max}^2}}{\frac{1}{d_{min}^2} - \frac{1}{d_{max}^2}} \quad (13)$$

Like Nguyen et al. [12], larger distances are weighted with smaller weights where more noise is expected. The main difference between the two functions is the way how they handle the range interval $[d_{min}, d_{max}]$ of the camera. While Nguyen et al. [12] only use the minimal range as a normalization and ignores the upper bound, Hemmat et al. [7, 8, 9] use both bounds to compute weights between 0 and 1. Especially for larger intervals, the former scores two different high depth values with quite the same small weight, whereas the latter returns small weights which differ much more from each other to indicate the difference of the depth values.

Hemmat et al. [7, 8, 9] also used a modified update behavior. Instead of integrating all captured scans, only those with higher or similar weights than the previous ones are used. The idea here is to suppress measurements with higher noise than previously acquired ones to improve the quality. But as mentioned before, such modifications are not part of the considered class so we skip this step. For a better comparison in Figure 3, we also set the scaling parameter W_{max} in the original function of Hemmat et al. [7, 8, 9] to 1.

Angle based functions Another source of information about the noise level is contained in the angle between the surface normal and the view direction of the camera. Similar to the previously modeled distribution, noise increases significantly if this angle gets larger. Large angles indicate a relatively strong increase of the depth in the neighborhood of the desired pixel meaning that the surface is orientated away from the camera. In this case, measuring

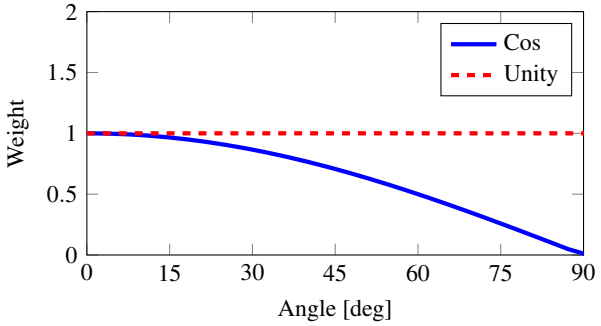


Figure 4: Angle based weight functions.

the correct depth becomes harder or even impossible, so a high noise level is expected here. To model this uncertainty, Curless and Levoy [5] use the cosine of the angle to compute weights.

$$w_{Cos}(\theta) = \cos\theta \tag{14}$$

As seen in Figure 4, high weights are assigned to voxels where the surface is orientated to the camera and the angle is small. With increasing angles, measurements become more and more inaccurate so the weights decrease. Angles around 90° indicate a rapid increase in depth and a high potential error. Measurements can be completely wrong now so no voxel should change its information and weights should be close to 0.

4.3 Combined functions

In the previous section, we introduced some easy to understand functions developed in the last past years. Researchers concentrated on several effects and achieved impressive results. However, these functions can not be easily extended. To overcome this issue, we separate them as described in the previous section by skipping terms that are not related to the desired class. The combination is now performed using a multiplication of the functions each chosen from a unique class.

$$w_{visibility} = f_{visibility}(sdf) \tag{15}$$
$$w_{depth} = f_{depth}(d) \tag{16}$$
$$w_{angle} = f_{angle}(\theta) \tag{17}$$
$$w_{combined} = w_{visibility} \cdot w_{depth} \cdot w_{angle} \tag{18}$$

One important aspect of a weight function is its bounds. We assume that all used functions are non-negative and bounded by some individual small constant.

$$\forall sdf\colon 0 \leq w_{visibility}(sdf) \leq W_{max\ visibility} \tag{19}$$
$$\forall d\colon 0 \leq w_{depth}(d) \leq W_{max\ depth} \tag{20}$$
$$\forall \theta\colon 0 \leq w_{angle}(\theta) \leq W_{max\ angle} \tag{21}$$

Since the combined values are bounded by the product of the individual bounds, these constants should be small. This ensures that the combined function never reaches extremely large values. In our case, the combined bound is given by

$$W_{max\ combined} = W_{max\ visibility} \cdot W_{max\ depth} \cdot W_{max\ angle} \tag{22}$$

Ideally, all of them are set to 1 resulting in a combined bound of 1. Because large weights might be problematic in some implementations, future extensions like additional classes can now be integrated easily and the update behavior of the algorithm still remains as expected.

Naturally, also more than three strategies can be combined to compute weights. But using two functions of the same class does not necessarily increase the complexity of the entire strategy. The combination of these two functions is still a function of the same strategy class and is limited to its properties. So it could be defined directly without using this approach. However, this also can increase readability and lead to a finer grading of the modeled limitations.

5 Evaluation

In this part, we discuss the results achieved by our new technique and compare it with current approaches.

5.1 Test Environment

To allow applying the introduced algorithm and test the weight functions with existent 3D models, we need to compute depth maps on-the-fly. For construction, a virtual camera consisting of extrinsic and intrinsic parameters has to be defined first. The extrinsic parameters are also known as the pose and are already computed during pose estimation. The intrinsic parameters contain the camera resolution, its range and field of view. With these parameters, we compute the projection matrix introduced by Zhang [23].

$$P = \begin{pmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{23}$$

The skewness parameter γ is dropped here since we only consider a virtual camera. The remaining coefficients can be expressed in terms of the camera resolution $n_x \times n_y$ and the horizontal field of view $fov_{horizontal}$.

$$\alpha = f \cdot u_0 = \beta \qquad f = \frac{1}{\tan\left(\frac{fov_{horizontal}}{2}\right)} \tag{24}$$

$$u_0 = \frac{n_x}{2} \qquad v_0 = \frac{n_y}{2} \tag{25}$$

To construct the depth map now, we first transform the given point cloud to the local camera coordinate system using the estimated pose. Then, we compute the distances to the camera and project all points by the projection matrix P. Points falling into the same pixel are summarized by the one with minimal distance to the camera. Finally, we store these distances in an image to obtain the final depth map.

		W_{single}						$W_{combined}$			
		Unity	KinFu	CM3D	NM	DA	Cos	KinFu + NM + Cos	KinFu + DA + Cos	CM3D + NM + Cos	CM3D + DA + Cos
Dragon	$TSDF_{KinFu}$	3.7052	1.9951	2.4717	3.0804	2.9989	3.3534	1.8656	1.8429	2.2461	2.2374
	$TSDF_{NM}$	3.3918	1.8711	2.4328	2.6763	2.5801	2.9604	1.7657	1.7434	2.2153	2.2042
Angel	$TSDF_{KinFu}$	2.9870	1.1662	1.5912	2.8978	2.9313	2.7380	1.1271	1.1290	1.5045	1.4977
	$TSDF_{NM}$	2.6760	1.1051	1.5117	2.5358	2.5328	2.4360	1.0835	1.0848	1.4385	1.4322

Table 1: Comparison of the mean error (ME) between different TSDF and weight functions (in mm).

		W_{single}						$W_{combined}$			
		Unity	KinFu	CM3D	NM	DA	Cos	KinFu + NM + Cos	KinFu + DA + Cos	CM3D + NM + Cos	CM3D + DA + Cos
Dragon	$TSDF_{KinFu}$	5.6066	2.9967	3.3495	4.6405	4.5039	5.1886	2.7912	2.7653	3.2000	3.2545
	$TSDF_{NM}$	5.2296	2.9073	3.3482	4.1469	3.9830	4.6493	2.7319	2.7061	3.1820	3.2213
Angel	$TSDF_{KinFu}$	4.5709	1.8169	2.1430	4.3310	4.3838	4.2221	1.7622	1.7800	2.0666	2.0574
	$TSDF_{NM}$	4.1803	1.7674	2.0573	3.8763	3.8563	3.8216	1.7310	1.7415	2.0097	1.9961

Table 2: Comparison of the root-mean-square error (RMSE) between different TSDF and weight functions (in mm).

For testing, we use the 3D models of the Asian Dragon and the Christian angel Lucy provided by the Stanford Computer Graphics Laboratory [17]. Reconstruction is performed on a Intel Core i7-4930K CPU, 32 GiB RAM and a Nvidia GeForce GTX780 with 3 GiB VRAM. Our implementation uses a hash table with 2^{20} buckets each containing 2 entries. The reconstruction is stored in a predefined voxel buffer with a total number of 2^{18} voxel blocks and 8^3 voxels per block. We use a voxel size of 1 mm and a truncation region of 12 mm. Our virtual camera has a range from 1.25 m to 2.25 m and captures depth maps using a resolution of 1920×1080 pixels with a horizontal field of view of 60°. Since the angel stands upright, the camera is rotated in this kind of situation and uses a resolution of 1080×1920 pixels with a vertical field of view of 60°. Additionally, we add artificial noise to each depth sample according to the previously mentioned Gaussian noise distribution with depth-dependent standard deviation $\sigma_z(d)$.

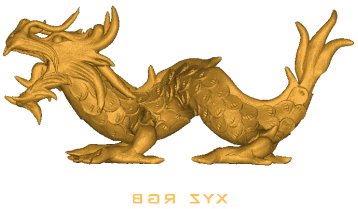
For reconstruction, all models are placed 1.75 m in front of the camera and scaled such that the heights of their bounding boxes are equal to 75% of the vertical height of the frustum at this depth. This ensures that the depth maps contains the full object independently of its original location in space.

5.2 Results

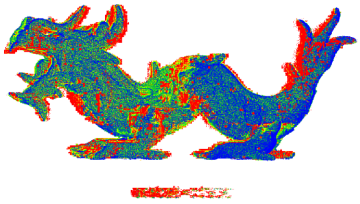
Reconstruction is performed by a single 360° round of the camera with 360 depth maps captured in total. Averaged reconstruction times per frame are 231.6 ms ($\sim$4 fps), with 17.7 ms (7.6%) for depth map creation, 89.8 ms (38.8%) for integration, 100.2 ms (43.3%) for surface extraction and 23.9 ms (10.3%) for surface composting. Since the used resolution is much higher than the one of the Kinect, this demonstrates the scalability of the volumetric data structure of Curless and Levoy [5].

Tests are performed among all possible combinations of TSDF and weight functions and shown in Table 1 and Table 2. More precisely, we test the unity function, the visibility based functions KinFu (Izadi et al. [10], Newcombe et al. [11]) and CM3D (Sturm et al. [18]), the depth based functions NM (Nguyen et al. [12]) and DA (Hemmat et al. [7, 8, 9]), and the angle based function Cos (Curless and Levoy [5]) as defined before. These functions only compute a single weight and are the reference of our technique. As the combined weights, we use all possible combinations consisting of three different functions each chosen from a unique class.

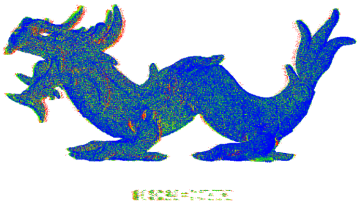
Because we consider a 360° reconstruction, visibility based functions perform best. KinectFusion seems to achieve the best result since it has the lowest error. However, the point density of its reconstruction is highly irregular. While there is a high density at the front side, the back side only has a low one resulting in a low total number of samples and many small holes. This is caused by using zero weight on the back side which means that no update is performed and relating voxel blocks are deleted by the garbage collection. Sturm et al. [18] use a small positive weight to overcome this and achieve very good results but with a slightly higher error. Depth based and angle based functions perform worse which is shown especially by the root-mean-square error that penalizes non-regular recon-



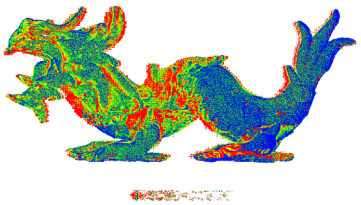
(a) Original with 3 609 600 points.



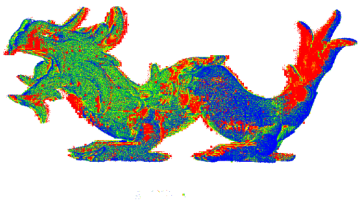
(b) Error heat map of the reconstruction with 4 883 046 points using the functions $TSDF_{KinFu}$ and W_{Unity}.



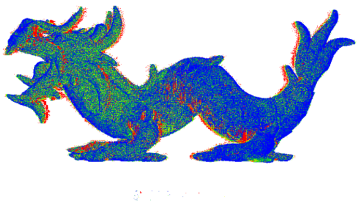
(c) Error heat map of the reconstruction with 6 130 290 points using the functions $TSDF_{KinFu}$ and W_{CM3D}.



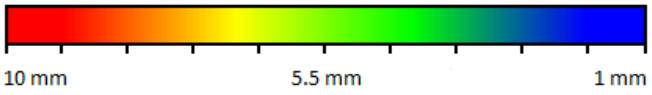
(d) Error heat map of the reconstruction with 4 819 152 points using the functions $TSDF_{KinFu}$ and W_{DA}.



(e) Error heat map of the reconstruction with 4 421 626 points using the functions $TSDF_{KinFu}$ and W_{Cos}.



(f) Error heat map of the reconstruction with 5 684 977 points using the functions $TSDF_{KinFu}$ and $W_{CM3D+DA+Cos}$.



(g) Error heat map legend.

Figure 5: Asian dragon, used from Stanford Computer Graphics Laboratory [17].

structions. Furthermore, reconstructions are affected by many artifacts and only have a slightly better quality than the unity function (see Figure 5).

Best results among all test scenes are achieved by our new weighting technique. All combinations outperform the strategies which only use a single weight and achieve improvements up to 10%. Especially the combination with the function of Sturm et al. [18] performs best and achieves a lower reconstruction error than current approaches. However, not all regions of the reconstruction have a lower error. As shown in Figure 5, the depth based and angle based functions produce samples with high error at the head of the dragon while the visibility based ones creates most samples with low error. As a result, the negative behavior of a function is also integrated in the overall strategy and can increase the error of some samples. Nevertheless, the positive and intended behavior of such a function overweights its risks, so the averaged error over all samples is reduced.

6 Conclusion

We presented a new weighting technique which combines existing strategies and assigns them to a certain class that represents its properties. These classes can extend the knowledge of the underlying problem and accelerate development of more sophisticated strategies. For a complete solution that captures all desired limitations, several functions are combined, each taken from a unique class.

We demonstrated high quality reconstructions with a smaller error than current state-of-the-art approaches. We believe that the advantages of combined functions are even more evident when more classes are developed and the complexity increases. However, combining arbitrary functions which might lower the error does not necessarily lead to better results. The functions we used are developed carefully and proven to perform good, so the combination of functions should also be chosen very carefully to achieve good results.

References

[1] J. Amanatides and A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *In Eurographics '87*, pages 3–10, 1987.

[2] P. J. Besl and N. D. McKay. A Method for Registration of 3-D Shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[3] J. Chen, D. Bautembach, and S. Izadi. Scalable Real-time Volumetric Surface Reconstruction. *ACM Trans. Graph.*, 32:113:1–113:16, 2013.

[4] Y. Chen and G. Medioni. Object Modelling by Registration of Multiple Range Images. *Image Vision Computing (IVC)*, 10(3):145–155, 1992.

[5] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 303–312, 1996.

[6] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. In *Computer Graphics Forum*, volume 24, pages 303–312, 2005.

[7] H. J. Hemmat, E. Bondarev, and P. H. N. de With. Exploring Distance-Aware Weighting Strategies for Accurate Reconstruction of Voxel-Based 3D Synthetic Models. In *MultiMedia Modeling - 20th Anniversary International Conference, MMM 2014, Dublin, Ireland, January 6-10, 2014, Proceedings, Part I*, pages 412–423, 2014.

[8] H. J. Hemmat, E. Bondarev, G. Dubbelman, and P. H. N. de With. Improved ICP-based Pose Estimation by Distance-aware 3D Mapping. In *VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications, Volume 3, Lisbon, Portugal, 5-8 January, 2014*, pages 360–367, 2014.

[9] H. J. Hemmat, E. Bondarev, G. Dubbelman, and P. H. N. de With. Evaluation of Distance-Aware KinFu Algorithm for Stereo Outdoor Data. In *VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications, Volume 2, Lisbon, Portugal, 5-8 January, 2014*, pages 746–751, 2014.

[10] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 559–568, 2011.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *IEEE ISMAR*, 2011.

[12] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking. In *3DIMPVT*, pages 524–530, 2012.

[13] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11, 2013.

[14] Nvidia. CUDA. `http://www.nvidia.com/object/cuda_home_new.html`, 2007. Accessed March 31, 2015.

[15] F. Reichl, M. G. Chajdas, K. Bürger, and R. Westermann. Hybrid Sample-based Surface Rendering. In *Vision, Modeling and Visualization*, pages 47–54, 2012.

[16] H. Roth and M. Vona. Moving Volume KinectFusion. In *Proceedings of the British Machine Vision Conference*, pages 112.1–112.11, 2012.

[17] Stanford Computer Graphics Laboratory. The Stanford 3D Scanning Repository. `http://graphics.stanford.edu/data/3Dscanrep/`, 1996. Accessed March 31, 2015.

[18] J. Sturm, E. Bylow, F. Kahl, and D. Cremers. CopyMe3D: Scanning and Printing Persons in 3D. In *German Conference on Pattern Recognition*, 2013.

[19] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald. Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous. Technical report, Computer Science and Artificial Intelligence Laboratory, MIT, 2012.

[20] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[21] K. Xiao, D. Z. Chen, X. S. Hu, and B. Zhou. Efficient implementation of the 3D-DDA ray traversal algorithm on GPU and its application in radiation dose calculation. *Medical Physics*, 39(12):7619–7625, 2012.

[22] M. Zeng, F. Zhao, J. Zheng, and X. Liu. A Memory-Efficient Kinectfusion Using Octree. In *Proceedings of the First International Conference on Computational Visual Media*, pages 234–241, 2012.

[23] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22 (11):1330–1334, 2000.