LEVEL-OF-DETAIL VISUALIZATION OF ARCHITECTURAL MODELS

S. BELBLIDIA, J.P. PERRIN
CRAI (Research Center in Architecture and Engineering),
School of Architecture, Nancy, France

Abstract

The work presented in this paper aims to use level-of-detail representation in realizing interactive walkthroughs or ignoring useless details in large architectural models. In order to choose the right representation of a model, we have to evaluate the error comitted when using a simplified version instead of the full description of an object. This error depends on the object deformation during the simplification process but also on the importance of this object in the current viewing conditions. This "visible" error is used with different visualization strategies to find the model representation which satisfies either a quality criterion or a cost condition.

1. Introduction

When visualizing large architectural databases, even powerful graphic workstations can not ensure interactive walkthroughs. Furthermore, the great accuracy of some models are useless in many applications or viewing contexts. The level-of-detail representation allows to render a model with different accuracies which depend on the application requirements.

Before the visualization step, a level-of-detail model must be generated following these two steps:

- organizing the database in a hierarchical structure which contains simple and composite objects.
- creating several versions of these objects using an automatic simplification algorithm.

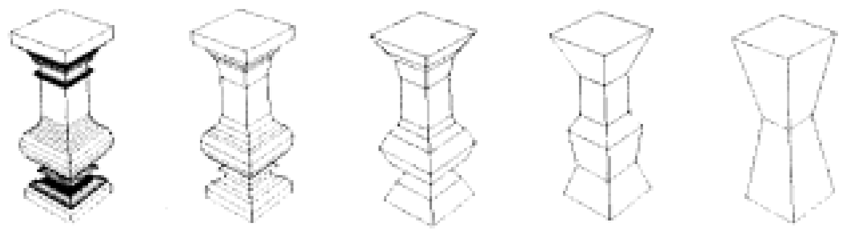
In a previous paper, we presented the improvements we introduced [11 to an existing simplification algorithm [2] . We focus here on the visualization stage where the main problem is to choose the adequate model representation. We present two rendering algorithms which allow to find the representation that matches either a quality criterion (section 4.1) or a rendering cost (section 4.2). To make it possible, we need to evaluate the approximation error when using a simplified version (section 2) and the rendering cost of this version in time units(section 3). The main apport of these algorithms in regard to other published works [3][4] is the use of an approximation error deducted from the simplification stage.

We are now experimenting the software on a complex architectural model composed of thousands of polygons (section 5).

2. Error evaluation

2.1 GEOMETRIC ERROR

When we simplify an object using the automatic simplification algorithm, we evaluate the difference between the original shape and the simplified version by storing the maximum vertex displacement [1]. This purely geometric factor is independent from the viewing context and the object importance.



original version	version 1	version 2	version 3	version 4
206 polygons	**86 polygons**	**46 polygons**	**22 polygons**	**10 polygons**

Figure 1: Simplified versions automatically generated.

2.2 IMPORTANCE CRITERIA

The geometric factor computed above does not include any information about the object position or importance in regard to other objects of the model. What we need to evaluate is the "visible" error on the screen which is computed by weighting the geometric error with several factors depending on the following criteria.

2.2.1 *Focal distance*

The visible error is direcly proportional to the focal distance. The approximation error is more visible with a 200 mm camera than a 35 mm camera.

2.2.2 *Object-camera Distance*

The visible error is in inverse proportion to the object-camera distance. The error is more perceptible with close views.

2.2.3 *Object Position on the Screen*

fn some visualization contexts, objects which are close to the center of the screen are more important. The relationship between this importance and the distance to the center of the screen is a gaussian function which aspect is user-controlled.

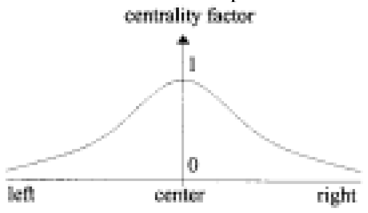


Figure 2: Variation of the centrality factor

2.2.4 *Object semantics*

Some objects could be more important in a complex architectural model. For these objects, a semantical importance factor is fixed by the user in order to major the visible error.

2.3 VISIBLE ERROR

The visible error for a given triplet (object, version, camera) is computed by weighting the geometric error with the importance factors.

$$\mathrm{VisibleError}(o,v,c) = \underbrace{\mathrm{GeometricError}(o,v)\cdot\frac{\mathrm{FocalDistance}(c)}{\mathrm{Distance}(o,c)}}_{projection}\cdot\mathrm{Centrality}(o)\cdot\mathrm{Semantics}(o)$$

The two first terms of this expression represent the projection of the geometric error on the screen and are always used. The two last ones are optional factors.

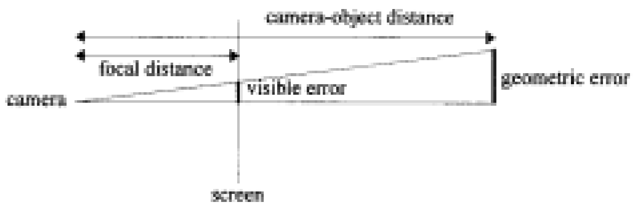


Figure 3: Visible error before weighting with centrality and semantics factors

3. Cost evaluation

The rendering cost of an object version depends on two values : the number of polygons it contains and the number of pixels it covers on the screen. The cost of a given triplet (object, version, camera) is a linear combination of these values:

$$\mathrm{Cost}(o, v, c) = C_1 \cdot \mathrm{Polygons}(o, v) + C_2 \cdot \mathrm{Pixels}(o, v, c)$$

where C1 and C1 are constants which depend on the rendering algorithm and the accuracy options used.

4. Rendering strategies

When a complex architectural model is composed of hundreds of simple objects, each one with several versions, the number of possible representations is very large. The more accurate and the less detailed are two particular ones.

The visualization algorithm must be able to find the adequate representation according to user-specified criteria. We have implemented two strategies which handle either a quality criterion or a cost condition.

The algorithms presented in the sections 4.1 and 4.2 operate only on simple objects. They also accept composite objects in extended versions which use recursive calls.

For both strategies, the different versions of an object 0 are sorted in a quality decreasing order. The full version is indexed 0 and the less detailed is indexed n-I.

4.1 THE QUALITY STRATEGY

4.1.1 Principle

The user specifies a quality criterion as a number of pixels. A given version of an object has the required quality if the visible error of the triplet (object,version,camera) - converted in pixels, for the current window size - is lower than the user-specified threshold. If this condition is not satisfied, the visible error is evaluated with a better version.

With the quality strategy, the algorithm determines independently for each object the version to use for given viewing conditions.

4.1.2 Algorithm

objectVersion (obj : object, cam : camera, qualityThreshold : real) : integer boolean stop; integer ver;

```
stop ← FALSE ;
ver ← numVersions - 1 ;
while (stop = FALSE and ver > 0) do
  if (visibleError (obj, ver, cam) < qualityThreshold) then
    stop ← TRUE ;
  else
    ver ← ver-1 ;
  endif
endwhile
return version
```

4.2 THE COST STRATEGY

4.2.1 Principle

The user specifies a target rendering time as a number of seconds. The rendering cost for the whole model must be lower than the user-specified threshold.

4.2.2 Algorithm

modelRepresentation(initList: quadrupletList, c: camera, costThreshold: real): quadrupletList
quadrupletList list;
quadruplet quad;
real modelCost;
object obj;
integer ver;

```
list ← initList ;
modelCost ← cost (list) ;
quad ← head (list) ;
while (modelCost < costThreshold and not EndOf (list)) do
  obj ← object (quad) ;
  ver ← version (quad) ;
  if (cost (obj, ver+1) - cost (obj,ver) <= costThreshold - modelCost) then
    removeQuadruplet (list, quad) ;
    insertQuadruplet (list, obj, ver+1, visibleError (obj, ver+1, cam), cost (obj, ver+1, cam)) ;
    modelCost ←modelCost + cost (obj, ver+1, cam) - cost (obj, ver, cam) ;
    quad ← head (list) ;
  else
    quad ← next (quad) ;
  endif
endwhile
return list
```

The data structure we use is a list of quadruplets (object, version, visibleError, cost). In its initial state, this list contains all the objects in their less detailed version. If the initial cost of the model is greater than the user-specified threshold, some objects can not be displayed. For each camera position, this initial list is sorted with a visible error decreasing order. The refinement procedure begin with the first objects of the list since they are those with a greater visible error. This algorithm ensures

- the rendering time is optimally used. If the head of list can not be refined, the algorithm attempts on the next object in the list.
- the rendering time is equally dispatched on the objects. The algorithm avoids to render some objects with a great accuracy when others are not refined at all.

5. Experimentation

All these features have been developed on Silicon Graphics workstations using Open Inventor C++ graphics library which provides interface and rendering facilities. We are experimenting the software on an urban area : Place Stanislas in Nancy (France). The place is surrounded of seven classical buildings (City Hall, Opera, Grand Hotel, Museum, ...) with ornemental details. The level-of-detail approach matches very well this kind of built environments.

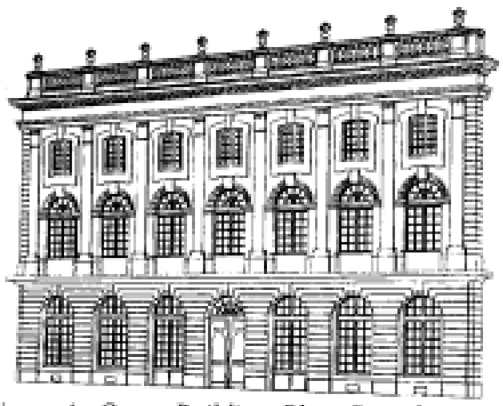


Figure 4 : Opera Building, Place Stanislas, Nancy

6. References

1. Belblidia, S., Perrin, J.P., Paul, J.C. (1995) Multi-Resolution Rendering of Architectural Models, CAAD Futures '95 Conference.
2. Rossignac, J., Borrel, P., (1993) Multi-Resolution 3D Approximations for Rendering Complex Scenes, Conference on geometric Modeling in Computer Graphics, 453-465.
3. Funkhouser, T.A., Sequin, C.H (1993) Adaptative Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments, Siggraph '93 Conference, 247-254.
4. Maciel, P.W.C., Shirley, P. (1995) Visual Navigation of Large Environments using Textured Clusters, Symposium '95 on Interactive 3D Graphics, 95-102.