Small Project: Automatic ragdoll creation from 3D models

Dion Gerritzen, 3220494

January 23, 2014

1 Introduction

This 'small project' was done in collaboration with Robert van Alphen and under supervision of Nicolas Pronost. Our goal in this project was to create a full-automatic (i.e. without user intervention) method that creates a physics ragdoll from an arbitrary 3D model. Primarily used in games and films, a ragdoll is a collection of constrained rigid bodies, often in the shape that resembles a humanoid, which can be used to simulate physics-based movements. Creating these ragdolls is generally a manual process, and includes creating an animation skeleton, binding the skeleton to the character and creating collision shapes for separate body segments. This can be a very time consuming task and depends on the animator's skill (e.g. non-optimal joint locations). Luckily, a lot of research has been done to automate (parts of) that process. We thought it would be useful to combine some of these parts to create a method to automatically create these ragdolls.

Originally, this method would be divided into three phases, namely skeleton creation (section 3.1), skeleton template matching and ragdoll creation (section 3.2). But because of time constraints we decided to drop the template matching phase, which was intended to compare the generated skeleton with a database to find a matching skeleton. For the remaining two phases we both selected different methods to implement, which enabled us to compare the results of our methods as part of the experiment. Because I am writing this report before Robert was able to finish his implementation, the comparison is included only in his report.

Previous work related to this subject is presented in section 2, where also the methods I have implemented are explained. In section 3, I introduce the changes I have made to the previously mentioned methods and my motivation behind these changes. I analyze my implementation and present the results of my experiments in section 4, and conclude my findings in section 5.

2 Related work

A lot of research has been done on the principle of skeleton creation, but not a lot on automatically creating a fully functional ragdoll. Therefore we had to combine multiple methods to create our own method.

2.1 Skeleton creation

For the skeleton creation there are several interesting papers, such as the method by Pan et al. (2009) [1] that uses a '3D silhouette' to extract a curve skeleton from a 3D mesh. This is the paper I have chosen to implement and will describe their method in more detail.

2.1.1 Primary 3D silhouette detection

Pan et al. (2009) [1] describe a skeleton creation method using 3D silhouettes (i.e. 2D silhouettes (or outlines) with added depth values) to extract a curve skeleton, which can then be used to create an animation skeleton. By working with two dimensional data, this method is a lot faster than conventional skeletonization methods that use three dimensional data.

The first step in creating this silhouette is to find an orientation for the input model in which it can be projected. This orientation is chosen in such a way that the projection has the largest possible surface area, i.e. the orientation of the model which results in the least amount of self-occlusion. Then the 3D vertices are projected onto the two-dimensional plane orthogonal to that orientation so we get a new set of 2D vertices.

They then use a two-part method to construct a silhouette from the 2D projection, consisting of a 'global search' and a 'local search'.

2.1.1.1 Global search

The global search starts at the vertex $c_1'(x, y)$ from the projection that has the highest y-value. Then they find all vertices from the projection within a distance $r = \max \ \|q_i(x, y, z) - q(x, y, z)\|$, where $q_i(x, y, z)$ is the neighbor vertex connected with the current vertex $q(x, y, z)$ (in this case $c_1'(x, y)$). From those candidates they take the vertex $c_2'(x, y)$ that gives the smallest angle in a polar coordinate system with $c_1'(x, y)$ as origin. Vertex $c_2'(x, y)$ becomes the second vertex of the silhouette. For each of the next vertices they search again for all vertices within distance r of the previous vertex and from those candidates they select the vertex $c_3'(x, y)$ with the maximum angle $\angle c_3' c_2' c_1'$. This is illustrated in figure 1. The process continues until the first vertex is reached again. A 2D silhouette is now created and can be used to find the 3D vertices for the 3D silhouette. This is done by matching the x and y coordinates of the 2D vertices from the silhouette with the x and y-coordinates of the vertices from the 3D model.

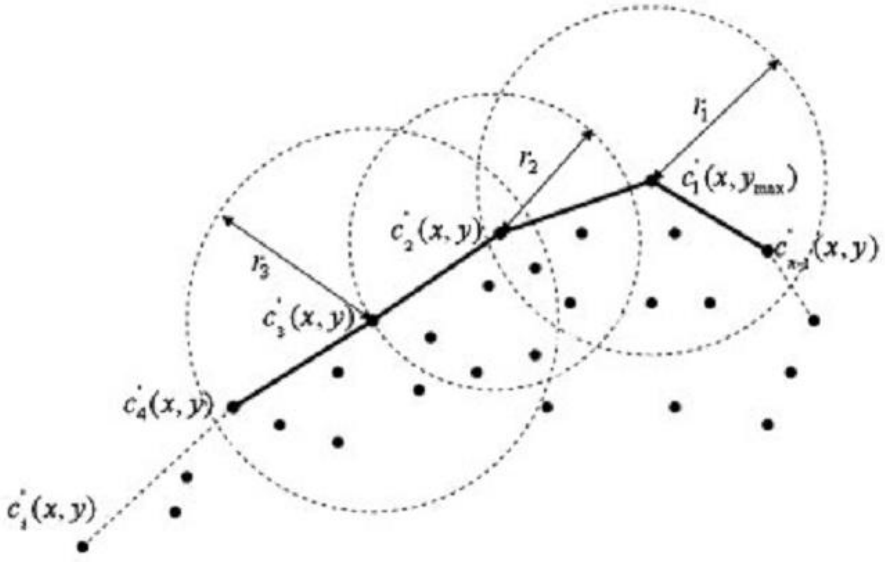

Figure 1. Illustration of global search, taken from [1]

2.1.1.2 Local Search

The second step is the local search, which connects the 3D vertices together according to the topology of the original mesh. They start at an arbitrary vertex $c_i''(x, y, z)$ and find all its neighbor vertices in the mesh. From the neighbors they find a vertex $c_{i2}(x, y, z)$ which has the maximum angle $\angle c_{i2} c_i'' c_{i-1j}$, where $c_{i-1j}(x, y, z)$ is the previous vertex of $c_i''(x, y, z)$ when $c_{i-1}''(x, y, z)$ and $c_i''(x, y, z)$ are connected. This process repeats until the currently detected vertex $c_{ij}(x, y, z)$ meets the end vertex $c_{i+1}''(x, y, z)$ or when the distance between those vertices is less than r. This is done for all consecutive vertices retrieved from the global search step, until the first and last vertices are connected and form a 3D silhouette.

2.1.2 Curve skeleton extraction

Pan et al. (2009) [1] state that the next step is to create a triangulation from the 3D silhouette. In their method, they extended the constrained Delaunay triangulation-based method as described by Igarashi et al. (1999) [2] to detect 3D medial axes. They perform 3D constrained Delaunay triangulation for the 3D silhouette, regardless of its z coordinate.

The triangles that are created can be classified into three categories: triangles without external edges (junction triangles), triangles with one external edge (sleeve triangles) and triangles with two external edges (terminal triangles). The medial axis (or curve skeleton) can then be created by connecting the midpoints of the interior edges, linking the terminal triangles with the junction triangles (or junction to junction or terminal to terminal) through the sleeve triangles. The midpoint of an edge is calculated by taking the two corresponding 3D vertices of the triangle edge and calculating the center between those vertices.

The creation of the medial axis is done in a recursive way. Because the topology of the input model is unknown, we start the algorithm at a random terminal triangle and calculate the midpoint of its only interior edge. This is the first vertex of the medial axis. The next recursion step depends on what type the next triangle is (i.e. the triangle that shares the interior edge with the current triangle):

- If the next triangle is a sleeve triangle, the next medial axis vertex is the midpoint of the opposing interior edge.
- If the next triangle is a junction triangle, the medial axis ends. For the two remaining interior edges the algorithm goes into recursion, starting at the midpoint of these edges, making a new medial axis for each branch.
- If the next triangle is a terminal edge, the medial axis ends. The current recursion branch stops as well.

The algorithm runs until all recursion branches have stopped. Then the whole triangulation is covered and the curve skeleton is formed by the set of medial axes. This process is illustrated in figure 2.

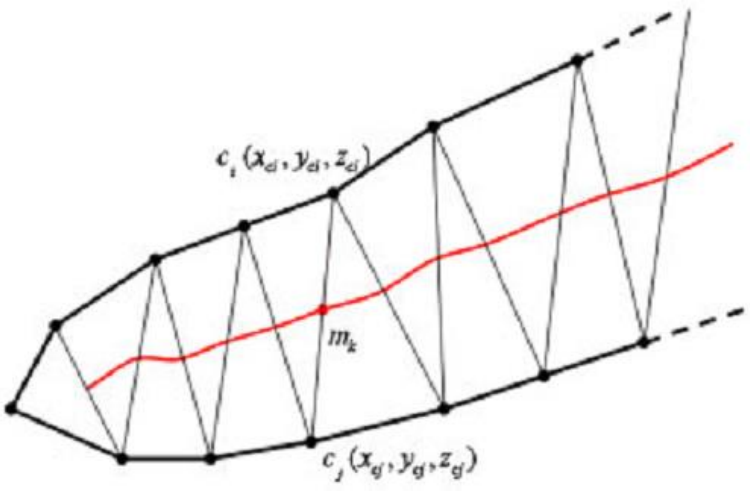

Figure 2. Illustration of curve skeleton extraction, taken from [1]

2.1.3 Curve skeleton refinement

To acquire an accurate curve skeleton, Pan et al. (2009) [1] state that a second projection is necessary to complement the missing data, i.e. the correct z coordinates of the curve skeleton.

Because a single second projection will likely contain insufficient useful data due to occlusion, each curve skeleton branch gets its own 'second projection'.

To get these projections, a decomposition of the input model is made. For every 3D vertex in the model they check for the distance between that vertex and the points on the curve skeleton, according to the following formula:

$$s(v_i, k) = \min_{j=(1,n)}^{k} \left| distance\left(v_i, m_{k,j}(x,y,z)\right) - l_{k,j} \right|$$

where v_i is the current vertex, point $m_{k,j}(x,y,z)$ is the jth point of the kth branch of the curve skeleton, and $l_{k,j}$ is the distance between $m_{k,j}(x,y,z)$ and the primary silhouette. Then vertex v_i is assigned to the kth branch of the curve skeleton, creating the decomposition.

Then a 3D silhouette is created for each of the curve skeleton branches. This is done by projecting the vertices assigned to a curve skeleton branch to a plane perpendicular to both the first projection plane and the vector formed by the end points of the associated curve skeleton branch. Then the silhouette is made using the global search method mentioned in the curve skeleton creation part.

Next they refine the medial axes. This is done by searching for the four nearest neighbors for all vertices in the curve skeleton branch, from the set of vertices from the corresponding second silhouette. The average z-coordinate value of the four nearest neighbors then replaces the z-coordinate of the vertex in the curve skeleton branch, bringing the curve skeleton closer to the center of the model. This is shown in figure 3.

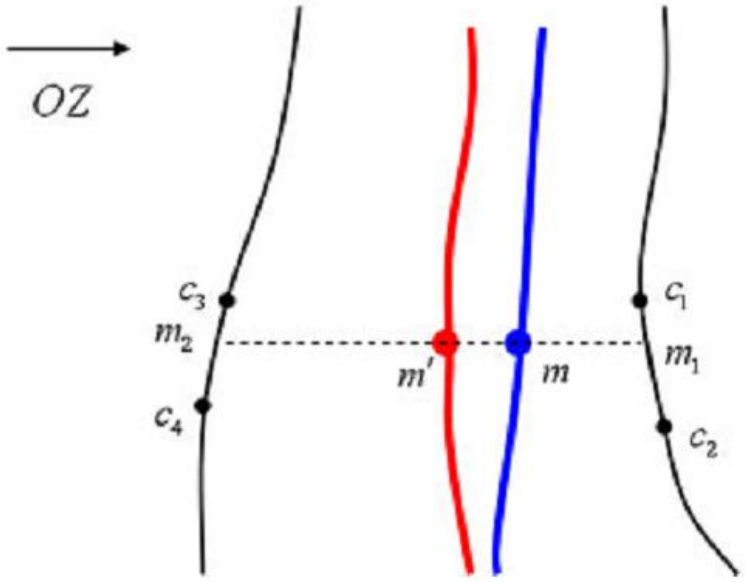

Figure 3. Illustration of curve skeleton refinement, taken from [1]

2.1.4 Animation skeleton generation

Pan et al. (2009) [1] briefly mention that the curve skeleton can be converted to an animation skeleton. This is done by locating skeletal joints on the curve skeleton and linking them with straight line segments.

The paper states that where the start and end points of the medial axes meet the junction triangles, the key skeletal joints are formed. After these key points are connected, the hierarchical structure is constructed and a coarse discrete animation skeleton is generated. To identify the middle joint

points, i.e. the points between the start and end joint points of each curve skeleton branch, the authors developed two methods:

- The first method is a down sampling method described by Au et al. (2008) [3] which calculates bending angles in the curve skeleton to determine if a skeleton joint must be placed, e.g. when the angle between two nodes in a curve skeleton branch exceeds 18°.
- The second method uses a database of skeleton templates to find a match for the newly created curve skeleton. This method is not described in the paper and is left for future work.

After all skeletal joints are located, the root bone is selected using a method described by Wade et al. (2002) [4]. The animation skeleton is now finished.

2.2 Other skeleton creation methods

Further papers we found about skeleton creation include a paper by Katz et al. (2003) [5] that describes a method that uses fuzzy clustering and minimal cuts to determine joint locations for a skeleton. They compute a hierarchical decomposition by segmenting the mesh at regions of deep concavities. The skeleton is then formed by placing joints at the center of the boundary between the segments. This method was implemented by Robert for his part of this project.

Other candidate papers we considered were, among others, Liu et al. (2003) [6] who describe a method to automatically generate an animation skeleton of a model using repulsive force fields. A modified thinning algorithm then generates an initial skeleton, which is further refined to become the final result. Au et al. (2008) [3] present a simple skeleton extraction method based on mesh contraction. The method contracts the mesh geometry into a zero-volume skeletal shape by applying implicit Laplacian smoothing with global positional constraints. The contracted mesh is then converted into a one-dimensional curve skeleton. The skeleton is refined by exploiting the induced skeleton-mesh mapping. Baran et al. (2007) [7] present a method for animating characters automatically by adapting a generic skeleton to a character and attaching it to the surface.

2.3 Ragdoll creation

Papers on collision shape creation are often intended for physics based modeling, but those methods are also relevant for ragdoll creation, as a ragdoll is essentially a collection of constrained collision shapes. The method I chose to implement is described in the paper by Liu et al. (2007) [8], which is a method to construct a bounding volume hierarchy (BVH) consisting of ellipsoid shapes that approximates the given model. The volume occupied by the given model is divided into sub-volumes where each is approximated by a volume bounding ellipsoid. Then, each sub-volume is subdivided into ellipsoids for the next level in the hierarchy. These bounding ellipsoid sets at a chosen hierarchy level can then be used as collision shapes in ragdolls.

An entirely different approach is presented by Bae et al. (2012) [9], describing a user-guided volumetric algorithm to approximated 3D meshes. They first construct a medial axis transform of the input mesh, and segment the medial axis into parts using a region growing method. Then, the object surface is decomposed into regions based on the segmented medial axis. Each region is approximated with a swept sphere volume. These regions can be interactively refined by splitting and/or merging using a sketch-based input.

The original plan for this project was to implement these methods for the ragdoll creation phase, but due to the aforementioned time constraints we chose to approach this in a more simplistic way, as described in the next section.

3 Changes in the algorithm

3.1 Skeleton creation

For the implementation of the skeleton creation phase I mainly followed the method by Pan et al. (2009) [1] described in the previous section, but I also made several changes to make it work (better) or to adjust it to my likings. In this section I will describe these changes and explain why I made these changes.

3.1.1 Primary 3D silhouette detection

The paper describes a two-part method to construct a silhouette from the 2D projection, but I have chosen to use an alternative method for this. This is because I couldn't get their method to work with the way they described the algorithm. They state that in order to find the next vertex $c_3'(x,y)$ for the silhouette, you need to find the maximum angle $\angle c_3'c_2'c_1'$, where $c_2'(x,y)$ and $c_1'(x,y)$ are the previous vertices. In practice, the situation depicted in figure 4 occurs in almost every test model. All my attempts to change their method by using other criteria for choosing $c_3'(x,y)$ have failed, so I decided to use an entirely different method. The outcome of my method should be more or less identical to theirs so it will not result in a different skeleton in the end.

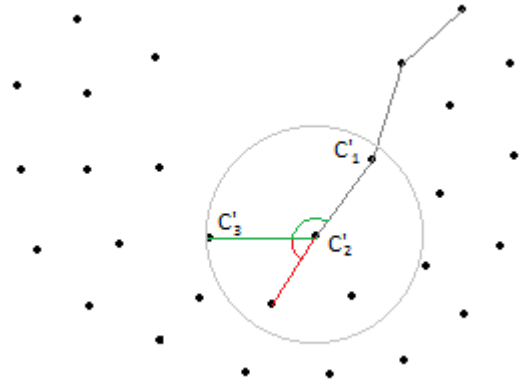

Figure 4. Illustration of the silhouette detection going wrong
The green line depicts the desired edge, the red line the found edge

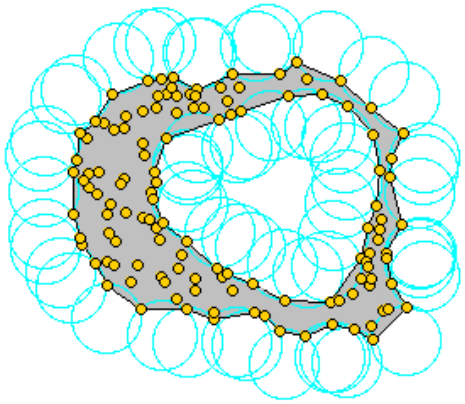

Figure 5. Illustration of an alpha shape

Instead of using the two-level-search algorithm of Pan et al. (2009) [1], I used an alpha shape [10] to construct the approximate shape of the 2D projection. An alpha shape is created by trying to fit a circle, with α being the squared radius of that circle, between pairs of points from a set. In this case the set consists of the 2D projection vertices. If the circle fits around just two points, i.e. if the two points are able to lie on the boundary of the circle and no further points inside, an edge is formed between these two points. This forms an outline of the point set, as seen in figure 5. Unfortunately this introduces a parameter to the algorithm that needs to be set, namely the squared radius α of the sphere that determines the size of the cavities in the alpha shape. Luckily, CGAL contains a function that finds the 'optimal' α value, i.e. the smallest α for which the following two properties hold: the alpha shape consists of one solid component and all data points are either on the boundary or in the interior of the alpha shape. Using this function I was able to avoid having to manually input a parameter for each model, thus maintaining the goal of making the algorithm fully automatic.

3.1.2 Curve skeleton extraction

According to Pan et al. (2009) [1], the next step is to create a triangulation from the 3D silhouette while ignoring the 'third coordinate'. Instead, I was able to use the 2D vertices from the 2D silhouette to create a triangulation, because I still had them stored.

In addition to the curve skeleton extraction method from the paper, I implemented a simple filter that keeps medial axes that are too small out of the curve skeleton. This filter checks for the size of each medial axis and marks them as 'deleted' when their size (amount of edges) is below a certain threshold. I am only marking them as deleted as opposed to physically removing them so the overall structure of the curve skeleton stays intact. I found that filtering medial axes with a size smaller than 3 edges gave the best results for all test models. I did this because in practice it appeared that these small medial axes can cause 'artifacts' in the curve skeleton, such as little lumps in curves or on the endpoints of the model (see figure 6). Filtering out these small medial axes remove the artifacts and a natural looking curve skeleton remains.

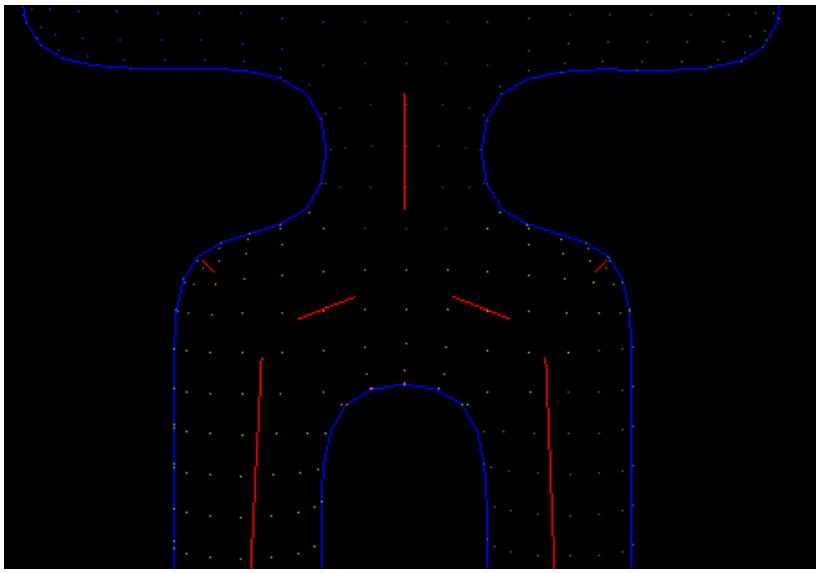

Figure 6. Undesired artifacts in the curve skeleton can be seen in the hips of this model

3.1.3 Animation skeleton generation

The paper mentions that the start and end points of the medial axes on the midpoints of the junction triangle edges form the skeletal joint points of the animation skeleton, but doesn't explain how to construct a hierarchical skeleton from a collection of medial axes. This is, however, important for a well-functioning animation skeleton, so I had to come up with my own method.

The top of the hierarchy is the root, so a logical first step would be to find a suitable root. Assuming the skeleton joints are located at the end points of the medial axes and the root lies generally in the center of a skeleton, I consider the most centered end point of all medial axes as the root bone.

When the root bone is found, I add the rest of the bones recursively to the skeleton.

First I find the triangle where the end point of the medial axis the previous bone was created from belongs to.

- If the triangle is a junction triangle, create a child bone on the midpoint of the remaining edges that have a medial axis attached to them. From each edge, repeat the recursion. For edges that have no medial axis (junction triangle edges without a medial axis attached can

occur because of the filter that 'deletes' small medial axes), find the next junction or terminal triangle as if there is still a medial axis.
- If the triangle is a terminal triangle, end the current recursion branch.

This algorithm runs until all recursion branches have ended. We now have an animation skeleton that is suitable to be used for a ragdoll.

3.2 Ragdoll creation

The ragdoll creation phase is loosely based on the paper by Liu et al. (2007) [8], in the sense that I'm using ellipsoid shapes for the rigid bodies. Also the method to create these ellipsoid shapes differs from the paper and I will explain below how and why I made these changes.

3.2.1 Rigid bodies

The paper presents a method for generating a bounding volume hierarchy (BVH) with ellipsoids as primitives. Their paper inspired me to use ellipsoidal collision shapes for the rigid bodies in the ragdoll, but without using the BVH because of time constraints. Also, instead of using voxelization to approximate the size of the ellipsoids, I'm using a much simpler method using axis-aligned bounding boxes (AABB).

Each body segment of the model's decomposition has its own skeleton bone, so each segment will have its own rigid body. To determine the dimensions of the ellipsoid we will be using for a body segment, we will use the AABB of the body segment vertices. Because the rigid bodies of the body segments will be ellipsoid-shaped, an AABB will be a relatively tight fit. But because a body segment can be oriented in every possible way, the AABB may still be (a lot) larger than necessary. To obtain a right sized bounding box, I rotate the body segment vertices in such a way that the first principle component is pointing upwards (i.e. into the positive y-direction). Now the AABB will be a tight fit with the body segment. In retrospect, an oriented bounding box (OBB) would have been helpful here. But because the standard Ogre library doesn't contain OBB's I decided to come up with a solution myself.

After that we can create an ellipsoidal collision shape, using the dimensions of the axis-aligned bounding box. The collision shape we now have is still oriented 'upwards', so we need to rotate the shape back to its original orientation using the inverse of the rotation used to rotate the body segment vertices.

Before I used the rigid bodies in a ragdoll simulation, I wanted to set the damping for each rigid body (a value between 0.0 and 1.0, inclusive), to prevent the body parts from moving around too fast. Because we want this algorithm to be a fully automatic process I dynamically adapt the rigid body's damping to the size of the bounding box used to create its collision shape. I came up with two different ways of calculating the damping, but neither of them are an optimal solution:

- The 'bounding box volume method' takes the log of the tenth root of the bounding box's volume as the damping
- The 'bounding box length method' takes the log of the square root of the bounding box's length (y-axis) as the damping

Both methods provide reasonable results, when converting the values to the 0.0 to 1.0-range, but there still arise some problems with relatively large models. Primarily damping values close to zero for very small rigid bodies within a large ragdoll. These small rigid bodies still move around way faster than desired. There are surely better alternatives for calculating the damping, but the above methods are sufficient for now.

3.2.2 Cone twist constraints

To connect the rigid bodies to each other I used cone twist constraints, since that constraint is generally useful in ragdolls. This type of constraint consists of one twist axis and two hinge axes. The constraints are placed between the rigid bodies, at the location of the bone rotation points. I made sure that the twist axis is pointing from one rigid body towards the contact point of the other rigid body, so that the bodies can move in a way comparable to how an upper arm moves inside a shoulder socket.

The limits of the constraints are currently not set, because of a bug where rigid bodies spontaneously start moving when any limit is applied to the constraints. Unfortunately, I was unable to fix this bug, which means that the rigid bodies are currently not constrained in the degree of their movements. I will take this into account for the analysis.

4 Analysis

To test how well this method performs in terms of execution time I measured the amounts of time each section of the algorithm needed to run (see Table 1). The experiments were done on a 2.5 GHz Intel Core i5-3210M with 8 GB of RAM.

	'Darwinia'	Armadillo (low poly)	Male t-pose	Octopus	Sinbad	Armadillo (high poly)
Vertices	930	1731	2048	4368	5392	17299
Faces	1856	3458	3264	7996	7612	34594
Body segments	8	22	12	19	14	50
3D silhouette detection	7	16	17	54	48	258
Triangulation creation	2	3	3	11	2	15
Curve skeleton extraction	2	8	7	46	4	107
Decomposition creation	2	29	19	186	38	3282
Curve skeleton refinement	6	11	15	33	36	119
Animation skeleton creation	< 1	3	1	3	1	56
Ragdoll creation	2	3	1	3	2	7
Total time	**21**	**73**	**63**	**336**	**131**	**3844**

Table 1. Execution times of different input models (measured in milliseconds)

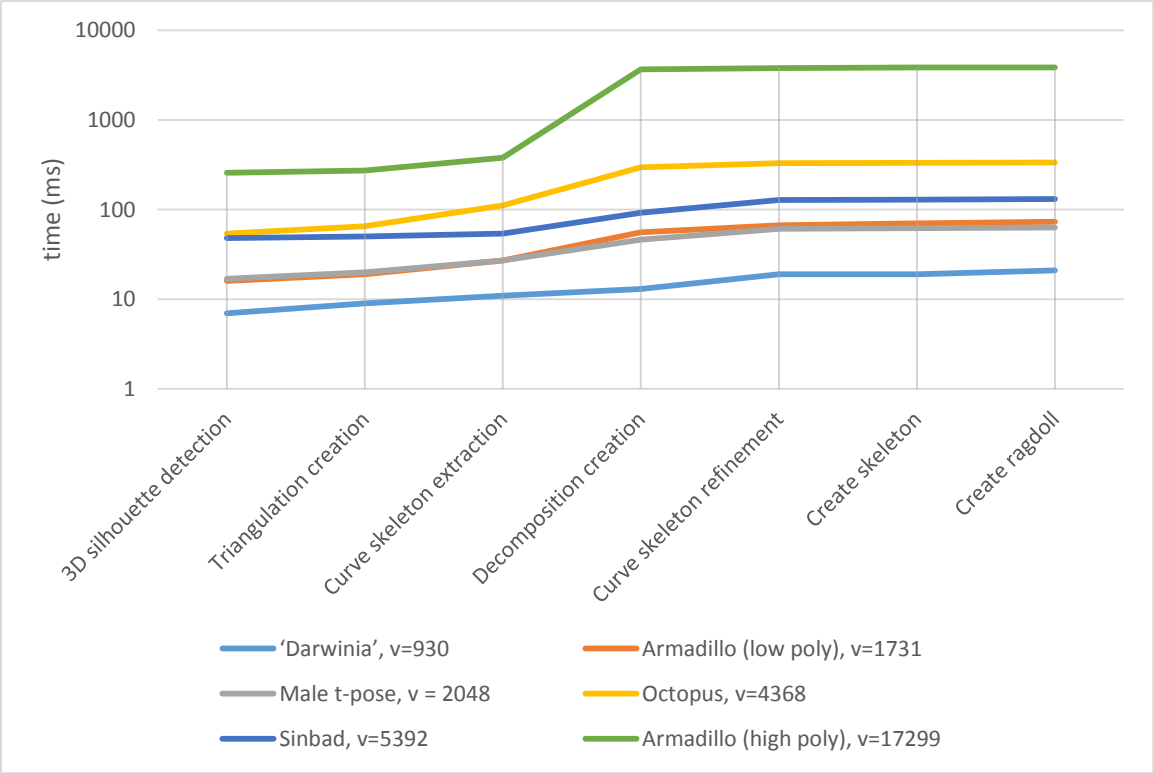

Figure 7. Execution times from table 1 displayed in a graph (logarithmic scale). Note that the high poly armadillo model (green) takes a disproportionate amount of time in the decomposition phase compared to the low poly version (orange)

Creating the decomposition generally takes the most amount of time, especially when the curve skeleton has a lot of medial axes. The time it takes to create the ragdoll is very little, since it only depends on the amount of body segments.

When looking at the data of the four fastest generated models (i.e. Darwinia, armadillo (low poly), male t-pose and Sinbad), one would say the algorithm runs in linear time. But then the other two models (armadillo (high poly) and octopus) throw a spanner in the works. They both take a disproportionate amount of time for the curve skeleton extraction and the creation of the decomposition used for the curve skeleton refinement, compared to the rest of the models. Unfortunately I could not come up with a logical explanation for this behavior based on the available data.

In terms of how the ragdolls look and behave in the physics simulation I am pretty satisfied. Although low-polygon models, such as the 'Darwinia' model (figure 8 and 9) and the male t-pose (figure 10 and 11), don't have very detailed ragdolls (i.e. single rigid bodies for arms and/or legs), the placement of the rigid bodies are in most cases logical. The medium and high-polygon models however are often a bit of a mess. Especially the high-res armadillo model (figure 12) doesn't look very good. The rigid bodies in the ragdoll are intersecting with each other in a lot of places so they get pushed away, deforming the model. In fact, four out of the six test models show deformations due to rigid bodies getting pushed away. Only the two models with the lowest amount of polygons (i.e. Darwinia and male t-pose) maintain their intended form.

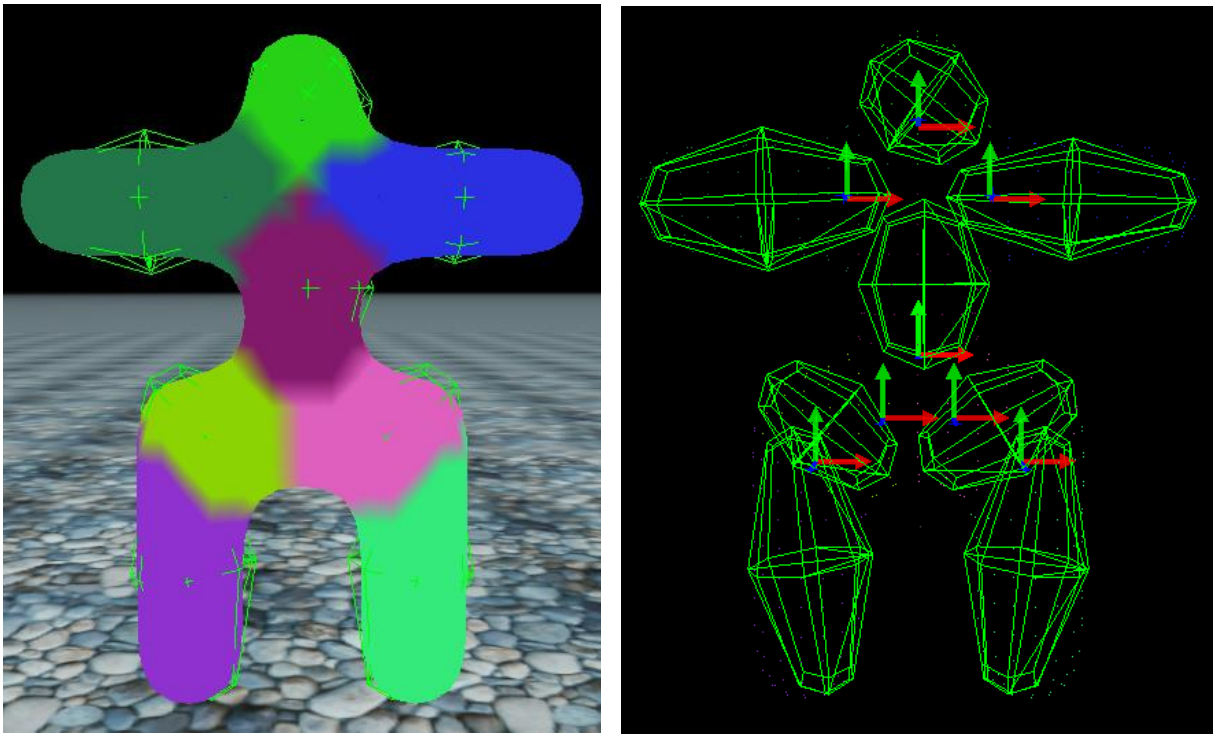

Figure 8 and 9. The 'Darwinia' model
left: mesh decomposition, right: rigid bodies and skeletal joints

The ragdoll physics simulation looks a bit exaggerated, caused by the lack of constraint limits. Therefore it is hard to judge whether the ragdoll behaves realistically. But based on the ragdoll shapes and structures of the models, I'd say the non-deformed models behave as expected. The models that are deformed however, probably won't look good in motion either.

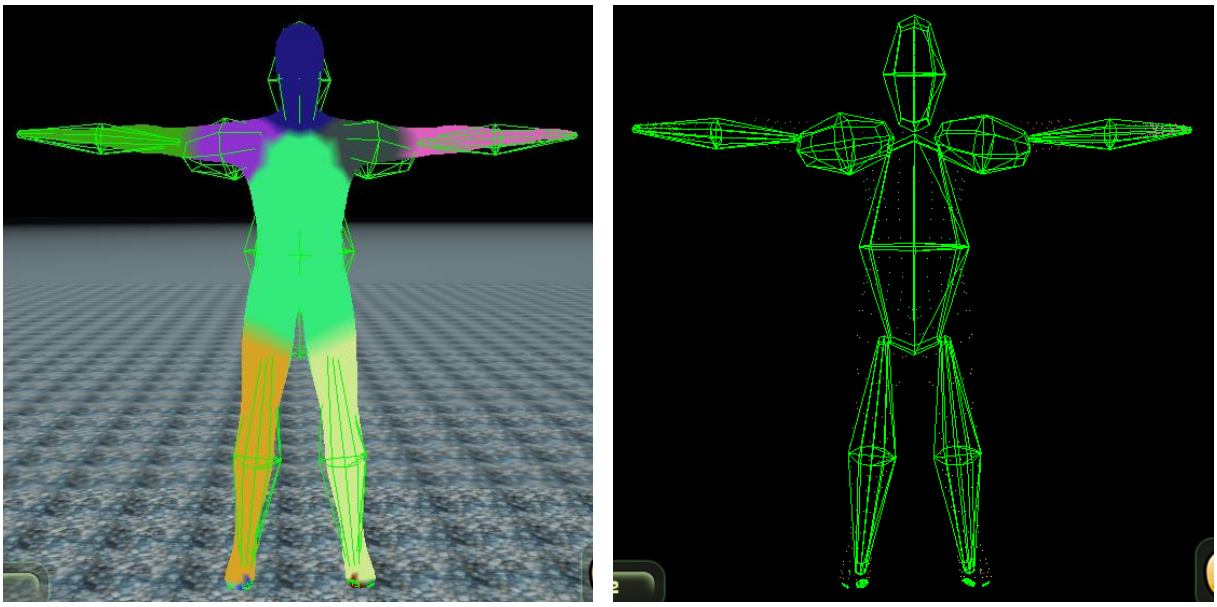

Figure 10 and 11. The male T-pose model
left: mesh decomposition, right: rigid bodies

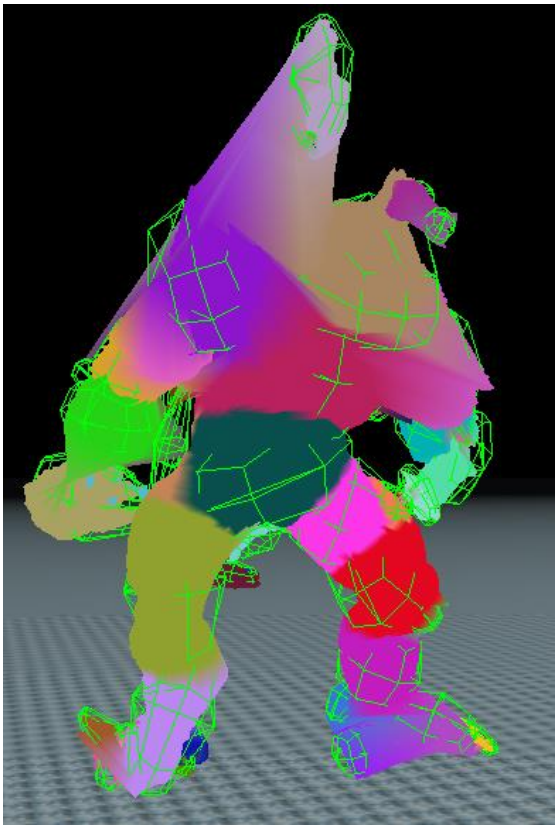

Figure 12. The high resolution armadillo model, greatly deformed due to colliding rigid bodies

There were also some minor problems with certain test models that have a lot of self-occlusion in their projection. The armadillo model (figure 13), for instance, has its head partially in front of its upper torso and its tail is pointed backwards, perpendicular to the first projection. The head and tail areas in the projection then have a lot of vertices close together and some vertices may even share 2D positions. This can produce certain artifacts like jagged (zigzag shaped) 3D silhouettes, instead of straight lines. This is caused by the fact that the alpha shape method only uses the 2D vertices, so the

consecutive vertices in the 2D silhouette may connect differently in the 3D silhouette. Fortunately, the jaggedness only occurs in the z-direction. This means that the curve skeleton extraction is unaffected, since that only uses the x and y-coordinate values.

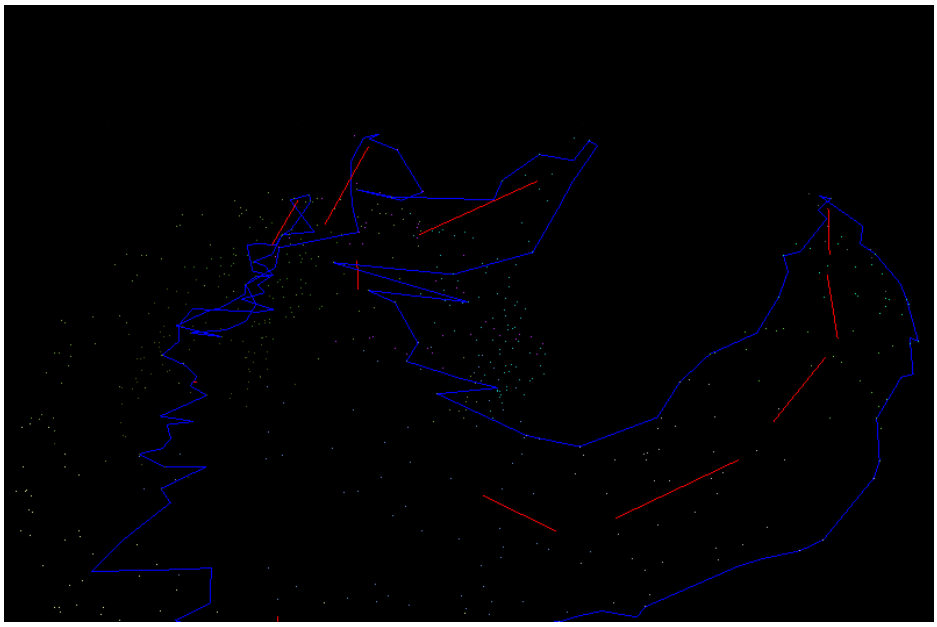

Figure 13. Perspective view of the armadillo 3D silhouette (blue), note the jaggedness of the edges

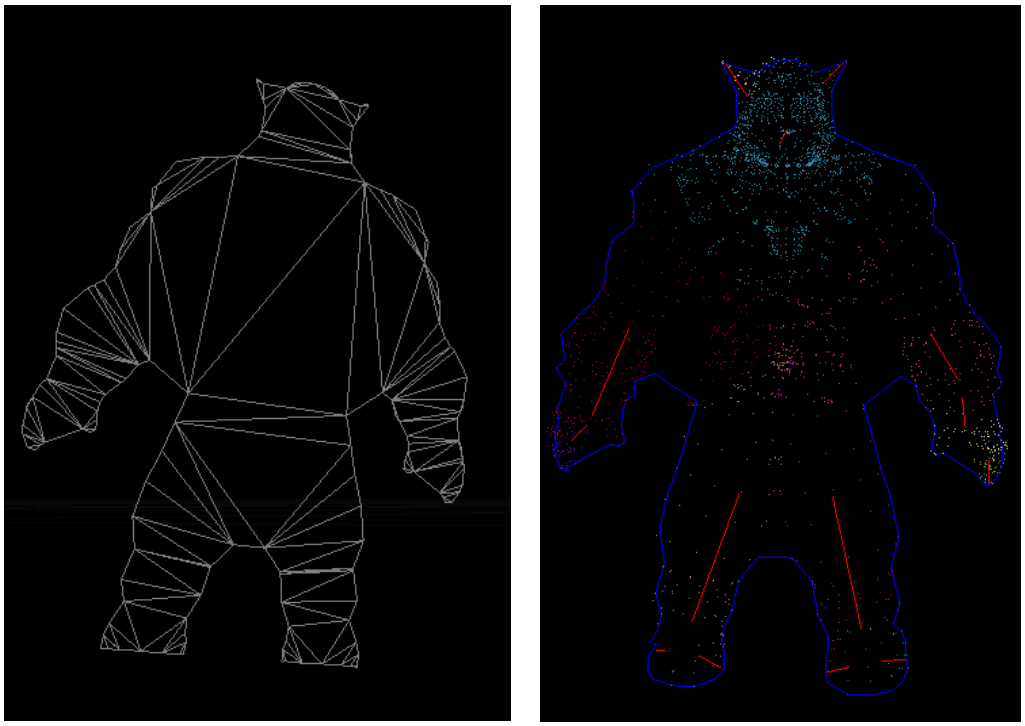

Figure 14. The Sinbad model, note that the triangulation (left) does not produce a spine for the curve skeleton (right)

The biggest remark, however, is that this method primarily depends on how the Delaunay triangulation looks like. Because you don't have any influence on how the triangulation turns out, it can occur that at some parts in the model there won't be any junction triangle on which a skeletal joint can be formed while you might expect one. Or vice versa, that a junction triangle is formed on a position where you don't want a skeletal joint.

A good example of this can be seen in figure 14. The torso of the Sinbad model is completely occupied by two large junction triangles. This means that there won't be a medial axis that can function as spine, which in its turn can't be used to form a body segment. The result is that this model is missing a rigid body for its torso, which is very undesirable.

5 Conclusion

This skeleton extraction method can still be improved a lot in the triangulation part. If the triangulation can be influenced in some way that a proper decomposition can be formed, and also that the afterwards filtering of 'short' medial axes to prevent artifacts won't be necessary, the animation skeleton would be a lot more practical in use.

The ragdoll can obviously be improved by fixing the several bugs that I failed to fix, but also the way rigid bodies are created should be revised. It occurs too often that the created rigid bodies are intersecting with each other, thus deforming the model.

Overall, this method is a great tool to quickly create an animation skeleton, which could serve as the base of a manually constructed skeleton. Since the computation time is very low it can be easily applied to a model for a quick skeleton preview or working base. Although the ragdoll creation still leaves much to be desired, I think it has potential to become a handy tool for quick (real-time) automatic ragdoll construction.

6 References

[1] J. Pan, X. Yang, X. Xie, P. Willis, J. J. Zhang (2009), "Automatic rigging for animation characters with 3D silhouette", *Computer Animation and Virtual Worlds 2009, Volume 20*, pp. 121-131.

[2] T. Igarashi, S. Matsuoko, H. Tanaka (1999), "Teddy: a sketching interface for 3D freeform design", *Proceedings of ACM SIGGRAPH 99*, pp. 212-219.

[3] K. Au, L. Tai, K. Chu, O. Daniel, Y. Lee (2008), "Skeleton Extraction by Mesh Contraction", *Proceedings of ACM SIGGRAPH 2008*, pp. 124-132.

[4] L. Wade, R. E. Parent (2002), "Automated generation of control skeletons for use in animation", *The Visual Computer 2002, Volume 18, Number 2*, pp. 97-110.

[5] S. Katz, A Tal (2003), "Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts", *SIGGRAPH '03 ACM SIGGRAPH 2003 Papers*, pp. 954-961.

[6] P. C. Liu, F. C. Wu, W. C. Ma, R. H. Liang, M. Ouhyoung (2003), "Automatic animation skeleton using repulsive force field", *Computer Graphics and Applications, 2003*, pp. 409-413.

[7] I. Baran, J. Popović (2007), "Automatic Rigging and Animation of 3D Characters", *Proceedings of ACM SIGGRAPH 2007, Volume 26, Issue 3*, Article No. 72.

[8] S. Liu, C. C. L. Wang, K.-C. Hui, X. Jin, H. Zhao (2007), "Ellipsoid-tree construction for solid objects", *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp. 303-308.

[9] M. Bae, J. Kim, Y. J. Kim (2012), "User-guided volumetric approximation using swept sphere volumes for physically based animation", *Computer Animation and Virtual Worlds 2012, Volume 23*, pp. 385-394.

[10] T. K. F. Da (2013), "CGAL 4.3 - 2D Alpha Shapes: User Manual", http://doc.cgal.org/latest/Alpha_shapes_2/index.html, accessed on February 3, 2014.