Converting Skeletal Structures to Quad-Dominant Meshes

J. A. Bærentzen\textsuperscript{a}, M. K. Misztal\textsuperscript{a}, K. Welnicka

\textsuperscript{a}Department of Informatics and Mathematical Modelling, Technical University of Denmark

Abstract

We propose the Skeleton to Quad-dominant polygonal Mesh algorithm (SQM), which converts skeletal structures to meshes composed entirely of polar and annular regions. Both types of regions have a regular structure where all faces are quads except for a single ring of triangles at the center of each polar region. The algorithm produces high quality meshes which contain irregular vertices only at the poles or where several regions join. It is trivial to produce a stripe parametrization for the output meshes which also lend themselves well to polar subdivision. After an initial description of SQM, we analyze its properties, and present two extensions to the basic algorithm: the first ensures that mirror symmetry is preserved by the algorithm, and the second allows for objects of non-spherical topology.

1. Introduction

In 3D graphics skeletal structures are often used as armatures for articulating 3D models, and, for this purpose, they play a prominent role since skeletal subspace deformation \cite{1} is one of the most widely used techniques for animation of characters or creatures. However, the skeletal representation is also an excellent shape abstraction: edges correspond to parts, and nodes define junctions where these parts meet.

In order to design a shape, we only need tools to create a node (attaching it to a parent node), and to move a node. For a skeleton based modeling tool we would also soon realize the need for, say, an edge splitting tool, or a tool that changes the size of a node. Nevertheless, even allowing for these additional tools, we maintain that it is straightforward to design and implement a skeleton based modeling system. If we add to such a system the capability to convert the skeletal representation to a polygonal mesh, i.e. inverse skeletonization, the result is a very effective tool for creating coarse shapes. A polygonal mesh editing system might be more flexible and allow for modeling of details, but if the goal is to rough out a base shape from scratch, designing a skeletal structure is simpler than directly creating a corresponding polygonal mesh.

We are not the first to come to this conclusion. Indeed the Z-Spheres tool in the commercial application Z-Brush allows for modeling a 3D shape as a skeletal structure, and a similar paradigm was recently explored in \cite{2}. In both cases, the tools are intended for modeling of coarse shapes to be used as the starting points for sculpting of details using completely different techniques.

Of course, interactive modeling of coarse shapes is by no means the only application of inverse skeletonization. It is much easier to procedurally generate a skeleton than a surface representation of a shape. Indeed, \textit{L-Systems}, which are widely used to create models of plants \cite{3}, produce what is in effect a textual representation of a skeleton.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{fig1.png}
\caption{The SQM algorithm proposed in this paper produces meshes which are composed only of regions which can be flattened into polar or annular regions as shown above left. The algorithm computes the number of radial subdivisions such that the regions can be stitched together without additional faces or T-junctions. For the skeleton (middle) we create an annular and three polar regions as shown on the right.}
\end{figure}

1.1. Contributions

In this paper, we propose a novel solution for inverse skeletonization which we call the \textit{Skeleton to Quad-dominant polygonal Mesh} (SQM) algorithm. SQM produces a mesh composed exclusively of polar and annular regions (c.f. Figure 1). To ensure that these regions can be stitched together without introducing glue faces or T-junctions, we need to solve for the number of radial subdivisions in each region. These numbers are found as the integer solution to a set of linear equations which depend on the entire skeleton.

Benefits of the SQM algorithm include that it produces meshes, which have few irregular vertices compared to other work (c.f Section 5) and which are easily stripe parametrizable \cite{4}. An example of a mesh produced by the algorithm is shown in Figure 2.

2. Related Work

In rather early work, Jules Bloomenthal \cite{5} demonstrated how to model a (botanical) tree, specifically a maple. Here the
Figure 2: We propose an algorithm (SQM) which converts a skeletal structure (left) to a quad dominant polygonal mesh (right). The meshes produced lend themselves well to polar subdivision schemes, and they are trivial to imbue with a stripe parameterization as illustrated on the right where the mesh is shown subdivided and textured.

geometry of bifurcations (and only bifurcations) is modelled explicitly.

More recently, Srinivasan et al. [6] propose an interesting method where general wireframe models are turned into 3D meshes. This is done by defining a cross sectional profile for each tubular segment. The branch node geometry is created by finding the convex hull of these profiles at the point where they connect with the branch node. In a recent paper, Ji et al. [2] introduce B-Mesh, a system where the conversion procedure is very similar to [6] except that the swept profile is always a quadrilateral. We emphasize that the approach in [6, 2] is almost the opposite of ours, since we first create geometry for the junctions and then create the connecting tubes as described in detail later.

Several authors propose to convert a graph to a mesh by iteratively combining cuboid elements [7, 8, 9]. Taking another road, Vaillancourt and Egli [10] used implicit surfaces for the purpose of creating placeholder geometry for games from skeletal structures.

Since SQM produces a stripe parametrization along with the mesh, it worth noting that Kalberer et al. [4] discuss precisely how to stripe parametrize tubular structures given a triangle mesh as input.

3. The SQM Algorithm

Previous researchers have proposed algorithms where tubular parts are first created independently. Then these parts are stitched at branch nodes. This is arguably simpler than our algorithm, but the stitching requires additional faces which serve as “glue” geometry.

A central idea of the SQM algorithm is to create tubes for each vertex that can be stitched together at branch nodes with no such glue. To do so, we first create a polyhedron for each branch node. These polyhedra are then refined to ensure that we can bridge them with tubes consisting of a single loop of quadrilaterals.

The algorithm can be described as a four step procedure. The four steps are illustrated in Figure 3 and explained below.

Algorithm Input

The input to the algorithm is a skeleton, i.e. a tree with a geometric position associated with each node. Each node also has a user defined radius which is used to define the size of the shape at the node.

Algorithm Step #1: BNP Generation

To simplify Step #3 of the algorithm, we first straighten the edges incident on a connection node by rotating the edge to the child node such that it is parallel with the edge from the parent node. Note that the connection nodes are not removed.

The first part of the algorithm is the creation of branch node polyhedra (BNP). For a branch node in the skeleton, its BNP is a polyhedron whose vertices correspond to the paths (which we recall have been straightened) incident on the node. In practice we compute it by intersecting the paths with the sphere associated with the node. The intersection points are then triangulated. Thus, vertices of the BNP correspond to branch-free paths leading either to other branch nodes or to leaf nodes. We will denote the vertices path vertices. Several BNP are shown in Figure 4.

Triangulating the path vertices could be done in a number of ways, but Delaunay triangulation [11] of points in a spherical domain is straightforward [12] and produces the same mesh as a convex hull algorithm [12]. We adapted the simple algorithm by Bourke [13] to the spherical domain by modifying the in-circle predicate.

The branch node polyhedra are subdivided by inserting a vertex on the middle of each edge and the center of each face. The face vertex is then connected to all vertices of the face thereby dividing the containing triangle into six triangles. This subdivision serves to separate path vertices so that a path vertex does not have another path vertex in its link (i.e. the set of vertices which share an edge with v and the edges which connect these vertices, forming a loop around v). Conversely, all triangles are now incident on precisely one path vertex. Thus, the newly in-
serted vertices, which we denote node vertices, delimit the part of the BNP that belongs to a single path vertex.

The inserted node vertices are projected onto the sphere associated with the node, producing the result illustrated in Figure 4b. Without this projection some BNP would have little or no volume. For instance, the BNP arising from a node with three incident edges consists of just two triangles, and that would become an issue when creating the tunnels in Step 3 of this algorithm.

**Algorithm Step #2: BNP Refinement**

Given a pair of BNP whose nodes are connected via a path (that does not contain branch nodes), we can create a tube between these two BNP by removing the path nodes and their incident triangles at either end and joining the holes with a tubular mesh. Unfortunately, the path vertices could have different valencies, and then it is impossible to make a tube of quadrilaterals alone. Therefore, the purpose of BNP refinement is to make the valencies equal by splitting edges in the link of the vertex with lower valency until the valencies are equal (c.f. Figure 4c).

We will introduce the notion of Link Intersection Edges (LIE). An LIE is simply a set of edges in a subdivided BNP which belong to the links of two path vertices as shown in Figure 4b&c (bold cyan lines). By splitting an arbitrary edge of the LIE we increase the valencies of both the associated vertices by one. A very simple smoothing scheme is used to equalize the lengths of edges in a given LIE.

An LIE can be specified by a triple index \( \langle ijk \rangle \) where \( j \) is the node to whose BNP the LIE belongs, the intersection is between the links of path vertices corresponding to paths \( ij \) and \( k j \). The LIE is obviously an empty set in cases where the links are disjoint.

For a given path, \( ij \), we can describe the equality of vertex valencies as \( E_{ij} = 0 \) where

\[
E_{ij} = \sum_{k \in L_{ij}} L_{jit} - \sum_{k \in L_{ij}} L_{ijk},
\]

(1)

where \( L_{ijk} \) is the number of edges in the LIE given by \( \langle ijk \rangle \). This number is initially two for any LIE after subdivision and it increases by one for each split in the refinement process. \( L_{ij} \) is the set defined such that node \( k \in L_{ij} \) if there is a graph edge \( jk \) and the links of the path vertices on the BNP of node \( j \), which correspond to the paths to \( i \) and \( k \), have non-empty intersections.

For a path vertex corresponding to node \( j \) and incident on the \( ij \) path: if \( E_{ij} > 0 \) then \( E_{ij} \) tells us how many splits are needed in the link of the path vertex in order to make \( E_{ij} \) zero. The need \( N_{ijk} \) for refining LIE \( \langle ijk \rangle \) is simply the sum of splits (clamped at zero) required for both the paths to which the LIE belongs.

\[
N_{ijk} = \max(0, E_{ij}) + \max(0, E_{kj}).
\]

(2)

Our algorithm for simultaneously satisfying all equations of the form (1) always refines the LIE with the greatest need. In the following algorithm, let \( \langle \ast \rangle \) be the selected LIE.

**Refinement Algorithm:**

1. We select an arbitrary LIE denoted \( \langle \ast \rangle \).
2. We then visit all LIEs and for each, say \( \langle lmn \rangle \), we replace \( \langle \ast \rangle \) with \( \langle lmn \rangle \) if one of the following two conditions are true:

![Figure 3: The skeleton shown in (a). In step 1a polyhedron (BNP) is created and subdivided for each branch node leading to the result in (b). In step 2, the BNP are refined (c), and in Step 3 tubes consisting only of quadrilaterals are created between the BNP. Vertices are also created for connection nodes (d). In Step 4 the vertices are given their final positions (e).]
(a) The need is greater: $N_{\text{num}} > N_v$
(b) The need is equal: $N_{\text{num}} = N_v$ and $L_{\text{num}} < L_v$. In other words, we prefer to refine less refined LIEs if the need is the same.

3. When all LIEs have been tried, we split an edge in $(\ast)$ if $N_v > 0$. The new vertex is projected onto the sphere associated with the node.

4. Go to 1 unless $N_v = 0$.

Algorithm Step #3: Creating the tubular structure
All vertices which are connected via a path now have identical valencies and we can remove the incident triangles and bridge the holes via a tube consisting exclusively of quadrilaterals that connects the links of the two former vertices. Since we initially straightened the mesh, the tubes are also straight, and to form a tube, we simply pick the quadrilaterals whose edges (those that align with graph edge) have the smallest sum of square lengths. Thus, all faces of the original BNP are removed except those incident on path vertices corresponding to leaf nodes. These faces remain triangles and form the polar regions. The faces replaced by quad-only tubes form the annular regions.

All mesh edges are now subdivided to take the connection nodes into account. If a mesh edge corresponds to a path between two branch nodes, we split the edge into a number of segments equal to the number of skeletal edges on the corresponding path. The vertices are given a preliminary position by pushing them to a distance from their node corresponding to the node radius. Finally, we split the faces by connecting vertices that correspond to the same connection nodes. When this is done, we have obtained the final mesh connectivity.

Algorithm Step #4: Vertex Placement
After step 3 we have preliminary vertex positions (as shown in Figure 3d). We apply the inverse of the rotation used to straighten the mesh to bring these positions back to the original pose. However, by balancing attraction to a skin around the skeleton and smoothing, we can obtain a more controllable and taut mesh. We therefore employ three iterations of a simple smoothing and attraction scheme.

The smoothing is Laplacian smoothing [14], but after each iteration, vertices belonging to the same node are moved so their barycenter coincides with the node. The attraction step moves vertices toward a skin around the model. This skin is given by the cones which precisely contains the spheres associated with the incident nodes [15]. Clearly, this procedure is heuristic based and other procedures could be envisioned. That being said, we find that it works very well for the organic shapes we mostly target in this paper. A single parameter allows the user to adjust the amount of smoothing vs skin attraction.

4. Algorithm Analysis and Extensions

SQM produces meshes with a highly regular structure since the geometry consists only of annular and polar regions. In some ways this collection of polar and annular charts is different from what is easily created with an interactive polygon modeling tool. Designers using such tools often model using irregular vertices denoted N-poles (valency 3) or E-poles (valency 5) [16]. Given a regular quad mesh region where one extrudes out a single face, the result is four E-poles around the base of the extrusion and four N-poles at the tip. In contrast, SQM does not extrude from regular regions but joins tubular parts, and the vertices where these join have a valency that is precisely two times the number of tubular parts that join (often we have valency six corresponding to three tubes meeting).

In Figure 5 we have shown the structure of an SQM generated mesh on two models. Red curves bound the polar and annular regions while the blue curves flow from the poles to other poles or terminate where more than two tubular regions join. As explained the meshes have a good edge flow, which is well suited for subdivision, and in particular subdivision using polar schemes as discussed below.

4.1. Graphs with Loops

A limitation of our algorithm is the fact that it is restricted to tree structures as opposed to general graphs which may have loops. However, this limitation can always be overcome by the simple expedient of cutting the loops, e.g. by removing a single edge from every loop until the graph is a purely branching structure. After the mesh has been generated we can then close the gap by creating a bridge corresponding to the removed edge using a method similar to the one employed in Step 3 as illustrated in Figure 6(a,b). It is only possible to close the loop with a quad only connection if the valencies of the bridged vertices are equal, but clearly we can bridge any pair of vertices by removing the faces in their one rings and creating a tunnel consisting of both quads and triangles.

4.2. Symmetry and Mesh Quality

Our algorithm does not necessarily produce a symmetrical mesh even if the input skeleton is symmetric. To remedy this...
issue, we have included explicit mirror symmetry information in the interactive modeling tool used to test this work which means that we can in practice guarantee at least mirror symmetry. When an LIE is selected for refinement in Step 2, we look up the symmetrical counterpart for the nodes that determine the LIE. From these symmetrical nodes, we can find the symmetrical LIE and refine that in the same way.

The BNP produced by Delaunay triangulation can sometimes lead to surprising meshes. This is illustrated in Figure 6c where a mesh is created from the skeleton of a paw. Observe that the “ring” finger appears to be behind the other finger. The reason is that this digit corresponds to a BNP vertex not directly connected to the vertex which corresponds to the path towards the root node. By flipping edges whose dihedral angle is less than 45$^\circ$ if the flipped edge has the path vertex corresponding to the path to the parent node, we can obtain a much better edge flow as illustrated in Figure 6d.

The Delaunay triangulation in Step 1 produces symmetrical BNP from symmetrical skeletons unless four or more points are co-circular since these configurations have non-unique triangulations. We solve this problem by merging triangular faces if they are nearly co-circular. Thus, N co-circular points produce an N-gon rather than N-2 triangles. These N-gons represent points where N tubular parts join; observe how in Figure 6e – where no merging of triangles has taken place – the branches meet in a rather haphazard fashion whereas in Figure 6g the branches meet in a symmetrical fashion. Figure 6f and Figure 6h show the corresponding BNP.

4.3. Subdivision and Parametrization

The leaf nodes of a skeleton used as input to our algorithm always give rise to polar regions of the mesh, i.e. a single triangle fan surrounded by one or more rings of quadrilaterals. Such structures are not ideally suited for Catmull Clark (CC) subdivision since ripples (unwanted curvature variations) can form near the pole. To address this issue we have implemented the polar C2PS scheme [17] (admittedly slightly simplified). C2PS is an extension of Catmull-Clark which has the benefit that is C2 even at the poles. Away from the poles, the scheme is identical to Catmull-Clark.

It is straightforward to generate a periodic parametrization of the meshes produced by the algorithm if we simply let each face of the original (unsubdivided) mesh correspond to a parameterization domain $(u,v) \in [0,1] \times [0,1]$ in such a way that the $v$ coordinate corresponds to the direction along the corresponding graph edge, and $u$ to the perpendicular direction. In the case of triangle faces, we can simply regard the vertex corresponding to the leaf node as two collapsed vertices. Thus, we let the triangles correspond to the same domain as the quadrilaterals but pinched together at the pole vertex. See Figure 6i (right).

5. Results

This project was implemented in C++ using a half-edge based data structure to represent the mesh [14]. Our half-edge library is quite general purpose, and the operations required

Figure 6: The model shown in (a) has its genus changed by bridging some vertices corresponding to symmetrical leaf nodes (b). If we flip edges of the BNP to promote edges that connect BNP vertices with the vertex corresponding to the path towards the root node, we obtain a much better edge flow on the final mesh (c,d). By merging faces of the BNP to form N-gons (compare f to h), we can have N tubes meeting in a central vertex greatly promoting symmetry (compare e and g). Polar subdivision and stripe parametrization are illustrated in (i).
(edge flip, face split, edge split, one-ring merge, and face bridging) have many uses and are found in a number of other mesh libraries. OpenGL and Blender were used for interactive and off-line rendering, respectively.

We have created an interactive 3D modelling system that allows a user to create and edit a skeleton representation of a shape. The user can interactively add nodes either by splitting edges or connecting to a parent. It is also possible to change the radii of nodes and to move existing nodes. The system includes a mirror-symmetry mode.

Performance

It is clearly interesting to measure the performance of the algorithm and the quality of the meshes produced. Performance numbers are shown in Table 1. The most important thing to point out is that the skeleton to mesh procedure runs in a fraction of a second for all of the interactively created skeletons. For the large models, none of which are created interactively, Step 3 dominates the run time because we compute paths between all pairs of nodes. That was quite useful during the initial development of the algorithm, but now this step is an obvious candidate for optimization since we only need the distances between adjacent branch nodes.

Mesh Quality

Regarding mesh quality, we compare our results to modelling with implicits (metaballs), Z-Sphere in Z-Brush from Pixologic and the B-Mesh system by Ji et al. [2]. Results using implicits and Z-Spheres are shown in Figure 7 (top, left). The implicit surface polygonizer clearly produces a mesh which does not in any way reflect the graph structure. However, Z-Brush produces a mesh which aligns well with the skeleton, but in some cases the edge flow is surprising. For instance in Figure 7 (top, right) the left arm does not join well with the rest. As observed in [2] it is the responsibility of the user to resolve these issues where needed.

The most interesting system to compare with is B-Mesh described in [2]. The authors of that paper present their system as a tool for interactive modelling with the goal of producing quad dominant base meshes (for further editing using e.g. displacement maps) with good edge flow. However, their method introduces junction geometry by design: i.e. some faces (triangles or quads) cannot be related to a particular edge but to a branch node. Whereas in our system, the irregular vertices are either poles corresponding to leaf nodes or vertices where more than two tubular structures join, the B-mesh system introduces several irregular vertices where more than two tubular structures join as illustrated in Figure 7 (middle and bottom). For instance, compare the junction between tail, right leg and body in the middle left and middle right images. An analysis reveals that B-Mesh produces three to four times more irregular vertices than SQM (even including poles) for a very comparable number of total vertices. This is perhaps not surprising since at the tips of the skeletons SQM produces a single vertex of high valence whereas B-Mesh generates four valence 3 vertices. B-Mesh also produces some triangular faces that give rise to up to four irregular vertices after a single step of subdivision.

<table>
<thead>
<tr>
<th>Valency</th>
<th>Cat SQM</th>
<th>Antcat SQM</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>1016</td>
<td>1550</td>
</tr>
<tr>
<td>irregular</td>
<td>98</td>
<td>124</td>
</tr>
</tbody>
</table>

6. Conclusions & Future Work

Clearly, there are many artists who could easily have created excellent quad dominant polygonal meshes for the models shown in this paper, but a skilled 3D artist’s time is precious. Our hypothesis is that, for many shapes, the structure of a good polygonal mesh can be derived from the connectivity of its parts. The proposed method supports this hypothesis since it produces meshes with a very simple edge flow and few irregular vertices. The method is easily extended to produce a stripe parametrization, and the meshes lend themselves particularly well to polar subdivision. Utilizing automatic skeleton to

Figure 7: Metaballs were used to create the model shown in the top, left image. ZSpheres in ZBrush were used for the character on the top, right. The cat model (middle) and antcat (bottom) were created using B-Mesh. In the images on the left, B-Mesh was also used to generate the mesh from the skeleton shown on the left. In the images on the right, our method (SQM) was used. Note that mesh geometry cannot be directly compared since different methods for vertex placement are used.
Table 1: This table shows statistics for most of the models in this paper. From left to right the table contains the name, figure where shown, number of nodes produced by SQM, and timings for each step of the algorithm and the total run-time.

<table>
<thead>
<tr>
<th>Model</th>
<th>Fig</th>
<th>total</th>
<th>leaf</th>
<th>conn</th>
<th>brnch</th>
<th># vert.</th>
<th>1, BNP gen.</th>
<th>2, BNP ref.</th>
<th>3, tubular</th>
<th>4, v. place.</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dragon</td>
<td>5</td>
<td>127</td>
<td>37</td>
<td>70</td>
<td>20</td>
<td>653</td>
<td>0.0027</td>
<td>0.003</td>
<td>0.024</td>
<td>0.053</td>
<td>0.083</td>
</tr>
<tr>
<td>Vehicle</td>
<td>1</td>
<td>49</td>
<td>16</td>
<td>27</td>
<td>6</td>
<td>310</td>
<td>0.0011</td>
<td>0.00096</td>
<td>0.0026</td>
<td>0.025</td>
<td>0.03</td>
</tr>
<tr>
<td>Antcat</td>
<td>7</td>
<td>60</td>
<td>12</td>
<td>42</td>
<td>6</td>
<td>310</td>
<td>0.00085</td>
<td>0.00053</td>
<td>0.0039</td>
<td>0.025</td>
<td>0.031</td>
</tr>
<tr>
<td>Cat</td>
<td>7</td>
<td>42</td>
<td>10</td>
<td>27</td>
<td>5</td>
<td>248</td>
<td>0.00063</td>
<td>0.00072</td>
<td>0.0019</td>
<td>0.02</td>
<td>0.024</td>
</tr>
<tr>
<td>Pipes</td>
<td>2</td>
<td>491</td>
<td>26</td>
<td>444</td>
<td>21</td>
<td>2231</td>
<td>0.0023</td>
<td>0.0016</td>
<td>1.5</td>
<td>0.19</td>
<td>1.7</td>
</tr>
<tr>
<td>Goat</td>
<td>2</td>
<td>128</td>
<td>26</td>
<td>92</td>
<td>10</td>
<td>894</td>
<td>0.0018</td>
<td>0.0022</td>
<td>0.028</td>
<td>0.073</td>
<td>0.11</td>
</tr>
<tr>
<td>Tree</td>
<td>6</td>
<td>953</td>
<td>239</td>
<td>593</td>
<td>121</td>
<td>6053</td>
<td>0.027</td>
<td>0.11</td>
<td>7.5</td>
<td>0.5</td>
<td>8.1</td>
</tr>
<tr>
<td>Dog</td>
<td>6i</td>
<td>28</td>
<td>8</td>
<td>17</td>
<td>3</td>
<td>140</td>
<td>0.00047</td>
<td>0.00027</td>
<td>0.0085</td>
<td>0.011</td>
<td>0.013</td>
</tr>
<tr>
<td>Armadillo</td>
<td>7</td>
<td>69</td>
<td>18</td>
<td>44</td>
<td>7</td>
<td>521</td>
<td>0.0013</td>
<td>0.0017</td>
<td>0.0062</td>
<td>0.042</td>
<td>0.051</td>
</tr>
</tbody>
</table>

There is still some work ahead. It is well known that artists sometimes introduce irregular vertices around anatomical joints, and our method does not reflect that. Also, shapes with concavities or parts that are not tubular (e.g. heads) are not yet supported. However, we believe the algorithm is extensible to an even broader class of shapes, and we are presently working on generalizing it. Another obvious next step (which we have started to take as shown in Figure 8) is to apply the algorithm as the mesh generation back-end to various methods for interactive and procedural shape generation.

Acknowledgements

We thank Zhongping Ji for providing us with skeletons created using B-Mesh. This work was partially funded by a grant from the Danish Agency for Science, Technology and Innovation.

References


