Mesh Simplification for Interactive Applications

Carlos González
Dpto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón de la Plana, Spain
cgonzale@lsi.uji.es

Jesús Gumbau
Dpto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón de la Plana, Spain
jgumbau@lsi.uji.es

Miguel Chover
Dpto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón de la Plana, Spain
chover@lsi.uji.es

Pascual Castelló
Dpto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón de la Plana, Spain
castellp@lsi.uji.es

ABSTRACT

Meshes used in real-time applications are usually composed of sub-meshes which contain vertices with different sets of attributes. This kind of mesh cannot be used directly in the current graphics pipeline architecture because meshes for interactive applications usually duplicate vertices to ensure that every vertex has a single set of attributes. This fact causes apparently contiguous surfaces to be split into pieces or patches, and so traditional simplification error metrics will fail in any attempt they make to simplify them.

Here we present a method for this kind of mesh which is based on edge collapses and takes into account the information about attributes that contribute to obtain a more realistic appearance of the object, like normals and texture coordinates, in the error metric and recalculating this information after the simplification steps.

Keywords

Interactive applications, simplification methods, error metric, texture coordinates, normals.

1. INTRODUCTION

Simplification methods make it possible to reduce the amount of geometry needed to represent an object while maintaining the visual quality as much as possible, which benefits the performance of the GPU.

Many articles about simplification techniques can be found in the literature. The vast majority of them are based on geometry metrics to define the order of collapses. These methods are relatively fast and offer good simplifications.

However, more recently, other kinds of methods have been developed that are not based on a geometric metric. Instead they use image-based or viewpoint-driven metrics to provide a visual error metric which can potentially offer better results than the traditional geometry-based error metrics. Due to the visual nature of these methods, they can take advantage of visual aspects such as occlusion to decimate those parts of the objects which are hidden or that are not important because they are parts of the object that are not "seen".

Moreover, these methods can detect how per-vertex information affects the visual appearance of the mesh and take that information into account for the simplification metric. This is important because simplifying an object without taking this information into consideration can, for example, cause valid geometric simplifications that completely distort texture coordinates or per-vertex normals.

Meshes used for real-time applications are usually composed of sub-meshes that contain vertices with different sets of attributes. To ensure that every vertex has a single set of attributes, they need to duplicate vertices. If these meshes were used directly by the current graphics pipeline architecture a distorted final object would be obtained.

A method for mesh simplification is presented in this section. It is based on edge collapses and was designed to take these situations into account in order to provide valid simplifications for real-time meshes, taking into account vertices with more than one single set of attributes while preserving the appearance of the object.

This method is especially useful for sub-mesh simplification because most of the current methods collapse vertices which are in the same spatial location, thus losing attribute information at the union of sub-meshes. Our method can handle this situation to provide good simplifications. Moreover, it takes into account the information about attributes that help to obtain a more realistic appearance of the object, such as normals and texture coordinates, in the error metric and recalculating this information after the simplification steps.

The rest of this paper is structured as follows. In chapter 2 we describe the background to this research. In chapter 3 we explain the method set out here. Chapter 4 shows some results and we conclude the paper in chapter 5.

2. STATE OF THE ART

Algorithms for mesh simplification can be classified as belonging four different types:

- *Vertex decimation* [Ciam96a] [Sch97a]. These methods are based on the removal of vertices from the mesh. Once a vertex is removed, all faces using that vertex are also removed and then the hole is re-triangularised. Because of the way it creates triangles, this kind of algorithms is limited to manifold meshes.

- *Vertex clustering* [Low97a] [Ros93a]. These methods are based on an inclusion box divided into several cells. All vertices that are included in a cell are collapsed into one single vertex and the triangles that share the removed vertices are updated. These methods tend to be very fast but the visual appearance of the final mesh is not relatively very good.

- *Edge contraction* [Gar97a]. These methods use an iterative selection of edges to be removed to decrease the level of detail. At each step, a single edge is selected for removal (or a pair of unconnected vertices). All faces sharing that edge are also removed, and the faces which share just one of the vertices of that edge are updated to cap the hole. Degenerated faces and edges are also removed.

- *Morphological operations* [Noo03a]. These methods apply morphological operations (in contrast volumetric objects), like erosion and dilation, to decrease the level of detail of objects. They are very fast and tend to offer good results.

The most extended and precise methods for surface simplification [Gar99a] [Hec97a] [Pup97a] use techniques based on iterative edge contraction. These methods make it possible to contract edges and join vertices so that the connectivity of the mesh is preserved. A weight is assigned to each edge in a pre-process that considers to the geometric importance of that edge in the simplification.

One of the most relevant improvements to the geometry-oriented simplification methods is the incorporation of vertex attributes, such as texture coordinates and normals, into the simplification metric. Hoppe extended his initial work [Hop96a] by proposing a new technique based on quadrics incorporating colour and texture coordinates [Hop99a]. The authors of the Qslim [Gar97a] [Gar98a] algorithm also extended their metric to take into account this kind of information.

Cohen et al [Coh98a] developed an algorithm based on edge collapses which convert vertex positions, diffuse colours and normals into texture and normal maps. This algorithm is based on a texture deviation metric.

Lindstrom et al. [Lin00a] handle the problem as a visual approach by creating a purely image-based metric. Basically, their method determines the cost of a collapse operation by rendering the model from a set of viewpoints. Then the algorithm compares the resulting images and adds the per-pixel error as an extra value for each pixel. All edges are then sorted using this error information so that the first edge collapses are those which have the least error.

The main advantage of the metric is that it offers a good balance between the geometry of the object and its vertex attributes in a natural way, without the user having to assign any weight to them. Its main disadvantage is its high temporal cost.

Luebke et al. [Lue01a] presented a method for view-dependent simplification using perceptual error metrics. Later, Williams et al [Wil03a] extended this work to shaded and textured meshes.

Zhang et al. [Zha02a] proposed a new algorithm that takes visibility into account. This work defines a new visibility function that considers the surface of the model and a set of cameras located on the surface of a virtual sphere surrounding the model. The number of cameras influences the precision and the temporal cost of the algorithm. Luebke et al. used up to 258 cameras. To guide the simplification process, they combined their visibility algorithm with Garland's quadric-based error metric [Gar97a].

Recently, Lee et al [Lee05a] introduced the *saliency* concept as an error metric, which was used for mesh simplification algorithms. Basically their work consists in the generation of a saliency map to be used in the Qslim [Zha02a] simplification algorithm.

3. METHOD

The method presented here takes into account the properties of the meshes used for games, so it can effectively handle the presence of duplicated vertices with different attributes, which are considered in the simplification. Algorithm 1 shows a pseudo-code of the method. This method extends a viewpoint-driven simplification method [Cas07a] with different improvements in order to present better appearance in the final objects. But the user can use only the needed extensions depending on the requirements.

```
Begin
For each edge (a) of the Model
   Collapse edge a
   Calculate collapse cost
   Insert the information (a,
        cost) in the queue q
   Undo the collapse of edge a
End For
While (queue q is not empty)
   Extract from queue q the edge
   with the least cost (a)
   Collapse edge a
   RecalculateCoordsText(a)
   Recalculate costs of neighbour
   edges of a and update position in
   queue q
End While
RecalculateNormals(Model)
End
```

Algorithm 1. Pseudo-code of the method.

Error metric

The error metric employed to test our simplification uses a viewpoint-driven model [Cas07a]. Its objective is to provide simplifications so that the simplified mesh makes the appearance as similar as possible to the initial un-simplified mesh. To do this, it applies the entropy from a viewpoint, a concept which is taken from the "Theory of Information". The entropy of a viewpoint is obtained from the distribution of the projected areas of the polygons of the mesh. Projected areas are calculated by analysing the frame buffer using a histogram. Different viewpoint-driven simplification methods can be found in the literature [Lue97a][ElS99a][Hop97a].

Several viewpoints are uniformly distributed around the mesh. The variation in the entropy of each of these viewpoints after a collapse operation is used to calculate the error that the operation has caused. The collapse operation used is an edge collapse whose resulting vertex is one of the two vertices of the collapsed edge. The collapses in the two directions are simulated to determine which collapse produces the least changes in the curvature of the mesh. It is also possible to determine the exact location of the resulting vertex by taking into account the curvature of the mesh.

The following equation shows the formula used to calculate the entropy of a viewpoint v, where Hv is the final entropy for v, Nf is the number of polygons in the mesh, a_i is the projected area of the polygon i and a_t is the total projected area.

$$H_v = \sum_{i=0}^{N_f} \frac{a_i}{a_t} \log \frac{a_i}{a_t}$$

The simplification process is an iterative process. Initially, the error caused by an edge collapse is calculated and stored in a heap. After that, the edge with the smallest associated cost is extracted from the heap. This process is repeated until the minimum desired polygon count is reached or there are no more possible collapse candidates in the heap.

Models usually contain additional attributes such as texture coordinates or normals. Interactive applications, such as games or virtual worlds, need to show correctly textured objects, because textures play a very important role in the final appearance of a rendered object.

If texture information is considered in the simplification method, the texture of the final object will be distorted. To take this into account, we use an extension to the error metric that takes into account texture information [Gon07a]. This extension is based on the existing borders in the texture image. Border detection is based on the Canny method for border detection in image space [Can83a] [Can86a]. After obtaining the borders of the texture image, texture coordinates are used to check whether an edge is colliding against any of these generated borders or not. All the edges that collide with a border are penalised by incrementing their cost. This will cause that edge collapses that would distort the final appearance of the mesh are avoided as much as possible during the simplification process. Figure 1 shows the eye model, its texture image and the borders obtained by the border detection method. Figures 2 and 3 show three levels of simplification applied to the eye model. Artifacts caused by a simplification method that does not take into account texture information can be observed in Figure 2. Figure 3 shows how the new texture extension solves these problems.

Figure 1. Eye model (left), texture image (centre) and borders detected in the texture.

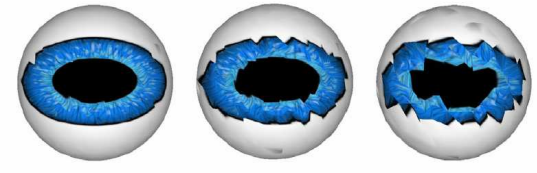

Figure 2. Eye model simplified to 75% (left), 50% (middle) and 25% (right) without applying the texture extension metric.

Figure 3. Eye model simplified to 75% (left), 50% (middle) and 25% (right) applying the texture extension for appearance preserving.

Simplifying with holes (duplicated vertices)

Meshes used in games often present different vertices with the same spatial coordinates. This is necessary to represent vertices with more than a single set of attributes (like the corners of a cube), but causes invisible holes in the mesh. These holes will become visible if the simplification method does not take that situation into account.

To solve this problem the simplification strategy makes use of three new concepts: real edge, twin edge and fake edge. Real edge refers to the collapsed edge; the twin edge is the edge joining two vertices which have the same spatial coordinates as the collapsed vertices. The meaning of fake edge is well illustrated in Figure 4. These new concepts help to avoid the existence of holes in the final object.

Figure 4 presents a simple example of a simplification step. This figure shows a part of a mesh composed by two different submeshes (green and yellow). The edge (va,vb) has been chosen by the simplification algorithm as the edge to be collapsed (real edge). After that, edge (vc,vd) is determined as the twin edge for (va,vb) and must also be collapsed in order to avoid a hole. However, this is not always sufficient to completely avoid holes in this kind of mesh because in some cases there is not an edge that can be collapsed to cap the hole (see Figure 4 - b, c, d). In these cases a new vertex (fake vertex) must be generated to create a fake edge so that it can be collapsed to effectively cap the hole. This fake vertex will be initially topologically disconnected and it will be used in the fake edge collapse. The attributes (normal, texture coordinates, bone assignments, and so forth) of the fake vertex will be calculated to reduce visual artifacts in the simplified mesh.

Note that we only need to introduce fake vertices if we are restricted to working only with indices (such as in some multiresolution algorithms [Ram04a]). Other vertices are simply translated to cap the holes resulting in face elimination. A pseudo-code is presented in algorithm 2.

This method is very useful for sub-meshes because it prevents holes from appearing in the joints of the sub-meshes when the simplification algorithm collapses and edge of one of the two sub-meshes.

```
While (EdgeList ≠ Ø)
   realEdge=extract(EdgeList)
   addSimplifList(realEdge)
   If(∃Twin(realEdge, EdgeList))
       twinEdge= Twin (realEdge,
                       EdgeList)
       addSimplifList(twinEdge)
   End If
   While (∃
  disconnectedVert(realEdge,
                   EdgeList))
       fakeVertex=
       disconnectedVert(realEdge,
                        EdgeList)
       fakeEdge= FakeEdge(realEdge,
                          fakeVertex)

   calculateAttributes(fakeVertex)
       addVertexList(fakeVertex)
   End While
End While
```

Algorithm 2. Pseudo-code for edge contraction.

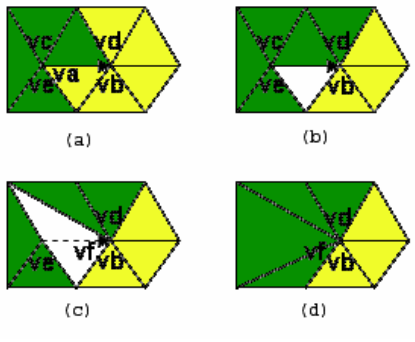

Figure 4. The edge collapse va → vb (real edge) forces the collapses of two other edjes: vc → vd (twin edge) and ve → vf (fake edge).

Texture coordinates

To obtain a better texture distribution over the simplified object, texture coordinates can be recalculated at each simplification step, which enhances the appearance of the simplified model. In an edge collapse, texture coordinates are recalculated taking into account the displacement of the vertex using a linear interpolation.

The following equations will allow us to calculate the displacement to be applied to the texture coordinates in order to avoid texture distortions in the simplified mesh.

$$\left.\begin{aligned} Q'_x &= \alpha U_x + \beta V_x + \gamma N_x \\ Q'_y &= \alpha U_y + \beta V_y + \gamma N_y \\ Q'_z &= \alpha U_z + \beta V_z + \gamma N_z \end{aligned}\right\}$$

where $\vec{Q}' = \vec{Q} - \overrightarrow{P^1}$, $\vec{U}' = \overrightarrow{P^2} - \overrightarrow{P^1}$, $\vec{V}' = \overrightarrow{P^3} - \overrightarrow{P^1}$, where $\{\vec{P}^1, \vec{P}^2, \vec{P}^3\}$ are the three vertices of the modified triangle, $\vec{Q}$ is the new position of the modified vertex and $\vec{N}$ is the normal of the triangle.

On solving this equation system we obtain the values for the unknowns α, β and γ, which are the coordinates of *Q* expressed in the coordinate system of the triangle. They also represent how the vertex has moved after the collapse, and hence they can be used to calculate the new texture coordinate for that vertex (see Figure 5). Thus, the new texture coordinates are calculated as follows:

$$\left.\begin{aligned} T_u^{res} &= \alpha(T_u^2 - T_u^1) + \beta(T_u^3 - T_u^1) + T_u^1 \\ T_v^{res} &= \alpha(T_v^2 - T_v^1) + \beta(T_v^3 - T_v^1) + T_v^1 \end{aligned}\right\}$$

where $\vec{T}_1$, $\vec{T}_2$ and $\vec{T}_3$ are the original texture coordinates for the three modified vertices of each modified triangle after the collapse.

It must be borne in mind that we have only used α and β in our equation because texture coordinates are two-dimensional vectors contained in the plane formed by the triangle. Because γ represents the displacement along the normal vector we do not need it. Figure 6 shows an example for this texture coordinate correction.

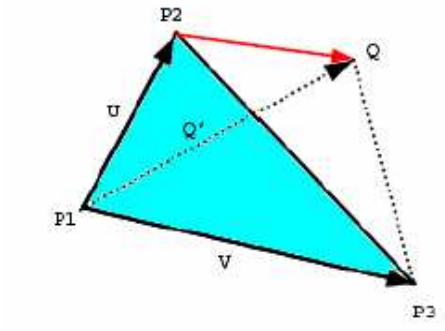

Figure 5. Displacing the new vertex after the collapse. P2 collapses to Q.

Normals

The final value of the normals influences the visual aspect of the simplified mesh and can cause invalid shading if they are not calculated correctly. So after the simplification process has ended, a renormalisation process can be applied to the final mesh to ensure correct calculation of all the normals.

To calculate correct normals for the simplified model it must be determined whether a polygon needs per-vertex or per-face normals. For example, the vertices of a cube must have face normals and the vertices of a sphere should have vertex normals.

The process used to calculate normals for the simplified model is the following:

- Duplicate vertices so that each vertex is used by exactly one triangle.
- Each vertex normal is initialised to the normal of the face using that vertex.
- All vertices sharing the same position and having faces with an angle less than a certain reference value (defined by the user) are collapsed into a single vertex with a normalised average normal.

A pseudo-code is shown in the algorithm 3. Figure 7 shows the results of applying our normal calculations over a simplified mesh

```
Function ExtendModel
Begin
count= 0
For ∀ Vertex ∈ Model do
   IndexN(Vertex)= count
   count++
   Normal(Vertex)= Normal(Face)
End For
End


Function CalculateNormals
Begin
For ∀ Vertex1 ∈ Model do
For ∀ Vertex2 ∈ Model do
   If Vertex1≠Vertex2 and
   Coordinates(Vertex1)==
   Coordinates (Vertex2)
       If Angle(Face1, Face2)<
                    EPSILON
             Normal(Vertex1)+=
             Normal(Face2)

             IndexN(Vertex2)=
             IndexN(Vertex1)
       End If
   End If
End For
End For
End
```

Algorithm 3. Pseudo-code for normals.

4. RESULTS

Our simplification method allows simplified objects to preserve the appearance of the original models as much as possible because both geometric considerations and vertex attributes are taken into account for the simplification.

Figures 8 and 9 show two examples of objects simplified using our method. Figure 8 shows the ninja model simplified to a 50% of its original geometric complexity. Figure 9 shows two simplifications for the racing car. Notice how texture details are preserved over the simplified objects.

A table is also presented showing times for various simplifications of different models, as well as the initial and final number of polygons (see Table 1). The greater part of the time is spent on calculating the entropy, due to the large number of renderings of the model that need to be performed in order to obtain such information.

Triangles Original	Triangles Final	Time (seconds)
815	100	12.27
4 698	500	93.45
4 806	1 200	97.33
6 592	500	153.18
8 468	500	226.78
10 474	1 000	300.07
13 810	1 000	453.34

Table 1. Simplification times for our method.

Figure 6. Original model (left), object simplified without recalculating its texture coordinates (centre) and the same object with application of texture coordinates perturbation.

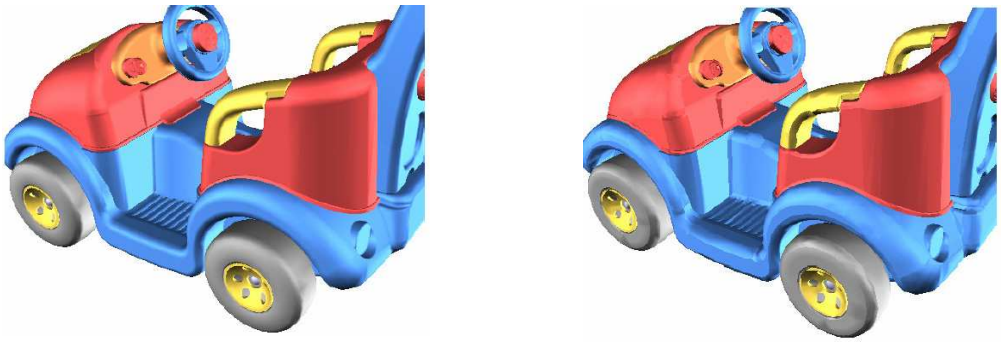

Figure 7. Original model (left) and the simplified model after recalculating (right).

Figure 8. Ninja model. From left to right: original model front view (1008 triangles), original model rear view, model simplified to 50% (504 triangles), front and rear views.

Figure 9. Racing car model. From left to right: original model (8345 triangles), model simplified to 66% (5506 triangles) and 33% (2753 triangles).

5. CONCLUSIONS

A simplification method has been presented which is designed for use with real-time meshes such as those used for games. This method preserves the final appearance of the simplified model as much as possible, taking into account that the meshes typically used for interactive applications often duplicate vertices in order to allow them to represent different per-face attributes, and also bearing in mind the normal and texture coordinate perturbation caused by edge collapses.

Due to the way duplicated vertices are handled his method is capable of simplifying sub-meshes while avoiding visual distortions in the final appearance of the mesh. This is especially useful if the user wants to simplify a single sub-mesh avoiding to loose per-vertex information, as occurs in other simplification methods that tend to collapse duplicated vertices. Thus, the texture coordinates and normals that are needed to obtain an accurate visual simplification can be preserved with this method.

6. ACKNOWLEDGMENTS

This work has been supported by the Spanish Ministry of Education and Science (MATER project TIN2004-07451-C03-03, TIN2005-08863-C03-03), the European Union (GAMETOOLS project IST-2-004363), the Jaume I University (PREDOC/2005/12, PREDOC/2006/54) and FEDER funds.

7. REFERENCES

[Can83a] Canny, J. “A variational approach to edge detection”. In AAAI-83. 1983.

[Can86a] Canny, J. “A computational approach to edge detection”. IEEE Trans. Pattern Analysis and Machine Intelligence, pp. 679-698. 1986.
[Cas07a] Castelló, P., Sbert, M. Chover, M. Feixas, M. “Viewpoint Entropy-driven Simplification”. Proc. of 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, pp. 249-256. 2007.
[Ciam96a] Ciampalini, A., Cignoni, P., Montani, C. and Scopigno R. “Multiresolution decimation based on global error”. Technical report, Centre National de la Recherche Scientifique, Paris, France, 1996.
[Coh98a] Cohen, J., Olano, M., Manocha, D. “Appearance preserving simplification”. SIGGRAPH '98, vol. 32, pp. 115-122. 1998.
[ElS99a] El-Sana, J., Varshney, A. “Generalized view-dependet simplification”. Eurographics ’99. Milano (Italy). 1999.
[Gar97a] Garland, M. and Heckbert, P. “Surface simplification using quadric error metrics”. In SIGGRAPH ’97: 24th annual conference on Computer graphics and interactive techniques, pp. 209–216. 1997.
[Gar98a] Garland, M., Heckbert, P. S. “Simplifying surfaces with color and texture using quadric error metrics”. VIS '98: Proc. of the conference on Visualization '98, pp. 263-269. 1998.
[Gar99a] Garland, M. “Multiresolution Modeling: Survey & Future Opportunities”. State of the Art Reports of EUROGRAPHICS'99, vol. 14 (4), pp. 111-131. 1999.
[Gon07a] González, C., Castelló, P., Chover, M. “A texture-based metric extension for simplification methods”. 2nd International Conference on Computer Graphics Theory and Applications, pp. 69-76. 2007.
[Hec97a] Heckbert, P. and Garland, M. “Survey of polygonal surface simplification algorithms”. Technical report, Multiresolution Surface Modeling Course Notes of SIGGRAPH’97, 1997.
[Hop96a] Hoppe H. “Progressive meshes”. SIGGRAPH '96: Proc. of the 23rd annual conference on Computer graphics and interactive techniques, pp. 99-108. 1996.
[Hop97a] Hoppe, H. “View-dependent refinement of progressive meshes”. SIGGRAPH ’97, pp. 189-197. 1997.
[Hop99a] Hoppe, H. “New quadric metric for simplifying meshes with appearance attributes”. VIS '99: Proc. of the conference on Visualization '99. IEEE Computer Society Press, pps 59-66.
[Lee05a] Lee C.H., Varshney A., Jacobs D.W. Mesh saliency. ACM Trans. Graph. 24, 3, pp. 659-666. 2005.
[Lin00a] Lindstrom, P., Turk, G. “Image-driven simplification”. ACM TOG 19, 3, pp. 204-241. 2000.
[Low97a] Low, K. and Tan, T. “Model simplification using vertex-clustering”. In SI3D ’97: Proceedings of the 1997 symposium on Interactive 3D graphics, pp. 75–ff. ACM Press, 1997.
[Lue97a] Luebke, D., Erikson, C. “View-dependent simplification of arbitrary polygonal enviroments”. SIGGRAPH ’97, pp. 199-208. 1997.
[Lue01a] Luebke, D.P., Hallen, B. “Perceptually-driven simplification for interactive rendering”. Proc. of the 12th Eurographics Workshop on Rendering Techniques, pp. 223-234. London, UK. 2001.
[Noo03a] Nooruddin, F. and Turk, G. “Simplification and repair of polygonal models using volumetric techniques”. IEEE Transactions on Visualization and Computer Graphics, 9(2): 191-205. 2003.
[Pup97a] Puppo, E. and Scopigno, R. “Simplification, lod and multiresolution - principles and applications”. Tutorial Notes of EUROGRAPHICS’ 97, 16(3). 1997.
[Ram04a] Ramos, J. F. and Chover, M. “Lodstrips: Level of detail strips,” in International Conference on Computational Science, 2004, pp. 107–114.
[Ros93a] Rossignac, J. and Borrel, P. “Multi-resolution 3d approximations for rendering complex scenes”. In B. Falcidieno and T. Kunii, eds, Modeling in Computer Graphics: Methods and Applications, pp. 455–465. Springer- Verlag, 1993.
[Sch97a] Schroeder, W. J. “A topology modifying progressive decimation algorithm”. In VIS ’97: 8th conference on Visualization ’97, pp. 205–ff. IEEE Computer Society Press. Los Alamitos, CA, USA. 1997.
[Wil03a] Williams N., Luebke D., Cohen J.D., Kelley M., Schubert B. “Perceptually guided simplification of lit, textured meshes”. Proc. of the 2003 symposium on Interactive 3D graphics. ACM Press, pp. 113-121. 2003.
[Zha02a] Zhang E., Turk G. “Visibility-guided simplification”. Proc. of IEEE Visualization 2002, vol. 31, pp. 267-274. Nov, 2002.