# Compressing Aerodynamic Hazard Data

Y. Zhou [1]†, N. Boullé [2], D. A. W. Barton [3], E. Campillo-Funollet [4], and C. L. Hall [5]

[1] *Department of Mathematical Sciences, University of Bath, Bath BA2 7AY, UK*
[2] *Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK*
[3] *Department of Engineering Mathematics, University of Bristol, Bristol BS8 1TW, UK*
[4] *Carr Lab, School of Life Sciences, University of Sussex, Brighton BN1 9QG, UK*
[5] *Department of Engineering Mathematics, University of Bristol, Bristol BS8 1TW, UK*

(*Communicated to* MIIR *on 14 December 2021*)

**Study Group:** ESGI 162, Leeds, July, 2020.

**Communicated by:** J. A. Ward, School of Mathematics, University of Leeds.

**Industrial Partner:** Zenotech

**Presenter:** David Standingford

**Team Members:**
D. A. W. Barton, University of Bristol;
N. Boullé, University of Oxford;
E. Campillo-Funollet, University of Sussex;
C. L. Hall, University of Bristol;
H. Nguyen, WIAS Berlin;
S. Ruangdech, University of Bristol;
Y. Zhou, University of Bath.

**Industrial Sector:** Data compression, CFD simulation compression, Drones, mobile data accessing

**Tools:** Python, Julia

**Key Words:** Data compression, computer graphics, octree, geometric primitives

**MSC2020 Codes:** 65D18, 68W25

† Corresponding Author: yz3259@bath.ac.uk

**Summary**

Data compression of three-dimensional computational fluid dynamics (CFD) simulation data is crucial to allow effective data-streaming for drone navigation and control. This problem is computationally challenging due to the complexity of the geometrical features present in the CFD data, and cannot be tackled by standard compression techniques such as sphere-tree. In this report, we present two different methods based on octree and cuboid primitives to compress velocity isosurfaces and volumetric data in three dimensions. Our volume compression method achieves a 1400 compression rate of raw simulation data and allows parallel computing.

## 1 Introduction

In order to provide safe flying paths for small unmanned aerial vehicles (UAVs), Zenotech stores information about the urban aerodynamic environment. The UAVs need to access these data dynamically in order to update their flying trajectories, avoiding hazardous zones.

It is simplest form, the data is stored in VTK format, as volumes, storing the value of the quantities of interest. For example, wind speed is what of the hazard factors. The VTK file is the result of a computational fluid dynamics (CFD) simulation, providing the wind speed at every point in space. Figure 1 depicts the result of one of this simulations on the London city area.

Since the final application of the data is to avoid entering hazardous zones, e.g. areas with strong winds, Zenotech already preprocessed the data to compute the isosurfaces of the values of interest. In principle, this reduces the size of the data, from three-dimensional (3D) volumes to two-dimensional (2D) surfaces on a 3D environment, and still contains enough information to design a safe flying trajectory, by avoiding crossing the isosurfaces into hazardous zones.

Zenotech is interested in algorithms which can compress the hazardous zones data, since the bandwidth, computation power and storage capacities of the UAVs are limited. Therefore, the goal of the Study Group is to explore different approaches to compress the aerodynamic environment data, balancing the compression rate and the computational power to calculate collisions with boundaries of hazardous zones.

### 1.1 Approaches explored in this report

The problem of computing collisions is widely considered in the film and game industries [5, 6]. These methods are usually very efficient, although they might have specific computational requirements (e.g. GPUs). Furthermore, the main goal in film and game applications is to obtain a good visualisation. In consequence, the exact error bounds are usually not considered. For the purpose of this Study Group, control on the error introduced by the compression process is critical. We explore out-of-the-box methods from computers graphics in Section 2.

Another intuitive approach to the problem is to use an octree [8] to store the infor-

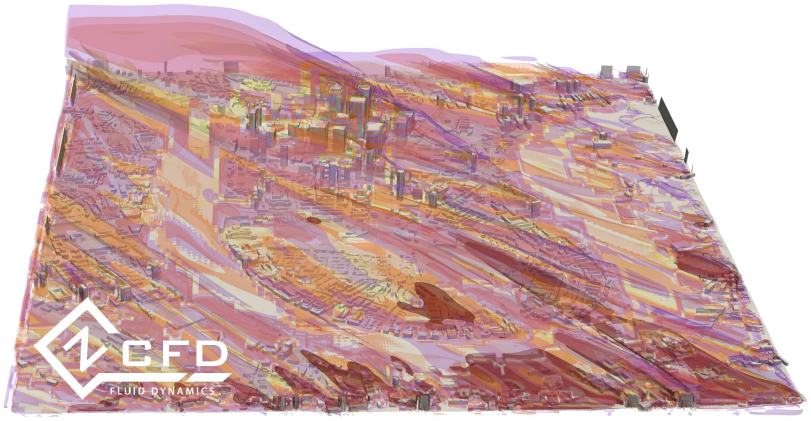


Figure 1. CFD simulation of the London city center

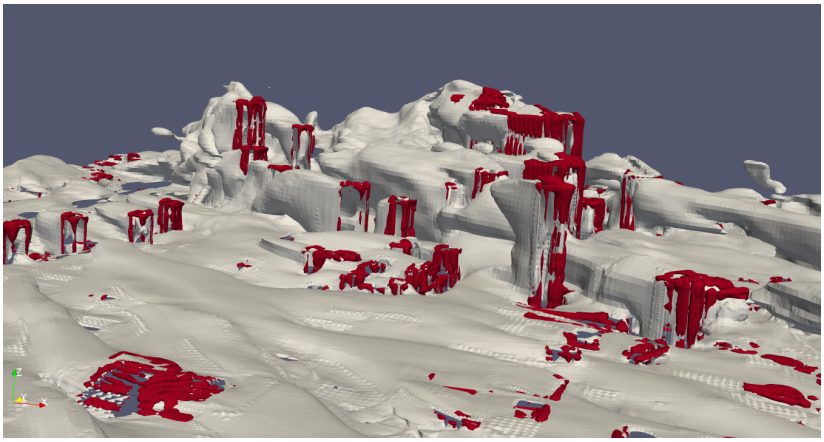

Figure 2. Example of data set containing wind speed and shear

mation about a sequence of volume elements. Although these structures do not provide computational efficiency, they have high compression rates and Zenotech suggested that they might still be of interest. We explore the octree method in Section 3.

Finally, we explore in Section 4 an algorithmic compression method, based on an *ad-hoc* description of cubic elements in terms of cuboids. In simple terms, the method seeks to merge adjoining cubes into larger cuboids. The approach conserves the spatial resolution of the original representation, and describes the data in a format that can be readily used to compute collisions, without decompressing the data first.

### 1.2 Other approaches

We discussed several approaches during the Study Group that we did not finally explore. First, the approach presented in Section 4 seems amenable to further analysis, in particular to understand the algorithmic complexity of the method and to obtain bounds on the compression rate. The problem can be described in terms of minimal coverings

of coloured graphs, but we were not able to find any reference studying this particular problem.

The problem of representing the surface of a point cloud can be explored from an optimisation perspective. For instance, [2] uses the bees algorithm to fit primitive shapes to point clouds; similarly [6] solves the same problem by means of machine learning methods. In both cases, the resulting surfaces provide a compressed form of the data once described in terms of the primitive shapes. [9] is a patented algorithms to compress point data with error constraints. [10] presents a method based on the sparse representation of surfaces that is able to represent non-smooth features. These methods can be of interest, but further work is required to assess their compression rates for the aerodynamic environment data, as well as the algorithms to use the compressed data to find collisions.

## 2 Hierarchical bounding volumes

The problem of representing a closed surface with an appropriate bounding volume is very important in computer graphics (especially with gaming applications) for collision detection. Since the underlying application in this project is also collision detection, it seems likely that some of the techniques developed in computer games will also be useful.

A lot of the techniques for describing objects with collision-detection applications take a hierarchical approach. Essentially, there is a very coarse description of the object (as a single bounding sphere, for example), but then there are subsequent levels of refinement in which the details of the object can be represented using a larger number of smaller primitive volumes. The Octree method described in section 3 is an example of this sort of algorithm, but it is not alone.

One frequently-referenced algorithm for computing hierarchies of bounding volumes for collision detection is the sphere-tree algorithm developed by Bradshaw [3] with code publicly available at [4]. This method represents any bounded surface by a hierarchy of sets of spheres. The most basic representation will use a small number of spheres and will not be particularly refined; further along the hierarchy, the representation will use a larger number of spheres and will be very close to the original surface.

The original sphere-tree idea and many of its extensions depend on identifying the "skeleton" of the object and using this as an anchor for constructing a hierarchy of approximations. This idea of identifying the skeleton leads to the "medial axis" method in [3], which can then be supplemented by various techniques for ensuring that the sphere representation is as efficient as possible. There are various extensions of the original sphere-tree algorithm; some of these involve improved methods for computing the medial axis "skeleton" [5], while others extend the method to use ellipsoids and other primitive volumes [7].

Since the sphere-tree algorithms can be applied "out-of-the-box" using the software from [4], we attempted to apply the method to the given data. The results are shown in Figure 3.

Figure 3 clearly indicates that the sphere-tree algorithm has not been very successful at identifying and compressing the surface data. On reflection, this result should not be particularly surprising. The sphere-tree method works best for relatively compact shapes with a clearly-identifiable "skeleton". However, the surface that we are trying to

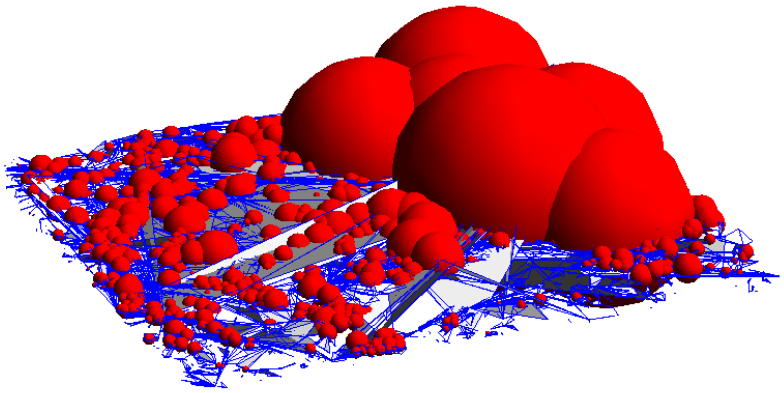

Figure 3. Results of naively applying the sphere tree algorithm available from [4] to the surface data

represent can be thought of as the boundary of an object that is mostly long and narrow with occasional "lumps" close to the buildings. This makes the choice of a sphere-tree particularly inappropriate — it is like trying to represent the volume of a sheet of paper using a set of spheres; a large number of spheres would be required to achieve any accuracy at all because of the simple fact that a sheet of paper is very different in shape from a sphere. An additional problem is that the surface that we are trying to represent may consist of multiple disconnected parts, which the sphere tree algorithm may struggle to represent accurately.

While the sphere-tree algorithm is inappropriate in this case, the literature search conducted during the study group indicates that there has been a lot of activity on using hierarchies of bounding volumes to represent different objects in computer graphics for collision detection. It is quite possible that there are approaches in the literature using different hierarchies of bounding volumes that would be appropriate for surfaces like those in this problem, which bound volumes with high aspect ratios. However, the high aspect ratios also suggest that it may be more appropriate to use different approaches, such as the cuboid approach described in Section 4, which attempt to compress volumes rather than surfaces and that exploit the "blockiness" of the hazard surfaces.

## 3 Octree method for compressing isosurfaces

Tree-like structures have been used extensively for many decades in computer graphics to speed up computations in 3D environments. Most notably, the first-person 3D shooter, Doom, made extensive use of binary-space-partition trees (BSP trees) to enable real-time (pseudo-) 3D computation on (by modern standards) very underpowered hardware. Motivated by this, the use of tree structures was investigated for storage and computation of the isosurfaces provided by Zenotech.

It should be noted that these tree structures are not directly amenable to parallelised

implementation (e.g., on a GPU), however, Zenotech noted that the primary constraints are in the communications channel and so this might not be an issue.

### 3.1 Description of the method

The method presented here provides a guaranteed level of accuracy that can be tuned according to the needs of the user. There are three steps to the method.

(1) Convert the isosurface, represented by a triangular mesh, into a structured grid of cubes such that every triangle is fully enclosed by cubes of a specified size (e.g., $1\,\mathrm{m}^3$). The grid is structured such that no cubes overlap but that adjacency cubes abut. This approach ensures a fixed level of accuracy.

- Further optimisations could be performed at this stage, for example, only retaining cubes that enclose a pre-specified area of triangles such that small intrusions are discarded. We do not consider such optimisations here.

(2) A hierarchical structured grid is created by merging together adjacent cubes to form cubes of double the size (eight times the volume). Again, this is done in such a way as the larger cubes do not overlap.

- Alternative ways of merging adjacent cubes together could be explored in order to exploit known structure in the isosurfaces. For example, if it is known that there is a tendency for strands to form in a particular direction (that is cubes forming a long, slender structure) then merging could be performed preferentially in that direction.

(3) A binary code representing the tree structure is created from the hierarchical structured grids.

We do not consider the details of 1. in the description above other than to note that a fast AABB-triangle intersection code is required[1]. By iterating over all triangles in the isosurface mesh, this step is trivially parallelisable. Similarly, 2. above is straightforward to implement, though memory intensive in nature.

Of most interest is the generation of the tree structure and associated binary code in 3.; this process is illustrated in figure 4 on 2D data (for simplicity of the visualisation; the generalisation to 3D data is straightforward). The gridded data, arising from step 1. is shown in panel (a) of figure 4; from this data, a series of hierarchical structured grids is generated in step 1. as shown in panels (b)–(d). The binary code used to represent the data is generated in 4 bit sequences (8 bit sequences for 3D data) and is generated as follows, starting at level $n = 1$ (panel (b)).

(1) For each of the current 4 squares under consideration, emit a "1" for filled squares and "0" for empty squares.

[1] See for example the AABB-triangle intersection method of Akenine-Möller [1], with associated C source code available from http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/.

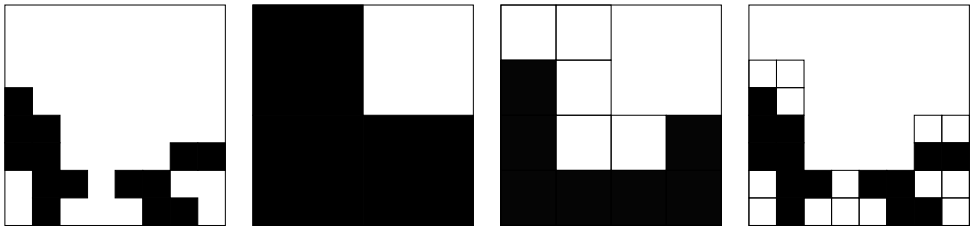


Figure 4. An illustration of tree generation using 2D data (the generalisation to 3D is straightforward). Panel (a) shows the initial data generated by completely enclosing a triangular isosurface mesh by squares on a regular mesh. Panel (b) shows the first-level ($n = 1$) division into squares that completely enclose the squares in panel (a). Panels (c) and (d) show further subdivisions ($n = 2$ and $n = 3$ respectively) until the length scale of the original data is reached.

(2) For each filled square, consider the 4 squares at level $n + 1$ directly covered by the current filled square and recursively apply this algorithm until $n = n_{\max}$.

This provides a depth-first representation of the data, though it is also possible to generate a breadth-first representation in a similar manner. The key thing to note is that the tree terminates as soon as all the squares below are empty, thus providing an efficient representation of sparse data.

Applying this process to the data shown in figure 4, working clockwise from the top-left square, yields

$n = 1$ `1011`
$n = 2$ `0001`
$n = 3$ `0001`
$n = 2$ `0111`
$n = 3$ `0011`
$n = 3$ `0001`
$n = 3$ `1110`
$n = 2$ `1011`
$n = 3$ `1111`
$n = 3$ `1000`
$n = 3$ `0110`

This represents the provided data as 44 bits whereas the original dense representation would be 64 bits. In this example the data is not very sparse and so the compression is relatively limited; in three dimensions, using actual data from Zenotech the achieved compression is much greater.

An additional optimisation is to note that, aside from at level $n = 1$, it is not possible to generate a `0000` bit sequence. As such, that sequence can be used to represent (and so compress) other features in the tree. Here we use the bit sequence `0000` to represent a completely full sub-tree below the current node in the tree.

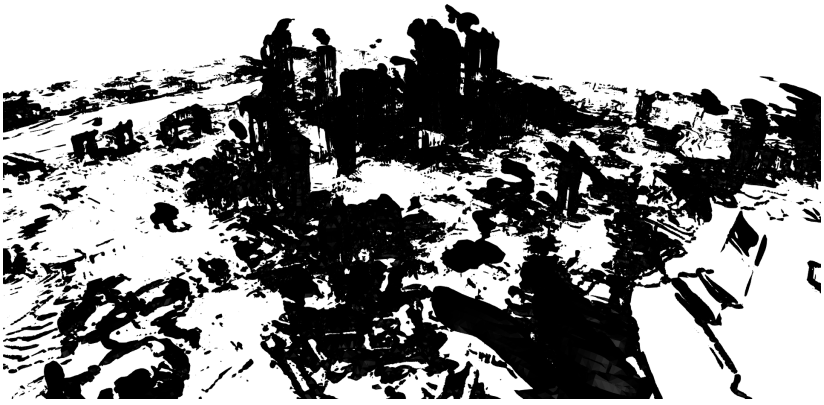

Figure 5. A visualisation of the isosurfaces represented by a triangular mesh provided in `shear_0p001`.

### 3.2 Numerical results

The representation above was tested on the sample isosurface data provided as binary STL files by Zenotech, specifically the `shear_0p001`, `shear_0p010`, and `shear_0p100` datasets. The largest example is `shear_0p001` and is shown in figure 5; the other two datasets contain significantly smaller numbers of triangles in the isosurface mesh.

Due to memory and computational constraints, a $1\,\mathrm{km}^3$ subset of the data taken from the centre of the domain was used as a representative data sample from each dataset. In principle, there is no reason that a larger dataset could not be used; in this case intermediate calculations were stored as a dense 3D matrix for ease of implementation but other, less memory demanding, approaches could be used. The subset of data, with all triangles in the mesh enclosed by $1\,\mathrm{m}^3$ cubes on a structured grid (as per step 1. in section 3.1) is shown in figure 6.

To compare different levels of approximation, the gridded data was also down sampled to $2\,\mathrm{m}^3$, $4\,\mathrm{m}^3$, and $8\,\mathrm{m}^3$.

Applying the binary coding to the original triangular mesh shows significant reductions in the size of the datasets as shown in tables 1 and 2. It is assumed that the original uncompressed triangular mesh data is stored as $9\times32$ bit floating point values (i.e., 36 bytes) per triangle in a binary format (similar to the original binary STL format provided). Standard compression schemes (e.g., RLE, LZW, LZMA, etc) could be applied on top of any of these binary encodings but we ignore this for the purposes of this report.

The tabulated data demonstrates that this type of encoding is capable of achieving the levels of compression that Zenotech is interested in. While it may not be suitable

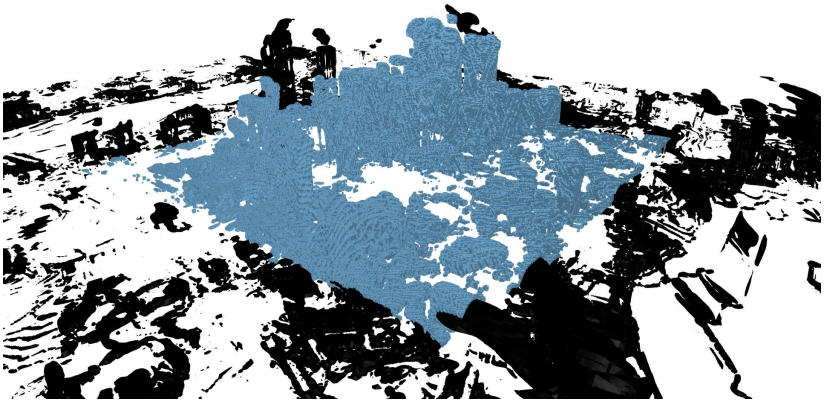

Figure 6. The subset ($1\,\mathrm{km}^3$) of the data considered (as shown in figure 5), with all triangles enclosed by cubes on a regular grid.

Table 1. Storage size (in bytes) of different isosurface representations. The uncompressed triangular mesh data is assumed to be stored as $9\times32$ bit floating point values (i.e., 36 bytes) per triangle. The binary tree is stored as described in section 3.1.

| Representation | `shear_0p001` | `shear_0p010` | `shear_0p100` |
|---|---|---|---|
| Triangles (uncompressed) | 50,122,872 | 13,373,100 | 475,092 |
| Binary tree (1 m cube) | 1,653,652 | 339,730 | 14,960 |
| Binary tree (2 m cube) | 369,363 | 94,642 | 6,384 |
| Binary tree (4 m cube) | 83,038 | 29,002 | 3,140 |
| Binary tree (8 m cube) | 19,953 | 9,175 | 1,525 |

Table 2. Compression factor of the different representations relative to the uncompressed triangular mesh data. Absolute storage sizes are given in table 1.

| Representation | `shear_0p001` | `shear_0p010` | `shear_0p100` |
|---|---|---|---|
| Binary tree (1 m cube) | 30 | 39 | 32 |
| Binary tree (2 m cube) | 136 | 141 | 74 |
| Binary tree (4 m cube) | 604 | 461 | 151 |
| Binary tree (8 m cube) | 2,512 | 1,458 | 312 |

for direct computation due to difficulties with parallel computing, as a storage format it is highly effective. Moreover, assuming that for the applications of interest, the drones are not moving more than a few tens of metres per second, it might be more effective to expand the binary tree into a dense matrix representation on a 1 m structured grid for use with a route planning algorithm since even GPU accelerated computation is unlikely to beat a direct lookup table. (For example, a $1\,\mathrm{km}\times1\,\mathrm{km}\times128\,\mathrm{m}$ volume with a simple binary mask using $1\,\mathrm{m}^3$ cubes can be represented in only 16 MiB.)

### 3.3 Further optimisations

The representation in section 3.1 implicitly assumes homogeneity in the spatial structure in all directions. However, due to the nature of the data that is unlikely to be the case. As such, further optimisations might be possible, such as considering variable splittings in the tree structure where the direction of splitting is considered as an extra optimisation parameter. Equally, implementing special codes within the tree (the `0000` bit pattern as mentioned in section 3.1) may be used more beneficially than representing a full subtree. Moreover, there may be further benefits to be gained by careful evaluation of the transformation from the triangular mesh to the enclosing cubes.

## 4 Compressing volume data with cuboid primitives

In this section, we describe another approach for compressing the data provided by Zenothech. The original problem proposed by Zenotech was to use geometric primitives to approximate two dimensional isosurfaces. However, the windspeed and shear force isosurfaces are not smooth and contain several discontinuities, especially in the dangerous zones data representing high wind speed and high shear forces. This is due to the complexity of air flow over buildings in London. One example of shear isosurface is displayed in Fig. 7.

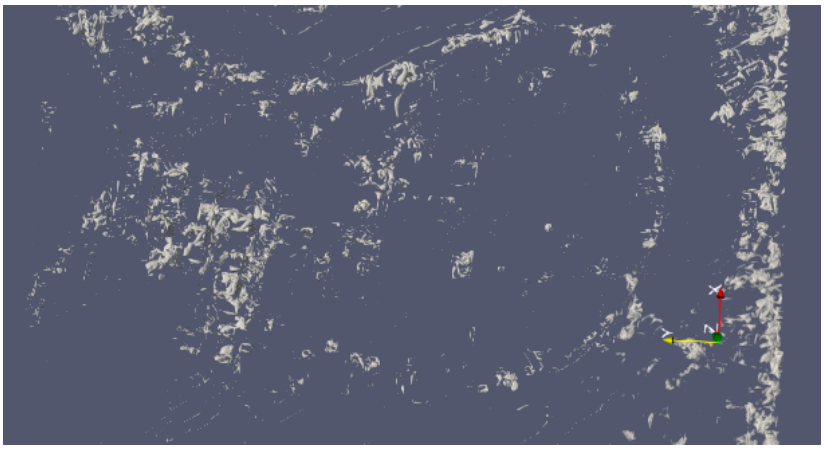

Figure 7. Plot of the shear force isosurface data.

Those discontinuous shear force isosurfaces are of high interest, because they are the dangerous zones and we need to preserve the precision of those structures. In such situation, there is no good models that use primitives to significantly reduce the storage for isosurfaces. However, the complexity of airflow was mainly caused by the clustering of buildings, which means those cloud points data has the natural structure of boxes or cuboid. This follows naturally from the nature of those volumetric data we are facing in this project.

Cuboids have the attributes of length, width, height and stores the location and orientation. Therefore many more information such as the volume can be obtained from its

simple vector structure: $(x, y, z, \text{length}, \text{width}, \text{height})$. The cloud data are simulation results from meshes, which means those data can be projected onto structured meshes and thus our approximation preserves the accuracy within the size of one mesh. Therefore, those cuboid primitives can approximate the cloud data with both high accuracy and low space requirements.

### 4.1 Description of the agglomerating method

The agglomerating method that we employ allows the approximation of the dangerous zones with high wind speed while keeping a good compression rate of the data. This method consists of the following three steps.

(1) From a tolerance $\epsilon$, construct a structured mesh of cuboids with minimum diameter $\epsilon$. Then, project the shear and wind-speed data onto the structured mesh grids to obtain an approximation of the shear force and wind-speed on a regular mesh.

(2) Identify the dangerous areas and group dangerous and the remained safe zones into elementary cubes:

   (a) Take a dangerous shear threshold and flag the cell elements of those structured mesh when they are below the threshold.
   (b) Take a dangerous wind speed threshold and flag the cells elements of those structured grids when they are above the threshold.
   (c) Group the flagged cells together to create cubes of dangerous regions and those of safe regions.

(3) Merge those cubes into cuboids of safe regions using the merging algorithm presented below.

**Subsample data onto structured mesh grids**
We subsample the 3D simulation data to structured meshes with diameter of either 2m, 5m, or 10m. Since the data are airflow of one region in London, and drones use the data a detect the safe and dangerous regions to fly in, we can define the dangerous areas as follows.

- Cubes where the shear value is lower than a threshold $S = 1$, represent buildings. This represents collisions if the drones fly too close to buildings.

- Cubes where the magnitude of the wind speed $V$ is larger than a threshold $V_{max} = 5$, means dangerous fast air flow, i.e. drones need to avoid flying into fast wind.

One example of isosurface of the dangerous shear force area is shown in Fig. 8.

The isosurface plot of buildings from the subsampled 3D data is shown in Fig. 8. The next step is using the flagged data, we group single data points into same-sized

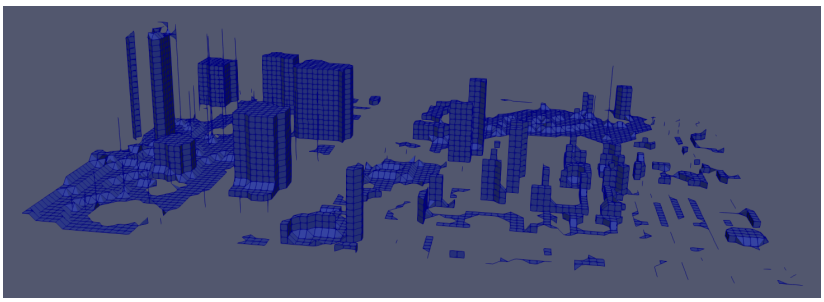

Figure 8. Approximation of the building zones into elementary cubes. This figure shows the isosurface of structured mesh of buildings in the 10m dataset.

cubes $X$. Then save the vector location of one vertex of a cube. Orientation of cubes are maintained by selecting the left-front bottom corner vertex. The cubes approximating the zones where the buildings are located are presented in Fig. 9.

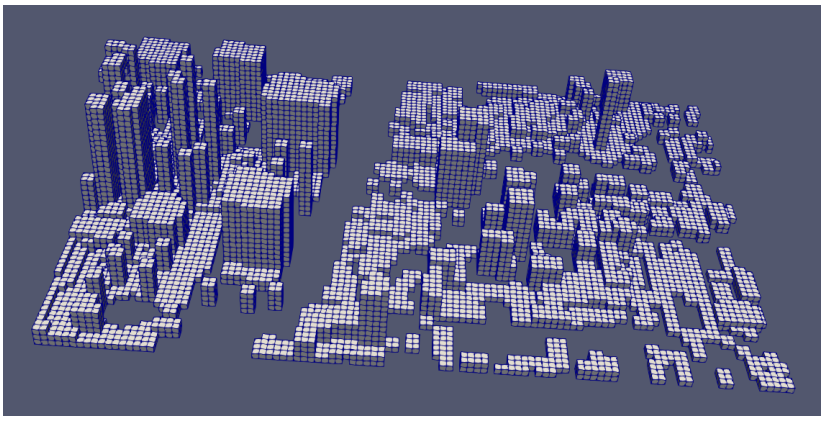

Figure 9. Isosurface cubes plots of buildings. This figure presented the the same information from Fig. 9, only with all data bounded inside cubes.

**Algorithm for finding the connected components**

Here, we describe the algorithm that we employ to group the elementary cubes approximating the safe volumes into connected components. The algorithm has a linear complexity of $\mathcal{O}(n)$ into the number of cubes. The algorithm is as follows:

(1) Let $X = \{x_i\}_{i=1}^{n}$ be the set of elementary cubes of length $\epsilon$.
(2) Initialize the color $c_i$ (i.e. connected component number) of every element in $X$ to 0.
(3) Loop over the cubes $x_i \in X$.
    (a) Find the set of neighbours of $x_i$: $N(x_i)$.

(b) If one of the cubes in $N(x_i)$ is colored with color $c$, color $x_i$ and the other neighbours with color $c$.

(c) Otherwise, choose the color $\max(c)+1$ for $x_i$ and its neighbours $X = X \backslash (x_i \cup N(x_i))$.

In Fig. 10, we show the decomposition of the elementary cubes into disconnected components.

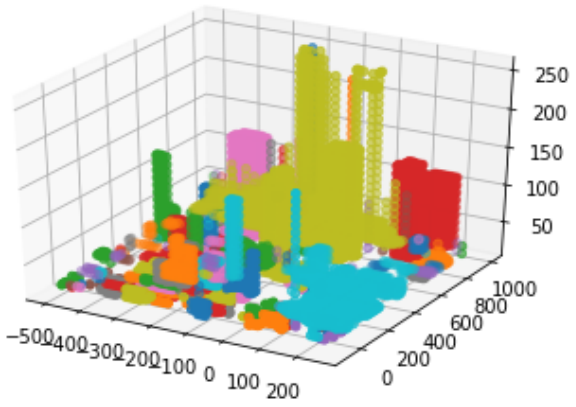


Figure 10. Separation of the elementary cubes into disconnected components

We then gather the cube approximation to the safe volumes by connecting components along the $x$, $y$ and $z$ axis. Details are explained in the following paragraphs.

**Merging cubes into cuboids and separation into disconnected components** Here, we describe the algorithm that we developed to loop through all the separated cubes from the previous algorithm and connect those that should be physically connected. Then we compress the total data further by forming those disconnected components. The algorithm is as follows:

(1) Loop over the disconnected components $X = \{x_i\}_{i=1}^{n}$;

(2) Merging cubes and permutation for best performance:

(a) **Growing boxes** - If the resolution of data is low, i.e. diameter $h >= 5m$, we connect $x_i$ along the descending directions of $x$, $y$ and $z$;

(b) **Permutation optimised growing boxes** - If the resolution of data is high, i.e. diameter $h < 5m$, we run a full permutation and find the best connecting order.

(3) Plot the cuboids and calculate approximation errors.

The algorithms for growing boxes and permutation optimisation are shown as below:

**The growing-box algorithm**
Once we have all boxed labeled with dangerous flags, we need to merge those boxes into separated bodies to achieve the compressing goal.

(1) Take a list of coordinates $L$ and a list of direction to merge cells and return a list of cuboids;
(2) Sort the list $L$ according to the reverse of perm so that the order of the list is coherent with the direction of expansion;
(3) Permute the lengths of the cuboids so that they are given in the order $Lx$, $Ly$ and $Lz$;
(4) Sort the list by ascending $x$, $y$ and $z$ coordinates;
(5) Try to extend each cube in the corresponding direction.

**Optimisation algorithm using permutation of connection directions**
The growing box algorithm only merges the cells in ascending $x$, $y$ and $z$ directions. In order to obtain a better compression result, one could try combination with descending order like $[x, -y, z]$. This can be achieved by applying the transformation $y \to -y$ to the data. Following this idea, we are currently trying $3! = 6$ combinations but we could test $2^3 = 8$ combinations of ascending/descending directions per permutation. Finally, the algorithm returning the combination with the minimum amount of boxes works as follows:

(1) Loop over boxes with the growing box algorithm;
(2) Save the list with minimum amount of boxes;
(3) Connect boxes according to the list would return the best compression results.

**A simple illustration of the above methods:**
We use a small fraction of the dataset to demonstrate how all the algorithms work together.

The original data are simply data points with cubic meshes, we use the *finding the connected components algorithm* to label them as dangerous and not dangerous. Then we plot the isosurface of those labelled boxes as in Fig. 11. Those shown here are part of some buildings. We can see from the figure, those unit cubes preserve the fine details of buildings.

Then we put the example data in Fig. 11 through the *merging cubes into cuboids and separation into disconnected components algorithm*. The returned the results are separated cuboid bodies based on their structure feature. The plot of the result is shown in Fig. 12:

From Fig. 11 and Fig. 12, we have demonstrated how the algorithm works. Now, we can distribute all disconnected sets of cubes onto different cores for the merging algorithm.

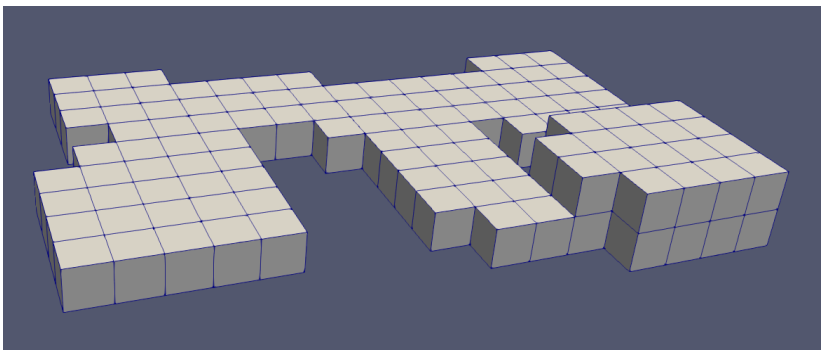

Figure 11. Isosurface cubes

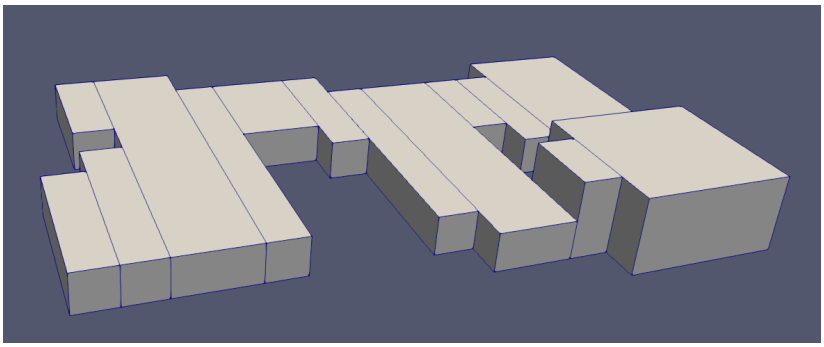

Figure 12. Isosurface cuboids

Put the location of boxes through the algorithm, we obtain the locations and sizes of cuboids that are approximating the buildings. By plotting different cuboids with different colour in Fig. 10, we can see groups of buildings' isosurfaces with their inner boxes are merged together.

Even if we connect those boxes in serial, the benefit of having disconnected sets is that we can find the optimal expansion directions for each disconnected set so we get an improved compression size. The smoothness might matters more. In our testing example, we can reduce the number of elements from 129 cubes (Fig. 11) to 14 cuboids (see Fig. 12).

### 4.2 Numerical results

This model was tested on the sub-sampled data by Zenotech, with mesh diameters of 10m, 5m and 2m. The largest and more accurate dataset example is the one with mesh diameter 2m, whereas the smallest and less accurate one has a mesh diameter of 10m. The compression results are presented in Table 3:

As shown in Table 3, our cuboid-primitive approximation for the $10m$ data gives a $40\times$ compression with running time 0.8 seconds. For the $5m$ resolution data, our model gives $200\times$ compression. This improved compression result is expected as the higher resolution data provides smoother shapes when fitting boxes and cuboids into the the data points. This will allow a smoother merging process therefore improving the compressed results.

Table 3. Number of cuboids approximating the data together with the execution time needed to run the merging cubes algorithm. Timings were performed on a AMD Ryzen 3600 @4.1 GHz without parallelization.

| Size (m) | Cubes | Cuboids | Compression factor | Time (s) |
|---|---|---|---|---|
| 10 | $10^5$ | 2981 | 40× | 0.8 |
| 5 | $7 \times 10^5$ | 3427 | 200× | 5 |
| 2 | $10^7$ | 7369 | 1400× | 170 |

For the biggest dataset, we achieved 1400× results with running time 170 seconds without parallelization.

Another convenience with our compression method is the simplicity in converting to parallel computing. Because data points are simulated as cuboids, partition of meshes can use the geological separation of different regions. This is one naive approach because our data clouds are airflow around buildings. Other parallel partition methods which are commonly used for structured meshes should be implemented easily.

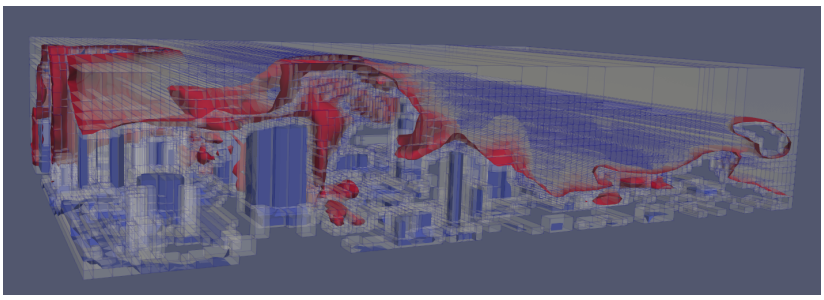

Figure 13. Compressed results with cuboid primitives

The total approximation result for the full $2m$ dataset is presented in Fig. 13. This 3D graph combined the approximation in cuboid boxes and buildings', shear force and wind speed isosurfaces. Blue building shaped surfaces are the building regions and red regions means high wind or high shear. We can observe from the side view of this combined data, that the cuboids envelope the buildings and dangerous wind regions within the dangerous boxes. Therefore, our compressed data preserved the fine details required for drones to operate safely as well as allow them to approach a dangerous region fairly close with the distance of less than a cube diameter.

**Measuring the error**

Estimating cloud points estimations accuracy could be extremely complicated. However, due to the simple structure of our approximation mesh, we could calculate our errors using a straightforward method, which is shown as follows:

(1) Run the compression algorithm then we obtain the set of cuboids. Sum up the volume of those cuboids to get an approximate dangerous volume.

(2) Plot the volume with respect to the size of the structured mesh $h$ to see the convergence.

(3) Plot the cost function, i.e. the number of cuboids with respect to $h$.

## 5 Conclusions and further directions

Out-of-the-box methods from computer graphics did not perform well to compress the aerodynamic environment data. The main reason is the complex geometries of the surfaces of hazardous zones. The method of hierarchical bounding volumes using spheres as a primitive did show poor results to describe the isosurfaces of hazardous areas, but other primitives could be used.

Octree methods exhibit good rates, although the methods may not be suitable for direct computation of safe paths. A more detailed study of the computational requirements would help to understand the applicability of the methods. A natural optimisation of the method presented in Section 3 is to exploit the spatial heterogeneity of the isosurfaces, as well as refining the low level implementation of the octree representation.

Compressing the data using cuboid primitives shows very good compression rates, as well as easy computation of collisions with surfaces. Even with a simple implementation, the method scales well with respect to the accuracy. The simple structure of the approximation mesh also allows the estimation of the error in the approximation.

### 5.1 Recommendations

The methods presented in this report show that high compression rates can be achieved. We tested the methods with a limited dataset; shear force and vortex data should be compressed applying the same methods to verify the compression rates.

Other primitives can be used in the cuboid compression method, as well as for the hierarchical bounding volumes. The balance between algorithmic complexity, accuracy of the representation and compression rate will be dependent on particular dataset, and using different primitives can help to achieve better results.

Finally, some of the proposed algorithms can be optimised with a careful implementation. This is the case of the octree algorithm, which can allow for better compression by optimising the implementation of the data; similarly, we did not study in detail the implementation of the methods presented in Section 4.

The problem proposed by Zenotech inspired several interesting research questions that can be explored in the future. In particular, understanding the algorithmic complexity of the cuboid merging methods, as well as exploring sparse representations of surfaces, would allow to compute bounds on the computational time and compression rates that can be achieved for the aerodynamic environment data.

## Acknowledgements

N.B. was supported by the EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling through grant EP/L015803/1 in collaboration with Simula Research Laboratory.

Y.Z. was supported by the University of Bath and the Chinese Scholarship Council.

## References

[1] Akenine-Möllser, T. (2001). Fast 3D triangle-box overlap testing. *J. Graph. Tools*, **6**(1), 29–33.

[2] Baronti, L., Alston, M., Mavrakis, N., Ghalamzan E., Amir M., & Castellani, M. (2019). Primitive Shape Fitting in Point Clouds Using the Bees Algorithm. *Appl. Sci.*, **9**(23), 5198.

[3] Bradshaw, G. (2002). *Bounding Volume Hierarchies for Level-of-Detail Collision Handling*. Ph.D. thesis, Trinity College Dublin.

[4] Bradshaw, G. (2003). *Sphere-tree construction toolkit*. URL: http://isg.cs.tcd.ie/spheretree/ Last accessed: 2020-11-25.

[5] Broutta, A., Coeurjolly, D., & Sivignon, I. (2009). Hierarchical Discrete Medial Axis for Sphere-Tree Construction. *Pages 56–67 of: International Workshop on Combinatorial Image Analysis*. Springer.

[6] Li, L., Sung, M., Dubrovina, A., Yi, L., & Guibas, L. J. (2019). Supervised fitting of geometric primitives to 3d point clouds. *Pages 2652–2660 of: Proc. IEEE Conference on Computer Vision and Pattern Recognition*.

[7] Lu, Lin, Choi, Yi-King, Wang, Wenping, & Kim, Myung-Soo. (2007). Variational 3d shape segmentation for bounding volume computation. *Computer graphics forum*, **26**(3), 329–338.

[8] Meagher, D. (1982). Geometric modeling using octree encoding. *Comput. Gr. Image Process.*, **19**(2), 129–147.

[9] Paffenroth, R. C., Nong, R., Leed, W. D., & Lundberg, S. M. (2014). *Lossy compression of data points using point-wise error constraints*. US Patent 8,811,758.

[10] Xu, L., Wang, R., Yang, Z., Deng, J., Chen, F., & Liu, L. (2016). Surface approximation via sparse representation and parameterization optimization. *Comput. Aided Design*, **78**, 179–187.