A Dynamic Programming Approach to Search Similar Portions of 3D Models

MOTOFUMI T. SUZUKI
National Institute of Multimedia Education
2-12 Wakaba, Mihama-ku, Chiba-shi 261-0014, JAPAN

Abstract: The use of 3D models is gaining wide popularity since they are very important for graphics in various software applications, virtual reality and web3D. Recently, similarity search techniques for 3D models have been investigated intensively to retrieve 3D models from the Internet. Most similarity search techniques are suited for comparing each individual 3D model from databases. However, our similarity search techniques can compare not only each individual 3D model, but also similar portions of 3D models. Using our technique, each 3D model is divided into a huge number of pieces, and shape descriptors are extracted from these pieces to compare similarities. Although there are a large number of combinations for comparing the similarities for portions of 3D models, we have applied a dynamic programming approach to reduce the exponential computing time associated with evaluating similarities. Our technique makes fast similarity searches for portions of 3D models possible.

Key-Words: Similarity Search, 3D Model, Dynamic Programming, Partial Search

1 Introduction
Various 3D models are available on the Internet, and the number of the available models is increasing rapidly. Various search techniques have been investigated to retrieve 3D models from databases [1-8]. These search techniques extract shape descriptors from each 3D model in databases and use these descriptors as database indices. The indexing of databases is done automatically by using software programs without time consuming human indexing operations. Various shape descriptors [1-8] have been proposed recently, and the search results rely on the ability of these shape descriptors to adequately describe the 3D models of the shape features. Therefore, it is very important that the system uses proper shape descriptors to search for 3D model form databases. Currently, most of these search techniques and shape descriptors are suited for comparing each individual 3D model from databases, and they can not handle searches to find similar portions of 3D models. Search techniques to handle such partial searches [15,16] have not been investigated very extensively. Using such a search technique, systems can compare not only each individual 3D model but also each portion of a 3D model by dividing the 3D model into pieces. This technique extracts shape descriptors of the divided pieces of 3D models as indices to find partially similar portions. Although the generated pieces cause the system to compare exponentially a large number of shape descriptors, our system avoids this problem by using a dynamic programming approach for a more efficient computation.

2 Searching Similar Portions of 3D Models
In this section, (1) an overview of the similarity search, (2) an examination of the shape descriptor extraction, and (3) a string sequence comparison based on a dynamic programming approach are described.

2.1 An Overview of the Similarity Search
The basic steps of the similarity search are shown in Figure 1. These steps can be categorized into two phases. The first phase "Shape descriptor extraction" includes (1) Triangulation, (2) Face division, (3) Removal of redundant faces and (4) Shape descriptor extraction. The second phase "String sequence comparison" includes (5) Conversion of shape descriptors into string sequences and (6) Sequence alignment by a dynamic programming approach. Details of these two phases are explained in the following sub-sections.
2.2 An Examination of the Shape Descriptor Extraction

3D models are divided into numerous polygonal pieces for finding shape features. 3D models are triangulated [9,10,11] so that polygonal pieces form simple triangles. One of the properties of each triangle involves a connection to three neighboring triangles if the 3D models do not cross or collide with each other. By connecting these triangles based on neighboring relations, sets of triangles can be created. There are a number of combinations to create such sets of triangles. These combinations can be represented as a binary tree (except the first node which is ternary) data structure as shown in Figure 2. To express the triangle relations, the binary trees are created for each triangle in a 3D model. For example, 100 binary trees are needed if the 3D model is made from 100 triangle faces. The created binary trees contain redundant branch nodes such as (1) a repeating branch and (2) a reverse branch. The repeating branch is created by double counting the same triangles, and the reverse branch is counted by the same nodes in other binary trees. The number of tree nodes can be decreased by at least half by removing the reverse branches.

Once the 3D model pieces are represented as tree structures and unnecessary nodes are eliminated, shape descriptors are computed as shown in Figure 3 using the following procedures:

1. Three normal vectors on the vertices of each triangulated face have to be computed if these normal vectors are not available in the 3D model data files.
2. The three normal vectors are averaged as the normal vectors of the triangle faces.
3. All the normal vectors of the triangle faces containing each face set created are averaged as the normal vectors of face set Ng.
4. Angles created between the normal vector Ng and each normal vector N0, N1, and N2 are calculated.
5. Calculated angles are sorted in ascending order.

If only the angles are used as shape features, all the face sets which contain similar angle relations are matched regardless of the area size of the face sets. This may be a useful result for many applications. However, in our experiment, we assume that large faces have more influence or power to determine the shape features. For instance, angles created by larger faces are weighted more than those of smaller faces. The area of the triangles S0, S1 and S2 are computed by Heron's formula using the length of 3 edges of the
triangle and the semiperimeter. Once all the areas of the triangles are computed, the area sizes of the face sets may be found by adding triangle area sizes which are contained in the face sets. Figure 4 shows how the computed angles of the shape descriptors are distributed to the shape descriptor tables based on the area size of the faces. The tables are computed so that larger triangles are treated as more influential in terms of their shape descriptor power. Since it is necessary to compare the face sets which contain a different number of faces, normalization of the shape descriptor tables is required. The number of slots in the shape descriptor tables can be increased or decreased by modifying the location of the slot separators in the descriptor tables. All the modified shape descriptor tables must have the same quantity of slots to compute the similarity distances for every search query.

Since the numerical values of shape features are reflected in the string sequences, the system can find similar portions of 3D models by searching similar string sequences against the query key. The length of the string sequences is eventually quite long because of a large number of index slots. The huge numbers of string sequences are generated from a 3D model which contains a large number of triangle faces. It is quite difficult to compare the similarity of string sequences in polynomial time by using a brute force approach which tries to match and evaluates similarities for all the combinations of string sequences. A huge amount of computing time is especially needed if the two sequences of strings have different lengths. The computing costs to compare two sequences of strings are known as $O(m+nCm)$ where $m$ and $n$ are the length of the two strings. The brute force approach is not very practical for computing using standard personal computers. Therefore, we apply a dynamic programming approach for string sequence comparisons in which the computing costs are $O(mn)$. It is not a very fast approach for large values of $m$ and $n$. However, it is reasonable approach and it could be efficient if multiple processors are used. Details of the dynamic programming approach are discussed in the next sub-section.

### 2.3 A String Sequence Comparison Based on a Dynamic Programming Approach

Dynamic programming is often applied in the construction of algorithms to solve certain optimization problems. It is a bottom-up technique in which the smallest sub-instances are explicitly solved first, and the results of these are used to construct solutions for progressively larger sub-instances. The dynamic programming approach is used to find the optimal alignment between sequences of two strings. There are three basic steps for sequence alignment which include (1) matrix initialization, (2) matrix scoring, and (3) trace back. These steps can be demonstrated by aligning the two string sequences. For example, suppose there are two string sequences "baaddecbda" and "bbadceba" which are created from shape descriptor tables, and we want to align these two sequences. For the first step, a matrix is created as shown in Figure 5. The first row and column of the matrix is filled with 0 for initial values.
Secondly, each position in the matrix is filled with a score based on the scoring equation 1. \( M \) represents the score for the matrix position. \( G \) represents a gap penalty score, and a value of ",-2" is assigned. \( S \) represents the match/mismatch score, and value "2" is assigned to \( S \) for the match and value "-1" is assigned for the mismatch.

\[
f_{i,j} = \begin{cases} 
M_{i-1,j-1} + S_{i,j} \\
M_{i,j-1} + G \\
M_{i-1,j} + G
\end{cases} \quad \text{Equation (1)}
\]

The score for the matrix positions, the left position \( M(i-1,j) \), the above position \( M(i,j-1) \) and the diagonal position \( M(i-1, j-1) \) are needed to calculate the score for the matrix position \( M(i,j) \).

Scoring starts from the matrix position \( M(1,1) \) which is near the upper left corner of the matrix. The score for the matrix position \( M(1,1) \) is value 2, and the score is derived from the following calculation steps:

\[
\begin{align*}
\text{Max}( M(0,0) + S(1,1), \ M(1,0) - G, \ M(0,1) - G ) &= \text{Max}( 0 + 2, \ 0 - 2, \ 0 - 2 ) \\
&= \text{Max}( 2, -2, -2 ) \\
&= 2
\end{align*}
\]

In a similar manner, the scoring process is repeated until all the matrix positions are filled with scores. Figure 6 shows the matrix filled with scores.

After the scoring of the matrix, a traceback process is carried out. Figure 7 shows the path of the matrix traceback. The traceback process begins from the lower right corner of the matrix in Figure 7. The traceback path can be found by checking how the scores for the matrix positions are calculated. Complete traceback steps are shown in Figure 8. There are three possible predecessors for traceback, which include (1) left, (2) above and (3) diagonal. Occasionally, there are several possible predecessors available for traceback. In that case multiple sequence alignment solutions are produced and the system has to choose one predecessor for traceback. For example, as shown in Figure 8, there are two ways for traceback at the row 5, column 6 position. The traceback can be either in the diagonal position or the left position.
Based on the traceback route, two string sequences are aligned as follows:

\[
\begin{array}{cccccccc}
   b & a & a & d & d & c & a & b \\
   b & b & a & * & d & c & * & b & * & * & a
\end{array}
\]

The "*" represents an empty string. The alignment is stretched over the entire sequence lengths to include as many matching strings as possible.

The dynamic programming approach can find the optimal alignment between sequences of two strings with different lengths. Since the algorithm can handle alignment of the different lengths of strings, the system does not have to compute all the possible combinations of various lengths of strings by a brute force approach. Therefore, the algorithm helps the system avoid computing an exponentially large number of comparisons. When sequences of strings are aligned, sequence alignment scores are computed. String sequences are matched well for higher alignment scores. The system can find similar sequences against a query key sequence by sorting the alignment score in ascending order. Since each string sequence corresponds to the shape feature of triangle faces, the system can display similar sets of triangle faces as search results.

### 3 Experiments and Results

The system was implemented based on the algorithms described in the previous section. We have tested the search efficiency by using the implemented system.

#### 3.1 Search System

The system is implemented by a C++ language and the Open Inventor graphics libraries[13] running on a Linux operating system. The system can be run on standard personal computers. For the search experiment, we have used a PC which consists of a Pentium 4 2.66 MHz CPU with 1024 Mbytes of memory. Figure 9 shows a user interface of the system. Each window can display a 3D model with similar portions by using a different rendering color.

#### 3.2 Search Efficiencies

The number of comparisons depends on how many triangles are contained in a 3D model and how triangles are connected to each other. Three types of search methods are compared in terms of the number of comparisons. The search methods are (a) a search based on a brute force approach, (b) a search using binary trees, and (c) a search based on a dynamic programming approach. The average of these three search methods are compared among five 3D models, and sizes of 5, 10, 20, and 30 triangle face sets are used for the query key portion. Figure 10 shows the time (seconds) needed to search similar portions of 3D models for these methods. Search method (a) consists of checking all the combinations of triangle sets, and it
is almost impossible to compare the large number of triangle faces in polynomial time. Although, search method (b) can reduce the number of comparisons compared with search method (a), a large number of comparisons are still needed, and it does not work efficiently for the query keys which contain a large number of triangles. Search methods (a) and (b) are logically computable, but they can not find solutions in polynomial time for the large number of triangles. Search method (c) can dramatically reduce the number of comparisons. However, because of the characteristics of the dynamic programming approach, the search results are possibly not the best but are as optimal as possible when the system needs to find a solution in polynomial time.

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brute force</td>
<td>2.0</td>
<td>56.0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Trees</td>
<td>1.2</td>
<td>22.1</td>
<td>3838.1</td>
<td>75398.0</td>
</tr>
<tr>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>1.1</td>
<td>20.1</td>
<td>433.0</td>
<td>3646.1</td>
</tr>
</tbody>
</table>

Figure 10: Search efficiency (a) Brute force, (b) Binary tree, and (c) Dynamic programming

### 3.3 Examples of Search Results
Figures 11, 12, 13 and 14 show example search results. The similar portion of the query key is in the upper left corner in each figure and is rendered in a different color. Some portions of the 3D model are used for the search keys, and area sizes of search keys are 5% of the entire surface areas of each 3D model. As shown in Figures 11, 12, 13 and 14, similar portions are searched from each 3D model.

#### Figure 11: Search results for querying portion of a chess piece

#### Figure 12: Search results for querying portion of a chess piece

#### Figure 13: Search results for querying portion of a stomach

#### Figure 14: Search results for querying portion of an umbrella

### 4 Conclusions and Future Work
In this paper, we proposed a search method to find similar portions of 3D models by using a dynamic programming approach. The use of a dynamic programming approach greatly accelerates the similarity search, and makes it possible to handle an extremely large amount of similarity matching. We have implemented the 3D model search system based on an algorithm and have tested the search system. The experiments showed that the system response was sufficiently quick. Currently, the system can search for partially similar portions against a single 3D model at the same time. The system will be improved to detect similar portions from multiple 3D models simultaneously.
Acknowledgements:
This research was supported by research grants from the Okawa Foundation for Information and Telecommunication (#2002-02-29) and a grant-in-aid from the Ministry of Education, Science, Sports and Culture, Japan (#15700115)

References: