Comparison of sample results produced by QSplat and ROD-TV, rendered at a higher resolution than the scanning resolution. The shading primitives used in QSplat is (A) points, (B) quadrilaterals, (C) ellipses, (D) spheres, (E) circles, and (F) round points. Shading primitives produced by ROD-TV are shown in (G), where per-vertex normals are not reconstructed, and in (H), where per-vertex normals are reconstructed.
Figure 1.2 - Noise Generation in LUNG Dataset

A 25% LUNG dataset consists of two input components. The first component is the original noise-free LUNG dataset (3,035 tokens). The second component is the noise generation. In this case, we make use of the impulse noise model, where noises randomly occur within a given range. For a 25% noise dataset, there are 759 random tokens (3,035 * 0.25 = 759).
Figure 1.3 - Robustness of ROD-TV [part 1]

The original LUNG dataset (3,305 tokens) with 25% noise (759 tokens) is illustrated in the left. After processing sparse tensor voting, each token site gets the surface saliency information, which indicates the likelihood of a token lying on a smooth surface. The figure on the right shows a point rendering in quadrilateral mode without any surface saliency cut-off.

Figure 1.4 - Robustness of ROD-TV [part 2]

The above figure shows a point rendering using sphere as a rendering primitives on LUNG dataset with 25% noise. Since we don’t know the direction of the site, sphere is used as a rendering primitive. In close-up view, we can visualize the artifact surface.
Figure 1.5 - Robustness of ROD-Tv [part 3]
The above figure illustrates a point rendering with quadrilateral as primitives. The surface saliency cutoff is set at 0.35. Compared to Figure 1.3, a large portion of noise is being removed. However, this saliency threshold is too low to reject all the noise point in point rendering. The surface normal of each token site can be visualized in a close up view.

Figure 1.6 - Robustness of ROD-Tv [part 4]
The above figure illustrates point rendering with a quadrilateral as primitives. The surface saliency cutoff is set at 0.53. Compared to Figure 1.3, all noise tokens are being removed. Compared to Figure 1.5, some true tokens are also being eliminated and we conclude that this saliency threshold is too high. Holds are produced during point rendering. The surface normal of each token site can be visualized close up.
Figure 1.7 - Robustness of Tensor Voting

We reconstruct a surface from a noisy/noise-free dataset with tensor voting. (A) A LUNG dataset with no noise (B) LUNG dataset with 25% noise (C) A LUNG dataset with 100% noise. Compared to the point rendering result, tensor voting is more noise robust.

Figure 1.8 - Robustness of Rod-Tv [part 5]

The original LUNG dataset (3,305 tokens) with 25% noise (759 tokens) is illustrated on the left. The figure on the right shows our result by means of the noise robust tensor voting algorithm. The surface reconstructed by tensor voting is much smoother than the point rendering one.
Figure 1.9 - Efficiency of ROD-TV (zoom-in part 1)
A surface reconstructed on the whole dataset is inefficient. To speed up the computation process, surface reconstruction on demand is proposed, so-called ROD-TV. In the figure, a zoom-in view of the LUNG dataset is shown, which only consists of 1,823 tokens (Full dataset is 3,305 tokens). The surface is reconstructed under high resolution.

Figure 1.10 - Efficiency of ROD-TV (zoom-in part 2)
This figure shows another zoom-in view of the LUNG dataset which consists of 1,805 tokens only. The high resolution surface can be found in the screen-space view.
Figure 1.11 - Efficiency of ROD-TV [zoom-out]

Compared to Figure 1.8 and Figure 1.9, we move the viewing position further away the dataset. A low-resolution surface is then constructed as shown in the above figure. The whole surface consists of only 232 tokens. The low resolution surface can be visualized in the close-up view.
Figure 2.1 - Algorithm grouping in both graphics and visions

Axis of View-Dependence – representation or description can be either viewer-centered or object-centered, axis of levels-of-detail axis – dataset can be represented in a single-scale or at multi-scale, axis of Primitive Connectivity – input points are fully connected to each other or isolated.
Figure 3.6 - Geometrically describe a second order symmetric tensor in 3-D space
Figure 3.10 - A generic second order symmetric tensor

It can be linearly decomposed into ball-like tensor, plate-like tensor and stick-like tensor
Figure 3.12 - Discrete version on the 3-D stick voting field
([left]: XY-plane; [middle]: YZ-plane; [right]: ZX-plane)
Figure 3.13 - Discrete version on the 3-D plate voting field
([left]: XY-plane; [middle]: YZ-plane; [right]: ZX-plane)
Figure 3.14 - Discrete version on the 3-D ball voting field

([left]: XY-plane; [middle]: YZ-plane; [right]: ZX-plane)
Algorithm 3.1 NormalVote( Votee, Voter )
A vector vote on the most likely normal direction is return.

\[ \mathbf{v} = \text{Votee.pos} - \text{Voter.pos} \]

/* case 1: voter and votee are at the same position */
if Votee.pos == Voter.pos then
  /* Vote is cast by Voter without decaying */
  StickVote.dir <- Voter.dir
  StickVote.mag <- Voter.mag
  StickVote.pos <- Voter.pos

/* case 2: voter and votee are on the straight line */
else if arg( Voter.dir, V ) = \Pi/2 then
  /* Vote is cast by Voter with decaying */
  StickVote.dir <- Voter.dir
  StickVote.mag <- exp( -(r^2+c^2)/(\sigma^2) ) [equation 3.4]
  StickVote.pos <- Voter.pos

/* case 3: voter and votee are connected with a HIGH curvature value */
else if arg( Voter.dir, V ) < \Pi/4 then
  /* Zero Vector */
  StickVote <- ZeroVector

/* case 4: voter and votee are connected with a LOW curvature value */
else
  /* Vote is cast by Voter in osculating circle */
  /* compute radius of osculating circle */
  radius <- 1/cos( arg( Voter.direction, V ) ) X Voter.dir
  /* compute center of osculating circle */
  center <- radius - V/2
  StickVote.dir <- center - Voter.pos
  StickVote.mag <- exp( -(r^2+c^2)/(\sigma^2) ) [equation 3.4]
  StickVote.pos <- Voter.pos
  and if
  return StickVote

Algorithm 3.1 - Operation to generate vector vote between voter site and votee site
Algorithm 3.2 TensorVote(Votee, Voter)

A stick vector vote on the most likely normal direction is generated by NormalVote routine (Algorithm 1). Plate and Ball tensor vote can be obtain by integrating the resulting Stick vector vote in 2-space and 3-space respectively.

```cpp
for all 0 ≤ i, j < 2 do
    TensorVote[i][j] ← 0
end for

/* Compute Stick Component */
stickSaliency ← Voter.lambdaMax - Voter.lambdaMid
if stickSaliency > 0 then
    Vote ← NormalVote( Voter, Votee )
    TensorVote ← Combine( TensorVote, Vote, stickSaliency )
end if

/* Compute Plate Component */
plateSaliency ← Voter.lambdaMid - Voter.lambdaMin
if plateSaliency > 0 then
    /* uniform sampling points in a circle */
    sample[i] ← GenRandomUniformPts()
    for all 0 ≤ k < sample.count do
        sample[k].dir ← sample[k].pos - Voter.pos
        VoterTransform ← Voter dot sample[k].dir
        Vote ← NormalVote( VoterTransform, Votee )
        TensorVote ← Combine( TensorVote, Vote, plateSaliency )
    end for
end if

/* Compute Ball Component */
ballSaliency ← Voter.lambdaMin
if ballSaliency > 0 then
    /* uniform sampling points in a sphere */
    sample ← GenRandomUniformPts()
    for all 0 ≤ k < sample.count do
        VoterTransform ← Voter * sample[k].dir
        Vote ← NormalVote( VoterTransform, Votee )
        TensorVote ← Combine( TensorVote, Vote, ballSaliency )
    end for
end if

return TensorVote
```

Algorithm 3.2 - Operation to generate tensor vote between voter site and votee site

Algorithm 3.3 Combine( TensorVote, StickVote, Weight )

Given a stick vote, this function first transforms it into a tensor, and then performs tensor addition.

```cpp
for all i, j such that 0 ≤ i, j < 2 do
    TensorVote[i][j] ← TensorVote[i][j] + Weight × StickVote[i] × transpose(StickVote[i])
end for
```

Algorithm 3.3 - Operation to combine weighted vector vote into current tensor vote
Given a set of tokens (1), which are all isolated points, we can construct a Hierarchical Bound Sphere using a bottom-up approach. It means that the finest resolution is constructed first and then next resolution and so on. (2) – (6) show the details on the bounding sphere construction in each resolution level. You may notice that the number of representative tokens reduces in each level. Details of the algorithm can be found in [Tong 02].
Figure 4.5 - The construction of Grid Pyramid in bottom-up fashion

(A) A sample scene in 2-D space is given. There are 12 tokens in the scene with a dimension of 8x8. (B) Building the highest resolution (level 3) in the Grid Pyramid. (No. of cells = 64 with size equals to 1) (C) Building the next resolution (level 2) of (B) in the Grid Pyramid. (No. of cells = 16 with size equals to 2) (D) Building the next resolution (level 1) of (C) in the Grid Pyramid (No. of cells = 4 with size equal to 4) (E) Building the lowest resolution (level 0) in the Grid Pyramid (No. of cells = 1 with size equal to 1, which is the dimension of the entire scene. (F) A tree representation of the Grid Pyramid. Note that we can achieve different resolutions by traversing the height of the tree. (G) It is a key of the tree, which indicates the ordering of the child cells. It is in anti-clockwise fashion starting at the lower-left child cell. (H) Another key of the tree which tells you the node type of the tree. An empty cell is labeled as incomplete and colored in white. A full cell is labeled as complete and colored in black. A partial cell labeled as incomplete and colored in grey. Empty = does not contain any token in a cell. Full = contain exact one token in a cell. Grey = contains more than one token and has child cells.
Figure 4.6 - Grid Pyramid operations: smoothing operator and quantization operator

(A) An intermediate level of construction on a Grid Pyramid is shown. The input is level k-1 (hi-resolution) of the Grid Pyramid and the output is level k-1 (lo-resolution) of the Grid Pyramid. This involves two operations, namely the smoothing operation and the quantizing operation. (B) Smoothing operator – the input is level k-1 which consists of 3 tokens and 4 grid cells. After the smoothing process, the output is smoothed level k-1 which contains 1 token and 4 grid cells. In this case, the black box on the smoothing process is averaging. (C) The quantizing operator – the input is smoothed level k-1 with 1 token and 4 grid cells. After the quantizing process, the output is level k with 1 token and 1 grid cell. In this case, the black box on the quantizing process is condensing 4 smaller cells into 1 big cell.
Figure 4.9 - The construction of Octree in top-down fashion

(A) A sample scene in 2-D space is given. There are 12 tokens in the scene with a dimension of 8x8. To build a Octree from Pyramid Grid, we need to take a look on the node type. There are three kinds of node types. They are complete, incomplete and semi-complete node. For complete node, if it contains exactly 1 token, then no further subdivision is carrying out on this node. For incomplete node, if it is a empty cell, then no subdivision is carrying out too. However, for remaining semi-complete node, since it contains more than 1 token, subdivision on the current node is needed. Complete node and incomplete node are classified as homogenous. Semi-complete node is classified as in-homogenous. (B) It shows the construction on the root level of Octree. It examined the root node which is an in-complete node, subdivision is carrying out and four children quadrants are produced. By repeating the subdivision progress from root to leaf as shown in (B) – (E). Finally Octree is produced. (F) tis is a tree representation. You can find that the number of cells is lesser compared to Pyramid Grid (Figure 4.5.F). There are 25 cells in Octree and there are 85 cells in Grid Pyramid. Therefore Octree is a compact data structure for hierarchical data representation.
Figure 4.11 - Data multi-resolution on the PregnantWoman dataset

(N) The original dataset, which consists of 41,836 data points. (A) The highest resolution level. (M) The lowest resolution level. (B) – (L) The transition levels between the highest and lowest resolutions.
<table>
<thead>
<tr>
<th>Multi-resolution level</th>
<th>Tree height</th>
<th>Cell size</th>
<th>No. of cells in Grid Pyramid</th>
<th>No. of cells in Octree</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>0.015625</td>
<td>$2^{12}x2^{12}x2^{12}$</td>
<td>41,580</td>
<td>68,719,435,130</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0.03125</td>
<td>$2^{11}x2^{11}x2^{11}$</td>
<td>41,575</td>
<td>8,589,893,014</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.625</td>
<td>$2^{10}x2^{10}x2^{10}$</td>
<td>41,540</td>
<td>1,073,700,284</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0.125</td>
<td>$2^9x2^9x2^9$</td>
<td>41,425</td>
<td>134,176,303</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0.25</td>
<td>$2^8x2^8x2^8$</td>
<td>40,929</td>
<td>16,736,287</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0.5</td>
<td>$2^7x2^7x2^7$</td>
<td>32,447</td>
<td>2,064,705</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>$2^6x2^6x2^6$</td>
<td>10,363</td>
<td>251,781</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2</td>
<td>$2^5x2^5x2^5$</td>
<td>2,790</td>
<td>29,978</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>$2^4x2^4x2^4$</td>
<td>712</td>
<td>3,384</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>8</td>
<td>$2^3x2^3x2^3$</td>
<td>178</td>
<td>334</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>16</td>
<td>$2^2x2^2x2^2$</td>
<td>48</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>32</td>
<td>$2^1x2^1x2^1$</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>64</td>
<td>$2^0x2^0x2^0$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3 - Experimental results on the PREGNANTWOMAN dataset

![Graph showing comparison between Grid Pyramid and Octree on the Pregnant Woman dataset](image)

Figure 4.12 - A comparison between Grid Pyramid and Octree on the PREGNANTWOMAN dataset
Figure 4.13 - Data multi-resolution on the MONKEYSADDLE dataset

(M) The original dataset, which consists of 19,209 data points. (A) The highest resolution level. (L) The lowest resolution level. (B) – (K) The transition levels between the highest and lowest resolutions.
Figure 4.14 - A comparison between Grid Pyramid and Octree on the MONKEYSADDLE dataset
Figure 5.6 - Hierarchical View Frustum Culling

A set of tokens and its bounding boxes are shown on the left (A). This scene is rendered with the hierarchical view frustum culling from the viewpoint of the camera. The corresponding tree representation of the input tokens is shown on the right (B). The root of the tree intersects with the frustum, and the traversal continues with testing its children. Since parent of the token 1 and 2 is fully outside the frustum, all of its children are being culled without any further testing. However, the bounding box of the token 6 is entirely inside the frustum, it is not culled and its sub-tree is added into potential visible set directly. For the intersection case, the parent of tokens 3, 4, and 5 intersects the frustum, a test is carrying out on its children until the leaf node is reached. If the leaf node is entirely inside the frustum, the node is not culled. If the leaf node is entirely outside the frustum, the node is culled. If the leaf node is intersects with the frustum, Token-Frustum Intersection Test is carrying out.
Figure 5.7 - General Idea on Occlusion Culling

The illustration on the left (A) shows a sample scene with six tokens. All of them are inside the frustum. However, some of them are invisible since they are being occluded by the others. To figure out the visible set, occlusion culling is performed. (C) An occluder set – a group of tokens we want to extract for further processing. (B) An occlude set – a group of tokens we want to cull after the occlusion test.
Figure 5.12 - Frustum Culling on the Teapot dataset [view point 1]
Figure 5.13 - Frustum Culling on the Teapot dataset [view point2]
Figure 5.14 - Occlusion Culling on the Teapot dataset
Figure 5.15 - Frustum Culling on the HAND dataset
Figure 5.16 - Visibility Test on the Hand dataset
Figure 6.1 - The fundamental concept of LOD

(A) A complex dataset is simplified. In high resolution, the dataset consists of 10,242 tokens. Nevertheless, it contains 272 tokens only in a less detailed representation. (B) Showing the general idea on the LOD. Given camera/viewer position, a highly detailed dataset is shown when its location is close to the camera/viewer. However, the dataset resolution decreases as the distance from the camera/viewer increases.
Figure 6.4 - Range-Based Method in LOD Selection

It shows that how range range-based LOD work. RANGE-0, RANGE-1 and RANGE-2 are user-defined value in the object space. If the object-viewer distance is less than RANGE-0, dataset with level 0 (named LOD 0) is rendered. Similarly, if the distance is greater than RANGE-0 but less than RANGE-1, then LOD 1 is rendered. The detailed level in the dataset decrease as the distance increases.
Figure 6.6 - Token neighborhood with different kernel size

(A) Input visible token set, where token $p$ is being shaded. (B) When we apply the voting kernel with size equal to 2-sigma, there are three tokens in $p$’s neighbourhood. (C) If we apply the kernel with size equal to 3-sigma, there are six tokens in $p$’s neighbourhood. Therefore, number of tokens in the neighbourhood is proportional to the kernel size.
Figure 6.11 - Experimental result on the surface connectivity of PLANE dataset

(A) Input is a PLANE dataset with 256 tokens. After visibility analysis, there are only 128 tokens remaining and we do surface reconstruction on them. (B) Surface connectivity with 0-neighbourhood (C) Surface connectivity with 1-neighbourhood (D) Surface connectivity with 2-neighbourhood (E) Surface connectivity with 3-neighbourhood
Figure 6.12 - Experimental result on HAND dataset (part 1)

(A) Original hand dataset with 38,220 tokens. (B) Initiating viewer position in 3-D space with view-object distance equal to 17.46 units. (C) After visibility analysis and levels of detail (LOD) analysis in hierarchical traversal, there are 6,029 tokens remaining.

[Remark: tree height 8 / 12 - rod level 12 - 8 = 4]
Figure 6.13 - Experimental result on HAND dataset (part 2)

(A) Surface reconstruction on visible tokens, which is generated from hierarchical traversal. It shows the result in world space. (B) Visible tokens (6,029 tokens out of 38,220 tokens) in screen space. (C) Visible tokens’ surface patches in screen space. There are 35,441 triangular patches in total.
Figure 6.14 - Experimental result on HAND dataset (part 3)
(A) zoom-in view on the visible tokens (B) close-up view of the visible tokens' surface (C) surface and its normal (D) surface and its triangular wire-frame

Figure 6.15 - Experimental result on HAND dataset (part 4)
(A) zoom-out a little bit on the visible tokens (B) Flat shading (C) Interpolative shading
Figure 6.16 - Experimental result on HAND dataset (part 5)

(A) Initiating viewer position in 3-D space with view-object distance equal to 112 units. (B) After visibility analysis and levels of detail (LOD) analysis in hierarchical traversal, there are 808 tokens remaining. (C) Only frontier surfaces are being rendered. All the occluded or back faces are culled. There are 7,281 surface patches as a result. (D) Another view of the visible tokens

Figure 6.17 - Experimental result on HAND dataset (part 6)

Rendering in (A) Tokens (B) Wire-Frame (C) Flat Shading (D) Interpolative Shading mode
Figure 6.18 - Experimental result on comparison between QSplat point rendering and ROD-TV in HAND dataset

(A) QSplat point rendering using sphere as a primitive (B) ROD-TV in low-resolution [view-object distance = 112 units and total number of surface = 7,281 triangular patches] (C) ROD-TV in high-resolution [view-object distance = 17.46 units and total number of surface = 35,441 triangular patches]