Autonomous Void Detection and Characterisation in Point Clouds and Triangular Meshes

Benjamin Bird
Department of Electrical and Electronic Engineering, University of Manchester, Manchester, United Kingdom
E-mail: benjamin.bird@postgrad.manchester.ac.uk

Thomas Wright
Department of Electrical and Electronic Engineering, University of Manchester, Manchester, United Kingdom
E-mail: thomas.wright@postgrad.manchester.ac.uk

Simon Watson
Department of Electrical and Electronic Engineering, University of Manchester, Manchester, United Kingdom
E-mail: simon.watson@manchester.ac.uk

Barry Lennox
Department of Electrical and Electronic Engineering, University of Manchester, Manchester, United Kingdom
E-mail: barry.lennox@manchester.ac.uk

Abstract:
In this paper we propose and demonstrate a novel void characterisation algorithm which is able to distinguish between internal and external voids that are present in point clouds of both manifold and non-manifold objects and 3D scenes. We demonstrate the capabilities of our algorithm using several point clouds representing both scenes and objects. Our algorithm is shown in both a descriptive overview format as well as pseudocode. We also compare a variety of different void detection algorithms and then present a novel refinement to the best performing of these algorithms. Our refinement allows for voids in point clouds to be detected more efficiently, with fewer false positives and with over an order of magnitude improvement in terms of run time. We show our run time performance and compare it to results obtained using alternative algorithms, when tested using popular single board computers. This comparison is important as our work is intended for online robotics applications, where hardware is typically of low computational power. The target application for this work is 3D scene reconstruction to aid in the decommissioning of nuclear facilities.
1 INTRODUCTION

3D scene reconstruction presents many complex challenges, particularly when integrated into robotic systems where power and weight constraints limit the computational power of the onboard hardware. A robot, whether a mobile exploratory robot or fixed industrial manipulator needs an accurate model of its surroundings in order to have any form of meaningful autonomous function, Hornung et al. (2013). For this model to be accurate and therefore useful, it must be complete, which in the case of an individual object within the scene means that it must be watertight (geometrically complete), Dey et al. (2003). This is particularly relevant in tasks where manipulation of objects is required, such as tele-operated laser or plasma cutting using an industrial manipulator, as the user does not necessarily have the freedom to completely inspect the object, Bogue et al. (2011), IAEA et al (2017).

A common problem with 3D scene reconstruction is the existence of voids within the captured data, Jun et al. (2005). Voids are areas within the scan where data about the scene is missing and can result when the surface at that particular location has poor reflective qualities relative to the type of sensor being used, Fang et al. (2004), or when another object is in the way and the scanning device cannot see that object (which is referred to as occlusion), Bendels et al. (2006).

The manner in which voids are repaired is dependent on several factors, such as size, location with respect to the boundaries of the point cloud and the surface geometry surrounding them, Dey et al. (2003). However, before any void is repaired, it must first be classified as either internal or external, where external voids are characterised as the edges of a point cloud and internal voids are those surrounded by point cloud data. The reason that the distinction is required between internal and external voids is that void repair algorithms, such as Park et al. (2005), which use artificial data, rely on the surrounding data when repairing the void, Reuter et al. (2007), Kim et al (2017), Chollet et al. (2017), Neteler et al (2012) or in the case of Park et al. (2005), determine a void by propagating two separate active contours from either side of a prospective void. These approaches cannot be applied if a void is at the edge of a point cloud. Whilst manual methods have been proposed that are able to determine whether a void is internal or external, such as in Bendels et al. (2006), and Yang et al. (2017), these methods require considerable user intervention and can become
Autonomous Void Detection and Characterisation in Point Clouds and Triangular Meshes

infeasible in any relatively complex scene. A more appropriate approach would be to apply an automated routine that would identify all voids in a point cloud and then characterise them as internal or external.

Following characterisation, internal voids could be repaired using, for example, the artificial data generation methods proposed by Park et al. (2005) and Reuter et al. (2007) and the external ones either ignored or repaired by gathering more data in the unscanned regions in the environment until the scan is manifold (we define manifold in this work to be an indication as to whether point cloud data is captured in a surrounding fashion, i.e 360 degree data).

External voids are typically found in situations where a 3D environment is scanned from between two arbitrary locations. For example, if a wall is scanned from the floor to the roof, external voids would be located at the edge of the point cloud at both the top and the bottom of the wall. This type of point cloud is referred to in this work as non-manifold. As stated earlier, internal voids are typically found when the object being scanned is occluded by something, or where the surface of the object has poor reflective properties and are hence found in both manifold and non-manifold point clouds. An example of a point cloud that has been converted into a triangular mesh, that highlights both an internal and external void is shown in Fig 1.

In this work we utilised the technique proposed by Jun et al. (2005) for the initial detection of voids. However, modification of this technique (discussed in Section 3) allowed significant improvements to be made to it, in terms of run time performance and minimising

![Figure 1: Depiction of the boundary of an external void and the boundary of an internal void](image)
erroneous outputs. Furthermore, we present a novel algorithm to enable the automatic characterisation and subsequent repair of the voids. The modification to the void detection algorithm and our novel void characterisation algorithm are demonstrated using a variety of point clouds, that includes both scenes and objects. Application to these point clouds clearly demonstrates the improvements in both processing time and output quality that the proposed techniques offer.

The remainder of the paper is organised as follows: In section 2 a literature review is presented on previous void detection work; in section 3 the authors show how an existing void detection algorithm has been modified to improve its computational efficiency by an order of magnitude; in section 4, the novel void characterisation algorithm is presented; and in section 5 the novel algorithm is assessed in terms of its performance.

2 PREVIOUS WORK ON VOID DETECTION

Several different approaches have been proposed in the literature for the detection of voids and in this work the most suitable of these techniques have been evaluated and compared. The particular approaches that were chosen for this study were algorithms that were found to be most frequently applied in research studies or had been demonstrated to produce accurate results in comparative studies. The methods chosen are as follows: The Marching Cube detection and correction algorithm by Park et al. (2005); the Vertex Angle Check algorithm by Dey et al. (2003); the Single Triangle Vertex Identification algorithm by Jun et al. (2005); the K Neighbour Angle Criteria algorithm by Bendels et al. (2006); and the shape controlled geometry completion algorithm proposed by Yang et al. (2017). Each method is described below, with a subset of them evaluated in the following subsection.

The Marching Cubes void detection and correction algorithm presented by Park et al. (2005) has been found to be an effective detection and correction algorithm for subjectively small voids within the point cloud (where the void is small in relation to its surroundings), provided it is manifold. However, the algorithm is unable to process none manifold point clouds, and therefore cannot be considered as a universal and automated point cloud void detection, characterisation and correction algorithm. It was therefore not evaluated further.

Yang et al proposed a novel algorithm in Yang et al. (2017) which was able to fill point cloud voids in geometrically complex areas, including areas of high frequency geometry (referred to as sharp in their work). Their method relies on the user detecting the point cloud voids, after which their algorithm calculates the surface normals at the void boundary. Finally, their method contracts the void boundary iteratively, by using the surface normals to project new points into the void.

Unfortunately, Yang et al’s algorithm, Yang et al. (2017) cannot process both low and high frequency geometry (smooth and sharp) void locations at the same time. That is to say input parameters for one scenario will not work for a different scenario. Their algorithm also requires significant user intervention in detecting the voids in the point cloud at start up, as well as selecting optimum start up parameters. For this reason, it was not evaluated further.

Dey et al. (2003) present an algorithm which takes an input point cloud and turns it into a polygonal mesh using a Delaunay Triangulation algorithm. It then performs an angle check between each vertex in the polygonal mesh and a function of its local neighbour’s position with respect to itself. If this satisfies a user specified input parameter, the vertex is regarded as being on the boundary.
Jun et al. (2005) also present an algorithm where the initial step is to perform a Delaunay Triangulation of the input point cloud, to form a polygonal mesh. Their void detection method relies on the fact that in a triangular mesh, triangles which surround a void contain vertices that are not shared with other polygons, illustrated in Fig 1. A simple data search is performed on the polygonal mesh to retrieve the vertices that are only present in a single polygon and hence lie on the boundary of a void. Bendels et al. (2006) describe another algorithm which takes the input point cloud and evaluates each individual point using an angle criteria function. This takes the input point and a modified set of k-nearest neighbours to determine whether or not the point lies on the edge of a void.

2.1 Comparison

None of the algorithms discussed above are able to distinguish between internal and external voids. However, Jun et al’s algorithm - Jun et al. (2005) is arguably the most universal approach for identification as the size of the void has no effect on the algorithm’s ability to specify if a vertex surrounds a void or not. Yang et al’s algorithm - Yang et al. (2017) has been shown to be effective with void areas in high frequency geometry, but requires significant user intervention and precise initial configuration, much like both Dey et al. (2003) and Bendels et al. (2006), who both show that their algorithms have to be given specific initial parameters to detect voids in particular data sets. An important limitation of Yang et al’s algorithm is that it cannot cope with both smooth geometry voids and high frequency voids simultaneously, Yang et al. (2017).

The void detection techniques that have been reviewed are able to detect point cloud voids under ideal circumstances, but due to the probabilistic nature of Bendels et al. (2006)’s algorithm, in that depending on start and surface conditions, it is not guaranteed to find any voids and for particular point cloud densities and surface geometries, the algorithm can result in the failure to detect voids, this is discussed in length by the authors themselves. This can be mitigated by an appropriate choice of initial conditions for the algorithms, such as $\theta$ and $\rho$ for Dey et al. (2003) and the number of neighbours in the neighbour search for Bendels et al. (2006). This aspect can be considered a positive feature of Dey et al. (2003)’s algorithm, as it can be used as a filter to disregard artificial voids, which can be produced when using Delaunay Triangulation. These artificial voids are small voids where point cloud data is present at a local minima, and is sub optimal for meshing, causing small voids to be inserted. Additional artificial voids caused by the Delaunay meshing algorithm are double hulls, where two parallel hulls are generated to represent a real world hull. This is again caused by the sparse nature of points at boundary edges.

Problems associated with artificial voids can occur when using Jun et al. (2005)’s void detection algorithm, because of the use of Delaunay Triangulation (discussed in detail in section 3). However this algorithm is superior to others in the respect that it will detect a void in the mesh no matter what the surface conditions are. Both the operational time complexity (cost in terms of loop interactions) and actual complexity (cost in terms of mathematical complexity) of all three algorithms should also be taken into account. However, the authors have not analysed operational time complexity nor discussed the complexity of the algorithms. However, by using reasonable assumptions, best and worst case scenarios for each algorithm can be determined. A comparison of the different methods in terms of operational time complexity is shown in Fig 2.

Fig 2 shows the increasing operational complexity with respect to the number of points in the point cloud. It shows Dey et al’s algorithm, Dey et al. (2003) to be the most efficient
under this performance metric, with Bendels et al algorithm, Bendels et al. (2006) the next most efficient for point clouds greater than 10,000 points, provided k is set to 10. The next most efficient algorithm is Jun et al’s algorithm, Jun et al. (2005). This is then followed by Bendels et al’s algorithm, Bendels et al. (2006) with the number of k-nearest neighbours set to 15 and 20 respectively. As can be seen from Fig 2, the operational complexity increase for k = 15 and k = 20 is considerable: 3 and 10 orders of magnitude respectively. This will eventually become lower than the operational complexity of Jun et al. (2005) and Dey et al. (2003), as the number of points in the point cloud trends to infinity. However the running time for this would also trend to infinity. It should be noted that Bendels et al. (2006) stated that k = 15 was the optimum setting for useful void detection and therefore the greater efficiency at k = 10 is unlikely to be relevant.

The functional complexity of Dey et al. (2003) is also far greater than Jun et al. (2005), and therefore the real world run time for this algorithm would be slower than Jun et al. (2005), as it consists of several floating point operations per loop iteration, as opposed to the integer operations of generating and searching a Delaunay mesh. This algorithm also requires the input parameters to be finely tuned, as opposed to the Delaunay search algorithm presented in Jun et al. (2005), which which requires no such starting parameters.

Table 1 provides a summary comparing the performance of the various algorithms. This table shows that Jun et al’s algorithm provides the best starting point for a void characterisation algorithm and was therefore chosen as the basis for the work subsequently described in this paper.
Table 1  Comparison table, summarising the four different void detection algorithms

<table>
<thead>
<tr>
<th>Algorithm name</th>
<th>Void type</th>
<th>Manifold</th>
<th>Noise resilience</th>
<th>Time complexity 100k points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park et al [7] -</td>
<td>Internal only</td>
<td>Point cloud</td>
<td>Will increase computational time due to more points</td>
<td>Un tested due not being able to process non-manifold point clouds</td>
</tr>
<tr>
<td>Marching Cube</td>
<td></td>
<td>must be manifold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dey et al [2] -</td>
<td>Internal and</td>
<td>Manifold and</td>
<td>Can cause algorithm to crash due to undefined angle between two points occupying the same space</td>
<td>1.03E+10</td>
</tr>
<tr>
<td>Vertex Angle Check</td>
<td>external</td>
<td>non-manifold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun et al [4] -</td>
<td>Internal and</td>
<td>Manifold and</td>
<td>Will increase computational time due to more points</td>
<td>3.17E+12</td>
</tr>
<tr>
<td>Single Triangle Vertex</td>
<td>external</td>
<td>non-manifold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bendels et al [1] -</td>
<td>Internal and</td>
<td>Manifold and</td>
<td>Can cause algorithm to crash due to undefined angle between two points occupying the same space</td>
<td>1.31E+17</td>
</tr>
<tr>
<td>K Neighbour Angle Criteria</td>
<td>external</td>
<td>non-manifold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 BALL PIVOTING MODIFICATION

In this section, a modification to Jun et al’s algorithm for void detection is introduced that gives an order of magnitude computational speed increase for subsequent processes due to an increased signal to noise ratio (SNR) in the output.

A step-by-step summary of the void detection algorithm proposed by Jun et al is shown in Algorithm 1.

Algorithm 1 Jun et al’s void detection algorithm

1: procedure (point cloud)
2: for point in point cloud do
3: add to mesh and re triangulate affected areas
4: end for
5: for edge in mesh do
6: if Edge is not in mesh then
7: add edge to list of void encompassing edges
8: end if
9: end for
10: end procedure
return Void edge

The Delaunay Triangulation algorithm is well documented, Su et al. (1997), and straightforward to reproduce. However when using this technique an additional step is required after its implementation to eliminate unwanted surfaces. In building the mesh, triangles are used to represent surfaces, however, some of the triangles will form surfaces that do not exist in reality. This is shown in the central image of Fig 3, where it can be seen that a shroud is effectively drawn over the figure. To correct for this, a filter needs to be applied which removes edges that exceed a specified length. An arbitrary value of twice the average edge length was found to be effective in this work. The resulting mesh is shown in the right image of Fig 3.
Delaunay triangulation is commonly used for generating triangular meshes, Jun et al. (2005), Su et al. (1997), Bendels et al. (2006), however as discussed previously it is prone to generating artefacts in the resulting mesh and therefore in this study a variety of different mesh generation algorithms were investigated. There are several alternative mesh generation algorithms, Cignoni et al. (2008), that could be implemented in preference to Delaunay Triangulation. For example, the Ball Pivoting mesh generation algorithm described by Bernardini et al in Bernardini et al. (1999) and Poisson Surface Reconstruction, Hoppe et al. (2008). The latter was not considered in this study because of both its functional and implementation complexity.

Bernardini et al show that their algorithm is simple to implement, and there are several open source implementations of this algorithm that are readily available, such as Cignoni et al. (2008) and Tang et al. (2013).

Experimental results showed that not only does the Ball Pivoting algorithm produce superior quality meshes, in terms of both the smoothness of surface reconstruction and low noise output, it also required significantly less run time to generate a mesh. The reduced run time, Mesh T(s), is illustrated in Table 2 which shows that the time to produce the mesh was consistently lower when using the Ball Pivoting algorithm. The reason for this is that Delauney Triangulation can result in the creation of "double hulls", rather than a single hull, as would exist in reality, Di Angelo et al (2011). The double hulls increase the number of void edges which as a consequence increases the computation time to generate the mesh. In contrast, meshes resulting from the Ball Pivoting algorithm do not contain any double hull artefacts, and therefore have considerably fewer void edges.

A visual comparison between the two meshing algorithms is illustrated in Fig 4, which shows that the two resulting meshes have the same overall shape. Table 2 shows a detailed comparison of the two algorithms. The results presented in this table highlight the superior performance gained by using the proposed Ball Pivoting mesh generation algorithm, when compared with the Delaunay Triangulation based approach. In particular it can be seen that the meshing time of the Ball Pivoting algorithm is approximately half that obtained when using Delaunay Triangulation. This was consistent for all the point clouds that were tested, which varied in size from approximately 2,000 to 100,000 points, and contained

---

**Figure 3**: Delaunay Triangulation filter demo. Input point cloud left, Delaunay Triangulated mesh centre, filtered mesh right.
both manifold and non-manifold objects and scenes, with varying levels of noise (between 0.001% and 5% erroneous noise data).

It should also be noted that using the Delaunay variant of Jun et al’s algorithm has an exponential memory requirement, which created memory problems when processing some of the images. Additional information regarding the performance of the computer used to benchmark these algorithms is provided in the Appendix.

Figure 4: Museum piece with Ball Pivoting mesh generation (left) and Delaunay mesh generation (right)

Further analysis of Table 2 shows that when Jun et al’s algorithm is used with the Ball Pivoting algorithm, void detection time is improved by over an order of magnitude. Both instances of Jun et al’s algorithm locate the voids within the point cloud faster than Bendels et al’s algorithm, by approximately 25% in point clouds of 30,000 points or more and is approximately an order of magnitude faster for point clouds of less than 30,000 points. Bendels et al’s algorithm also proved to be incapable of processing point clouds with duplicated points, which were prevalent in point cloud B, shown in Fig 7. This image was composed of two merged point clouds taken from different vantage points. The reason that Bendels et al’s algorithm cannot be applied to such point clouds is that the algorithm requires knowledge of the angle between the points, which for two points occupying the same location cannot be defined. To alleviate this problem, a filter was applied to the point cloud to remove duplicated points. Bendels et al’s algorithm also showed poor performance in our testing with respect to point clouds with small voids and noisy data, failing to detect the voids in these point clouds entirely. This was most pronounced in point cloud A, where the technique detected no voids.
Table 2: Comparison table, showing benchmark times for Jun et al.'s algorithm with the Delaunay mesh generation algorithm and Ball Pivoting algorithm, and Bendels et al.'s algorithm. Int and ext voids are the number of internal and external voids, and the processing time $T(S)$ is the time taken to characterise them as such, which is shown to give a better comparison of the different algorithms effectiveness. Note that Jun et al.'s algorithm with an excessive number of points caused a segmentation fault, due to the test bench hardware running out of memory. Point clouds A through E are shown in Figs 6, 7, 8, 3, 9 respectively.

<table>
<thead>
<tr>
<th>Point Cloud / Points</th>
<th>Algorithm 1</th>
<th>Mesh $T(s)$</th>
<th>Faces</th>
<th>Void Detect $T(s)$</th>
<th>Total Voids</th>
<th>Int Voids</th>
<th>Ext Voids</th>
<th>Process $T(s)$</th>
<th>Total $T(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A / 2466</td>
<td>Jun et al [4] Delaunay</td>
<td>1.53</td>
<td>9285</td>
<td>16.428</td>
<td>186</td>
<td>176</td>
<td>10</td>
<td>0.472</td>
<td>18.43</td>
</tr>
<tr>
<td></td>
<td>Jun et al [4] Ball pivoting</td>
<td>0.72</td>
<td>4873</td>
<td>0.419</td>
<td>48</td>
<td>40</td>
<td>8</td>
<td>0.061</td>
<td>1.20</td>
</tr>
<tr>
<td>C / 31490</td>
<td>Jun et al [4] Delaunay</td>
<td>14.44</td>
<td>323783</td>
<td>2415.98</td>
<td>262</td>
<td>122</td>
<td>140</td>
<td>0.77</td>
<td>2431.19</td>
</tr>
<tr>
<td></td>
<td>Jun et al [4] Ball pivoting</td>
<td>6.41</td>
<td>56522</td>
<td>70.91</td>
<td>4070</td>
<td>3734</td>
<td>336</td>
<td>144.34</td>
<td>221.66</td>
</tr>
<tr>
<td>D / 1973</td>
<td>Jun et al [4] Delaunay</td>
<td>1.37</td>
<td>9804</td>
<td>10.75</td>
<td>114</td>
<td>29</td>
<td>85</td>
<td>0.64</td>
<td>12.76</td>
</tr>
<tr>
<td></td>
<td>Jun et al [4] Ball pivoting</td>
<td>0.69</td>
<td>3674</td>
<td>0.27</td>
<td>159</td>
<td>102</td>
<td>57</td>
<td>0.95</td>
<td>1.91</td>
</tr>
<tr>
<td>E / 98238</td>
<td>Jun et al [4] Delaunay</td>
<td>102.08</td>
<td>825487</td>
<td>Seg Fault</td>
<td>Seg Fault</td>
<td>Seg Fault</td>
<td>Seg Fault</td>
<td>Seg Fault</td>
<td>Seg fault</td>
</tr>
</tbody>
</table>

A through E are shown in Figs 6, 7, 8, 3, 9 respectively.
Autonomous Void Detection and Characterisation in Point Clouds and Triangular Meshes

The results in Table 2 show that the Ball Pivoting technique allows for a greater than order of magnitude improvement in run time for void detection, which is particularly relevant if the technique is to be deployed on low power processors.

4 VOID CHARACTERISATION

After locating the voids in a point cloud, the next step is to characterise them, i.e. determining whether they are internal or external. The main contribution of this paper is in automating this process, which was achieved through the steps listed below:

1. **Down Sampling** - The point cloud is first down sampled using $k$-means clustering. It was found through experimentation that sampling using $k$-means clustering preserved the overall geometry of the point cloud, whilst greatly reducing the computational time for any further operations due to the reduction of points. If point cloud quality is poor due to measurement noise, consolidation algorithms such as that proposed by Wang et al. (2013) or Liu et al. (2012) can be used to increase the SNR, however this can introduce considerable processing time.

2. **Meshing** - The point cloud is meshed into a triangular mesh using the Ball Pivoting algorithm as discussed in section 3.

3. **Single Edge Search** - Edges which only occur in a single triangle are searched for in the mesh. This produces a list of triangle edges which form a boundary line of either an internal or external void.

4. **Boundary Line Plane Extraction** - The centre point of each boundary line is determined and a plane, perpendicular to this line, is calculated with its centre at the middle of the boundary line and its radius that of a pre-defined value (defined with consideration of the mean boundary length), as shown in Fig 5. Each boundary line is then further evaluated, by checking to see if it lies on the plane previously defined. If it is found to lie within the radius of the disk on that plane, it is added to a list of boundary lines which surround internal voids. If it does not lie within the disk then it is added to a second list comprising boundary lines which make up the edge of the point cloud (external voids).

The 5-point procedure above is illustrated in the pseudocode shown in Algorithm 2

5 VOID CHARACTERISATION ASSESSMENT

This section provides results when our proposed characterisation algorithm was applied to point clouds representing manifold objects as well as point clouds, which were captured as part of larger 3D scenes. To perform the tests, five point clouds were utilised. These point clouds are as follows: Two objects - the museum piece (point cloud D) shown in Fig 3 with 1973 points and a pair of scissors (point cloud A) comprising of 2466 points shown in the right image of Fig 6. Three real world scenes, one of which is shown in Fig 7 (point cloud B), which was produced using measurements taken from 2 vantage points within an office environment and consists of 12467 points, a singular vantage point of a meeting room with several glass windows (point cloud C) consisting of 31490 points shown in Fig 8 and a final
Algorithm 2 Void characterisation

procedure Void characterisation(mesh)
    for edge in mesh do
        if edge not repeated then
            copy centre of edge to edge centre list
            copy edge to void edge list
        else
            continue
        end if
    end for
    for edge centre list in void list do
        Find plane perpendicular to edge
        Find 3 points with radius R on plane from edge centre point
        Determine plane equation of plane which points lie on
    end for
    for edge in mesh do
        for plane in plane list do
            if edge intersects plane then
                if intersection < R away from edge centre point then
                    add edge to external void list
                else
                    add edge to internal void list
                end if
            end if
        end for
    end for
    return external void list, internal void list
end procedure
manifold scene of a large room, (point cloud E) shown in Figs 9 and 10 consisting of 98,238 points. These point clouds were chosen as they represent varying levels of complexity, noise and contain different examples of real-world scenes.

Each of the point clouds was processed using the algorithm described in Section 4 and the results are now discussed.

Figs 6 to 10 demonstrate the ability of the proposed algorithm, utilising a mesh generated using the Ball Pivoting algorithm to differentiate between internal and external voids, which are depicted by green and red points respectively. Delaunay mesh generation was also used in the authors experimentation but was found to significantly increase the computational time of the void characterisation algorithm. Figs 6 to 10 show that the algorithm performs better with point clouds with greater uniformity and limited erroneous points surrounding them. Fig 6 shows that the external voids of the museum piece have been correctly identified (the bottom of the point cloud is void of points as the model was not scanned completely) The same can be said for the scissors, also shown in this figure. The base of this point cloud (the blade of the scissors) was successfully identified as the external void, although due to the Ball Pivoting algorithm wrapping over slightly at the edges, only the middle of the scissor blade was identified (shown as red points). The internal voids in both point clouds have been correctly identified in green. In both these examples, the proposed algorithm identified all voids correctly with no erroneous results.

Fig 7 provides an example where the proposed void characterisation algorithm (using a Ball Pivoting generated mesh) performs poorly. External voids are again shown in red and internal voids in green in this figure. In this point cloud, the algorithm has failed to correctly characterise the majority of the voids. For example, the right hand edge of the point cloud
marks the edge of the imaged room and should be marked as an external void, as should the internal voids identified at the top of the figure. The reason for the poor performance is that this point cloud is only partial, only showing the corner of a room, i.e. the image is not manifold. This reduces the amount of points that the void characterisation algorithm can use to distinguish between internal and external voids.

Fig 8 shows the algorithm’s inability to deal with relatively large voids within the point cloud. In this figure there are four windows (highlighted in yellow) which have been classed as both internal and external voids. The algorithm should have identified these as internal voids, as they are entirely surrounded by point cloud data. This figure further highlights that the algorithm struggles to identify external voids, around the perimeter of the image, when the point cloud is only partial.

**Figure 6**: Point cloud D / 2466 points - Museum piece (left) and point cloud A / 1973 Scissors with characterised voids (right) - meshed using the Ball Pivoting mesh generation algorithm
Figure 7: Point cloud B / 12467 points - Wall point cloud with characterised voids - meshed using the Ball Pivoting mesh generation algorithm

Figure 8: Point cloud C / 31490 points - Meeting room manifold scene, high and low resolution input point clouds with characterised voids. It should be noted that this scene contains a series of windows in the back right hand corner, which have not been correctly characterised for the most part - meshed using the Ball Pivoting mesh generation algorithm
The use of the Ball pivoting algorithm, whilst not immune to erroneous void detection, does show significant improvements in classifying voids in manifold objects, when compared to the Delaunay based approach. The Ball pivoting based void detection and characterisation algorithm shows similar void locations (albeit with fewer erroneous voids detected) in non manifold objects, shown in Fig 8, which was generated using the Ball pivoting algorithm.

Furthermore, the Ball Pivoting approach requires significantly less computation time. This is shown in Table 2.

**Figure 9**: Point cloud E / 98238 points - Manifold room with characterised voids - meshed using the Ball Pivoting mesh generation algorithm

**Figure 10**: Point cloud E / 98238 points - External view of manifold room, showing point cloud on the left, and mesh on the right and characterised voids - meshed using the Ball Pivoting mesh generation algorithm

Figs 9 and 10 show a 3D scene that has been imaged in the style of a manifold object. The scan was taken from a single vantage point with a LiDAR rotated through 360 degrees. This particular 3D scene contains two additional rooms, located in the region marked as containing an external void (red points). There are open doors into these rooms but only a small portion of these rooms was visible to the LiDAR. Figs 9 and 10 show that the void detection algorithm using the ball pivoting generated mesh was able to detect the voids present due to occlusion and poor reflective surfaces (these are evident from the large number of green points marked on the image). Voids that are due to the point cloud terminating have
## Table 3: Summary table of void characterisation algorithm performance (using Ball Pivoting mesh generation algorithm)

<table>
<thead>
<tr>
<th>Point cloud</th>
<th>Type</th>
<th>Internal voids</th>
<th>External voids</th>
<th>Manifold estimation</th>
<th>Characterisation accuracy</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A / 2466</td>
<td>Manifold object with single external void</td>
<td>44</td>
<td>4</td>
<td>95%</td>
<td>100%</td>
<td>Correctly characterises all voids.</td>
</tr>
<tr>
<td>B / 12467</td>
<td>Non-manifold scene - corner of room</td>
<td>1017</td>
<td>863</td>
<td>20%</td>
<td>15%</td>
<td>Failed to characterise most of perimeter as external, but correctly characterises all internal voids.</td>
</tr>
<tr>
<td>C / 31490</td>
<td>Manifold scene with large section missing</td>
<td>3734</td>
<td>336</td>
<td>65%</td>
<td>50%</td>
<td>Correctly identified void areas, but accuracy not absolute</td>
</tr>
<tr>
<td>D / 1973</td>
<td>Manifold object</td>
<td>102</td>
<td>57</td>
<td>95%</td>
<td>100%</td>
<td>Correctly characterises all voids.</td>
</tr>
<tr>
<td>E / 98238</td>
<td>Manifold object with single external void</td>
<td>10203</td>
<td>3595</td>
<td>95%</td>
<td>95%</td>
<td>Correctly identified void areas, but accuracy not absolute</td>
</tr>
</tbody>
</table>
also been identified as the areas in the images highlighted in red. The red areas represent the boundaries within the external rooms that only partial scans have been made and hence have been correctly identified as external voids. Whilst the void characterisation algorithm has mis-characterised a number of voids, it is evident from Figs 9 and 10 that the proposed algorithm provides sufficient information to highlight areas where information is missing.

Figs 7-10 show that the more manifold the object or scene is, the greater the accuracy of the void characterisation algorithm. This is evident when Figs 9 and 10 are compared to Figs 7 and 8. The scenes in Figs 7 and 8 are not manifold and as a consequence the proposed void characterisation algorithm performs poorly.

To quantify the effectiveness of the proposed characterisation algorithm, we have estimated the effectiveness of the algorithm in terms of how many internal voids have correctly been characterised, and an estimation as to what degree the input point cloud is manifold in our definition of the word, as discussed in section 1. This information is shown in Table 3, which shows a direct correlation between the degree to which the point cloud is manifold and our algorithm’s ability to correctly characterise the voids as internal and external. As stated previously, whilst our algorithm is not 100% accurate, it does provide information that can help locate external and internal voids.

CONCLUSION

The goal of this work was to review, extend and improve void detection techniques, and to provide a methodology for automatically characterising identified voids to enable point cloud repair. Detection of voids has been achieved by modifying the algorithm proposed by Jun et al. (2005) by using a Ball Pivoting mesh generation algorithm instead of a Delaunay mesh generation algorithm. This has lead to an order of magnitude speed increase in the void detection algorithm when used on a variety of point cloud examples. A novel void characterisation algorithm was then proposed, and was shown to be able to differentiate between external and internal voids with sufficient accuracy to provide the general physical location of the voids.

The next step in this work will be to focus on the automated re-scanning of areas of interest identified by the void correction algorithm, and to compare this repair procedure with one that uses artificial data, to produce complete point clouds and by extension, water tight meshes.

References


6 Appendix

The benchmarking was done using the benchmarking tool Sysbench Kopytov et al. (2004) to calculate prime numbers. The system is a Linux VM (Ubuntu 16.04 LTS) under Virtual Box, running on a 2016 Macbook Pro Retina Touch Bar with i5 processor clocked at 2.9 GHz, 256 KB L2 cache, 4 MB L3 cache and 8 GB of LPDDR3 memory clocked at 2.133 GHz. Also shown is the comparative performance of the Raspberry pi family of single board computers, which are commonly used as the embedded computers of many mobile and fixed robots.
The Authors would like to thank the Engineering and Physical Sciences Research Council (EPSRC) and the National Nuclear Laboratory (NNL) for funding this research, through case number EP/R026084 and industrial case number 14220132 respectively.

Table 4  Sysbench performance benchmark of system and the raspberry pi family of computers.

<table>
<thead>
<tr>
<th>Name</th>
<th>min (ms)</th>
<th>avg (ms)</th>
<th>max (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux VM</td>
<td>2.26</td>
<td>2.54</td>
<td>6.77</td>
</tr>
<tr>
<td>Pi 0</td>
<td>90.55</td>
<td>90.61</td>
<td>114.35</td>
</tr>
<tr>
<td>Pi 1</td>
<td>131.42</td>
<td>131.83</td>
<td>299.87</td>
</tr>
<tr>
<td>Pi 2</td>
<td>76.43</td>
<td>78.82</td>
<td>82.1</td>
</tr>
<tr>
<td>Pi 3</td>
<td>47.69</td>
<td>47.7</td>
<td>49.91</td>
</tr>
</tbody>
</table>