UNIVERSITAT JAUME I

Department of Languages and Computer Systems

Simplification Techniques for Interactive Applications

Ph. D. Dissertation

Carlos González Ballester

Advisors: Dr. Miguel Chover Sellés and Dr. Ricardo Quirós Bauset

Castelló, July 2010

UNIVERSITAT JAUME I
Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT JAUME·I

Técnicas de Simplificación para Aplicaciones Interactivas

Tesis doctoral
Carlos González Ballester
Directores: Dr. Miguel Chover Sellés y Dr. Ricardo Quirós Bauset

Castellón, Julio 2010

El uso de gráficos tridimensionales en aplicaciones interactivas está cada vez más extendido en campos cotidianos como los juegos, la enseñanza, los entornos de aprendizaje, la realidad virtual o el cine. Los escenarios mostrados en las aplicaciones interactivas tienden a presentar mundos y personajes detallados, intentando ser lo más realistas posibles. Los modelos tridimensionales detallados usados en este tipo de aplicaciones requieren mucha complejidad geométrica. Sin embargo, no siempre el hardware disponible es capaz de soportar y manejar toda esta geometría manteniendo una visualización suave para los usuarios. Los métodos de simplificación intentan solucionar este problema, generando versiones simplificadas de los modelos, los cuales presentan menos geometría que los originales. Esta simplificación se ha de llevar a cabo con un criterio razonable, con el fin de mantener lo máximo posible la apariencia de los modelos originales. Sin embargo, no únicamente la geometría es importante en los modelos tridimensionales, ya que éstos normalmente están también compuestos de atributos adicionales, los cuales son muy importantes para el aspecto final de los modelos. Existe mucho trabajo previo de simplificación, sin embargo aún existen numerosos puntos sin una solución eficiente. Por lo tanto, esta tesis se centra en técnicas de simplificación para modelos tridimensionales normalmente usados en aplicaciones interactivas.

Objeto y objetivos de la investigación

Esta tesis se centra en presentar nuevas técnicas como solución para la simplificación de objetos normalmente usados en aplicaciones interactivas tridimensionales. Los últimos avances surgidos en la informática gráfica han permitido presentar aplicaciones interactivas usadas diariamente con gráficos tridimensionales. Estas aplicaciones son diariamente usadas en una gran variedad de campos disciplinares y profesionales.

Las escenas 3D se representan normalmente mediante el uso de mallas poligonales. Una malla poligonal es una colección de vértices, aristas y caras que definen la forma de un objeto tridimensional. Uno de los polígonos más usados para la composición de una malla poligonal es el triángulo. Por lo tanto, los objetos tridimensionales pueden ser representados mediante triángulos interconectados entre sí. El estudio de mallas poligonales es un amplio subcampo de los gráficos por ordenador y el modelado geométrico.

Las nuevas tecnologías y los avances en los ordenadores han hecho posible la aparición de escenas con entornos altamente detallados en las aplicaciones interactivas. Algunas aplicaciones requieren el uso de escenas con apariencia lo más realista posible, lo que conlleva el uso de objetos detallados en las escenas. Así pues, las escenas necesitarán objetos con alta complejidad geométrica. Sin embargo, el hardware disponible no siempre puede soportar y manejar toda la geometría de las escenas manteniendo una visualización suave.

Se han presentado distintas soluciones a este problema, siendo la simplificación de los objetos una de las más extendidas. Los métodos de simplificación producen objetos menos detallados, reduciendo la complejidad geométrica de los originales. Por lo tanto, los objetos simplificados están compuestos por una menor cantidad de geometría que los originales. Las aplicaciones interactivas necesitan dibujar y manejar escenas tridimensionales con una visualización suave. Por lo que el uso de objetos simplificados puede ayudar a conseguir esto, debido a la reducción de información geométrica que producen.

Sin embargo, se necesita un criterio razonable a seguir para reducir la geometría de los objetos. Los métodos de simplificación buscan preservar la apariencia visual de los objetos, ya que los objetos simplificados tienen que presentar una apariencia similar a los originales. Los objetos tridimensionales usados en aplicaciones interactivas están normalmente compuestos por atributos adicionales a su geometría. Por tanto, en los métodos de simplificación se puede considerar no únicamente la geometría de los objetos, sino también algunos atributos adicionales, como las normales y las coordenadas de textura. Además, los modelos tridimensionales pueden estar compuestos de diferentes subobjetos, como por ejemplo la mayoría de objetos obtenidos en un proceso de diseño asistido por ordenador [TBG09].

Hay también técnicas adicionales que ayudan a mejorar el aspecto de los objetos simplificados, como el uso de mapas de normales. Los mapas de normales son imágenes con la información de las normales de modelos detallados, los cuales, al ser aplicados sobre los objetos simplificados, dotan de una apariencia más parecida a los objetos originales. Algunos artículos de estudio de métodos de simplificación pueden ser encontrados en la literatura [CMS98] [Lue01].

No únicamente el aspecto final en las escenas es importante, ya que hay veces en que el tiempo empleado por ciertos métodos puede ser también determinante. Durante los últimos años el hardware gráfico ha evolucionado mucho, y han aparecido nuevas arquitecturas y capacidades del hardware. Por lo tanto, existe la posibilidad de hacer uso de más herramientas y métodos cada vez más eficientes.

Pueden aparecer muchos problemas cuando se requiere una simplificación. Se tienen que tener en cuenta muchas consideraciones, dependiendo de cada problema particular. Quedan, por lo tanto, numerosos puntos a mejorar y estudiar en el campo de la simplificación. Esta tesis presenta nuevas soluciones para la simplificación de objetos normalmente usados en aplicaciones interactivas.

Planteamiento y metodología utilizados

Esta tesis tiene como objetivo la presentación de nuevas técnicas útiles para la simplificación de modelos tridimensionales normalmente usados en aplicaciones interactivas. Así pues, aquí se presentan el planteamiento y la metodología utilizados.

Estado del arte

Como paso previo al desarrollo de cualquier técnica es necesario hacer un estudio del estado del arte dentro del campo de investigación, a partir del cual se puede plantear el problema y presentar nuevas soluciones.

Desarrollo de técnicas de simplificación de modelos tridimensionales normalmente usados en aplicaciones interactivas

Basándose en el estado del arte, se han planteado varios problemas relativos a la simplificación de modelos tridimensionales normalmente usados en aplicaciones interactivas, a los cuales se les da solución con las aportaciones presentadas.

Evaluación y comparación de los resultados

Dependiendo de los resultados, se han hecho evaluaciones cualitativas y cuantitativas de los mismos. Además, estos resultados han sido comparados con los obtenidos con otros métodos conocidos presentados en la literatura.

Aportaciones originales

Las aportaciones resumidas de esta tesis son:

Método de simplificación asistido por el usuario para mallas de triángulos preservando fronteras

Los objetos obtenidos tras un proceso de diseño asistido por ordenador están normalmente compuestos por muchos subobjetos interconectados entre sí. Esto puede presentar problemas en la simplificación si se usan algunos métodos bien conocidos ([CMS98] [Lue01]), ya que pueden obtenerse agujeros. El usuario puede requerir además que los subobjetos tengan cada uno un nivel de detalle distinto al de los demás. Y esto puede también ser requerido mientras se mantiene un número total de polígonos para todo el objeto. Las herramientas de simplificación existentes no presentan métodos de simplificación eficientes para este tipo de objetos y no suelen hacer uso de las mejores métricas para una simplificación geométrica. Es por ello que este proceso suele ser realizado a mano por diseñadores, lo cual es un trabajo muy costoso y elaborado. Por tanto, presentamos un método de simplificación asistido por el usuario para este tipo de objetos. Este método preserva fronteras entre subobjetos y permite al usuario manejar el nivel de simplificación de los subobjetos. Esto puede además realizarse mientras un número total requerido de polígonos es mantenido para todo el objeto.

Extensión de métricas de error de métodos de simplificación basada en la información de texturas

Las texturas son muy importantes en el aspecto final de los objetos simplificados en aplicaciones interactivas. Sin embargo, existen muchos métodos de simplificación que no tienen en cuenta la información de las texturas a la hora de establecer el criterio de simplificación, lo cual puede producir grandes distorsiones cuando se aplica la textura a los modelos simplificados. Por ello, presentamos una extensión para las métricas de los métodos de simplificación que no consideran la información de texturas, de modo que puedan tener en cuenta esta información. Además, se ha hecho uso de esta extensión para métricas para la consideración de otros atributos adicionales a la geometría, como las normales.

Simplificación basada en la segmentación para modelos con texturas

Presentamos un método de simplificación que considera la información de las texturas en su métrica de error. Conocidos conceptos matemáticos, como la entropía y la información mutua [Sha48] y sus fórmulas generalizadas [Tsa88] [HC67], han sido usados en este método. Considerando la información de la textura en la métrica de error, los objetos simplificados presentan un aspecto más parecido a los originales que aquellos objetos simplificados obtenidos con métodos que no consideran la información de la textura. Los resultados han sido también medidos cuantitativamente con una conocida métrica de error: RMSE [LT00].

Aceleración por hardware de simplificación basada en el punto de vista

Los últimos avances del hardware gráfico permiten distribuir la carga entre la CPU y la GPU. Además, CUDA [nVi09a] permite paralelizar instrucciones dentro de la misma GPU. Por lo tanto, hemos aplicado estos avances para acelerar el tiempo de un método de simplificación basado en el punto de vista, puesto que este tipo de métodos suele tener un alto coste temporal.

Generación de mapas de normales en la GPU

Un mapa de normales es una imagen 2D con información sobre las normales de un objeto tridimensional. Aplicando el mapa de normales de un objeto original a una versión simplificada del mismo, obtenemos un objeto simplificado con mayor apariencia al original. Los nuevos avances en el hardware gráfico permiten acelerar algunas aplicaciones, programando código que será directamente ejecutado en la GPU. Presentamos dos métodos de generación de mapas de normales para objetos tridimensionales basados en la GPU.

Conclusiones obtenidas y futuras líneas de investigación

Conclusiones

Las aplicaciones interactivas necesitan manejar las escenas con una visualización suave para el usuario. Además, se requieren modelos con una buena apariencia. Sin embargo, no siempre el hardware disponible puede soportar y manejar toda la geometría necesaria para ello. Una de las soluciones que tratan este problema es el uso de métodos de simplificación ([CMS98] [Lue01]). Estos métodos reducen la complejidad geométrica de los modelos, intentando mantener la apariencia original de los mismos. Por lo tanto, en tesis se presentan nuevas técnicas en el área de la simplificación para objetos normalmente usados en aplicaciones interactivas.

En el Capítulo 2 se presenta un estado del arte referente a los trabajos realizados en el campo de estudio en el que se enmarca esta tesis. Se puede observar que se han realizado muchos trabajos y estudios relativos a la simplificación. Sin embargo, aún existen diferentes puntos clave sin soluciones eficientes. Por lo que en esta tesis se presentan varias soluciones dirigidas a algunos de estos problemas.

Los modelos obtenidos tras un proceso de diseño asistido por ordenador suelen estar formados por un alto número de subobjetos. Si se simplifican estos modelos con algún método de simplificación que no considere estas propiedades, se pueden obtener grandes distorsiones y agujeros en los modelos simplificados. Las herramientas de simplificación existentes no presentan un proceso automático eficiente para trabajar con mallas obtenidas de modelos CAD con la posibilidad de simplificar los subobjetos a distintos niveles de detalle. Este proceso suele ser elaborado y estas herramientas no siempre usan las mejores métricas para la simplificación. En el Capítulo 3 introducimos un nuevo método de simplificación asistido por el usuario para modelos obtenidos tras un procesos de diseño asistido por el ordenador. Este método permite obtener distintos niveles de simplificación en los distintos subobjetos de un modelo. El método presentado es un método asistido por el usuario, ya que el usuario puede manejar de manera independiente el nivel de simplificación de cada subobjeto, simplificándolos más o dando más detalle a los ya simplificados. Además, el usuario puede requerir un número total de polígonos en el objeto final. Por lo tanto, si el usuario cambia un nivel de simplificación de un subobjeto, otros subobjetos serán modificados de manera automática para preservar el número total de polígonos. Todo ello se realiza mientras las fronteras son preservadas, por lo que el método evitan la aparición de agujeros en el modelos simplificado. También se consideran las normales y coordenadas de textura en el método. En los resultados mostrados se puede observar que el método presentado soluciona de manera satisfactoria la simplificación automática y asistida por el usuario de este tipo de modelos.

No sólo la geometría es importante para el aspecto final de los modelos. Las texturas juegan un rol muy importante en la apariencia final de los objetos para el usuario. Sin embargo, hay muchos métodos de simplificación que no consideran esta información. Por lo tanto, los objetos simplificados pueden presentar grandes distorsiones cuando se les aplica las texturas. En el Capítulo 4 se presenta una extensión de métrica de error útil para tener en cuenta la información de textura en aquellos métodos de simplificación que no la consideran. Con esta extensión los métodos que no consideran las texturas en su métrica pueden fácilmente tenerlas en cuenta. Se puede apreciar en los resultados mostrados que se produce una gran mejora visual haciendo uso de nuestra extensión. Además, consideramos que este tipo de modelos normalmente pueden tener vértices duplicados en las mismas coordenadas espaciales. Esto se debe a que, muchas veces, se requieren en la misma posición diferentes atributos, como normales o coordenadas de textura. Haciendo uso de la solución presentada, se evitará la aparición de agujeros en los modelos simplificados, permitiendo diferentes atributos en las mismas coordenadas espaciales. Se presentan también experimentos del uso de esta extensión para preservar otros atributos, como las normales, pudiendo observarse mejores resultados visuales aplicando nuestra métrica.

Además, la preservación de texturas se contempla también en el Capítulo 5. En este capítulo se presenta un método de simplificación que considera la información de textura en su métrica de error y su aceleración, usando nuevas tecnologías en el hardware gráfico. En la Sección 5.2 se presenta un nuevo método de simplificación basado en la segmentación de las texturas para definir su métrica de error. En este método se han usado conocidos conceptos matemáticos de la Teoría de Información [Sha48]. Estos conceptos permiten presentar un método de simplificación robusto que preserva texturas. Se puede apreciar en los resultados que se producen simplificaciones que preservan la apariencia de las texturas. Estos resultados han sido medidos cuantitativamente con una conocida métrica de error (RMSE [LT00]). Este método es un método de simplificación basado en el punto de vista. Este tipo de métodos poseen normalmente un alto coste temporal. Por lo tanto, en la Sección 5.3 hemos implementado y evaluado técnicas para la aceleración de un método de simplificación basado en el punto de vista, como el presentado en la Sección 5.2, usando CUDA. Se puede comprobar que se obtienen mejores resultados temporales usando estos avances del hardware gráfico.

Una técnica comúnmente usada complementariamente a la simplificación es el uso de mapas de normales. Los mapas de normales son imágenes 2D con información sobre las normales de un modelo tridimensional. La aplicación de los mapas de normales de modelos detallados sobre modelos simplificados puede ayudar a dar más detalle sin necesidad de añadir geometría adicional. En el Capítulo 6 se hace uso de los nuevos avances del hardware gráfico y se presentan dos métodos diferentes para la generación de mapas de normales basados en la GPU. Por un lado, el primer método presentado (Sección 6.2) es muy rápido y es útil para modelos simplificados que preservan sus coordenadas de textura y sus texturas abarcan todo el espacio de textura. Por otro lado, el segundo método (Sección 6.3) está basado en el renderizado de las normales a partir de los triángulos del modelo simplificado. Ambos métodos son rápidos y presentan resultados con una calidad similar a la producida por los métodos implementados en la CPU. Estos métodos pueden además ser usados para obtener otros parámetros de la superficie de los modelos, como mapas de colores o mapas de alturas.

Futuras líneas de investigación

En esta sección se sugiere y discute algunas de las posibilidades de trabajo futuro relacionadas con esta tesis.

El estudio de tareas de paralelización ha sido objeto de investigación desde hace años. Por lo tanto, el estudio de la paralelización de métodos como el presentado en el Capítulo 3 podría ser posible. Este método es un método de simplificación asistido por el usuario para modelos con las propiedades de objetos normalmente obtenidos en un proceso de diseño asistido por ordenador. Por tanto, sugerimos el estudio de la paralelización de la simplificación de subobjetos.

Atributos adicionales a la geometría, como las texturas y las normales, son muy importantes para el aspecto final de modelos normalmente usados en aplicaciones interactivas. En el Capítulo 4 presentamos una extensión para las métricas de error útil para aquellos métodos que no consideren la información de las texturas, de modo que puedan tener en cuenta esta información en su métrica. Sugerimos el estudio de nuevas posibilidades matemáticas para el cálculo del coste de las aristas, basados en conceptos matemáticos importantes (como en el método presentado en el Capítulo 5), con el fin de poder contrastar resultados. Además, se ha hecho uso de esta extensión para preservar otros atributos, como las normales. Por lo tanto, podríamos estudiar la posibilidad de mezclar los valores de las coordenadas de textura y de las normales para calcular el coste de error asociado a cada arista o bien considerar otros nuevos atributos, como colores.

En el Capítulo 5 presentamos un método de simplificación basado en la segmentación de las texturas. Este método hace uso de importantes y conocidos conceptos de la Teoría de la Información y considera la información de la textura en su métrica de error. Sugerimos el estudio de la posibilidad de considerar otros atributos, como normales, en la métrica de error. Este proceso podría ser orientado a considerar otro atributo sin tener los demás en cuenta, o a obtener una combinación de la información de todos los atributos, de modo que se obtenga una métrica que contemple una mezcla de toda esta información. Otra sugerencia es probar con otros conceptos matemáticos robustos para el cálculo del coste de las aristas.

Con el fin de medir las mejoras visuales obtenidas con nuestras técnicas de simplificación que preservan texturas, se ha hecho uso de una métrica conocida (RMSE [LT00]). Esta métrica, sin embargo, podría no ser la más apropiada para algunos casos específicos. Esto se debe a que objetos simplificados con una buena apariencia visual para el usuario y con texturas bien preservadas podrían ser altamente penalizados por el RMSE debido a la eliminación de algunas partes del contorno de los modelos. Por lo tanto, sugerimos el estudio de nuevas métricas que puedan ser útiles para medir la calidad visual de los modelos simplificados para el usuario.

El tiempo empleado por algunos métodos podría ser importante. Los métodos de simplificación basados en el punto de vista, como el presentado en la Sección 5.2 implican normalmente un alto coste temporal. Debido a ello, en la Sección 5.3 hemos presentado una aceleración del método presentado en este capítulo. Las tarjetas gráficas se encuentran en continuo avance. Por lo tanto, otra sugerencia de trabajo futuro es estudiar las posibilidades de las generaciones de tarjetas gráficas para obtener una aceleración más eficiente del método.

La aplicación de estos mapas de normales sobre las versiones simplificadas de los modelos dotará a los mismos de más detalles. En el Capítulo 6 se presentan dos nuevos métodos para la generación de mapas de normales basados en la GPU. Sugerimos el estudio de métodos basados en la GPU que generen mapas con otra información, como mapas de colores o mapas de alturas. Otra sugerencia de estudio es el uso de la arquitectura CUDA para la aceleración del método presentado en la Sección 6.3. Usando CUDA algunas tareas de la generación de los mapas de normales podrían ser paralelizadas en la GPU.

... to my parents (Antonio and Agustina), brothers (José Antonio and Miguel Ángel), nephew and nieces (Thomas, Nuria and Cristina) and sisters in law (María José and Yasuko).

Preface

Abstract

Interactive applications with 3D graphics are used everyday in a lot of different fields, such as games, teaching, learning environments and virtual reality. The scenarios showed in interactive applications usually tend to present detailed worlds and characters, being the most realistic as possible. Detailed 3D models require a lot of geometric complexity. But not always the available graphics hardware can handle and manage all this geometry maintaining a realistic frame rate. Simplification methods attempt to solve this problem, by generating simplified versions of the original 3D models. These simplified models present less geometry than the original ones. This simplification has to be done with a reasonable criterion in order to maintain as possible the appearance of the original models. But the geometry is not the only important factor in 3D models. They are also composed of additional attributes that are important for the final aspect of the models for the viewer. In the literature we can find a lot of work presented about simplification. However, there are still several points without an efficient solution. Therefore, this thesis focuses on simplification techniques for 3D models usually used in interactive applications.

Keywords: simplification methods, user-assisted simplification, CAD models, normal map, attribute preservation, graphics hardware, GPU-based methods.

Funding

This work has been supported by the Jaume I University (PREDOC/2005/12), the Spanish Ministry of Education and Science (TIN2004-07451-C03-03, FIT-350101-2004-15, TIN2005-08863-C03-03, TIN2007-68066-C04-01, TIN2007-68066-C04-02, TIN2009-14103-C03- 01), Caja Castellón-Bancaja Foundation (P1-1B2007-56, P1-1B2009-34), the European Union (IST-2-004363, Ref. 226487) and FEDER funds.

Acknowledgements

Accomplishing a thesis is an important step in one's life. In this process a great amount of research knowledge and procedures are acquired. This goal could not be possible to fulfill without senior researchers that supervise the work. For this reason, I would like to thank to my thesis advisors, Miguel Chover and Ricardo Quirós, for their support and tutoring during all this process.

I would also like to thank to all the people that has participated directly and indirectly in my research education, especially to all my colleagues of the University Jaume I and the universities where I did research stays, the Technical University of Vienna and the University of Vigo.

Last but not least, I could not achieve this without all the support, friendship and encouragement of all my family and friends. Without them I would not be the same. I prefer not to write names, because they are a lot and I do not want to forget anybody. But they know who they are and that they are a very important part of my life. Thanks a lot to all of them.

Index

X INDEX

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Background and Motivation

3D computer graphics uses a three-dimensional representation of geometric data and they are under continuous study and improvement. Computer graphics advances have enabled three-dimensional graphics to be present in a large number of applications in everyday use. Three-dimensional graphic applications are commonly employed in different fields, such as virtual reality, teaching, interactive worlds, computer-aided design and cinema.

Three-dimensional scenes are usually represented with polygonal meshes. A polygonal mesh is a collection of vertices, edges and faces that defines the shape of a three-dimensional object. One of the polygon types that is most often used for this purpose is the triangle. Thus, three-dimensional objects can be represented with interconnected triangles. The study of polygonal meshes is a large subfield of computer graphics and geometric modeling. Figure 1.1 depicts a three-dimensional polygonal object.

New technologies and computing advances have made it possible to create scenes with very detailed environments. Some applications require realistic scenes which entails highly detailed objects in the scene. Therefore, the scene will need objects with high geometric complexity. However, the available graphics hardware cannot always handle all the geometry of the scene with a high frame rate, without slow transitions in the visualization.

Different solutions have been presented to this problem. One of these solutions is the simplification of the objects. Simplification methods produce less detailed objects, by reducing the geometric complexity of the original ones. Interactive applications need to render and manage 3D scenes with a high frame

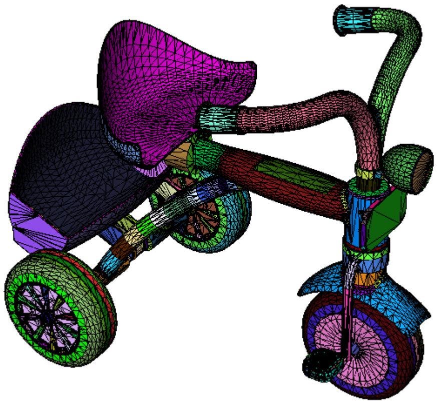


Figure 1.1: Three-dimensional polygonal object

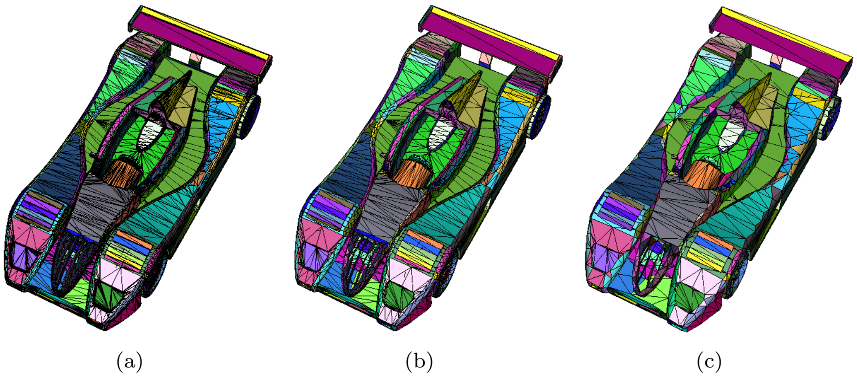


Figure 1.2: The original racing car object (a), a simplification of it to 25% of its original geometry and a simplification to 10% of its original geometry

rate. Thus, the use of simplified objects can help to achieve this, because of the reduction of the geometric information. In Figure 1.2 an original object and two simplified versions of it are depicted.

Interactive applications can use simplified models when these models do not have a great importance in the scene, for example, when they are far away from the viewer. However, a reasonable criterion is needed to reduce the geometry of the objects. Simplification methods attempt to preserve the appearance of objects, meaning that the simplified objects must have to present a similar appearance than the original ones. The simplification operation and the error metric used in a simplification method will establish, respectively, how and when each part of the geometry is simplified.

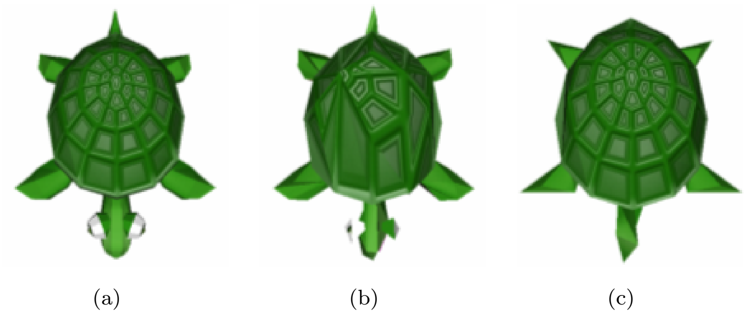


Figure 1.3: Original turtle model (a) and its simplification to 25% without considering texture information (b) and considering it (c)

Three-dimensional objects used in interactive applications are usually composed of attributes in addition to their geometry. Therefore, in simplification methods, not only the geometry of the objects could be considered, but also the additional attributes, such as normals and texture coordinates. The consideration of these attributes will produce simplified objects with a more accurate preservation of these attributes and, consequently, the viewer will see them as more similar to the original objects. If these additional attributes are not considered, the viewer may see simplified objects with great distortions. However, there are many simplification methods that do not consider this information. An example of this is shown in Figure 1.3, which depicts an original model and its simplification to 25% of its original geometry are depicted. This simplification has been performed by the method presented in [CSCF08], which does not consider the texture information in its error metric, and with one of the solutions presented in this thesis (Chapter 4). It can be appreciated that the consideration of these additional attributes helps to present better visual results.

Moreover, three-dimensional objects can also be composed of different subobjects as is the case with the vast majority of objects created in a computer-aided design [TBG09]. The simplification of objects composed of different subobjects can present difficulties with many simplification methods presented in the literature, such as the appearance of holes or undesired distortions in the boundaries of the subobjects (Figure 1.4). In Figures 1.1 and 1.2 the subobjects are represented by different colors.

A lot of simplification techniques, based on different criteria, have been presented in the literature. There are simplification methods that consider only the geometry of the objects. There are also viewpoint-driven simplification methods. These methods take not only the geometric information into account, but also the visual information by, for example, simplifying first those parts of the objects that are not going to be visible to the user. Considering only the geometric information of the objects in the error metric can produce good

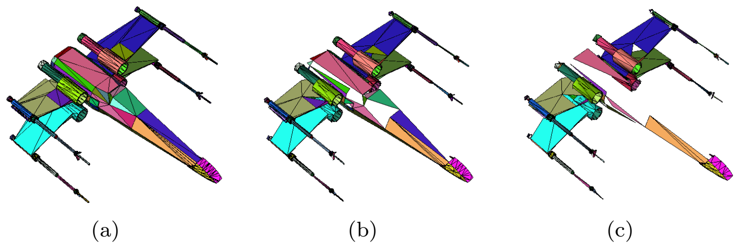


Figure 1.4: Simplifications of X-wing model without considering the properties of the meshes obtained after a computer-aided design process. Original model (a) and simplifications to 50% (b) and 30% (c) without considering the subobject information.

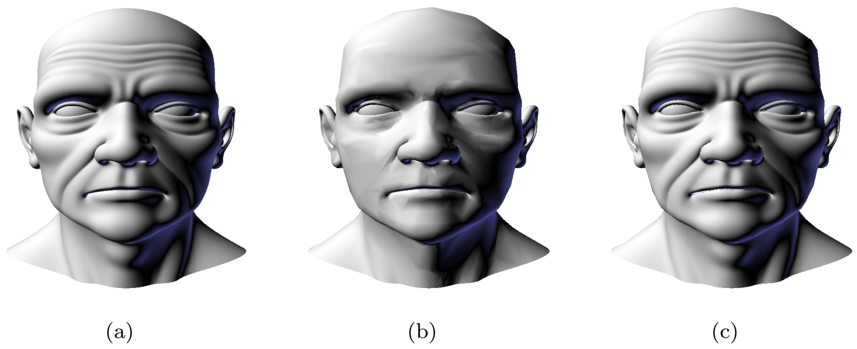


Figure 1.5: Normal mapping example: an original model (a), a simplified version of it (b) and the simplified version with the normal map of the original model applied (c)

geometric simplifications, but the consideration of the additional attributes to the geometry could be necessary in order to avoid possible visual distortions. Some surveys of simplification work can be found in [CMS98] [Lue01].

A common technique additional to the simplification algorithms is the use of normal maps. A normal map is an image of RGB elements that contains the information about the normals of a three-dimensional object. By applying the normal map of a detailed object to a simplified version of it, more detail is obtained. This is because it enables the use of correct per-pixel lighting. Therefore, more detail can be obtained in the simplified models without adding more geometry. Figure 1.5 shows the effect of applying a normal, extracted from a highly detailed model, onto a coarser mesh that is used for rendering the model in real-time without losing hardly quality.

The final appearance of the models in the scenes is not the only important requirement in interactive applications. The runtime requirements can also be important. Great advances in graphics hardware have been made during the last few years. New architectures and hardware capabilities have appeared, allowing the parallelization of tasks between the CPU and the GPU. Therefore, more possibilities are offered and more efficient methods can be implemented.

A lot of problems can appear when a simplification is needed. Different considerations have to be taken into account, depending on each particular problem. We have considered all the above-mentioned problems and presented different solutions.

1.2 Contributions and Overview

1.2.1 Contributions

The main aim of this thesis is to present different techniques for the simplification of three-dimensional objects usually used in interactive applications. In this section we provide a brief description of each contribution.

User-assisted Simplification Method for Triangle Meshes Preserving Boundaries Many three-dimensional objects are composed of different subobjects. An example can be the objects obtained from a computer-aided design process. This kind of object is composed of a lot of different subobjects. The simplification of this kind of object can present problems in the simplification when using well-known simplification methods ([CMS98] [Lue01]). Holes or artifacts can be produced. There are existing tools that allow the simplification of this kind of meshes. However, these tools do not use the best error metrics, used in simplification methods that can be found in the literature. Moreover, they do not always enable the possibility of simplifying the subobjects of the models to different levels of detail to be presented to the user. This is why automatic simplification processes are not usually employed for this purpose and the reduction of geometry is most commonly done by hand by the designers. This is a very elaborate process.

Therefore, a user-assisted simplification method for this kind of object is presented. This method preserves boundaries between subobjects and allows the user to manage the level of simplification of the subobjects. This can also be performed while maintaining a total number of polygons in the simplified object. This is very useful for obtaining simplified objects with some parts more or less simplified than others. The user can test with different levels of simplification at any time (for the whole object and each subobject) in order to find a simplification that satisfies his or her requirements.

A Texture-based Metric Extension for Simplification Methods Many interactive applications make use of three-dimensional textured objects. Textures are very important in the final appearance of the simplified objects in interactive applications. If the texture information is not taken into account in the simplification process, good geometric results can also be obtained, but the simplified objects may be greatly distorted when the textures are applied to them. This will produce a dramatic visual difference between the original objects and the simplified ones.

Some interactive applications may need to use simplified objects with an accurate texture appearance. However, there are a lot of simplification methods that do not consider the texture information in order to establish the simplification criterion.

We present a texture-based metric extension that takes texture information into account in order to establish the simplification criterion. This extension can be very useful for the metric of those simplification methods that do not consider the texture information. This work is based on the detection of the borders of the textures of the models. This extension can be used with any error metric that makes use of the edge collapse operation.

By applying this metric extension, more accurate texture appearance is obtained in the simplified objects. This metric extension will modify the order in which the polygons are simplified, attempting to simplify first those parts of the object that will produce less distortion in the texture.

Moreover, the consideration of this work for preserving other attributes, such as normals, is also exposed.

Segmentation-based Simplification for Textured Models

In the literature, we can find different simplification methods, such as [CSCF07], that make use of concepts of the Information Theory in order to establish the simplification criterion of the methods.

We present a simplification method that considers texture information in its error metric, using well-known concepts of Information Theory, like entropy [Sha48], generalized entropy [Tsa88], mutual information [Sha48] and generalized mutual information [HC67]. This is a view-dependent simplification method and it is based on the idea of the segmentation of the textures. In this method, we calculate how much the appearance of a textured model will change after each possible simplification step. This calculation is measured with these important concepts of Information Theory. The main idea is to obtain the simplification order that minimizes the change of the visual appearance for the viewer after simplifying each polygon.

Viewpoint-driven Simplification Acceleration by Hardware

The visual results are not only the important factor in the simplification methods. The time taken by some methods can also be important. View-dependent methods are usually slower than those based only on the geometry of the objects. This is because view-dependent methods usually perform a lot of renders in order to obtain the visual changes in the objects after each simplification step.

Graphics hardware is continuously improving. In recent years, new architectures and capabilities have appeared in the most recent generations of graphics cards. These improvements enable new and more efficient simplification methods, making use of these graphics hardware advances.

As one of the latest graphics hardware advances a new architecture called CUDA was introduced by nVidia [nVi09a]. This architecture enables the parallelization of instructions on the GPU. Therefore, we have used these new capabilities on the graphics hardware to speed up a viewpoint-driven simplification method. The use of these graphics hardware advances enables processing time to be significantly reduced.

GPU-based normal map generation A normal map is an image with information about the normals of a three-dimensional object. The use of normal maps can help to present simplified objects with a more similar appearance to the original ones. By applying the normal map of an original object to a simplified version of it, we obtain a simplified object with the normals of the original one. Therefore, it will present a more detailed shape, similar to the original object.

New graphics hardware advances enable some applications to be sped up, by programming code that will be executed directly on the GPU. Therefore, we have made use of these hardware advances in order to implement normal map generation methods.

We present two GPU-based methods for obtaining a normal map of a three-dimensional object. The first one is a very fast method that is useful for simplified models with some texture restrictions. The second method renders the normals of the detailed model through the triangles of the coarse model.

1.2.2 Overview

The rest of this thesis is structured as follows:

- Chapter 2. Previous Work: In this chapter, we introduce some work in the literature related to this thesis. First, we introduce some basic concepts of simplification. Then, we present a classification based on the simplification operation and the error metric used in the simplification methods. We also present the previous work on CAD model simplification, the parallelization of the simplification and the simplification methods based on the GPU. Finally, we introduce the previous work on the generation of normal maps and we finish this chapter with our conclusions.

- Chapter 3. User-assisted Simplification Method for Triangle Meshes Preserving Boundaries: In this chapter our user-assisted simplification method for meshes composed of several subobjects is presented. This method allows the subobjects to be simplified with different levels of detail while preserving the boundaries. Several experimental results are explained in order to present the different possibilities of the method.

- Chapter 4. A Texture-based Metric Extension for Simplification Methods: In this chapter we present our metric extension that is useful for considering the texture information of the models in those simplification methods that do not already consider it. Moreover, geometric aspects of the models most commonly used in interactive applications are also considered. The results have also been measured with a well-known error metric (RMSE) [LT00].

- Chapter 5. Segmentation-based Simplification for Textured Models: In this chapter we present our simplification method for textured models based on well-known mathematical concepts. Different metrics have been applied and compared. Moreover, a speed-up, making use of the latest graphics hardware advances, for viewpoint-driven simplification methods is also presented.

  We also present a quantitative study using different possibilities of the concepts of Information Theory. This quantitative study is measured with a well-known error metric (RMSE) [LT00].

- Chapter 6. GPU-based normal map generation: In this chapter we present our work related with the speed-up of the generation of normal maps. Two different GPU-based normal map generation methods are presented. First, we introduce a very fast normal map generation method for simplified models with texture correspondence. And secondly, we present another method that requires fewer restrictions to be accomplished in the models.

- Chapter 7. Conclusions and Future Work: Finally, we present our overall conclusions discussing the advantages and drawbacks of the presented work, and future work, proposing new ideas.

Chapter 2

Previous Work

2.1 Introduction

Polygonal models are commonly used in computer graphics, due to their mathematical simplicity. However, three-dimensional scenes are usually represented with a high degree of polygonal complexity. Thus, models used in three-dimensional scenes are usually composed of a great number of polygons. But the available hardware cannot always handle all this geometry with a high frame rate. Therefore, in interactive applications the accuracy of the models and the time required to process them must be taken into account.

In recent years, different solutions have been developed for interactive applications. One of these solutions is the simplification of the objects, which intends to reduce their polygonal complexity while maintaining the appearance of the final objects as much as possible. Therefore, simplification methods allow the amount of geometry needed to represent an object to be reduced, which benefits the performance of the GPU. This is performed by the simplification methods, attempting to maintain the aspect of the objects for the viewers. Many articles about simplification techniques have appeared in the literature and some surveys of such works can be found in [HG97] [PS97] [CMS98] [Gar99] [Lue01], including simplification methods for CAD models [TBG09].

The models commonly employed in interactive applications are polygonal meshes. In three-dimensional computer graphics, polygonal meshes are objects that represent their surfaces with a mesh of polygons. A mesh M could be composed of submeshes (S). Each submesh $s \in S$ is composed of their corresponding vertices, edges and faces. Information about the connectivity between these components of the geometry is commonly stored in a model. The geometry of a three-dimensional model can also be complemented with additional attributes, such as normals and texture coordinates.

In this chapter the previous work about simplification algorithms is presented. A lot of work has been realized during the last years. However, depending on the criteria of the simplification methods and the user's requirements about the simplification, different algorithms could be used. Therefore, we classify the previous work about simplification methods depending on their main characteristics.

First of all, we introduce some concepts in order to clarify some ideas about mesh simplification. Simplification concepts can be found in the literature [LRC+02].

A mesh is manifold when every edge is shared by exactly two faces. A mesh is manifold with boundaries when it can also have edges that pertain to only one face. Figure 2.1 depicts an example of a manifold mesh with boundaries, while Figure 2.2 shows some examples of non-manifold meshes.

Topology is an area of mathematics that deals with spatial properties, like the continuity and other concepts derived from it, regardless of the size and shape of the models. A simplification algorithm is a topology-preserving algorithm if it maintains connectivity and does not close holes. Some algorithms are simply topology tolerant, that is, they only simplify those parts of the models that are manifold.

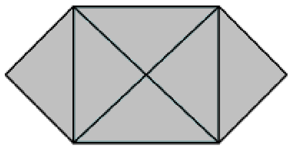


Figure 2.1: Example of a manifold mesh

A simplification method makes use of a simplification operation and an error metric. The simplification operation will define the way in which the geometry is removed in the simplification process. The error metric will establish the order of the simplification steps, assigning different simplification costs to the different parts of the geometry of the model. This order usually attempts to minimize the visual changes for the viewer after each simplification step. Therefore, we present a classification of simplification methods depending on their simplification operation (Section 2.2) and on their error metric (Section 2.3).

We classify in three groups the methods depending on their error metric. We first introduce those that consider only geometric information in their error metrics (Section 2.3.1). Then, those that take also visual aspects for the viewer depending on the user's point of view (Section 2.3.2). And those that consider

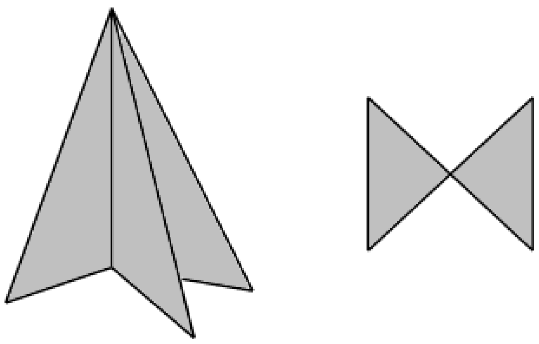


Figure 2.2: Example of non-manifold meshes

additional attributes to the geometry of the models (Section 2.3.3).

Meshes obtained after a computer-aided design process present special properties, because they are usually composed of a lot of subobjects that are not necessarily interconnected. Therefore, a section about CAD models simplification methods is presented in Section 2.4.

The visual appearance of the simplified models is not the only important factor in a simplification process. The time employed by the methods could also be important. We explain some new trends in simplification methods with parallel architectures (Section 2.5) and GPU programming (Section 2.6).

Finally, we expose some works in normal maps (Section 2.7), as a post-process additional to the simplification. With normal maps a final appearance in the simplified models more similar to the original ones is obtained.

2.2 Simplification operation

The *simplification operation* is the operation that is performed at each simplification step in order to remove geometry. Algorithms for mesh simplification can be classified according to the simplification operations. Simplification operations can be classified into local simplification operations and global simplification operations. Local simplification operations work with a small part of the geometry of the models. However, global simplification operations are more complex than local simplification operations and work by modifying the topology of the meshes in a controlled fashion.

Some of the most used local simplification operations are:

- Vertex decimation [CCMS97][Sch97]. These methods are based on the removal of vertices from the mesh. Once a vertex is removed, all the faces using that vertex are also removed and then the hole is re-triangulated.

Due to the way it creates triangles, this kind of algorithm is limited to manifold meshes. An example can be seen in Figure 2.3.

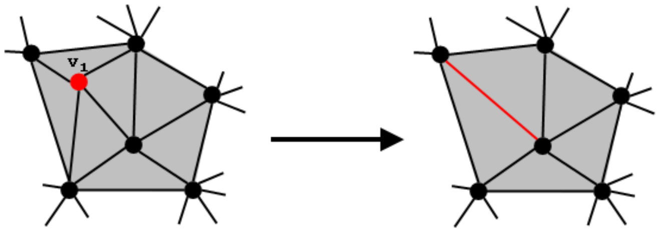


Figure 2.3: Example of vertex decimation operation

- Vertex clustering [LT97][RB97]. 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.An example can be seen in Figure 2.4.

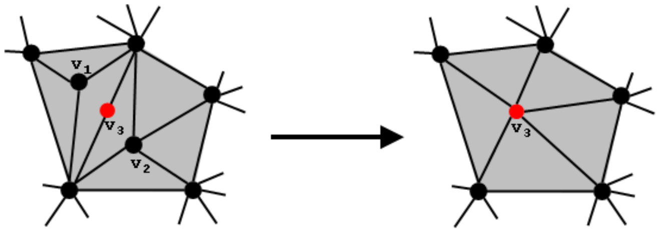


Figure 2.4: Example of vertex clustering operation

- Edge contraction [GH97]. These methods use an iterative selection of edges to be removed in order 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. An example can be seen in Figure 2.5.

And some of the global simplification operations are:

- Low-pass filtering [HHVW96]. These methods are useful for gradual elimination of high frequency details. The idea is to convert the objects into multiresolution volume rasters using a controlled filtering and sampling technique and generate a multiresolution triangle-mesh hierarchy by applying the Marching Cubes algorithm [LC87].

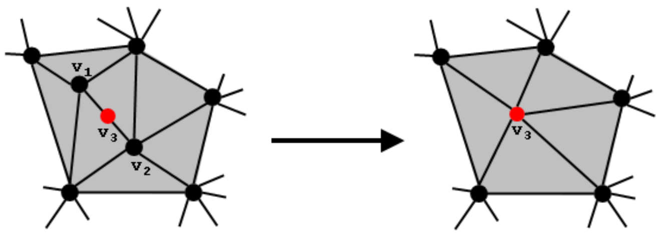


Figure 2.5: Example of edge contraction operation

- Morphological operations [NT03]. These methods apply morphological operations (in contrast to volumetric objects), like erosion and dilation, to decrease the level of detail of objects. Both operations are usually used in conjunction with each other. Dilation increases the bounds of the volume and erosion decreases them. These operations are usually used in 2D images. They are very fast and tend to offer good results. An example with 2D images is shown in Figure 2.6.

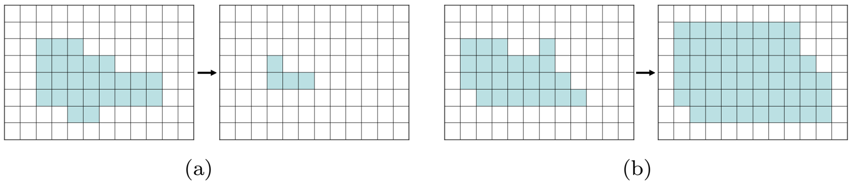


Figure 2.6: Erosion (a) and dilation (b) operations

2.3 Error metric

The *error metric* of a simplification method is the metric employed in the method in order to establish the order of the simplification steps. Many simplification methods are based solely on the geometry of the models (subsection 2.3.1) and other methods are also based on the viewer's point of view (subsection 2.3.2). Moreover, recent simplification methods attempt to preserve attributes additional to the geometry of the models (subsection 2.3.3). Methods based on geometry take only this information into account in order to establish the error metric that will give us the order of the simplification steps. However, simplification methods based on the user's point of view also take visual information to compute the error metric, that is, they will, for example, remove first those parts of the object that are not visible to the user.

2.3.1 Geometry-based simplification methods

Schroeder et al. [SZL92] presented an algorithm to reduce the number of triangles in a triangle mesh, generated by the marching cubes algorithm [LC87]. The algorithm presented by Schroeder et al. works by removing vertices and retriangulating the resulting holes. The decimation criterion is based on vertex distance to plane or vertex distance to edge. This algorithm is topology-tolerant. In [Sch97] they extended their work to make the algorithm topology-modifying.

Rossignac and Borrell [RB97] proposed the idea of simplifying the mesh by clustering. They define the relative perceptual importance of the vertices by assigning weights. Then, they group the vertices into clusters and for each cluster a representative vertex is computed. This method can achieve a high data reduction rate with very low computational cost by omitting the preservation of topology of the models. This method was extended in [KTKN05] in order to make it topology-preserving. The work presented in [LT97] is a thorough study of the vertex-clustering method with careful consideration of approximation quality and smoothness in transitions between different levels of simplification during interactive viewing.

The problems of compression/simplification and level-of-detail control have been addressed by Turk [Tur92], Schroeder et al. [SZL92], Hoppe et al. [HDD+93], Rossignac and Borrel [RB97] and Varsney [Var94]. Lounsbery [LDW97] developed a technique for creating multiresolution representations for meshes with subdivision connectivity. Unfortunately, the meshes usually used in practice typically do not meet this requirement. In [EDD+95] a method for solving this requirement is presented. The method is based on the approximation of an arbitrary initial mesh by another mesh that has subdivision connectivity. The problem is that it is not always easy to find a base mesh.

The method proposed by He et al. [HHK+95] performs a gradual elimination of high frequency details by sampling and low-pass filtering the object into multi-resolution volume buffers and applying the marching cubes algorithm to generate a multi-resolution triangle-mesh hierarchy. This method presents poor results for models with sharp edges and this algorithm is not topology-tolerant.

Hoppe [Hop96] presented progressive meshes, a new format for storing and transmitting arbitrary triangle meshes. It makes use of edge collapse and vertex split operations. With a series of edge collapse and vertex split operations, a mesh can be simplified and returned to its original level of detail. An efficient implementation of this method is found in [Hop98].

Cohen [CVM+96] presented a method based on the creation of envelopes for guiding the simplification process. The envelopes cannot intersect and are located around the original mesh. This method preserves the topology.

Some of the most widely extended methods for surface simplification [Gar99] [HG97] [PS97] 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 the geometric importance of that edge in the simplification.

One of the most important simplification algorithms is QSlim [GH97]. Qslim computes the error metric by associating a set of planes with each vertex of the model. The set of planes at a vertex is initialized to be the planes of the triangles that meet at that vertex. The edges are stored in a heap ordered according to their associated error and the edges with the lowest costs are the first to be contracted. At each simplification step one edge of the model is removed, as can be seen in Figure 2.5.

QSlim minimizes the function which gives the squared distance of a vertex v from its associated set of planes, that is, those defined by the polygons that contain this vertex. This function Δ(v) is defined as:

$$\Delta(v) = \sum_{p \in planes(v)} (p^T)^2 = \sum_{p \in planes(v)} v^T (pp^T)^2 v = v^T \left(\sum_{p \in planes(v)} pp^T \right) v \tag{2.1}$$

Thus, the Qslim algorithm operates as follows:

1. A quadric error associated to each vertex is computed.
2. A collapse cost is associated to each edge.
3. The edges are stored in a queue according to their collapse cost.
4. The edge with the lowest cost is collapsed.
5. The costs of the adjacent edges are recalculated.
6. Repeat steps 4 and 5 until we obtain the desired level of detail.

In [DEGN99] the authors presented a topology-preserving method. They introduced the link conditions. If these conditions are satisfied, the complex obtained after an edge collapse is guaranteed to be homeomorphic to the original one.

Velho [Vel01], inspired by his previous work on hierarchical 4-k meshes [VC00], developed a simplification method for polygonal meshes that is useful for variable simplifications. A fast implementation of this algorithm was presented in [VTV$^+$04].

Wu et al. [WHST01] introduced a mesh simplification scheme based on face constriction. This work uses statistical measures to distinguish between rough and flat parts in the models. This method marks a face constriction as illegal if a triangle belongs to a boundary or is non-manifold.

In [KG03] Kho and Garland presented a user-guided simplification method that extends Qslim in order to simplify the selected regions of the model.

Wu et al. [WHC04] presented triangle mesh simplification method based on a quadric error metric, which can preserve the global geometry features of the original model in and efficient manner. After finding all global geometry features by detecting the crease angles of two connecting faces, every edge is assigned a weight according to the relationship between it and the global geometry features. Then the quadric error metric is modified to postpone the simplification of global geometry features by adding the assigned weight to the contraction cost of every edge.

Garland and Zhou [GZ05] presented a method for simplifying simplicial complexes of any type embedded in Euclidean spaces of any dimension.

Both the geometry of the object and also the texture frequencies were considered in [XSX05]. To make the method more precise, pixels are subdivided into subpixels.

Vivodtzev et al. [VBLT05] presented a test for changes in the topology after an edge collapse in meshes with embedded polylines. To do so, they define the extended complex, which encodes both the topology of the mesh and the topology of the embedded polylines.

Jong et al. [JTY06] presented a method that uses torsion detection to improve the Quadric Error Metric of Vertex-Pair Contraction and retain the physical features of the models.

In [TFC08] Tang et al. presented a method that can preserve some interesting parts of the model with a high resolution, while the rest of the model is simplified with a lower resolution.

Daniels et al. [DSSC08] introduced a simplification algorithm for meshes composed of quadrilateral elements. Quadrilateral connectivity is maintained during the simplification by an extension of a quadric error metric to quad meshes.

Some works have been presented for very complex models composed of a great amount of data. These works refer to out-of-core simplification. In [WK03] input and output meshes of arbitrary sizes can be handled. The algorithm reads the input from a data stream in a single pass and applies decimation operations on the data kept in the main memory. In [ILGS03] the authors show how out-of-core mesh techniques can be adapted to the processing sequence paradigm ([IG03] [IGS03]). A processing sequence is an ordered sequence of indexed triangles and vertices that represents a mesh. This representation allows very large meshes to be streamed through main memory. At any time, only a small portion of the mesh is kept in-core. Vo et al. [VC07] proposed a two-step approach for streaming simplification of large tetrahedral meshes.

2.3.2 Viewpoint-driven simplification methods

Different viewpoint-driven simplification methods can be found in the literature. Hoppe [Hop97], for instance, presented a viewpoint-driven version of the progressive meshes algorithm [Hop96]. The criterion of this method is based on the view frustum, surface orientation, and screen-space. The first parts of the object it simplifies are those that are not visible to the user.

Luebke et al. [LE97] presented Hierarchical Dynamic Simplification (HDS), which works with a hierarchy of clusters of vertices. When a cluster occupies a volume on the screen below a certain threshold, all vertices within that cluster are collapsed.

Hoppe [Hop97] extended his previous work [Hop96] to selectively refine an arbitrary progressive mesh, depending on view parameters. The refinement criterion is based on the view frustum, surface orientation, and screen-space geometric error.

El-Sana and Varshney [EsV99] performed view-dependent geometry and topology simplifications using a view-dependent tree of general vertex-pair collapses.

Lindstrom et al. [LT00] take a visual approach to deal with 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. The algorithm then 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.

Luebke et al. [LH01] presented a method for view-dependent simplification using perceptual error metrics, where simplifications may be ordered according to their perceptibility.

Zhang et al. [ZT02] 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 [GH97].

Lee et al. [LVJ05] 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 algorithm as in the simplification algorithm described in [ZT02].

Recently, Castelló [CSCF07] presented a view-dependent method based on the entropy from a viewpoint, a concept which is taken from the ”Theory of Information” [Sha48][CT91]. 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 analyzing the frame buffer using a histogram. Mutual information concept was used in [CSCF08].

2.3.3 Attribute-preserving simplification methods

One of the most relevant improvements to the simplification methods is the incorporation of vertex attributes, such as texture coordinates and normals, into the simplification metric.

Hoppe extended his initial work [Hop96] by incorporating color and texture coordinates [Hop99]. The authors of the Qslim [GH97] algorithm also extended their metric so as to take this kind of information into account [GH98].

Cohen et al [COM98] developed an algorithm based on edge collapses which converts vertex positions, diffuse colors and normals into texture and normal maps. This algorithm is based on a texture deviation metric.

In [CMSR98] a general approach for preserving detail on simplified meshes is presented. The high frequency information lost after simplification is encoded through texture. This approach allows any attribute value defined on the high resolution mesh to be preserved.

Erikson and Manocha presented a method called General and Automatic Polygonal Simplification [EM98] that works on models that contain both non-manifold geometry and surface attributes. This method uses a distance threshold and surface area preservation to join unconnected regions of an object.

Sander et al. presented a method [SSGH01] that extended the work introduced in [Hop96]. This method subdivides the surface in patches, on the grounds of its coplanarity. It then generates parameterization minimizing the stretch deviation. It calculates an adequate size for each object in the texture domain and simplifies the mesh, minimizing the texture deviation [COM98] and preserving the boundaries. And finally, it optimizes the parameterization with a different objective function and regroups all the patches again.

Fahn et al. [FCS02] proposed a method based on the quadric error metric introduced by Garland and Heckbert [GH97] to preserve face colors and boundary edges during the simplification process, using a new constraint scheme.

The method presented in [CC06] recalculates a new texture for each simplification step, using an indexing map to avoid loss of precision.

The main advantage of the metric used in the method presented by Lindstrom et al. [LT00] 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.

Williams et al. [WLC$^+$03] extended the method presented by Luebke et al. [LH01] to shaded and textured meshes, and used parametric texture deviation to bound the size of the changes in the simplification.

2.4 CAD model simplification

Different physics-based methods for CAD model simplification can be found in the literature [TBG09]. These methods can be classified into different types of techniques, like techniques based on surface entities [FRL00] [DKK$^+$05], techniques based on volumetric entities [ABA02] [Lee05b] [Lee05a] [LL06] [LLK06], techniques based on explicit features [DKKN06] [RHG$^+$01] and techniques based on dimension reduction [SFM05] [FLM03].

Fine et al. [FRL00] introduced idealization operators for Finite Element Analysis. These idealizations are carried out through a vertex removal process which transforms the geometry of a part while preserving it within a discrete envelope defined around its initial geometry. Date et al. [DKK$^+$05] for controlling the properties of a given multimaterial tetrahedral mesh for finite-element analysis. The authors reported vertex and edge collapse based technique for mesh model simplification and refinement.

Andújar et al. [ABA02] presented a method for generating coarse-level approximations of topologically complex models. Lee et al. [Lee05b] [Lee05a] [LL06] [LLK06] presented a feature-based non-manifold modeling system is developed for CAD and CAE applications.

Date et al. [DKKN06] proposed a feature and resolution control method of triangular meshes to realize efficient mesh uses. This method includes feature recognition, mesh simplification and feature recovery. Ribelles et al. [RHG$^+$01] proposed a method for recognizing and suppressing features. This method makes use of the face clustering operation.

Sud et al. [SFM05] presented an algorithm to compute a simplified medial axis of a polyhedron. This simplification algorithm removes unstable features. Moreover, it preserves the topological structure. In [FLM03] Foskey et al. used the concept of θ-MAT for the simplification of polyhedral mesh models.

However, our work is included in the context of real-time rendering. Meshes are obtained from CAD models after a meshing process [LC87] [Lun91] [WSO03]. Over the last few years many simplification methods for triangle meshes have been developed. A survey of them can be found in [LH01]. But meshes obtained from CAD models are usually composed of a great number of submeshes. These submeshes are not necessarily interconnected. Simplification methods do not always produce simplified models with the levels of detail of the different subobjects required by the user. Moreover, some of these methods can produce undesired artifacts, such as holes or distortion, with meshes obtained from CAD models because these models are normally composed of different subobjects without any interconnection between them.

2.5 Parallel mesh simplification

A great number of simplification algorithms can be found in the literature [HG97] [PS97] [CMS98] [Gar99] [Lue01]. These methods allow high compression ratios to be obtained, but, unfortunately, many of them suffer from excessive memory consumption and high execution times. One solution to solve these limitations is provided by the use of scalable parallel computer systems. Times can be improved by using systems with distributed memory. When considering parallel model simplification algorithms, two key factors have to be taken into account:

- The computation should be as fast as possible; often this is one of the main reasons for parallelizing an application.
- The generation of high quality simplified models must be guaranteed.

An efficient data-parallel mesh simplification algorithm is presented in [SR00]. In this work the authors use the Adsmith-Library [LKF96] to simulate a distributed shared memory. The original mesh is partitioned into submeshes and the method distributes each submesh to a child processor for parallel simplification.

Langis et al. [LRD99] presented a parallel method for progressive mesh simplification. This method works by considering the original mesh as a graph and partitioning it. Each partition is sent to a processor of a parallel system and all the partitions are converted in parallel into the progressive mesh format using a serial algorithm on each processor. The results are then merged together to produce a single large progressive mesh file, considering the borders. A progressive mesh (PM) is a continuous mesh representation of a given 3D object which makes it possible to efficiently access all mesh representations between a low and a high level of resolution.

In [FS01] the authors present a fast algorithm for triangular mesh simplification in parallel environments using vertex decimation operations. The authors define an independent set of vertices to avoid critical sections.

Approaches that divide the models into a number of equally-sized chunks and distribute them to a number of potentially heterogeneous workstations are usually likely to fail. Brodsky et al. [BP03] proposed a general parallel framework for simplification of very large meshes that provides an intelligent partitioning of the model. They tested this framework by implementing a parallel version of [Bro00].

2.6 GPU-based simplification methods

Recent advances in real-time rendering have provided a new way to speed up the simplification rates by implementing simplification methods on the GPU. The work can therefore be distributed between the CPU and the GPU, thereby parallelizing the simplification tasks.

Recent GPUs execute shader programs, which allow operations in the graphics pipeline. That is, a programmable graphics pipeline is obtained, and programmers can modify the traditional and static instructions performed by the GPU. The vertex shader allows per-vertex computations to be performed. Once the primitives to pixels have been converted into pixels, the pixel shader can compute the colors of the fragments. This provides a way to parallelize arithmetic and texturing computations. The geometry shader is presented in [Bly06], and allows per-face computations, since it has access to the adjacency information. Figure 2.7 shows the architecture of the dynamic pipeline.

CUDA (Compute Unified Device Architecture) is a new compiler and tool package developed by nVidia that allows some tasks to be parallelized in the GPU. This architecture provides new possibilities of speeding up the simplification methods. For example, in [RO08] some methods that are useful for mesh operations are implemented in CUDA. Figure 2.8 depicts the CUDA architecture.

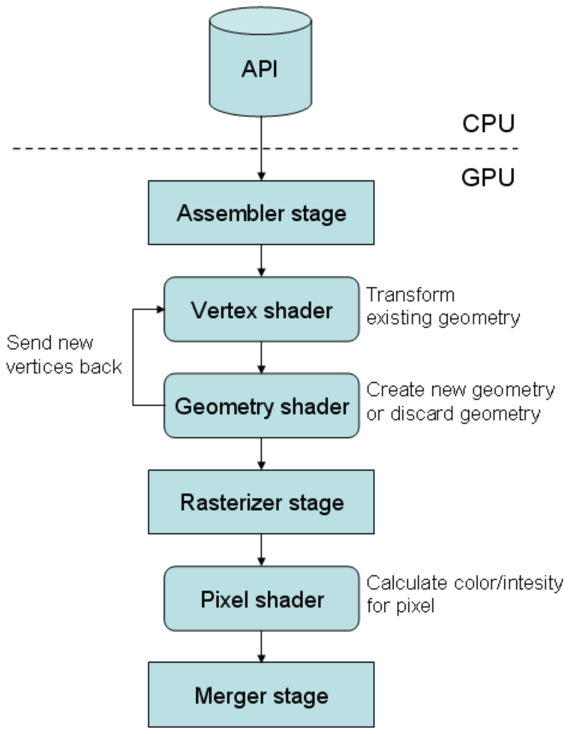


Figure 2.7: Dynamic graphics pipeline

Some methods for interactive visualization of large multiresolution geometric models at interactive rates, such as [SM05], make use of the GPU to perform geomorphing to render the objects. These methods need a hierarchy of the levels of detail.

Traditionally, CPU-based mesh simplification has been a slow operation, performed as a pre-process on static meshes. The advances in the GPU programming, for example [Bly06], have allowed GPU-based simplification methods to appear. The real-time mesh simplification method using the GPU

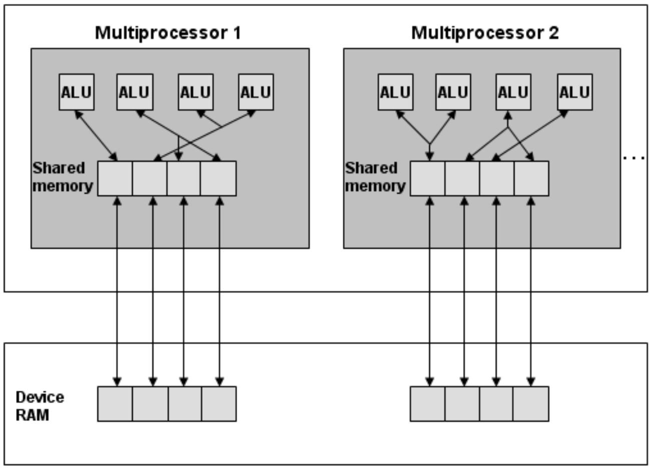


Figure 2.8: CUDA architecture

presented in [DT08] uses cluster-quadric maps to encode the mapping from cluster-cell index to cluster quadric in a render target. Moreover, it makes use of non-uniform clustering using warping functions and probabilistic octrees.

Hjelmervik and León [HL07] presented a method based on the GPU for meshes usually employed in mechanical-based applications. The method works by performing the computations for all the vertices in parallel using graphics hardware, and then utilizes the CPU to maintain the data structure representing the triangulation.

2.7 Normal maps

A normal map is an image with the information about the normals of a high resolution model. This normal map can be applied to a low resolution model in order to obtain a similar appearance to the more detailed one. A normal map contains information about the surface of the object and so it may be altered in order to modify the appearance of the object without changing its geometry. In this way, the normal map can be used in the low resolution model, so that it takes on the aspect of a high resolution one, thus avoiding the need to create more triangles with the resulting savings in computational and temporal costs.

Some other authors have presented works about the generation of normal maps for simplified meshes. [SSGH01] [SGR96] [CMSR98] generate an atlas for the model so that they can sample the color and normal values of the surface to be stored in a texture, which will be applied over a simplified version of the same original model. However, these methods need the coarse version of the mesh to be a simplified version of the sampled mesh, which is a disadvantage.

Some applications for generating normal maps have already been presented, but all of them generate the map by software with the corresponding CPU usage, like the method presented in [TCRS00].

Although other authors' [WFG02] implementations take advantage of graphics hardware for the generation of normal maps, they do not exploit it completely as they only use the rasterizing stage of the GPU, performing other tasks on the CPU. Moreover, this method has some limitations: it cannot generate the normals for faces that are occluded by other parts of the model, and it does not take full advantage of the capabilities of the graphics hardware and has to perform some read-backs from the color buffer.

Normal maps can be created by 3D edition programs, such as, for example, *3D Studio MAX 7* or *Maya 6.0*. Applications exclusively dedicated to the creation of normal maps also exist, examples being *ATI's NormalMapper* [ATI02] or *nVidia Melody* [nVi04a].

nVidia's Melody is an independent program, which presents a simple interface with different options to load and generate the normal map.

Ati's Normal Mapper offers libraries and is managed by a command line.

The software applications by nVidia and ATI make use of the object at two levels of detail, so that the normals are cast from the low resolution model to the high resolution model, and the normal from the high resolution model is taken from the point where they intersect in order to be applied to the low resolution model. Their disadvantage is that they do not take advantage of the graphics hardware. They allow the generation of normal maps in the object and tangent spaces.

Later to the publication of our methods about the generation of normal maps on the GPU appeared the method presented in [TI07]. Teixeira et al. presented a generation of normal maps based on the GPU that works similar to the methods presented in [nVi04a] [ATI02], by performing a ray-tracing from the simplified model to the detailed one in order to find and assign the nearest normals.

2.8 Conclusions

This chapter has presented a state-of-the-art review of this thesis. During the last few years, a lot of simplification work has been presented and we have analyzed the most relevant previous work.

First, we introduced some basic concepts in order to comprehend this work. Then, we classified the previous work on simplification methods according to their main criteria. Here, we have introduced the most commonly used simplification operations and error metric criteria. We also introduced methods dealing with CAD model simplification. Moreover, we studied new trends about parallel simplification methods and methods performed using the new advances and capabilities of the graphics hardware. Finally, we have presented some previous methods for generating normal maps.

Different solutions to simplification problems have been presented in previous work. However, there are still some points to be solved in the simplification of the models that are usually used in interactive applications.

There is no automatic and user-assisted method that takes the properties of meshes obtained after a CAD process into account. The existing simplification tools do not use the best error metrics and do not allow a total control of the level of simplification for each subobject and the whole object. Moreover, simplification methods in the literature do not consider the properties of this kind of mesh. This is why this process is usually made by hand by designers. This is a difficult and elaborate process. The control of the different levels of simplification in each subobject and an automatic process to maintain the total number of triangles could be needed. This would save the designers from having to do this complex process by hand. In this thesis we present a method to solve this problem (Chapter 3).

Moreover, the models usually used in interactive applications, such as games, virtual reality, teaching or interactive worlds, can have some geometric properties (like, for example, duplicated vertices) and additional attributes (like normals and textures) that must be taken into account in order to obtain simplified models with a similar appearance to the original ones. In this thesis, we present several techniques that consider these properties and attributes. There are a lot of simplification methods that do not consider texture preservation and, therefore, great visual distortions are obtained when the texture is applied to the simplified models. We present an error metric extension useful for considering the texture information in those methods that do not take it into account (Chapter 4) and a method based on the segmentation of the textures (Chapter 5). The results exposed in these chapters present a high improvement in the texture preservation for simplification methods.

The visual aspect of the simplified models is important. However, the time employed by these methods could also be an important factor. This is why we also bear in mind the latest advances in graphics hardware in order to speed up slow simplification methods, like the viewpoint-driven algorithms, and normal map generation methods. Times are exposed in Chapters 5 and 6. It can be observed that great temporal reductions are obtained.

Chapter 3

User-assisted Simplification Method for Triangle Meshes Preserving Boundaries

3.1 Introduction

Nowadays 3D scenes are commonly represented with a high degree of polygonal complexity and many of the objects used in the scenes are generated from computer-aided design tools. Thus, polygonal models converted from CAD models are usually composed of a great number of polygons. This conversion to polygonal meshes (like X3D) is usually performed in order to render and manage the models in interactive applications running in a network. The current available hardware cannot however always handle all this geometry in a realistic way. Therefore, in interactive applications the accuracy of the models and the time required to process them must be taken into account.

In recent years, different solutions have been developed for interactive applications. One of these is the simplification of the objects, which attempts to reduce their polygonal complexity, while maintaining the appearance of the final object as much as possible. Simplification methods allow the amount of geometry needed to represent an object to be reduced, trying to maintain the visual quality, which benefits the performance of the GPU. Therefore, simplification methods produce objects with less geometry than the original ones. Interactive applications need to render and manage 3D scenes with a realistic framerate. Thus, the use of simplified objects can help to achieve this, because of the reduction of the geometric information. Many articles about simplification techniques have appeared in the literature and some surveys of such works can be found in [CMS98] [Lue01], including simplification methods for CAD models [TBG09]. We can distinguish between different criteria for mesh simplification, like geometry-based and viewpoint-driven simplification methods. Many of them are based on geometric metrics to define the order of the simplification steps. These methods are relatively fast and usually offer simplifications with a good appearance. On the other hand, methods based on the user's point of view try to generate not only good geometric results, but also realistic results for the viewer by removing, for example, parts of the object that are not visible to the user. These methods are usually slower than methods based on the geometry. A simplification method makes use of a simplification operation and an error metric.

Complex shapes with several parts (subobjects) can be obtained from a design process. These CAD models are converted to polygonal meshes to be rendered and managed with a high framerate in interactive applications running in a network and subobjects are treated as submeshes. Different works for meshing objects can be found in the literature [LC87] [Lun91] [WSO03]. Simplification of polygonal meshes obtained from CAD models requires special attention. These meshes are usually composed of a great number of different submeshes that are not necessarily interconnected. This is a difference between this kind of mesh and other kind of mesh usually used in interactive applications, such as games. Therefore, if this characteristic is not taken into account in the simplification process, undesired artifacts could be obtained with meshes obtained from CAD models, such as holes or distortion between the subobjects. Moreover, users can demand simplifications at different levels of detail of the different subobjects of the model. That is, with different percentages of simplification, usually given as the relation between the number of faces in the simplified model and the original number of faces. For example, a wheel of a car will need more level of detail than other parts of a car, because it usually has a rounded shape in the original model and if it is very simplified a great distortion could be obtained on it.

Simplification methods do not always present simplifications that satisfy the user's requirements about the total number of triangles in the simplified model and the levels of detail of the different subobjects. For example, the user can demand a total number of triangles in the simplified model while some parts of the model are maintained or simplified to a specific percentage of simplification. Simplification process is not usually used for this purpose, because this process is usually done by hand by the designers. Moreover, simplification tools do not present an automatic process to work with meshes generated from CAD models with the possibility of simplifying the subobjects to different percentages of simplification. It is usually an elaborate process and these applications do not always use the best error metrics.

We present a user-assisted mesh simplification method applied to CAD models converted to triangle meshes. This method allows the different subobjects to be simplified at different levels of detail, avoiding the appearance of holes and preserving the boundaries between the subobjects. This way, the user can simplify the whole model and modify some parts, by simplifying more or by refining the desired subobjects. This can be performed while the total number of triangles in the simplified model is maintained. In the presented method any metric based on edge collapse operation can be used. Therefore, the best and new metrics can be integrated in the method. Textures and normals, that play an important role in the final aspect of the model, are also considered.

This is a user-assisted method, because the user can test simplifying some parts of the model or refining other simplified ones. The goal is to obtain a simplified version of the model that satisfies the user's requirements about the total number of triangles in the simplified model and the levels of detail of the different subobjects. Two different error metrics have been used in order to present the results: a geometry-based error metric (QSlim quadrics [GH97]) and a viewpoint-driven error metric (VMI [CSCF08]).

The main contributions of the presented method are:

- The different parts (subobjects) of the models can be simplified with different levels of detail. This is a user-assisted method because the user can simplify the whole model and then modify the desired subobjects (by simplifying more or by refining these parts). These modifications can be done while the total number of triangles is maintained. Relative simplifications can also be performed, by maintaining a subobject always to a percentage of simplification of any other.

- Boundaries between subobjects are preserved and no distortion between subobjects is produced in simplified models. The method prevents the appearance of holes between the different subobjects in simplified models.

Models do not only consist of geometry, like vertices, edges and faces. They also usually have associated properties that allow the model to present a more realistic appearance, like normals or texture coordinates. These properties are also taken into account in this method. We recalculate texture coordinates after each simplification step and normals after the whole simplification process. Moreover, any metric based on edge collapse operation can be used in the presented method. The user can also choose the kind of edge collapse operation to be performed.

3.2 Simplification method

The presented method receives a mesh obtained from a CAD model, generated by a meshing process. This mesh can be divided into different subobjects, treated as submeshes. The data structure of the input meshes accepted by this method is detailed in subsection 3.2.1. In order to simplify a mesh obtained from a CAD model without losing information about the model, we store the following information:

- Subobject information: the vertices, edges and faces that pertain to any particular subobject.
- Boundary information: we mark each edge that pertains to any boundary as a *boundary edge*.
- Connectivity: information about the neighborhood of each edge of the model.

This is the information that the method needs to work correctly. This way, the different subobjects of the model can be simplified to different percentages of simplification, while the coherence and connectivity between the subobjects are maintained.

The method is divided into a pre-process that merges the model, an iterative contraction phase and a post-process to split the model into its initial subobjects. Figure 3.1 shows a general scheme of the process. All these steps are detailed in subsection 3.2.2.

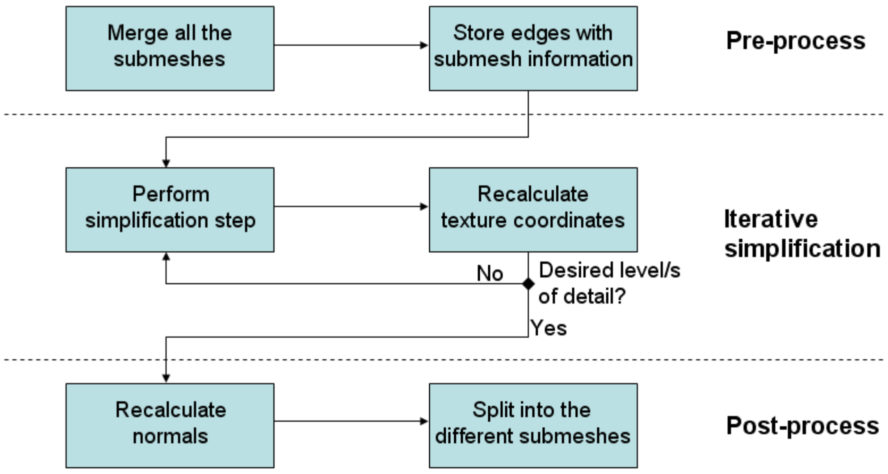

Merge all the submeshes
Store edges with submesh information
Pre-process
Perform simplification step
Recalculate texture coordinates
Iterative simplification
No
Desired level/s of detail?
Yes
Recalculate normals
Split into the different submeshes
Post-process

Figure 3.1: Scheme of the method

3.2.1 Terminology

This method works with meshes with the common structure of a mesh defined in Section 2.1. In the method presented here any error metric based on edge collapse operation can be used. This operation is applied at each simplification step and works by removing an edge and unifying its vertices. The final vertex can be located at a new position. However, the operation is called half edge collapse variant when one vertex is collapsed into another one, that is, no new vertices are created because we always refer to an existing vertex in order to compute the operation. As shown in Figure 3.2, an edge with vertices v_1 and v_2 collapses into vertex v_2, thus producing the simplified mesh.

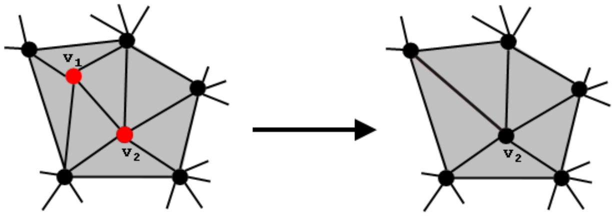


Figure 3.2: Example of half edge collapse operation

The presented method makes also use of split operation. A split operation is the inverse operation of an edge collapse. That is, from one vertex a new edge will be generated.

A percentage of simplification usually refers to the relation between the number of faces in the simplified model ($tfaces_{simpl}$) and the number of faces in the original model ($tfaces_{orig}$). And it is given by $\frac{tfaces_{simpl}}{tfaces_{orig}} * 100$.

3.2.2 Simplification steps

The method receives the model with the information defined in section 3.2.1 and the different levels of simplification of the subobjects required by the user as parameters. These levels of simplification represent the levels of detail of the subobjects. The user can demand different levels of detail to the different subobjects of the model. This will produce an automatic simplification, preserving boundaries. The user can try giving different levels of detail until a simplification that satisfies the user's requirements is obtained. The user-assisted capability of the method is explained in detail in section 3.2.7.

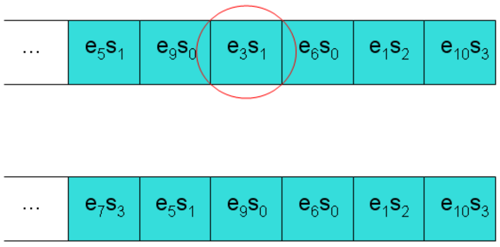
Pre-process

First of all, a pre-process step is performed, which merges all the submeshes in a single virtual mesh, and stores all the information about the original connectivity of the meshes. This is a virtual mesh because it is a temporal mesh that will be deleted after the simplified model is obtained. This step is based on the distance of the vertices of different meshes in the original object. Moreover, the edges between two neighboring submeshes are marked as boundaries.

Iterative simplification

Next, the mesh is simplified. To do so, we store the edges in their simplification order (Figure 3.3). This order is obtained by the associated cost given by the metric. Every stored edge will also have an associated identifier of subobject to know to which subobject pertains the edge. And for every subobject of the model we store the number of edges in both the original and the simplified version. This way, when we have to simplify a subobject we collapse the first edge with the identifier of the subobject. And we repeat this process until the desired level of detail of the subobject is obtained. Thus, we simplify each subobject to its associated level of detail. Moreover, after each simplification step is performed, the texture coordinates of the affected vertices are recalculated by a linear interpolation.

We have to take into account that the simplification of a submesh may affect the neighboring submeshes. That is, if edge collapse operations are produced near a boundary of a submesh, the edges of the boundaries will be moved and consequently the neighboring submeshes will also need to be moved in order to avoid generating a hole. But this will only be produced when the demanded level of detail is very low, because we perform the algorithm explained in section 3.2.4 to preserve the boundaries. Consequently, these edges will be collapsed in the last steps of the simplification process.



Number of edges		
Subobject ID	Original model	Simplified model
0	8345	8345
1	6216	6215
2	12076	12076
3	4114	4114

Figure 3.3: Example of simplification step. A collapse of an edge of the subobject 1 is produced

The simplification of the subobjects at different levels of detail also offer the possibility to the user of maintaining a relative simplification, that is, to maintain some subobjects simplified to a multiple of the level of detail of other subobjects.

The user can also require a total number of triangles (by giving a single level of detail for the whole model), and then modifying the levels of detail of the different subobjects. The total number of triangles will automatically be preserved. This is explained in detail in section 3.2.7.

Post-process

After simplifying the mesh, we recalculate all the normals of the model. Here we have to take into consideration that, depending on the shape of the different parts of the model, vertex normals or face normals will be needed. Therefore, depending on the angle of the affected faces, the normals are calculated per vertex or per face.

Finally, we retrieve the simplified mesh and split it into different submeshes, using the interconnectivity information about the original submeshes that was stored in the pre-process step.

3.2.3 Decimation computation

This method sets a decimation factor for each edge. The decimation cost reflects how much the appearance of the object will change, so that the edges with lowest values are the first to be collapsed. The decimation cost computation depends on the error metric used.

For each edge contraction $(v_1, v_2) \rightarrow v$ the following steps are performed:

- Collapse the edge that contains the vertices v_1 and v_2.
- Remove all triangles that share the edge.
- Remap all triangles shared by v_1 to v_2.
- Recalculate the cost (decimation coefficient) for the vertex based on the new connectivity information.

3.2.4 Boundary preservation

Using the information stored in the pre-process step, the implemented algorithm allows us to preserve the boundaries between the different subobjects. This technique is useful for any manifold model with boundaries. There are different works in the literature that preserve boundaries in different fields [SSGH01] [FCS02]. However, our method maintains the boundaries between the subobjects, allowing the user-assisted simplification of the different subobjects.

The method marks an edge in the model as a boundary when it only has one associated face. Two edges of different subobjects form a boundary between these subobjects, when the distance between their corresponding vertices is lower than a threshold. When an edge with a vertex in a boundary has to be remapped, the other vertex will overlap the first one. To do this, an attribute that indicates whether the vertex pertains to a boundary or is an internal vertex is stored in the data structure of the vertices. By so doing, the original shape of the boundary remains unchanged for as long as it is possible. A general scheme to this approach can be seen in Algorithm 1.

The user can select what kind of edge collapse will be performed, depending on the user's requirements and limitations. As we explain in Section 3.2.2 we store the simplification sequence in order to be able to refine some simplified parts of the model. Thus, for example, if the user wants to store less information about the changes produced in the simplification process, half edge collapse operation will be applied in order to not create new vertex positions. But if the user wants optimal positions for the collapsed edge, a full collapse operation will be performed. This operation will return a new vertex, located on a new position depending on the criterion of the collapse. In this case, the new vertex coordinates (or displacement relative to the originals) will be stored. Therefore, the information relative to all the affected vertices and faces must be stored (simplification sequence). With this information all the previous simplification steps can be recovered.

Our boundary preservation technique preserves the boundaries between subobjects. The boundaries of the whole model (edges of a subobject that are not connected to any other edges) are usually preserved by the metric of the simplification methods. For this consideration, this kind of edge could have assigned by the metric a high simplification cost in order to be maintained until the last simplification steps.

Algorithm 1 General algorithm for boundary preservation

```
vertex_to_move = None;
if mesh.boundaries.has_vertex(v1) then
    if mesh.boundaries.has_vertex(v2) then
        vertex_to_move = normal_placement(v1,v2);
    else
        vertex_to_move = v2;
    end if
else if mesh.boundaries.has_vertex(v2) then
    vertex_to_move = v1;
else
    vertex_to_move = normal_placement(v1,v2);
end if
```

If a vertex in the edge is classified as a boundary to be preserved, the algorithm showed in Algorithm 1 is executed. If both vertices are in a boundary (or different boundaries) we compute a normal vertex placement, that is, the final vertex will be located to the position given by the metric. This position is the one that minimizes the contraction cost. If only one vertex is in a boundary, the other vertex of the edge is the one to be moved. Otherwise, if no vertices are on a boundary, we also compute a normal vertex placement. This way, if the user wants to modify the level of detail of any subobject, the algorithm will perform the simplification operations, preserving automatically the boundaries between subobjects.

We must distinguish between this possibility and the option offered by some simplification methods, like QSlim [GH97], that allows borders to be preserved. Borders and boundaries are not the same concept. A border is an edge that pertains to only one face of the model. However, a boundary between two submeshes is an edge that pertains to two different faces, because we have merged the whole model in a pre-process before the simplification steps are performed.

The presented method preserves the boundaries, however the topology preservation depends on the metric used.

3.2.5 Texture coordinates recalculation

At each simplification step texture coordinates for each modified vertex need to be recalculated in order to maintain the mapping appearance. The new texture coordinate is calculated based on the displacement of the mapped vertex using a linear interpolation. An example of a modified triangle is shown in Figure 3.4, where the vertex P_2 is moved to the position Q.

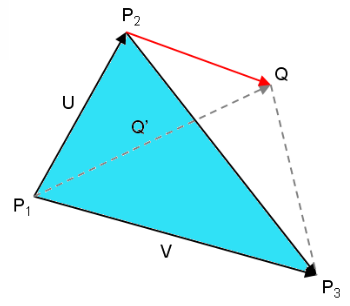


Figure 3.4: Example of a modified triangle

To obtain the offset that must be applied to the texture coordinates of the modified vertex, we propose the following system of equations:

$$Q'_x = \alpha U_x + \beta V_x + \gamma N_x \tag{3.1}$$
$$Q'_y = \alpha U_y + \beta V_y + \gamma N_y \tag{3.2}$$
$$Q'_z = \alpha U_z + \beta V_z + \gamma N_z \tag{3.3}$$

having $\vec{Q'} = \vec{Q} - \vec{P^1}, \vec{U} = \vec{P^2} - \vec{P^1}, \vec{V} = \vec{P^3} - \vec{P^1}$, where $\vec{P^1}, \vec{P^2}, \vec{P^3}$ are the three vertices of a modified triangle, $\vec{Q}$ is the new position of the modified vertex and $\vec{N}$ is the triangle normal. α, β and γ are the coordinates of $\vec{Q}$ expressed in the triangle coordinate system. They also express how much the modified vertex has been moved in triangle coordinate system units, so that they can be used to calculate the perturbed texture coordinate for the modified vertex. We use the following formula:

$$T_u^{res} = \alpha\left(T_u^2 - T_u^1\right) + \beta\left(T_u^3 - T_u^1\right) + T_u^1 \tag{3.4}$$

$$T_v^{res} = \alpha\left(T_v^2 - T_v^1\right) + \beta\left(T_v^3 - T_v^1\right) + T_v^1 \tag{3.5}$$

where $\vec{T_1}$, $\vec{T_2}$ and $\vec{T_3}$ are the original texture coordinates of the three vertices, $\vec{T^{res}}$ is the new texture coordinate for the modified vertex and α and β are the coefficients calculated from the equations 3.1, 3.2 and 3.3.

Note that we only use α and β because texture coordinates are two-dimensional vectors contained in the plane formed by the triangle. As γ is the displacement along the normal vector of the triangle, we do not need it.

In Figure 3.5 we can see an example of the use of this texture coordinate interpolation. It can be observed that using texture recalculation more accurate textures are obtained.

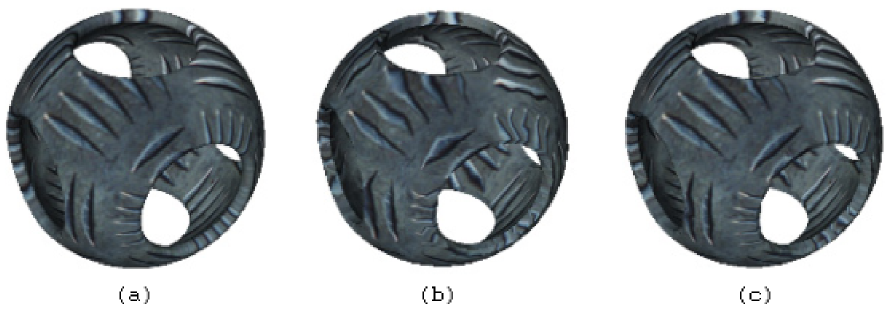


Figure 3.5: Original ball model (a), ball model simplified to 30% without texture coordinate interpolation (b), and ball model simplification to 30% with application of texture coordinate interpolation

This texture coordinates recalculation produce good results in the majority of cases. However, it could produce artifacts if the textures of the models contain various regions. Therefore, a work about region-based texture-preserving simplification is presented in Chapter 5.

3.2.6 Normals recalculation

After simplifying the model we recalculate all the normals of the model. Depending on the shape of the parts of the object, the normals will be calculated per face or per vertex. This is because some parts of the models need to use per-vertex normals (for example smooth and rounded parts) and other ones need to use per-face normals (for example corners). To do this, we perform the following algorithm:

- We expand the model, so that each face will have different indices of normals. Thus, we obtain the number of vertices as the number of faces multiplied by three.

- The face normal is assigned to each vertex.

- For each vertex we review all the other vertices of the model. If any vertex is located at the same spatial coordinate and the angle between their respective faces is lower than a threshold angle given by the user (we use 30 degrees) we add the normal of this vertex and the indices of normals will be the same.

An example of the use of normals recalculation is shown in Figure 3.6. It shows that a combination of per-face and per-vertex normals produce the best results.

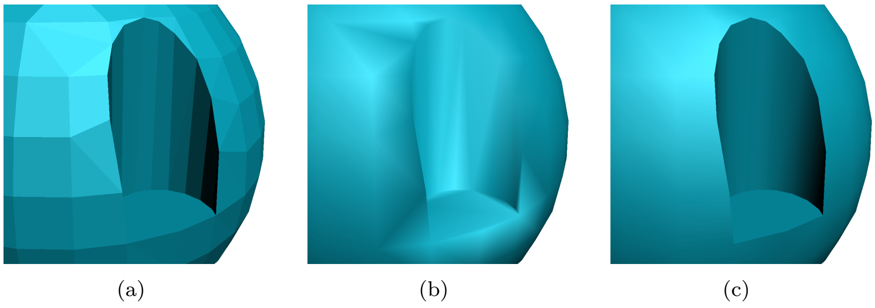


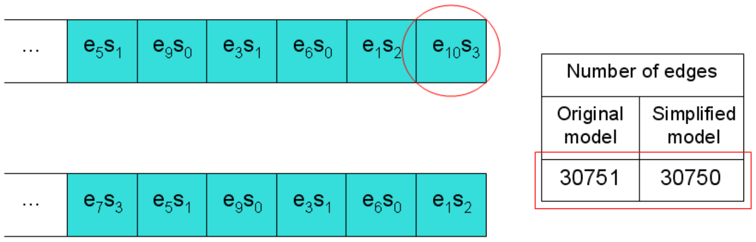
Figure 3.6: Part of a three-dimensional object with the per-face normals (a), with the per-vertex normals (b) and taking into account the angle between the faces (c)

3.2.7 User-assisted simplification

The presented method is user-assisted because the user can test simplifying different parts of the model. Simplifying the whole model with a unique level of detail does not always produce a simplification that satisfies the user's requirements about the total number of triangles in the simplified model and the levels of detail of the different subobjects. Our method gives the total control to the user for giving different levels of detail to the different subobjects. This will produce an automatic simplification that preserves the boundaries between the subobjects, maintaining the coherence in the model. The user can change the different levels of details until a desired simplification of the model is obtained.

A possible strategy for the user to simplify the model could be the next: if the user wants a maximum number of triangles in the simplified model to be rendered, it can be limited by giving a level of detail for the whole model. The model will be simplified to this level of detail. After that, the user can change the level of detail of any subobject of the model. The user can give more level of detail to a simplified subobject or simplify it more. This can be done until a simplified model that satisfies the user's requirements about the number of triangles in the subobjects is obtained. If a level of detail for the whole model was given, a change in a local level of detail will automatically affect to the other local levels of detail in order to maintain the final number of triangles. This will not affect to the subobjects that have an associated level of detail introduced by the user, because these subobjects have exactly the number of polygons that the user wants for them. Examples are discussed in Section 3.3.

In order to simplify the whole model with a single level of detail preventing the appearance of holes between the different subobjects and preserving the boundaries, we follow the same steps explained in section 3.2.2, but without considering the subobject information associated to the edges (Figure 3.7). We store the edges in the order of their simplification costs given by the error metric. We then simplify normally with the simplification method extracting the edges in a sequential order.



Number of edges	
Original model	Simplified model
30751	30750

Figure 3.7: Example of simplification step. A collapse of an edge is produced

When the user demands to simplify more a specific subobject, this simplification will be performed normally with the subobject simplification explained in section 3.2.2. After this, the algorithm will add more faces to the simplified model in order to maintain the required level of detail for the whole model. To do that, the stored simplification sequence is used. Hence, the algorithm will perform vertex split operations with the collapsed edges of the subobjects that have not a required level of detail introduced by the user. With this operation the vertex obtained after a collapse will produce the edge again and the affected faces will be retriangulized. This is performed until the total level of detail is obtained. A pseudo-code of this algorithm is presented in Figure 2. In this Figure we can see that the vertex split operations are performed in order to obtain the desired total level of detail. The extraction of a simplification step from the simplification sequence will return the last collapse performed. Thus, the vertex split operations will undo the last collapse operations. The vertex split operation is not performed if the extracted simplification step pertains to a submesh that has an associated level of detail introduced by the user, because this submesh has exactly the number of polygons that the user wants for it.

On the other hand, when the user refines a simplified subobject, this refinement will produce split operations in this subobject. For this purpose, the simplification sequence will be used. This way, the last collapse operations stored with the information of this subobject will be undone, generating the edges that had been collapsed. This is performed until the desired level of detail in the subobject is obtained. And more edge collapse operations from other subobjects that do not have an associated local level of detail must be performed in order to maintain the desired total number of triangles in the final object. This is also considered in Algorithm 2.

If the user requires leaving a subobject intact, he only has to assign the corresponding level of detail of this subobject to 100%. Therefore, the simplification will not affect to this particular subobject.

Algorithm 2 Pseudo-code for reobtaining the desired total number of triangles after a modification in any submesh

```
function OBTAIN_TOTALLEVELOFDETAIL(Model m, LOD total_lod, SIMPL_SEQUENCE simpl_seq)
    if $totalLevelofDetail(Model) < total\_lod$ then
        while $gobalLevelofDetail(Model) \neq total\_lod$ do
            simpl_step= extract_last_simpl_step(simpl_seq)
            if simpl_step.submesh has not local level of detail assigned by the user then
                performSplit(simpl_step.info, m)
            end if
        end while
    else if $totalLevelofDetail(Model) > total\_lod$ then
        while $gobalLevelofDetail(Model) \neq total\_lod$ do
            edge= extract edge of subobject without associated level of detail
            performCollapse(edge, m)
        end while
    end if
end function
```

An example is shown in Figures 3.8, 3.9 and 3.10. Figure 3.8 shows the levels of detail of a model simplified with a total 50% percentage. The edges have been collapsed in the order of their simplification costs, without taking the subobject information into account. In Figure 3.9 we can see that the first subobject has been simplified to 25%. The levels of detail of the other subobjects have to be modified in order to maintain the total number of triangles. Therefore, the other subobjects have been refined with the information stored in the simplification sequence. On the other hand, in Figure 3.10 the first subobject has been refined to its 100% of its original geometry. Thus, the other subobjects have been simplified in order to maintain the total number of triangles.

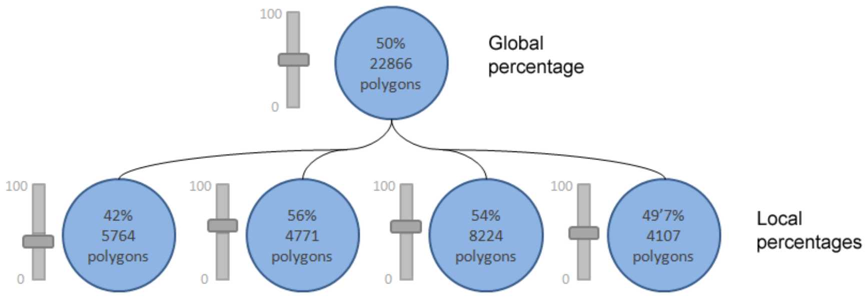


Figure 3.8: Levels of detail of a model simplified to 50% of its original geometry

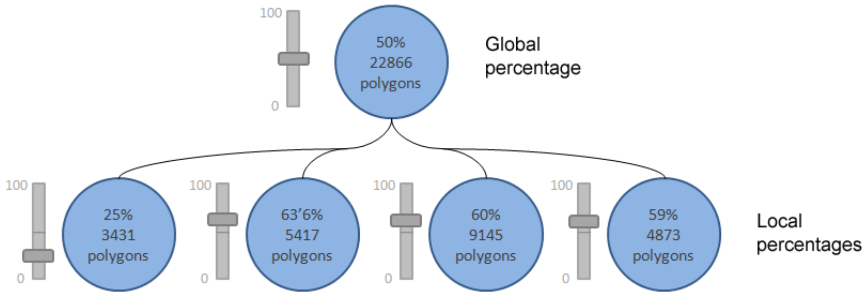


Figure 3.9: Levels of detail of a model simplified to 50% of its original geometry, simplifying the first subobject to 25%

3.3 Results

We have tested several models with our method and some examples are exposed here. The different characteristics of the method are divided into subsections. The models used to present the results are introduced in Figure 3.11 and their geometric details given in Table 3.1. The subobjects of the models are represented with different colors. All the percentages of simplification are relative to the number of triangles of the models.

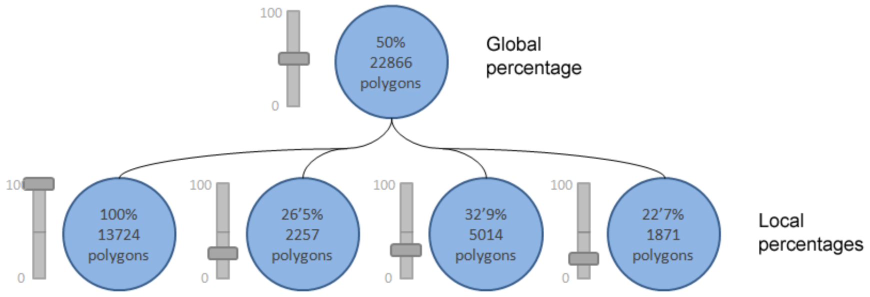


Figure 3.10: Levels of detail of a model simplified to 50% of its original geometry, maintaining the level of detail of the first subobject at 100%

Table 3.1: Details of the models presented in the results subsections

Model	Subfigure	Triangles	Vertices	Subobjects
Casino	3.11(a)	315	357	10
Yacht	3.11(b)	532	352	11
Speaker	3.11(c)	5 202	5,276	184
X-wing	3.11(d)	8,716	10,026	1,004
Cellular	3.11(e)	11,728	14,259	1,541
Ball	3.11(f)	12,264	8 272	22
Racing car	3.11(g)	42,964	41,620	1,579
Toy car	3.11(h)	59,589	56,442	2,007

3.3.1 Simplification of subobjects

The presented method allows subobjects to be simplified at different levels of detail. This way the user can simplify more some parts of a simplified model or refine other ones by giving them more level of detail. Some examples are shown in this subsection. In Figure 3.12 we can see the simplification of a subobject of the speaker model. Figures 3.13 and 3.14 show a refinement of some parts of the model in a simplified version. Figure 3.13 shows the racing car model simplified to 10% and a refinement of the subobjects that make up the wheel at different levels of detail. And Figure 3.14 shows the toy car model simplified to 10% and a refinement of the subobjects that make up the steering wheel at different levels of detail. In Figure 3.15 we can see a wheel of a model refined with different levels of detail. In Subfigure 3.15a the original model is shown, in Subfigure 3.15b the whole model has been simplified to 10% of its original geometry, and in Subfigure 3.15c and Subfigure 3.15d a refinement of the wheel has been performed. In Subfigure 3.15c the wheel is recovered at 50% and in Subfigure 3.15d at 100% of its original geometry. It can be appreciated that the boundaries between the simplified parts are preserved. In Figure 3.16

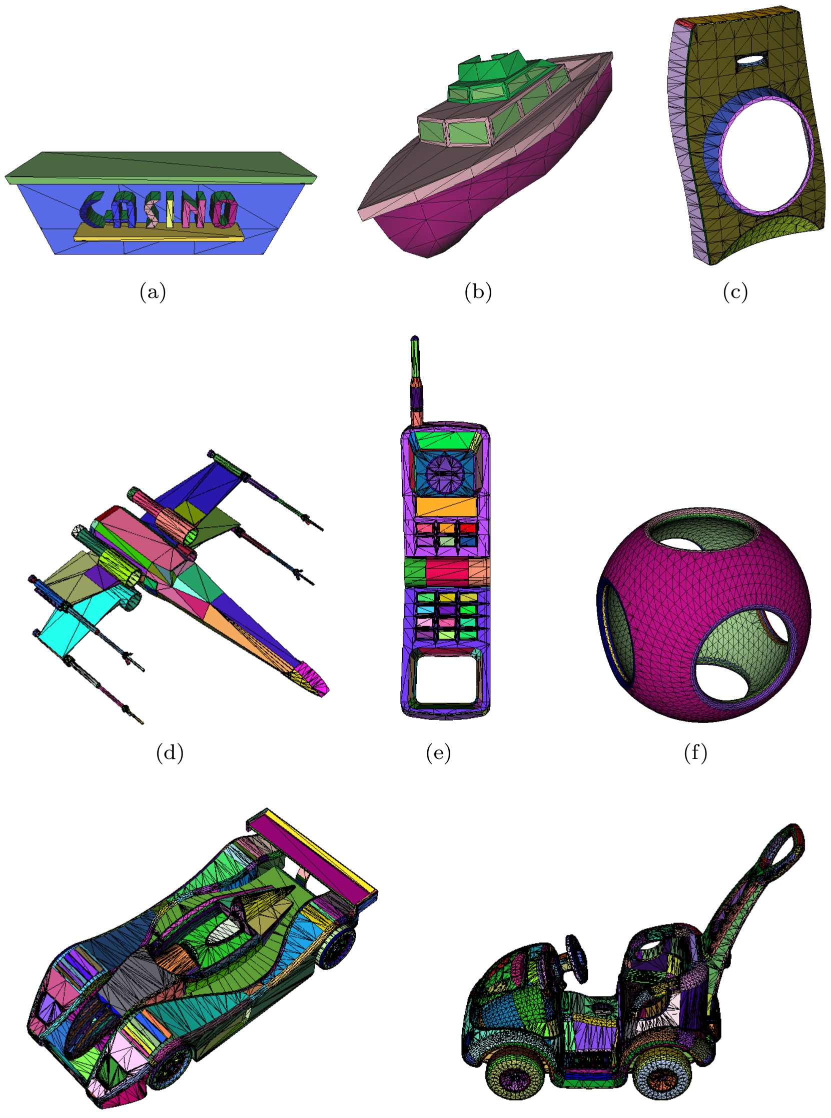


Figure 3.11: Models presented in the results subsections

more than one subobject of the ball model are simplified at different levels of detail. And Figure 3.17 shows a relative simplification in the ball model, where a subobject is maintained to the half level of detail than other one.

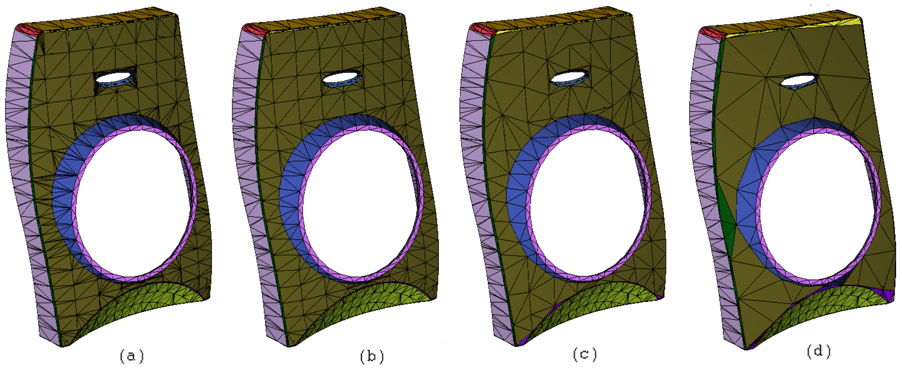


Figure 3.12: Original speaker model (a) and simplifications of the front subobject represented by a brown color to 50% (b), 30% (c) and 10% (d)

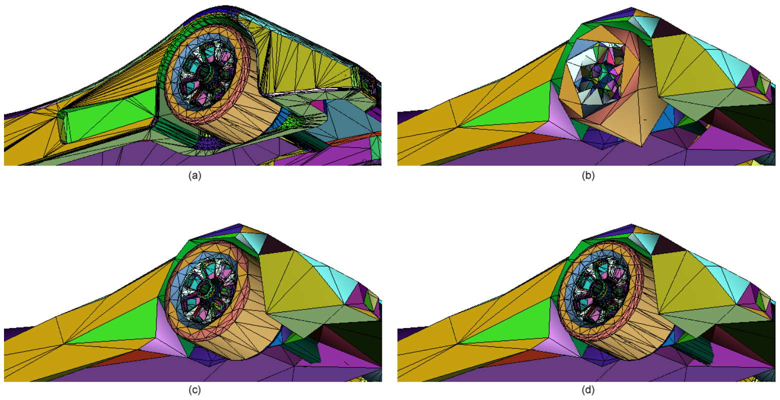


Figure 3.13: Original racing car model (a), model simplification to 10% (b) and refinements of the subobjects that make up the wheel at 50% (c) and 100% (d)

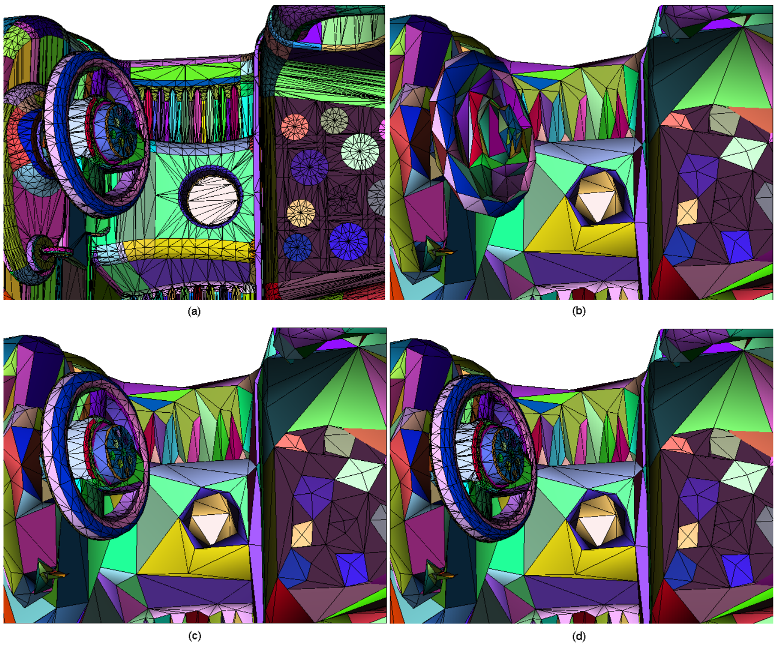


Figure 3.14: Original toy car model (a), model simplification to 10% (b) and refinements of the subobjects that make up the steering wheel at 50% (c) and 100% (d)

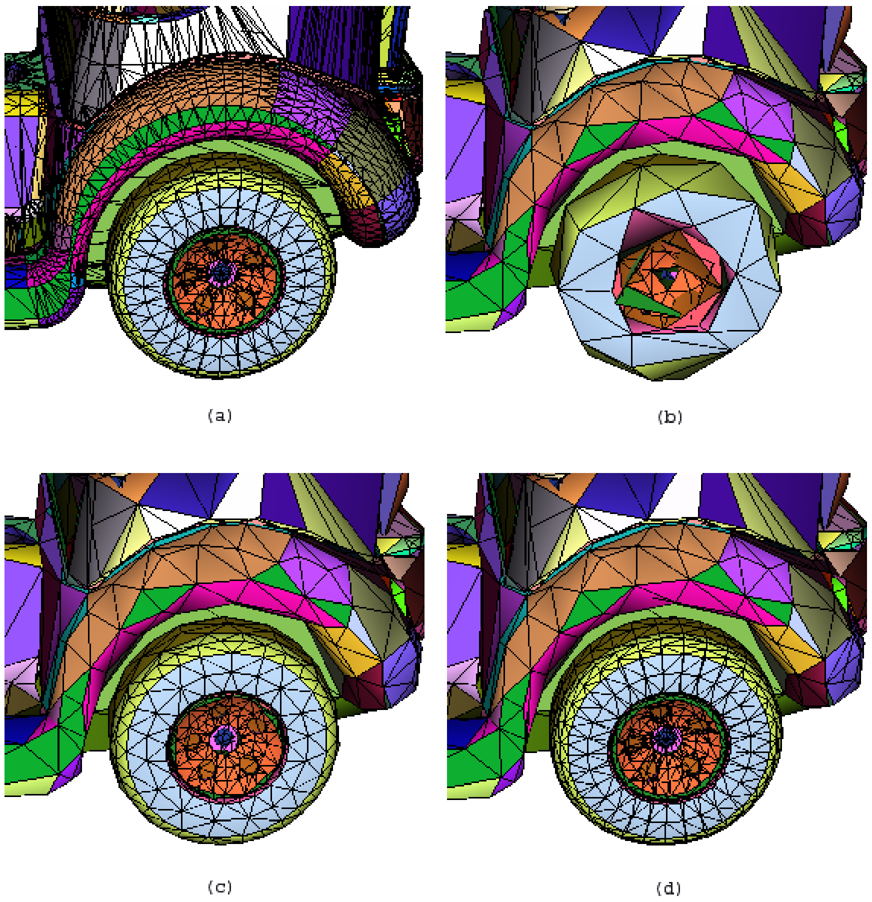


Figure 3.15: Example of subobject simplification with our method. In (a) the original model is shown. In (b) the whole model is simplified to 10%. And in (c) and (d) the wheel of the model is refined to 50% and 100% respectively of its original geometry

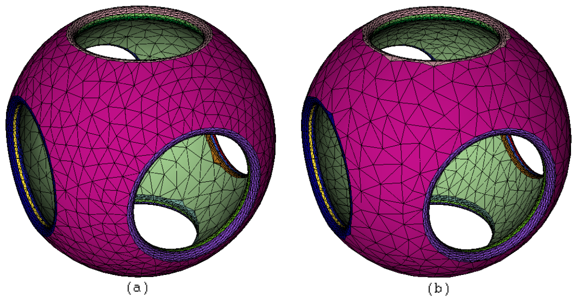


Figure 3.16: Simplifications of the subobjects represented by the pink and green colors. In (a) the pink subobject is simplified to 50% and the green subobject is simplified to 30%. In (b) the pink subobject is simplified at 30% and the green subobject is simplified to 50%

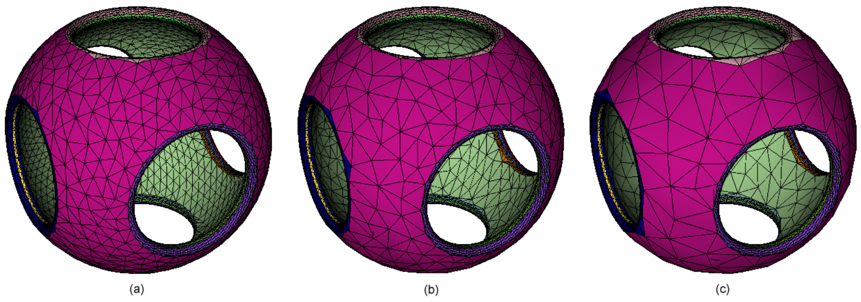


Figure 3.17: Relative simplification. The subobject represented by the green color is simplified to 80%, 40% and 20%, while the subobject represented by pink color is always maintained at the half level of detail, that is, 40%, 20% and 10%, respectively

3.3.2 Boundary preservation

In this subsection we justify why we unify the model and take boundaries into account. If the model is not unified and boundaries are not taken into account holes and distortion can be produced. We present some examples of simplifying the models by applying QSlim [GH97] without unifying the models (that is, generating a model with an unique subobject) and the simplifications applying our method (Figures 3.18 and 3.19). We can appreciate that the boundaries between subobjects have been preserved with our method. Therefore, the unification of the models is done in order to avoid the appearance of holes between the subobjects.

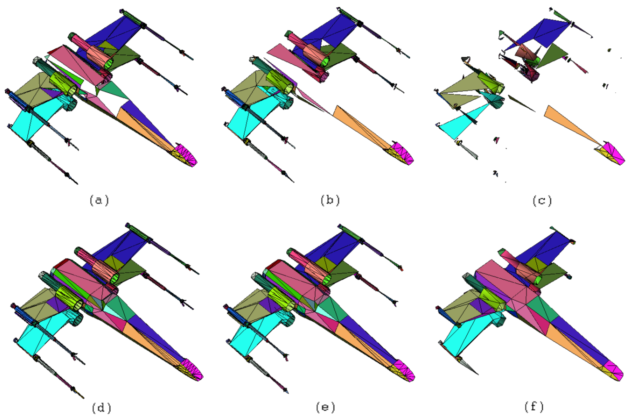


Figure 3.18: Simplifications of X-wing model with QSlim to 50% (a), 30% (b) and 10% (c) and with our method to 50% (d), 30% (e) and 10% (f)

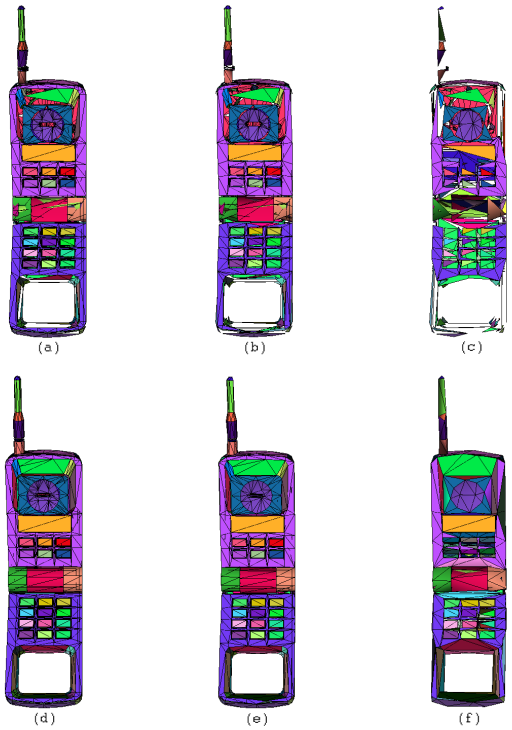


Figure 3.19: Simplifications of cellular model with QSlim to 50% (a), 30% (b) and 10% (c) and with our method to 50% (d), 30% (e) and 10% (f)

3.3.3 Texture coordinates recalculation

Textures play an important role in the final appearance of a simplified model. Therefore, a recalculation of the texture coordinates of vertices that have been affected after a simplification step is performed. An example of this recalculation was presented in Figure 3.5. In this figure we can see a simplification applying the texture coordinates recalculation and a simplification without applying it. It can be observed that the textures are more accurate with the texture coordinate interpolation.

3.3.4 Normals recalculation

Normals are also an important attribute to be taken into account in order to achieve a final appearance of the simplified model that is as similar as possible to the original one. We have exposed a post-process method that considers the angle between faces in order to calculate per-vertex or per-face normals. In Figure 3.20 we can see two points of view of an object in which the normals have been recalculated. In the parts of the model with angles between faces higher than the threshold, normals have been calculated per face, as for example in corners or the back lights of the model. And in the parts of the model with low angles between faces normals have been calculated per vertex, as for example in the wheels.

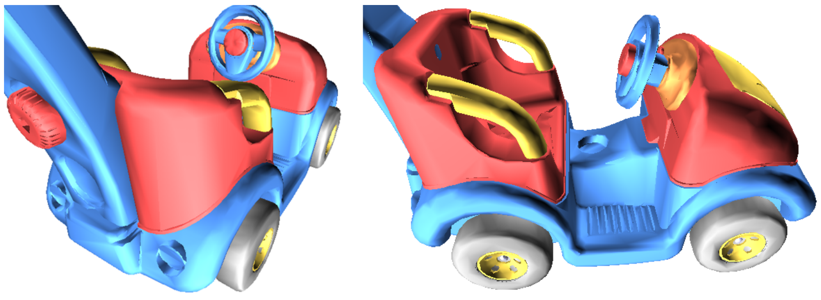


Figure 3.20: Toy car model simplified to 30%, in which the normals have been recalculated

3.3.5 Using different error metrics

In the presented method any metric based on edge collapse operation can be used. Here two different error metrics have been used in order to present the results: a geometry-based error metric (QSlim quadrics [GH97]) and a viewpoint-driven error metric (VMI [CSCF08]). The viewpoint-driven error metrics usually produce better visual results for the viewer, but are slower than the geometric-based metrics, due to the number of renders to be performed.

Figures 3.21 and 3.22 show simplifications of the casino model and yacht models using both metrics. We can observe that the simplified models maintain the subobject information and the boundaries between them are preserved. Table 3.2 shows some statistics of these simplifications.

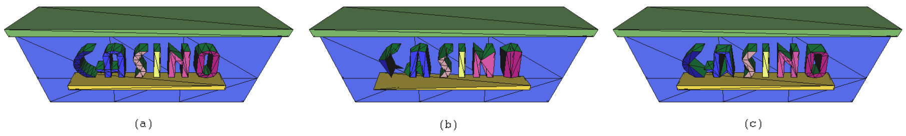


Figure 3.21: Original casino model (a) and simplifications of the model to 60% with our method using QSlim quadrics (b) and VMI (c)

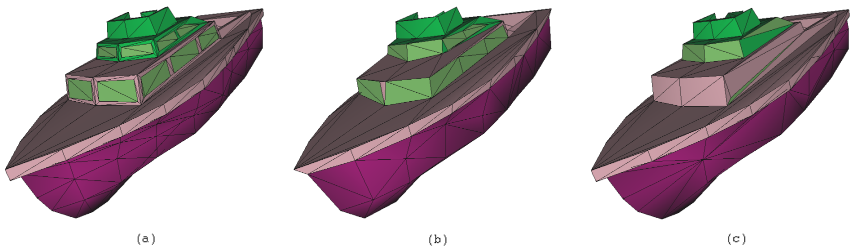


Figure 3.22: Original yacht model (a) and simplifications of the model to 50% with our method using QSlim quadrics (b) and VMI (c)

Table 3.2: Models presented in the results subsections

Model	Triangles removed	Error metric	Time (sec)
Casino	189	QSlim	0.016
		VMI	5.062
Yacht	266	QSlim	0.032
		VMI	7.25

3.3.6 Temporal cost

The temporal cost introduced by this method during the simplification is negligible. We state this because:

- The possibility of simplifying the different submeshes of the model with different levels of detail does not introduce any temporal cost. The only difference is the order in which the edges are extracted.

- The computational and temporal cost introduced to maintain the boundaries is only the cost introduced by conditional operations.

- The computational and temporal cost introduced to recalculate the texture coordinates is only the cost introduced by a linear interpolation.

3.4 Conclusions

A user-assisted simplification method for triangle meshes generated from CAD models has been presented. This kind of mesh normally consists of different and independent subobjects. These subobjects are not necessarily interconnected. Thus, simplifications produced by some of the existing methods would usually produce distortion and holes between the subobjects of the models. Simplification methods do not always present simplifications that satisfy the users' requirements about the total number of triangles in the simplified model and the levels of detail of the different subobjects. Users can demand different levels of detail in each submesh. Therefore, some parts of the models have to be modified. Simplification process is not usually used for this purpose, because this process is done by hand by the designers. Moreover, simplification tools do not offer an automatic process to manage this kind of mesh with the possibility of simplifying the subobjects to different percentages of simplification.

The presented method allows the different subobjects of the meshes obtained from CAD models to be simplified with different levels of detail. It also presents the possibility of simplifying whole models with a single level of detail. Therefore, the user can simplify the whole model and then modify some parts, by simplifying more or by refining the desired subobjects. This can be performed while the total number of triangles is maintained. This way, the user can try with different levels of detail of the subobjects to find a simplification that satisfies the user's requirements, while the total number of triangles in the final object required by the user is maintained. Moreover, simplifications with proportions between the levels of detail of different subobjects can be performed. Boundaries are always preserved and no holes are produced in the simplified objects. This method also takes into account the information of texture coordinates and normals, which are necessary to obtain an accurate visual simplification. Any metric based on edge collapse operation and any kind of edge collapse operation can be used in this method.

This method has been implemented for metrics based on edge collapse operation. Hence, a possible option for future work is to extend this work in order to be able to use any simplification operation.

We can see in the results set out in this section how our method accurately preserves the boundaries and no holes are produced when we simplify either the whole model or independent subobjects with different levels of details. We also present the final appearance of simplified models with texture coordinates and normals recalculation.

Moreover, the temporal cost introduced by this extension during simplification can be considered to be negligible in comparison to the total simplification time.

CHAPTER 4

A Texture-based Metric Extension for Simplification Methods

4.1 Introduction

A simplification process produces objects with less geometry than the original ones, attempting to preserve the appearance of the original objects. However, the final geometry is not the only important factor in objects obtained after a simplification process. Models usually consist of additional attributes to their geometry. Interactive applications need to present the simplified models with a good aspect. And textures play an important role on their appearance. Therefore, good textured models must be presented in the scene.

There are a lot of simplification methods that do not consider texture information in the error metric. If texture information is not taken into account in the error metric, the order of the simplification steps will be established without considering the texture of the model. Therefore, simplified objects can present a great distortion when the texture is applied. This will produce unsuitable simplified models to be shown in interactive applications.

Moreover, meshes used for real-time applications are usually composed of submeshes 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.

We present a texture-based metric extension useful for those simplification methods that do not consider the texture information in their error metric. Moreover, a common geometric property of meshes usually used in interactive applications (like the use of duplicated vertices at the same spatial coordinates) is considered in this work. Moreover, the consideration of this work for preserving other attributes additional to the geometry of the models, such as normals, is also presented.

The presented metric extension avoids the early collapse of edges that crashes with no uniform regions of the texture. The detection of these regions is performed by an edge detector method based on Canny [Can83] [Can86]. For testing the new error metric extension, the simplification method presented in [CSCF08], based on edge collapses, has been used. It can be observed that simplified models with this new metric present more realistic results than before. This metric extension produces a modification in the order of the edge collapses. It is very useful for multiresolution models. The computational cost of this metric extension is negligible compared with the simplification time.

The error metric extension presented here is based on the different information given by the texture image. It tries to distinguish the borders in the texture and uses this information to modify the order of the collapses. This error metric extension has been tested with the simplification method presented in [CSCF08], that is based on edge collapse operations. This method did not originally take into account textured models. Thus, a simplified model usually produced a great distortion on the texture. Trying to improve that, we have extended the method with the presented error metric, preserving the textures. This metric produces a later simplification of the regions of the model that contain abrupt changes of the texture.

This extension is very useful for the generation of simplification sequences of multiresolution models, commonly used in games. Multiresolution models present the possibility of been rendered in the scene in different levels of detail, depending on various factors such the distance of the object to the viewer, the relative importance of the object in the scene, etc.

4.1.1 Motivation

It is very important to use a simplification that produces good textured simplified objects, because of the visual importance of the texture.

There are a lot of simplification methods for three-dimensional models, but only a few of them consider the texture information in its error metric [GH98] [Hop99] [XSX05]. Therefore, the methods which do not consider the texture information usually present simplified models with distorted textures. Usually, the methods which consider this information use a specific metric only valid for them.

Here we present a solution to this problem. So, the presented error metric extension is useful for taking texture information into account in those methods that do not consider it in their error metric.

4.2 Error metric extension

We have developed a new texture-based error metric extension for simplification algorithms which use edge collapse operation. It is based on the shape of the texture, in order to present a more realistic aspect of the simplified model when the texture is applied. Simplification methods which use edge collapses assign a cost to each edge that determines the order of the collapses. Depending on the borders of the texture, we modify the cost of each edge in order to penalize those edges that intersect these borders. We explain the steps performed to do that.

First of all we detect the borders of the texture. This is done by an edge detector method based on Canny [Can83] [Can86]. This edge detector works in a multi-stage process. A Gaussian convolution is applied in order to smooth the texture. Then, regions of the texture with high first spatial derivatives are highlighted applying a simple 2D first derivative operator. Edges give rise to ridges in the gradient magnitude image. Then, non-maximal-suppression is applied, that is, all pixels that are not actually on the ridge top are set to zero. These pixels would be drawn as a thin line in the output. Two thresholds are used to apply hysteresis, allowing the continuity of noisy edges.

Depending on different factors, the quantity and thickness of output borders can change:

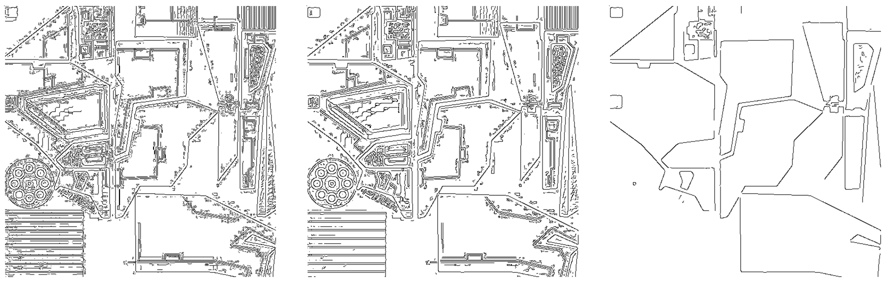
- The size of the gaussian filter: Depending on how much the texture is smoothed by the Gaussian convolution, fewer clear lines would be marked as borders or not.

- Thresholds: the low and high thresholds would give to the algorithm what we think is relevant information or not.

Figure 4.1 shows the difference obtaining the borders of a texture by using different values for the parameters.

Once edges detection has been performed, we have as result an image with these borders. We store in a matrix the values (white or black) of each pixel of this image. Now, we have in a data structure the shape of the borders and we can work with them. If we applied this image to the 3D model, we could see which edges intersect with borders (see Figure 4.2). So, if an edge that intersects any of these borders is collapsed a great distortion on the texture would be obtained. Therefore, these edges must have a high cost of collapse.

We have to know which edges cross any border. In order to do that, we use the texels of each edge of the model. So, we know how each edge is located in the texture. With a few simple 2D operations we can determine whether this edge rendered on the texture crosses any border. Let E be the set of these



(a) (b) (c)

Figure 4.1: Borders of the texture of the robot model using the following parameters: (a) size of gaussian filter = 0.1, low threshold = 0.1, high threshold = 0.2; (b) size of gaussian filter = 0.9, low threshold = 0.1, high threshold = 0.2; (c) size of gaussian filter = 0.9, low threshold = 0.8, high threshold = 0.9

affected edges. Figure 4.2 shows the sphere model textured. In this model the edges that have a part in one black region and other part in one white region would pertain to E.

We store all the active edges in a heap, where each edge has an associated cost. Therefore, edges with lower cost will be first collapsed. We then modify the previous costs of the edges that pertain to E, to be lately collapsed (functions of Algorithm 3). Function $getE(Texture, Model)$ returns the set of the edges that intersect with any border of the texture. It is called in the beginning of the process. And the cost of each edge is given by the function $computeEdgeCost(e, E)$.

The relative area of a region of the model is the area of this region divided into the sum of the areas of all the triangles of the model. We add to the previous cost of each edge the relative area of the triangles that contain the edge that we are analyzing.

So, we define the total area of the model as the sum of the areas of all the triangles of the model (4.1). Thus, for one specific edge the additional cost will be the sum of the areas of the triangles which contain this edge divided by the total area of the model (4.2). This way, the area of each triangle plays an important role in the order of the edge collapses, causing that the triangles with lower areas will be first removed than the triangles with similar previous cost and higher areas (if the model is manifold, in an edge collapse one or two triangles are removed). Therefore, the cost for each edge $e \in E$ (c_F) is calculated as follows:

$$A_T = \sum_{i=1}^{n} a_i \tag{4.1}$$

being n the number of triangles of the model and a_i the actual area of the triangle i.

$$c_F = c_e + \frac{\sum_{i=1}^{t} a_i}{A_T} \tag{4.2}$$

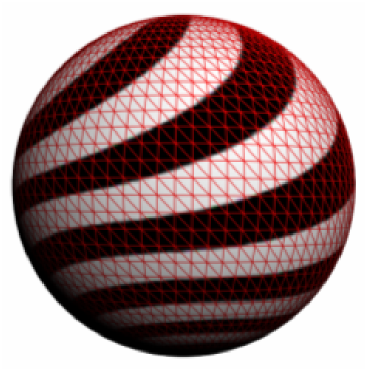


Figure 4.2: A textured sphere model. It can be seen that edges intersect with borders

This extension metric is also useful for maintaining other attributes additional to the geometry of the model, like for example the normals. The idea is the same. The only difference is the consideration of other images in order to obtain the borders to assign the cost associated to each edge. In the particular case of the normals of the model, the normal map will be considered. An example of the border detection in normal maps is shown in Section 4.4.2.

4.2.1 Justification of the metric

The extension is based on texture information and it is clear which edges have to be penalized, but we have to know how to change their collapse cost. We have chosen as error extension the relative area of the triangles that contain the collapsed edge, because the more area has a triangle the more noticeable will be its removal in the simplified object.

Another possible error metric extension that we have considered was the relative area of these triangles in the texture domain, because we are taking into account the texture information in this metric. The texture coordinates of an object may not be uniformly distributed. So, small triangles in the 3D space may be parameterized with a large triangle in the texture domain. An example is shown in Figure 4.3, where the eye of the ninja model is almost such large as any other part of the body. Therefore, good simplification results by using this possibility will depend on the importance of the sizes in the texture given by the designers.

Algorithm 3 Pseudo-code of computing the cost of an edge

```
function getE(Texture, Model)
    E = Ø
    M = EdgeDetection(Texture)
    for each e of the Model do
        if e crashes with a border of M then
            Insert(e, E)
        end if
    end for
    Return E
end function

function computeTextureError(e, E)
    if e in E then
        t = getTriangles(e)
        Ct = relativeArea(t)
    else
        Ct = 0
    end if
    Return Ct
end function

function computeEdgeCost(e, E)
    Ce = computeEdgeCollapseError(e)
    Ct = computeTextureError(e, E)
    Return Ce+Ct
end function
```

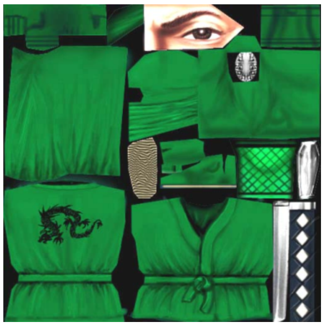


Figure 4.3: Texture of the ninja model

4.3 Submeshes consideration

We also consider a common geometric property of meshes used in interactive applications. Meshes used in interactive applications 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 can make use of three names: 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 4.4.

Figure 4.4 presents a simple example of a simplification step. This figure shows a part of a mesh composed by different submeshes. 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 4.4 - 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 [RC04]). Other vertices are simply translated to cap the holes resulting in face elimination. A pseudo-code is presented in Algorithm 4.

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

4.4 Results

Several models have been tested with the new error metric and it can be observed in the results showed in this section that the texture in the simplified models is more accurate to the original models than before. Moreover, we have also tested with the border detection in normal maps in order to preserve the normals of a model.

Therefore, in Section 4.4.1 a study of the use of the error metric extension with different models is presented. Section 4.4.2 shows that this error metric

Algorithm 4 Pseudo-code for edge contraction

```
while $EdgeList \neq \emptyset$ do
    realEdge=extract(EdgeList)
    addSimplifList(realEdge)
    if Twin(realEdge, EdgeList) then
        twinEdge= Twin (realEdge, EdgeList)
        addSimplifList(twinEdge)
    end if
    while disconnectedVert(realEdge, EdgeList) do
        fakeVertex= disconnectedVert(realEdge, EdgeList)
        fakeEdge= FakeEdge(realEdge, fakeVertex)
        calculateAttributes(fakeVertex)
        addVertexList(fakeVertex)
    end while
end while
```

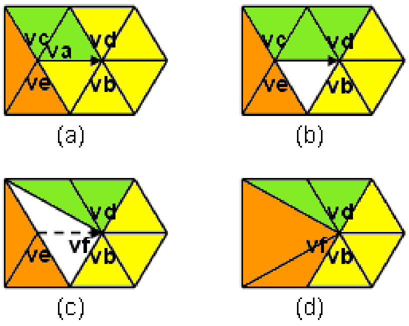


Figure 4.4: The edge collapse between va and vb (real edge) forces the collapses of two other edges: the one formed by vc and vd (twin edge) and the one formed by ve and vf (fake edge)

extension is also useful for preserving other attributes, such as the normals of the models.

4.4.1 Texture preservation

The number of edges of the simplified models remains unaltered, but the order of the simplification of these edges has been different. Now the edges that crash in the texture domain with any border obtained by the edge detector method have a higher error cost. Thus, those parts of the model that have fewer edges crashing borders are more simplified than before. The border detection process is performed as a pre-process. The borders detection time depends on the resolution of the texture, being a very fast process. Moreover, the computational cost performed by this metric at each simplification step is negligible compared with the simplification time.

The models have been simplified by the simplification method presented in [CSCF08], based on edge collapse operations. We have chosen the parameters of the edge detector method that we have considered appropriate to obtain the most relevant borders of the texture. But giving other values to these parameters we would obtain more or fewer borders of the texture and, consequently, more or fewer edges that have to be reordered in the collapse order.

Next, some simplified models are shown. Figure 4.5 shows the original eye model. In Figure 4.6 the texture of this model and the borders of this texture are shown. In Figures 4.7 and 4.8 it can be seen the difference between applying the metric or not applying it in the eye model. Three levels of simplification are shown (75%, 50% and 25%). In Figure 4.10 it can be seen the texture of the ninja model and its borders detected by the edge detector method. Figures 4.11 and 4.12 show a 50% simplification of this model without applying the new metric and applying it. First the original model is shown (left), then the simplified model without applying the metric (centre) and then the simplified model applying the metric (right). In Figure 4.14 the texture of the robot model and its borders are shown. In Figures 4.15 and 4.16 it can be seen the difference between applying the metric with the robot model and not applying it in a simplification of 50%. Figure 4.18 shows the texture of the turtle model and its obtained borders. Figures 4.19 and 4.20 show a simplification of 25% of the geometry of the turtle model without applying the new metric and then applying it. Better visual results are obtained with all the models.

Table 4.1 shows the number of polygons of each of these models. It can be seen that this method preserves much better textures than methods that do not take textures into account in the simplification process. We have used the root mean square error (RMSE) of the pixel-to-pixel image difference [LT00] to measure the mean visual error between the original and the simplified models. This error was taken using 24 viewpoints and 512x512 resolution images; both the number of viewpoints and the resolution were different from those used to simplify all models. We must emphasize that each viewpoint was different from the one used during the simplification process. The RMSE error values for the simplified models applying and without applying our metric are showed in Figures 4.9, 4.13, 4.17 and 4.21. We can appreciate in the results that better visual results for the viewer always obtained by using our metric extension. It can be observed that better quantitative results are obtained by applying our metric with the models eye, ninja and robot. However, with the turtle model we obtain higher values with our metric than without applying it. This is because RMSE measures the difference between pixels in the images. The user can observe a better visual aspect in the simplified version of the turtle model by applying our extension (Figures 4.19 and 4.20), with a more accurate texture appearance. However, RMSE penalizes a lot the parts of the silhouette that have disappeared in the simplification process, like for example the eyes of the turtle model.

Table 4.1: Number of polygons of each model

Model	Number of polygons
Robot	308
Turtle	640
Ninja	1,008
Eye	5,400

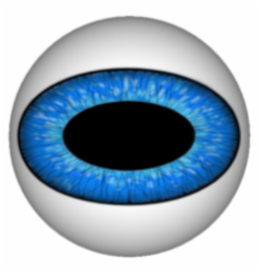


Figure 4.5: Original eye model

Figure 4.6: Borders of the eye model detected by the border detection method with sigma = 0.75, low threshold= 0.5 and high threshold = 0.6

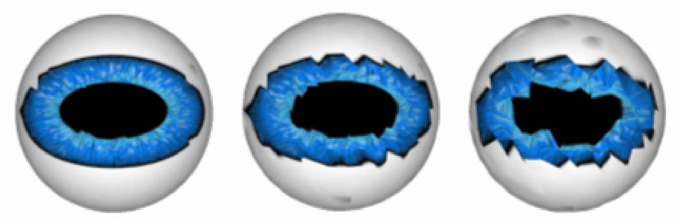


Figure 4.7: eye model simplified at 75%, 50% and 25% without applying our texture-based error metric

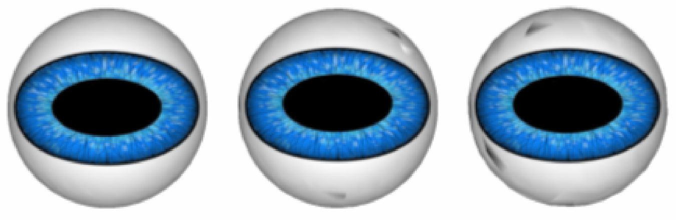
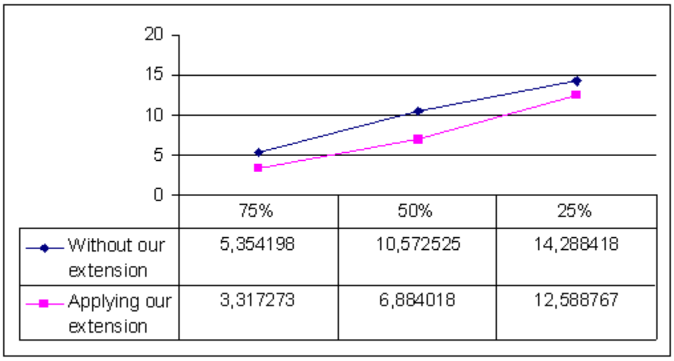


Figure 4.8: eye model simplified at 75%, 50% and 25% applying our texture-based error metric



	75%	50%	25%
◆ Without our extension	5,354198	10,572525	14,288418
■ Applying our extension	3,317273	6,884018	12,588767

Figure 4.9: RMSE errors of the eye model obtained without our extension and applying it for the simplifications to 75%, 50% and 25% of its original geometry

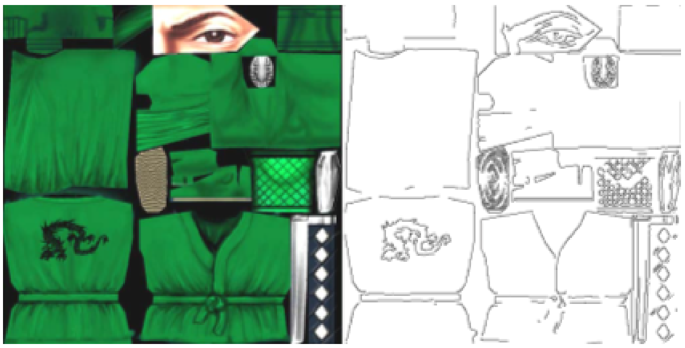


Figure 4.10: Borders of the ninja model detected by the border detection method with sigma = 0.75, low threshold= 0.5 and high threshold = 0.6

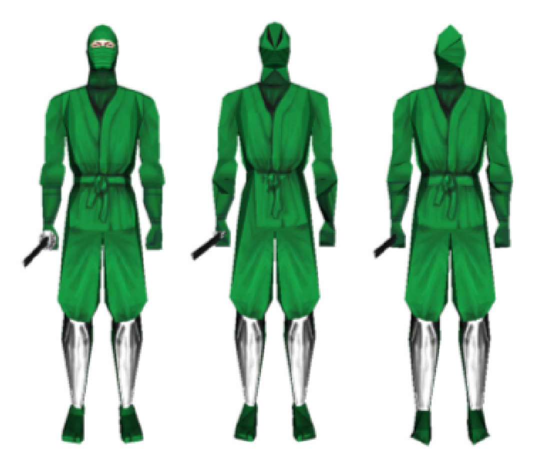


Figure 4.11: Front of the ninja model. Original model (left) and the model simplified at 50% without applying our texture-based error metric (centre) and applying it (right)

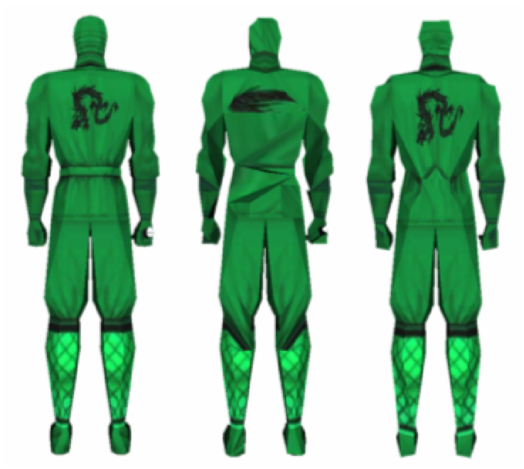
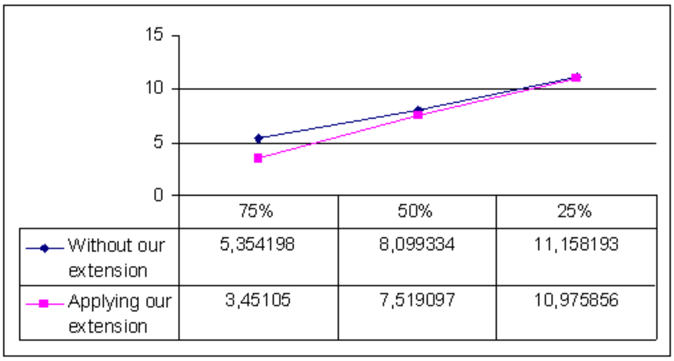


Figure 4.12: Back of the ninja model. Original model (left) and the model simplified at 50% without applying our texture-based error metric (centre) and applying it (right)



	75%	50%	25%
◆ Without our extension	5,354198	8,099334	11,158193
■ Applying our extension	3,45105	7,519097	10,975856

Figure 4.13: RMSE errors of the ninja model obtained without our extension and applying it for the simplifications to 75%, 50% and 25% of its original geometry

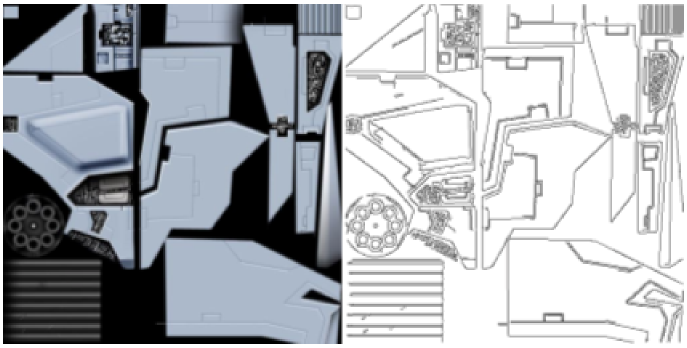


Figure 4.14: Borders of the robot model detected by the border detection method with sigma = 0.75, low threshold= 0.5 and high threshold = 0.6

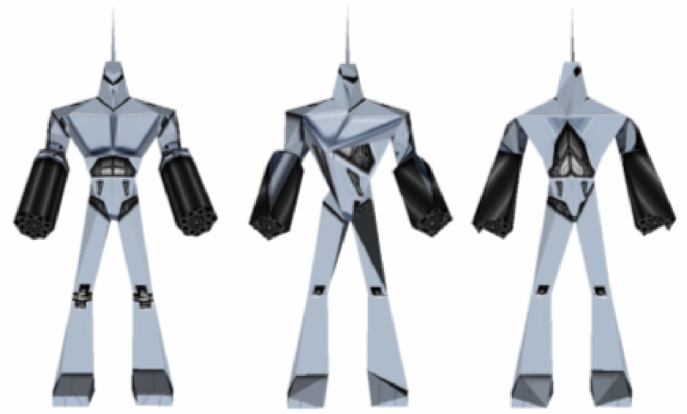


Figure 4.15: Front of the robot model. Original model (left) and the model simplified at 50% without applying our texture-based error metric (centre) and applying it (right)

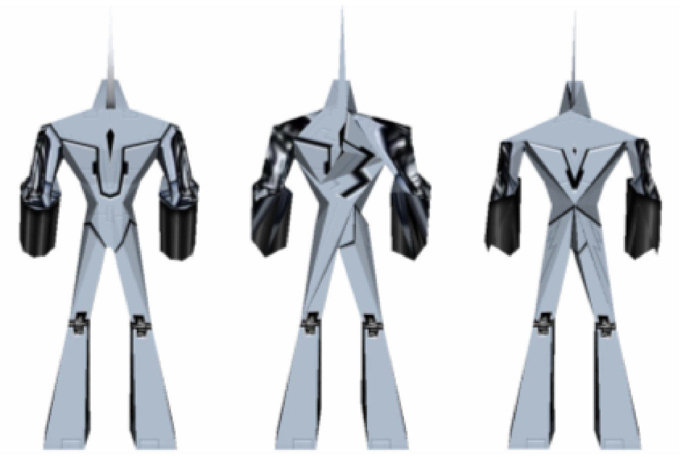
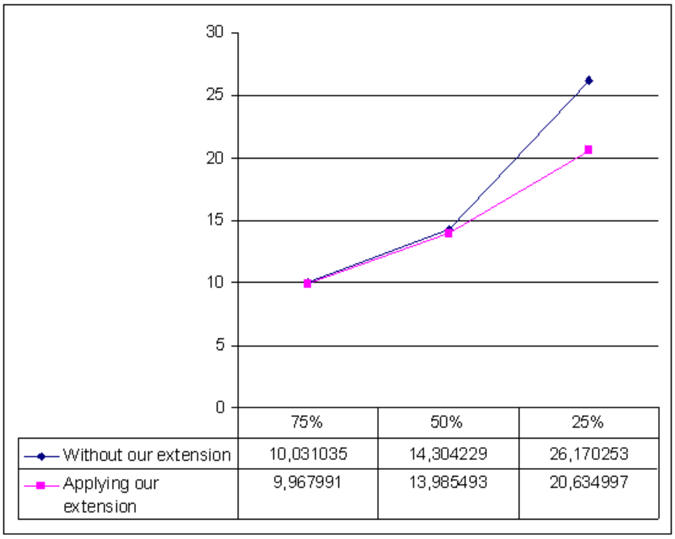


Figure 4.16: Back of the robot model. Original model (left) and the model simplified at 50% without applying our texture-based error metric (centre) and applying it (right)



	75%	50%	25%
◆ Without our extension	10,031035	14,304229	26,170253
■ Applying our extension	9,967991	13,985493	20,634997

Figure 4.17: RMSE errors of the robot model obtained without our extension and applying it for the simplifications to 75%, 50% and 25% of its original geometry

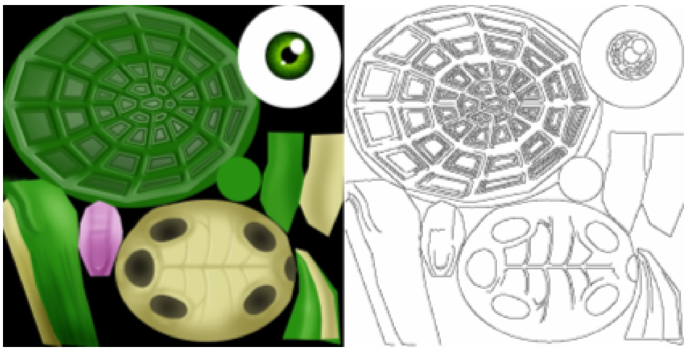


Figure 4.18: Borders of the turtle model detected by the border detection method with sigma = 0.75, low threshold= 0.5 and high threshold = 0.6

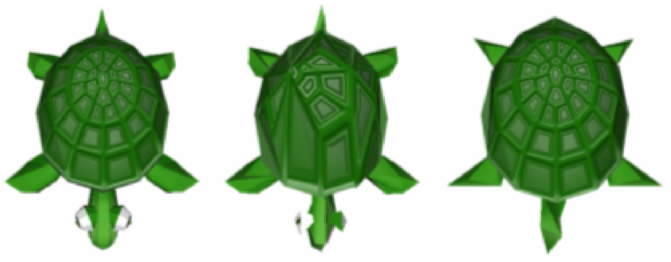


Figure 4.19: Front of turtle model. Original model (left) and the model simplified at 25% without applying our texture-based error metric (centre) and applying it (right)

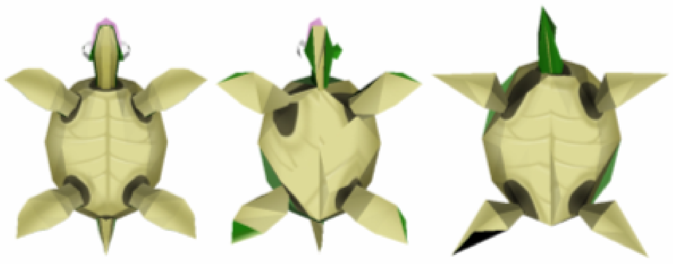
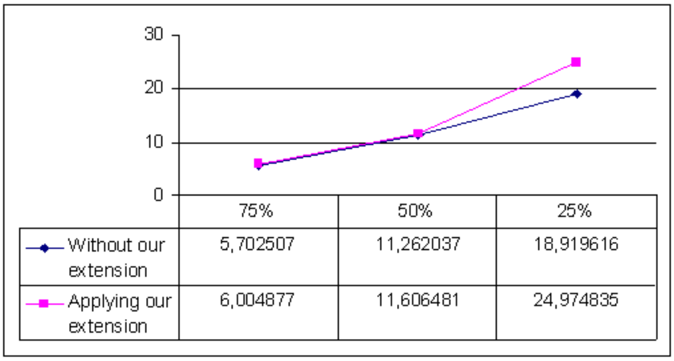


Figure 4.20: Back of turtle model. Original model (left) and the model simplified at 25% without applying our texture-based error metric (centre) and applying it (right)



	75%	50%	25%
◆ Without our extension	5,702507	11,262037	18,919616
■ Applying our extension	6,004877	11,606481	24,974835

Figure 4.21: RMSE errors of the turtle model obtained without our extension and applying it for the simplifications to 75%, 50% and 25% of its original geometry

4.4.2 Other attributes preservation

We have also applied this error metric extension for preserving other additional attributes, such as normals. An example of this use is shown in this section. Figure 4.22 shows the normal map of the robot model and the borders obtained after applying the border detection method. The simplification result of using this normals preservation is shown in Figures 4.23 and 4.24. In this figure a simplification of the robot model to 45% of its original geometry is shown without applying our extension and applying it. Better visual results can be appreciated by applying our extension.

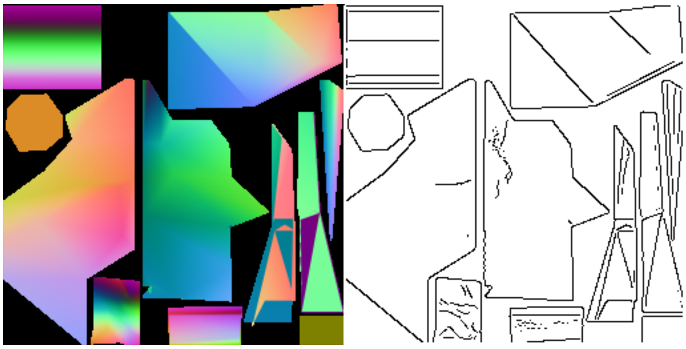


Figure 4.22: Normal map of the robot model (a) and its borders detected by the border detection method with sigma = 0.7, low threshold= 0.5 and high threshold = 0.6

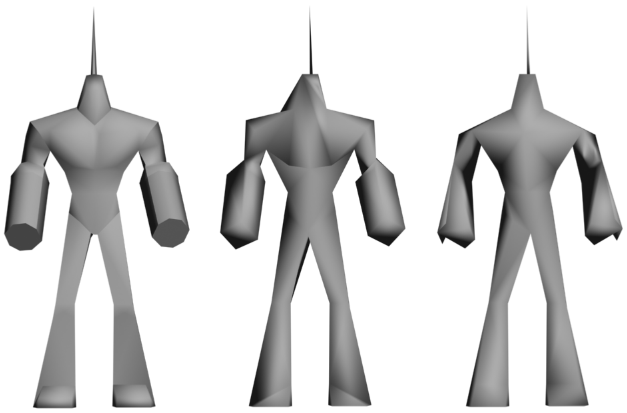


Figure 4.23: Front of the original robot model (left), its simplification to 45% without considering the border detection in normal maps (centre) and considering it (right)

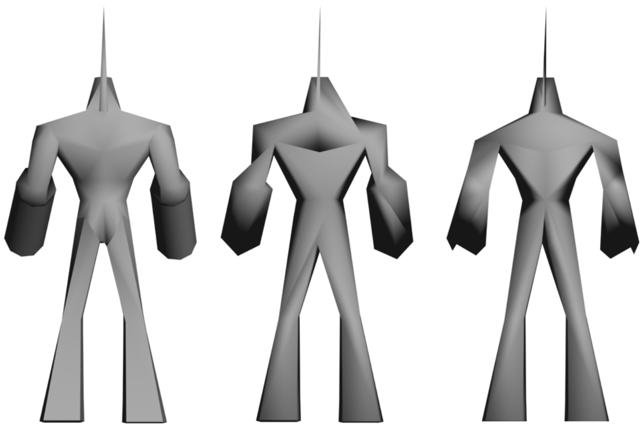


Figure 4.24: Back of the original robot model (left), its simplification to 45% without considering the border detection in normal maps (centre) and considering it (right)

4.5 Conclusions

In this chapter, a texture-based error metric extension for simplification methods that uses edge collapse operations has been presented. With this extension the texture information of textured models is also considered in those simplification methods that did not take it into account before. The original error and the new error based on texture information are both used in the weighting of the edges.

When extending the previous error with this metric, the simplification order of the regions that previously had a similar collapse cost may change. After applying the metric, edge collapses would be produced earlier in the regions with fewer changes in the texture. Thus, the regions with great changes in the texture are simplified later than the others that have fewer changes in the texture and a similar previous error. Therefore, this metric extension avoids the early simplification of triangles which contain abrupt changes in the texture, which prevents great texture distortions from appearing in simplified models. Better visual results are obtained by applying our metric in comparison to methods that do not consider the texture information in their error metric. Results have also been tested with the metric RMSE [LT00].

This metric extension has also been tested for preserving other additional attributes, such as normals, by obtaining the borders of the normal maps of the models. Better visual results can also be appreciated by applying our extension for this purpose.

The computational cost introduced by this metric is negligible in comparison to the simplification cost.

Moreover, a common geometric property of meshes used in interactive applications is considered. This kind of mesh can have duplicated vertices in the same spatial position in order to assign them different attributes, like normals or texture coordinates. Simplifying these meshes without taking this property into account can produce holes and great distortions in the simplified objects. Therefore, a solution to consider this property has been also presented in this chapter.

Thus, this contribution presents a new metric extension useful for extending the error metric of the simplification methods in order to take the texture information into account.

CHAPTER 5

Viewpoint-driven Simplification Using Textures

5.1 Introduction

In this chapter we present a simplification method with new metrics that consider texture information in the simplification process (Section 5.2). As explained before, textures play a very important role in the final aspect of the objects for the viewer. Good geometric simplifications that do not consider the texture information can produce great distortions when the objects are textured. Therefore, the method presented here does not only consider the changes of the geometry after each simplification step, but it also measures the changes in the final aspect of the model with its texture applied on it. This method is based on the segmentation of the texture images and makes use of well-known concepts of the Information Theory.

The latest graphics hardware advances allow some applications to be accelerated. Viewpoint-driven simplification methods are usually slow, due to the number of renders necessary to complete the process. Therefore, the latest graphics hardware capabilities have been used to speed up a viewpoint-driven simplification method, like the one presented here (Section 5.3).

5.2 Segmentation-based simplification for textured models

Here we present a new simplification method for triangle meshes that takes texture shapes into account in its error metric.

The presented method is a viewpoint-driven simplification method and uses edge collapse operation. An edge collapse is a simplification operation that removes edges by merging the vertices of the edges. The final vertex can be placed at one of the original vertices (half-edge collapse) or can be moved to other spatial coordinates. Figure 5.1 shows an example of a half-edge collapse operation.

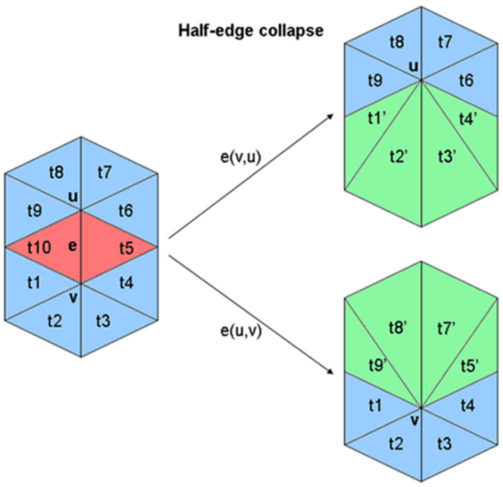


Figure 5.1: The half-edge collapse operation. In this example the edge e is collapsed into vertex u (see e(v, u)), but is also collapsed into v (see e(u, v)). Triangles t10 and t5 are removed

We use different concepts from Information Theory, such as the discrete entropy and mutual information defined by Shannon [Sha48], the generalized entropy presented in [Tsa88] and generalized mutual information presented in [HC67] in order to compute the error metric of the method. We can compare the use of each of them. Therefore, we also present a quantitative comparison of the use of each of them by obtaining the mean visual error given by the root mean square error (RMSE) of the pixel-to-pixel image difference defined as in [LT00].

Our method is a viewpoint-driven simplification method. Therefore, we use some cameras around the object in order to obtain the cost associated to each edge. These associated costs will give us the simplification order. In order to take the texture information into account, we use an image segmentation method. This will produce another image with different colored regions. Considering texture information, edge collapses that produce a great change in the texture will have a high associated cost. Therefore, we penalize those edges that their collapse can produce great distortions in the final aspect of the model. The main steps of this method are:

- The segmentation of the texture image, generating a new image with some colored regions.
- The calculation of the initial costs associated to the edges of the model. This will take into account the information obtained after the segmentation step.
- The simplification algorithm. After each simplification step, some costs will be recalculated. This will also consider the segmentation information.

5.2.1 Simplification method

We propose a viewpoint-driven simplification method based on edge collapse operation. Its metric makes use of the texture segmentation information to assign the cost associated to each edge of the model.

The main parts of the method are the segmentation of the texture image, the initial computation of edge costs and the iterative simplification process. First, the method performs the segmentation of the texture. With this process, a color-coded image with the most relevant regions of the model is obtained. After that, it calculates the initial cost associated to each edge by using the defined error metrics (Subsection 5.2.2). These error costs will establish the simplification order. And, finally, the iterative simplification process is performed. After each edge collapse, the cost of some specific edges in the affected regions is recalculated. The method performs the edge collapse operations until the desired level of simplification is obtained. A general workflow of the method is shown in Figure 5.2.

The presented method considers texture shape in its metric. The algorithm of simplification will produce simplified models considering the shape of the texture applied on them. This way, great distortions in the applied texture of simplified models will be avoided.

We divide the textures into regions. To do this, we perform a segmentation of the texture image. We use the method presented in [FH04]. With this method, a new color-coded image with the different regions of the texture is obtained. The authors defined a predicate for measuring the evidence for a boundary between two regions using a graph-based representation of the image in order to produce the segmentation. Additionally, we have modified the color selection in order to have a different color for each region. Therefore, we can identify each region with a unique color.

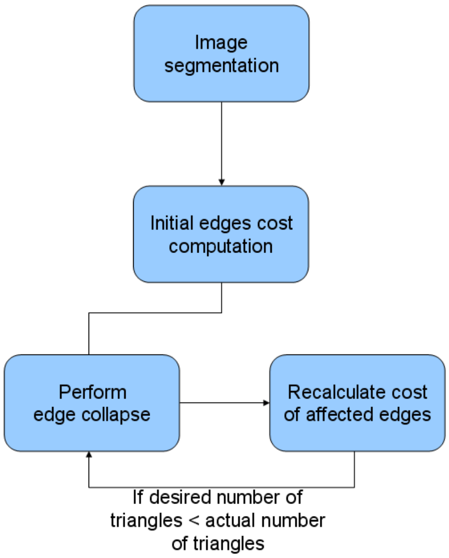


Image segmentation
Initial edges cost computation
Perform edge collapse
Recalculate cost of affected edges
If desired number of triangles < actual number of triangles

Figure 5.2: General workflow of the method

Figure 5.3 shows the segmentation of a texture image. This method accept different parameters, such as σ (parameter of a Gaussian distribution used in the segmentation process) and k (useful to compute a defined threshold function). Depending on the value of these parameters, we will obtain different segmentation results (see [FH04]).

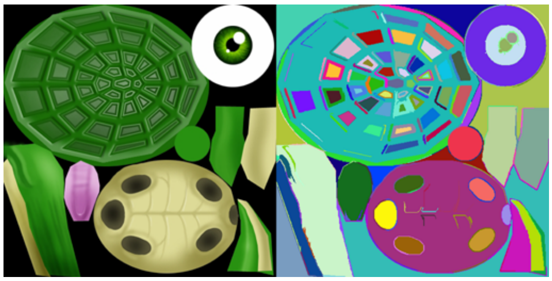


Figure 5.3: Original texture of a turtle model (left) and the image after the segmentation process (right)

In Section 5.2.2 we introduce the metrics presented here to consider the texture information in the simplification process. And in Section 5.2.3 we explain in detail the simplification steps.

5.2.2 Error metrics

During the last years some authors have combined the idea of some Information-theoretic measures of similarity with simplification methods ([LT00], [CSCF08]). The results presented in [CSCF07] proved that this kind of measures can be used with efficiency in the context of polygonal simplification. In this section we present new measures to take textures into account in a simplification process.

Region viewpoint entropy

From the viewpoint entropy defined in [VFSW01] for polygonal models, we present here the region viewpoint entropy. Applying the image obtained after the segmentation process to the object, we can calculate the relative area of the regions projected over the sphere of directions centered at viewpoint v. Therefore, we define the region viewpoint entropy as

$$H_{rv} = -\sum_{i=0}^{N_r} \frac{ar_i}{ar_t} \log \frac{ar_i}{ar_t}, \tag{5.1}$$

where N_r is the number of regions in the segmentation image, ar_i is the area of the region i projected over the sphere, ar_0 represents the projected area of the background in open scenes, and $ar_t = \sum_{i=0}^{N_r} ar_i$ is the total area of the sphere. Maximum entropy is obtained when a certain viewpoint can see all the regions with the same projected area. The best viewpoint is defined as the one that has maximum entropy.

Region viewpoint generalized entropy

We present here the region viewpoint generalized entropy. Entropy is considered a particular case of the generalized entropy definitions proposed by Rényi [R\'61] and Harvda and Charvát [HC67]. Tsallis [Tsa88] used the Harvda-Charvát entropy in order to generalize the Boltzmann entropy in statistical mechanics. Therefore, we have considered the so-called Harvda-Charvát-Tsallis entropy in order to define the region viewpoint generalized entropy. The Harvda-Charvát-Tsallis entropy is defined by

$$H^T_\alpha(X) = k\frac{1 - \sum_{x\in X} p(x)^\alpha}{\alpha - 1}. \tag{5.2}$$

This entropy recovers the Shannon discrete entropy when $\alpha \to 1$.

Applying the image obtained after the segmentation process to the object, we can calculate the relative area of the regions projected over the sphere of directions centered at viewpoint v. Therefore, we define the region viewpoint entropy is defined as

$$Hrv^T_\alpha(X) = k\frac{1 - \sum_{i=0}^{N_r} (\frac{ar_i}{ar_t})^\alpha}{\alpha - 1}, \tag{5.3}$$

where N_r is the number of regions in the segmentation image, ar_i is the area of the region i projected over the sphere, ar_0 represents the projected area of the background in open scenes, and $ar_t = \sum_{i=0}^{N_r} ar_i$ is the total area of the sphere. Maximum entropy is obtained when a certain viewpoint can see all the regions with the same projected area. For extension, the best viewpoint is defined as the one that has maximum entropy.

Note that the equation defined here can be also useful using the area of the polygons of a model instead of the regions, like in the viewpoint entropy, defined in [VFSW01].

Region viewpoint mutual information

From the viewpoint mutual information introduced in [VFSG06] [FSG09], we present here the region viewpoint mutual information. The viewpoint mutual information was introduced to select the best views. Taking the information of the regions of the texture into account, an information channel $V \to R$ between the random variables V and R is defined. This channel represents, respectively, a set of viewpoints and the set of regions of a textured model. Viewpoints are indexed by v and regions by r. The marginal probability distribution of V is given by $p(v) = \frac{1}{N_v}$, where N_v is the number of viewpoints. That is, the same probability is assigned to each viewpoint. The conditional probability $p(r \mid v) = \frac{a_r}{a_t}$ is defined by the normalized projected area of region r over the sphere of directions centered at viewpoint v. Conditional probabilities fulfill $\sum_{r \in R} p(r \mid v) = 1$. Note that with this notation viewpoint entropy can be rewritten as $H_v = -\sum_{r \in R} p(r \mid v) \log p(r \mid v)$. Finally, the marginal probability distribution of R is given by $p(r) = \sum_{v \in V} p(v)p(r \mid v) = \frac{1}{N_v} \sum_{v \in V} p(r \mid v)$, which represents the average projected area of region r, i.e., the probability of a region r to be hit (seen) by a random ray cast from the viewpoint sphere.

From this channel, the mutual information between V and R, which expresses the degree of dependence or correlation between a set of viewpoints and the regions of the textured model, is given by

$$I(V;R) = \sum_{v \in V} p(v) \sum_{r \in R} p(r \mid v) \log \frac{p(r \mid v)}{p(r)} = \frac{1}{N_v} \sum_{v \in V} I(v;R), \tag{5.4}$$

where

$$I(v;R) = \sum_{r \in R} p(r \mid v) \log \frac{p(r \mid v)}{p(r)}, \tag{5.5}$$

called region viewpoint mutual information (RVMI), represents the degree of correlation between the viewpoint v and the set of regions R, and it is a measure of the quality of viewpoint v. Quality is considered here equivalent to representativeness. The best viewpoint is defined as the one that has minimum

RVMI. High values of the measure mean a high degree of dependence between the viewpoint v and the object, indicating a highly coupled view. On the other hand, low values correspond to the most representative or relevant views, showing the maximum possible number of regions in a balanced way.

Region viewpoint generalized mutual information

We present here the region viewpoint generalized mutual information. Taneja [Tan88] and Tsallis [Tsa98] introduced the generalized mutual information. From the Harvda-Charvát-Tsallis mutual information and taking the regions of the texture of a model into account, we define the region viewpoint generalized mutual information. The Harvda-Charvát-Tsallis mutual information $I_\alpha^T(X,Y)$ between two discrete random variables X and Y is defined by

$$I_\alpha^T(X;Y) = \frac{1}{1-\alpha}\left(1 - \sum_{x\in X}\sum_{y\in Y}\frac{p(x,y)^\alpha}{p(x)^{\alpha-1}p(y)^{\alpha-1}}\right). \tag{5.6}$$

Therefore, similar to region viewpoint entropy (subsection 5.2.2), the generalized mutual information between V and R is given by

$$I_\alpha^T(V;R) = \sum_{v\in V} p(v)\frac{1}{\alpha-1}\sum_{r\in R} p(r \mid v)\left(\frac{p(r\mid v)^{\alpha-1}}{p(r)^{\alpha-1}} - 1\right), \tag{5.7}$$

where

$$I_\alpha^T(v;R) = \frac{1}{\alpha-1}\sum_{r} p(r \mid v)\left(\frac{p(r\mid v)^{\alpha-1}}{p(r)^{\alpha-1}} - 1\right) \tag{5.8}$$

is the region viewpoint generalized mutual information. This definition recovers the region viewpoint mutual information when $\alpha \to 1$.

Note that the equation defined here can be also useful using the area of the polygons of a model instead of the regions, like in the viewpoint mutual information, introduced in [VFSG06] [FSG09].

5.2.3 Simplification steps

The simplification process is divided into two main steps: the initial edge cost computation and the iterative simplification process.

Initial edge cost computation

We perform an initial edge cost computation in order to assign a cost to each edge. This will establish the order of the edge collapse operations. The associated cost of an edge represents how the regions are going to change after the collapse of this edge. The edges with high associated cost will be collapsed in the last simplification steps. Therefore, the edges collapsed first will be those that will produce less change in the texture regions. This way, the method collapses first the edges that will produce less distortion in the texture aspect in the simplified model. The background is considered as another region. Thus, models will also maintain their external geometric appearance by giving high costs to those edges that their collapse will produce a great distortion in the silhouette of the model, because the changes in the region formed by the background will also penalize these edge collapses. We use a histogram to calculate the projected area of each region for each camera.

We define the cost associated to each edge as the sum of the differences, before simplifying and after simplifying, of the metric used in the method. That is,

$$c_e = \sum_{v \in V} \mid M_{rv} - M'_{rv} \mid \tag{5.9}$$

being M_{rv} and M'_{rv} the actual and previous value of the metric used in the method. In order to assign the initial cost associated to each edge, the method works as follows:

- Step 1. Locate some cameras around the object. The distribution of these cameras is based on platonic solids. That is, we place a camera at each of the vertices of the selected platonic solid (for example, an icosahedron will produce 12 cameras and a dodecahedron will produce 20 cameras). The cameras will look at the center of the sphere formed by the solid. We have used 20 cameras, like in ([LT00], [CSCF08]).

- Step 2. Render the model in the center of the sphere, textured with the image obtained after the segmentation process. See Figure 5.4.

- Step 3. Calculate the initial M_{rv} for the model textured with the segmented texture image.

- Step 4. Perform an edge collapse without an associated cost (initially, there is no edge with an associated cost).

- Step 5. Calculate the actual M_{rv} and assign the difference with the original M_rv to the collapsed edge.

- Step 6. Undo the edge collapse.

- Step 7. Until all the edges have an associated initial cost, go to Step 4.

Figure 5.4 shows an example of a model textured with the image obtained after the segmentation of its texture. Some cameras are located around the model.

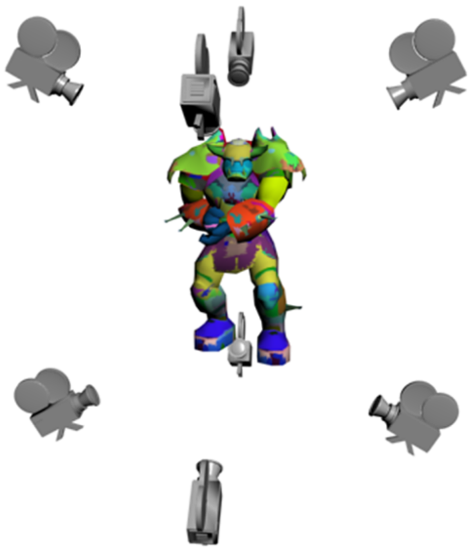


Figure 5.4: Model textured with the segmented texture image and cameras around it

Iterative simplification process

After assigning the initial collapse costs to all the edges in the model, the method will perform an iterative simplification process. Normally, the level of simplification is indicated as the number of desired final faces. The method has a count of the faces in the model at each moment. Therefore, the simplification process will be performed until the desired level of simplification is obtained. Edges are collapsed in the order given by their associated cost. Edges with a low associated cost will be collapsed before than edges with a high associated cost. After collapsing an edge, the cost of some edges must be recalculated. These edges are the ones that are adjacent to the vertices adjacent to the vertex resulting from a collapse (see Figure 5.5). This is because the regions of the texture applied on the model, may change when the geometry is altered. To do this, we create a viewport from each camera in order to analyze only these edges. Therefore, we avoid recalculating the cost of all the edges in the model again. The iterative simplification process works as follows:

- Step 1. Extract the edge E with the lowest associated cost.
- Step 2. Perform collapse of E.
- Step 3. Retriangulate the affected faces.
- Step 4. Recalculate cost of the affected faces.

- Step 5. While the number of faces if greater than the desired number of faces, go to Step 1.

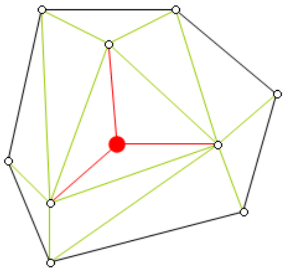


Figure 5.5: Edges to be recalculated after an edge collapse. The red point is the vertex obtained after the collapse. The red edges are the edges that contain this vertex. The green edges are those that are adjacent to the vertices adjacent to the vertex obtained after the edge collapse operation

5.3 Acceleration by hardware

Recent advances in real-time rendering have provided a new way to speed up the simplification rates, by the GPU implementation of simplification methods. This way, the work can be distributed between the CPU and the GPU, parallelizing the simplification tasks.

CUDA (Compute Unified Device Architecture) is a new compiler and tool package developed by nVidia, which allows the parallelization of some tasks in the GPU. This architecture provides new possibilities to speed up the simplification methods. For example, in [RO08] some methods useful for mesh operations are implemented in CUDA. Figure 5.6 depicts the CUDA architecture.

View-dependent simplification methods are usually slower than methods based on geometry. This is because they need to render several times the object to be simplified. Here, we propose a way to use CUDA to speed up a viewpoint-driven simplification method.

We have accelerated the method presented in Section 5.2, using the region viewpoint entropy and region viewpoint mutual information as formulas to compute the error metric.

Description of the acceleration

First, we have calculated the region viewpoint entropy and region viewpoint mutual information with CUDA. This way we take profit of the capabilities of new graphics hardware.

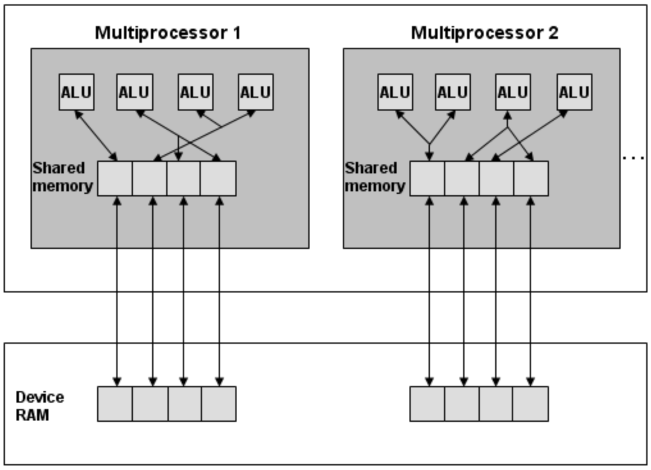


Figure 5.6: CUDA architecture

In order to compute the region viewpoint entropy and region viewpoint mutual information, the projected areas of the regions of the model have to be calculated. To obtain the projected areas from different points of view, we locate some cameras surrounding the object. The location of the cameras is performed by a uniform distribution. We locate the cameras around the object in the vertices of different platonic solids, like a dodecahedron (12 cameras) and an icosahedron (20 cameras). We render the model from each camera, storing all the data of each camera in a unique pixel buffer object. We make use of a *pixel buffer object* [nVi04b] to transfer the data. This is the fast data transfer to and from a graphics card through DMA (Direct Memory Access), without involving CPU cycles.

Now the histograms from the different cameras can be obtained. nVidia presented a method for computing histograms for nVidia CUDA compatible devices [nVi07], but this works with a range of 64 or 256 bins. This is because the shared memory is limited to 64 Kb. An efficient histogram for any number of bins is presented in [SK07]. This work presents two possible implementations. The first one allows atomic updates to be simulated in software, and the second one avoids the collision updates.

For computing the histograms, the method presented in [SK07] is used. This method allows atomic updates to be simulated in the shared memory of a block. Note that this can also be performed directly with the last graphic cards (from nVidia GTX generation). Therefore, this implementation could also be performed by using atomic updates with this kind of graphic card. However, the method presented in [SK07] works with any nVidia CUDA compatible device. We store all the histograms in a same array, which will be used to perform the calculation of the region viewpoint entropy and the region viewpoint mutual information.

In order to achieve the metric computation, we parallelize the calculation by two different CUDA kernels. For the computation of the region viewpoint entropy we just need the histograms already calculated. Atomic operations are used in order to avoid conflicts in the same memory address. For the calculation of the region viewpoint mutual information we have to take into account that $p(r \mid v)$ is an array calculated as the histogram values divided into the resolution of the viewport. Moreover, $p(r)$ can be calculated as the mean histogram. This way, the computation of the region viewpoint mutual information can also be parallelized, using atomic operations.

With the nVidia CUDA GPU occupancy calculator [nVi09b] we can adjust the number of blocks and threads to be executed in each kernel, in order to obtain the maximum performance

Figure 5.7 shows the workflow of the method.

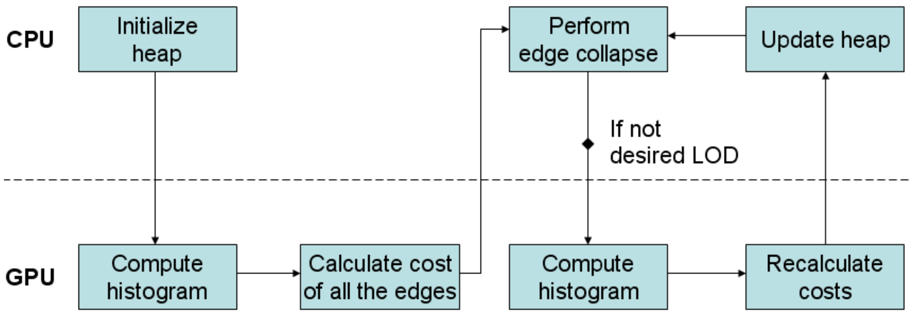


Figure 5.7: Workflow of the accelerated method

5.4 Results

This section presents the results of the method. These results are divided in two subsections. On the one hand, Subsection 5.4.1 presents the visual improvement of the simplified textured models produced by using our method. On the other hand, Subsection 5.4.2 presents the temporal improvements produced by the speed-up presented in Section 5.3.

5.4.1 Visual improvement

Several models have been tested with the presented method. We present the results compared to another simplification method [CSCF07] which has similar simplification steps, but without taking textures into account.

This section shows some models and different simplifications of them. The models are the shirt model (Figure 5.9), the head model (Figure 5.13), the woman model (Figure 5.17) and the astroboy model (Figure 5.21). All these models have been simplified with the method presented in [CSCF07] and different configurations of our method. We present simplifications using the region viewpoint entropy and mutual information using the definition presented by Shannon [Sha48] in the error metric and with region viewpoint generalized entropy and region viewpoint generalized mutual information using different values of the alpha parameter. Therefore, the shirt model is simplified to 50% of its original geometry (Figures 5.9), the head model is simplified to 50% (Figures 5.13), the woman model is simplified to 50% (Figures 5.17) and the astroboy model is simplified to 30% (Figures 5.21). Their original and segmented textures are shown in Figures 5.8, 5.12, 5.16 and 5.20

It can be seen that this method preserves much better textures than methods that do not take textures into account in the simplification process. We have used the root mean square error (RMSE) of the pixel-to-pixel image difference defined as in [LT00] to measure the mean visual error between the original and the simplified models. This error was taken using 24 viewpoints and 512x512 resolution images; both the number of viewpoints and the resolution were different from those used to simplify all models. We must emphasize that each viewpoint was different from the one used during the simplification process. It can be seen that lower values of the error are obtained with our method. In general, considering the region viewpoint entropies as metric, better values of the error are obtained by using the Shannon definition or with values of the alpha parameter near to 1. Note that the generalized entropy recovers Shannon discrete entropy when $\alpha \to 1$ (this is commented in Subsection 5.2.2). Using the region viewpoint mutual informations as metric better results can be obtained than with the region viewpoint entropies in some cases, depending on the model. Therefore, by using all the metrics proposed here better results than without considering the texture information are obtained. Thus, by using the different configurations of the metrics presented here, the user can test and obtain simplifications that minimize the distortions of the texture due to the simplification process.

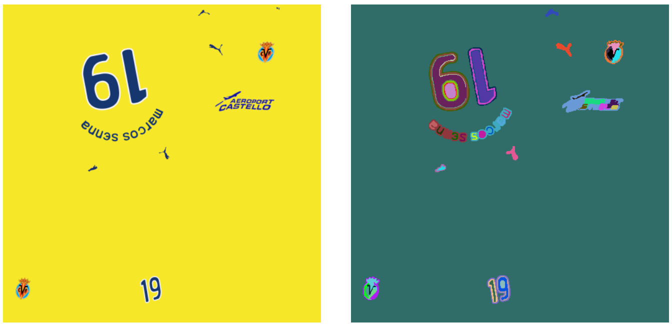
Moreover, we present the evolution of the obtained RMSE for different percentages of simplification of each model using the region viewpoint entropy and the region viewpoint mutual information as error metrics. This can be seen in Figures 5.11, 5.15, 5.19 and 5.23. In these graphics we can see that in some cases the use of the region viewpoint entropy presents smoother transitions than the region viewpoint mutual information. This could avoid abrupt changes between two different levels of simplification of the models.

The temporal cost of this method is similar to the temporal cost obtained in other viewpoint-driven simplification methods. This is because the main difference is the way of computing the error metric, which is the calculation of simple mathematical operations. Moreover, the texture image segmentation is a fast process. It depends on the resolution of the texture and it can be considered as a pre-process.

Table 5.1 shows the geometric characteristics of the models used to present the results.

Table 5.1: Details of the models presented in the results section

Model	Triangles	Vertices
Shirt	1,040	554
Head	2,872	1,438
Woman	4,130	2,460
Astroboy	4,822	2,474



(a) (b)

Figure 5.8: Original (a) and segmented (b) textures of the shirt model

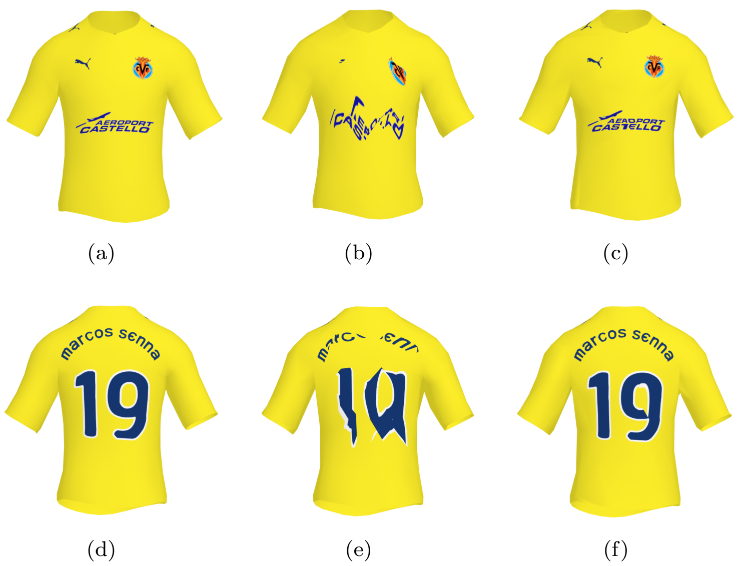


Figure 5.9: Original shirt model (a,d), simplifications to 50% without considering textures in the metric (b,e) and with the region viewpoint generalized entropy with alpha=1,25 (c,f)

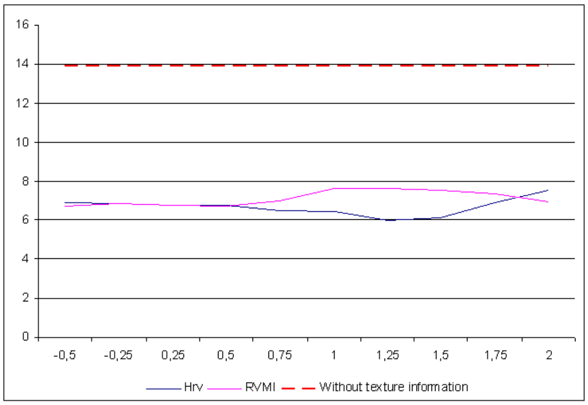


Figure 5.10: RMSE errors of the shirt model simplifications to 50% depending on the alpha parameter

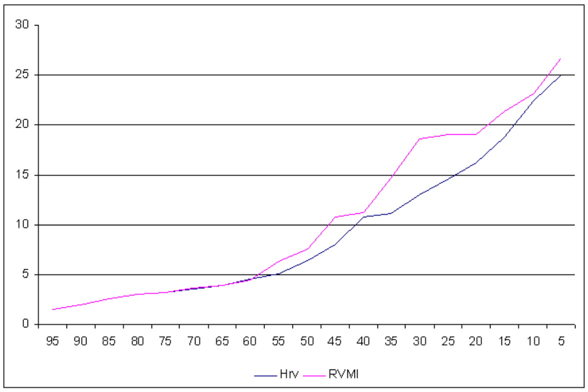


Figure 5.11: RMSE errors of the senna model depending on the percentage of simplification using the region viewpoint entropy and the region viewpoint mutual information as metrics

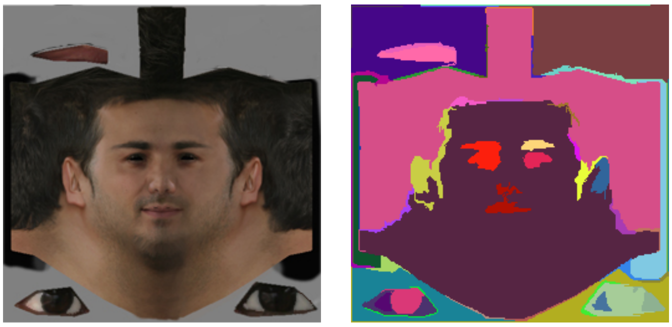
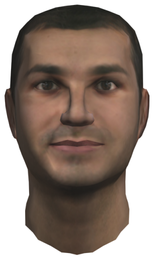
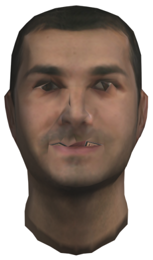
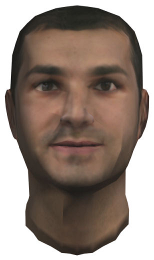
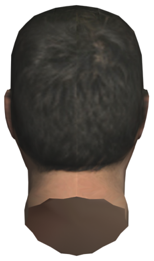


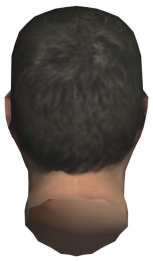
Figure 5.12: Original (a) and segmented (b) textures of the head model


(a)


(b)


(c)


(d)


(e)

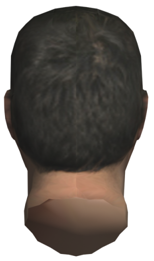

(f)

Figure 5.13: Original head model (a,d), simplifications to 50% without considering textures in the metric (b,e) and with the region viewpoint Shannon entropy (c,f)

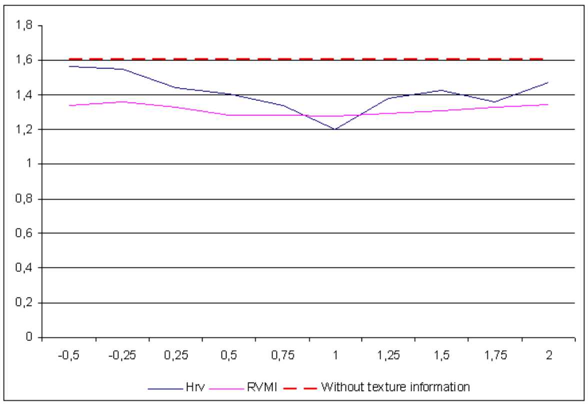


Figure 5.14: RMSE errors of the head model simplifications to 50% depending on the alpha parameter

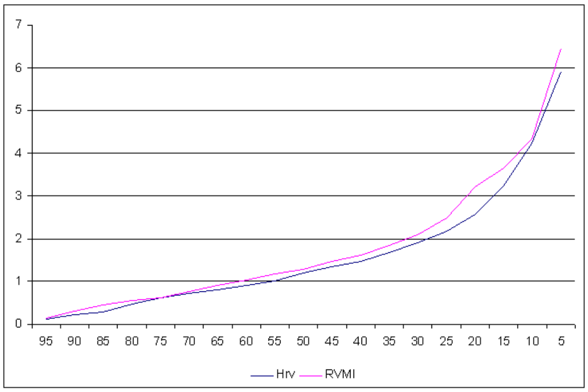


Figure 5.15: RMSE errors of the head model depending on the percentage of simplification using the region viewpoint entropy and the region viewpoint mutual information as metrics

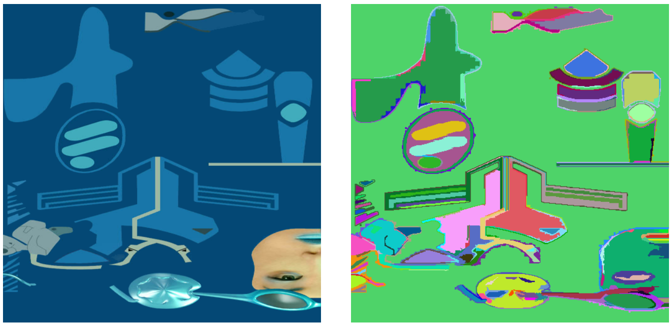


Figure 5.16: Original (a) and segmented (b) textures of the woman model

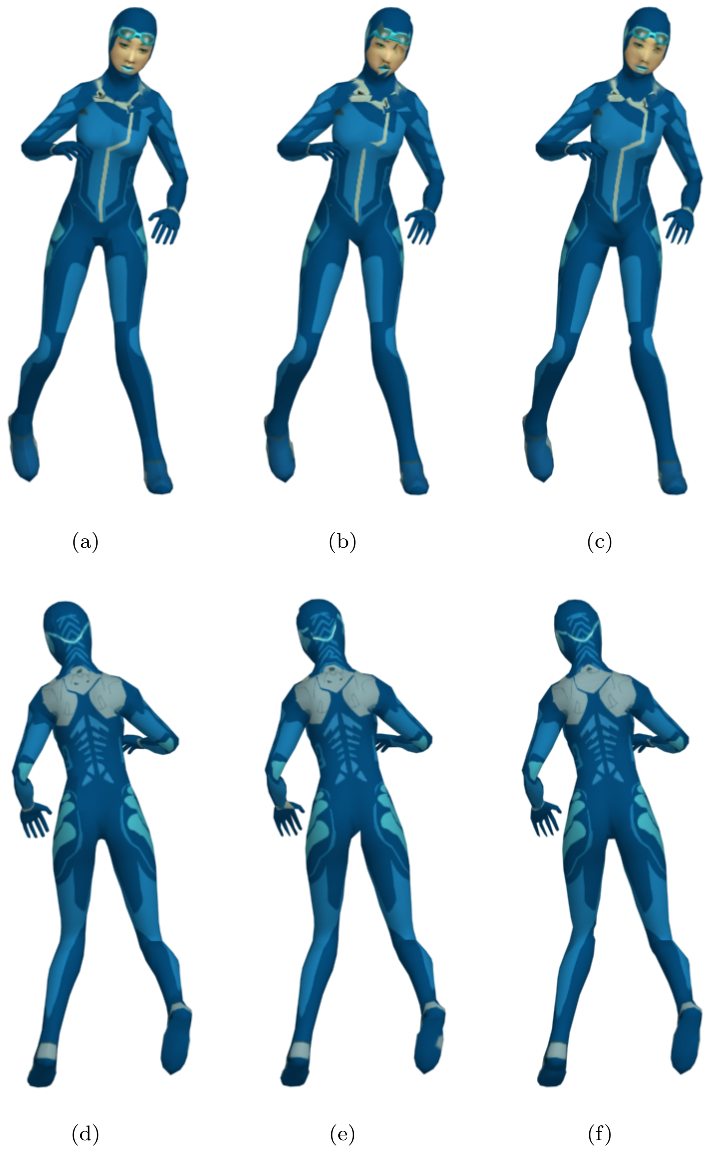


Figure 5.17: Original woman model (a,d), simplifications to 50% without considering textures in the metric (b,e) and with the region viewpoint generalized entropy with alpha=1,25 (c,f)

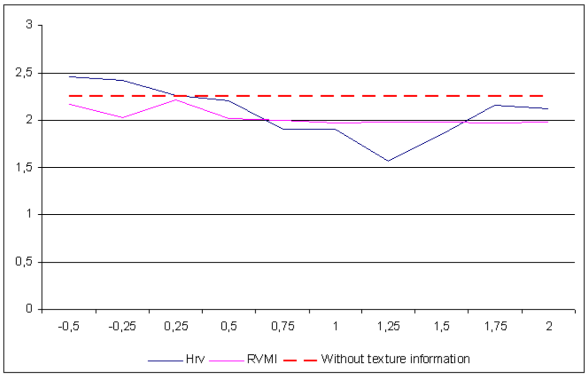


Figure 5.18: RMSE errors of the woman model simplifications to 50% depending on the alpha parameter

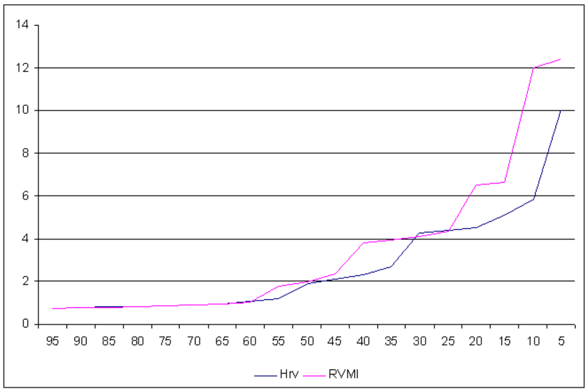


Figure 5.19: RMSE errors of the woman model depending on the percentage of simplification using the region viewpoint entropy and the region viewpoint mutual information as metrics

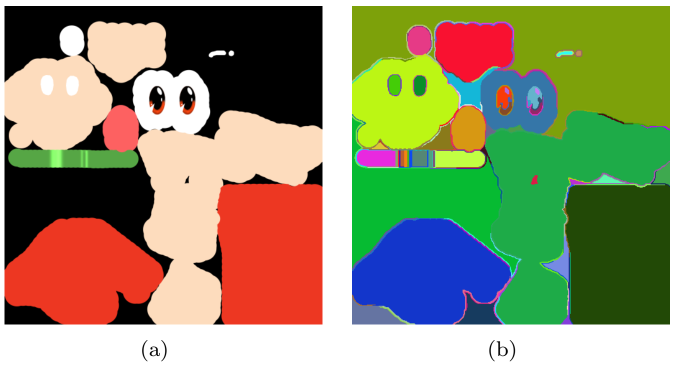


Figure 5.20: Original (a) and segmented (b) textures of the astroboy model

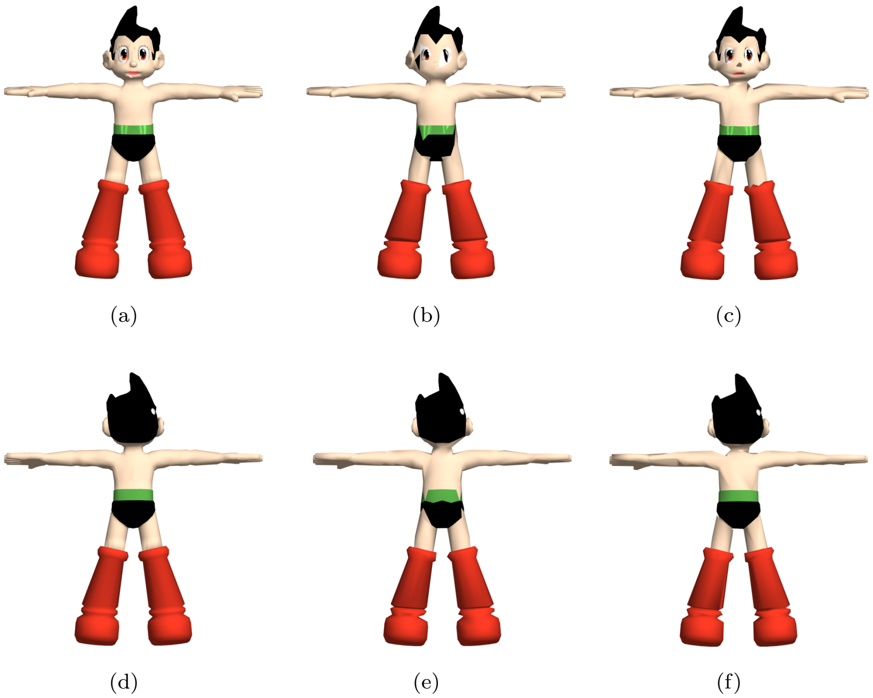


Figure 5.21: Original astroboy model (a,d), simplifications to 30% without considering textures in the metric (b,e) and with the region viewpoint generalized mutual information with alpha=0,75 (c,f)

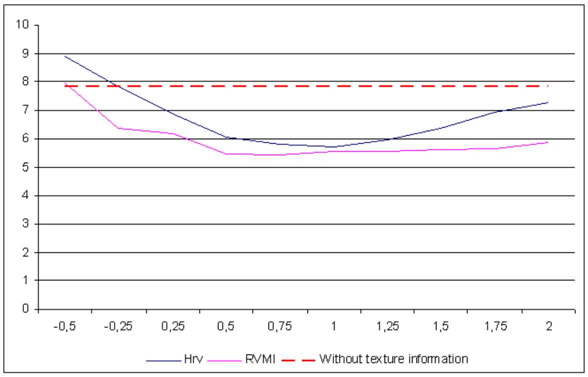


Figure 5.22: RMSE errors of the astroboy model simplifications to 30% depending on the alpha parameter

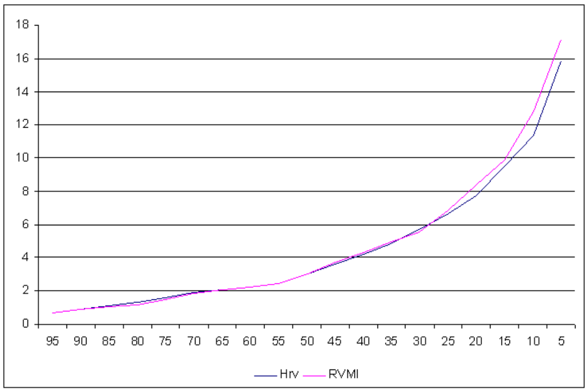


Figure 5.23: RMSE errors of the astroboy model depending on the percentage of simplification using the region viewpoint entropy and the region viewpoint mutual information as metrics

5.4.2 Temporal improvement

We have tested the speed-up presented in Section 5.3 by using the region viewpoint entropy and the region viewpoint mutual information as metrics with several simplifications. We can see that we obtain improved temporal results with this speed-up. In Table 5.2 we can see the times obtained with several simplifications. All the times were obtained in an Intel Pentium 4 3GHz, 2GB RAM with a nVidia GeForce 9800GTX+.

Table 5.2: Times using CPU implementations and times improved using CUDA

Triangles removed	Method	Metric	Simplification time (sec)
654	Original	Hrv	21.836
		RVMI	21.451
	Accelerated	Hrv	12.874
		RVMI	11.963
1,100	Original	Hrv	43.391
		RVMI	42.128
	Accelerated	Hrv	30.876
		RVMI	29.684
3,569	Original	Hrv	161.137
		RVMI	147.194
	Accelerated	Hrv	121.169
		RVMI	114.226
5,580	Original	Hrv	302.157
		RVMI	283.545
	Accelerated	Hrv	243.554
		RVMI	221.718

5.5 Conclusions

We have presented a new segmentation-based viewpoint-driven simplification method for textured triangle meshes in Section 5.2. Interactive applications tend to present models with a well-textured appearance. Therefore, textures play an important role in this kind of application.

Simplification methods allow the applications to present models with less geometry, reducing the GPU load. There are not many simplification methods that consider texture information during the simplification process. If this information is not taken into account, simplified models with great distortions in their texture appearance can be produced. This method considers texture information in its error metric. This way we obtain simplified models with a more accurate texture appearance than with simplification methods that do not consider this information. Moreover, well-known mathematical concepts, such as discrete and generalized entropies and mutual informations have been used to formulate the error metric.

Results have been compared and measured with a well-known metric (RMSE). The exposed results show the improvement in the texture appearance of the simplified models using this method, compared with the methods that do not consider texture information for simplifying the objects. Moreover, different configurations of our method have also been compared. In general, using the region viewpoint entropies as metric, better values of the error are obtained by using the Shannon definition or with values of the alpha parameter near to 1 (generalized entropy recovers Shannon discrete entropy when $\alpha \to 1$). Better results could also be obtained by using the region viewpoint mutual informations as metric. It depends on the models. Therefore, by using the different configurations of the metrics proposed here, different texture-preserving simplifications of the models will be obtained.

The temporal cost of this method without applying any acceleration technique (like the one presented in Section 5.3) is similar than the temporal costs of other viewpoint-driven simplification methods in the literature.

In Section 5.3, we have used CUDA architecture in order to speed-up a viewpoint-driven simplification method, like the one presented in 5.2. Results show a speed-up of the time employed performing all the instructions necessary to calculate the metric. More advances in the architecture will allow us to obtain still better results.

CHAPTER 6

GPU-based normal map generation

6.1 Introduction

A normal map is a two-dimensional image that contains RGB color elements which are interpreted as three-dimensional vectors containing the direction of the normal of the surface at each point. This is especially useful in real-time applications when the normal map is applied to a three-dimensional simplified model, because more detail in the simplified model can be specified without needing more geometry. This feature enables the use of correct per-pixel lighting by using the Phong lighting equation.

Normal map generation is a key part in real-time applications, such as video-games or virtual reality, due to the intensive use of techniques such as normal-mapping, used to increase the realism of the scenes.

Compared to traditional per-vertex lighting, per-pixel lighting with normal maps gives the rendered model a great amount of surface detail, which can be appreciated through the lighting interactions of the light and the surface. This technique gives detail to meshes without having to add real geometric detail. Normal maps can be built in relation to three spaces: world space, object space and tangent space.

- *World space*: each pixel stores its orientation in the world space, and no additional computation is required to obtain the normal value. It works correctly with static meshes.

- *Object space*: each pixel stores its orientation in the object space, so it would be necessary to apply the object transformation matrix to obtain the normal. It works correctly with moving objects.
- *Tangent space*: each pixel stores its orientation in relation to the face that the pixel belongs to. It is ideal for deformable objects.

Figure 6.1 shows the normal map of a model in the world space.

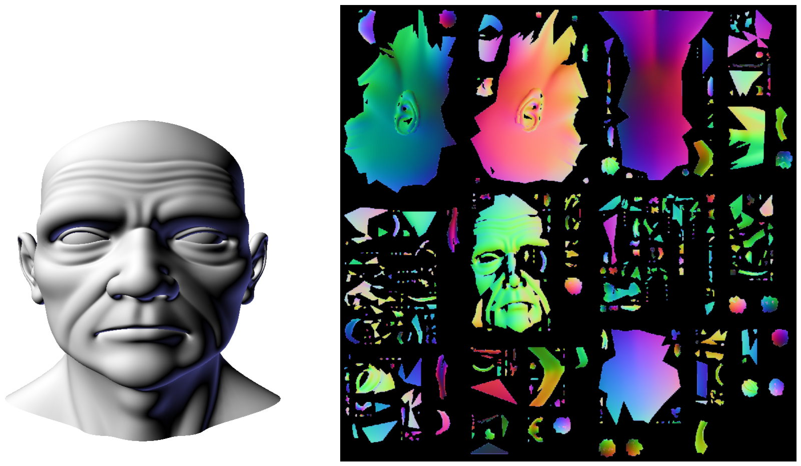


Figure 6.1: Original model (a) and its normal map in the world space (b)

Here we present two GPU-based methods for the generation of normal maps, using *vertex* and *pixel shaders*. It involves a greater processing velocity than the existing methods, because they make use of the CPU to generate the map. The majority of present-day methods are generated by software by programming the necessary instructions for the normal map generation so that the CPU processes them. Traditional normal map generation methods [nVi04a] [ATI02] use raytracing from the simplified model to generate the normal map. Then, they apply as normal in the simplified model the normal in the detailed model that is intersected by the ray. They offer the possibility of generating the normal maps in the object and the tangent spaces.

Although normal maps can be easily generated when both the reference and the working meshes use a common texture coordinate system, this is not always the case and thus, it is not trivial to implement on the graphics hardware. This is the reason why this kind of tools is often implemented in software.

The high programmability of current graphics hardware allows the implementation of these kinds of methods on the GPU. Therefore, the great scalability of the graphics hardware, which is increased even more each year, can be used to perform this task.

Vertex and *pixel shaders* are used for the hardware generation of normal maps. *Vertex shaders* and *pixel shaders* are small fragments of programmable code, which state the way that the GPU uses the vertices and pixels of the image. So, OpenGL sends the geometry of the object to the *graphics pipeline*, which works with it. However, by using *vertex shaders* and *pixel shaders* we can specify how the GPU has to work with this geometry. Some works related with this topic exist in the literature, examples of which are [EJRW96][HEG04][PAC97][Sch02][Vio02].

Here we present two methods for the generation of normal maps by hardware: a very fast method for simplified models with texture correspondence (Subsection 6.2) and a method that renders the normals of the detailed model through the triangles of the coarse model (Subsection 6.3).

It must be said that after our publications a normal map generation based on the GPU appeared in [TI07]. This method works by implementing the traditional methods [nVi04a] [ATI02] on the GPU, that is, by performing a ray-tracing from the low resolution model to the high resolution model and assigning the nearest normals. Our methods, however, are based on ideas that are easier to implement. Moreover, the times of the method presented in Subsection 6.3 do not depend on the resolution of the normal map.

6.2 Fast GPU-based normal map generation for simplified models with texture correspondence

Here we present a very fast hardware generation of normal maps that uses *vertex* and *pixel shaders*. This idea involves a real-time normal map generation of the object. This method is very fast and easy to implement.

It has to be considered that the presented method can present a more realistic aspect of the object. This is due to the fact that software methods perform operations to calculate the normal and the presented method apply the real normal value. The difference in quality between the presented method and the existing methods can be assessed as being almost negligible.

Although in this method two restrictions have to be accomplished:

- Texture coordinates have to be distributed so that the texture should be correctly applied to both models. That is, the same texture can be applied with good appearance in both models. Moreover, the texture should occupy the whole texture space domain. That is, empty texture space between different regions of the texture is not allowed.

- Two or more triangles should not share the same texel, otherwise the colors generated for the normal map would superpose. However, this requirement is studied in the literature [IC01][WB98], making emphasis on the method presented in [SSGH01].

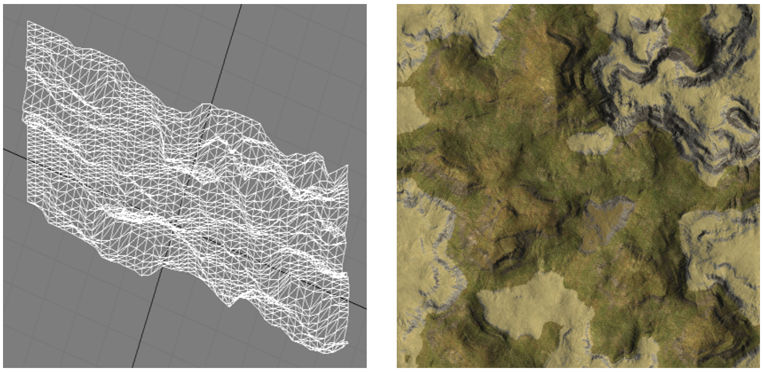

(a) (b)

Figure 6.2: A detailed terrain model (a) and its texture (b)

In order to accomplish the first restriction the texture has to be expanded occupying the whole texture space domain, without holes. Moreover, the original texture coordinates must be valid for texturing correctly the simplified model with the same texture. This method works perfectly for terrains and walls, because these objects usually meet the requirements. An example is shown in Figure 6.2. In this Figure we can see a detailed three-dimensional terrain model and its texture. If the simplified versions of this model maintain the original texture coordinates, the texture can be applied correctly to all the simplified versions, even if the simplified model is a plane made up of two triangles.

6.2.1 Description of the method

Unlike the majority of the normal map generation methods, the presented method generates the maps by *hardware*, by making use of *vertex* and *pixel shaders*. The idea is to generate the normal map of the high resolution model in order to assign it to the low resolution model so that it takes on the aspect of a more detailed object without increasing its geometry. The normal map will be generated in the world space.

So, in order to generate a normal map with this method, a *vertex shaders* and a *pixel shaders* will be used.

The enabled *shaders* here perform the following:

- The *vertex shader* flattens the image, so it transforms the coordinates of each vertex depending on the texture coordinates. In other words, it is taken as coordinates x=u, y=v, z=0, where (x,y,z) are the coordinates of the object, and (u,v) are the coordinates of the texture. Moreover, it passes the normal through the *pipeline*. The pseudo-code with these instructions is shown in Algorithm 5.

- The *pixel shader* generates the normal map so that it passes the normal coordinates to the RGB components of the resulting color in this pixel. For this purpose, it is necessary to convert the normal values into the accepted range by the RGB plane, that is, [0,1]. Algorithm 6 shows the pseudo-code of this *pixel shader*.

Algorithm 5 *Vertex shader* for normal map generation

```
$result.pos.x = in.texcoord.u$;
$result.pos.y = in.texcoord.v$;
$result.pos.z = 0$;
$result.normal.x = in.normal.x$;
$result.normal.y = in.normal.y$;
$result.normal.z = in.normal.z$;
```

Algorithm 6 *Pixel shader* for normal map generation

```
$normal = in.normal.range(0, 1)$;
$result.color.r = normal.x$;
$result.color.g = normal.y$;
$result.color.b = normal.z$;
```

The result is directly stored in a texture so that it may be applied as a normal map.

Although some methods exist that work by *hardware* in order to apply the normal map to the low resolution model, this process is here commented. When the normal map has been generated, the created texture is applied to the low resolution model. This map will represent the virtual direction of the surface at each point. This could be implemented with two shaders that work as follows:

- The *vertex shader* transforms the position of each vertex with the actual transformation matrix (”Model-view-projection”, MVP) and passes the texture coordinates (normal map) to the *pipeline*, and the position of each vertex is transformed by the model transformation matrix (M) to calculate the lightning in the *pixel shader*. Algorithm 7 shows the pseudo-code of this vertex shader.

- The *pixel shader* obtains the color of each texture point, given by the normal map. This normal map will be applied to the object. Moreover, the *pixel shader* calculates the light direction, subtracting to the light position the *fragments* position (candidate points as being pixels) of the object. Finally, it converts the normals into the range [-1,1] and the scalar product between the light direction and the normal is calculated to illuminate the object. Algorithm 8 shows the pseudo-code of this *pixel shader.*

Algorithm 7 *Vertex shader* for normal map application

```
$result.pos = MVP * in.pos;$
$result.mpos = M * in.pos;$
$result.texcoord = in.texcoord;$
```

Algorithm 8 *Pixel shader* for normal map application

```
$u = in.texcoord.u;$
$v = in.texcoord.v;$
$colortex = normalmap[u, v];$
$light.dir = in.mpos - in.light.pos;$
$light.dir.normalize();$
$normal = colortex.range(-1, 1);$
$color = light.dir^{normal};$
$color = color.range(0, 1);$
$result.color = color;$
```

6.2.2 Results

The presented method has been tested with some 3D models. The expected results were obtained, so by using an object without a highly complex mesh, an image of the object with a more detailed appearance is displayed.

The obtained times are not comparable with those of present-day *software* methods, since a few milliseconds are taken by the presented method to generate the corresponding normal map.

The times with *ATI NormalMapper* [ATI02], *nVidia Melody* [nVi04a], and the presented method using the tested models are shown in Table 6.1. Next, several images are shown in order to compare the quality of this method with the ATI's and nVidia's methods. For this purpose, the *Tarrasque* model at two levels of detail (725 and 6 117 polygons) has been used.

Figure 6.3 shows the meshes of both the low resolution and high resolution models, and the high resolution model rendered.

Figure 6.4 displays the normal map of *Tarrasque* model generated by our method and the low resolution model with the normal map of the high resolution model created by our method. Figures 6.5 and 6.5 display the normal maps generated by ATI's and nVidia's methods and the corresponding low resolution models with these normal maps applied.

The quality of our method is similar to that of the ATI's and nVidia's methods, as seen in the images.

Moreover, terrain and wall objects have been tested. The method works perfectly for this kind of objects, because they usually meet the requirements of this method. We show an example with *Crater* object. Figure 6.7 displays the high resolution model (199 126 polygons), the low resolution model (9 079 polygons) and the plane meshes. Figure 6.8 displays the normal map of the high resolution model of *Crater*. And Figure 6.9 shows the renders of the high resolution model and both the low resolution model and the plane with the normal map applied.

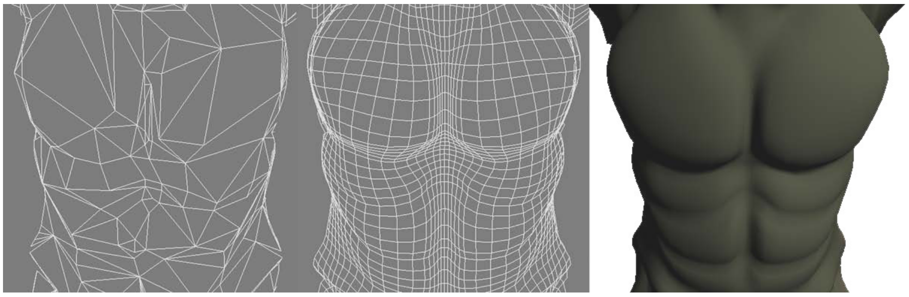


Figure 6.3: Low and high resolution model meshes of *Tarrasque* with the rendered model

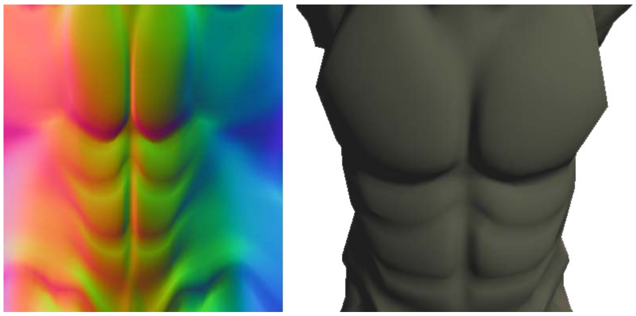


Figure 6.4: Normal map of the high resolution model generated by our method and the low resolution model with it applied

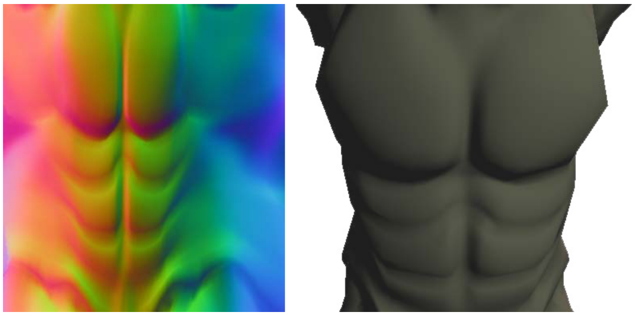


Figure 6.5: Normal map of the high resolution model generated by ATI's method and the low resolution model with it applied

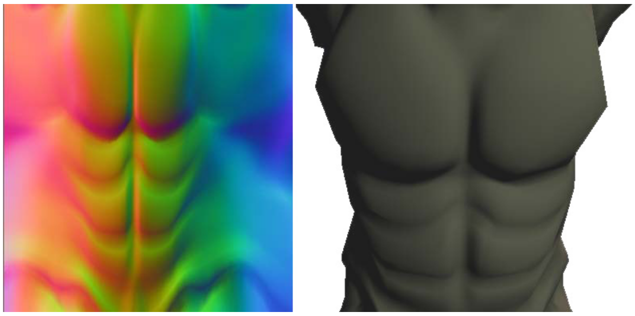


Figure 6.6: Normal map of the high resolution model generated by nVidia's method and the low resolution model with it applied

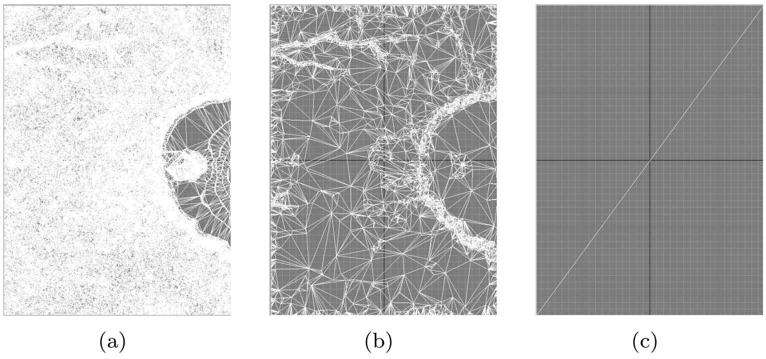


Figure 6.7: High (a) and low (b) resolution model meshes of *Crater* and a plane object (c)

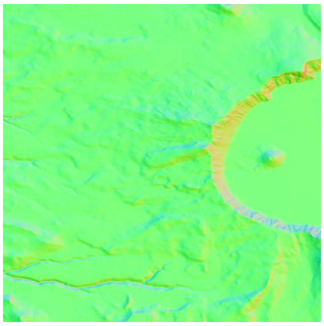


Figure 6.8: Normal map of the high resolution model of *Crater*

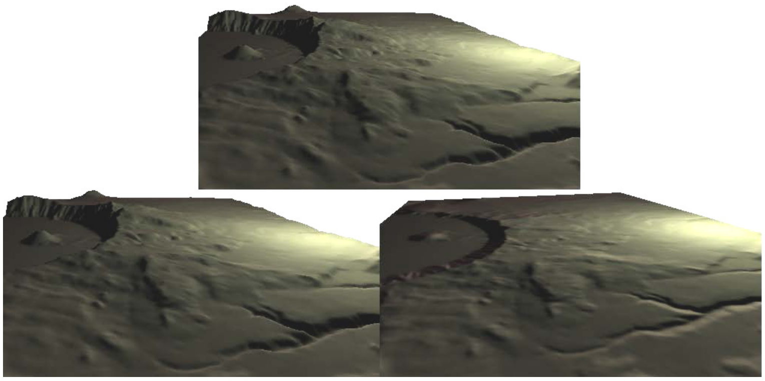


Figure 6.9: Renders of the high resolution model (above), low resolution model with the normal map applied (left bottom) and the plane with the normal map (right bottom)

Polygons of high res model	Time of our method	Time of ATI	Time of nVidia
1696	0.08	16850	16306
7910	0.53	24125	50589
48048	4.72	97704	160975
61644	6.02	129969	179569

Table 6.1: Table of times in milliseconds of the normal map generation

6.3 Generalized GPU-based normal map generation

The method presented here generates the normal map completely on the GPU, so that all the work is performed by the graphics hardware. This has the advantage of taking profit of a highly parallelizable hardware which will quickly increment its performance in the next years. Coarse versions of highly detailed meshes are often modeled from scratch (as in the video game industry for example), so we can not expect any correspondence between the two meshes other than geometric proximity. Having this in mind, this method avoids the requirement of generating the coarse mesh from the highly detailed mesh, and, as a consequence, they can be modeled separately. Therefore, the restriction about texture correspondence in the method presented in Section 6.2 is not a requirement for this method.

6.3.1 Description of the method

We assume that the simplified version of a model is a good approximation of it and that both are situated in the same spatial position, and have the same sizes and orientation. To calculate the normal map the high resolution model will be rendered for each triangle (T) of the low resolution model. The high resolution model will be rendered through the triangle T of the low resolution model in the inverse direction of the normal of the triangle. We will use the stencil buffer in order to discard all those parts of the model projected outside of T.

For every fragment of the high resolution model projected through T, the direction of the normal will be extracted and stored into the normal map. In order to store this information, a transformation matrix must be calculated at each render (detailed in this subsection). This matrix will transform the triangle T to the two-dimensional triangle t that is formed by the texture coordinates of T.

The algorithm is performed until all the triangles of the low resolution model have been used to render the high resolution model and the information about the normals for them have been stored.

Easily explained, the algorithm works as follows: the normals of all those pixels of the high resolution mesh rendered through the window formed by the triangle T will become the part of the normal map applied to T.

The only requirement of the method is that the texture coordinates of the high resolution model must be fully unwrapped in a way that there are no triangles overlapped in texture space.

Transformation matrices

For each iteration of the algorithm, a transformation matrix (which encapsulates the model, view and projection transformations) must be calculated. This matrix transforms the triangle T (composed by the vertices v_0, v_1 and v_2)

to the two-dimensional triangle t (composed by the texture coordinates of T: t_0, t_1 and t_2). Figure 6.10 shows this process. Once obtained, this matrix will be applied to every vertex of the high resolution model, so that all the triangles visible through T will be projected onto the area defined by t.

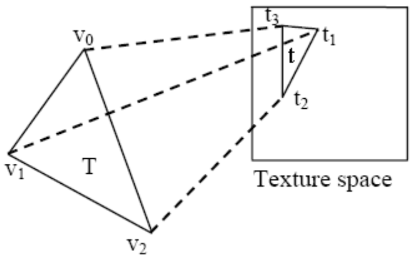


Figure 6.10: The transformation matrix which converts the 3D triangle T into a 3D triangle composed by the texture coordinates of T must be calculated at each step

Model/View matrix calculation The model/view matrix (MV) is derived from the three parameters that define a virtual camera which will be configured so that its viewing direction is parallel to the normal of T, looking to the center of the triangle and located at a certain distance of T. As we will use an orthographic projection, the distance T will not affect the final projection.

To define a virtual camera, a third parameter is needed: the roll angle usually specified as the up vector. This vector is calculated using its texture coordinates.

Let (t_1, t_2, t_3) be the texture coordinates of the three vertices of $T(v_1, v_2, v_3)$. The vertical value on our two-dimensional space will be assumed to be the vector (0,1) (see Figure 6.11). Having this in mind we can propose the following equation system:

$$\begin{aligned} 0 &= \alpha(t_x^2 - t_x^1) + \beta(t_x^3 - t_x^1) \\ 1 &= \alpha(t_y^2 - t_y^1) + \beta(t_y^3 - t_y^1) \end{aligned} \tag{6.1}$$

Working out the values of α and β will let to calculate the desired vector in the following way:

$$\vec{UP} = \left\| \alpha(\vec{v^2} - \vec{v^3}) + \beta(\vec{v^3} - \vec{v^1}) \right\| \tag{6.2}$$

Projection matrix We will use a pseudo-orthogonal projection matrix to project the vertices of the high resolution model. Similar to an orthogonal matrix, it will not modify the X and Y coordinates depending on the value of Z. However,

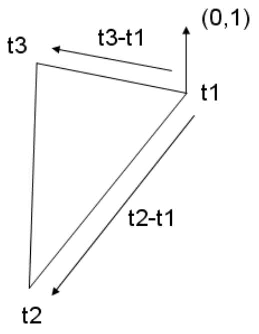


Figure 6.11: Calculation of the α and β parameters in order to obtain the up vector

its behavior is not exactly the same of a common orthogonal matrix, as we will explain later.

We need to calculate a projection matrix (P) which transforms the coordinates of T into its texture coordinates t, so we propound the following equations:

$$Pv^i = t^i \qquad \forall i \in \{0,1,2\} \tag{6.3}$$

The problem here is that the matrix P is a homogeneous transform matrix (it has 16 elements), and thus it can not be solved directly because we have not enough equations.

As we are looking for a pseudo-orthogonal matrix which transforms the X and Y coordinates of each vertex to the texture coordinates, our calculations are based on the equation system of a orthogonal projection matrix. This matrix is given by

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.4}$$

where f, n, r, l, t and b give the parameters of the frustrum.

Therefore, we calculate the value of the unknowns (Pn) shown in equation 6.5.

$$P^{'} = \begin{pmatrix} P_A & P_B & 0 & P_C \\ P_D & P_E & 0 & P_F \\ 0 & 0 & -\dfrac{2}{f-n} & -\dfrac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} w_x^i \\ w_y^i \\ w_z^i \\ 1 \end{pmatrix} = \begin{pmatrix} Nt_x^i \\ Nt_y^i \\ 0 \\ 1 \end{pmatrix} \tag{6.5}$$

where f and n give the distance from the eye to the far and near clipping planes, i refers to each one of the three vertices of T, and w^i refers to those vertices pre-multiplied by the MV matrix, as shown below:

$$MV\vec{v^i} = \vec{w^i} \qquad \forall i \in \{1,2,3\} \tag{6.6}$$

Nt_x^i and Nt_y^i are the normalized device coordinates for each one of the transformed vertices. As after perspective division, the visible coordinates in the screen are located in the range [-1,1]. Nevertheless, we want them to be in the range [0,1] of the texture coordinates, then we have to take that into account. They are calculated using the following formula:

$$(Nt_x^i, Nt_y^i) = 2(t_x^i, t_y^i) - 1 \qquad \forall i \in \{1,2,3\} \tag{6.7}$$

After solving the equation system 6.5 we get the desired pseudo-orthogonal matrix which projects every vertex of the high resolution model in a way that all the triangles visible through T will be rendered inside the area formed by the texture coordinates of T.

Framebuffer

As explained before, the number of renders of the high resolution model needed to be performed is equal to the triangle count of the low resolution model. However, we are only interested in the pixels which are inside the area formed by the triangle T projected with the matrix MVP. Setting up the stencil buffer to discard all pixels which are outside the projection of T, is a very simple way to discard unwanted pixels and to protect those parts which have been already rendered.

Finally, the pixel shader rescales every unmasked normal to the range [0,1], so that the normal vector can be encoded as a color in the RGB space.

The auto-occlusion problem

Sometimes, there are some parts of the models that will cause this method to fail. This happens when there is another part of the model between the camera and the real target surface. This problem is clearly shown in Figure 6.12, where a part of the model is incorrectly occluding the desired surface (colored in red), causing the normal map to be completely invalid.

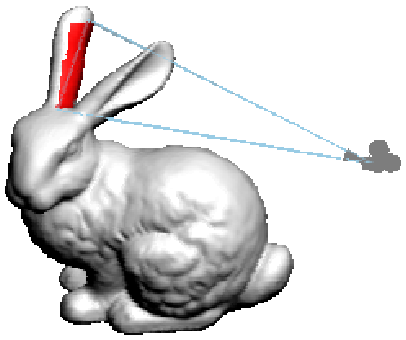


Figure 6.12: Auto-occlusion problem: the ear next to the camera is occluding the real desired geometry

To solve this problem we have developed a technique called *vertex mirroring*. Basically we consider that if a pixel is going to be drawn more than once (some parts of the model overlap in screen space), then the valid pixel will be that one which is closer to T. This is similar to what raytracing-based normal map generation algorithms do: if some polygons intersect the ray, take the one which is closer to T.

Let Π be the plane containing T. Let N be the normal of Π, v^i be each one of the vertices of the high resolution model and k^i be the distance between Π and v^i. Then the final position of v^i is recalculated as follows:

$$\vec{v^i} = \vec{v^i} - 2clamp(k^i, 0, k^i)\vec{N} \qquad (6.8)$$

The function $clamp(a, b, c)$ will trunk the value a inside the range $[b, c]$. This ensures that all vertices of M_{HR} are in front of the plane Π, because those vertex that are behind that plane are mirrored through it. After performing this step, we can use the standard depth test to ensure that each rendered pixel is the nearest possible to T.

This technique can be implemented in a vertex shader for optimal performance, in a clear, elegant and efficient way.

Normal map border expansion

Once the previous process is over, the normal map is correctly calculated. However, due to the texture filtering methods used in real-time hardware, normals could incorrectly interpolate with their neighboring "‘empty"’ texels. To solve this problem, we need to create an artificial region surrounding every part of the normal map.

To detect those texels that do not contain a valid normal, an extra pass rendering a full screen quad textured with the previously generated normal map will be performed. For each pixel, the pixel shader of the normal map generator will check if the texel belonging to the pixel being processed has a module lower than 1, which means that it does not contain a valid normal (because all normal must be unitary). If that happens, the pixel must be filled with the average normalized value of its neighboring texels which contain a valid normal.

At the end of the process a 1-pixel sized frontier is created around all parts of the normal map that did not contain a valid normal. This process can be repeated with the resulting texture to expand the frontier to a user defined size.

6.3.2 Results

All the tests were performed on an Intel Pentium 4 3GHz, 2GB RAM with a nVidia GeForce 9800GTX+ and can be divided into two categories: performance and quality tests. Table 6.2 shows a study of total times required to generate the normal maps for two models with different polygonal complexity. For each model (HRM, first column) different coarse approximations are used (LRM, second column) to generate the normal map. The column on the right shows the time in milliseconds needed to calculate the normal map for a certain combination of meshes.

Table 6.2: Total times in milliseconds needed to generate the normal maps for two different models, using a set of different coarse approximations

Triangles HRM	*Triangles LRM*	*Time (ms)*
	250	77,05
	500	146,85
69 451	1 000	276,72
	2 000	558,91
	2 500	682,51
	3 000	848,53
	250	15,14
	500	31,89
16 843	1 000	59,13
	2 000	101,47
	2 500	119,18
	3 000	145,34

Furthermore, some tests have been done to compare our results with a software based approach, which is implemented in the nVidia Melody tool. These results are not shown in a table because that tool does not display time information. However, the times needed for the application to calculate the normal maps for the same high resolution model used in our tests vary from 2 to 6 seconds for the better and worst cases. These times are worse than our results.

The other studied parameter is how the size of the normal map affects to our method. As one can imagine, the bottleneck of our application is the huge amount of vertices needed to be processed, because the pixel operations performed are very light weight. Table 6.3 shows how our method is independent of the size of the normal map.

Figure 6.14 shows the results of the generated normal maps for 2 different models: the bunny and the monster. The column on the left shows the high resolution models (HRM). The column on the centre shows the coarse versions of each mesh (LRM) used to calculate the normal map. Finally, the column on the right shows the final normal map applied to the coarse mesh, so one can check the final visual quality of the normal map. Figure 6.13 shows the normal maps generated for its use in Figure 6.14.

Table 6.3: Generation times at different resolutions

Triangles HRM / LRM	***Resolution***	***Time (ms)***
	128x128	442
69 451 / 1 000	512x512	441
	1 024x1 024	443

6.4 Conclusions

In this Chapter, we have presented two GPU-based methods for normal map generation. A normal map is a two-dimensional image with the information about the normals of a three-dimensional model. By applying a normal map of a detailed model to a simplified version of it, more detail is obtained without needing more geometry. The first presented method (Subsection 6.2) is a very fast method and easy to implement. It is useful for simplified models with a texture correspondence. The second presented method (Subsection 6.3) renders the normals of the detailed model through the triangles of the simplified model.

The first method presented here is faster than the second one due to the number of renders needed in the second method. However, the second method avoids the requirement of the texture correspondence that is needed to be accomplished in the first method.

As seen before, the second presented method (Subsection 6.3) is highly dependent of the number of triangles of both models (the original and the simplified ones). However, this limitation can be reduced by using some kind of hierarchical culling method to discard most of the unneeded geometry. Moreover, reducing the number of renders needed by grouping triangles of low resolution model would also be possible. This would highly accelerate the total generation times, although this has been left as future work.

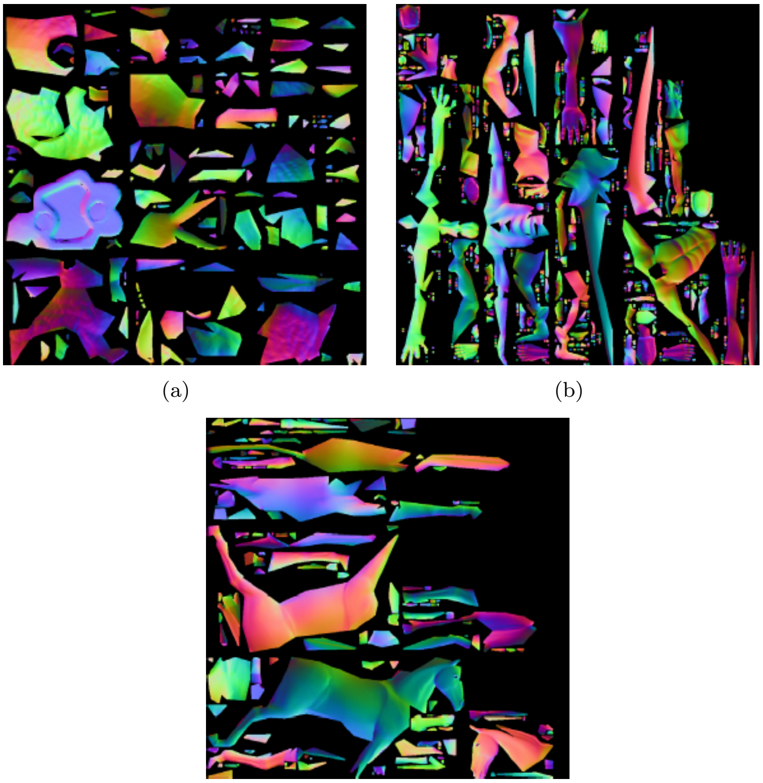

(a) (b) (c)

Figure 6.13: Normal maps of the bunny (a) and monster (b) models

These methods exploit the graphics hardware in a way that it takes advantage of the parallelization of the GPU in various ways. On the one hand, the graphics hardware uses several shading processors in parallel, which is inherent to the graphics pipeline. On the other hand, there is a parallelization between the GPU and the CPU, which is useful to calculate matrices on the CPU while the GPU is performing each render.

Although these two GPU-based methods for normal map generation have been used to calculate the normals of a high resolution polygonal mesh, they could also be used to obtain other surface parameters such as diffuse color maps, height maps or specular maps.

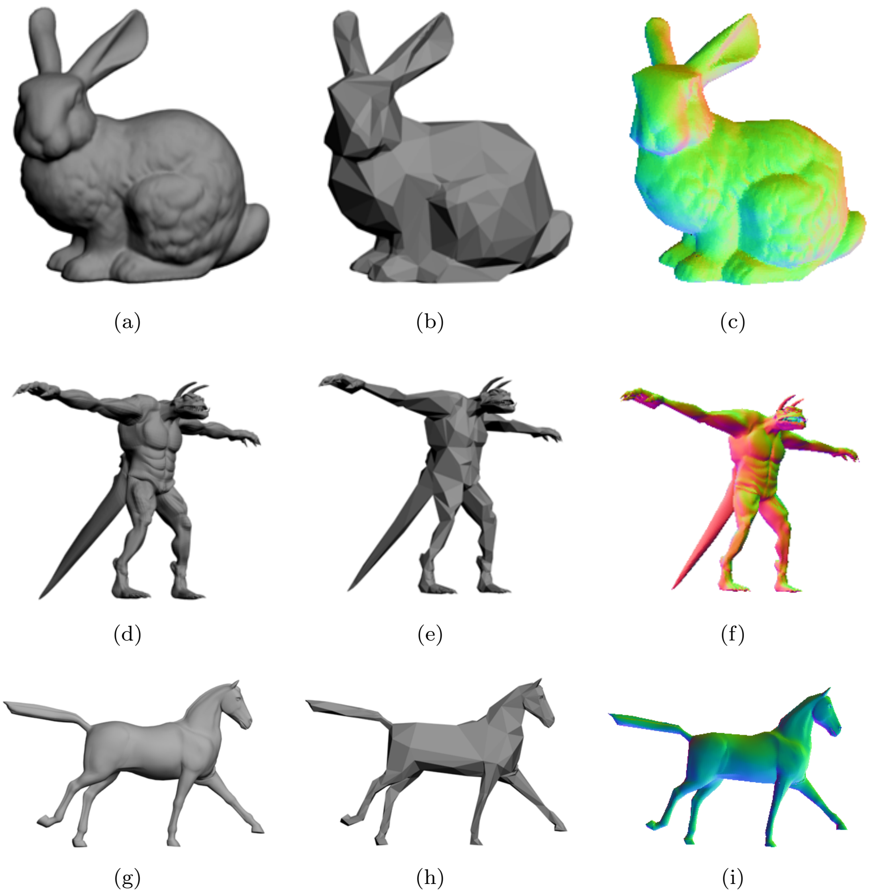


Figure 6.14: The column on the left show the high resolution models (HRM). The column in the middle shows the coarse versions (LRM) of those meshes. Finally, the column on the right show the resulting normal map applied to the low resolution model (LRM)

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, work on techniques related to three-dimensional objects simplification for their use in interactive applications has been presented. Interactive applications need to manage scenes with a high frame rate. Moreover, models with a good appearance are required. However, the available graphics hardware cannot always handle all the geometry in the scene. One of the solutions to this problem is the use of simplification methods ([CMS98] [Lue01]). These methods reduce the geometric complexity of the models, attempting to preserve the original appearance. Therefore, the main aim of this thesis is to present new techniques in the area of mesh simplification for interactive applications.

A state-of-the-art relative of work related to that presented in this thesis was presented in Chapter 2. Many papers have been published about simplification, but there are still several problems without efficient solutions. Therefore, in this thesis several solutions for the simplification of the models usually used in interactive applications have been presented, such as a user-assisted efficient simplification for meshes obtained from a computer-aided design process, the preservation of additional attributes to the geometry of the models and the acceleration of methods making use of the latest graphics hardware advances.

Models obtained after a computer-aided design process are usually composed of different interconnected subobjects. Simplifying this kind of model with a simplification method that does not consider these characteristics can produce simplified models with holes and great distortions. Moreover, simplification tools do not present an automatic and efficient process to work with meshes generated from CAD models with the possibility of simplifying the subobjects to different percentages of simplification. It is usually an elaborate process and these tools do not always use the best error metrics. This process is usually done by hand by the designers. Therefore, in Chapter 3 we presented a new user-assisted simplification method for models obtained from a computer-aided design process.

The method presented here is user-assisted because the user can independently manage the level of simplification of each subobject, by simplifying them more or adding more detail to the simplified ones. Moreover, the user can also demand a total number of triangles in the simplified object. Therefore, when the level of simplification of a subobject is changed by the user, other subobjects will automatically be modified in order to maintain the total number of triangles. This is performed while preserving the boundaries between the subobjects. Therefore, the method will avoid the appearance of holes in the simplified object. Normals and texture coordinates are also taken into account in this method.

Thus, the method presented in Chapter 3 solves an existing problem with the meshes obtained after a computer-aided design process, presenting efficient simplifications of this kind of model.

Geometry is not the only factor that is important for the final appearance of the models. Interactive applications usually use models with additional attributes, such as textures. Textures play a very important role in the final appearance of the objects for the viewer. There are a lot of simplification methods that do not consider the texture information. Therefore, simplified objects can appear greatly distorted when textures are applied to them. In Chapter 4 we presented an error metric extension that is useful for taking texture information into account in those simplification methods that do not already consider it. This extension enables methods that do not consider texture information to easily take it into account. More accurate textured models are obtained by taking textures into account compared to the methods that do not consider this information. In the presented results obtained by applying our extension, great visual improvements can be appreciated compared with the original methods that do not consider the texture information in their error metrics. A quantitative study has also been done with a well-known metric: RMSE [LT00]. The consideration of this work for the preservation of other attributes, such as normals, has also been tested, presenting also better visual results by using our extension.

Moreover, textures are also considered in the method presented in Chapter 5. This is a simplification method that takes texture information into account in its error metric and its speed-up using new graphics hardware technologies. Section 5.2 presents a brand new simplification method based on the segmentation of the texture information. In this method, well-known mathematical concepts from Information Theory have been used. These concepts enabled the development of a robust simplification method that preserves textures. In the obtained results it can be observed that textures are maintained, producing simplified models with a similar appearance to the original ones for the viewer. These results have also been measured with the RMSE [LT00]. This quantitative study shows that great improvements in the visual results are obtained by using our method.

The visual appearance of the models is not the only important factor. Sometimes the time employed by some methods can also be important. Viewpoint-driven simplification methods are usually slow. The last graphics hardware advances enable some graphic applications to be sped up by programming code that will be directly executed by the GPU. Therefore, in Section 5.3 we implemented and tested techniques to speed up a viewpoint-driven simplification method, like the one presented in Section 5.2, using CUDA. We can see in the results that better times than those obtained with methods implemented only in the CPU are obtained by using these graphics hardware advances.

A common technique employed as a post-process of the simplification methods is the use of normal maps. A normal map is a two-dimensional image with the information about the normals of a model. By applying the normal map of a detailed model to a simplified version of it, more detail is obtained without needing more geometry. In Chapter 6 we made use of the new advances in graphics hardware and two different GPU-based normal map generation methods were presented. The time taken by some methods can be important to the users. The first method (Section 6.2) is very fast and it is useful for models with a texture correspondence. It is very fast and easy to implement. The second method (Section 6.3) renders the normals of the detailed model through the triangles of the simplified model and avoids the correspondence requirement of the first method. Both methods are fast and present good results. These methods can also be used for obtaining other surface parameters such as diffuse color maps, height maps or specular maps. We improve the times of methods implemented on the CPU, obtaining normal maps with a similar appearance and quality.

7.2 Future work

In this section we suggest and discuss some of the possibilities of future research related to this thesis. Simplification techniques are continuously improving, due to the appearance of new techniques and GPU advances.

The parallelization of tasks has been an object of research study for some years. Therefore, the study of parallelization of methods like the one presented in Chapter 3 could be possible. This method is a user-assisted simplification method for models with the properties of the objects usually obtained in a computer-aided design process. We suggest the study of parallelizing the simplification of the different subobjects of the models by, for example, simulating a distributed system.

Additional attributes to the geometry, like textures and normals, are very important in the final appearance of the models commonly used in interactive applications. We presented an error metric extension useful for considering the texture information in those simplification methods that do not already take this information into account. We used the areas of the triangles as main concept in order to penalize an edge by a greater or lesser amount. This gives good results, as shown in Section 4.4. We have also tested this metric extension for the preservation of other attributes, like normals. We can see in the results presented in Section 4.4.2 that this extension is also useful and produces good visual results for the normals preservation. However, we suggest the study of new mathematical possibilities, based on well-known mathematical concepts (like in the method presented in Chapter 5), in order to compare the extension with other mathematical possibilities. More mathematical concepts could also be tested in methods that consider textures in their error metric, like the one presented in Chapter 5.

Moreover, another suggestion for study in techniques of texture preservation is to consider the possibility of mixing the texture coordinates and the normals values in order to determine the error cost associated with each edge. This could also be oriented to considering another attribute (such as colors) without taking the other ones into account, or to obtaining a combination of all the attributes information in order to obtain an error metric that considers a mixture of all this information.

In order to measure the visual improvements obtained with our texture-preserving simplification techniques, we have used a well-known metric: RMSE [LT00]. This metric, however, may not always be the best metric for some specific cases. This is because it can greatly penalize some models that preserve a good appearance (also with accurate textures) for the viewer, because of the disappearance of some parts of their silhouette. Therefore, we propose the study of new metrics for the visual perception of the models for the user.

The time taken by some methods may also be important. Viewpoint-driven simplification methods, like the one presented in Section 5.2 are usually slow, due to the number of renders that they need to perform. Therefore, in Section 5.3 we presented a speed-up for this kind of simplification methods. Graphic cards are in continuously improving. Therefore, another suggestion for future work is to study the possibilities of the future graphics card generations in order to obtain a more efficient speed-up for similar methods.

Normal maps can help to give more detail to the simplified models without adding more geometry. With these methods, images with the information of the normals of original models are obtained. The application of these images to the simplified versions of the models will give more details to the simplified models. Graphics hardware advances enable us to speed-up and parallelize several tasks that were once only performed on the CPU. Therefore, in Chapter 6 two GPU-based normal map generation methods were presented. We suggest the study of GPU-based methods that generate maps with information of other attributes, like color maps or height maps. Another suggestion of study is the use of the CUDA architecture for the acceleration of this kind of methods. By using CUDA some tasks of the normal map generation could be parallelized on the GPU.

7.3 Publications and Research Projects

7.3.1 Publications

The publications that support the contributions of this thesis are:

- Journals
  - Carlos González, Jesús Gumbau, Miguel Chover, Francisco Ramos, Ricardo Quirós. User-assisted simplification method for triangle meshes preserving boundaries. Computer-Aided Design, 41(12):1095-1106. 2009. ISSN: 0010-4485.
  - Carlos González, Pascual Castelló, Miguel Chover, Mateu Sbert, Miquel Feixas. Segmentation-based Simplification Method for Textured Polygonal Meshes. Submitted to Computer Graphics Forum.
- Conferences
  - Carlos González, Pascual Castelló, Miguel Chover, Mateu Sbert, Miquel Feixas. Viewpoint Entropy-driven Simplification Method for Triangle Meshes. Proceedings of the Fifth International Conference on Computer Graphics Theory and Applications. GRAPP '10. 2010. pp. 30-37. ISBN: 978-989-674-026-9.
  - Carlos González, Jesús Gumbau, Miguel Chover, Pascual Castelló. Mesh Simplification for Interactive Applications. Proceedings of the 16th International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision. WSCG '08. 2008. pp. 87-94. ISBN: 978-80-86943-16-9.
  - Jesús Gumbau, Carlos González, Miguel Chover. GPU-Based Normal MAP Generation. Proceedings of the Third International Conference on Computer Graphics Theory and Applications. GRAPP '08. 2008. Insticc. pp. 62-67. ISBN: 978-989-8111-20-3.
  - Carlos González, Pascual Castelló, Miguel Chover. A Texture-Based Metric Extension for Simplification Methods. Proceedings of the Second International Conference on Computer Graphics Theory and Applications. GRAPP '07. 2007. Insticc. pp. 69-76. ISBN: 978-972-8865-71-9.

  - Carlos González, Jesús Gumbau, Miguel Chover, Pascual Castelló. Simplificación de mallas para juegos. XVII Congreso Español de Informática Gráfica. CEIG '07. 2007. Thomson. pp. 19-27. ISBN: 978-84-9732-595-0.
  - Jesús Gumbau, Carlos González, Miguel Chover. Fast GPU-based normal map generation for simplified models. Proceedings of the 14th International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision. WSCG '06. 2006. pp. 15-16. ISBN: 80-86943-04-6.
  - Carlos González, Jesús Gumbau, Miguel Chover. Generación por hardware de mapas de normales. XV Congreso Español de Informática Gráfica. CEIG '05. 2005. Thomson. pp. 281-284. ISBN: 84-9732-431-5.

- Book chapters
  - Carlos González, Jesús Gumbau. Introducción a OpenGL 2.0. OpenGL en Fichas II: Aspectos Avanzados. Universitat Politècnica de València. 2008. ISBN: 978-84-8363-352-6. pp. 167-174.
  - Carlos González, Jesús Gumbau. Introducción a GLSL. OpenGL en Fichas II: Aspectos Avanzados. Universitat Politècnica de València. 2008. ISBN: 978-84-8363-352-6. pp. 175-182.
  - Carlos González, Jesús Gumbau. Ejemplos completos de Shaders. OpenGL en Fichas II: Aspectos Avanzados. Universitat Politècnica de València. 2008. ISBN: 978-84-8363-352-6. pp. 205-212.

7.3.2 Research Projects

The participation in research projects related to this thesis work is:

- VISUALCAD: desarrollo de un optimizador de diseños modelados en sistemas CAD para su visualización en tiempo real (FIT-350101-2004-15). MINER (PROFIT).
- GameTools: Advanced Tools for Developing Highly Realistic Computer Games. European Union (IST-2-004363).
- Interfaces Avanzadas para Campos de Luz Aumentados. Spanish Ministry of Education and Science (TIN2005-08863-C03-03).
- Nueva generación de aplicaciones para la gestión de riesgos medioambientales: Gestión de incendios con infraestructuras de datos espaciales con servicios de geoprocesamiento, visualización avanzada. Caja Castellón-Bancaja Foundation (P1-1B2009-34).

- Campos de luz avanzados para visualización autoestereoscópica interactiva 3D. Spanish Ministry of Education and Science (TIN2009-14103-C03-01)
- EUROGEOSS: A EUROPEAN APPROACH TO GEOSS. European Union (ref. 226487).

Bibliography

[ABA02] C. Andújar, P. Brunet, and D. Ayala. Topology reducing surface simplification using discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, 2002.

[ATI02] ATI. Normal mapper tool. www.ati.com/developer/tools.html, 2002.

[Bly06] David Blythe. The direct3d 10 system. *ACM Transactions on Graphics*, 25(3):724–734, 2006.

[BP03] Dmitry Brodsky and Jan Baekgaard Pedersen. A parallel framework for simplification of massive meshes. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, page 4, Washington, DC, USA, 2003. IEEE Computer Society.

[Bro00] Dmitry Brodsky. Model simplification through refinement. In *Proc. of Graphics Interface*, pages 221–228, 2000.

[Can83] John F. Canny. A variational approach to edge detection. In *AAAI*, pages 54–58, 1983.

[Can86] John F Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[CC06] Chin-Chun Chen and Jung-Hong Chuang. Texture adaptation for progressive meshes. In *EUROGRAPHICS*, volume 25, pages 343–350, 2006.

[CCMS97] Andrea Ciampalini, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13:228–246, 1997.

[CMS98] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22:37–54, 1998.

[CMSR98] Paolo Cignoni, Claudio Montani, Roberto Scopigno, and Claudio Rocchini. A general method for preserving attribute values on simplified meshes. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 59–66, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[COM98] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122, New York, NY, USA, 1998. ACM.

[CSCF07] Pascual Castelló, Mateu Sbert, Miguel Chover, and Miquel Feixas. Viewpoint entropy-driven simplification. In *WSCG '07: Proceedings of 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 249–256, 2007.

[CSCF08] P. Castelló, M. Sbert, M. Chover, and M. Feixas. Technical section: Viewpoint-driven simplification using mutual information. *Comput. Graph.*, 32(4):451–463, 2008.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience, New York, NY, USA, 1991.

[CVM$^+$96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1996. ACM.

[DEGN99] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S*, 66:23–45, 1999.

[DKK$^+$05] H. Date, S. Kanai, T. Kisinami, I. Nishigaki, and T. Dohi. High-quality and property controlled finite element mesh generation from triangular meshes using the multiresolution technique. *Journal of Computing and Information Science in Engineering*, 5(4):266–276, 2005.

[DKKN06] Hiroaki Date, Satoshi Kanai, Takeshi Kishinami, and Ichiro Nishigaki. Flexible feature and resolution control of triangular meshes. In *Proceedings of the sixth IASTED international conference on visualization, inmaging and image processing.* Sandia National Laboratories, 2006.

[DSSC08] Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. Quadrilateral mesh simplification. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–9, New York, NY, USA, 2008. ACM.

[DT08] Christopher DeCoro and Natalya Tatarchuk. Implementing real-time mesh simplification using the gpu. In *Wolfgang Engel. ShaderX6*. Charles River Media, 2008.

[EDD$^+$95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182, New York, NY, USA, 1995. ACM.

[EJRW96] I. Ernst, D. Jackel, H. Russeler, and O. Wittig. Hardware-supported bump mapping. *Computers and Graphics*, 20(4):515–521, 1996.

[EM98] Carl Erikson and Dinesh Manocha. Gaps: General and automatic polygonal simplification. Technical report, Chapel Hill, NC, USA, 1998.

[EsV99] Jihad El-sana and Amitabh Varshney. Generalized view-dependent simplification. In *EUROGRAPHICS*, 1999.

[FCS02] Chin-Shyurng Fahn, Hung-Kuang Chen, and Yi-Haur Shiau. Polygonal mesh simplification with face color and boundary edge preservation using quadric error metric. In *MSE '02: Proceedings of the Fourth IEEE International Symposium on Multimedia Software Engineering*, page 174, Washington, DC, USA, 2002. IEEE Computer Society.

[FH04] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 2004.

[FLM03] M. Foskey, M.C. Lin, and D. Manocha. Efficient computation of a simplified medial axis. *Journal of Computing and Information Science in Engineering*, 3(4):274–284, 2003.

[FRL00] L. Fine, L. Remondini, and J.-C. Leon. Automated generation of fea models through idealization operators. *International Journal for Numerical Methods in Engineering*, 49(1-2):83–108, 2000.

[FS01] Martin Franc and Václav Skala. Parallel triangular mesh decimation without sorting. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 22, Washington, DC, USA, 2001. IEEE Computer Society.

[FSG09] Miquel Feixas, Mateu Sbert, and Francisco González. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception (TAP)*, 6(1), 2009.

[Gar99] Michael Garland. State of the art reports multiresolution modeling: Survey & future opportunities. In *EUROGRAPHICS*, 1999.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 263–269, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[GZ05] Michael Garland and Yuan Zhou. Quadric-based simplification in any dimension. *ACM Trans. Graph.*, 24(2):209–239, 2005.

[HC67] J. Harvda and F. Charvát. Quantification method of classification processes. concept of structural alpha-entropy. *Kybernetika*, 3:30–35, 1967.

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH*, pages 19–26, 1993.

[HEG04] J. Hirche, A. Ehlert, and S. Guthe. Hardware accelerated per-pixel displacement mapping. *Graphics Interface*, 2004.

[HG97] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH*, 1997.

[HHK$^+$95] Taosong He, Lichan Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel based object simplification. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 296, Washington, DC, USA, 1995. IEEE Computer Society.

[HHVW96] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, 1996.

[HL07] Jon Hjelmervik and Jean-Claude Leon. Gpu-accelerated shape simplification for mechanical-based applications. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and*

Applications 2007, pages 91–102, Washington, DC, USA, 2007. IEEE Computer Society.

[Hop96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Hop98] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

[Hop99] Hugues Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *VISUALIZATION '99: Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, Washington, DC, USA, 1999. IEEE Computer Society.

[IC01] T. Igarashi and D. Cosgrove. Adaptative unwrapping for interactive texture painting. In *Symposium on Interactive 3D Graphics*, pages 209–216, 2001.

[IG03] Martin Isenburg and Stefan Gumhold. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 935–942, New York, NY, USA, 2003. ACM.

[IGS03] Martin Isenburg, Stefan Gumhold, and Jack Snoeyink. Processing sequences: A new paradigm for out-of-core processing on large meshes. preprint available at http://www.cs.unc.edu/ĩsenburg/oocc, 2003.

[ILGS03] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, and Jack Snoeyink. Large mesh simplification using processing sequences. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 61, Washington, DC, USA, 2003. IEEE Computer Society.

[JTY06] Bin Shyan Jong, Juin-Ling Tseng, and Wen Hao Yang. An efficient and low-error mesh simplification method based on torsion detection. *The Visual Computer*, 22:56–67, 2006.

[KG03] Youngihn Kho and Michael Garland. User-guided simplification. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 123–126, New York, NY, USA, 2003. ACM.

[KTKN05] Takayuki Kanaya, Yuji Teshima, Ken-ichi Kobori, and Koji Nishio. A topology-preserving polygonal simplification using vertex clustering. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 117–120, New York, NY, USA, 2005. ACM.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[LDW97] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Trans. Graph.*, 16(1):34–73, 1997.

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Lee05a] S.-H. Lee. A cad-cae integration approach using feature-based multi-resolution and multi-abstraction modeling techniques. *Computer-Aided Design*, 37(9):941–955, 2005.

[Lee05b] Sang Hun Lee. Feature-based multiresolution modeling of solids. *ACM Transactions on Graphics*, 24(4):1417–1441, 2005.

[LH01] David P. Luebke and Benjamin Hallen. Perceptually-driven simplification for interactive rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 223–234, London, UK, 2001. Springer-Verlag.

[LKF96] Wen-Yew Liang, Chun-Ta King, and Lai Feipei. Adsmith: An efficient object-based distributed shared memory system on pvm. In *Porceedings of the 1996 International Symposium on Parallel Architectures*, pages 173–179, 1996.

[LL06] S.-H. Lee and K. Lee. Feature-based multi-resolution techniques for product design. *Journal of Zheijang University - Science A*, 7(9):1535–1543, 2006.

[LLK06] S.-H. Lee, K. Lee, and S.-C. Kim. History-based selective tanfor operations for feature-based multi-resolution modeling. In *Proceedings of ICCSA*, 2006.

[LRC$^+$02] David Luebke, Martin Reddy, Jonathan D Cohen, Amitabd Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for*

3D graphics. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[LRD99] Christian Langis, Gerard Roth, and Frank Dehne. Mesh simplification in parallel. pages 281–290, 1999.

[LT97] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff., New York, NY, USA, 1997. ACM.

[LT00] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Trans. Graph.*, 19(3):204–241, 2000.

[Lue01] David P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Comput. Graph. Appl.*, 21(3):24–35, 2001.

[Lun91] Charles Lund. The supermesh: Run-time meshing from coarse cad models using msgmesh. In *Proceedings of the MSC 1991 World Users' Conference*, number 42, March 1991.

[LVJ05] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. *ACM Transactions on Graphics*, 24(3):659–666, 2005.

[NT03] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[nVi04a] nVidia. nvidia melody user guide.
http://developer.nvidia.com/object/melody_home.html, 2004.

[nVi04b] nVidia. Pixel buffer object specification.
www.opengl.org/registry/specs/arb/pixel_buffer_object.txt, 2004.

[nVi07] nVidia. Histogram calculation in cuda.
http://developer.download.nvidia.com, 2007.

[nVi09a] nVidia. Compute unified device architecture (cuda) programming guide.
http://developer.nvidia.com/object/cuda.html, 2009.

[nVi09b] nVidia. Cuda gpu occupancy calculator.
http://developer.download.nvidia.com, 2009.

[PAC97] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 303–306. ACM, 1997.

[PS97] Enrico Puppo and Roberto Scopigno. Simplification, lod and multiresolution - principles and applications. In *EUROGRAPHICS*, volume 16, 1997.

[Ŕ61] A. Rényi. On measures of entropy and information. In *Proc. Fourth Berkeley Symp. Math. Stat. and Probability*, page University of California Press, 1961.

[RB97] Jack Rossignac and Paul Borrel. Multiresolution 3d approximations for rendering complex scenes. In *Geometric Modelling in Computer Graphics*, volume 1, pages 455–465. Springer, 1997.

[RC04] Francisco Ramos and Miguel Chover. Lodstrips: Level of detail strips. In *International Conference on Computational Science*, pages 107–114, 2004.

[RHG$^+$01] J. Ribelles, P. S. Heckbert, M. Garland, T. Stahovich, and V. Shivastava. Finding and removing features from polyhedra. In *Proceedings of ASME DETC'01*, 2001.

[RO08] Antonio J. Rueda and Lidia M. Ortega. Geometric algorithms on cuda. In *Proceedings of International Conference on Computer Graphics Theory and Applications '08*, pages 107–112. Insticc, 2008.

[Sch97] William J. Schroeder. A topology modifying progressive decimation algorithm. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 205–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[Sch02] G. Schrocker. Hardware accelerated per pixel shading. In *CESCG 2002*, 2002.

[SFM05] Avneesh Sud, Mark Foskey, and Dinesh Manocha. Homotopy-preserving medial axis simplification. In *Proceedings of the 2005 ACM symposium on solid and physical modeling*, 2005.

[SGR96] Marc Soucy, Guy Godin, and Marc Rioux. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, 12(10):503–514, 1996.

[Sha48] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 1948.

[SK07] Ramtin Shams and Rodney A Kennedy. Efficient histogram algorithms for nvidia cuda compatible devices. In *Proceedings International Conference on Signal Processing and Communications Systems (ICSPCS)*, pages 418–422, Gold Coast, Australia, 2007.

[SM05] Pedro V. Sander and Jason L. Mitchell. Progressive buffers: view-dependent geometry and texture lod rendering. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, pages 129–138, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[SR00] Olaf Schmidt and Mathias Rasch. Parallel mesh simplification. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, volume 3, pages 1361–1367. CSREA Press, 2000.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 2001. ACM.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM.

[Tan88] I.J. Taneja. Bivariate measures of type a and their applications. *Tamkang Journal of Mathematics*, 19(3):63–74, 1988.

[TBG09] Atul Thakur, Ashis Gopal Banerjee, and Satyandra K. Gupta. A survey of cad model simplification techniques for physics-based simulation applications. *Computer-Aided Design*, 41(2):65–80, 2009.

[TCRS00] Marco Tarini, Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Real time, accurate, multi-featured rendering of bump mapped surfaces. *Computers & Graphics*, 19(3), 2000.

[TFC08] Yan Tang, Xin Fu, and Rui Che. 3d model simplification method based on roi. In *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 495–497, Washington, DC, USA, 2008. IEEE Computer Society.

[TI07] D. Teixeira and Move Interactive. Baking normal maps on the gpu. In Addison-Wesley Professional, editor, *GPU Gems 3*, chapter 22. December 2007.

[Tsa88] C Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 1988.

[Tsa98] C. Tsallis. Generalized entropy-based criterion for consistent testing. *Physical Review E*, 58:1442–1445, 1998.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph.*, 26(2):55–64, 1992.

[Var94] Amitabh Varshney. Hierarchical geometric approximations. Technical report, Chapel Hill, NC, USA, 1994.

[VBLT05] Fabien Vivodtzev, Georges-Pierre Bonneau, and Paul Le Texier. Topology preserving simplification of 2d non-manifold meshes with embedded structures. *The Visual Computer*, 21:679–688, 2005.

[VC00] Luiz Velho and Jonas Comes. Hierarchical 4-k meshes: Concepts and applications. *Computer Graphics Forum*, 2000.

[VC07] Huy T. Vo and Steven P. Callahan. Streaming simplification of tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):145–155, 2007. Member-Lindstrom,, Peter and Member-Pascucci,, Valerio and Member-Silva,, Claudio T.

[Vel01] Luiz Velho. Mesh simplification using four-face clusters. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 200, Washington, DC, USA, 2001. IEEE Computer Society.

[VFSG06] Ivan Viola, Miquel Feixas, Mateu Sbert, and Eduard Groller. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):933–940, 2006.

[VFSW01] P. P. Vázquez, M. Feixas, M. Sbert, and Heidrich W. Viewpoint selection using viewpoint entropy. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference*, pages 273–280, 2001.

[Vio02] I. Viola. Applications of hardware accelerated filtering. In *CESCG 2002*, 2002.

[VTV$^+$04] Antônio Wilson Vieira, Lewiner Thomas, Luiz Velho, Hélio Lopes, and Geovan Tavares. Stellar mesh simplification using probabilistic optimization. *Computer Graphics Forum*, 23(4):825–838, 2004.

[WB98] D. Weinsteun and J. Brederson. Importance driven texture coordinate optimization. pages 97–104, 1998.

[WFG02] Yigang Wang, Bernd Fröhlich, and Martin Göbel. Fast normal map generation for simplified meshes. *J. Graph. Tools*, 7(4):69–82, 2002.

[WHC04] Yong Wu, Yuanjun He, and Hongming Cai. Qem-based mesh simplification with global geometry features preserved. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 50–57, New York, NY, USA, 2004. ACM.

[WHST01] Jian-Hua Wu, Shi-Min Hu, Jia-Guang Sun, and Chiew-Lan Tai. An effective feature-preserving mesh simplification scheme based on face constriction. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, page 12, Washington, DC, USA, 2001. IEEE Computer Society.

[WK03] Jian-Hua Wu and Leif Kobbelt. A stream algorithm for the decimation of massive meshes. In *Proceedings of Graphics Interface*, pages 185–192, 2003.

[WLC$^+$03] Nathaniel Williams, David Luebke, Jonathan D. Cohen, Michael Kelley, and Brenden Schubert. Perceptually guided simplification of lit, textured meshes. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 113–121, New York, NY, USA, 2003. ACM.

[WSO03] David R. White, Sunil Saigal, and Steven J. Owen. Meshing complexity of single part cad models. In *Proceedings of the 12th International Meshing Roundtable*, pages 121–134, 2003.

[XSX05] Aiguo Xu, Shouqian Sun, and Kaiqiang Xu. Texture information driven triangle mesh simplification. In *Proceedings of Computer Graphics and Imaging '05*, pages 43–48, 2005.

[ZT02] Eugene Zhang and Greg Turk. Visibility-guided simplification. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 267–274, Washington, DC, USA, 2002. IEEE Computer Society.