

USING DENSITY-BASED CLUSTERING TO IMPROVE
SKELETON EMBEDDING IN THE PINOCCHIO AUTOMATIC
RIGGING SYSTEM

by

HAOLEI WANG

B.S., Dongbei University of Finance and Economics, 2009

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2012

Approved by:

Major Professor
William H. Hsu

Abstract

Automatic rigging is a targeting approach that takes a 3-D character mesh and an adapted skeleton and automatically embeds it into the mesh. Automating the embedding step provides a savings over traditional character rigging approaches, which require manual guidance, at the cost of occasional errors in recognizing parts of the mesh and aligning bones of the skeleton with it. In this thesis, I examine the problem of reducing such errors in an auto-rigging system and apply a density-based clustering algorithm to correct errors in a particular system, Pinocchio (Baran & Popovic, 2007). I show how the density-based clustering algorithm DBSCAN (*Ester et al.*, 1996) is able to filter out some impossible vertices to correct errors at character extremities (hair, hands, and feet) and those resulting from clothing that hides extremities such as legs.

Keywords: computer graphics, automatic rigging, skeleton embedding, character modeling, clustering algorithms

Copyright

Haolei Wang

2012

Table of Contents

Table of Contents	iv
List of Figures	vi
List of Tables	vii
Acknowledgements	viii
1 Introduction	1
2 Background and Related Work	5
2.1 Basic Concepts	5
2.1.1 Points and Vectors	5
2.1.2 Geometric Primitives	6
2.1.3 Joints, Skeleton and Degrees of Freedom(DOF)	6
2.1.4 Animation and motion capture	6
2.1.5 Automatic Rigging	7
2.2 Density-based approach	8
2.2.1 Basic Concepts	8
2.3 Previous Work	10
2.3.1 Skeleton Embedding and Skeleton Extraction	10
2.3.2 Distance Map and Medial Surface	11
2.4 Pinocchio API	13
2.4.1 Packed Spheres	13
2.4.2 Constructed Graph	13
2.4.3 Reduced Skeleton	13
2.4.4 Penalty Function	14
2.4.5 Discretized Embedding	15
2.4.6 Refinement	16
2.5 Problem Definition and Research Objective	16
3 Methodology	19
3.1 Clustering Algorithm	19
3.1.1 Possible Vertices	19
3.1.2 Cluster versus Z-value	21
3.1.3 Algorithm	22
3.2 Adjustments	25

4 Experiments and Limitation	31
4.1 Applicable Scope	31
4.2 Testing System Limits	33
5 Performance and Discussion	35
6 Conclusions and Future Work	37
Bibliography	40

List of Figures

1.1	Optional: Teddy: A Sketching Interface for 3D Freeform Design	2
1.2	Optional:All 16 test cases for Pinocchio.	3
2.1	Skeleton used in modeling. The cone-shaped objects represent bones while the spheres between them represent joints.	7
2.2	Human walking motion capture data. The points represent the positions of markers on testing body.	8
2.3	Density reachable (a) and density-connected (b). (Han and Kamber ¹¹) . . .	10
2.4	Distance map/field/transform. White grids represent feature points, which are margin points in this case and the grids with digits represent the background points.	11
2.5	Pinocchio API	12
3.1	The revised Pinocchio API with two new modules.	19
3.2	The clustering algorithm. Note that even though some vertices rest below the threshold, they are eliminated by the clustering process.	21
3.3	Adjustment for feet. The hip joint is selected to be standard for the alignment two feet.	26
3.4	Adjustment for hands. The lower hand is set to be the standard in order to create a symmetric joint position for another.	27
4.1	All three wrongly rigged characters are now correct.	32
4.2	Stretched and squashed characters used for testing the applicable scope of our method.	32
4.3	Extend test for our algorithm.	33
5.1	Statistic on tested cases.	35

List of Tables

- 2.1 Numbers generated in Pinocchio for joints of reduced and unreduced skeleton. 14

Acknowledgments

I am grateful to Baran and Popvic³ for letting me make use of their work as base to our research and providing source code as well. I would thank Baran and Popvic³, Igarashi et al.¹², Wade²¹ and Ester et al.⁹ for using their images for explanation. I also thank my advisor Dr. William Hsu for his assistance and support during the work. Finally my thanks to my other committee members Dr.Daniel A. Anderson and Dr.Mitch Neilsen.

Chapter 1

Introduction

Animated 3D models are widely used among filming, gaming, commercial industries and even research programs for the beauty of their natural, life-like performance. With them, people can present some fantastic or even impossible scenarios vividly without risking injury or damage through stunt work. However, with the increasing demand for 3D characters, the traditional process of hand-made characters are severely challenged by the quality of the work and efficiency. A more advanced approach is needed to meet this demand.

When a new 3D model is constructed, it is not yet suitable for animation. People who wants to animate a certain character has to first insert joints into the mesh and connect them to simulate bones for all living creatures. After the connection of bone structure, each bone is dispatched to control certain part of the character so that this portion of the body will move when the corresponding bone makes a transition. Finally people set up keyframes and rig the character by translating and rotating these bones as they planned one frame after another till the animation ends. These steps are part of the typical procedure associated with a single second of animation on the screen.

There are some advantages in doing so: people do not have to be limited by those rules in the real world and can rig the character to perform some twisted or impossible behaviors (like flying and transformation). Also, by keyframe rigging people can grasp more precise control over the animation. However, the drawbacks are also significant. First, such a process requires certain amount of knowledge in computer graphics and people who don't

own the knowledge may not be able to perform the process. Second, the manually rigged characters act more or less inauthentic to real life that even experts need to go back and forth to check the consistency of the animation. Third, at most of the time, the characters of their products fall into the category of normal creatures that they share approximate body skeleton and behave roughly the same. It will be a time consuming task to reiterate this process if no automatic tools come to help.

Igarashi et al.¹² becomes one of the precursors who involve in developing user-friendly system that simplify this process. Teddy, which was presented by them in 1999, is a gesture-based sketching interface for 3D freedom design that people can produce simple shaped objects with minimum widgets and operations. Teddy did not involve animation processing but did give a hint on finding the topology of the mesh that it triangulated the polygon to estimate the spine of the object. Igarashi et al.¹²'s later work focused on animation control but these work required input models to be articulated ones and required input models to be articulated and manipulations to be as rigid as possible, and used space-time constraints to compute result animations.



Figure 1.1: *Teddy: A Sketching Interface for 3D Freeform Design*

In this paper we present an improved version of Baran and Popvic³, an automatic rigging and animation system called Pinocchio. Comparing to previous approaches, Pinocchio

focuses on finding joint positions and transferring animations from existing motion data that reduces the complexity of the conventional process. It takes a 3D mesh and a generic skeleton (with motion data) as input and outputs the adapted skeleton to the mesh and embeds it into the mesh and grants motion to the mesh. Pinocchio differs from those methods we aforementioned because it concentrates on matching the joint positions using a combination of weights on models topological structure and uses motion data, a more natural and precise yet easier way to present the animation. Although Pinocchio gives quite satisfying results under most cases, we find some issues that can be improved in Pinocchio. Among 16 test cases in their paper, 3 out of them are wrongly rigged. This becomes the objective of our work. We will introduce an revised Pinocchio pipeline to correct these problems by adding two new modules to the original process, one to correct some noise vertices using the concept of density-reachability from data mining and another to correct the inappropriate limb joint positions.

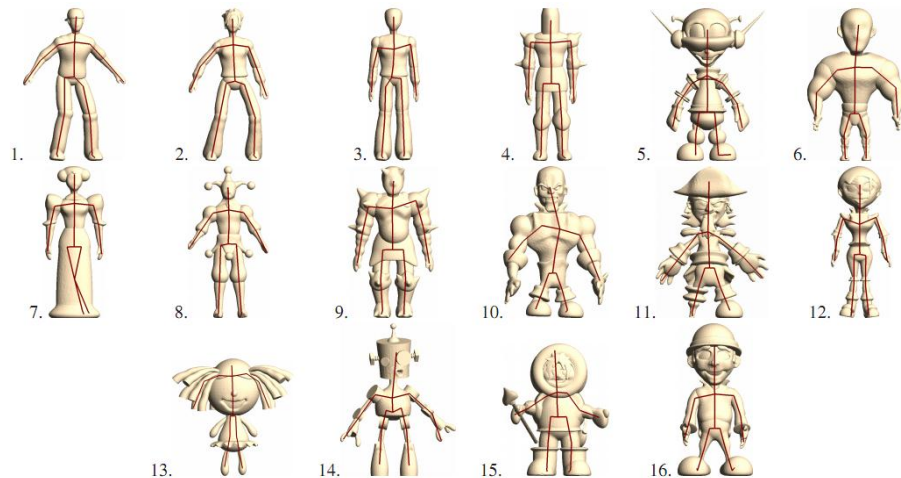


Figure 1.2: *“Test Results for Skeleton Embedding”*. Note that character 7, 10 and 13 are wrongly rigged.

We will introduce the background and summarize the techniques used in Pinocchio and discover the problems appear in the results in Chapter 2 . In chapter 3 we will describe the two new modules and how we apply them to Pinocchio. In chapter 4 we will demonstrate the designed experiments and test results. Then, in chapter 5 we will discuss the performance

and applicable range after the revision and talks about the balance between generality and precision in auto-rigging problem. Finally we conclude our work and touch upon the possible future work in Section 6.

Chapter 2

Background and Related Work

Before we survey previous work related to this topic, there are some basic and important concepts that are shared across all the methods: meshes (geometric primitives), skeletons and animations. This serves as a quick overview over some fundamental concepts for those who don't have graphics backgrounds. We summarize concepts from Marc-Guindon¹⁷, Apodaca and Gritz², and Eberly⁸, for a more completed tutorial the reader are recommended to these books.

2.1 Basic Concepts

2.1.1 Points and Vectors

We all know that a point in 3D space can be explained as a tuple (x,y,z) when a coordinate system is established. By putting a 3D model into this space, we can think of the model as a collection of vertices. Any point p can be described as $p(x,y,z)$ where x , y , z respectively represent a real value projected to its corresponding axis. A vector is used to depict the difference between two points. Given two points p and q , a vector $\vec{V}=q-p$ is used to represent a directed edge that goes from p to q . The calculations between vectors have different purpose than that of digits. For example, given two vectors V_1 and V_2 , $V_1 \bullet V_2 = \|V_1\| \|V_2\| \cos \theta$, where θ is the angle between the two vectors.

2.1.2 Geometric Primitives

All 3D models are composed of basic geometric primitives. Commonly used geometric primitives in 3D include *cubes*, *spheres*, *columns*, *tetrahedrons* etc. These primitives are considered basic to any other shapes. Each geometric primitives is composed by some geometric faces and each face is composed by some vertices - points in the Cartesian coordinate system and edges - the line segment between them. Thus, terms such as “3-D character”, “3-D mesh”, or “3-D models” are typically defined using basic units of vertices and edges rather than geometric primitives. This is especially the case for computer calculations because computers can only recognize vertices but not polygons constructed by these vertices.

2.1.3 Joints, Skeleton and Degrees of Freedom(DOF)

Like living creatures, the body of 3D models are also controlled by their skeleton. In graphics we call the location between two bones a joint. Joints are constructed to allow movement and provide mechanical support for the model. We place joints in the right place to represent articulations and we connect these joints to get bones. We retrieve a skeletal structure of the model by connecting all the joints and bones together. Like us, different joints can rotate within different range. We call the valid displacement of these joints *degrees of freedom* (DOFs). Joints are thus classified by their DOFs. For example our wrist articulations are one-degree joint that they can only swing up and down while the elbow joints are two-degree joints that they have one more degree of freedom. By placing skeleton in the models, we are able to control the whole model with designated movements that form animations.

2.1.4 Animation and motion capture

Animation is the display of consecutive images. To be specific in 3D animation, we keyframe the models, posing them differently in each frame and save all the frames and play them in sequence. Animating a 3D character requires constructing a skeletal structure of the model and embed the skeleton in to the model so that each part of the body mesh will be controlled

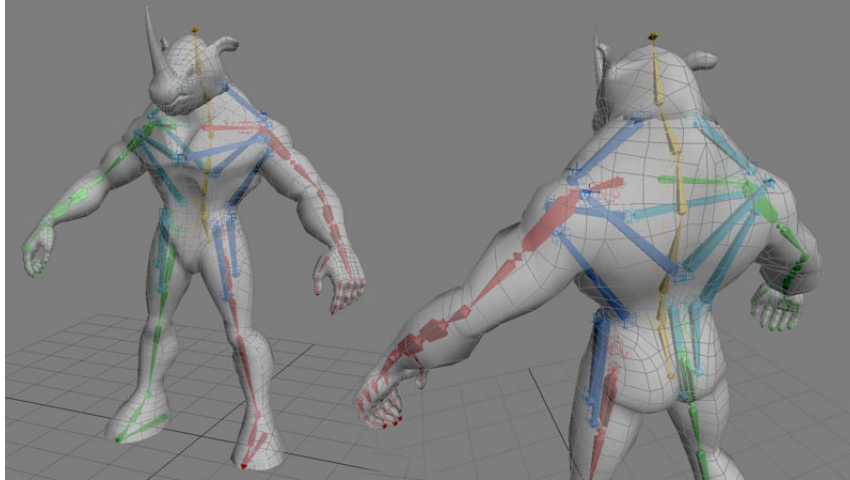


Figure 2.1: *Skeleton used in modeling. The cone-shaped objects represent bones while the spheres between them represent joints.*

by certain bones. We should differentiate animation from motion capture. “Motion capture or mocap, are terms used to describe the process of recording movement and translating that movement on to a digital model. The mocap data simply refers to the data after the translation. In term of quality and time-effectiveness, mo-cap data outstrips the conventional ways. These features along with its perfect reusage makes auto-rig technique more superior to previous work.

2.1.5 Automatic Rigging

Automatic rigging refers to the process of creating articulated figures by embedding computer generated skeletal structure into the given models and performing animation using predefined animations. Automatic rigging techniques generate a skeleton and an attachment. A skeleton is a collection of joints and bones that fit inside the model while an attachment defines how the model is anchored to the control the skeleton. Since computers are not able to recognize different parts of the body and there’s no information by default to provide data to infer the body sturcture, plenty of approaches were invented to achieve this goal.

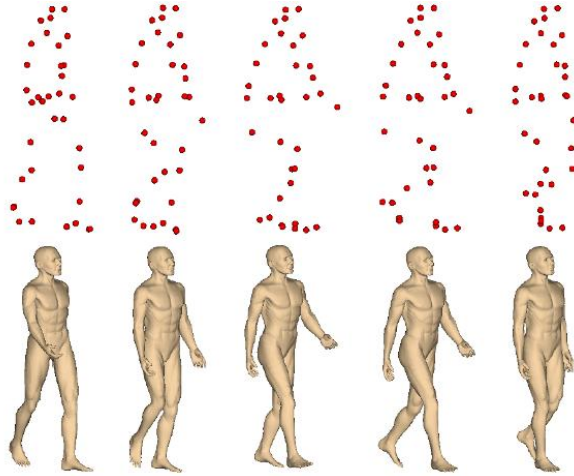


Figure 2.2: *Human walking motion capture data. The points represent the positions of markers on testing body.*

2.2 Density-based approach

The intuition we introduce this approach stems from the observation of the misplaced models generated by Pinocchio. Baran and Popvic³ established a general rule in judging whether a selected vertex is eligible for embedding or not using penalty functions. However, this rule is inadequate to filter out some similar shaped parts that satisfy this rule, such as the hair in character 13 that was recognized as the doll's arm. The situation inspired us to further modify the existing process that will help rule out these problems. Our initial attempt using z-value as a threshold would not entirely eliminate those parts as we do not own precise control over the vertices. Then the idea of cluster comes into the picture that if we are able to group up the vertices into clusters, we will possess the power of excluding all the violating vertices to get a correct result.

2.2.1 Basic Concepts

Clusters are used in finding group of data in spatial database. Ester et al.⁹ published their algorithm DBSCAN along with the very important concept of density-reachability that gives us a huge clue on our problem. We summarize their technique here for a comprehensive

understanding of this idea.

When trying to group large amount of data in database, people found that within a cluster, the density of data in the cluster was higher than that of the noise data. This distinction became the intuition for them to invent an algorithm that could automatically recognize clusters based on the density of data. A series of concept should be made clear to help understanding the idea of density-reachable that we used in our approach. We recap some of the important concepts below.

Eps-neighborhood of a point. We say the Eps-neighborhood of a point P(denoted as $N_{Eps}(p)$) is defined as $N_{Eps}(p)=\{ q \in D | \text{dist}(p,q) \leq Eps \}$, that if the distance between two points is less than Eps we say point q is an Eps-neighborhood of p. However, Eps itself could not distinguish whether a point belongs to a cluster or not. For instance, a noise point near the cluster has the same Eps distance as that within the cluster, then we can exclude this noise point. A requirement on number of points in distance was then added to hold the case. They called the second definition directly density reachable.

Directly density-reachable. Point p is directly density-reachable from point q if

- 1) $p \in N_{Eps}(q)$ and
- 2) $|N_{Eps}(q)| \geq \text{MinPts}$

where MinPts refers to the minimum required numbers of points in range. Note that this definition is symmetric to core points — points that lie inside of a cluster, but asymmetrical between a core point and a border point.

Density-reachable. A point is density reachable from a point q if there exists a chain of point $q \dots p$ that each two consecutive points are directly density-reachable.

Density-connected. A point p is density-connected to a point q if there is a point o such that both p and q are density-reachable from o.

Using the above concepts we are able to define cluster and noise. A cluster is a set of density-connected points that any two points inside it are density-connected. Then a noise point is defined as one that does not belong to any cluster.

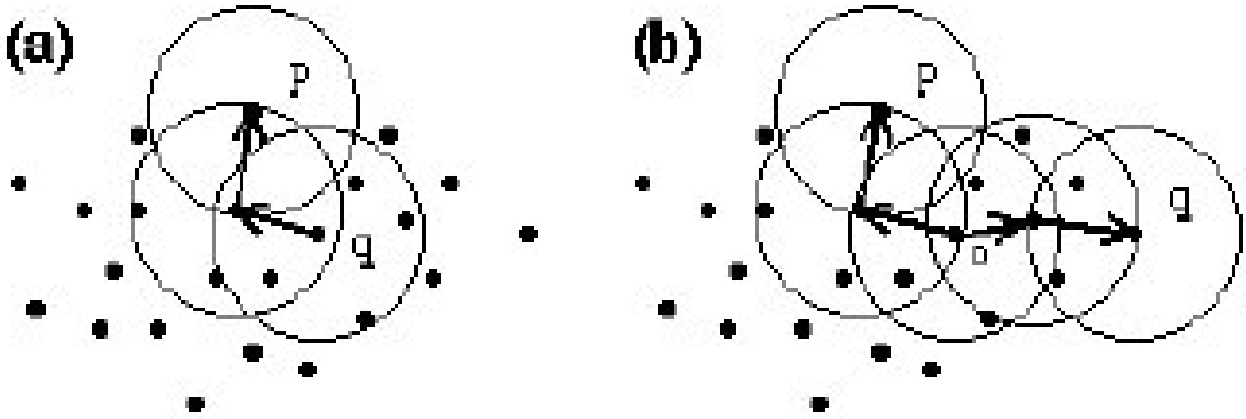


Figure 2.3: *Density reachable (a) and density-connected (b).* (Han and Kamber¹¹)

2.3 Previous Work

2.3.1 Skeleton Embedding and Skeleton Extraction

There are two ways of generating a skeleton. One is called skeleton extraction and another called skeleton embedding. Both aim to build up the skeleton based on the information provided by the model itself. In speaking of automatic rigging, Baran and Popvic³ pointed out that the advantage skeleton embedding has over skeleton extraction is that the topology of the extracting result may vary from the target skeleton and this may prevent computer from matching these two to finish the rigging. A.¹ used an approximate cluster to develop a hierarchical decomposition of a mesh, and extracted a skeleton out of it. E. et al.⁷]and C. et al.⁶ both used voxel descriptions to decompose the mesh and fit either ellipsoids or superquadrics to estimate a skeleton. Later S. et al.²⁰ developed a different segmentation approach based on feature point extraction. M. et al.¹⁵ also presented a similar decomposition method based on approximate convex decompositions of shape. J. et al.¹⁴ used feature points to estimate the skeleton from a static shape while other approaches used various types of distance functions C. et al.⁵, P. et al.¹⁸, gaussian curvature M. and e G.¹⁶ and probability distributions J. and M.¹³

2.3.2 Distance Map and Medial Surface

Wade²¹ introduced his approach in finding approximate medial surface using distance map. A distance map is the output of a grid of discrete points with each point marked either a feature point or a background point. A calculation is involved that iterates from those background points that are nearest to the feature points and ends till all the background points are calculated. Figure 2.3 shows the idea. The calculation varies depends on the desire of the distinction between points. If people want to exaggerate this distinction, they would use Euclidean Distance. In fact Pinocchio uses this because it exactly depicts the distance between two vertices in space. Pinocchio computes the distance map on an octree. It rescales the mesh to unit volume and builds up a kd-tree over the mesh. It then uses top down method to iteratively split every cell till they fall into the tolerance of the distance.

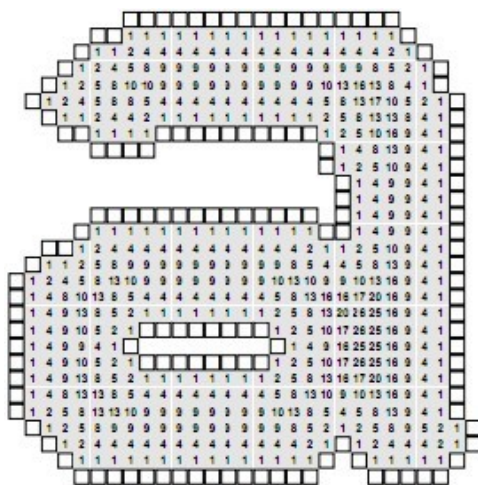


Figure 2.4: *Distance map/field/transform. White grids represent feature points, which are margin points in this case and the grids with digits represent the background points.*

Wade²¹ also illustrated how to construct the medial surface. We explained his idea of finding medial surface as such: for every background point in the distance map, build up a sphere with its Euclidean Distance in 3D space and see if the sphere will be well inside the mesh and touch more than three feature points. Then all the satisfying points group together to form the medial surface. Pinocchio works similarly but it filters out points that form 120 degrees and greater which are likely to be noisy parts.

Pinocchio thus takes two main steps to finish the rigging: skeleton embedding and skin attaching. Figure 2.4 shows all the modules of the API.

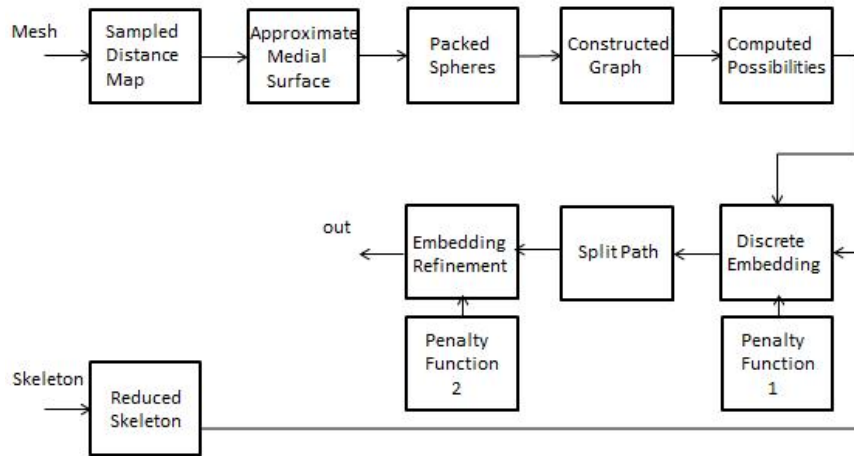


Figure 2.5: *Pinocchio API*

2.4 Pinocchio API

Pinocchio takes in two parameters: a 3D mesh and an adapted skeleton. It begins with the rescaled mesh and computes the distance map and approximate medial surface as we described in Section 2.3.2.

2.4.1 Packed Spheres

The next procedure involved in Pinocchio is packing spheres. It queues the medial surface points by their distance to the feature points in ascending order and constructs a sphere with a point as center and the distance as radius if this point is the largest existing point that is not in any existing sphere. In this way we diffuse the medial surface points and is prepared for next step.

2.4.2 Constructed Graph

The next step of discretization constructs a graph by connecting some of the sphere centers. Pinocchio connects two centers if the two spheres intersect or if it is necessary like the neck and the shoulder. For a more mathematical and formulated description, take a look at the paper for detail. After this step, Pinocchio finishes its discretization steps and ready to embed the skeleton into this graph. We need to illustrate the data structure used here for future convenience. For every vertex that displayed on the graph, Pinocchio allocates a `vector3 (x,y,z)` to store its coordinates and another vector for all the edges that come out of it. We will be needing this structure when we talk about our new algorithm.

2.4.3 Reduced Skeleton

For the input skeleton, Pinocchio reduces the bone numbers to simplify the embedding. All the bone chains have been merged to reduce the degrees of freedom. After successfully embed these joints, Pinocchio will reverse the reduced skeleton to the original unreduced one by adding those joints back according to the proportion of length in the bone chain.

Skeleton Joint	Unreduced#	Reduced#
shoulders	0	0
back	1	
hips	2	1
head	3	2
Lthigh	4	
Lknee	5	
Lankle	6	
Lfoot	7	3
Rthigh	8	
Rknee	9	
Rankle	10	
Rfoot	11	4
Lshoulder	12	
Lelbow	13	
Lhand	14	5
Rshoulder	15	
Relbow	16	
Rhand	17	6

Table 2.1: *Numbers generated in Pinocchio for joints of reduced and unreduced skeleton.*

We will primarily talk about human-shaped biped characters because they appear more frequently than others. Pinocchio can also deal with quadruped characters like horses and centuars and the techniques are similar. We will give a sequence of numbers that represents the order of these joints. We can see from figure that the unreduced skeleton has 18 joints but only 7 in the reduced skeleton. Pinocchio stores them in a vector that we will use in later chapters. 2.1 shows all the 18 joints appear in the unreduced skeleton and 7 joints appear in the reduced skeleton.

2.4.4 Penalty Function

The most important contribution Baran and Popvic⁴ have made to Pinocchio is the discrete penalty function. As they mentioned in their paper, this penalty function impacts greatly on generality and quality of the results. Nine minor penalty functions are combined together using a studied weight to balance between each other. The nine penalty functions are:

- It penalizes short bones. It penalizes a bone in the reduced skeleton if the length is too short comparing to that in the unreduced skeleton.
- It penalizes on the orientation between joints if the direction between two joints are different from that in the given skeleton.
- It penalizes different length of symmetric bones. This is self-explanatory.
- It penalizes bone chains that sharing vertices. If two or more bone chains share a vertex with a distance to mesh surface less than 0.2 or a shorter bone chain is overlapped by a longer one then a penalty will apply to these joints.
- It penalizes feet joints if they are not grounded.
- It penalizes zero length bones.
- It penalizes improperly oriented bone segments.
- It penalizes degree-one joints that should be further from their parent joints but are not.
- It penalizes joints that are embedded close to each other but are far along the bone path.

Baran and Popvic³ studied this penalty function through more than 400 of embeds to guarantee the generality and correctness of the coefficients appeared in these functions. Again, for detailed information please read their documentation on this penalty function. We summarize this function just for a preparation of discrete embedding.

2.4.5 Discretized Embedding

The discrete embedding step involves a branch-and-bound method to return an optimal solution from the search tree of possible embedments. Pinocchio creates a priority queue to maintain partial embedding information that ordered by the lower bound estimation.

Because the joints of the skeleton are in order as in figure ,and the penalty function consists of 9 weighted subfunctions and a term that incorporates the dependence between different positions, a lower bound is estimated by calculating the existing penalties and ignoring the un-embedded joints' dependence. In this way once all skeleton joints are embedded, it is guaranteed to be the optimal one.

2.4.6 Refinement

A continuous refinement is given after the initial embedding to adjust the orientation and length of the bones to make the skeleton more adaptive to the mesh. A new penalty function is given that combines four minor aspects of penalties:

- Pinocchio penalizes bones that are not near the center of the object.
- It penalizes bones that are too short when projected onto their counterparts in the given skeleton.
- It penalizes improperly oriented bones.
- It penalizes asymmetry in bones that are marked symmetric.

Note that some of the terms are similar to the penalty function we described earlier but they use different measurements and factors each time.

To this point Pinocchio has done its job in skeleton embedding. Skin attachment involves a separate method called *Linear Blend Skinning*(LBS) to allocate certain part of the mesh to a designated bone. We skip the introduction to this part since our research does not involve this section. For detailed information please read their document.

2.5 Problem Definition and Research Objective

I now illustrate the problem discovered in Pinocchio and give a intuitive idea of the solution. Given all 16 test results, Pinocchio managed to rig 13 of them with 3 of them incorrectly

embedded. As we can see in Figure 1, characters 7, 10 and 13 are incorrectly rigged. Character 7 is a female in dress and Pinocchio has no capability in detecting shape in early steps. Usually when the character is biped, the result of packing sphere will naturally give a biped shape because the boundary of the mesh constrains the radius of the spheres. However, in such a case, much wider space will force the system to choose approximate the center of the dress to pack its first (largest) sphere. Although Pinocchio places a penalty on them, it will not save the day under such an extreme case that the parameters set up in the penalty function are for normal biped characters. Character 10 and 13 illustrate the same problem: Pinocchio has no capability to eliminate vertices in impossible portions, like the glove in Character 10 and the hair in Character 13. It is almost certain that none of these problems can be solved before we can figure out which vertices are to be taken. Fortunately, we have such information gathered during the process. In fact, before embedding, Pinocchio computes the possibility for each vertex in the constructed graph and categorizes them into different groups. This becomes the entry point of our work. Consider character 13, the question is how Pinocchio confuses the hair with hand. When considering a vertex to be a candidate of limb vertex or not, Pinocchio checks every neighbour that has an edge connecting them in the constructed graph and see if (a) the sphere of the two vertices are relatively same and (b) angle between them is no more than 135 degree. This rules out most of the vertices that are well inside the mesh and those around big spheres like head, shoulder and hip. However, things like hair or tentacles or sharp edges such as armours will not be excluded. If we can further filter out these vertices then the results will perhaps be satisfiable.

A naive approach will choose the z-value of the vertices to be the threshold. Indeed, when we first did that, we corrected character 13 with a fast speeded search. However, two shortcomings are very obvious: (a) the threshold needs to be changed for every character and will lose the generality, (b) the accuracy of the threshold goes beyond human eye's capability. Consider that we set up the z-value threshold to be 0.6 of the body's height.

However, if a hair vertex has z -value 0.59, then the system will still give an undesirable result. to avoid such situations, we introduce the clustering algorithm.

Chapter 3

Methodology

I now explain my approach to fixing these problems. Figure 3.1 shows the revised API with two new modules: a filter consisting of a clustering algorithm to further eliminate some impossible vertices and an adjustment section to correct hands and feet issues.

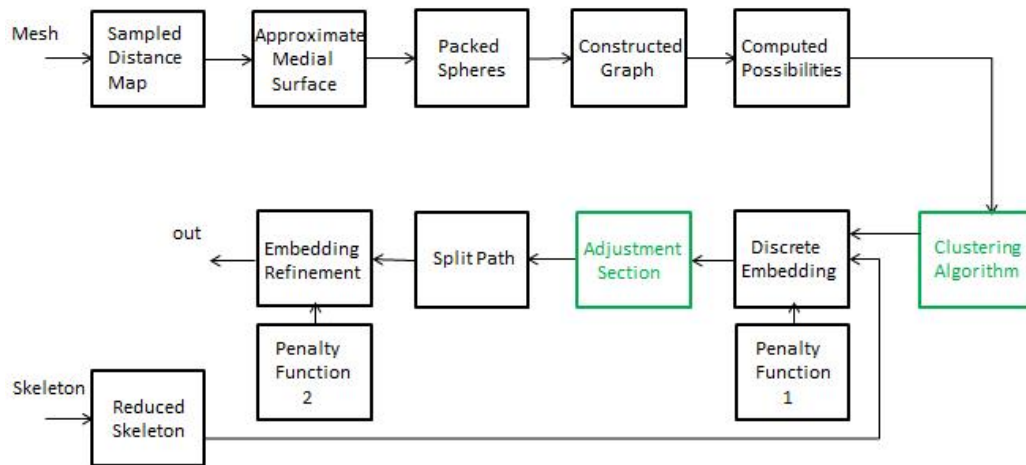


Figure 3.1: The revised Pinocchio API with two new modules.

3.1 Clustering Algorithm

3.1.1 Possible Vertices

We introduced the Pinocchio API in detail in Chapter 2 and we know that the discrete embedding is based upon a constructed graph and a reduced skeleton. There is a step

that was not described in their paper that raised our attention. This step computed a set of possible vertices that could most probably be the positions for the body joints. They categorized all the vertices found in the discrete graph into 3 groups: limb vertices, torso vertices and all vertices.

Limb Vertices. To estimate a vertex (or a sphere center in the graph) for a hand or foot joint, they set up two conditions:

1) For every vertex, traverse its neighbor and compare the radius of its sphere with that of itself. If any of the radius is two times larger than another, then this vertex will not be considered as a possible vertex. This is due to the concerning that typically the shape around ankle and wrist should stay stable so that the positions like neck and hip can be eliminated by this condition.

2) Given a vertex p and any two from their neighbors o and q , if it satisfies below equation, then it should be eliminated:

$$(\vec{o} - \vec{p}).\text{normalize}() \times (\vec{p} - \vec{q}).\text{normalize}() \geq 0.8$$

We know that the product of two normal vectors equals to the arccos amount of the angle between them. Given the arccos graph, we can see that the requirement is more than 135° . With this condition some similar structure with unsatisfying angles will be excluded.

Torso Vertices. To estimate a vertex on the spine, Pinocchio simply sorts all the vertices by the radius of the spheres in ascending order and see if the number of vertices exceeds 50 or not. If so, the first 50 vertices are stored as candidate of the joint positions and the rest are excluded. Otherwise all the vertices should be included.

All Vertices. In case that the above method did a wrong guess, all the vertices are still stored once for the worst case. Now we take a look at the results for Character 13. Two vertices at the end of the doll's hair were mistaken to be the positions of hand joints. This is because those two vertices satisfied all the conditions listed above in limb vertices and when being embedded, as those vertices were queued earlier in the priority queue, the system picked them up and finished the embedment. If we guess it right, all the hair vertices

at the similar position should be included in the limb vertices set where there's no way for the system to choose the right ones.

3.1.2 Cluster versus Z-value

Our algorithm aims eliminating all these incorrect vertices. We decide to add a filter to the limb vertices section. The very first idea is to use the height, that is the z-value feature of these vertices to make the distinguishment. To do so, we establish a threshold for the z-value of these vertices and exclude those that exceed this threshold. We managed to do that. However, then we realized that this fixed threshold can not hold for every other models and a more crucial difficulty is the precision of this threshold. Because Pinocchio rescale the model to unit height and typically a model consists of 10,000 to 100,000 vertices, each vertex reaches the order of magnitude up to $1e^{-7}$ of unit length, we can not set up a threshold to guarantee all the impossible vertices to be eliminated. That's how we come up to the clustering algorithm.

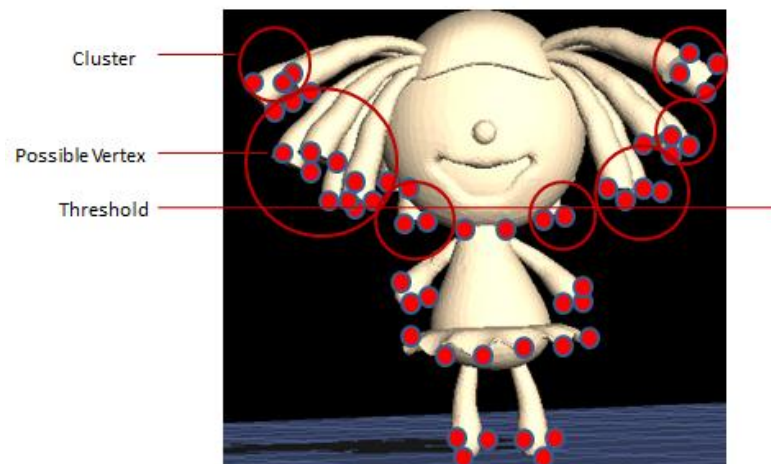


Figure 3.2: *The clustering algorithm. Note that even though some vertices rest below the threshold, they are eliminated by the clustering process.*

As you can see in figure 3.2 ,we manually set up a threshold at a approximate height, then some of the vertices are eliminated and some are not. However, if we first gather these vertices as clusters, then eliminate the whole cluster when one of the cluster member is

eliminated, then we are happy to see that all the impossible vertices are eliminated.

3.1.3 Algorithm

The algorithm works as follow: When the limb vertices section ends, we collect all the candidate vertices and start clustering:

- 1) Get the maximum height of all the vertices, the z-value h . We use this value to determine the Eps distance and the threshold.
- 2) Traverse the collection of the candidate vertices. If there's no cluster exists, calculate the distance from this vertex to all its neighbors and see if any of them satisfies the Eps distance. If so, put both this vertex and the neighbor to a new cluster and eliminate them from the collection. If not, mark this vertex as a noise vertex and eliminate it from the collection.
- 3) If there are some existing clusters, first try if this vertex is density-connected to any of them, if so, append this vertex to that cluster and eliminate it from the collection. If not, iterate the second step and either form a new cluster or mark it as a noise vertex.

The pseudocode of the algorithm is given below:

Clustering Algorithm takes(vector candidateVerts,ConstructedGraph graph) returns vector

```
for  $i = 1 \rightarrow graph.size()$  do
     $zMax \leftarrow Max(graph[i][2])$  {zMax gets highest z-value of the model}
end for
 $threshold = 0.6 * zMax$  {set up threshold}
for  $i = 1 \rightarrow candidateVerts.size()$  do
    if  $flag(graph[candidateVerts[i]]) == false$  then
        {false before it is evaluated}
        if  $clusters.size() > 0$  then
            {and we have already got some clusters before}
```

```

for  $j = 1 \rightarrow clusters.size()$  do
  if  $densityConnected(graph[candidateVerts[i]], graph[clusters[j][0]])$  then
     $clusters[j] \leftarrow candidateVerts[i]$ 
     $flag(graph[candidateVerts[i]]) = true$ 
  end if
end for
if  $flag(graph[candidateVerts[i]]) == false$  then
  {fit no existing clusters}
  for  $j = 1 \rightarrow candidateVerts.size()$  do
    if  $flag(graph[candidateVerts[i]]) == false \ \&\&$ 
     $Eps(graph[candidateVerts[i], graph[candidateVerts[j]])$  then
       $clusters \leftarrow newCluster(candidateVerts[i], candidateVerts[j])$ 
       $flag(graph[candidateVerts[i]]) = true$ 
       $flag(graph[candidateVerts[j]]) = true$ 
    end if {see if a new cluster will form}
  end for
else
   $noise \leftarrow candidateVerts[i]$ 
   $flag(graph[candidateVerts[i]]) = true$  {all failed, this is a noise vertex}
end if
else
  {no cluster formed yet, try to form one}
  for  $j = 1 \rightarrow candidateVerts.size()$  do
    if  $flag(graph[candidateVerts[i]]) == false \ \&\&$ 
     $Eps(graph[candidateVerts[i], graph[candidateVerts[j]])$  then
      {see if a new cluster will form}
       $clusters \leftarrow newCluster(candidateVerts[i], candidateVerts[j])$ 
    end if
  end for

```

```

        flag(graph[candidateVerts[i]]) = true
    end if
end for
if flag(graph[candidateVerts[i]]) == false then
    noise ← candidateVerts[i]
    flag(graph[candidateVerts[i]]) = true
end if
end if
end if
{continue if this vertex has been evaluated}
end for {evaluation phase finished, now begin elimination phase} new vector out;
for i = 1 → clusters.size() do
    for j = 1 → cluster[i].size() do
        if graph[cluster[i][j]][2] > threshold then
            {comparing z-values}
            break;
        end if
    end for
    if j < cluster[i].size() then
        {traversal not finished, some vertices above threshold, put this cluster into candidate
        list }
        out ← cluster[i];
    end if
end for
for all noise[i] do
    if graph[noise[i]][2] < threshold then
        noise.delete[i];

```



```
    end if
end for out ← noise;
return out;
```

The inputs of the algorithm are the constructed graph and the collection of all the candidate vertices from limb vertices function. The output is a subset of the collection that satisfies our conditions.

There are two parameters that we set up manually after a series of experiments: the threshold h and the Eps distance d . For the threshold we choose $h=0.6 * \text{Max}(z\text{-value})$ that holds for most of the cases. It is true that this threshold will malfunction once the impossible vertices break through the height, but as no algorithm can deal with all the possibilities, we did find slight improvement to the results. For another parameter, the Eps distance, we set $d=0.01 * \text{Max}(z\text{-value})$ so that this value would not be too large to include some weird vertices but too small to leave some of the cluster members as noise points. We will talk about the expanded test on the range and efficiency of this algorithm in the following chapters.

3.2 Adjustments

We managed to fix the problem appeared in Character 13, and another observation found that the rest two cases fell into a similar situation: Pinocchio could not adjust the position of symmetric joints like hands and feet. In Character 7 the left foot was wrong because the character was in dress and Pinocchio has no capability in detecting the outline of the mesh. After the discretization steps, the largest spheres in the center of the dress were first added but the correct ones were eliminated because they intersected with this large sphere. In character 10 the slant skeleton stems from the incorrectly embedded left hand. Pinocchio misplaced the hand joint onto a vertex on the edge of his glove. This vertex survived because it met all the conditions in limb vertices section and it was low enough that our threshold

would never eliminate it. We thus separately placed a sequence of adjustment function to correct these issues.

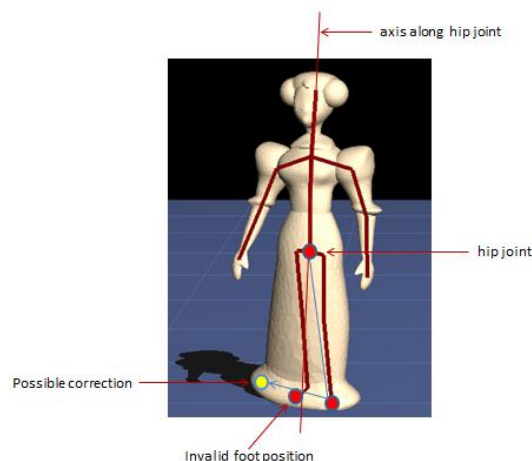


Figure 3.3: *Adjustment for feet. The hip joint is selected to be standard for the alignment two feet.*

We noticed that almost all the cases were correctly recognized and embedded the torso vertices such as head, neck, spine and hip. We chose the vertex of hip joint to be the standard of our adjustment because in the reduced skeleton two feet joints were directly connected to this joint. Here we call the vertex at hip joint hip , the left foot lf and the right foot rf . So in Cartesian coordinate we have $hip(x,y,z)$, $lf(x,y,z)$ and $rf(x,y,z)$. Under normal case the x -value from right foot to hip and from hip to left foot should be positive and relatively equal, i.e., $rf(x)-hip(x) \approx hip(x)-lf(x) > 0$. If such a case like character 7 was detected violating this measurement, we would input these three vertices together with the collection of all possible vertices to the function to make adjustment. The algorithm is given below:

- 1) If $rf(x)-hip(x) < 0$, then right foot is the one to be adjusted, if $hip(x)-lf(x) < 0$, then left foot is the one to be adjusted.
- 2) Assuming the left foot (as in Character 7) is the one to be adjusted, we manually create a correct position for it using the right foot: $new\ lf(x,y,z) = lf'(hip(x)-[rf(x)-hip(x)], lf(y), lf(z))$
- 3) We traverse all the candidate vertices and find one that lies nearest to this fate vertex

and order this vertex to be the new position of the left foot. The opposite situation is symmetric.

Notice that we don't have the case that $rf(x)-hip(x)<0$ and $hip(x)-lf(x)<0$, which means the two feet lie across each other. This is impossible because Pinocchio embeds these two joints independently and if there is a possible vertex on the left side, it will never embed the left foot to the right side.

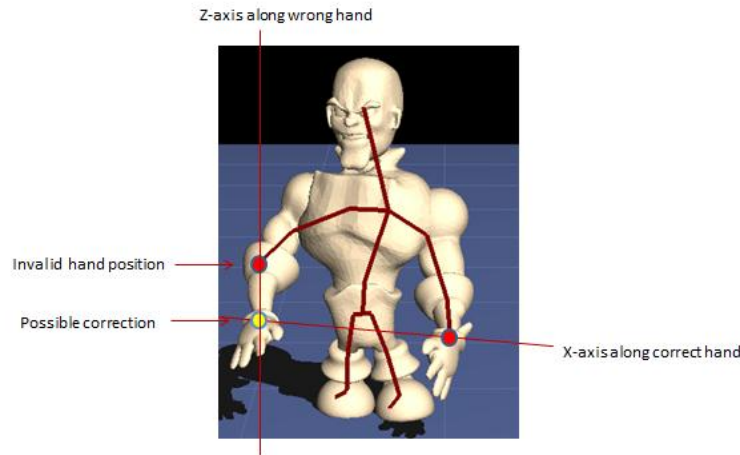


Figure 3.4: *Adjustment for hands. The lower hand is set to be the standard in order to create a symmetric joint position for another.*

We found hip joint to be the measurement of foot position. Unfortunately we were not able to find such a standard in dealing with hand adjustment. However, we noticed that all the mistakes that happened on hands were inside the arms, in other words, we did not run into a situation where Pinocchio embedded a hand correctly in its position while having another one embedded in the torso or leg. Then we could assert that the lower side of the embedding would be more trustworthy. Again we suppose the left hand to be $lh(x,y,z)$ and right hand to be $rh(x,y,z)$. Here's the algorithm:

- 1) If $lh(y)<rh(y)$, then we assume the right hand to be adjusted and the left hand if $lh(y)>rh(y)$.
- 2) Assuming left hand is the one to be adjusted. then we create a new vertex for it: $new\ lh(x,y,z) = lh'(lh(x),rh(y),lh(z))$.

3) We traverse all the candidate vertices to find the nearest point p and p will be the new position for the left hand.

And the pseudocode shows the integrated module we developed for Pinocchio. The inputs are the set of candidate vertices, the constructed graph and the set of initial embedding results.

```
embeddingAdjustment takes(embeddingResults ER, constructedGraph CG, candidateV-
verts CV)
```

```
returns void
```

```
vertex3 v1=graph[ER[1]] {skeleton hip}
```

```
vertex3 v3=graph[ER[3]] {skeleton left foot}
```

```
vertex3 v4=graph[ER[4]] {skeleton right foot}
```

```
vertex3 v5=graph[ER[5]] {skeleton left hand}
```

```
vertex3 v6=graph[ER[6]] {skeleton right hand} {adjustment of hands}
```

```
if v5[2] > v6[2] then
```

```
    {adjust left hand}
```

```
    vertex3 vFab(v5[0],v5[1],v6[2]);
```

```
    {construct a new left hand position}
```

```
    for  $i = 1 \rightarrow CV.size()$  do
```

```
        if  $dist(CG[CV[i]], vFab) < minDist$  then
```

```
            {minDist stores shortest distance}
```

```
             $minDist = dist(CG[CV[i]], vFab)$  ;
```

```
            vAdj=CV[i];
```

```
        end if
```

```
    end for
```

```
    v5=vAdj;
```

```
else
```

```
    vertex3 vFab(v6[0],v6[1],v5[2]); {right hand}
```

```

for  $i = 1 \rightarrow CV.size()$  do
  if  $dist(CG[CV[i]], vFab) < minDist$  then
    {minDist stores shortest distance}
     $minDist = dist(CG[CV[i]], vFab)$  ;
     $vAdj = CV[i]$ ;
  end if
end for
 $v6 = vAdj$ ;
end if{feet adjustment}
if  $(v1[0] - v3[0]) < 0$  then
  {left foot is misplaced}
   $vertex3\ vFab(v1[0] - (v4[0] - v1[0]), v3[1], v3[2])$ ;
  for  $i = 1 \rightarrow CV.size()$  do
    if  $dist(CG[CV[i]], vFab) < minDist$  then
      {minDist stores shortest distance}
       $minDist = dist(CG[CV[i]], vFab)$  ;
       $vAdj = CV[i]$ ;
    end if
  end for
   $v3 = vAdj$ ;
else if  $(v4[0] - v1[0]) < 0$  then
  {(})right foot issue)
   $vertex3\ vFab(v1[0] + (v1[0] - v3[0]), v4[1], v4[2])$ ;
  for  $i = 1 \rightarrow CV.size()$  do
    if  $dist(CG[CV[i]], vFab) < minDist$  then
      {minDist stores shortest distance}
       $minDist = dist(CG[CV[i]], vFab)$  ;

```

```

    vAdj=CV[i];
  end if
end for
v4=vAdj;
end if
return ;

```

Vertices v_1 , v_3 , v_4 , v_5 and v_6 represent the hip, left foot, right foot, left hand and right hand joint respectively according to the reduced skeletal structure 2.1. We did not consider the different in x and y value because normally the model should be presented in T-pose or standard standing pose and we assumed this is the case. A faster searching strategy involves storing the clusters we previously calculated and eliminating those with unmatched z-values. We did not try that since the number of vertices in the constructed graph is usually under 15,000 and the time usage of this module takes less than 1% of total run time.

Chapter 4

Experiments and Limitation

We have presented our approach in fixing the problems occurred in Pinocchio. To test our method, we observe its performance on the test cases that Pinocchio used. Our minimum goal is to fix all three incorrectly rigged characters while maintain the correctness on the rest 13 characters. Our ideal goal is to extend the applicable scope of the system that correctly rig as many characters as possible.

4.1 Applicable Scope

We tested all the 16 cases used for testing Pinocchio. The result was delightful that all 3 cases were rectified and the new modules did not affect the correctness on rest 13 cases. Figure shows the result. Character 7, the woman in dress was remedied by the adjustment that the left foot was placed in an approximate symmetric position to the right foot. Character 10's left hand joint was moved down to the hand mesh and then the whole skeleton was adjusted to fit the mesh. We eliminated the hair vertices in Character 13 and thus the system was then able to recognize the hand positions of the character.

We also built up a simple model and stretched or squashed it into four different copies, as shown in figure. We then tried to place the same skeleton into these models. Model(A) was a x-axis stretched model and although the hip joint was embedded higher than the suggested place, we found the animation worked out well without obstruction. Model(B) was a y-axis stretched model that was embedded well. Model(C) in principle violated the precondition

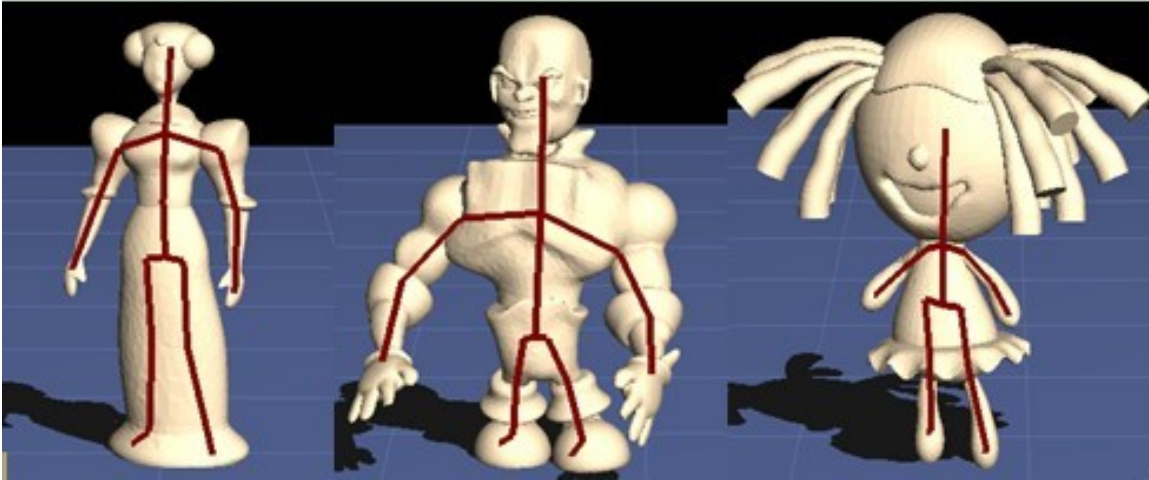


Figure 4.1: *All three wrongly rigged characters are now correct.*

of Pinocchio that the model should be proportioned roughly the same to the skeleton. We found two hand joints were embedded outside the mesh but we suggest that it should be correct under normal case. Model(D) was squashed with short legs and the skeleton was placed well inside the mesh.

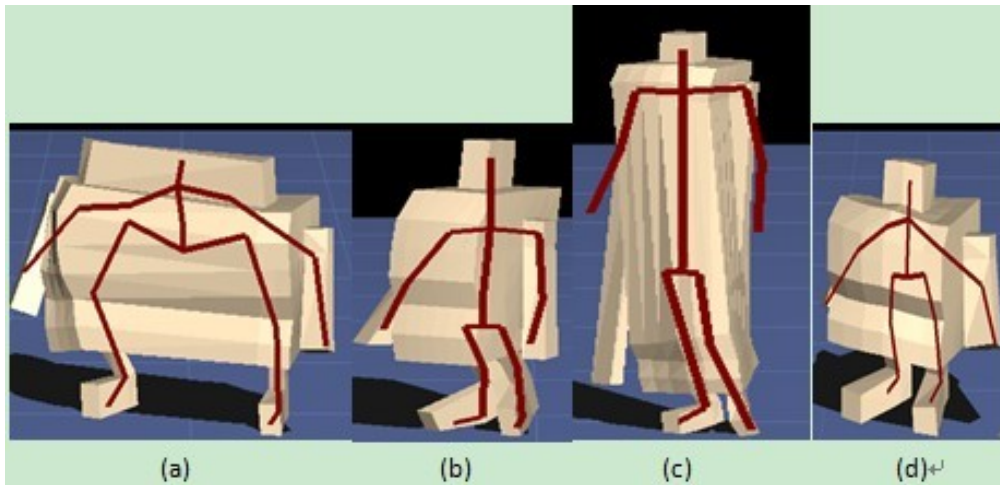


Figure 4.2: *Stretched and squashed characters used for testing the applicable scope of our method.*

4.2 Testing System Limits

We designed an extend test to discover the boundaries and limitation of our algorithm. We edited Character 13 and modified it into several cases to test the border of our algorithm. From figure 4.2 we can see that in model(a) the doll's hair was stretched down to the lower body and the proportion was changed by elongating its legs. Model(b) also stretched the hair but stayed in the same proportion. We only changed the proportion to the doll in model(c) by extending its legs to an extreme extent. We did the same to model(d) except that the proportion was set equal to our threshold in our algorithm, which was 0.6. We purely stretched the hair down in model(e) to test the limitation of Eps distance we set up for the clusters.

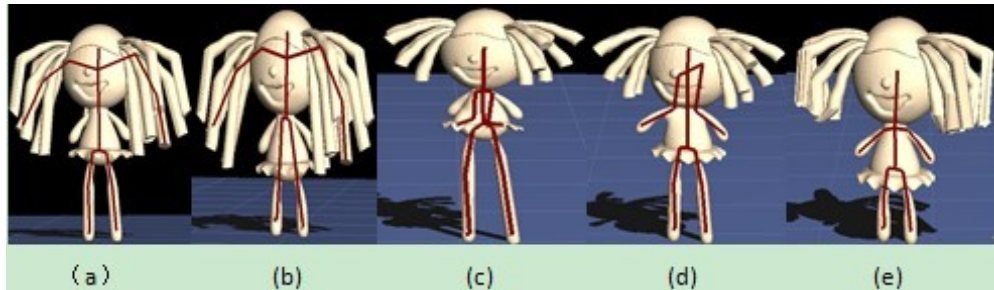


Figure 4.3: *Extend test for our algorithm.*

Models(A) and (B) failed because the hair often went below our threshold and all the clusters at the endpoint were not detected. This case represented the primary aspect of our limitation that once the unwanted clusters went far below the threshold, there is no way to eliminate them. We've thought about additional checking methods in length and width, but the fact that each model varied from shape to shape hindered this idea. A more involved idea was to automatically test the threshold. For example, we started at threshold=0.1 to see if the embedding succeeds. If not, we iterated this step by lifting the threshold to 0.2 and so on. This method indeed would eliminate the limitation of the height but we found it difficult to implement. The reason was because as we mentioned, Pinocchio used a bounded heuristic search to finish the embedment that could not be a clear sign of success or failure.

We've run into such situations that Pinocchio finished embedding in a few seconds with an incorrect result, but finished in several minutes with a correct one. We could not terminate during a embedding iteration with a single judgment of time and traversed vertices. Plus we found under normal cases, our threshold of 0.6 satisfied most of our test cases. Then we could deal with those cases as special ones.

Model(C) and(D) failed because they violated the assumption that the model should be proportioned roughly the same as the skeleton. Model(C) had extremely long legs that we deliberately made to let the hand vertices went above the threshold. As we expected, the hand vertices were eliminated as impossible vertices and thus we got the incorrect result. However, we got all four limbs embedded correctly in model(D) but a incorrect shoulder joint. The doll had a huge face with almost no neck, this became another limitation for our algorithm that some fabricated structures of body were not correctly recognized as expected.

Model(E)was a border test case on the limitation of the threshold. With such a proportioned character, we had its hair drooping down to its shoulder that roughly at 0.6 of its height and we correctly eliminated those hair vertices to embed the joints into the mesh.

Chapter 5

Performance and Discussion

Baran and Popvic³ specified the performance for Pinocchio in runtime efficiency, quality of results and generality. We follow their pattern to assess our revised API.

Figure 5.1 shows the statistics of all 16 test cases including the total number of vertices after the top down kd-tree split and that after the medial surface algorithm, the size of priority queue in heuristic search for the embedding, the possible vertices we get after clustering algorithm and runtime in total. Since our work only affects the possible vertices, the size of heuristic search and total runtime, we provides a comparison between the original API and our revised API.

Character #	Time (s)		Possible Vertices		Heuristic Search		Full Split	Medial Surface
1	13.95	13.88	73	34	153	158	19433	8539
2	14.40	13.75	59	36	149	149	10241	7971
3	14.32	13.91	67	38	151	151	16801	9932
4	17.40	16.82	58	34	149	163	21913	9963
5	30.51	29.80	342	162	215	179	44921	15788
6	15.92	16.28	81	49	160	160	23425	10162
7	19.26	18.29	86	43	3021	2990	19465	14105
8	15.83	15.21	84	52	155	155	18009	6406
9	20.44	20.22	112	64	457	425	28625	9793
10	32.30	28.39	266	175	1159	405	40729	14499
11	49.74	49.17	704	256	416	490	64937	30747
12	21.05	20.15	173	55	160	163	23097	5820
13	41.53	36.94	403	69	1185	1136	62793	24751
14	19.16	19.31	179	82	153	149	28769	6256
15	41.55	32.81	254	136	3010	3075	40721	12792
16	25.63	25.49	415	115	150	150	43801	19411

Figure 5.1: *Statistic on tested cases.*

The number of vertices ranges from 10,241 to 64,937 and the runtime ranges from 13.88s

to 49.17s. Compared to Pinocchio, the runtime decreases instead of increasing. This is because our algorithm reduces the size of candidate vertices which in turn reduces the size of heuristic search, plus in Pinocchio, embedding costs more than 15% of total time usage on average while an iteration over candidate vertices costs only about 1%. Comparing the size of candidate vertices to be tested for embedding, our clustering algorithm filters out 35% to 65% of the total vertices which greatly reduces the running runtime. Notice that the size of heuristic space search doesn't decrease much because the clusters that include the possible vertices are not filtered by our algorithm (-in fact there is no way to filter them out). However, as we did for Character 13, as long as we can eliminate those confusing vertices and reveal the structure of the mesh correctly before the system, the speed of embedding will raise apparently.

The quality of the embedding stays equal to the original API because our algorithm doesn't involve any technique in finding more detailed resolution of the mesh. Pinocchio uses LBS method(discussed in chapter 2) to specify the controlling part of a bone that works well for muscles but often shows its shorthand on presenting armors and and soft materials such as the dress on Character 7.

The generality slightly extends as discussed in chapter 4. Our new algorithm makes the API eligible for rigging characters with average length of hairs and dress. For those fabricated characters, as long as they proportioned similar to the biped skeleton, we can guarantee to produce a satisfied result.

We are far from satisfied with the revised system because we did not manage to expand the applicable range of the system greatly. Of course we realized that it would be intractable for the system to rig a extremely odd character that people set up to break the system. After all, there is no information provided with the models and we can barely recognize it using graphics knowledge. However, the purpose of inventing automatic rigging is to help emancipate model designers from iteratively rigging normal characters rather than work out a special case. Given this goal, the results are more satisfactory.

Chapter 6

Conclusions and Future Work

The problem addressed in this work stems from the observation of incorrectly embedded characters. The old API failed to detect and rule out those body parts that have similar shape to the limbs, plus a post checking section was needed to traverse the embedded skeleton and correct the wrongly placed joints. The objective of our work is to correct these issues while not to narrowing down the applicable scope of the original API.

We presented our revised Pinocchio API for automatic rigging in this paper and discussed limitations that we haven not yet solved. We introduced original API process and the problems occurred in their test results. We described DBSCAN, a well-known algorithm used in data mining, as the foundation of our algorithm. In our promoted API, two new modules were added to correct the hair, hands and feet issues appeared in Pinocchio. We used clustering algorithm to eliminate confusing vertices before the discrete embedding and adjusted the inappropriate hand and foot joint positions after the initial embedding. We fixed all the 3 wrongly rigged characters among all 16 test cases and did not affect the correctness on them. We also designed some extra test cases to verify the efficiency and the border of our methods.

We have some suggestions and a vision plan for future work. First, it would be a worth while improvement if we can develop a method for eliminating the threshold and is purely based on the clusters we gathered. This should especially work well for those models that have detailed resolution on ends of the limbs such as fingers and nails that the density of

vertices are heavier than the other places.

We have seen some shape recognition methods using databases that worked well. People depicts the shape using simple geometric primitives and the system will search out corresponding objects. This methods galvanizes us to design a similar interface combining with such a database that people will find a more appropriate skeletal structure to fit their mesh rather than our situation where a human skeleton for all biped characters.

Poirier and Paquette¹⁹ provides a multi resolution approach for extracting skeleton with highly detailed information and we can adapt it to retrieve more precise results. Gleicher¹⁰ provides a method in adjusting the segments of body mesh to adapt same skeleton to scaled characters that would save the repetitive processing time.

A more practical promotion to our current work maybe is to provide a user-friendly interface that can reflect the results on screen before finally outputting the animation files. Meanwhile, people can manually point out the correct joint positions to inform the system to redo the embedding.

In the end, automatic rigging technique originated due to a growing number of modeling tools and the corresponding need to reduce the burden of model designers on repetitive processing of manual riggings. It is worth while to investigate and research on this topic in order to provide some effective and convenience tools to relegate the problem.

Bibliography

- [1] A., K. S. T. (2003). Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transaction 2003*.
- [2] Apodaca, A. and Gritz, L. *AdvancedRenderMan - Creating CGI for Motion Pictures*.
- [3] Baran, I. and Popvic, J. (2007a). Automatic rigging and animation of 3d character. *In Proceedings of ACM SIGGRAPH 2007*.
- [4] Baran, I. and Popvic, J. (2007b). Penalty functions for automatic rigging and animation of 3d characters. <http://web.archive.org/web/20090411032339/http://people.csail.mit.edu/ibaran/penalty.pdf>.
- [5] C., M. W., C., W. F., and M., O. (2003). Skeleton extraction of 3d objects with radial basis functions. *In SMI 03: Proceedings of Shape Modeling International 2003*.
- [6] C., T., E., D. A., M., M., and P., S. H. (2004). Marker-free kinematic skeleton estimation from sequences of volume data. *In VRST -04: Proceedings of the ACM symposium on Virtual reality software an technology*.
- [7] E., D. A., C., T., M., M., and P., S. H. (2004). m3: Marker-free model reconstruction and motion tracking from 3d voxel data. *In Proceedings of Pacific Graphics 2004*.
- [8] Eberly, D. (2007). *3D Game Engine Design - A Practical Approach To Real-Time Computer Graphics, 2nd edition*. Morgan Kaufmann.
- [9] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density based algorithm for discovering clusters in large spatial database with noise. *In Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*.

- [10] Gleicher, M. (1998). Retargetting motion to new characters. *PREPRINT April 27, 1998 To appear at SIGGRAPH 98*.
- [11] Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.
- [12] Igarashi, T., MATSUOKA, S., and TANAKA, H. (1999). Teddy: A sketching interface for 3d freeform design. *In Proceedings of ACM SIGGRAPH 1999*.
- [13] J., F. and M., S. (2006). Bayesian estimation of the shape skeleton. *J. Vis. 6, 6*.
- [14] J., T., P., V. J., and M., D. (2006). 3d mesh skeleton extraction using topological and geometrical analyses. *In Proceedings of Pacific Conference 2006*.
- [15] M., L. J., P., K. J., and M., D. (2006). 3d mesh skeleton extraction using topological and geometrical analyses. *In Proceedings of the 2006 ACM symposium on Solid and physical modeling*.
- [16] M., M. and e G., P. (2002). Afne-invariant skeleton of 3d shapes. *In SMI 02: Proceedings of Shape Modeling International 2002*.
- [17] Marc-Guindon (2005). *Learning Maya 7 - The Modeling & Animation Handbook*. Alias.Systems Corp.
- [18] P., L., F., W., W., M., R., L., and M., O. (2003). Automatic animation skeleton construction using repulsive force field. *In Proceedings of Pacific Graphics 2003*.
- [19] Poirier, M. and Paquette, E. (2009). Retargeting control skeletons for 3d animation.
- [20] S., K., G., L., and A., T. (2005). Mesh segmentation using feature point and core extraction. *The Visual Computer 21, 8-10*.
- [21] Wade, L. (2000). Automated generation of control skeletons for use in animation. <http://accad.osu.edu/lwade/pubs/WadeDiss.pdf>.