**Faster and Simpler KinectFusion with Sorted Voxel Clouds**

**SVCs: Introduction**

A SVC is a sparse spatial data structure that maps particularly well to GPUs. It offers amortized constant cost for insertion, deletion and update when processing many elements in batches. Retrieval is logarithmic in the worst case.

The key idea is to quantize the xyz-world coordinates of the points in a point cloud, derive an integer key from them (for example the morton code, obtained by shuffling the bits) and sort them according to this key.

This is how one would implement insertion, deletion and update:

1. Assume the SVC is sorted and contains *N* elements.
2. Create a *change list* of *M* key-value pairs: the key matches the key in the SVC one wants to change, the value encapsulates the action (insert/delete/update) one wants to perform.
3. Sort the *change list* => O(*M*)
4. Merge the *change list* and the SVC like mergesort would do (when encountering the same key in both lists, the associated value in the *change list* tells one how to resolve the conflict (keep both/neither/only one or somehow merge the two values in a different way)) => O(*N* + *M*)

The result is a SVC with zero to *N* + *M* elements. If *M* is sufficiently large (in the order of magnitude of *N*), this results in amortized constant costs per element. The sorted property guarantees logarithmic retrieval.

Certain applications’ access patterns – like 3D surface reconstruction – allow batching/staging operations in a way that makes SVCs efficient.

By relying on sorting as the single primitve which makes this data structure work, the resulting implementation benefits two-fold:

1. The flat nature of the involved data structures (lists) and the ability to use library functions for implementing the core operations makes it very simple.
2. It scales trivially with every algorithmic improvement to sorting.

**SVCs: Application in 3D surface reconstrucion**

3D surface reconstruction consists of four main steps:

1. *Align* point clouds and compute new camera pose.
2. *Stream* data between memory systems.
3. *Integrate* new data
4. *Extract* surface

*Stream, Integrate*

Streaming and integration are easy to implement. Streaming reads and writes 3D blocks from and to the active buffer. Since the elements are layed out in morton order, start and end indices of 3D blocks can be computed with binary search. Streaming then becomes a linear copy.

Integration both updates existing voxels and adds new ones. This can be implemented by constructing a change list from the raw depth map and applying a merge step.

*Align, Extract*

These will be the big challenges of the paper. Not only are both of these operations non-trivial to implement, they are also dependant on one another.

Surface extraction can be implemented via raycasting or splatting. Raycasting seems prohibitive on SVCs since it barely reaches realtime performance on regular grids, which offer constant time random access. SVCs on the other hand offer sub-logarithmic random access in the best case, making them a factor slower than regular grids. (The fact that the keys implicitly encode a hierarchy allows one to implement ray-skipping, this might help a bit).

Splatting is generally more expensive than raycasting and its performance depends heavily on the desired quality but might map better to SVCs (as one simply traverses all points without the need for binary search). If the extracted surface should be used for aligning, it has to have a high quality (KF uses this trick: it projects the raw depth map into the coordinate system of the synthetic one and associates points with the same texel indices – avoiding more expensive searches (like k-nearest neighbors)).

Either way, noise has to be dealt with. Possible scenarios are:

* Directly splat the raw depth map into the volume and handle noise purely during surface extraction, for example by averaging points that project to the same pixel.
* Integrate all voxels inside a depth samples footprint. This aids surface extraction but decreases the sparseness of the SVC, making operations on it more expensive.
* Only splat the raw depth map into the volume, but run a filter (every (n-th?) frame) over the volume to remove noise (for example build a histogram and cut off low occurences or derive the surface defined by the point cloud) before extracting the surface (with then potentially simple means).

(Note: During tests conducted for the scalable fusion paper we found that a depth sample foot print of ~15cm converged in appearance to integrating the whole volume.)

Aligning can be solved via ICP either on the extracted and raw surfaces, which is very fast but requries the extracted surface to be high quality, or direclty on the point clouds which might make point association more expensive. Ideally, we find a fast way to perform high quality ICP on a noisy, sparse volume and a noisy depth map. This would make integration cheap (no footprint integration, no filters/etc.) and allow us to use surface extraction purely for user feedback, simply consuming the remaining frame time budget and therefore rendering at a device’s capabilities.

**Tasks**

1. Research combinations of integration and rendering techniques. Integrate noisy data either as ponits (possibly augmented with orientations and radii) or SDFs and extract surfaces via raycasting, rasterization, splatting or variations thereof.

(QSplat rendered scenes of comparable complexity to what we will be dealing with in sufficient quality in ~700ms. GPU performance increased by a factor of ~100x since 2000. Implement QSplat or a derivative and see how it performs today.)

1. Research aligning techniques, specifically ICP on a raw depth map and a noisy volume either filled with points (potentially augmented) or SDFs.