In the Proceedings of the 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'2001) (V. Skala, Ed.; ISBN 80-7082-713-0) vol. 3, pages 90-98, Plzen, Czech Republic, February 2001..

VISUALIZATION ISSUES IN VIRTUAL ENVIRONMENTS: FROM COMPUTER GRAPHICS TECHNIQUES TO INTENTIONAL VISUALIZATION

A. Angelidis

G. Fouquier

EPITA Research and Development Laboratory
14-16, rue Voltaire
F-94270 Le Kremlin-Bicêtre
France
{Alexis.Angelidis,Geoffroy.Fouquier}@lrde.epita.fr

ABSTRACT

Rendering efficiently large virtual environment scenes composed of many elements, dynamic objects, and a highly moving viewpoint is a major issue. This paper focuses on the first of the two viewing stage operations: required elements determination, the second being shading/filtering. We propose a classification, extending the existing computer graphic techniques toward display scalability requirements, that distinguishes two key points: keeping only required elements (culling), and keeping only required details (which includes traditional LODs). The mechanisms needed for display scalability are presented.

Keywords: virtual environment, LOD, intentional display, semantics.

1 INTRODUCTION

Sending all the elements of a virtual environment (VE for short) scene to a rendering engine is, considering the element density of wanted scenes and nowadays platforms performances, completely out of question. The alternative is to perform some pre-processing or real-time processing to extract from the scene only the minimum pieces of data needed. The selected data, that is supposed coherent, is a local fragment, that, however, does not lead to any restriction on the shading operation, as VE shading techniques work locally.

The computing time taken by data removal should be significantly inferior to that taken if all elements were sent to the renderer, and memory requirements be reasonable. There exist many techniques for this, and all involve at least one of two methods classes: unnecessary element removal and unnecessary detail removal. The choice of the methods leads to a performance/quality compromise, as referred by Green in [Green95].

In this paper, we focus on visualization issues in virtual environments. We do not discuss the problems due to network communication, which is a concurrent task *vis-a-vis* the visualization one [Fabre00a]. Section 2 establishes an extended classication of the techniques that may be used to solve visualization issues. Section 3 describes what mechanism should be established to take full advantage of LOD techniques. Lastly, we conclude in section 4.

We stress virtual environment specific issues using the symbol: $\hookrightarrow$.

2 SCENE SIMPLIFICATION

2.1 Culling

2.1.1 Culling Definition

Culling is an operation that consists in removing elements that do not contribute to the perceptible scene. Culling can be performed at various hierarchical levels. The lowest level , which is the most atomical, is polygon culling. The second level is geometry culling, a geometry being a structure of polygons linked to the same coordinates set. The third level is sub-space culling, a sub-space being a portion of space that can hold one or more polygons, geometries, objects, or sub-spaces. The last kind of culling is performed at object level, an object being a semantic item to which is assigned a geometry or sub-spaces attribute. We refer to polygons, geometries, objects and sub-spaces as *elements*. Those culled elements are of three types: outside viewing frustrum, occluded, and semantically ignored. Outside viewing frustrum and occluded elements are related to realism issues.

$\hookrightarrow$ In VEs, considering single polygons is absurd, higher level consideration are needed.

Outside Frustrum Culling The viewing frustrum delineates the user's view-space on the world. Outside elements should not be send to the renderer. It is a local portion of the scene.

$\hookrightarrow$ A VE's problematic element is landscape. It is not evident whether it should be considered as a single geometry, thus performing polygon culling, or considered as a set of objects, allowing other levels of culling . In the second case, it might be necessary to manage cracking problems between adjacent objects.

Occlusion Culling Elements that stand inside the viewing frustrum do not need to be drawn when occluded by other elements. It is interesting to notice that at sub-space level, an occluding sub-space is an occlusion by all its internal polygons.

Backface culling is a special case of occlusion culling that assumes all surfaces are closed, so backfaced polygons are occluded by the opposite side of their belonging geometry.

Semantic Culling Semantic culling is a subversive way of considering rendering: it aims only at the viewers' comprehension of the scene, and absolutely not at scenes' realism. This actually means that elements that would have been viewable with classic visibility calculation may be culled when enabling semantic culling. This culling has to be performed at semantic level for determining if an object is required for the scene comprehension. Classic algorithms assume that the scene is understood if every detail is rendered, which, contrary to a semantical scene graphs, work on geometric scene graphs.

A geometric graph can easily be incorporated in a semantic scene graph by replacing the geometric relations with basic edge relations such as "inside", "bounding", "next to"... However, culling on a minimum semantical graph that only incorporates a geometric scene graph reduces in performing frustrum culling (thus keeping rendering realism). But when objects are linked to the scene with edges such as "decorates" and when they are of an uninteresting node type for the viewer, then they can be thrown away knowing that the scene's meaning will be preserved.

Therefore objects may be culled semantically for two reasons: for their node type or according to relations between nodes in the graph. The second reason implies that object groups may be efficiently semantically culled. The power of such a culling resides in a graph's semantic, which is not a trivial issue. There are very few VEs that have emphasized this question [Fabre00b].

2.1.2 Culling Strategies

Exploiting coherence is the key to design an efficient culling algorithm. We will distinguish five categories of coherence, extending Green's classification [Green95]: element-space coherence, image-space coherence, element-space temporal coherence, image-space temporal coherence and semantic-space coherence.

Element-Space Coherence An algorithm that uses element-space coherence can determine effectively which elements stand inside some given sub-space. Therefore at rendering, knowing that some sub-space does not intersects the frustrum, all the objects inside this sub-space can be ignored. This sub-space culling technique should involve an element-space hierarchical structure, thus finding every geometry in $O(log(n))$ complexity in a scene composed of n elements. Kay and Kajiya have enumerated the desirable properties for any hierarchical scheme [Kay86]. This category includes bounding volumes [Groel95], octree, KD-tree [Ooi87], BSP tree [Naylo90, Torre90] and PVS [Airey90] algorithms.

↪ Element-space coherence is an efficient way to perform object level culling.
↪ Hierarchies can be inserted inside an object's structure in order to perform effective intra-polygon level culling.
↪ In VEs, the choice of an object-space coherent algorithm for the landscape can introduce restrictions on the world's dynamism.

Image-space coherence Knowing that an element is visible at a pixel, there are great chances for this element to be visible at surrounding pixels. Moreover, knowing that the closest elements at an area has been drawn, farther elements can be culled. This coherence has many applications in polygon and sub-space level culling. Early algorithms [Watt92] such as the Warnock algorithm or the Weiler-Atherton algorithm belong to that category. Image-space coherence is tightly linked with object-space coherence and many algorithms fall in those two classes. Naylor has proposed a BSP screen projection [Naylo92] and Green has developed a pyramidal Z-buffer working on top of an octree [Green93, Green95].

↪ There is an image-space coherent algorithm that is widespread, as it is a common feature of 3d acceleration cards: Z-buffer scan conversion, which is performed at polygon level. However, it does not cull polygons efficiently enough, and it is often coupled to a software element-space coherent algorithm.

Element-space-temporal Coherence A moving element's position is expected to be close to its previous position, hence allowing to manage every elements locally.

Element-space-temporal coherence has rarely been exploited to perform culling. Sudarsky developed a method using TBV (temporal bounding volume) [Sudar96] that culls an object from the scene during some period of time. Moreover, it can give efficient solutions to update structure, as illustrates the LCA (least common ancestor) algorithm developed by Sudarsky.

↪ A common VE algorithm based on element-space-temporal coherence is dead-reckoning.
↪ this coherence is useful when updating hierarchical structures.

Image-space-temporal coherence In an animation, knowing the visibile collection of elements at a frame can accelerate visibility computation for the elements in the next frame, as hidden objects are likely to remain hidden. These elements are likely to be objects and subspaces.

↪ This coherence permits to manage efficiently VE's many movements.

Semantic-space Coherence At a moment, a semantically culled object is likely to be culled for the same reasons at next moment. This is the same for visible objects.

2.2 LOD (Level of Detail)

2.2.1 LOD Definition

The other kind of data that can be removed is detail. Once a bounding volume's set of elements is determined to be in view, it does not always need to be drawn in full details, as should the elements be far away from the viewer, they would only contribute to a few pixels of the rendered image. Therefore, an element's representation should propose a LOD function that would give its required geometry. Most early LOD systems did not involve processing, and the sets of geometries had to be hand-drawn. Since then, methods have been developed to approximate geometries within a certain tolerance [Schma97].

These algorithms can be described by some caracteristics, often used as technical classifying criterias:

Specifying simplification level. There are two possible ways for the user to specify the amount of simplification: by the number of polygons or the simplification error.

Topology preservation. Some algorithms may fill holes and disconnect or connect object pieces. Algorithms presented in [Hoppe96] and [Ronfa96] discourage such topology simplifications.

Smooth transitions. When changing from one level to another, the introduction or suppression of details may create a visual popping effect. To prevent this, a transition primitive called geomorph is needed. Hoppe's progressive mesh method uses geomorphs on vertex split and edge collapse operations ([Hoppe96]).

Metric. The simplification process must be conduced in such way that the shape's characteristics are preserved, such as planar areas, sharp edges and pointed edges. The distortion can be measured using a metric. This metric can be either global (distance between the original mesh and the simplified mesh) [Eck95, Guzi95] or local (evaluates the distance between steps in the simplification process) [Hoppe96, Ronfa96]. Moreover, some LOD techniques use instead of a metric a geometric construction, ensuring that simplification does not exceed a certain limit [Cohen96, Algor95].

View dependence/independence. In a serie of LOD transitions parametrized by the distance to the viewer, the object is detailed an area at a time, chosen according to some heuristic. There are recent LOD algorithms parametrized by the view, that can continue to work locally on areas thus providing a non-uniformly detailed model [Eck95, Hoppe96, Cohen96].

The elements should fulfill both display quality and rendering speed, which are completely opposed, as on one side, the element quality requires an optimal representation, and on the other side, the speed restriction tolerates a perceptible element degradation.

Thus, we introduce three classes of LOD based on: viewer distance, frame per second, and viewer interest.

2.2.2 LOD Categories

Optimal Visual LOD This is the common class of LOD. Parameterized by the distance of the element to the observer, the OV LOD function provides the optimal representation for the element, without deterioration. It aims to have a minimal number of polygons and a maximum of details, as a consequence that adding details to the element would not change the scene quality. It could even deteriorate it, as simplification performs geometrical anti-aliasing.

Optimal Display Rate LOD Maintaining a moreless constant frame rate is a major requirement of VEs. That is what ODR LOD aims at while throwing away details. There is a polygon budget for rendering, implying that the elements' polygon richness has to be constrained, and this budget is maintained by the element's ODR LOD. This results in a perceptible but tolerated homogenous degradation of the scene's elements.

$\hookrightarrow$ In VEs, the number of elements is not determined. The ODR LOD has its place in VEs that aim at element number scalability, in contrast with basic VEs where the number of elements in the scene cannot increase above a pre-determined maximum.

Optimal Intentional LOD The viewer' s intention in the scene is always contained by the frustrum, but is never equal to it, so even if there are many elements in it, all do not need to be rendered at the same ODR LOD, thus implying a controlled heterogeneity of elements' detail levels. Hence, elements are weighted by the intention that the user grants to them, thus making possible to determine for each an optimal intentional LOD (OI LOD). OI LOD will be detailed in section 3. To our knowledge, this class is empty.

3 DETAILING OI LOD

3.1 Situating the Three LODs

OI LOD stands between total degradation and OV LOD.

If the polygon budget is large enough to draw every element at OV LOD, then OI LOD is the same as OV LOD.

In other cases, an appropriate polygon budget has to be assigned to the elements that is proportional to the intention the viewer grants them.

At best, the budget is sufficient to render the elements at OV LOD. At worst, a decision has to be made:

- the scene does not worth rendering if it does not have the required details, so rendering will be done at OV LOD, thus breaking frame rate.
- for each element, a polygon budget smaller than the OV LOD number of polygons is assigned per element, proportional to the intention that the viewer grants them.

ODR LOD is a special case of OI LOD, where the viewer's intention in every element is the same, and for which the second decision is always taken.

3.2 Interruptability

By interruptability, we mean that a rendered scene may be asked to the rendering engine without considering whether it had the time to complete it or not.

This is useful in a context where the number of frames per second is variable, controlled by the dynamism of the frustrum. Unfortunately, this produces incomplete and incoherent scenes. A way to anticipate the interruption is to control the number of polygons sent to the renderer, such as when the interruption occurs:

- if the scene has been completely rendered, the polygon budget per frame can be increased.
- if the scene has been partially rendered, the polygon budget per frame can be decreased.

3.3 OI LOD Mechanism

This mechanism aims at determining how to establish the elements' OI LOD.

Because of his motivations in the world, a viewer may be more interested in some objects than others, such as the objects he is looking for, potentially interacting objects or other avatars, the viewer having a different intention with every object.

The determination of the intention degree depends of the VE graph's structure, or some interface. In a VE based on a geometric scene graph, this degree could be determined by the angle between the viewing direction and the object's position.

Having each object's intention, a OILOD has to be determined, that respects the following constraints:

- considering each element's with a percentage of detail of the original element for optimal visual representation: $OVLOD$.
- considering each object's intention degree with a percentage for intention degree: $iDegree$.
- the polygon budget has to be shared among all the viewed scene objects, therefore we have a coefficient that weights every object degradation: $ODRLOD$. It is updated while the viewer moves.

The percentage of degradation for each object's optimal intention is thus given by $OILOD = OVLOD * iDegree * ODRLOD$.

We give a pseudo-code for the OI LOD mechanism for a scene contained in a tree structure:

```
r = 1000.0 / fps
ODRLOD = 0.0
while TRUE do
  t1 = mstime()
  scene.draw(ODRLOD)
  t2 = mstime()
  dt = t2 - t1;
  ddt = dt - r
  ODRLOD += f(dt, ddt)
endWhile
```

Where f is a piecewise continuous function that provides an increment for $ODRLOD$ according to the rendering variation *ddt* and rendering time *dt* (Fig. 1).

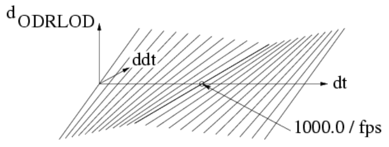

Figure 1: Function form

4 CONCLUSION

Classic culling and LOD algorithms are not designed to support effectively real-time VE rendering, as they omit to consider that the potential number of objects and their potential details is not restricted.

We have illustrated that the only way to support visible objects scalability is to perform geometry degradation, and that when the number of frames per second does not depend of the renderer, degradation has to be managed dynamically.

Moreover, widespread VEs do not consider the semantic meaning that can supply an object and the relation that can link such meanings, as presented in this paper.

REFERENCES

[Airey90] J.M. Airey, J.H. Rohlf, and F.P. Brooks Jr. Towards image realism with interactive updates rates in complex virtual building environments. volume 24, pages 41–50, March 1990.

[Algor95] M.-E. Algorri and F. Schmitt. Mesh simplification. In J. Rossignac and F. Sillion, editors, *Eurographics '96*, number 3, pages C77–C86, August 1995.

[Cohen96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Proceedings of SIGGRAPH*, pages 119–128, 1996.

[Eck95] M. Eck, T. DeRose, T. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH*, pages 173–182, Los Angeles, California, August 1995. Addison Wesley.

[Fabre00a] Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Géraud, and A. Demaille. An asynchronous architecture to manage communication, display, and user interaction in distributed virtual environments. In J.D. Mulder and R. van Liere, editors, *"Virtual Environments 2000", Proceedings of the 6th Eurographics Workshop on Virtual Environments (EGVE'2000)*, Computer Science / Eurographics Series, pages 105–113, Amsterdam, The Netherlands, June 2000. Springer-Verlag WienNewYork.

[Fabre00b] Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Géraud, and A. Demaille. A framework to dynamically manage distributed virtual environments. In J.-C. Heudin, editor, *Proceedings of the 2nd International Conference on Virtual Worlds (VW'2000)*, volume LNAI 1834 of *Lecture Notes in Computer Science Series*, pages 54–64, Paris, France, July 2000. Springer Verlag.

[Green93] N. Greene, M. Kass, and G. Miller. Hierarchical zbuffer visibility. In *Computer Graphics*, volume 27, pages 231–238, 1993.

[Green95] N. Greene. *Hierarchical rendering of complex environments.* PhD thesis, University of California, Santa Cruz, CA, USA, 1995. see chapter 2.

[Groel95] Eduard Groeller and Werner Purgathofer. Coherence in computer graphics. Technical Report TR-186-2-95-04, Institute of Computer Graphics 186-2, Technical University of Vienna, Austria, March 1995. human contact: technical-report@cg.tuwien.ac.at.

[Guzi95] A. Guziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, november 1995.

[Hoppe96] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH*, pages 99–108, 1996.

[Kay86] T. Kay and J. Kajiya. Ray tracing complex scenes. In *Proceeding of SIGGRAPH'86*, pages 268–278, August 1986.

[Naylo90] B.F. Naylor, J. Amantides, and W.C. Thibault. Merging bsp trees yields polyhedral set operations. In *Proceedings of SIGGRAPH*, volume 11, pages 115–124, Dallas, TX, USA, August 1990.

[Naylo92] B.F. Naylor. Partitioning tree image representation and generation from 3d geometric models. In *Proceedings of Graphics Interface*, pages 201–212, Vancouver, Canada, May 1992.

[Ooi87] K. Ooi, B. McDonell, and R. Sacks-Davis. Spatial kd-tree: A data structure for geographic database. In *BTW*, pages 247–258, 1987.

[Ronfa96] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. In *Computer Graphics Forum*, volume 15, August 1996.

[Schma97] Dieter Schmalstieg. Lodestar: An octree-based level of detail generator for VRML. In Rikk Carey and Paul Strauss, editors, *VRML 97: Second Symposium on the Virtual Reality Modeling Language*, New York City, NY, February 1997. ACM SIGGRAPH / ACM SIGCOMM, ACM Press. ISBN 0-89791-886-x.

[Sudar96] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, August 1996. Proceedings of Eurographics.

[Torre90] E. Torres. Optimization of the binary space partitioning algorithm (bsp) for the visualization of dynamic scenes. In *Proceedings of Eurographics*, pages 507–518, Montreux, Switzerland, September 1990. Elsevier Science Publishers B.V. (North-Holland).

[Watt92] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques – Theory and Practice.* ACM Press, 1992.