Abstract

In the article an idea for a novel way of mapping of textures onto a surface of 3D model is introduced. Our technique is based on two interlocking mappings. The first one maps surface vertices onto a computed skeleton and the second one maps the surrounding area of each skeleton segment into a rectangle with size based on the surface properties around the segment. Furthermore, these rectangles are packed into a squared texture called skeleton texture map (STM) by approximately solving a palette loading problem. Our technique enables the mapping of a texture onto the surface without necessity to store texture coordinates with the model data and it is also suitable for surfaces with a topology non-homotopic to a sphere with higher order genus and unlimited structure branching.

CR Categories: Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: skeleton, texture, mapping, parameterization

1 Introduction

Texture mapping is a commonly used and the most successful technique of improving visual quality of 3D surfaces in computer graphics. The problem of texture mapping can be formulated as retrieving a pair of texture coordinates for each surface point in order to put the surface into one-to-one correspondence with an image.

Our motivation behind this approach is to parameterize a model using a piece-wise easily parameterizable guiding structure that is dependent only on the topology of the model, not the surface. Moreover, such a parameterization would be used for generating of texture and geometry patterns around the skeleton. The textures encoding these patterns can be generated independently to the geometry and later applied on a set of models with the same skeleton branching. In a case, when the texture encodes a geometry, it can be used for procedurally generating of geometry around given skeleton tree. In addition, the parameterization would enable transferring of patterns between models with the same skeleton branching.

In our approach the mapping between the surface and the skeleton works in a deterministic way and the benefit of our approach is that \( uv \) coordinates can be computed during rendering process. In the first stage, a skeleton is extracted from an input mesh. For skeleton extraction, the algorithm from [Au et al. 2008] with minor modifications is used. The algorithm extracts the skeleton from a closed 2D-manifold mesh using iterative Laplacian contraction. If the input is a surface with boundaries, polygon soup or point cloud, we can use the adaptation of this algorithm [Cao et al. 2010] for handling such surfaces. During the skeleton construction we store the mapping of mesh vertices collapsed into each skeleton node. Furthermore, we compute and store some surface properties of the collapsed vertices into the corresponding skeleton node. These properties serve for priority weight estimation. These weights determine the importance of segment in a sense of level of detail and control how much of space the segment requires in the final texture. With the knowledge of relative size ratio between the segment textures, we solve the packing problem and allocate an area in the final texture for each segment. In the next step, we apply segment mapping and cover allocated area for each skeleton segment. To avoid aliasing and waxy look of textures a filtering has to be applied in the texture mapping. To guarantee that each texel has a well defined local neighborhood, each rectangular texture is enclosed in horizontal direction by mirrored parts of itself and in the vertical direction by texture parts of the neighbouring segments. A graph demonstrating a workflow of our algorithm can be seen in Figure 1. The proposed technique is also suitable for surfaces with topology non-homotopic to a sphere, higher order genus and very complex branching structure, where common mapping techniques often based on unwrapping mesh into 2D domain may fail (see Figure 2). The reason that our method handles this problem is that the texture is remapped into several textures, thus giving each segment its own texture space.
Figure 2: An automatic atlasing technique fails - polygons either overlap (left) or are distorted (right) if the branching topology in skeleton joint is too complex.

2 Related Work

Texture Mapping Some techniques have been developed for an automatic mapping of textures onto the model surface. An automatic planar parametrization for surfaces with disk topology based on unfolding the polygon mesh has been concerned in some works [Sheffer and Sturler 2000], [Lévy et al. 2002], [Floater 2003], while on unfolding the polygon mesh has been concerned in some works [Tirani et. al [Tarini et al. 2004] used cubes around the model to map the texture. A limitation of the poly cubes is that they got fixed level of detail over the surface. Another, more limiting drawback is that they encode only the surface, not the topology, therefore they are not general enough to encode topology of an arbitrary model. The shape is composed of axis-aligned unit cubes that are attached face to face. Such a configuration is limiting to branching of model topology, because each cube may have maximum of six neighbors. An automatic construction of polycube maps was introduced in [He et al. 2009] using a divide and conquer strategy. Wang et. al [Wang et al. 2007] made the construction of polycubes to be independent of actual geometry of 3D shape allowing different complexity and resolution for the polycube. Lin et. al [Lin et al. 2008] constructed polycube map by segmenting the mesh into a set of patches and then approximating these patches by basic polycube primitives.

A general goal in texture mapping is to get balance between the seams and texture distortion along with solving the problems with triangle overlapping. If the texture mapping domain is subdivided into too much components, the local neighborhoods of texels do not exist and problems with filtering, estimation from displacement map and generating of procedural patterns may occur, because definition of the transaction function may be too difficult. If the number of components is to small, mapping distortion and triangle overlapping problems come into the view. Skeleton segmentation is a natural topology-driven way of cutting a model into a set of components, thus overcoming the above problems.

Skeleton Extraction Numbers of algorithms have been proposed to compute a skeleton from an input mesh geometry. In [Shapira et al. 2008] authors proposed skeleton extraction based on a shape diameter function (SDF). The SDF is a scalar function defined on the mesh surface that expresses a measure of the diameter of the object’s volume in the neighborhood of each point on the surface. Thus, a set of random vertices is pushed in an inward normal direction into the volume of the model by a distance that equals to half of the SDF and a least-squares method is used to fit a high-degree curve into the shifted points. Similar approach was used in [Liu et al. 2003], where authors used so called repulsive force field. Sharf et. al [Sharf et al. 2007] introduced a method that is able to perform skeleton extraction on both, point clouds and polygonal meshes. The method uses evolution of a deformable model inside of the mesh. The initial extracted graph is noisy, and extraction of final skeleton require further filtering and merging.

Reeb graph based methods need a suitable real-value function, defined on the model surface for a successful extraction of a skeleton. Using this function, nodes of a 1D graph can be computed. In [Hilaga et al. 2001] a geodesic function was used for Reeb graph extraction. Alternatively, a method based on a harmonic function proposed by Aujay et. al [Aujay et al. 2007] captures after resampling all the features of the model well, but requires the user to specify the boundary condition explicitly.

Another approach is to apply a well defined filter on mesh vertices. In the first step, an input mesh is contracted using iterative Laplacian contraction. Then, a mesh decimation is used to simplify the contracted mesh into a curve-skeleton. Cao et. al [Cao et al. 2010] extended the idea of Laplacian contraction [Au et al. 2008] for a point cloud input. They used a definition of the Laplacian operator for a point cloud in order to perform a similar weighted filtering. When the mesh is contracted, mesh decimation cannot be used, because there is not defined an edge connectivity. Authors made selection of contracted points to be connected based on their euclidean dis-
3 Skeleton Texture Mapping

Skeleton texture mapping works as follows. First, a skeleton is extracted from an input mesh (Section 3.1). The skeleton is used for a segmentation of the model in a topology-driven way and to define the axis of each segment. A rectangular area in the final texture is reserved for each segment by solving a packing problem (Section 3.2). Then, geometry around each skeleton segment is mapped into a rectangular texture using capsule parameterization (Section 3.3).

A global STM coordinates are computed by shifting each rectangle from local rectangle coordinates into the positions precomputed by segment packing algorithm. During the mapping, topological constraints resulting into seams are resolved by doubled rendering of triangles that lie on the seams (Section 3.4). Finally, local neighborhoods of each texel is guaranteed by mirroring and boundary color duplication (Section 3.5). In the end, a discussion on robustness of our method can be found (Section 3.6).

3.1 Skeleton Extraction

For our method the Laplacian smoothing based extraction is the best choice, because the skeleton nodes are created by shrinking the original mesh vertices into the models volume and merging into skeleton nodes. Therefore, these methods give us the one-to-many mapping between skeleton segments and mesh vertices for free. Using some other method, it would require to find the association after the skeleton extraction.

That is why we have chosen [Au et al. 2008] as the base of the skeleton extraction algorithm. The algorithm runs in two steps. In the first step, the input mesh is contracted using an iterative Laplacian smoothing in few iterations. In each iteration, a Laplacian operator is constructed for each vertex and applied on the mesh. When the iteration process converges, the contracted mesh is homotopic to the original mesh and has almost zero volume. In the second step, the skeleton is constructed by simplifying the contracted mesh using a skeleton decimation algorithm [Garland and Heckbert 1997] with one change. The error matrices are computed over the edges, because the contracted mesh has zero area faces, so the original volume based algorithm cannot be used.

In addition, in the skeleton construction stage, we have to guarantee that each skeleton segment lies inside the mesh volume. Every iteration of the greedy algorithm, which is used for mesh simplification, we check if merging the vertex into another one does not move the skeleton segment out of the volume. If such a situation is detected, we forbid the merging of these vertices and the greedy algorithm jump to next pair of vertices with the lowest cost estimation.

3.2 Packing Problem

Given the skeleton segments and geometry associated with the segments, we map each segment triangle into a rectangular texture and pack all these rectangles into one final square texture. A rectangular texture with higher priority needs better storing of details, hence it will be stored in a bigger area than rectangle with lower priority. Determining relative size ratio between the rectangles, we can formulate the storing of these rectangles as a 2-dimensional distributor’s pallet loading problem [Virgin et al. 2010] of storing \(N\) rectangles with size \((R_i^W, R_i^H)\) into a unit square. Thus we maximize the sum of the area they cover (Equation 1) and we can scale them by a global constant \(s \in \mathbb{R}^+\).

\[
\arg\max_s \sum_{i=0}^{N-1} s^2 R_i^W R_i^H \leq 1. \tag{1}
\]

We use a binary search to find the correct scaling ratio \(s\) to fit the rectangles. The placing of the rectangles is done using a k-D tree and a recursive fitting (Figure 3). We explore the configurations for assigned \(s\) and if there is an acceptable configuration, we increase \(s\) and iterate the fitting again. If there is no next acceptable configuration, we take the last one as the solution.

\[\begin{array}{c}
\begin{array}{c}
7 \ 6 \ 5 \ 8 \\
2 \ 4 \\
1 \ 3
\end{array}
\end{array} \]

(a)

(b)

Figure 3: Remapped rectangular textures are packed into the squared texture (a) using a k-D tree (b).

3.3 Segment Mapping

A capsule around each skeleton segment is mapped into a rectangle, the central part uses cylindrical coordinates and two ends use the mapping of spherical caps as shown in Figure 4. Furthermore, the relative texture coordinates inside each rectangle are computed as in Equation 2 and Equation 3:

\[
u = \begin{cases} d(1 - \frac{\theta}{\sigma^2}) & t < 0 \\ d + t(w - 2d) & t \in [0, 1] \\ w + d(\frac{\theta}{\sigma^2} - 1) & t > 1. \end{cases} \tag{3}
\]

where \(w\) and \(h\) determine the size of the rectangle and \(d\) influences the area that is reserved for the part of the texture that is encoding the capsule caps. Cylindrical coordinates describing the central part are \((\phi, t)\) and spherical coordinates describing caps are \((\phi, \theta_i)\).

For each skeleton segment priority weights are computed from the surface area ratio and the segment length, which are used to determine the size of the rectangle and the cap. Furthermore, when storing texture of the segment caps, the cap triangles must have at least one vertex that does not lie on a circle with other two vertices. Such a triangle has all three angles \(\theta = 0\) and is rendered into STM.
as a line. To overcome this problem, such a special case can be
detected and additional rectangle with different transformation can be
used to encode these cap triangles. For example the same spherical projection with rotated bases or an orthogonal projection can be used.

Figure 4: Capsule parametrization along a skeleton segment between two nodes. Texture coordinates are computed from the rectangle and cap size and the pairs of angles $\theta, \phi$.

### 3.4 Topology Constraints

In order to make the mapping as effective as possible, we use the rasterizer to interpolate the values in the texture. Moreover, during the mapping of the capsule surface around the skeleton onto a planar texture, seams are inevitable. Triangles that lie on texture seam can not be rendered in a normal way, because it’s $uv$ coordinates lie on the opposite sides of texture and rasterizer interpolates texels through whole texture. It is not possible to tell the rasterizer to lie on the opposite sides of texture and rasterizer interpolates texels through the seam. Thus, we propose three ways how to solve this problem. First solution concerns generating two textures for each segment (Figure 5(a)), where the first seam is cut at angle $\phi = 0$ and the second one at $\phi = \pi$.

The biggest possible angle difference between two vertices in one triangle is $\pi$ and therefore if the seam occurs in one texture, it will not occur in the other one. The second solution uses the cube maps to encode texture around the skeleton (Figure 5(b)). In this case the seam the problem is solved by hard-wired hardware algorithm. We choose a third solution, because it was the easiest to implement and the most efficient one. We split the texture at $\varphi = 0$ and all mesh triangles that lie on the seam has to be treated in a special way. If an input mesh shares vertices through indices, vertices of seam triangles on one side of the seam have to be duplicated (Figure 6(a)) and different texture coordinates are assigned to the duplicated vertices to cover both sides of the texture seam. Triangle will be also duplicated during rendering into STM to cover both sides of the texture (Figure 6(b)).

Figure 5: Each segment can be encoded into two textures (a) or hardware cube mapping can be used (b).

Figure 6: Surface with cylindrical topology has to be cut (a). Triangles that lie on the resulting seam (b - dark green) have to be duplicated. During the rendering the vertices on the right side of the seam (red dots) are send to GPU twice with different $uv$ coordinates.

### 3.5 Segment Composing and Mirroring

Initially we render each skeleton segment into its own rectangular texture. Knowing the area where each rectangle should lie (calculated in Section 3.2), the rectangle is rendered into this area. We set a global mirroring parameter, which determines the percentage, how much of the rectangle should be used for mirroring area. Thus, we clip each rectangle area by this mirroring margin before rendering and render the rectangle into the clipped area only. Furthermore, we perform topologically correct mirroring in the margin area. Part of the texture that topologically belongs here is rendered into each mirroring margins. This means that in horizontal direction the texture is enclosed by mirrored parts of itself and in the vertical direction by texture parts of the neighboring segments. Performing this mirroring we achieve that each texel in the final texture is surrounded by texels that really lie in the closest neighborhood of the point on the model surface.

In the case of skeleton texture mapping, the transition function is defined by deterministic allocation of rectangles in the final texture. The jumping from one chart to the neighboring one can be done by searching for the neighbors in the skeleton structure. The skeleton structure gives us an id of neighboring segment and each segment has its own fixed coordinates in the texture space.

### 3.6 Discussion on Robustness

In this section we prove robustness of our approach. Consider an input closed 2D manifold mesh topologically equivalent to a sphere with $N$ handles. If the mesh is disconnected, a skeleton is computed for each component separately and then all the skeletons are packed into the same STM. For each handle a skeleton is constructed, where the first and the last nodes are overlapping and creating a cycle around the handle. The remaining geometry without handles can be always decomposed into cylinders and described by capsule-shaped domain. The robustness can be proved by mathematical induction. First we cover all the handles by segments and then all the remaining parts can be decomposed into cylinders, until we cover the whole model.

However, the whole idea of the algorithm is based on a fact that the extracted skeleton is reliable [Cornea et al. 2007]. Reliability refers to the property of the curve-skeleton that every boundary point (point on objects surface) is visible from at least one curve-skeleton location. In other words, for any boundary point, there exists a straight line connecting it to a curve-skeleton point that does not cross any boundary. The second property of the skeleton is the centeredness. The skeleton does not have to be centred perfectly, but more centered skeleton yields less distorted mapping. However, the skeleton has to lie within the model. This is guaranteed by shifting of skeleton nodes into the center of mass of vertices,
they were merged of. Furthermore, during the skeleton simplification we check, if the simplification does not cause penetration of the surface by the skeleton segment. If such a situation happens, the current step of simplification is skipped.

The global robustness of our approach may be violated by two facts. The first fact is that currently the skeleton can be correctly extracted from 2-manifold meshes only. This can be improved by a modification of the skeleton extraction algorithm to support a skeleton extraction from point clouds. The second and more relevant fact is that the extracted curve-skeleton may lay outside of the model in some special cases. The skeleton contraction process may produce a skeleton lying outside of the volume for certain objects where some local regions have the center-of-mass outside the object (e.g., one with a C-shape cross-section). Even the shifting of skeleton nodes into the center of mass cannot help, because the center of mass of the C-shape local cross section lies outside of the model. However, the skeleton extraction centers the skeleton well for most organic-shaped models [Au et al. 2008], where the texture mapping is relevant.

4 Results

Here we present an applications of skeleton texture mapping concerning remapping of textures. Given an input model that contains texture and stored $uv$ coordinates for each vertex we extract a skeleton and STM. Later, we can remove $uv$ coordinates from the data structure representing the model and the model can be rendered using STM. When rendering, the skeleton will be computed in the preprocessing stage and new STM $uv$ coordinates can be computed during rendering.

First, we compute STM coordinates using our proposed technique. Second, the model is rendered into a fragment buffer object (FBO). As vertex positions the computed STM coordinates are used. The rasterization interpolates the texture values over the STM. During the STM mapping back to model, vertices near skeleton nodes can be mapped to more segment rectangles. We consider the one with the best mapping properties, where the texture is remapped with the lowest distortion. It is a segment where dot product between the skeleton segment vector and the triangle normal is minimal. A remapped texture onto the model surface using proposed STM can be seen in Figure 7. Currently implemented algorithm for skeleton extraction works only with 2-manifold meshes and the iterative contraction is very time expensive operation for high resolution models. Therefore, we show results only on simple primitives with variety of extreme geometric properties (Figure 8).

4.1 Computation Time

Table 1 shows how much time each stage of skeleton texture mapping requires. The time was measured on an AMD Phenom II X4 3.41GHz with 4GB RAM, using a single thread implementation for the calculations. As we can see, the most time consuming parts of the implementation is solving the linear system (contraction of the mesh).

<table>
<thead>
<tr>
<th>Model</th>
<th>#vert.</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ypsilon</td>
<td>480</td>
<td>&lt;0.1</td>
<td>&lt;0.1</td>
<td>0.2</td>
<td>2.1</td>
<td>2.6</td>
</tr>
<tr>
<td>Donut</td>
<td>1728</td>
<td>8.4</td>
<td>1.6</td>
<td>0.4</td>
<td>2.1</td>
<td>12.6</td>
</tr>
<tr>
<td>Octopus</td>
<td>1842</td>
<td>14.2</td>
<td>1.3</td>
<td>1.0</td>
<td>2.6</td>
<td>20.2</td>
</tr>
<tr>
<td>Triple-torus</td>
<td>2256</td>
<td>16.5</td>
<td>4.8</td>
<td>0.3</td>
<td>2.4</td>
<td>24.1</td>
</tr>
<tr>
<td>Crossed cyl</td>
<td>3660</td>
<td>73.5</td>
<td>7.0</td>
<td>0.5</td>
<td>3.6</td>
<td>84.7</td>
</tr>
</tbody>
</table>

Table 1: Columns $t_1$, $t_2$, $t_3$, $t_4$ and $t_5$ show the running time (in seconds), mesh contraction, skeleton construction, solving the packing problem and skeleton texture composing respectively.

4.2 Packing Efficiency

Our algorithm does not solve the packing problem ideally, but for a set of testing models texture coverage varies from 75% to 99%. Such an efficiency is sufficient for our applications of STM and for any further work with it. Measured packing efficiency in comparison to number of segments can be seen in Table 2.

<table>
<thead>
<tr>
<th>Model</th>
<th># segments</th>
<th>% of area covered in STM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donut</td>
<td>24</td>
<td>99.7%</td>
</tr>
<tr>
<td>Triple-torus</td>
<td>24</td>
<td>91%</td>
</tr>
<tr>
<td>Ypsilon</td>
<td>7</td>
<td>83.7%</td>
</tr>
<tr>
<td>Octopus</td>
<td>16</td>
<td>78.7%</td>
</tr>
<tr>
<td>Crossed cylinders</td>
<td>12</td>
<td>74.3%</td>
</tr>
</tbody>
</table>

Table 2: Table shows the packing efficiency of our algorithm for solving the packing of $N$ rectangles into one squared texture. It is easier to pack many smaller segments than few bigger ones.

5 Applications and Future Work

There are many useful applications of STM in computer graphics. The mapping can be used for extracting a STM from a textured mesh with classical $uv$ coordinates and applying the STM onto a surface without parametrization. Furthermore capsule-shaped domain around each segment can be used for procedural generating of textures aligned by skeleton axis. Using our technique, such a procedurally generated texture can be mapped around arbitrary
skeleton segment. The model geometry is not necessary while generating the texture.

Another group of applications concern skeleton displacement map (SDM), a case where texture encodes vertex displacement from skeleton. SDM can be used as a data structure for model representation. The skeleton and SDM are extracted from an input model and later they are used for reconstruction of the original model surface. Thus, a per segment level of detail can be assigned to the skeleton during reconstruction stage.

Notice that such an applications are neither possible with another global parameterization methods nor Reeb graph based parameterization [Patane et al. 2004]. To our knowledge, the polycube maps is the only technique with such possible applications, however with constrained topology with maximum of six skeleton branches per node.

As the future work we would like to explore another mappings for the segment caps and additionally implement encoding of the caps based on cube and sphere maps. In addition, during the capsule mapping we would like to explore benefits of a least-squares conformal mapping [Lévy et al. 2002] and compare the performance and quality of the results of these approaches.

Furthermore, we plan to implement a general skeleton extraction algorithm based on skeleton extraction algorithm from point clouds [Cao et al. 2010]. Such an implementation would be able to extract correct skeletons also from non-manifold meshes. In addition, we would like to parallelize whole skeleton extraction process using OpenCL. By improving this two major drawbacks of the algorithm, we would get fast and robust algorithm for skeleton extraction from a general input geometry as a non-manifold mesh, a polygon soup or a point cloud. Thus, we would be able to perform the skeleton texture mapping on variety of more complex and challenging models in order to make a comparison with another texture mapping methods and further evaluation of mapping distortion.

6 Acknowledgment

Our project was partly supported by project VEGA No. 1/1106/11 and from the grant No. UK/121/2012.

References


