Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Base Manifold Meshes from Skeletons

Master Thesis

2013

Bc. Michal Piovarči

Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Base Manifold Meshes from Skeletons

Master Thesis

Study programme:	Computer Science
Study field:	9.2.1 Computer Science, informatics
Department:	Department of Computer Science
Supervisor:	RNDr. Martin Madaras

Bratislava, 2013

Bc. Michal Piovarči

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:
Study programme:

Field of Study:
Type of Thesis:
Language of Thesis:

Title:

Aim:

Supervisor:
Department:

Assigned:

Approved:

Guarantor of Study Programme

.....
Student

.....
Supervisor

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:
Study programme:

Field of Study:
Type of Thesis:
Language of Thesis:

Title:

Aim:

Supervisor:
Department:

Assigned:

Approved:

Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgement

I would like to thank my supervisor RNDr. Martin Madaras for his help and advices.

Abstract

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Keywords: lorem, ipsum, consectetur

Abstrakt

Lorem Ipsum je fiktívny text, používaný pri návrhu tlačovín a typografie. Lorem Ipsum je štandardným výplňovým textom už od 16. storočia, keď neznámy tlačiar zobral sadzobnicu plnú tlačových znakov a pomiešal ich, aby tak vytvoril vzorkovú knihu. Prežil nielen päť storočí, ale aj skok do elektronickej sadzby, a pritom zostal v podstate nezmenený. Spopularizovaný bol v 60-tych rokoch 20. storočia, vydaním hárkov Letraset, ktoré obsahovali pasáže Lorem Ipsum, a neskôr aj publikačným softvérom ako Aldus PageMaker, ktorý obsahoval verzie Lorem Ipsum.

Kľúčové slová: lorem, ipsum, consectetur

Preamble

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc tristique, sem et feugiat ornare, lorem eros mattis odio, et tempus lectus ipsum nec ante. Phasellus interdum nunc ut sapien semper porttitor. Nam mi erat, faucibus in fermentum eu, varius eu velit. Integer egestas iaculis varius. In pulvinar, ligula eget adipiscing suscipit, nisl ipsum aliquet arcu, eget tristique felis leo vitae magna. Nulla et magna sed justo accumsan ultrices a in leo. Suspendisse tincidunt malesuada leo, eget rhoncus ipsum fringilla at. Integer et tortor vitae nisl fermentum vestibulum. Fusce eu dui neque, a egestas nunc. Vivamus condimentum mi non arcu lacinia et aliquam risus euismod. Nunc ut risus nec elit luctus aliquet et sit amet magna. Vestibulum vehicula enim eget erat fermentum a lacinia purus varius.

Duis tempus sem sit amet elit accumsan ultricies. Curabitur a nibh ante, vitae pharetra nulla. Suspendisse non risus elit, in aliquam felis. Maecenas suscipit placerat commodo. Vivamus et molestie odio. Quisque ut augue mi. Quisque aliquam luctus est, ac dignissim ante adipiscing eget. Quisque volutpat, sem vitae placerat condimentum, nunc lorem malesuada leo, sit amet pretium nisi felis nec lorem. Pellentesque nisi ipsum, vestibulum sed lacinia sed, condimentum a turpis.

In posuere convallis lectus vel hendrerit. Cras suscipit mi risus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec ante nunc, cursus ac vulputate at, bibendum eget nisi. Nunc eget nunc sed massa blandit posuere id vel quam. Duis bibendum orci vel ligula tempor condimentum. Nulla pharetra tortor at risus dignissim fringilla. Nullam ac massa et nibh auctor vestibulum quis vitae ligula. Suspendisse ultrices eros sit amet lectus dictum dapibus. Sed congue, turpis nec aliquam fermentum, diam nisi cursus nibh, id vulputate massa tellus sit amet turpis.

Contents

List of Figures

List of Tables

Introduction

Insert introduction.

Chapter 1

Related Work

Generating base meshes from skeletons aids in modelling and rigging of more complex meshes. The most notable algorithms are B-Mesh[2] by Ji et al. and Skeleton to Quad Dominant Mesh[3] (SQM) by J. A. Bærentzen et al. This algorithms are capable of generating quad-dominant manifold meshes with good edge flow. Generated meshes are convenient because thanks to quad dominance and good edge flow they are easily skinned. Since the mesh is generated from skeleton we also implicitly know how much each bone affects vertices of the mesh during animation. We will also discuss some results from Michal Mc Donells master thesis Skeleton-based and interactive 3D modelling[1]. In particular the generation of capsule endings in SQM algorithm as we also wanted our algorithm to be capable of generating capsules.

B-Mesh algorithm and SQM algorithm present two different approaches to generation of base meshes from skeletons. B-Mesh algorithm firstly generates mesh for the limbs of the skeletons. These limb meshes are later joined together. On the other SQM algorithm firstly creates polyhedrons for branch nodes of the input skeleton. These polyhedrons are later joined with tubes consisting solely from quadrilaterals.

In this chapter we will describe both algorithms with their advantages and disadvantages. This serves to show why we have based our implementation on SQM algorithm.

1.1 B-Mesh algorithm

Input B-Mesh algorithm takes as input a skeleton with a set of spheres or ellipsoids. Each node of the input skeleton has assigned a sphere or ellipsoid that represents its local geometry. Auxiliary spheres can be added to more precisely affect the resulting geometry of generated base mesh.

The algorithm consist of five steps:

Step 1: Sphere generation - new spheres are generated along the bones of the skeleton.

Step 2: Sweeping - generation of mesh for skeletons limbs.

Step 3: Stitching - joining of limb meshes at branch nodes.

Step 4: B-Mesh evolution - subdivision and smoothing of generated mesh.

Step 5: B-Mesh fairing - fairing to improve edges flow.

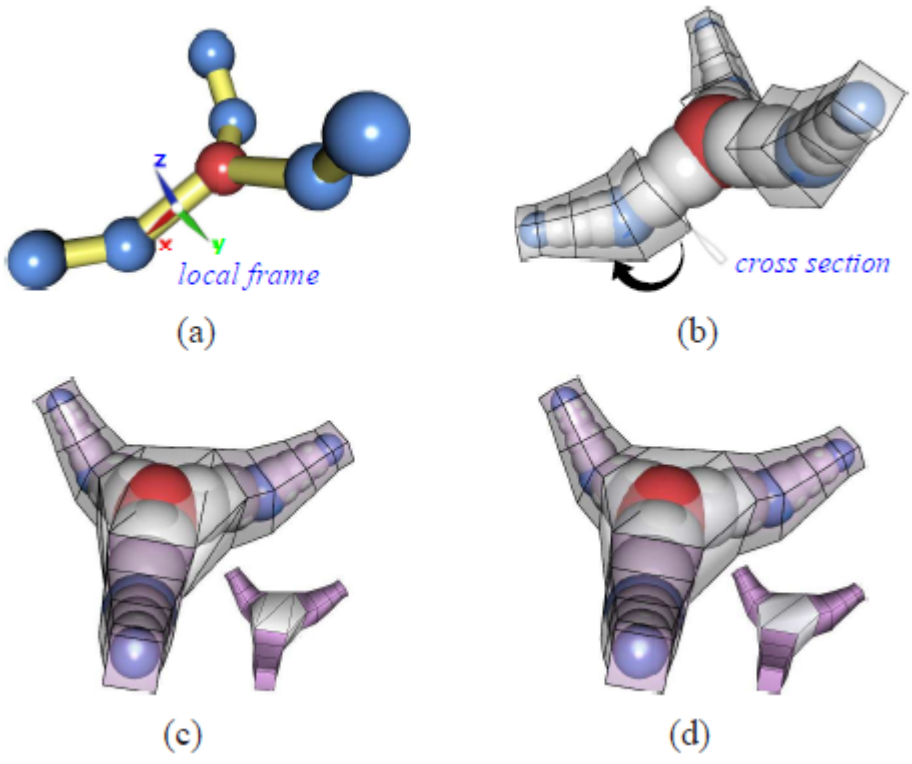

Figure 1.1: B-Mesh sweeping and stitching illustration. (a) local coordinates of a bone; (b) sweeping step; (c) stitching step; (d) after stitching simplification; Source [2]

Sphere generation Additional spheres are generated along the bones of the input skeleton. The distance between two spheres is defined by sampling step and the radius of each sphere is interpolated from radius's corresponding to the bones nodes. These generated spheres are used to refine the generated mesh and to calculate scalar field need in B-Mesh fairing step.

Sweeping For each bone its local coordinate system is generated as can be seen in Figure 1.1 (a). Limb mesh generation starts at branch nodes. For each new limb a rectangle aligned with limbs corresponding bones local axis is generated. Its points are then translated along the bones forming said limb and rotated around the connection nodes. After each translation the new points are connected with previous set of points in order to form a tube consisting of quadrilaterals. The resulting tubes can be seen in Figure 1.1 (b).

Stitching Limb meshes generated in sweeping step are now stitched together. This is done by generating a convex hull from all the points corresponding to each branch nodes. These points are the beginning of each limb mesh tube. The result can be seen in Figure 1.1 (c). Generated convex hull is composed from triangles. To simplify them into quadrilaterals a score between each pair of triangles is computed. The score measures how close to a plane is each pair of triangles. The results of stitching simplification are shown in Figure 1.1 (d).

B-Mesh evolution Catmull-Clark subdivision is used to smooth the stitched mesh. However after smoothing the mesh shrinks and deviates from the spheres. A scalar field is generated and used to evolve the mesh. Each vertex of the stitched mesh is translated according to the scalar field and its evolution speed. This means the further away is the vertex from its corresponding sphere the more it is attracted to it. In this phase the auxiliary spheres are used to manipulate the scalar field and thus the final shape of the mesh. This step can be repeated multiple times to further smooth the mesh.

B-Mesh fairing After the evolution step certain edges may not be aligned with their principal directions. New vertex positions are calculated by iterative minimization of a function.

Conclusion The biggest problem with B-Mesh algorithm for our use is its iterative nature. The number of iterations is an input parameter and we wanted to avoid input parameters, so that the base mesh generation is as automatic as possible. Without the evolution step B-Mesh

approximates input geometry worse than SQM. This is caused by the stitching phase which creates rectangular geometry at branch nodes, instead of the input spherical or elliptical geometry. Because of this evolution phase can not be avoided if we want to approximate the input as much as possible. B-Mesh algorithm is also running slower and produces more triangles than SQM according to [3]. The auxiliary spheres and ellipsoid nodes are interesting additions that were not present in SQM. But we wanted to modify the generated base mesh geometry directly on GPU during rendering so they are not advantageous to our intended use. B-mesh algorithm can generate capsules but it needs several evolution steps to precisely match capsules geometry. The algorithm looks like it can be naturally extended to support cyclic skeletons. Sweeping steps occurs only on limb paths and stitching occurs directly at branch nodes. Evolution and fairing moves vertices directly so it is independent from cycles in the input skeleton.

1.2 Skeleton to Quad Dominant Mesh algorithm

Input The input to SQM algorithm is a skeleton. Similarly to B-mesh for each of skeletons nodes a sphere is defined to represent local geometry. Contrary to B-mesh algorithm SQM does not support ellipsoid nodes. SQM algorithm has also an extra limitation that the root of the skeleton has to be a branch node.

The algorithm consist of four steps and one preprocessing step:

Preprocessing: Skeleton straightening - serves to simplify step number 3 of the algorithm.

Step 1: BNP generation - generation of branch node polyhedrons (BNPs).

Step 2: BNP refinement - subdivisions of BNPs.

Step 3: Creating the tubular structure - bridging of BNPs.

Step 4: Vertex placement - reverting straightened mesh to its original pose.

Skeleton straightening This is a preprocessing step of the algorithm that simplifies the generation of tubular structures. For each connection node its child is rotated, so that the edge between connection node and its child is parallel with the edge between connection node and its parent. This is usefully because during step 3 the algorithm needs to only generate straight tubes and does not need to take rotation into account.

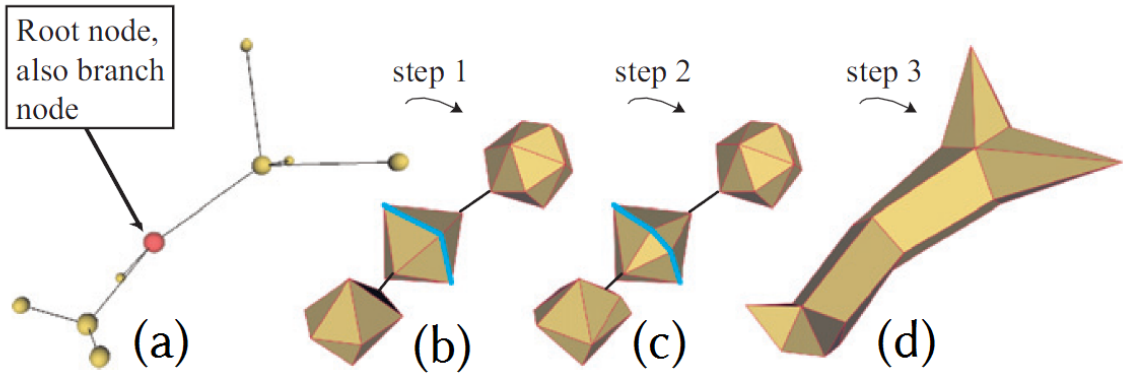

Figure 1.2: Steps of SQM algorithm. (a) the input skeleton; (b) generated BNPs; (c) refined BNPs; (d) BNPs bridges with quadrilateral tubes; Source [3]

BNP generation A branch node polyhedron(BNP) is a polyhedron assigned to a branch node. Vertices of a BNP correspond to a set of points that are generated by intersecting the sphere assigned to a branch node with each edge connected to said branch node. We will call this vertices intersection vertices. To form a BNP intersection vertices are triangulated. After that each triangle is split into six triangles by inserting one vertex in the middle of each triangle and in the middle of each of the edges of the triangle. These vertices are then projected back onto the sphere associated with a branch node. This projection is needed because if the intersection vertices are coplanar the generated polyhedron would have no volume. The result of this step can be seen in Figure 1.2 (b).

BNP refinement During step 3 of the algorithm BNPs connected via path are bridged with tubes consisting solely from quadrilaterals. This is done be connecting the one-rings of two corresponding intersection nodes with faces. To ensure that we can use only quadrilaterals the one-rings need to have the same valency. So each BNP is refined so that the valency of two intersection nodes lying on the same path are equal. This can be seen in Figure 1.2 (c).

Creating the tubular structure After previous step of the algorithm connected BNPs can be joined by a tube formed by quadrilaterals. The tube is divided into segments. Each of the segments corresponds to a connection node. Vertices corresponding to a certain connection node are projected onto its corresponding sphere. Leaf nodes are terminated with a triangle which central vertex corresponds to the leaf nodes vertex. The result is illustrated in Figure

1.2 (d).

Vertex placement The base mesh is now finished. All that remains is to reverse the rotations used to straighten the input skeleton. After final vertex placement the resulting mesh is smoothed using with three iterations of Laplacian smoothing and attraction scheme.

Conclusion Each of SQMs steps are one pass algorithms except the final smoothing and attraction scheme application. But since SQM generates branch node geometry by generating a polyhedron it better resembles the geometry of the input skeleton even without smoothing. Cyclic meshes can be problematic to implement, because in the refinement step of the algorithm a cyclic mesh may cause an infinite loop of refining. SQM produces smaller number of triangles because in the joining phase all triangles corresponding to branch nodes are removed and their former vertices are used to generate connecting tubes. On the other hand during the stitching phase B-Mesh introduces triangles into the mesh. SQM algorithm was also capable of generating cycles from two symmetric leaf node,s that were close to each other. However these cycles can not be created explicitly. Another limitation is that cycles have to lie on input skeletons axis of symmetry.

1.3 Skeleton-based and interactive 3D modelling

Michal Mc Donells master thesis studies the capabilities of SQM algorithm. It aims to extend SQM algorithm through pre-processing of SQMs input skeleton and post-processing of the output mesh. Altering SQM algorithm itself was out of scope of the thesis. Among the improvements to the algorithm were cone, cube, sphere and hemisphere leaf nodes, cube connection and branch nodes as well as concavities and branch node root requirement. The most important parts for us are spherical leaf nodes and branch node root requirement.

Spherical leaf node Original SQM algorithm was not able to create spheres at leaf nodes. This also limited its ability to generate capsules that we wanted to implement in our algorithm. In the thesis a solution was presented that a spherical node would be represented by several connection nodes. The first connection node would have the same radius as the desired sphere and each subsequent node would have its radius decreased until the desired number of subdivisions would be reached. We can see the process in Figure 1.3. An arc

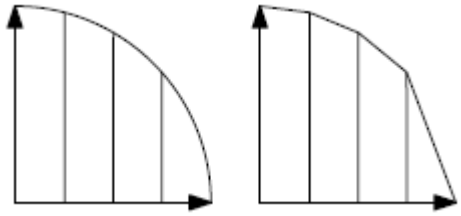

Figure 1.3: Left: arc subdivided at fixed intervals; Right: radii of newly inserted nodes sampled from an arc on the left; Source [1]

with equal radius as the original spherical node would be sampled at fixed intervals. With enough samples the resulting mesh would approximate a sphere. However in the original SQM algorithm after Laplacian smoothing the smoothed mesh did not approximate the input sphere.

Conclusion The approach to create spherical nodes presented in the thesis seems suitable for our needs. Since we would not implement Laplacian smoothing we would not have problems with deformation of the generated sphere. On the contrary this approach is beneficial to our needs, since we wanted to smooth the generated mesh in tessellation shaders and keeping spherical meshes generated without post-processing can potentially ease the work-flow.

Root branch node requirement One of the limitations of SQM algorithm is that the root node has to be a branch node. The proposed solution to this limitation was to generate pseudo paths from the root to ensure that it always will have at least three child nodes. For example, if the root has only two childes the new node would be generated in a direction perpendicular to a line connecting the intersection of roots associated sphere and edges from root to its childes.

Conclusion This pseudo path generation would disturb the edge flow as it would introduce new unnecessary edges. We believe that a better approach would be to re-root the tree and replace its former root with a node with at least three childes. Linear skeletons without branching and skeletons consisting only from a single node would become special cases that would be handled separately. This would allow us to avoid the requirement without altering the expected edge flow of the input skeleton.

Chapter 2

Proposed solution

We have picked SQM as the base for our algorithm. The main factors in this decision were that SQM is faster, produces smaller number of triangles, has better edge flow and even without smoothing the generated mesh better resembles the input skeleton. By avoiding the smoothing phase we do not need any input parameters to generate base mesh from an input skeleton. In this chapter we will explain each step of our proposed algorithm as well as extensions like elliptical nodes, cycles, etc.

2.1 Skeleton straightening

Skeleton straightening is a preprocessing step that simplifies bridging of branch node polyhedrons. Straightened skeleton is a skeleton, which nodes in every path between two branch nodes, two leaf nodes, or a branch node and a leaf node are co-linear. Also we have added an extra quality, that angles between branch nodes child nodes should be the same in straightened skeleton, as they are in the input skeleton. To achieve the first condition for each connection node we take the direction of a vector, formed by connection nodes parents position and connection nodes position. The direction vector can be seen in Figure 2.1 as the green arrow. Then we normalize the direction vector and multiply it by the distance between connection node and its child node. The distance is marked by the black curve in Figure 2.1. This vector represents the offset form connection nodes position, at which lies the straightened position of its child node. We then calculate rotation between connection nodes child original position and its new position, in respect to connection nodes position. Finally we rotate all descendants of the connection node. In order to conform to the second condition,

at each branch node we do not alter the position of its child nodes.

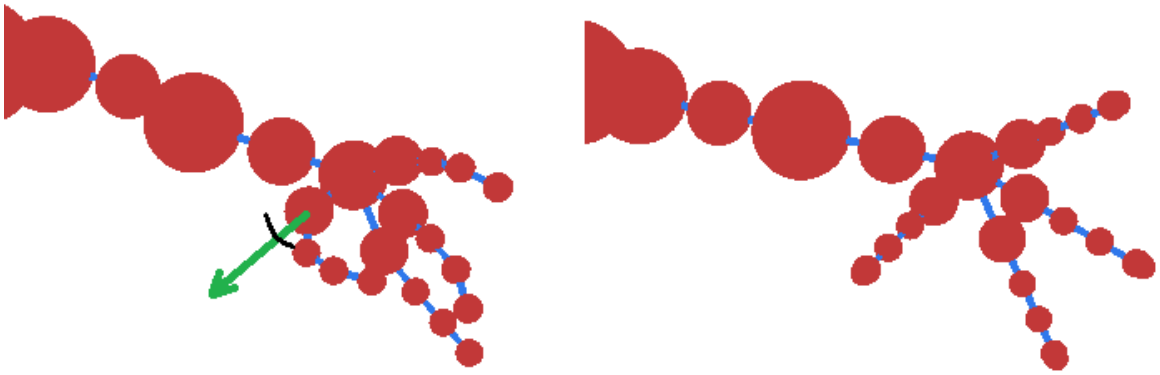

Figure 2.1: Skeleton straightening; Left: input skeleton, green arrow represents the direction from connection nodes parents position to connection nodes position, black curve marks the distance between connection node and its child; Right: straightened skeleton

2.1.1 Skinning matrices

In final vertex placement we need to undo the rotations applied to the input skeleton during straightening. We have decided that the best solution is to use skinning, since it can be implemented on GPU and we wanted to move all post-processing on the GPU. Straightened skeleton represents bind pose for skinning purposes and the input skeleton represents reference pose. Now we can calculate skinning matrices required to transform bind pose to reference pose. Traditionally that would require to find the rotation between two corresponding nodes in respect to they parent. Rotating all child nodes in bind skeleton using the same rotation and propagate the rotation calculation to child nodes. But since we know precisely how bind pose was constructed, we can exploit this knowledge and avoid the rotation of child nodes. In fact we do not even need the bind skeleton itself.

This can be seen in Figure 2.1.1. We want to calculate the rotation that would transform circle node to reference pose. We know that circle nodes parent square node is already in reference pose. We also know, that bind pose was constructed in such a way that all connection nodes childes are co-linear and preserve the distances between nodes. That means from squares reference pose we can calculate, where would be circle node, if we would apply

on it the same transformation matrices as were applied to square node. The distance between square and circle node remains constant in both poses. And the direction at which the circle node would be is the same as the direction from triangle node to square node, which is marked as green arrow in Figure 2.1.1. Now we only need to store the rotation between calculated circle node position, green circle in Figure 2.1.1 and its actual position red circle, with respect to its parent red square node.

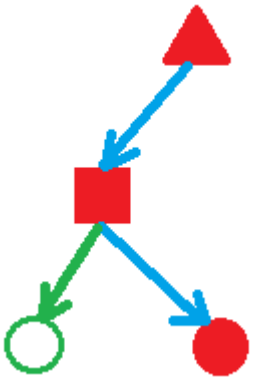

Figure 2.2: Rotation estimation from reference pose. Circle: node which rotation we want to estimate; Square: parent of circle node; Triangle: parent of square node; Blue arrows: edges in reference pose; Green circle: circle node position after applying squares skinning matrices; Green arrow: direction from square to green circle node;

2.2 BNP generation

For each branch node we calculate the direction from said branch node to each of its children and to its parent. Then we create rays, with origin in branch nodes position and direction corresponding to the direction calculated previously. These rays can be seen in Figure 2.2 (a) as blue arrows. We calculate the intersection of each of these rays with a sphere associated with the branch node. We store each intersection in a set of intersection points. Now we triangulate the intersection points. Different algorithms can be used to achieve the same effect, but we have picked Delaunay triangulation in spherical domain, which was used in the original paper. The algorithm works like standard Delaunay triangulation, but the predicate deciding whatever newly inserted triangle would lie in the circle of an already existing triangle is replaced. The new predicate compares angle between the newly inserted triangles

normal with normals of already existing triangles. Result of triangulation is shown in Figure 2.2 (b) as the blue triangle. The generated polyhedron is now subdivided by inserting a point in the center of each face and in the middle edge. The vertex inserted in the center of each face is then connected with all vertices corresponding to the same face. So each triangle is subdivided into six smaller triangles. The newly inserted points are then projected onto the sphere associated with the node. The subdivision and projection is necessary, because otherwise polyhedrons that would be generated with co-planar, or nearly co-planar intersection points would have no volume or very little volume respectively. To project the newly inserted vertices onto the sphere, we once again use a ray. The origin of the ray is the position of each newly inserted vertex. The direction of the ray is mean normal of the faces that are connected with the vertex. This means that for the vertices in the center of each face the normal of the subdivided face is used. For vertices inserted in the middle of each edge the mean normal of faces corresponding to that edge is used. Final polyhedron is shown in Figure 2.2 (c).

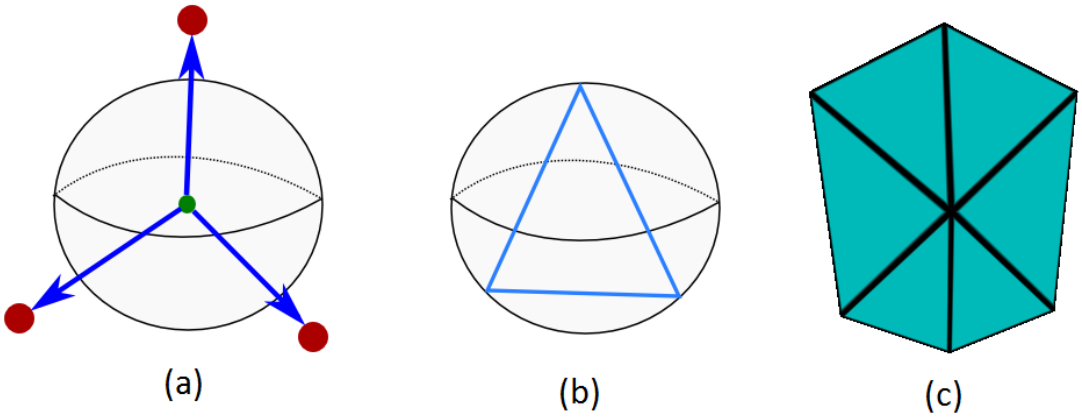

Figure 2.3: BNP generation process. (a) green is a branch node, blue arrows represent direction vectors of rays, red circles represent child nodes; (b) blue trinagle is the result of triangulation; (c) final subdivided BNP;

Triangulation of intersection vertices can sometimes create obtuse triangles. These are problematic, because when we insert the vertex in the middle of an obtuse triangle the one-rings of intersection vertices are not convex. That is if we would project them onto a plane defined by their principal axis and one of the vertices, the resulting polygon would not be convex. In Figure 2.2 we can see on the left how a polyhedron looks when it is generated

with obtuse triangles. Expected central vertex position is marked by red arrow, also expected edges are marked as yellow lines. This is not desirable, since it would cause problems during cyclic mesh generation. To remedy this situation we calculate the projection of the central vertex in a different manner. The origin of its ray is still vertex position. But we use a new direction vector. This vector is the normal of a triangle formed by the projected points that were inserted in the middle of each edge. The result can be seen in Figure 2.2 on the right.

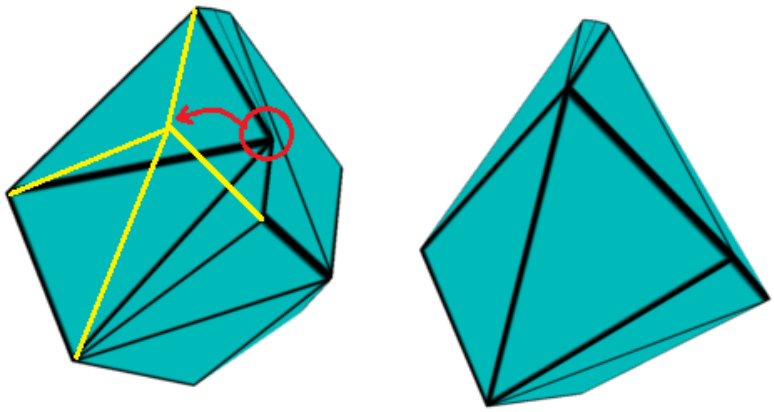

Figure 2.4: Obtuse triangle problem. Left: polyhedron with obtuse triangle. Red arrow marks vertex expected position and yellow lines mark expected edges; Right: polyhedron after applying our fix;

Chapter 3

Implementation

The main parts of the Skeleton to Quad Dominant Mesh algorithm, were discussed in previous chapter. Now we are going to describe each part in greater detail as well as our enhancements.

3.1 Skeleton Straightening

Skeleton straightening is a preprocessing which serves to simplify the third phase of Skeleton to Quad Dominant Mesh algorithm, that is joining of Branch Node Polyhedrons(BNP). Joining straight lines is easier than joining of arbitrary poly-lines. After this step each node is collinear with its parent node. In practice we rotate child nodes. The angle and axis of rotation are determined from the angle between two vectors **from** and **to**. **From** vector is the vector from parent to child node and **to** vector is the vector from parents parent to parent node. After the rotation of a BNP node all his child nodes are also rotated. This step serves to preserve the original skeleton structure.

During this step of preprocessing we also create and store skinning matrices. This matrices are used in last step of the algorithm named final vertex placement. This is an enhancement that allows us to finish the generated mesh in programmable shaders. Using skinning also allows us to prevent undesired mesh deformation that may occur with simple straightening.

Skinning matrices are computed from the original and straightened skeletons. Straightened skeleton represents the bind pose of the final generated mesh. Degree of freedom (DoF) of each node is calculated as the quaternion that is needed to rotate from bind pose

back to original skeleton. When we calculate DoFs in this way bind pose will always have zero values in its DoFs. So skinning matrices of bind pose will be identity matrices and we can omit them in calculation of subsequent skinning matrices.

3.2 BNP Generation

For each BNP we calculate the intersections of its associated sphere with paths that connect the BNP with its childes. We then triangulate the resulting points to create a mesh of the polyhedron. The triangulation is done using Delaunay triangulation in a spherical domain. We adopted the algorithm used in the original article.

Each BNP is then tessellated by inserting a vertex in the middle of each face and edge. We create a list of all faces that are not split yet and all edges that are already split. For each face we insert a vertex in its center and check its edges. We also insert a vertex in the middle of each edge that was not split yet. After this insertions the face is not a triangle. Next we connect each point of the face with the newly inserted central vertex so the face is triangulated. Finally we project all newly inserted vertices onto the sphere associated with current node. This serves to increase the volume of each node. Without it some BNPs will have little to no volume. For example BNPs created with only three intersection would be two faces stitched together with zero volume.

3.3 BNP Refinement

During the joining phase of the algorithm we want to join BNPs that are connected via a path. This connection is not possible to consist purely of quadrilaterals if the corresponding vertices have different valency (number of vertices in one-ring neighbourhood). Thus the purpose of BNP refinement is to ensure that each pair of connected vertices has equal valency. Link Intersection Edges (LIEs) is a set of edges which belong to the link of two intersection vertices. We increase the valency by splitting edges in a LIE. This split increases the valency of two intersection vertices simultaneously. We always split a representative edge and the we apply various smoothing schemes to equalize the lengths in a LIE. The smoothing schemes are: quaternion smoothing, neighbour avaraging smoothing, one-ring area weighted Laplacian smoothing and valency weighted Laplacian smoothing.

The algorithm loops through all BNPs. First we calculate the difference in valencies of each pair of vertices and mark it into a table. Intersection vertex with nodes parent is specially marked to not be subdivided more than is needed. Without this requirement a subdivision in child BNP could cause a need to subdivide parents BNP and thus create a possibly infinite loop.

Next we generate all LIEs of the BNP. For each LIE we store how many times it was refined, first and last vertex of LIE and quaternion describing the rotation from the first vertex of a LIE to its last vertex. This quaternion will be used to smooth inserted vertices if quaternion smoothing is enabled. We generate LIEs as follows. We loop through all intersection vertices. For each vertex we take an arbitrary vertex from its one-ring neighbourhood. We move backwards around the neighbourhood to find the first vertex of a LIE. For each vertex of one-ring neighbourhood we know its corresponding intersection vertices. If we fix two intersection vertices, then the first vertex of a LIE is the last vertex corresponding to these two intersection vertices while moving backwards. From the other side the last vertex of a LIE is the last vertex corresponding to the two selected intersection edges while moving forward.

In this fashion we can find the first vertex of the first LIE and after that we can construct all LIEs corresponding to a intersection vertex just by moving forward from the first vertex. When we find the first and last vertex of a LIE we try to construct quaternion representing rotation between them. We represent each of the points as a unit vector from corresponding node position and the vertex position. The axis of rotation is the cross product of these two vectors and the angle is their dot product. A problem may occur if the two vectors are linearly dependent. Then we can not decide the correct axis of rotation. In this case instead of using the first and last vertex of a LIE we use the first two vertices of a LIE to calculate the quaternion rotation. A LIE will always have at least two edges and three vertices due to the subdivision phase.

Generated LIEs are finally mapped to their corresponding intersection vertices, along with the number of splits required. Next we loop through all the intersection vertices. If the need to split is greater than zero, that is the valency of the intersection vertex is greater that the valency of its pair vertex. We then loop through all the LIEs of the intersection vertex and try

to find the best LIE to split. We pick the LIE witch has the biggest need to be split and was least refined. We prefer to pick a LIE that corresponds to a leaf node if there is no need to refine LIEs corresponding to other BNPs to avoid excessive subdivision. Intersection vertex corresponding to parent is picked first for splitting and after that it is forbidden to split it any further. This ensures that we wont force subdivision of parents BNP from its child.

3.3.1 Smoothing algorithms

Since we only subdivide the first edge of each LIE we need to smooth the resulting polyhedron. We have developed four smoothing methods: quaternion smoothing, neighbour avaraging smoothing, one-ring area weighted Laplacian smoothing and valency weighted Laplacian smoothing. In the end we selected quaternion smoothing as the most appropriate method, due to its speed and quality of output.

For quaternion smoothing we use the quaternions calculated during LIE generation phase. First and last point of each LIE are fixed so the angle and between them and their axis of rotation are also stable. We first count the number of vertices between first and last vertex of the LIE and then divide the angle by this number. Then we rotate the first point by a quaternion that is formed from the calculated angle and fixed axis of rotation. In the end the points are projected onto the sphere corresponding to current node. This method produces LIEs that lie on small circles of their corresponding sphere. The spacing between vertices is regular and thus is very suitable for our needs.

Averaging smoothing consists of averaging a vertex with its one-ring neighbourhood. We move from the last vertex of a LIE backwards towards the first vertex. We move each vertex of a LIE (except first and last) to the barycentre of its one-ring neighbourhood. This is a iterative approach that will potentially spread all vertices of a LIE evenly. We have found that one iteration of the algorithm is usually enough.

Laplacian smoothing consist of calculating standard Laplacian smoothing with different weight settings and then projecting smoothed nodes onto the sphere associated with current node. We used two weight settings. Valency weighted Laplacian used the valency of each vertex as the input weight. One-ring area weighted Laplacian used the one-ring area of each vertex as the input weight. Both of these methods produced similar results.

3.4 BNP Joining

All vertices that are connected via a path now have equal valency. For each intersection vertex we remove all faces of his one-ring neighbourhood. Then we select all the vertices of his one-ring neighbourhood and a ring of vertices that is parallel to them but is offset to the center of the next node. Then we can create quadrilaterals that will represent our mesh. We repeat this process for each node that is not a leaf or BNP node. In BNPs we instead of creating new set of vertices just connect with the vertices already existing in the BNP after we remove the unnecessary triangles. For leaf nodes we need to finish the mesh somehow. Various techniques can be used but the most useful for our purposes seem to be a ending in a point or in a capsule.

3.4.1 Point ending

Point ending is the simplest way of finishing the mesh. All the vertices of the last ring will simply form triangles with a vertex given by nodes position. Although relatively simple this approach offers the most control over the resulting mesh.

3.4.2 Capsule ending

Capsule ending means that leaf node will create a hemisphere on leaf nodes. There are various approaches to solve this problem. We could calculate the number of rings needed directly. But the algorithm is capable of creating tubular structures with varying radius. So the best approach seem to be to insert new nodes into the skeleton tree in a pre-process phase and let the algorithm create capsules on its own. This is very useful because we don't need to alter the algorithm itself and generated capsules will also work with tessellation.

After each capsule node we insert several new nodes into the skeleton. The number of nodes is depends on capsules radius. We have found that it is best that the number of new nodes is equal to the radius of the capsule, but to be at least five nodes. The inserted nodes are collinear with capsule node and its parent. They lie forward (direction from parent to capsule node position) from capsule nodes center. Inserted nodes are distributed according to a bezier curve that approximates the curvature of a unit sphere. for each node its radius is calculated according to formula:

$$newRadius = \sqrt{nodeRadius^2 * (1 - step^2)}$$

3.5 Final vertex placement

We now should apply the inverse of the rotation that we used to straighten the mesh to receive mesh in original pose. However a simple rotation could cause self intersections in the mesh and thus create non manifold meshes. In order to avoid this we used skinning to transform the vertices. The extraction of skinning matrices has been described in section 3.1. We use linear matrix combination. This method proves sufficient for our needs but other more sophisticated methods as quaternions or dual quaternions could be used.

Chapter 4

Example

4.1 Tables

In this section you can see example of tables.

1	2	3
4	5	6
7	8	9

Table 4.1: Numbers

And another one

A	B	C
D	E	F
G	H	I

Table 4.2: Letters

4.2 Figures

In this section you can see example of figures.

Figure 4.1: Johann Amos Comenius

4.3 Cross reference

In this chapter we used table 4.1 with numbers and table 4.2 with letters on page 20. Also, we used figure 4.1 with Johann Amos Comenius on page 21.

4.4 Citation

[3][1][2]

Appendix A

TEX

LATEX, TEX

Bibliography

[1] Michael Mc Donnell. *Skeleton-based and Interactive 3D Modeling*. PhD thesis, Technical University of Denmark, 2012.

[2] Zhongping Ji Ligang Liu and Yigang Wang. B-mesh: A fast modeling system for base meshes of 3d articulated shapes. *ComputGraphForum*, 29(7):2169–77, 2010.

[3] J. A. Bærentzen M. K. Misztal and K. Welnicka. Converting skeletal structures to quad dominant meshes. *Computers & Graphics*, 36(5):555–561, 2012.