REAL TIME GENERATION OF PROGRESSIVE MESHES FOR CHANGING ENVIRONMENTS

Enhua Wu
University of Macao, P.R. China
Guangzheng Fei
Chinese Academy of Sciences, .R. China

ABSTRACT

The progressive mesh representation is in general introduced for a continuous level of detail, and with it smooth transition between meshes can be achieved. However, the method is impractical for realtime simplification. Despite that, the realtime generation of progressive meshes is often required for applications such as in a robot simulation system or a flight simulation system where models are constantly modified by the actions of object collision, object destruction and reconstruction, etc. A new method is presented in this paper to tackle realtime simplification through progressive meshes. By the algorithm only the current mesh and readily accessible local information of each vertex, such as vertex position, edge length, neighboring vertices, adjacent faces, and face normals, are considered at each simplification step. All edge-collapse costs are calculated and sorted into a binary tree using the heap sort algorithm at the initial stage. At the iterative stages only the neighboring affected costs need be recalculated and rearranged in the binary tree. Other properties, such as vertex color and face texture, are processed in the same way as the geometry. The algorithm greatly improves the time performance under the constraint that the quality of the generated meshes be acceptable. Tests show that the algorithm is viable in a realtime simplification for medium-scale virtual models on PC platforms.

1. Introduction

Despite the rapid development in hardware technology, it is still very difficult to render scenes with a large number of detailed objects at an interactive frame rate. Level of detail (LoD) is one of the most important fields of study in reducing the geometric and rendering complexities of the scene. Research on simplification of three-dimensional polygonal objects has spanned the entire range from local [1,2,4,5,6] to global [7,8,9] algorithms, with several approaches in between [3,10].

Simplification algorithms such as those mentioned above are iteratively applied to obtain a hierarchy of successively coarser approximations to the input object. Such multiresolution hierarchies have been used in LoD-based rendering schemes to achieve higher frame update rates while maintaining good visual realism. These hierarchies have a number of distinct levels of detail: usually 5 to 10 for a given object. As a direct consequence of a limited number of levels of detail, we find that very often a noticeable flicker occurs when switching between different levels of detail (the so-called "popping effect"). A continuous level-of-detail representation is one of the best methods to reduce the popping effect, which produces a large number of levels of detail for the selection. Recently a new method known as *Progressive Meshes* [10, 15] introduces a polygon reduction technique, whereby an object is represented with the coarsest level of detail and an incremental detail record.

Construction of a LoD representation for a virtual environment is a time-consuming process. Therefore in most applications so far, the LoD generation is performed through a pre-computing process. However, in some applications, such as a robot simulation system or a flight simulation system, the objects are often destroyed or reconstructed. The changing environment of that kind poses a challenging problem for real-time LoD generation. As a matter of fact, the study in this paper originates from the requirement of a distributed interactive simulation (DIS) project for flight simulation across a few universities and research institutes.

In the remainder of this paper we first review the research by Hoppe [10] and state the problem we are going to solve in Section 2. This is followed by a description of the most critical measures of an object in Section 3. Our new method is described in Sections 4, 5 and 6. Using the method, the emphasis is placed on efficiency, where we begin with the basic algorithm, and proceed through a progression of simple optimizations. Then we extend our method to allow multiple properties such as vertex color and face normal. Finally, the paper concludes with a discussion of the results, ideas of future work, and a summary.

2. Previous Work

Progressive meshes offer an elegant solution for a continuous resolution representation of polygonal meshes. A polygonal mesh M = Mn is simplified into successively coarser meshes Mi by applying a sequence of edge collapses. An edge-collapse transformation and its dual, the vertex-split transformation, are shown in Figure 1.

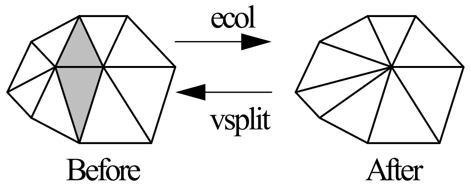

Figure 1. Edge collapse and vertex split

As we can see, the selected vertex is moved to a neighboring vertex. The triangles that contain the selected vertex now contain the vertex that was collapsed. In his previous work [3], Hoppe shows that the edge collapse operation alone generates good meshes for simplification. The vertex-split operation is the opposite of edge collapse. Storing position and adjacencies of a vertex that is collapsed into another makes it possible to undo the operation. To extend this idea, a sequence of vertices results in that the initial mesh can be reconstructed after collapsing all the edges.

The challenging part of Progressive Mesh polygon reduction is to find a good sequence of edges to collapse. Without a good means of selection it is easy to destroy important features, to generate folds, and to create other artifacts. When selecting the next vertex to remove, Hoppe considers the distance to the original mesh, resulting edge lengths, surface attributes, and feature (creases, ridges, and corners) information.

Hoppe measures the distance between a mesh and the initial mesh by using a number of points on the original mesh including the vertices as well as points scattered across the faces. The distance of a mesh is the mean square distance between the points scattered over the initial mesh and their corresponding closest points on the resulting mesh. When selecting an edge to collapse, minimizing this distance is one factor.

3. Importance Measures

We know that the key to good simplification lies in the choice of a good point-importance measure. By Hoppe's approach, after a weighted sum of the distance to the original mesh is calculated, and the resulting edge lengths, the change in surface attributes, and the consequence of feature (creases, ridges, and corners) information have been determined, the edge with the least cost is then chosen for the next collapse. After the collapse, information about vertices and edges in the affected neighborhood is recalculated and the process continued by selecting the next least cost edge collapse.

As a result of global optimization, Hoppe's approach produces a good simplification result for large and complex objects. However, Hoppe suggested that that those techniques are only intended for offline calculation. The run cost presented in his paper ranges from 15 minutes to a few hours. However, real-time simplification is very often required, considering the applications where objects need to be modified. For example, consider a flight simulation system or a robot simulation system where a collision between objects causes destruction of the environment, especially to parts of complex objects, so the objects need be modified and the pre-computed level-of-detail representations are no longer valid. Modifiable environments will not be practical unless there is a way to quickly reduce the polygons. By Hoppe's approach we can easily find that the most expensive operation is the selection of next edge to collapse at each iteration step.

Ultimately, the final selection of importance measures must depend upon the quality of the results it produces. Thus a good measure should be simple and fast. It should produce good results on arbitrary objects, and it should use only local information. Since we are simplifying detailed objects, the importance measure will be evaluated many times. Consequently, any cost inherent in the importance measure will be magnified many times due to its repetition. At the same time, the importance measure should use only local information in order to support some significant optimizations in the algorithm's running time.

The first measure we explored is edge length. Not all edges are equally important in a model. Obviously, larger faces (longer edges) are more visible to the user. In other words, lower visual error is produced to collapse a shorter edge than a longer one, and taking edge length as one importance measure is reasonable.

The next measure we explored was curvature. In everyday life, we experience objects with high curvature, such as peaks, pits, ridges, and valleys, are visually significant. The number of polygons needed to approximate the surface depends on the curvature. More polygons are needed for rough area than the smooth one. These observations suggest that curvature should be chosen as another measure of importance.

4. Fast Polygon Reduction Method

Recall from the previous section that at each step Hoppe looked at many factors including the comparing the next possible mesh with the initial model, the distance of many generated points over the faces, etc. The research here explores various approaches to quickly selecting edges for collapsing in the hope of finding a technique that does a reasonable job of polygon reduction. More precisely, next-edge selection algorithms are presented, which only consider the current mesh (instead of the initial mesh) and use only readily accessible information in the neighborhood of a vertex, including edge length, neighboring vertices, adjacent faces, and face normals. Taking edge length as the only importance measure, the technique of selecting the shortest edge for the next collapse was tested. This simple strategy actually worked surprisingly well, as you can see from the following result shown in Figure 2. To achieve better time performance, we actually take the following formula to calculate the distance of two vertices instead of the Euclidean distance formula:

$$dist(v_i, v_j) = \sum_{k=0,1,2,\ldots} \left| v_{ik} - v_{jk} \right|$$

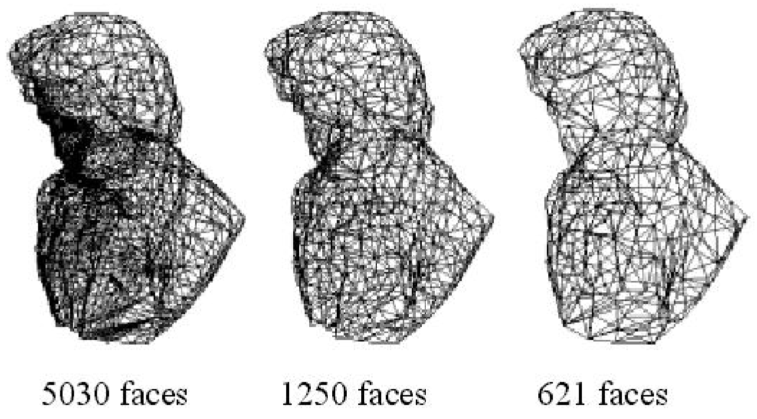

Figure 2. Using edge length as an importance measure

As was mentioned in the previous section, the next importance measure we explored is curvature. We selected the next edge collapse for removal based on how coplanar the neighboring faces are. To save more computing time, we calculated the costs in the simplest way. In particular for an edge collapse, say vertex A to vertex B, we considered the normal product of the two faces that have edge AB. The highest dot product is used for comparison when selecting the next edge for collapse. Suppose that all the faces in our models are triangles, and the two triangles that have the common edges are ABC and ABD. Then the following formula is used in our algorithm to calculate the curvature, where N refers to face normal:

$$Curvature(AB) = 1 - N_{ABC} \cdot N_{ABD}$$

Figure 3 shows a selection of the generated progressive meshes using curvature as the importance measure.

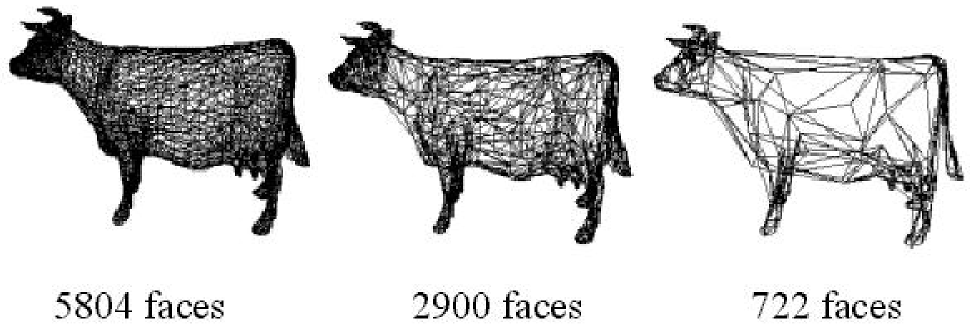

Figure 3: Using curvature as an importance measure

We took the two importance measures to guide the polygon reducing process, and found that each worked well in certain situations but not so well in others. It would thus seem to be a good idea to combine the two techniques. The simplest way to combine them is to take the weighted sum of the two measures. This turns out to be highly successful. An example of the result using this approach is shown in Figure 4 and Figure 5.

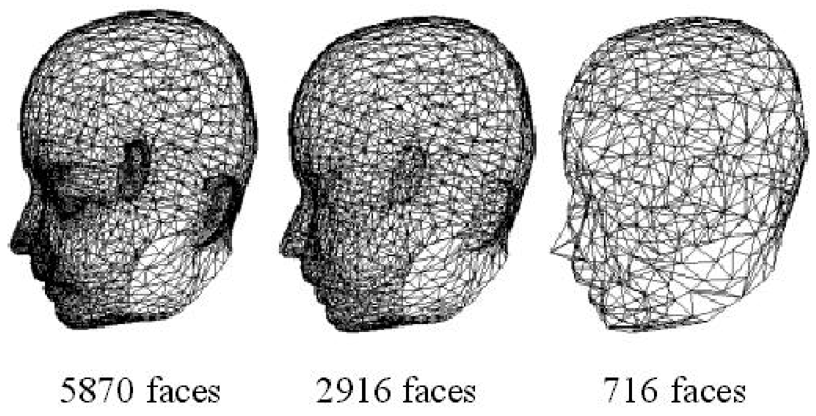

Figure 4. A hybrid of the two importance measures

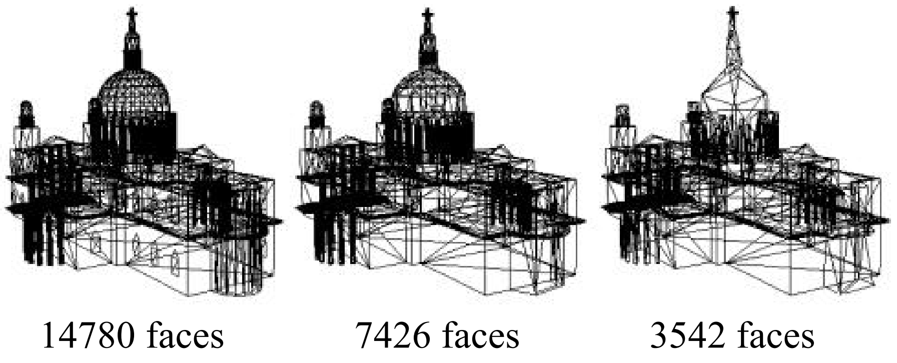

Figure 5. Progressive meshes of a model of a church

5. Acceleration Technique

Suppose the initial object to be simplified with n vertices will be simplified into an ultimately coarser model with m vertices (m <<n). From the investigation of our algorithm, we see that a vertex is reduced at each step from the mesh. Thus the total iteration runs (n-m) times. At each iterative step, the main operation include:

- Computing all the collapse costs of the affected vertices.
- Finding the vertex with the lowest collapse cost.

As for the first operation, we can greatly reduce the complexity in two ways. First, we can simplify the computation of the collapse cost as much as possible. Only two importance measures are considered in the algorithm, which is further simplified by using some inexpensive arithmetic operation. Secondly, since all the affected vertices are assumed to be in the neighborhood, and the corresponding data structure has been designed to support direct accessibility, the time performance is further optimized.

In the case of the second operation, if we try to find the vertex with the lowest collapse cost in an unsorted dataset, the total number of vertices to be traversed before reaching the desired is proportional to the size of the dataset, i.e., the time complexity is O(n). If we first sort the dataset, i.e., let the first vertex have the lowest collapse cost, then finding the operation costs only O(1). At the iterative stage, once the vertex with the lowest cost has been selected and removed, we should recalculate and rearrange the dataset. As is well known, the heap sort is not only one of the fastest sort algorithms, but it is also suitable for maintaining modifiable data, i.e. once the top vertex of the heap is removed, it is very easy to rearrange the heap. (In fact, the rearrangement of the heap is one of the basic operations of the heap sort algorithm, which requires no more than lg(n) moving operations). Consequently, we choose the heap sort to accelerate the second operation. Through the algorithm, we recognize that the second operation is, in fact, a dynamic heap sort process, where the data are changing all the time. The total complexity of the algorithm is O(n*lg(n)). Since the heap sort algorithm has the property of both speed and modifiability, the choice of the heap sort algorithm is the key point of reducing the computation complexity.

6. Data Structure and Codes

The following data structures are used in our algorithm:

1. Vertex list, a queue consisting of all the vertices. Information on each vertex includes

```
struct Vertex {
Vector position;                // vertex position
int  id;          // vertex index
Vertex *          neighbor;     // adjacent vertices
Triangle *face;                 // neighboring faces
    float         cost;         // vertex collapse cost
    Vertex *      collapse;             // vertex to collapse
};
```

2. Triangle list, a queue consisting of all the original triangles. Information on each triangle includes:

```
struct Triangle{
Vertex * vertex[3];     // three vertices
Vector normal;                  // unit normal vector
}
```

3. Vertex collapse cost heap, a binary tree consisting of some pointer to the object vertices, which are sorted on the basis of vertex collapse cost. Each node has the following structure:

```
struct CostHeap{
Vertex   * RefVertex; // pointer to a vertex
CostHeap * lchild;     // left child
CostHeap * rchild;     // right child
}
```

Here is the pseudocode of the algorithm presented in the paper:

```
Step 1  Input the initial mesh M = Mn;
Step2   For each vertex, find an edge that contains this vertex to have the lowest collapse cost. Let it
        be the collapse cost of this vertex;
Step3   Create an initial heap sorted on collapse cost of each vertex;
Step4   Use the top node of the heap to perform a collapse operation. Form the results into a
        sequence;
Step5   Recalculate the information of the affected neighbors and rearrange the heap;
Step6   If the simplification is inadequate, go back to Step 4,
Step7   Output the ultimate simplified mesh M0 and an edge collapse sequence;
```

From the analysis of the steps listed above, we see that at Step 2, where the collapse cost of each vertex is calculated, the complexity is O(n) calculated on the basis of the collapse cost computing operation. Step 3 creates a sorted heap, where the complexity as we know is O(n*lg(n)). Step 4 to

Step 6 is a process that rearranges the heap iteratively, and the basic operation is the calculation of collapse cost and the data comparison and moving operation. Its complexity is no more than O(n*lg(n)). Consequently we can draw a conclusion that the total complexity of our algorithm is O(n*lg(n)).

The algorithm was implemented on a PC platform with a Pentium II 400 MHz processor with 128M memory. A series of objects such as cow, human head, sculpture of Beethoven, church, and terrain were tested. The test shows that our algorithm not only runs fast enough to be used in realtime, but also preserves the surface shape very well. At the same time, the memory usage increased by storing the collapse cost heap in our algorithm is indistinct. Moreover, the pre-computing of Hoppe's algorithm is surprisingly expensive when processing the objects of similar scale. For example, the pre-computing time for simplifying an object with 5030 triangles is about an hour. The time performance of statistics of our algorithm is listed in the following table.

Objects	Initial Tris	Time(s)
cow 5804	0.062	
man head	5870	0.065
sculpture	5030	0.056
terrain	8192	0.104
church	14780	0.187

7. Other Properties

Many objects have surface properties beyond simple geometry. In computer graphics, the most common properties are surface normals, colors, and textures [12]. To produce approximations that faithfully represent the original, we must maintain these properties as well as the surface geometry. We shall assume that each vertex, in addition to its position in space, has some associated values that describe other properties. As with geometry, values will be linearly interpolated over the faces of the object. Consequently these properties must be continuous. Furthermore, we shall assume that the distance between two property values is measured with the usual Euclidean metric.

Consider first a Gouraud-shading model, for which each vertex has a color value [r, g, b]T associated with it. Naturally, we treated each vertex as a 6-dimensional vector of the form [x, y, z, r, g, b]T. From the study of the human visual system, we know that color has more visual importance than geometry. Thus we should scale the object so that the color components have a larger scale. Though it is difficult to give an optimal scale for each component, it can be offered as a user selected preference. Figure 6 shows the generated progressive meshes for a colored terrain. Since we have added a strict constraint to avoid border vertices to collapsing inner vertices, the reader can observe that the generated meshes retain their border property very well.

Second, we have already used surface normals to calculated curvature. Care must be taken to scale the normal vectors to unit length.

Third, if the surface textures are each mapped onto multiple surfaces and the covered vertices are all assigned corresponding texture coordinates, we simply consider the vertex texture coordinates (*s*, *t*). We shall treat them in the same way as geometry position, i.e. we calculate distance between two vertex texture coordinates. If the textures are each mapped onto a single face, we assume that the face must be large and has high visual importance, so as to not remove. Thus we add the strict constraint to these faces to prevent them from collapse.

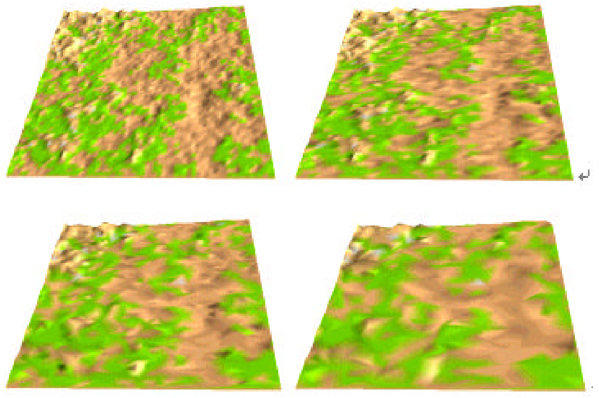

Figure 6. Progressive meshes of a colored terrain model

8. Summary and Future Work

Level of detail generation, in particular the smooth transition of progressive mesh is required in many VR applications where a dynamic environment is involved. The algorithm proposed in this paper can quickly produce good quality progressive meshes for medium-scale objects in modifiable environments. By the new approach, the generation of progressive meshes is based on two importance measures. The method is capable of rapidly producing quality approximations that preserve both surface shape and other important properties associated.

There are certainly further improvements that can be made to our algorithm. For example, we can organize all the properties to be associated with a vertex. In other words, the surface attributes are no longer needed. This would help solve the problems posed by surface normal discontinuities. We also believe that there must be a way to extend our algorithm to be view-dependent.

9. Acknowledgements

The research reported in the paper has been supported by the National Science Foundation of China (#69873044), High-tech Project of Distributed Interactive Simulation (863-306-ZD-10) and the Research Grant of University of Macao (RG021/98-99S).

REFERENCES

[1] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. "Decimation of triangle meshes". In Computer Graphics: Proceedings SIGGRAPH'92, Vol. 26, No. 2, pp. 65-70. ACM SIGGRAPH, 1992.
[2] J. Rossignac and P. Borrel. "Multi-resolution 3D approximations for rendering". In Modeling in Computer Graphics, pp. 455-465. Springer-Verlag, June-July 1993.
[3] H. Hoppe, T. DeRose, and T. Duchamp et al. "Mesh Optimization", Annual Conference Series, Computer Graphics, 1993, pp. 19-26.
[4] Guéziec. "Surface simplification with variable tolerance". In Proceedings of Second International Symposium on Medical Robotics and Computer Assisted Surgery, MRCAS' 95, 1995.
[5] B. Hamann. "A data reduction scheme for triangulated surfaces". Computer Aided Geometric Design, 11: pp. 197-214, 1994.
[6] P. Hinker and C. Hansen. "Geometric techniques for walkthrough applications". In Interactive Walkthrough of Large Geometric Database, Course Notes 32,SIGGRAPH' 95, pp.B1-B25, 1995.
[7] G. Turk. "Re-tiling polygonal surfaces". In Computer Graphics: Proceedings SIGGRAPH'92, Vol. 26, No. 2, pp. 55-64. ACM SIGGRAPH, 1992.

[8] J. Cohen, A. Varshney, D. Manocha, G. Turk, et al. "Simplification envelopes". In Proceedings of SIGGRAPH'96 (New Orleans, LA, August 4-9, 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 119-128. ACM SIGGRAPH, ACM Press, August 1996.

[9] T. D. Denose, M. Lounsbery, and A. Varshney. "Multiresolution analysis for surface of arbitrary topological type". Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[10] H. Hoppe, "Progressive meshes", In Proceedings of SIGGRAPH'96 (New Orleans, LA, August 4-9, 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 99-108. ACM SIGGRAPH, ACM Press, August 1996.

[11] P. Lindstrom, et al. "Real-time, continuous level of detail rendering of height field", In Proceedings of the 23rd Annual Conference on Computer Graphics, 1996, p.109.

[12] G. Bonneau, et al. "Level of detail visualization of scalar data sets on irregular surface meshes", In Proceedings of the conference on Visualization'98, 1998, pp. 73-77.

[13] W. Cai, et al. "An auto-adaptive dead reckoning algorithm for distributed interactive simulation", In Proceedings of the thirteenth workshop on Parallel and distributed simulation , 1999, pp. 82 -89.

[14] C. Huo, "An activity model for standards process for the Distributed Interactive Simulation (DIS)", In Proceedings of the Conference on Winter Simulation, 1993, p.1013.

[15] C.Y. Cheng, Z.G. Pan, J.Y. Shi, "A fast algorithm for generating progressive meshes", Journal of Image and Graphics, 3(11), 1998, pp. 946-949.

BIOGRAPHIES

Enhua Wu graduated from Tsinghua University in 1970 in Beijing and lectured at the university from 1970-1980. He conducted his Ph.D. research from 1980-1984 in the Department of Computer Science of University of Manchester, UK and received his Ph.D. degree in 1984. Since 1985 he has been working at the Institute of Software, Chinese Academy of Sciences, and has been a full professor since 1992. In recent years he has been invited to do visiting research by Lawrence Livermore National Lab. of the USA as well as Utah State University, Chinese University of Hong Kong, and the University of Hong Kong. He has also served as visiting professor in the Faculty of Science and Technology, University of Macao since September 1997. His main research interests are computer graphics, CAD and computer simulation, scientific visualization, physically based animation, and virtual reality.

Contact information:

Prof. Enhua Wu
Faculty of Science & Technology,
University of Macao, Macao
Email: mailto:fstehw@umac.mo, mailto:weh@ox.ios.ac.cn

Guangzheng Fei received his M.S. degree from the Department of Computer, Hefei University of Technology. Currently he is a Ph.D. candidate in the Institute of Software, Chinese Academy of Sciences. His current research interests include level of detail, progressive meshes, and distributed interactive simulation.

Contact information:

Guangzheng Fei
Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, China
Email: mailto:feigz@ox.ios.ac.cn