An Automatic Balancing Scheme for Multi-Articulated Virtual Objects

Nakhoon Baek *
Kyungpook Nat'l Univ.
Daegu 702-701, Korea
oceancru@gmail.com

Kwan-Hee Yoo
Chungbuk Nat'l Univ.
Cheongju Chungbuk 361-763, Korea
khyoo@chungbuk.ac.kr

Abstract

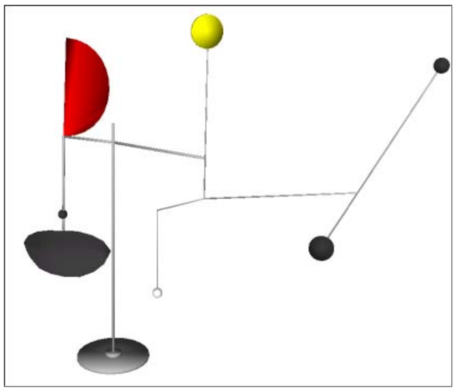
In many fields of computer science and other engineering areas, we often need to balance multi-articulated structures. In this paper, we formalize this kind of balancing problem from a more physical and theoretical point of view. Through describing details of all the solution steps, we finally represent a set of algorithms to automatically balance multi-articulated objects with tree topologies. Given the geometric configurations and masses at the leaf nodes of target multi-articulated objects, our algorithms achieve their balanced state through adjusting the mass of each node. To minimize the mass changes from the initial configuration, we use constraints of minimizing the norms of the mass differences between the initial masses and the final balanced masses. Actually, we use three different metrics, l_1, l_2 and l_∞ norms. These norms show slightly different behaviors in the minimization process, and users can select one of them according to their preferences and application purposes. We show all the details of algorithms, their time complexity analyses, and experimental results.
keywords: *balancing, tree-topology, minimization*

1 Introduction

In various fields including human computer interface, computer animation, mechanical engineering, and so on, we frequently use multi-articulated objects, whose components are linked to each other. Figure 1 shows virtual mobiles, as examples of such multi-articulated objects. Physically based techniques are then applied to generate their realistic motions. In this case, we need a set of physical parameters for each component, including mass, center of mass, moment of inertia, etc. There have been a few methods[4, 7, 8, 9, 10, 11] to automatically calculate these physical parameters from the given configurations. GPU-based implementations are also available[5].

Physically based techniques usually require the object to be initially in its balanced state. In fact, most real-world multi-articulated objects are in their balanced states. In our previous work[6], we have designed a virtual mobile for our physically based mobile simulation system, through configuring the shape of each component and assigning the mass and other physical properties for each component. However, the virtual mobiles

* This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (Grant 2011-0014886).


(a) *Steel Fish*

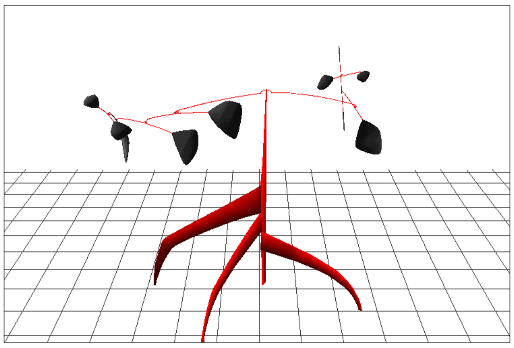

(b) *Southern Cross*

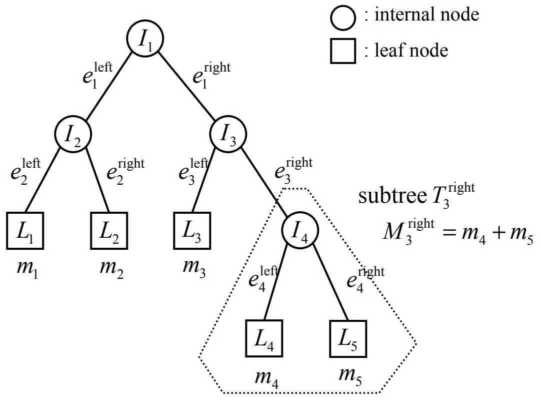
Figure 1. Virtual mobiles in their balanced states.

are difficult to maintain their balanced states, since the manually selected masses of the components usually result in rotational moments due to gravity. The fundamental problem is that our multi-articulated object, a mobile does not satisfy the balanced mass conditions at the links.

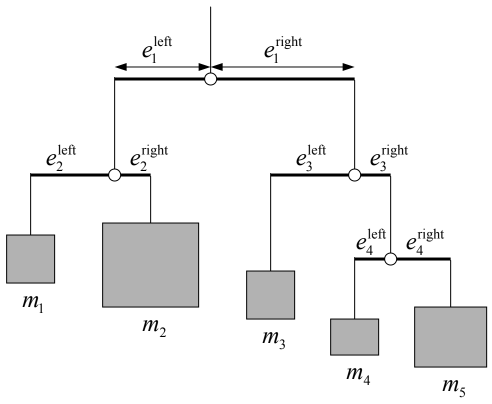
It had been impossible to find any previous results on systematically balancing multi-articulated objects. Thus, we used an iterative method to find the initial balanced states. To the best of our knowledge, there is still no research results on physically balancing multi-articulated objects. In this paper, we focus on the systematic way of physically balancing the multi-articulated objects, and present efficient and robust algorithms for finding the initial masses of multi-articulated objects with binary tree topology.

Since our algorithms focus on the balancing of tree-topology objects, other application areas may include the general-purpose load-balancing problems and network-topology optimization problems[2]. Although there have been some tree-balancing methods[3, 12, 2], they usually concentrated on the acceleration of inserting, deleting, and/or finding a node. Additionally, they usually achieved their goal through modifying the tree topology. Thus, previous tree-balancing methods are hard to apply for our physically balancing problem. In contrast, we focus on the change of leaf node masses, to finally get a balanced tree.

In section 2, we will describe the given problem more theoretically as the *weighted-leaf*



(a) a weighted-leaf binary tree



(b) its physical interpretation

Figure 2. A weighted-leaf binary tree and its physical interpretation.

binary tree balancing problem. Three kinds of minimization methods are also presented in this section. In the next section, we show the details of our balancing algorithms and their time complexity analyses. Experimental results on the practical virtual objects follow in section 4. Finally, we present our conclusions and future work.

2 Problem Definition

In this paper, we will present a systematic way of balancing a multi-articulated object with binary tree topology, to finally let the object be in its balanced state. As a start point, we will define a *weighted-leaf binary tree*, which is the theoretical model for our balancing problem.

A *weighted-leaf binary tree* is defined as a binary tree, in which each leaf node L_i has its corresponding positive mass m_i and each internal node has *zero mass.* Since it is a binary tree, an internal node I_j has its left sub-tree T_j^{left} and right sub-tree T_j^{right}. The total mass of T_j^{left} and T_j^{right} can be expressed as $M_j^{\text{left}} = \sum_{i \in T_j^{\text{left}}} m_i$ and $M_j^{\text{right}} = \sum_{i \in T_j^{\text{right}}} m_i$, respectively. Additionally, I_j has its left and right weighting factor, e_j^{left} and e_j^{right}, as shown in Figure 2(a).

We can physically interpret the weighted-leaf binary tree as a set of levers and masses. In this case, each internal node I_j corresponds to a lever with the arm length of e_j^{left} and e_j^{right} for each direction while each leaf node L_i to a mass of m_i, as shown in Figure 2(b). The physical laws show that the levers are in its balanced state if and only if $e_j^{\text{left}} \cdot M_j^{\text{left}} = e_j^{\text{right}} \cdot M_j^{\text{right}}$ for each internal node I_j.

However, it is hard to achieve the balanced state with arbitrary values of m_i's. In this paper, we present a systematic way of calculating balanced masses $\overline{m}_i$'s from the given values of m_i's. Figure 3 shows a conceptual diagram of our idea. In this way, we can achieve a balanced weighted-leaf binary tree without changing the tree topology.

The *weighted-leaf binary tree balancing problem* aims to find the balanced mass for each leaf node, with which $e_j^{\text{left}} \cdot \overline{M}_j^{\text{left}} = e_j^{\text{right}} \cdot \overline{M}_j^{\text{right}}$ for each internal node, where $\overline{M}_j^{\text{left}}$ and $\overline{M}_j^{\text{right}}$ are the total mass of left and right sub-trees respectively. Since we have $(n-1)$ internal nodes for n leaf nodes, we have $(n-1)$ equations for n unknowns. Thus, we need an additional constraint to solve this problem. In typical applications, the initial masses m_i's are given and we usually need to change the mass minimally to reserve the original configuration. Hence, we adopt the constraint of minimizing the difference of the initial masses m_i's and the balanced masses $\overline{m}_i$'s. To minimize the mass differences, we can use three different metrics: the l_1, l_2 and l_∞ norms of the mass differences. Actually, these norms have slightly different behavior in the minimization process, as shown in the followings.

Given the values of m_i's, the l_1-norm of the mass differences can be expressed as

$$||\overline{m}_i - m_i||_1 = \sum_{i=1}^{n} |\overline{m}_i - m_i|.$$

Thus, the minimization of this l_1-norm means minimizing the sum of all mass differences. Similarly, the l_∞-norm minimization means minimizing the maximum mass difference, since we can derive that

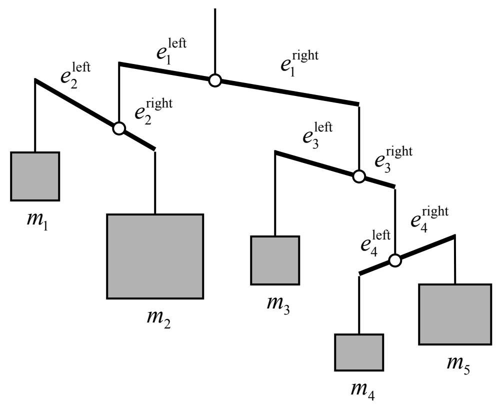
$$||\overline{m}_i - m_i||_\infty = \max_{i=1}^{n} |\overline{m}_i - m_i|.$$

In the case of the l_2-norm,

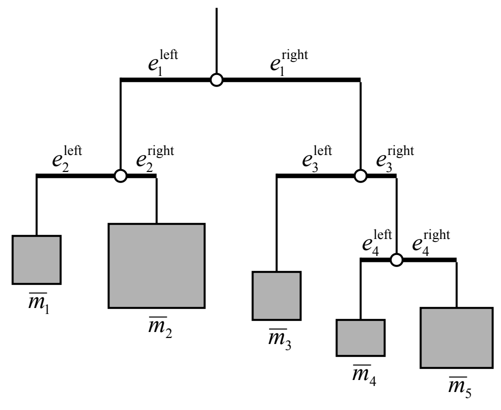
$$||\overline{m}_i - m_i||_2 = \sqrt{\sum_{i=1}^{n} (\overline{m}_i - m_i)^2},$$

where its minimization implies to avoid too much difference at a specific leaf node.

The l_1-norm minimization can show the smallest amount of total mass changes, while it also has the potential of too much change on a specific leaf node. We can minimize the maximum mass change through using the l_∞-norm. The l_2-norm minimization can give an intermediate solution. Hence, the user can select one of them according to his own preferences or the application purposes. The details of our solutions for the minimization of these norms are presented in the next section.



(a) initial configuration



(b) balanced configuration

Figure 3. Balancing a binary tree through changing the leaf node masses.

3 Balancing Algorithms

In this section, we will start from showing that the mass differences can be expressed as functions of the balanced mass of a specific leaf node, $\overline{m}_1$. Using this property, we present the weighted-leaf binary tree balancing algorithms for each of l_1, l_2 and l_∞ norms of the mass differences in the following sections.

3.1 Variable Substitution

For a weighted-leaf binary tree with n leaf nodes, we have $(n-1)$ equations for each internal node. These equations are actually linear combinations of n unknowns. We will first express the $(n-1)$ unknowns, $\overline{m}_2, \overline{m}_3, \cdots, \overline{m}_n$ in terms of $\overline{m}_1$, to finally represent the differences of m_i's and $\overline{m}_i$'s in terms of a single variable $\overline{m}_1$.

Suppose that a weighted-leaf binary tree is in its balanced state, with the leaf node masses $\overline{m}_i$'s. At a deepest internal node I_k, its left and right sub-trees are leaf nodes.

Therefore, $\overline{M}_k^{\text{left}}$ and $\overline{M}_k^{\text{right}}$ are equal to specific node masses $\overline{m}_p$ and $\overline{m}_q$, respectively. Since the tree is balanced, it is derived that $e_k^{\text{left}} \cdot \overline{M}_k^{\text{left}} = e_k^{\text{right}} \cdot \overline{M}_k^{\text{right}}$, or equivalently, $e_k^{\text{left}} \cdot \overline{m}_p = e_k^{\text{right}} \cdot \overline{m}_q$. Assuming that p is the smaller index, $\overline{m}_q$ can be calculated as $(e_k^{\text{left}}/e_k^{\text{right}})\overline{m}_p$. The total mass of the sub-tree whose root node is I_k can be expressed as $\overline{m}_p + \overline{m}_q = (1 + e_k^{\text{left}}/e_k^{\text{right}})\overline{m}_p$, with respect to the smaller index mass $\overline{m}_p$.

Using this approach in a bottom-up manner, we can build up the total mass of all the sub-trees in terms of the smallest index mass in the corresponding sub-trees. At the root node, the total mass of the whole tree is expressed in terms of $\overline{m}_1$. Now, we can propagate $\overline{m}_1$ from the root node to the leaf nodes. When the total mass of an internal node I_k is expressed in terms of $\overline{m}_1$, it implies that

$$\begin{aligned}\overline{M}_k^{\text{left}} + \overline{M}_k^{\text{right}} &= (1 + \frac{e_k^{\text{left}}}{e_k^{\text{right}}})\overline{M}_k^{\text{left}} \\ &= (\frac{e_k^{\text{right}}}{e_k^{\text{left}}} + 1)\overline{M}_k^{\text{right}}\end{aligned}$$

and both of $\overline{M}_k^{\text{left}}$ and $\overline{M}_k^{\text{right}}$ can be expressed in terms of $\overline{m}_1$. Applying this propagation in a top-down manner, we can finally express all the masses of leaf nodes in terms of $\overline{m}_1$ as $\overline{m}_i = c_i\overline{m}_1$, $1 \le i \le n$. Since we have assumed positive masses, c_i is the positive scaling factor for $\overline{m}_i$. The value of c_1 is trivially 1. Since the whole process only traverses the tree twice, it is easy to find that the total time complexity for these variable substitutions is $O(n)$.

3.2 Minimization of l_1-norm

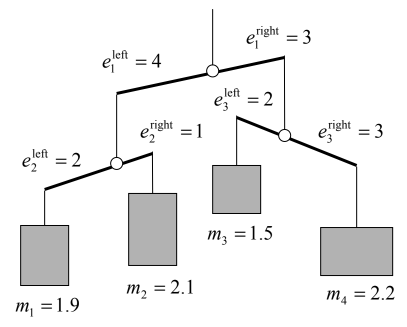
Using the variable substitution presented in the previous subsection, the l_1-norm of the mass differences can be calculated as

$$||\overline{m}_i - m_i||_1 = ||c_i\overline{m}_1 - m_i||_1 = \sum_{i=1}^{n} |c_i\overline{m}_1 - m_i|,$$

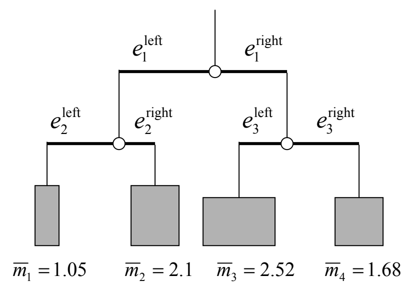
where m_i's are the initially given masses of the leaf nodes. Hence, the l_1-norm minimization becomes finding the minimum of the sum of folded line equations $|c_i\overline{m}_1 - m_i|$'s. As an example, the initial configuration of a weighted-leaf binary tree in Figure 4(a) gives the folded line graphs in Figure 4(c). The minimum of the folded line graph finally results in the balanced configuration of Figure 4(b).

The function of $\sum_{i=1}^{n} |c_i\overline{m}_1 - m_i|$ has discontinuity points at $c_i\overline{m}_1 - m_i = 0$, $1 \le i \le n$. Since these discontinuity points are only the candidates for the minimum of the l_1-norm, we can easily find the minimum through evaluating the l_1-norm at each candidate value of $\overline{m}_1$. With the value of $\overline{m}_1$ for the minimum l_1-norm, we can easily derive the values of balanced masses $\overline{m}_i$'s from the equality conditions $\overline{m}_i = c_i\overline{m}_1$. This process takes $O(n)$ time and thus it is optimal.

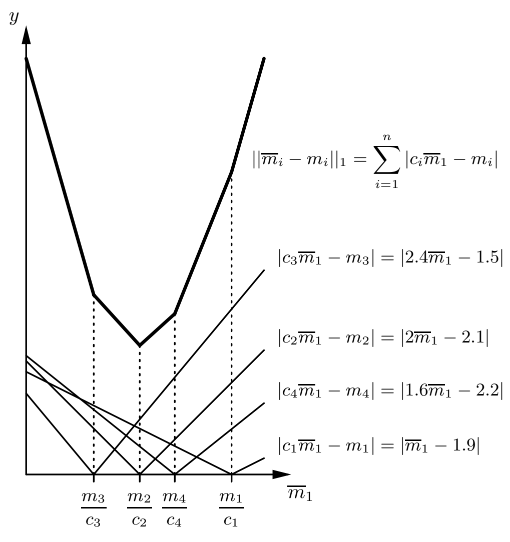
From more theoretical point of view, we additionally need to analyze the number of operations. In this case, the $O(n)$ candidate points are tested and each point requires $O(n)$ time additions to calculate $\sum_{i=1}^{n} |c_i\overline{m}_1 - m_i|$. Thus, the total number of operations is $O(n^2)$.



(a) initial configuration



(b) final configuration



(c) graphs for the l_1-norm minimization

Figure 4. An example of the l_1-norm minimization

Through sorting the candidate points and applying partial updates, the time complexity can be reduced to $O(n \log n)$. Letting the sorted candidate points be s_i, $1 \leq i \leq n$, we have $(n+1)$ intervals separated by these n candidate points. For the left-most interval $\overline{m}_1 \leq s_1$,

```
input: initial masses $m_i$'s and geometric configurations.
output: balanced masses $\overline{m}_i$'s.

apply variable substitution to get $c_i = \overline{m}_i/\overline{m}_1$, $1 \leq i \leq n$.
let $t_i$ be the candidate $\overline{m}_1$ values: $t_i = \overline{m}_i/c_i$.
{sorting in $O(n \log n)$ time}
sort $t_i$ to get the sorted candidates $s_i$'s.
{get the $s_i$ with the minimum value in $O(n)$ time}
calculate $min = \sum_{i=1}^{n} |c_i\overline{m}_1 - m_i|$ at $s_1$, using Eq. 1.
for $i = 2$ to $n$ do
  calculate $val = \sum_{i=1}^{n} |c_i\overline{m}_1 - m_i|$ at $s_i$, using Eq. 2.
  if $val < min$ then
    update $min = val$.
  end if
end for
{calculate the $\overline{m}_i$'s in $O(n)$ time}
let $\overline{m}_1$ be the $s_i$ value corresponding to the $min$ value.
for $i = 2$ to $n$ do
  $\overline{m}_i = c_i\overline{m}_1$.
end for
```

Figure 5. The l_1-norm minimization algorithm.

it is simply derived that

$$\begin{aligned}\sum_{i=1}^{n} |c_i\overline{m}_1 - m_i| &= \sum_{i=1}^{n} (-c_i\overline{m}_1 + m_i) \\ &= -\left(\sum_{i=1}^{n} c_i\right)\overline{m}_1 + \sum_{i=1}^{n} m_i \\ &= A_0\overline{m}_1 + B_0 \end{aligned} \tag{1}$$

Letting $s_1 = m_k/c_k$, the only one folded line equation $|c_k\overline{m}_1 - m_k|$ changes its sign for the next left-most interval $s_1 \leq \overline{m}_1 \leq s_2$. Thus, the l_1-norm for this interval can be incrementally calculated as:

$$\begin{aligned}&\sum_{i=1}^{n} |c_i\overline{m}_1 - m_i| \\ &= \sum_{i=1, i\neq k}^{n} (-c_i\overline{m}_1 + m_i) + (c_k\overline{m}_1 - m_k) \\ &= \sum_{i=1}^{n} (-c_i\overline{m}_1 + m_i) + 2\,(c_k\overline{m}_1 - m_k) \\ &= A_0\overline{m}_1 + B_0 + 2\,(c_k\overline{m}_1 - m_k) \\ &= (A_0 + 2c_k)\overline{m}_1 + (B_0 - 2m_k) \\ &= A_1\overline{m}_1 + B_1. \end{aligned} \tag{2}$$

In this way, the line equations for each interval can be calculated with only constant time operations. Through evaluating the line equation at the end points of each interval, we can get the $\overline{m}_1$ value for the minimum l_1-norm. This new approach can be performed in $O(n \log n)$ time, even counting the number of operations, and thus more suitable for the theoretical point of view. Overall processing can be summarized as Figure 5.

```
input: initial masses $m_i$'s and geometric configurations.
output: balanced masses $\overline{m}_i$'s.

apply variable substitution to get $c_i = \overline{m}_i / \overline{m}_1$, $1 \le i \le n$.
{calculate the $\overline{m}_i$'s in $O(n)$ time}
$\overline{m}_1 = \left(\sum_{i=1}^{n} c_i m_i\right) / \left(\sum_{i=1}^{n} c_i^2\right)$, using Eq. 3.
for $i = 2$ to $n$ do
    $\overline{m}_i = c_i \overline{m}_1$.
end for
```

Figure 6. The l_2-norm minimization algorithm.

3.3 Minimization of l_2-norm

The l_2-norm can be expressed as follows:

$$\begin{aligned} ||\overline{m}_i - m_i||_2 &= ||c_i \overline{m}_1 - m_i||_2 \\ &= \sqrt{\sum_{i=1}^{n} (c_i \overline{m}_1 - m_i)^2} \\ &= \sqrt{\left(\sum_{i=1}^{n} c_i^2\right) \overline{m}_1^2 - 2 \left(\sum_{i=1}^{n} c_i m_i\right) \overline{m}_1 + \sum_{i=1}^{n} m_i^2}. \end{aligned}$$

Since it is the square root of a quadric function of the single variable $\overline{m}_1$, the value of $\overline{m}_1$ which minimizes the l_2-norm is calculated as:

$$\overline{m}_1 = \left(\sum_{i=1}^{n} c_i m_i\right) / \left(\sum_{i=1}^{n} c_i^2\right). \tag{3}$$

After calculating $\overline{m}_1$, we can easily get the values of balanced masses, $\overline{m}_i = c_i \overline{m}_1$, $2 \le i \le n$. It is easy to find that this calculation can be performed in $O(n)$ time. The whole steps are shown in Figure 6.

3.4 Minimization of l_∞-norm

The l_∞-norm of the mass differences can be expressed in terms of a single variable $\overline{m}_1$ as follows:

$$||\overline{m}_i - m_i||_\infty = ||c_i \overline{m}_1 - m_i||_\infty = \max_{i=1}^{n} |c_i \overline{m}_1 - m_i|.$$

Therefore, the minimization problem becomes a kind of min-max problem. In other words, we need to find the minimum of the maximum of the folded line equations $|c_i \overline{m}_1 - m_i|$'s.

Plotting the folded lines $|c_i \overline{m}_1 - m_i|$'s on the xy-plane, the maximum of these equations corresponds to the upper-most boundary of the line equations, as shown in Figure 7. The l_∞-norm has its minimum value at the lowest y-value point along the upper-most boundary.

We use a line segment arrangement algorithm[1] to solve this min-max problem. After plotting the folded line equations, we find the upper-most boundary line segments. Finally, we find the lowest y-value point located on this boundary. This processing requires $O(n \log n + k)$ time, where k is the number of intersection points in the final line segment

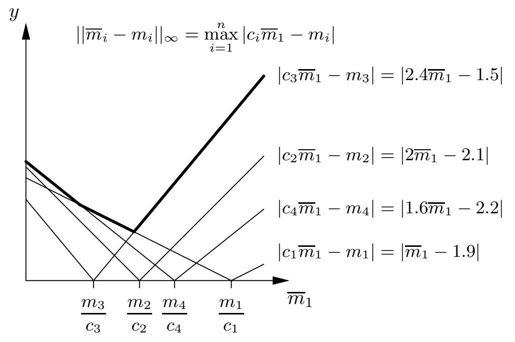


Figure 7. An example of the l_∞-norm minimization.

```
input: initial masses $m_i$'s and geometric configurations.
output: balanced masses $\overline{m}_i$'s.

apply variable substitution to get $c_i = \overline{m}_i/\overline{m}_1$, $1 \le i \le n$.
for $i = 1$ to $n$ do
    draw folded line segments $|c_i\overline{m}_1 - m_i|$'s.
end for
{line segment arrangement in $O(n \log n + k)$ time}
apply line segment arrangement algorithm to get the upper-most line segments.
get the minimum-$y$ position along the upper-most line segments.
{calculate the $\overline{m}_i$'s in $O(n)$ time}
let $\overline{m}_1$ be the $x$ coordinate of the minimum-$y$ position.
for $i = 2$ to $n$ do
    $\overline{m}_i = c_i\overline{m}_1$.
end for
```

Figure 8. The l_∞-norm minimization algorithm.

arrangement. When the lowest y-value point is found, the value of $\overline{m}_1$ corresponds to its x-coordinate value. All the values of balanced masses can be calculated in a linear time. Conclusively, we can solve the l_∞-norm minimization in $O(n \log n + k)$ time, based on the line segment arrangement algorithm, as shown in Figure 8.

4 Experimental Results

We used our multi-articulated objects balancing methods to the virtual mobile system[6]. Due to automatic tree balancing features, we can avoid the iterative adjustment of the component masses. Examples of the balanced mobiles are shown in Figure 1. As we have expected, the mobiles are naturally in its balanced state. We used the l_1-norm minimization for these examples. Due to its optimized behavior, their execution times were less than 1 msec.

5 Conclusion

In this paper, we formalized the weighted-leaf tree balancing problem, which is directly applicable to the balancing of multi-articulated objects. We showed that the weighted-leaf binary tree balancing problem can be transformed into a minimization problem of a single variable. The solutions for the l_1, l_2 and l_∞-norm minimizations are presented. These three different metrics can be selectively used for the user's preference of the mass adjusting behavior.

Theoretically analyzing their details, we show that the l_1, l_2 and l_∞-norm minimization can be achieved in $O(n \log n)$, $O(n)$ and $O(n \log n + k)$ operations, respectively, where n is the number of components in the target multi-articulated object and k is the number of line segment intersections during its processing. Though we have focused on the balancing problem and its solutions, we can extend it for more general applications and more general topologies. From the theoretical point of view, it may be a challenging problem to calculate the upper bound of this kind of minimization problems.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (Grant 2011-0014886).

References

[1] I Balaban. An optimal algorithm for finding segments intersections. In *SCG '95: Proc. the 11th Annual Symp. on Computational Geometry*, pages 211–219, 1995.

[2] R Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Inf.*, 1:290–306, 1972.

[3] R Bayer and E McCreight. *Organization and maintenance of large ordered indexes*, pages 245–262. Springer-Verlag New York, Inc., 2002.

[4] C Gonzalez-Ochoa, S McCammon, and J Peters. Computing moments of objects enclosed by piecewise polynomial surfaces. *ACM Trans. Graph.*, 17(3):143–157, 1998.

[5] J Kim, S Kim, H Ko, and D Terzopoulos. Fast GPU computation of the mass properties of a general shape and its application to buoyancy simulation. *Vis. Comput.*, 22(9):856–864, 2006.

[6] D Lee et al. Reproducing works of calder. *J. of Visualization and Computer Animation*, 12(2):81–91, 2001.

[7] Y T Lee and A Requicha. Algorithms for computing the volume and other integral properties of solids. I. known methods and open issues. *Commun. ACM*, 25(9):635–641, 1982.

[8] Y T Lee and A Requicha. Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation. *Commun. ACM*, 25(9):642–650, 1982.

[9] S L Lien and J T Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications*, 4:35–41, 1984.

[10] B Mirtich. Fast and accurate computation of polyhedral mass properties. *J. Graph. Tools*, 1(2):31–50, 1996.

[11] C Narayanaswami and W Franklin. Determination of mass properties of polygonal CSG objects in parallel. In *SMA '91: Proc. the First ACM symp. on Solid Modeling Foundations and CAD/CAM Applications*, pages 279–288, 1991.

[12] D Sleator and R Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.