Adaptive tessellation of connected primitives for interactive walkthroughs in complex industrial virtual environments

M. Krus[†‡], P. Bourdot[†], A. Osorio[†], F. Guisnel[‡] and G. Thibault[‡]

[†] LIMSI/CNRS, UPR 3251
BP 133
F-91403 Orsay Cedex

[‡] EDF/DER, IMA/TIEM/CAO
1, av du Gle de Gaulle
F-92141 Clamart Cedex

Email : Mike.Krus@edfgdf.fr, pb@limsi.fr, osorio@limsi.fr, Francoise.Guisnel@edfgdf.fr, Guillaume.Thibault@der.edfgdf.fr

Abstract. Geometrical primitives used in virtual environments are converted to an important amount triangles at rendering time. The meshes of the resulting simplifications usually introduce discontinuities between neighboring object. We extend a simple adaptive tessellation method which adapts the amount of triangles to the viewing conditions with connection information to ensure that the meshes of connected primitives remain continuous. An ergonomical study has validated this approach for applications using virtual environments.
Keywords: adaptive tessellation, virtual environments, user evaluation.

1 Introduction

The scenes used for virtual simulators of complex and hostile industrial environments contain a large number of technical networks (piping, cable trays, ventilation) which are usually modeled by combinations of simple primitives such as cylinders, tori and cones. These primitives are converted to triangles at rendering time, leading to a huge triangle count. In order to maintain a sufficient frame rate, the amount of resulting triangles must be reduced.

Many methods have been proposed to simplify meshes and build several levels of details which can be used at different distances from the observer. These algorithms often remain unsatisfactory because they increase the size of the database. In addition, they do not respect the topology of the objects and the relationship between neighboring objects. Thus, discontinuities appear in rendering for objects which were initially connected.

We have chosen a method of real time adaptive tessellation of the primitives based on the distance from the viewpoint. In order to prevent the creation of visual artifacts and ensure a “continuity” in the mesh, we describe the relations between primitives by the graph of their connections. By evaluating the density of points required at each connection, it is possible to manage the fitting between the tessellations of the connected primitives, producing a mesh which contains the lowest number of triangles acceptable for a given viewpoint.

The following section presents the context of this research. Section 3 presents our adaptive tessellation algorithm. Finally, sect. 4 presents the results of an ergonomical study conducted to evaluate the quality of the tessellation and give a sample of the measured performance gains.

2 Background

2.1 Industrial Context

Industrial installations such as power plants contain many technical networks such as piping for fluids, cable layouts and ventilation. They are usually modeled for CAD or VR applications using simple primitives such as cylinders, tori, boxes or pyramids, which are converted to triangles at rendering time. This produces an enormous amount of triangles which needs to be reduced to achieve an interactive frame rate for VR applications. The traditional approach was introduced in 1976 by Clarck who suggested the use of *levels of detail* [1]. Several versions of the same object are produced with decreasing number of triangles (see fig. 1 with 1906 and 368 triangles). While reducing the number of faces, LOD algorithms introduce a number of problems (see sect.2.2). Our approach is to use specific characteristics of these primitives to render them adaptively based on the current viewpoint.

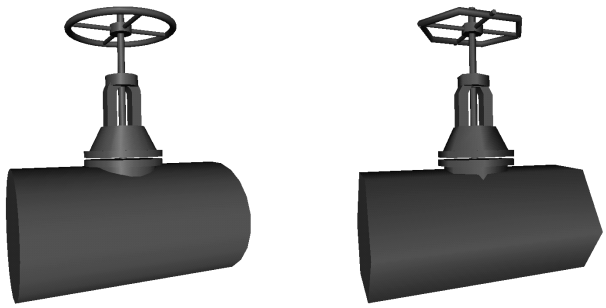


Fig. 1. Two versions of the same object

The major applications for this technology are training simulations [2], project review during the design phase, interactive walkthroughs for intranet based mission preparation or for the general public in visiting centers.

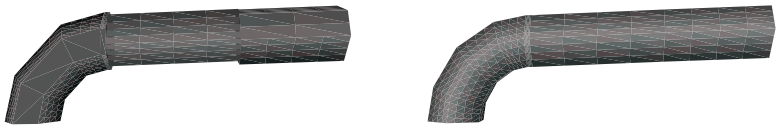
2.2 Related work

Levels of details. Simplification of arbitrary meshes is usually achieved by polygonal simplification (see [6] for an overview). Some type of heuristic is used to identify features of the geometry which should be preserved or removed. For example, polygons in planar area are merged into bigger ones. The tolerance on the deviation from the plane can be relaxed to decrease the number of resulting polygons. These heuristics are needed because topological information concerning the object is usually not available.

When using simple primitives such the ones we are concerned with, we could use an off-line tessellation algorithm which would produce several meshes for each primitive at different levels of details. However, this, as well as polygonal simplification methods, has several drawbacks:

– Since every primitive has several different representations, the size of database increases significantly. For scenes which initially contain an important number of primitives this can be a considerable handicap.

– It is not easy to know how many simplifications should be built and at what distance they should be used. It is a matter of finding a balance between the required quality of rendering and the size of the database. However, that knowledge is very application specific.
– Furthermore, a popping effect is usually noticeable when two representations of an object are swapped because the differences between them can be quite significant.
– Since topological information is not available, there is no way for the simplification algorithms to know that, for example, a given set of polygons form a cylinder. They are thus usually incapable of producing simplification which preserve that topological constraint.
– Different objects which are connected, i.e. topologically share some of their vertices and edges, often appear disconnected when simplified, leading to discontinuities and cracks in the mesh (see fig. 2(a)), particularly visible on low end image generators which do not support antialiasing. This happens because the primitives are simplified independently of each other and that the knowledge of their connection is not used by (or not available for) the simplification algorithms. Some algorithms resolve this problem by leaving border vertices unchanged. However, in the case of simple primitives such as cylinders, no simplification could then be performed since all vertices are on a border.



(a) Non connected primitives (b) Connected primitives

Fig. 2. Mesh discontinuities for adjacent primitives

Adaptive meshes. Some algorithms are capable of producing *geomorphs* which contain an entire range of simplifications [3, 4]. From the simplest form of a mesh, any level of detail can be obtained by selectively adding a number of triangles based on information compiled during the simplification process. This greatly reduces the popping effect since the difference between two successive meshes can be as small as one triangle. [5] extends this method so that parts of the object facing towards the observer will contain more triangles than other parts. These methods are well suited for complex meshes like terrains; however they introduce a significant overhead for simple primitives such as those we are interested in. Additionally, the connections between objects remain ignored.

Performer's *Active Surface Definition* (ASD, see [9]) enables several pre-computed levels of details to be combined in a single structure. At rendering time, based on distance, a mesh is constructed by interpolating between two of the simplifications, producing a morphing effect. Using the ASD, the amount of detail is not uniform over the entire mesh. This method is well suited for terrain models which span a large part of the scene so that some parts are close and others are far from the observer. However, extra care needs to be taken for parts of different objects which are connected usually by insuring that the connection points are not simplified.

It is not clear whether the use of such techniques, given their complexity, would yield a significant increase in the rendering performance of simple primitives. A way of applying these algorithms would be to merge the triangles of the primitives of a given pipe branch and build geomorphs or ASD structures from that. However, animation or manipulation within the simulator of individual primitives would then be impossible. Furthermore, using on single large object would reduce the efficiency of view frustum and occlusion culling (in our framework, as described below, only visible primitives are considered for adaptive tessellation).

Scene based simplification. Luebke and Erickson propose a way of adaptively simplifying entire scenes formed of simple triangles ("polygon soup") [7]. The scene is partitioned using a hierarchical structure. A single vertex is chosen for all vertices in a given node of the hierarchy. At rendering time, traversing of each branch of the hierarchy is halted when the screen size of the current node becomes inferior to a given area. Thus parts of the structure which are further away and which project to smaller areas on the screen are traversed less deeply and the corresponding geometry is simplified.

This method achieves adaptive tessellation of any collection of triangles. However it does not respect the topology of the objects. Furthermore, the hierarchical structure which is used to partition the scene needs to be updated whenever an object moves which makes it impractical for applications requiring animations.

3 Adaptive tessellation

Since traditional LOD algorithms perform poorly on the type of scenes we are concerned with, we decided to use adaptive tessellation of the primitive and enhance it by introducing connection information to produce continuous meshes (fig. 2(b)).

One additional constraint was that our design needed to be integrated in a modern library designed to build virtual environments which are based on a scene graph in which basic primitives (defined in their own coordinate system) are composed by translation and rotation in a hierarchical way. This is convenient because it often corresponds to hierarchical compositions of objects in the application domain (like the various part of an engine). Additionally, simple animations can be performed by changing the transformation, without needing to modify each vertex of all object individually.

3.1 Basic geometric primitives

The main type of primitives we manage adaptively in our system are those with a circular section such as a cylinder, a torus or a cone. The number of triangles needed to approximate the curvature of the objects can be very important. Adaptive tessellation is used to select a number of points on the section of those primitives.

Connected primitives have two connections which are used to build the list of vertices, normals and texture coordinates for the primitive. If two primitives of the same type are placed in the scene in such a way that they share an extremity (the end points and the radius coincide) then a shared connection is constructed. Tessellation is evaluated at each connection and a mesh is constructed between two connections to represent

a given primitive. In effect, constructing a mesh of several connected primitives can be viewed as a non uniform extrusion along a path linking the different connections, with constraints being propagated from one to the other (see sect. 3.2).

In order to determine if two primitives are connected, the ending points of each primitive need to be compared. If the primitives are expressed in different coordinate systems, the points need to be converted into a common systems. This increases the number of computations and introduces errors. In order to combine efficient connection detection with traditional primitives, we have introduced a double representation:

- *Connectable Primitives* which are defined in the current coordinate system. For example, a cylinder would be defined by the two end points and the radius. Connections between these primitives are managed by the *connection controller*.
- *Instance Primitives* which do not contain any geometry and are used in the scene graph. They each create a connectable primitive which is registered with the connection controller. When the graph is traversed, the transformations are applied to the primitives which update the parameters of the associated connectable primitive.

The torus is tessellated adaptively both on the section and on the number of segments used to rotate from one extremity to the other by a given angle around the axis of the torus (see the wheel of the tap in fig. 1). In our implementation, each step in the rotation is represented by an implicit connection to which other primitives cannot connect. This way, we have a uniform way of building vertices, normals and texture coordinates for all the primitives. The number of these implicit connections depends on the angle covered by the torus.

3.2 Building a mesh

In our current implementation, the connectable primitives are either cylinders, cones or tori. Each circular connection is defined by a radius, a center and a normal. For cylinders and cones, the normal is the axis of the primitive. For tori, it is the tangent of the arc defined by the major radius. A connection is either *bound* if it connects two different primitives, or *free* if it only belongs to one primitive (as is the case for the first or the last one in a list of connected primitives). The basic steps in the algorithm for constructing a mesh are:

– construct points, normals and texture coordinates;
  update these if one of the parameter of the primitive changes.
– for each frame and each primitive:
  – select a number of points for each connection.
  – build a mesh.

Constructing vertices. Points for each connection are constructed by using the simple parametric equation of the circle in the XY plane. The points are then translated to the origin of the connection and rotated so that the Z axis of the circle lies along the direction of the normal for the connection. However, special care needs to be taken to make sure that points of the two connections of the same primitive are aligned with each other. Otherwise some primitives would appear twisted.

This is done by defining an *up vector* in the direction where the first point of the circle will be. It is either the X axis or the Z axis depending on whether the normal of the connection lies along the Z axis or not. Given a list of connected primitives as explained in sect. 3.3, an up vector is chosen for first free connection. That up vector is then propagated from one connection to the other depending on the type of the primitive:

- if the primitive is a cylinder or a cone, the up vector is simply copied from the first connection to the second. Changing the up vector between two extremities of a cylinder or a cone would introduce a twisting effect which would be noticeable.
- if the primitive is a torus, the up vector is rotated around the axis of the torus by the same angle as the torus. This is done in steps for each of the implicit connections. Thus, the rotation of the up vector is made smoothly over all the connections as shown in fig. 3.

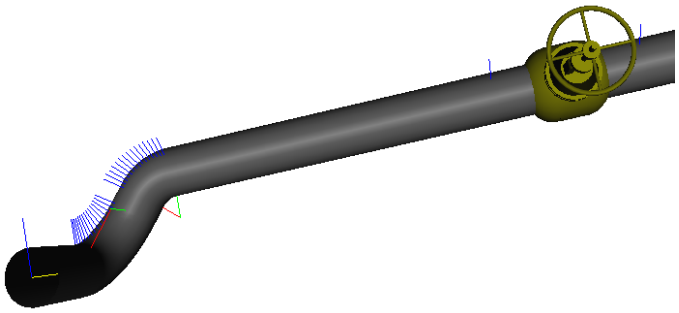


Fig. 3. Propagating the *up vector*

Selecting vertices. The adaptive tessellation algorithm selects a number of points from those stored in the connection. The selection criteria should ideally be based on the screen size: the smaller the primitives are, the lesser will the curvature be noticeable and fewer triangles will be needed to represent them. However screen size can be expensive to compute. In the case of the primitives we are using, two factors are considered to select an appropriate tessellation at each connection:

- *the distance:* more details are required for closer objects. For each object, the ergonomical study (see sect. 4) has enabled us to define a range of distances between which tessellation should vary linearly from the highest to the lowest possible value. Distance is measured at the center of each connection. This means that connected primitives can have a different number of vertices at each connection and that the resulting mesh would not be uniform.
- *the radius of the primitives:* the smaller the radius, the fewer vertices are used. This factor is in fact used to modify the value of the distance range discussed above. If a cylinder of one meter diameter reaches it's lower tessellation level at a distance of one hundred meters, then a cylinder of fifty centimeters diameter will reach it at only fifty meters.

We do not use the length of the cylinder as a factor for computing tessellation because the primary feature which makes a cylinder identifiable as one is the curvature which is controlled by the radius. Length has very little impact on the perception of the curvature.

Initially we tried selecting vertices continuously, adding points one by one as the primitive moves closer to the observer. However this produces a rotation effect since points are packed gradually closer to each other and additional points are added to close the circle. We have thus chosen to constrain the number of vertices so that it remains a multiple of three. This maintains three "stable" directions as the tessellation evolves suppressing the rotation effect.

Building triangles. Given a number of points for each connection, a triangle strip is build connecting the vertices. If the number of vertices is not the same for both connections, the strip will contain degenerate triangles (reduce to a segment). While this is generally not desirable, it has no effect on the rendering (since points and normals coincide) and enables to retain the use of strips which are quicker to render.

3.3 Managing connections

As explained in sect. 3.2, constraints need to be propagated from one connection to the other so that points remain aligned. For animation purposes or because of user interactions within the simulator, the position and orientation of a primitive might change. Each time this happens, the connection information needs to be updated. This is done in three steps:

1. The validity of all existing connections is checked. The non-valid connections are deleted and replaced by free connections for the each disconnected primitive.
2. Primitives with free connections are compared to each other to determine if new connections are established. If no connection can be found, the free primitives are placed in a temporary list.
3. Finally, constraints are propagated from one primitive to the other starting with those in the list constructed in the previous step. Given a primitive of that list, an up vector is chosen for its free connection and is recursively propagated from connection to connection (until a free connection is reached) using the method described in sect. 3.2.

This ensures that constraints can be propagated without having to explicitly build lists of primitives: the connections are used to get the chaining information.

4 Evaluation and Performance

Experiments. The aim of these experiments was to evaluate the loss of information when simplifying virtual 3D objects. Reddy showed that the caracteristics of the human visual system can be used to control the simplification objects [8]. We extend these results by showing that simplifications can be used at levels which are perceivable because cognitive processes facilitate the recognition of objects. We tested the perceptual

behavior of various types of users to determine the effect of field knowledge: workers from the piping and plumbing service of the CNPE (Energy Production Unit) of Tricastin; computer scientists and engineers, expert in VR applications; people non experts in either production installations or VR applications. The experiments were designed and conducted by A. Drouin, S. Tonnoir and M.-A. Amorim, of EDF.

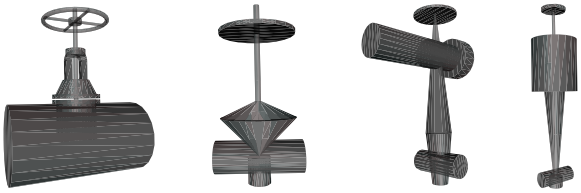


Fig. 4. Four taps for the experimentation

- Different versions of standard taps were displayed with various levels of tessellation and view angles (see fig. 4. The subjects were ask to adjust the distance of the objects to the distance they thought was most appropriate. We could thus reconstruct the curve of perception levels for 3D objects based on the distance from the observer.
- The same objects were presented close up at various angles and the tessellation was gradually decreased until the user pressed a key to indicate that he had sensed a change in the representation of the object. This enabled us to optimize the maximum number of triangles needed when the objects are presented close up.
- Having tested the perception of the tessellation, we still needed to test if participants were capable of identifying randomly presented taps based on the mental representation they had acquired. A close up view of a highly detailed version of each of the objects was displayed side by side by a random view of any object at any tessellation level, any orientation, and any distance. The user was asked to decide whether both displayed objects were identical.

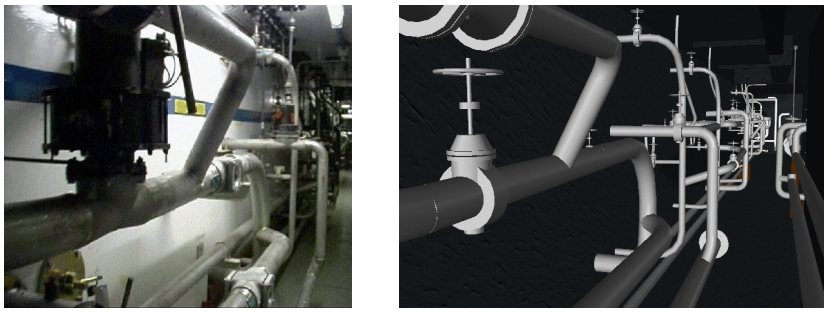


Fig. 5. Comparing real and virtual scenes

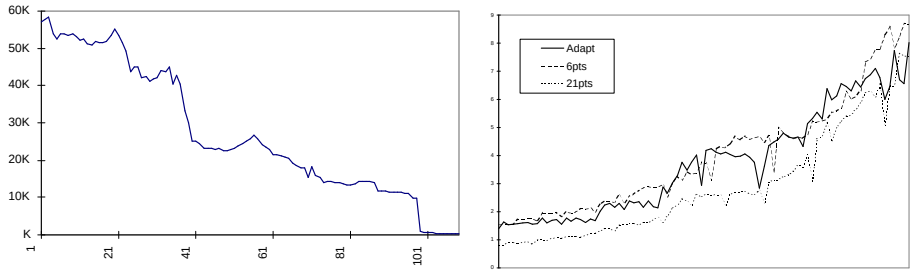
- The last experiment was intended to study the behavior of participants when they are required to process a complex environment of geometric and architectural configurations. We wanted to find out if an excessive degradation would impact on the efficiency of the simple viewpoint finding task. A picture of the real scene was displayed above a 3D model of the scene in which the user was ask to navigate until

he would find the point of view corresponding to where the picture had been taken from (see fig.5). The subjects were divided in four groups. Adaptive tessellation was activated for the first group but the three others where shown three different fixed levels of tessellation.

Results. The first three experiments proved that the appropriate tessellation for the circular section of the primitives should range from 6 vertices to 18. The minimum tessellation for a cylinder of one meter diameter should be reached at 30 meters.

The last experiment showed that adaptive tessellation improved the performance of subjects within the environment. High levels of fixed tessellation lower the frame rate and hinder the navigation. The lowest level of tessellation distorted the objects too much and made identification more difficult. The frame rate and triangle count for the adaptively tessellated environment were comparable to those obtained for a fixed tessellation of 12 vertices. However, triangles were better distributed since closer objects had more details than further ones. This lead to fewer identification errors and to a better recognition of objects in the environment.

Results also showed that dynamic visual perception is influenced by the operative knowledge as well as by the task that is assigned to the subject. The perception and recognition strategy of industry workers was influenced by their prior knowledge of the types of objects and installations used in the experimentation. It usually help them recognize simplified objects by accepting greater degradations and filtering out some parts of the object which were not important for the recognition (as the supporting pipes). However, it also sometimes lead them to consider only foreground objects and ignore pertinent information in the background, which introduced a few more errors in the navigation.



(a) Visible triangle count (b) Frame rates

Fig. 6. Performance measures

4.1 Performance

We have implemented this algorithm using the OpenGL Optimizer library. It is coupled to a view frustum and an occlusion culling algorithm so that only visible primitives are evaluated for adaptive tessellation. The performance measures where made on an environment representing a forty meters long corridor with one side covered with various pipes (see fig.5). It contains about 2300 primitives leading to maximum of 50K

triangles. The observer simply walked down that corridor while the frame rate and the number of displayed triangles were measured.

As shown in fig. 6(a), the number of triangles rises slowly as the users moves closer the objects and drops sharply when close objects move out of the view frustum. Figure 6(b) shows the frame rate for adaptive tessellation, and constant tessellation levels such that each cylinder is represented by six and twenty one vertices (which are the minimum and maximum values determined by the ergonomical study). The frame rate increases gradually as fewer objects are visible. The figure shows adaptive tessellation performs always better than constant high levels of tessellation which are required to maintain object comprehension.

5 Conclusion

In order to reduce the amount of triangles used to represent simple primitives, we have presented an extension of a tessellation algorithm which already which adapts the number of triangles to the observation distance. In this context, we showed that it is useful to manage connection information so that the meshes of adjacent primitives remains continuous. Using the connections, the alignment constraints can be propagated from one primitive to the next. The connections are updated each time an object moves within the scene.

An ergomical study was conducted to determine the appropriate levels of tessellation for a given distance. It has also shown that adaptive tessellation improves the performance of the user within the virtual environment by raising the frame rate while preserving the appearance of visible objects.

We plan to extend our design by including other factors to modulate the tessellation such as the incidence of the viewing direction, the speed of the object with respect to the observer or the peripheral location. Additionnaly, we intend to evaluate the inclusion of other types of primitives within our framework.

References

1. J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19:547–554, 1976.
2. A. Drouin, A. Schmid and G. Thibault. Training of Nuclear Workers using Virtual Environments. *IAEA Specialist's Meeting on Maintenance Training Centers for Nuclear Power Plants*, Charlotte, NC, October 1998.
3. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *SIGGRAPH '95*, August 1995.
4. H. Hoppe. Progressive meshes. In *SIGGRAPH'96*, 1996.
5. H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH'97*, 1997.
6. M. Krus, P. Bourdot, F. Guisnel, and G. Thibault. Levels of detail and polygonal simplification. *ACM's Crossroads*, 3.4, 1997.
7. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH'97*, 1997.
8. M. Reddy. Perceptually Modulated Level of Detail for Virtual Environments. PhD thesis, University of Edinburgh, 1997.
9. Silicon Graphics, Inc. *IRIS Performer Programmer's Guide*, 1998.