• 14000M057
Copic 1

**Multilevel Use of Coherence
for Complex Radiosity Environments**

Josep Vilaplana
Xavier Pueyo

Report LSI-93-6-R

UPC
Facultat d'Informàtica
de Barcelona - Biblioteca
- 1 MARÇ 1993

Multilevel Use of Coherence for Complex Radiosity Environments

Josep Vilaplana

Xavier Pueyo

February 16, 1993

Abstract

The performance of radiosity algorithm has been improved in the last years by means of using coherence when evaluating the visibility function for form factor computation. We present here a characterization of the uses of coherence for this goal. The characterization includes both, previously proposed techniques and unexplored ones. Afterwards we propose new uses of coherence for visibility computation in form factor determination and we discuss the results of its implementation.

1 Introduction

Most of the computational resources needed in radiosity algorithms are expended to perform the h (also called *hid*) function; i.e. to compute visibility from each sample point in the environment. In the worst case, this corresponds to an $O(n^2)$ problem.

In order to improve the efficiency of these algorithms, several papers have proposed specific uses of coherence that speed up the intervisibility computation (i.e. the h function) in radiosity environments. We study in the present paper the use of these and other types of coherence suitable for static environments. Our target is trying to give a framework for a useful exploitation of them, as well as proposing new uses of visibility coherence. We propose, in section 2, a characterization of the different kinds of coherence which can be exploited in a hierarchical way. Subsection 2.2 presents a brief description of the techniques proposed in previous papers and we discuss how these algorithms match with our characterization. In section 3 we propose the implementation of new uses of coherence at different levels of the hierarchy previously proposed. Finally, we present, in section 4, the results obtained with these implementations as well as a discussion of the complexity of the proposed techniques.

2 Use of coherence in radiosity

In this section we propose a characterization of the different types of coherence that may be useful to speed up the visibility computation for form factor evaluation. Some of them have been already used, others not yet. The suitability of each kind of coherence will depend on several parameters. Essentially, on the type of implementation (sequential / parallel, software / hardware) and on the complexity of the environment (number and location of the objects, number of surfaces, number of patches). In the present work, we do not take in account techniques for accelerating interactive walkthroughs in architectural models (Teller and Séquin, 1991), nor a technique based on the use of virtual walls (Arnaldi, Pueyo and Vilaplana, 1991).

Figure 1: a) An example of environment. b) Frustrums

2.1 Characterization

Our characterization is based on a top down analysis of the environment, as usual in visibility computation. We first assume that a certain type of structure can be built in order to organize the elements (objects or/and surfaces) of the environment. This technique has been widely used in ray tracing algorithms and it seems reasonable to explore in this direction. Going down in the analysis, the environment can be considered as a set of surfaces (we assume here that all surfaces are polygons) or as a set of objects which are in turn composed of surfaces. Finally, the surfaces are split up on patches (see figure 1a). We do not consider, in this first analysis, the patches being composed of elements. The features of the patches discussed in the following sections can be extended to elements.

In a surface, the visibility function must be performed for each of his patches in order to get their form factors. In fact, this function is computed several times for each patch; once for each frustrum (figure 1b) if we use the hemicube technique. So the results for neighbouring frustrums and patches will presumably be similar. Let's take a surface S (figure 1a) of the environment. Global clipping, view transformations and depth sort can be performed for all the patches of S . Since a view point on S is inside the environment, not all the objects are needed for a full visibility computation from this point. The plane of the surface splits the environment in two partitions: one containing all the likely objects to be visible (the visible half-space) and the other containing hidden objects (the hidden half-space). The first partition provides a **candidate set** of surfaces (or objects) for all the patches of S . Depending on the environment, orientation and location of a surface, its candidate set can be small in respect to the full environment, and thus, the cost of the visibility computation will be reduced for all the patches of the surface. The worst case (greater size of the candidate set) could be located on the walls of the enclosure. The surfaces (or objects) of a candidate set may, also, be presorted in order to store depth priority information valid for all the current patches. These global procedures may be helped by a suitable organization (data structure) of the environment.

Once the candidate set (with or with out depth information) has been obtained, we must compute the visibility for each patch P of S (figure 1a); in fact for each frustrum F of P (figure 1b). The goal of this step is to determine which patches of the candidate set are visible in the current frustrum; but, actually we can first compute the visibility at surface level. This can be performed in two ways:

Figure 2: Strategies of visibility computation for form factor computation at surface and patch levels

- Compute the visible surfaces and afterwards derive which are the visible patches (figure 2 a).
- Filter the fully invisible surfaces and later compute the visibility at patch level (figure 2 b).

In order to speed up the process of a patch (respectively a frustum) we can use the result of the neighbouring patches (frustrums) of the same surface (respectively patch) previously computed. This patch-to-patch (frustum-to-frustum) coherence has a strong relationship with the frame-to-frame coherence used in animation.

2.2 Previous work

Two structures have been proposed for radiosity environments in order to speed up the visibility function evaluation. Wang and Davis (1990) use an octtree of the environment which is, in addition, used for patch generation. Haines and Wallace (1991) proposed the use of a hierarchy of bounding boxes. Instead of structuring the environment, Campbell and Fussell (1990) proposed the use of the BSP trees of each object.

Marks *et al* (1990) and Haines and Wallace (1991) proposed solutions to exploit surface coherence by means of studying the presence of occluding objects in a volume defined by the current surface (S) and a surface of the candidate list. Actually, this idea was first exploited by Nishita and Nakamae (1985). At surface level we find another approach in (Campbell and Fussell, 1990). In this paper the authors compute visible surfaces to reduce the candidate list based in the BSP organisation of the surfaces of each object.

Algorithms based on a depth sort of the surfaces (or patches) from Wang and Davis and from Campbell and Fussell process the candidate list in front to back order. In z-buffer based

algorithms, this technique allows the stopping of the process when a surface has been assigned to each buffer element.

Tellier *et al* (1991) has proposed an algorithm to exploit patch-to-patch coherence. The visible element for a patch in a given direction is derived from the visibility of the previously processed patch in the same direction.

3 Proposed solutions

As stated above, before proceeding to the visibility computation of the patches of a given surface, we must determine its candidate set. To do this, we must classify all the objects of the environment according to their inclusion in one of both half-spaces bounded by the plane of the surface. We must pose some data structure of the environment that helps do this efficiently. Clearly, this data structure must be related to the sorting problem of the environment. In the past, some data structures have been proposed (BSP for hidden-surface removal (Fuchs, Kedem and Naylor, 1980), bounding volume hierarchies, uniform space subdivision (Fujimoto, Tanaka and Iwata, 1986), adaptative space subdivision for ray-tracing (Glassner, 1984)) lightening the sorting problem with a view independent pre-process and thus reducing the rendering cost because of the reduced search attained in a structured environment.

3.1 Environment's data structure

In our study, a hierarchical structure of the environment may be attractive if it offers a good tree pruning when we search the candidate set. A hierarchical bounding volumes or octree space subdivision could offer it. In any case, the extraction of a candidate set must be done more efficiently than traversing all the objects of the environment (Selecting it by clipping and culling).

Traversing the data structure will consist of testing the supporting plane of the surface at issue against the volume (voxel or bounding volume) associated at the non-terminal node and three cases can occur:

- The volume is in the visible half-space: All their children are in the visible half-space and all the leaves (the objects) of the sub-tree must be added to the candidate list.
- The volume is in the hidden half-space: All the sub-tree must be rejected and all their leaves will not be added to the candidate list.
- The volume is in both half-spaces: We must test the plane against the volumes of the children and proceed as before. If the node is a leaf, we must test the plane against the bounding volumes of the objects to decide which half-space they are in. The objects in both half-spaces will be included in the visible half-space and will be clipped afterwards.

A test of containment of a volume in one or both half-spaces defined by a plane can be made by checking each vertex of the volume on the equation of the plane (six floating point operations per vertex). Based on this test, a data structure must be chosen according to the following criteria:

- **The cost of the test:** The bounding volumes or subdivision space cells with a few possible vertices are desired because of the cost of the test. If one volume bounds an

object with less vertices than the volume itself, a classification test to the object instead of the volume is preferable.

- **The number of the tests:** The bounding volumes or subdivision space cells must be adjusted to the minimum space of its containment to avoid the resolution of the classification test in high levels of the hierarchy in those cases which can be done at lower levels.

The data structure proposed here is based on the binary space subdivision; i.e. a BSP (Fuchs, Kedem and Naylor, 1980; Kaplan, 1987) using similar subdivision rules as a bintree (Samet and Tamminen, 1985). The root of the tree contains a bounding box of the full environment. Each non terminal node of the tree contains one of the box cells resulting from the partition of the box of its parent by a chosen splitting plane. The extent of the cell includes the space contents of its descendants. A leaf node has also a box cell of the list of objects which it points to. The proposed subdivision method does not split the objects of the environment, i.e. the space is not subdivided according to a fixed rule as for example, subdividing one cell into other cells of the same size (As is done in (Fujimoto, Tanaka and Iwata, 1986; Glassner, 1984; Wang and Davis, 1990)). The subdivision is made leaving whole objects in the resulting cells. Thus, the cells at the same level of the tree may be of different size and a separating plane between objects of both cells must exist (see figure 3).

We consider it reasonable not to subdivide beyond the objects of the environment as the basic query that the data structure must help is the candidate set extraction. Note that the aim of the data structure is to collect objects contained in one half-space, and not the location of a surface point as occurs in ray tracing. However, if ray tracing is used, the proposed data structure can also be exploited.

The candidate set is composed of the surfaces of all the objects contained in the cells which are in the visible half-space and the cells which are in both half-spaces (from now on called intersected cells). These intersected cells can include surfaces outside the visible half-space. Thus, the candidate set obtained traversing the data structure is not exactly the ideal. In fact, we must test the surfaces included in the intersected cells for inclusion in the candidate set. An alternative could be to build a finer subdivision using another rule, (for example, an octree (Glassner, 1984)) and thus, avoiding useless intersecting cells because the resulting small subdivision cells (containing portions of the objects) allow an accurate discrimination. This may be true, but at a cost of more cells for testing when we are extracting the candidate set. Although the complexity of the objects in the environment can be a factor in deciding which size-level of subdivision may be the best, there are other conditions affecting the choice in addition to the cost of the extraction. First, the cost of building the structure must be added to the time of form factor computation, although this cost is outstandingly less than the other cost. Second, the extraction of the candidate set is made for a given surface and the same set is used for the form factor computation of all the patches of the surface. Thus, according to the density of patches per surface, the extract operation is less frequent than the form factor computation per patch and savings in cost of the extraction may be insignificant.

The choice of space subdivision cells in the environment structure is due to the inclusion of the walls in the structure. An alternative to the proposed structure that seems suitable to our intention could be a bounding volumes hierarchy (Rubin and Whitted, 1980; Weghorst, Hooper and Greenberg, 1984). In principle, its strength lies in having tighter bounding volumes per object than space subdivision volumes avoiding the inclusion in the candidate set of some volumes lying in both halfspaces. For example, in figure 3 we can see how surface s (its supporting plane

Figure 3: An environment and its corresponding tree. Dashed lines below the nodes suggest the corresponding partition made to the environment. The dotted line shows the supporting plane of the surface s (see text).

is the dotted line in figure 3) has a candidate set consisting of the subdivisions cells C, D, F, G, B and E . The bounding volumes of the objects would avoid the inclusion of B and E . But if we include the walls, (note that we must include portions of the walls of B and E in the candidate set) high level bounding volumes tend to become subdivision cells and thus lose the advantage of tight adjusted volumes. Moreover, another pending question is the cost of the in-half-space test which could be slightly more expensive than using the orthogonal subdivision cells. The latter can share vertex classification while we cannot assume it using orthogonal box bounding volumes. Clearly the use of other bounding volumes more adjusted to the shape of the object can affect the cost more.

A strategy for finding a balanced tree is required. Balanced trees are good for inserting new elements in the tree (Torres, 1990) accelerating its building. Depending on the query posed, a fast pruning can be made on a balanced tree, discarding half the objects of the environment at the first levels of the tree. The degree node of the tree (2) also reduces the cost of the pruning in relation to an octtree (8).

To build the hierarchy, the extents of all the surfaces of the environment are projected onto the world coordinate axes. The walls of the enclosure are excluded from this phase of building. Each axis will contain intervals filled by the extents and likely void intervals. If there is one, a void interval that splits the environment into two pieces of approximately the same number of objects (a median cut) is chosen. If a void interval is not found, the subdivision is finished at this axis. At the same level of recursion, we try the splitting on another axis. If no axis has void intervals, the subdivision is finished. A tolerance parameter must be decided giving a criterion of what is approximately the same number of objects. Note that this approach is only applicable to environments like that shown in figure 3 (objects can be separated by planes aligned to the axis) and are frequently found in architectural models where radiosity is applied.

Once the tree is built, the walls are inserted in the tree in a different way to the internal objects. The insertion of the wall is made by splitting it so that it can be fitted in the matching subdivision cell.

Figure 4: Fill-area test

3.2 Surface coherence

The patches belonging to a surface share a set of particular properties of the surface. We have seen it in the case of a candidate set: the planarity of the surface makes all their patches share the same candidate set. In the following paragraphs we will find other useful properties which we can exploit.

If a surface is not fully visible from a viewpoint, neither all its patches are visible. The reverse is also true. If partial visibility occurs, some patches may be partially or fully visible and others will not. For example, culling surfaces removes their patches for the remaining stages of the visibility algorithm. Since the cost of solving the visibility depends on the number of objects, it is plausible to design a strategy consisting of determining first the visibility at the level of surfaces, and afterwards the visibility of the patches from the visible surface solution.

In figure 4, we have a timing graph of a test consisting of projecting patches of surfaces of different size onto the buffer. The polygon filling algorithm used is a scan line z-buffer. The cost of this algorithm depends on the area of the buffer to be covered. But is not the same to project a surface, calling the algorithm once as to call it as many times as the number of patches of the surface. Data arrangement, initializations, and overlap of pixels between patches can introduce a significant overhead.

Some contrasts can be seen in figure 4. For example, a surface with 16 patches, covering an area of $4 \cdot 4$ pixels, suffers a cost increment of 350% over the same surface projected without patches. The same case for a surface area of $511 \cdot 511$ pixels, causes an increment of 3.27%.

If we know in advance that small pixel areas must cover the buffer, a likely fact, we can first project surfaces, and afterwards, using the solution of the buffer, project only the patches of the visible surfaces. Visibility solutions based on pixel output have the disadvantage of inspecting all the buffer to know which surfaces are visible. The cost of the operation will be $O(3 \cdot R^2)$, where R is the resolution of the hemicycle in its longest side. If the candidate set is sorted by depth, a front-to-back technique, as proposed by Wang and Davis (1990), can be used. With its advantages, visible surfaces can be known after its completion.

Figure 5: visibility regions on a plane containing hemicubes

3.3 Patch-to-patch coherence

As stated above, all the patches of a surface share the same candidate set for the visibility computation, and thus, determining the candidate set is done once for all the patches of a surface. Visibility computation based on classic hidden surface methods (Sutherland, Sproull and Schumaker, 1974), computes the view transformation as the first stage. This stage involves a significant floating point computation. Since all the patches of a surface are on the same plane, view transformations from the centre of one patch to the centre of another are related to a translation vector joining both centres and with only view components x, y . Depth or z information of the view coordinates is the same for all the patches of the surface. Keeping the view coordinates of the candidates for a first patch, we can compute incrementally the view coordinates for the remaining patches of the surface. Savings in floating computation will be made: Two floating point sums versus nine sums and nine products.

Two observations from the visibility of a plane related to a viewpoint give some mathematical tools helping the hidden surface removal problem. First, considering oriented planes, the sign of the dot product of the plane's normal and the view vector indicates if view vector can see the plane. For an object consisting of planar surfaces with their normals pointing to the outside of the object, the former test is useful for identifying front and back-faces. Removing the back faces we partially solve the visibility problem of the faces of a single object (Full solution if it is convex).

The second observation is made noting that a plane defines two half-spaces: a visible and a hidden half-space. The plane's normal points to the visible half-space. Only a view point lying in the visible half-space can see the plane with the appropriate view vector direction.

A traversal of a BSP (Fuchs, Kedem and Naylor, 1980) using this test, sorts the faces in Back-to-Front or Front-to-Back from a given viewpoint (Here, the visible and hidden half-spaces are seen as front and back half-spaces respectively). This fact defines regions in the space where a fixed set of faces of an object are visible for a view point lying in a region as explained in (Sutherland, Sproull and Schumaker, 1974). Examples are given in figure 5.

In figure 5 the ground line stands for a surface where we must place hemicubes centered in the patches of the surface (these patches are not represented in the figure). Thus, any viewpoint (centre of a patch) is moving always through the ground line. In front of these possible viewpoints

<i>From</i>	<i>to</i>
Left:	$(x', y', z') = (y, z, x)$
Right:	$(x', y', z') = (-y, z, x)$
Front:	$(x', y', z') = (x, z, -y)$
Back:	$(x', y', z') = (-x, z, y)$
Bottom:	$(x', y', z') = (x, -y, -z)$

Table 1: View Exchange coordinates between frustrums

we find the surfaces a, b, c, d, e, f, g . The supporting planes of these last surfaces define regions in the space where a viewpoint can only see a set of surfaces, and others not. These supporting planes intersect the ground line (our surface where we want to place the viewpoints) marking regions where a set of surfaces are visible. Thus, a viewpoint located on the left region of the ground line can only see the surfaces b, d, g as the figure 5 indicates.

To distinguish the surface where we place the hemi-cubes from the others we will name the remaining surfaces as facets. If plane's equation of a facet, $A \cdot x + B \cdot y + C \cdot z + D = 0$, is given in the view coordinates of a chosen patch, intersection of facet plane against the patch plane is found easily. Since the patch plane equation in view coordinates is $z = 0$, the splitting line equation is $A \cdot x + B \cdot y + D = 0$. The sign obtained substituting the patch center in the last equation will indicate if the patch sees the facet at issue (positive). The center of the patch in view coordinates is $(0, 0, 0)$ and thus, we must only check the sign of D . Further hemi-cubes placed in the same plane of the surface can take advantage of the view transformation of the former patch adding the difference vector between the two related patch centers (Vilaplana and Pueyo, 1991). The new visibility check for a face will consist of evaluating $A \cdot \Delta x + B \cdot \Delta y + D$ where Δx and Δy are the new patch center related to the view coordinates of the first patch. Dividing the equation by A , we can reduce the operations to execute: two sums and one product per face. We note that this visibility decision is for a facet, i.e., a surface, and thus, its patches are involved in the decision.

3.4 Frustrum-to-frustrum coherence

If we use the hemicube technique (Cohen and Greenberg, 1985) for form factor computation, we can introduce some improvements in the first formulation of the algorithm (Vilaplana and Pueyo, 1991). As is well known, five visibility solutions are needed on the hemicube, corresponding to its five facets (frustrums). Some stages of the visibility process for a view frustrum, like the view transformation and clipping, can be shared by the remaining frustrums.

View transformation coordinates of the lateral frustrums can be obtained by exchanging the top frustrum view coordinates and probably some changes of sign as is showed in Table 1. Clipping can be done at the same time for all the frustrums. A traditional Sutherland-Hodgman's clipping algorithm can be slightly modified in order to clip polygons for multiple views (Sutherland and Hodgman, 1974). The resulting algorithm decides the following plane to clip according to a decision tree showed in figure 6. A path of this decision tree can be seen as a pipe of planes of the traditional clipping algorithm. The path is decided according to the result of in/out point-plane test. The effect of the algorithm is that instead of rejecting vertices outside the current view clipping plane, they are supplied for clipping against planes of another adjacent view frustrum. On the other hand, the computed intersections of a clipping plane are used for the two adjacent views sharing the plane, thus avoiding repetition of the same computation

Figure 7: Radiosity timing with Hemicube resolution of 50 in VAX 6410, 9 patches per surface

and $T_{exchange} \ll T_{ViewTransf}$, whileas

$$v_f = 5 \cdot (T_{ViewTransf} + T_{FaceClip}) + T_{scan}$$

and $T_{HemicubeClip} < T_{FaceClip}$. Using surface coherence, allows the reduction of the cost of the view transformation stage using incremental computations and improving v .

Structuring the scene, provides new changes to the cost

$$T(N, R, S) = i + N \cdot (p + i' + \sum_{j=1}^N C(\vec{V}_j) \cdot v + 3 \cdot R^2 \cdot s + S \cdot e) \quad (3)$$

where, $C(\vec{V}_j)$ is the number of candidates for the visibility of the current patch and depends on the view vector $\vec{V}_j$ placed on the patch, and S is the number of the surfaces of the environment. The full cost of determining the candidates is given by the cost (e) of extracting it per surface. All the patches of one surface, share the same candidate set. Thus, equation 3 gives the best results with respect to equation 1, when for each patch we find frequently $C(\vec{V}_j) < N$ and there are more patches per surface. Note, that in a lot of cases, the walls of the fence have $C(\vec{V}_j) = N$. Objects inside the fence are the candidates for finding $C(\vec{V}_j) < N$.

This work was developed on a VAX 6410 using VAX-PASCAL language. In the implementation we have tested three radiosity algorithms. The first, named Classic in the following figures, is the high level first formulation of the hemicube technique (Cohen and Greenberg, 1985). The second uses patch-to-patch coherence and frustum-to-frustum coherence (sections 3.3 3.4) and is named as partial coherence in this discussion. The third uses the data structure here proposed (section 3.1), exploiting also the coherences used in the second algorithm, and is named full coherence in the figures. The test environments consist of a cube which inside contains a variable number of small cubes uniformly distributed. Figure 7 shows the timings for the three algorithms using a fixed division of each surface to 9 patches (54 patches per cube). Clearly, the time graphs suggest a quadratic law, but the increase is less when we use coherence.

The resulting speed-up is showed in figure 8. As the complexity of the scene increases, the speed-up improves by the use of the data structure, while the use of the partial coherence gives a constant speed-up in fact.

Figure 8: Speed up with Hemicube resolution of 50, 9 patches per surface

Tests with the same set of environments were made with different resolutions of the hemicube (30, 50, 70, 100) to find out the behavior of the proposed algorithms. In figure 9 we can see that as the resolution of the hemicube increases, the speed-up is slightly reduced. This is due to the fact that the cost of the scan conversion becomes a dominant cost factor. We expect in the future that the use of the ideas explained in subsection 3.2 will improve this behavior.

Another graph in figure 10, gives an idea of the contribution of the patch-to-patch coherence in the speed-up of the full coherence version.

5 Future work

We have seen how coherence may improve the performance of a basic radiosity algorithm. Now the new uses of coherence proposed above should be improved by means of making them cooperate with other, previously proposed, kinds of coherence. This will include good reliable comparisons between different techniques with similar goals.

Whether each type of coherence (of those proposed here and others) is or not suitable to be used for progressive radiosity should be studied. Some will, probably, be used without changes; but others will be adapted in order to be useful for progressive refinements. On the other hand, adaptive subdivision should be feasible without loss of the improvements introduced by the use of coherence.

Finally, the use of coherence for visibility computation will be extended to algorithms dealing with environments with diffuse and specular reflections. In this case coherence for radiosity algorithms will cooperate with the uses of coherence widely exploited for ray tracing.

Acknowledgements

The authors wish to thank P. Brunet and A. Vinacua for their helpful advice and various valuable observations.

Figure 9: Speed-up for full coherence on resolution of the hemicube (9 patches/surface)

Figure 10: Speed up for full coherence on number of patches per surface (Resolution of 50)

References

- Arnaldi, B., Pueyo, X., and Vilaplana, J. (1991). On the division of environments by virtual walls for radiosity computation. In *Advances on Rendering*. Springer-Verlag. Eurographic Seminars Series, In preparation.
- Campbell, A. and Fussell, D. S. (1990). Adaptive mesh generation for global diffuse illumination. In *ACM SIGGRAPH Conf. Proc.*, pages 155–164. Computer Graphics vol.24 num.4.
- Cohen, M. F. and Greenberg, D. P. (1985). The hemi-cube. a radiosity solution for complex environments. In *ACM SIGGRAPH Conf. Proc.*, pages 31–40. Computer Graphics vol.19 num.3.
- Fuchs, H., Kedem, Z., and Naylor, B. F. (1980). On visible surface generation by a priori tree structures. In *ACM SIGGRAPH Conf. Proc.*, pages 124–133. Computer Graphics vol.14 num.3.
- Fujimoto, A., Tanaka, T., and Iwata, K. (1986). Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26.
- Glassner, A. S. (1984). Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22.
- Haines, E. A. and Wallace, J. R. (1991). Shaft culling for efficient ray-traced radiosity. In *Second EUROGRAPHICS Workshop on Rendering*.
- Kaplan, M. R. (1987). The use of spatial coherence in ray tracing. In Rogers, D. F. and Earnshaw, R. A., editors, *Techniques for Computer Graphics*, pages 173–193. Springer-Verlag.
- Marks, J., Walsh, R., Christensen, J., and Friedell, M. (1990). Image and intervisibility coherence in rendering. In *Proceedings of Graphics Interface*, pages 17–30. Canadian Information Processing Society.
- Nishita, T. and Nakamae, E. (1985). Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *ACM SIGGRAPH Conf. Proc.*, pages 23–30. Computer Graphics vol.19 num.3.
- Rubin, S. M. and Whitted, T. (1980). A 3-dimensional representation for fast rendering of complex scenes. In *ACM SIGGRAPH Conf. Proc.*, pages 110–116. Computer Graphics vol. 14 num. 3.
- Samet, H. and Tamminen, M. (1985). Bintree, csg trees, and time. In *ACM SIGGRAPH Conf. Proc.*, pages 121–130. Computer Graphics vol. 19 num. 3.
- Sutherland, I. E. and Hodgman, G. W. (1974). Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42.
- Sutherland, I. E., Sproull, R. F., and Schumaker, R. A. (1974). A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1):1–55.
- Teller, S. J. and Séquin, C. H. (1991). Visibility preprocessing for interactive walkthroughs. In *ACM SIGGRAPH Conf. Proc.*, pages 61–69.

- Tellier, P., Maisel, E., Bouatouch, K., and Languenou, E. (1991). Using coherence to accelerate radiosity. Technical Report 616, IRISA Campus de Beaulieu, Rennes Cédex, France.
- Torres, E. (1990). Optimization of the binary space partition algorithm (bsp) for the visualization of dynamic scenes. In *EUROGRAPHICS Conf. Proc.*, pages 507–518.
- Vilaplana, J. and Pueyo, X. (1991). Exploiting coherence for clipping and view transformations in radiosity algorithms. In Bouatouch, K. and Bouville, C., editors, *Photorealism in Computer Graphics*, pages 137–149. Springer-Verlag. Eurographic Seminars Series.
- Wang, Y. and Davis, W. A. (1990). Octant priority for radiosity image rendering. In *Proceedings of Graphics Interface*, pages 83–91. Canadian Information Processing Society.
- Weghorst, H., Hooper, G., and Greenberg, D. P. (1984). Improved computational methods for fast ray tracing. *ACM Transactions on Computer Graphics*, 3(1):52–69.

Contents

1	Introduction	1
2	Use of coherence in radiosity	1
2.1	Characterization	2
2.2	Previous work	3
3	Proposed solutions	4
3.1	Environment's data structure	4
3.2	Surface coherence	7
3.3	Patch-to-patch coherence	8
3.4	Frustrum-to-frustrum coherence	9
4	Conclusions and results	10
5	Future work	12

List of Figures

1	a) An example of environment. b) Frustrums	2
2	Strategies of visibility computation for form factor computation at surface and patch levels	3
3	An environment and its corresponding tree. Dashed lines below the nodes suggest the corresponding partition made to the environment. The dotted line shows the supporting plane of the surface s (see text).	6
4	Fill-area test	7
5	visibility regions on a plane containing hemicubes	8
6	Decision tree for hemicube clipping planes. Left (right) branches means that the resulting plane/point test has given <i>in</i> (<i>out</i>). Nodes are clipping planes labeled according to the direction of their normals.	10
7	Radiosity timing with Hemicube resolution of 50 in VAX 6410, 9 patches per surface	11
8	Speed up with Hemicube resolution of 50, 9 patches per surface	12
9	Speed-up for full coherence on resolution of the hemicube (9 patches/surface)	13
10	Speed up for full coherence on number of patches per surface (Resolution of 50)	14

List of Tables

1	View Exchange coordinates between frustrums	9
---	---	---

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

List of research reports (1993).

LSI-93-1-R "A methodology for semantically enriching interoperable databases", Malú Castellanos.

LSI-93-2-R "extraction of data dependencies", Malú Castellanos and Fèlix Saltor.

LSI-93-3-R "The use of visibility coherence for radiosity computation", X. Pueyo.

LSI-93-4-R "An integral geometry based method for fast form-factor computation", Mateu Sbert.

LSI-93-5-R "Temporal coherence in progressive radiosity", D. Tost and X. Pueyo.

LSI-93-6-R "Multilevel Use of Coherence for Complex Radiosity Environments", Josep Vilaplana and Xavier Pueyo.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es