Form factor evaluation with regional BSP trees

Karel Nechvíle, Jiří Sochor
Masaryk University
Burešova 20, 602 00 Brno, Czech Republic
e–mail: kodl@fi.muni.cz, sochor@fi.muni.cz

Abstract

Form factor evaluation is an expensive and time consuming operation for radiosity applications. Visibility algorithms have to deal with a huge number of patches, rapidly increasing during an adaptive subdivision. This paper presents one possible approach to visibility part of complex radiosity algorithm - dividing the scene to non-overlapping regions and creating hierarchical BSP tree with many regional BSP trees. The regional BSP approach offers possibility to solve radiosity in acceptable time without special HW.

1 Introduction

Radiosity is one of the most important methods that are applicable to photorealistic rendering. Radiosity is predominantly implemented on high-end workstations that have power sufficient to do intensive computations with large volumes of data. Therefore we have tried to seek time and memory trade-off solution that will allow to solve radiosity on low-end stations. Modified method reduces the intensive and time consuming computations in the most critical part of radiosity algorithm, namely form factor computation. We use common paradigm; to speed up repeated computations we rely on preprocessing and additional memory.

2 Scene preprocessing

Form factor evaluation is the most difficult part of radiosity computation. Choosing the classical hemicube [2] for form factor computation, one has to solve the visibility of the whole scene in every iteration.

Form factor part of the radiosity algorithm solves the scene visibility for thousands of different places. For the scene with constant geometry it is worth seeking some sorting, that would help to speed up repeated calculations. We have chosen the following approach:

> *Our form factor solution uses painter's paradigm. We project pre-sorted patches on hemicube and paint them without z-buffer.*

This pre-processing has the following advantages: We don't need to interpolate and store the depth information in every hemicube pixel. Software solution of hemicube is

thus more simple without additional memory for software z-buffer. On the other hand we need to store information for alternate visibility solution. The proposed method is suitable for workstations with sufficient memory; no hardware z-buffer is needed. [1]

3 Scene sorting with BSP tree

A BSP [2] tree provides a simple and natural solution. With the help of BSP sorting one can easily establish the scene visibility from every viewpoint. However, it is impractical to partition the scene with only one BSP tree. BSP sorting can produce extremely large number of divided patches and this is inacceptable. Therefore we have chosen the hybrid solution: we decompose a scene into regions and use regionally restricted BSP sorting. The resulting data structure is named *the regional BSP tree* and is used to establish the local visibility of a patch in the region.

Our solution has been motivated by the following reasoning:

Radiosity scenes are defined as closed environments with active light sources [3]. Every object is described by B-rep model. There should be (almost) no problem to add BSP sorting to every object. CSG modeller is able to forecast the local visibility and to stamp it on an object.

If we have been successful in the placement of individual objects or small groups of objects into unambiguously defined subspaces and sorting objects inside these regions (with BSP trees), then we only need to decide the visibility on the regional level. For this task we also use a modified BSP tree that contains information about planes that separate the regions. Leaves of the regional BSP supertree point to the BSP subtrees of individual regions.

Ceartainly it is not easy to decompose a scene into non overlapping and easily sortable regions. Any algorithm starting from zero without any information about a scene, tends to give many different and unstable results. We have avoided this problem by omitting all these facts. The most of objects are solids and they do not penetrate each other. CSG modeller knows the way of processing an object and all constraints following its introducing. Therefore we believe that sorting information can and will be provided by someone else (perhaps a more clever fellow).

In fact, we do not need to store any physical object as one solid. We can split each object into several logical parts and even cut it into small pieces. For instance, an object "table" can be saved as one object or as a "whole-part" model of a desk and legs. In the latter model we would have an easier task to find suitable separating planes.

The different approach is similar to Warnock algorithm. A scene can be divided with respect to the number of patches residing in a specific part of the scene. If the number of patches exceeds the threshold value, the patches in subspace will be clipped to the specified volume and a subspace will be treated as a new region. The regional BSP tree is then constructed for the patches clipped to the subspace. In our implementation we use simple, axis oriented parallelepiped regions.

[1] That does not mean we exclude z-buffer completely. It is useful during scene rendering and it can speed up other algorithms as well.

[2] Binary Space Partitioning

[3] Light sources are useful if you want to publish at respectable conferences.

4 Data structures with pre-processed scene information

Topological relations are represented by the *winged-edge* structure (WES). WES contains the information about the solid's geometry and its neighbourhood relations. WES is supplied with additional information about surface/patch hierarchy. Hierarchy information is useful for the subdivision process. We have also added information about the regional subdivision of a scene. Figure.1 shows an example of a hierarchical data structure. This structure is used by the algorithm in Figure 2.

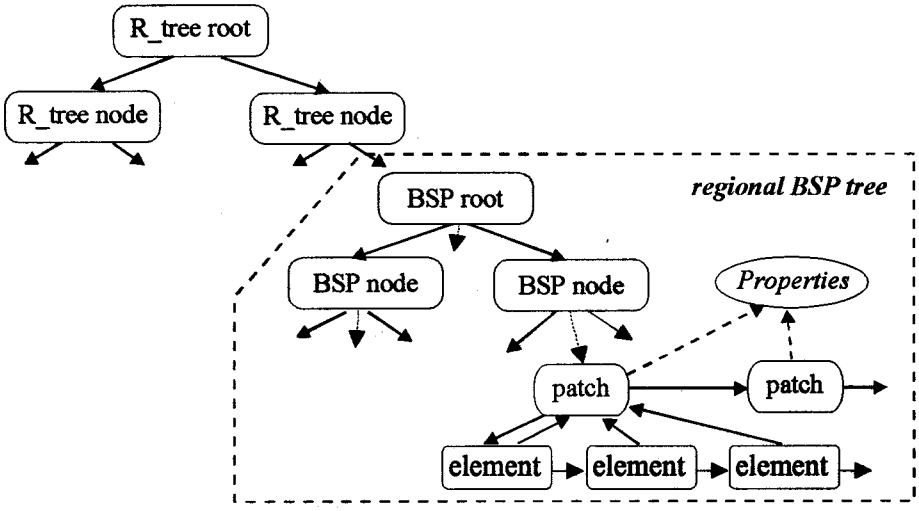

Figure 1: Data structures for scene subdivision and patch hierarchy

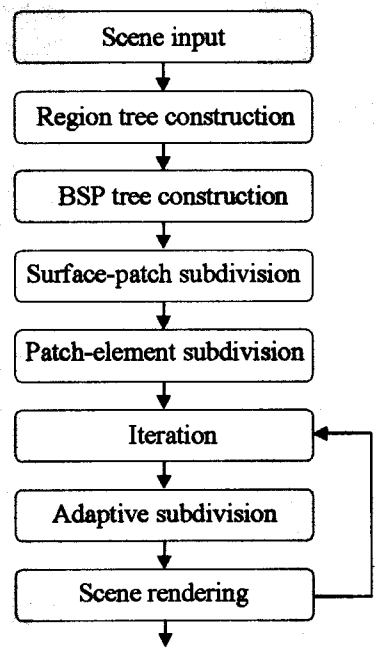

Figure 2: Radiosity with scene pre-processing

5 Radiosity and regional BSP trees

5.1 Assumptions

A scene is divided to subspaces that can be separated by planes into non-overlapping regions. The model of every region contains planar convex polygons (faces) and topological information (neighbourhood faces). Polygon attributes define the diffusion reflection coefficients for primaries R, G, B and initial non-shooted energy per surface unit.

5.2 Construction of regional tree

The algorithm finds the planes that separate regions. Only one region is left in every subspace. This way algorithm decides the regional visibility (Figure 3). With axis aligned planes, regions are restricted to parallelepipeds in 3D space.

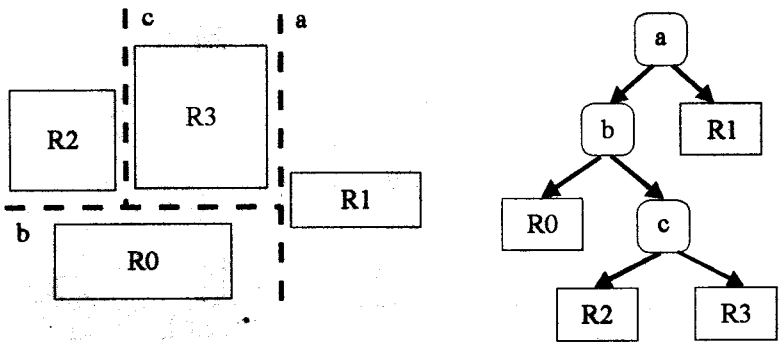

Figure 3: Regions, separating planes and regional tree (2D)

The search for separating planes is done as follows:

Using the bounding volume information, algorithm sorts regions independently for every axis (x, y, z) and creates 3 lists containing bounding volume projections to axises - limit lists. In the next step the algorithm searches lists for a place where the separating plane could be placed (a test checks the number of objects being projected to an axis). If such a place is found, this list is split and the new regional tree node will store the information about the separating plane. Regions stored in the tree fall in two categories - regions lying in front of the separating plane and regions lying at the back of the separating plane. Algorithm aplies the procedure recursively to front and to back regions. Process is terminated when the region limit-list is empty or contains only one region. Inner nodes of the resulting tree contain separating planes and its leaves store pointers to individual regions (regional BSP trees).

5.3 Construction of regional BSP tree

BSP tree is built by a well-known algorithm [3]. The algorithm selects a patch defining a separation plane and builds recursively lists with patches belonging to the front half space and to the back half space. For a root-patch selection we have used no heuristic; we rely on a well-balanced BSP model supplied by a pre-radiosity modelling process. The heuristic methods are discussed in [4].

An input face is split according to our BSP sorting. This is done before the radiosity refinement process starts its own adaptive subdivision. The parts of a splitted face have

common attributes (reflection, normal vector etc.). We store these attribute values out of BSP tree in the node called *Properties* (Figure 1).

5.4 Faces splitting to patches

The quality of a radiosity solution depends heavily on the subdivision process. We have used the results published in [1]. Triangulation is done as follows:

We transform a face (rotation + translation) to achieve more convenient position for further processing. The face is then split by a horizontal strip of a chosen width. The strip is triangulated in L direction. Two triangles are added to complete the strip at both ends and to ensure the convexity of a remaining shape. We repeat the procedure to triangulate the rest of the face (Figure 4).

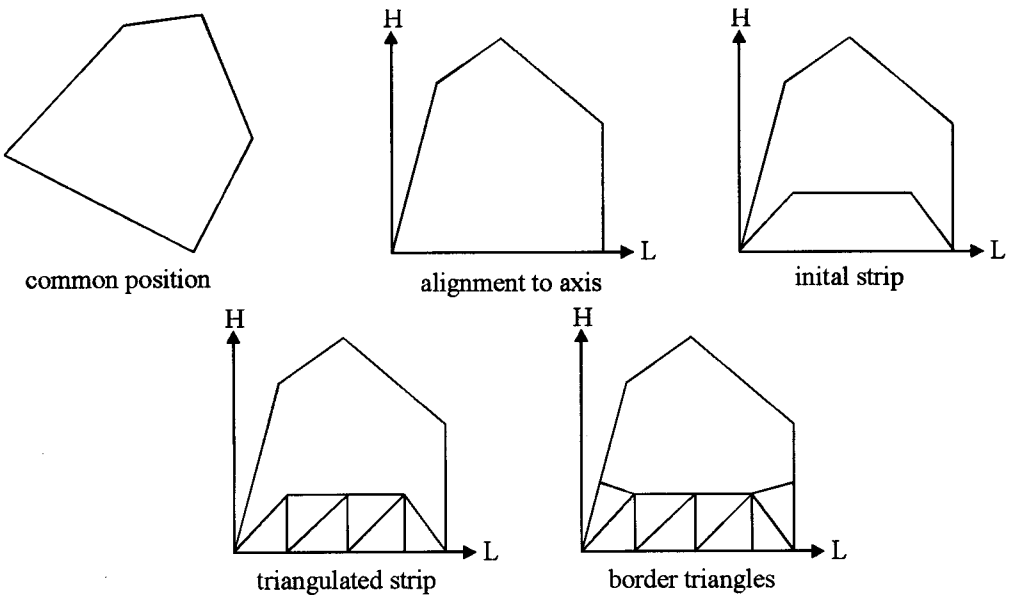

Figure 4: Face splitting to triangular patches

We receive the set of triangle patches with attributes of the original surface. Patches are chained and stored in one BSP node (Figure 1 - *patch* nodes).

5.5 Patch splitting to elements

The process that splits surfaces to patches produces patch-element tupples. To evaluate the radiosity gradients more precisely we need a mesh consisting of small area elements. The shadow borders can be estimated and used for the initial subdivision. Several iterations for light sources would provide some information about shadows and this enables to apply a more sophisticated patch subdivision.

Again, we have not used any of these approaches. We only specify an element/patch ratio assessing the density of the element-mesh compared to the patch-mesh. We keep data structures as simple as possible to lower the processing overhead.

5.6 Iteration

The iteration step of our algorithm follows the classical refinement step of the shooting radiosity. The tree traversal process gives the element ordering. The patch with the

most energy left is selected and a temporary co-ordinate system for a hemicube is set.

The algorithm projects all patches on the hemicube in a given order. The regional tree and BSP trees are traversed from back to front giving correctly depth-sorted patches/elements for a painter algorithm. Therefore we need not interpolate and store the depth.

In one pass for element edges we get the element projection on every hemicube wall. Projected borders are then scan-line filled with the element identifier. We speed the process with some tricks based on coherence information.

Another test tries to exclude the whole region from further processing. We use the bounding volume test. If the region passes the test, then BSP tree is traversed and more detailed tests are applied on the patch/element level. When all elements are projected, algorithm calculates from the hemicube-wall information all form factors by simple addition using pre computed *delta form factors.* BSP traversal process is described in Appendix A.

In our implementation we pre compute delta form factors for one quarter of the top plane and for half of the side plane of the hemicube. In this way we get form-factors for every receiving element with respect to selected shooting element.

In the next step tree structures are traversed once again and radiosity increments are placed on receiving elements. Compared with the z-buffer approach, we need not initialise hemicube walls at the beginning of every iteration.

5.7 Adaptive subdivision

The goal of an adaptive subdivision is to compute radiosity more precisely in all areas with notable gradient. This is not the only possible solution but it is used here because of its simplicity.

The first task of the adaptive subdivision is to identify places (elements) that should be divided. Looking for such places, one has to apply some metric evaluating the difference between real and computed values. We have used a simple heuristic based on radiosity difference of neighbouring elements. When the difference exceeds the threshold value, subdivision is done. The lower limit is controlled by the smallest acceptable edge length. The disadvantage of this simple heuristic is that it produces unnecessary subdivision in places with great but constant gradient.

An element is divided to 4 new elements. If neighbouring elements are not divided, then common edges contain T-vertices. Their existence may result in visible radiosity discontinuities. It is also important to produce the smooth transition of the mesh density. We have adopted the method of an element anchoring [1]. When we find T-vertices, we anchor the neighbouring elements (a special edge is added). Figure 5 shows typical configurations during the anchoring and subdivision process.

After the subdivision, one has to establish the initial radiosity for new elements, e.g. by interpolation, and to process these elements by the standard iteration procedure.

6 Scene rendering

Our implementation uses a linear interpolation for a smooth colour rendering. An element radiosity is transformed to vertices by simple averaging calculation. Values for patch edges can be extrapolated from inner-vertex values.

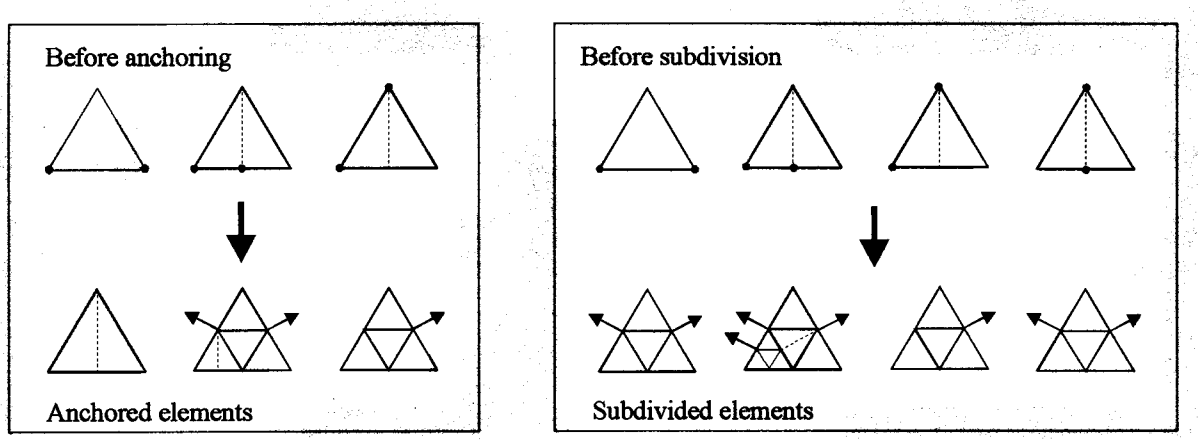

Figure 5: Element anchoring and subdivision

Vertices colours of triangular elements are linearly interpolated and every triangle is Gouraud shaded.

The radiosity can be smoothed by averaging carefully selected neighbour values. Averaging can suppress undesirable artefacts but it can also hide correct, important shadow borders.

7 Results

The modified method has been implemented in C on **Silicon Graphics**. Graphical outputs have been programmed with IRIS GL. GUI has been implemented with Forms library (public domain). Implementation has been tested on SG Indy R4000. We have used several simple testing scenes. Tests have been focused mainly on special cases, such as energy distribution between very close patches. We have also tested an adaptive subdivision, its depth and hemicube resolution, and influence of these factors on a rendering quality.

Due to the hemicube, our method suffers from all related disadvantages. The modified method also tends to alias in cases, where patches are projected on a hemicube with the extreme distortion. Using only painter's algorithm without the depth information, these situations have to be processed with an extra care, e.g. using higher resolution near the centre of the top hemicube wall.

The adaptive subdivision enables to compute shadow borders more precisely. Too fine resolution also leads to aliasing because small and close-by elements project on the same part of a hemicube wall. Aliasing can be also resolved with a higher hemicube resolution.

The choice of resolution is very important for the rendering quality. The radiosity computation based on regional BSP trees is feasible even with higher hemicube resolution. Till now, we have not had an opportunity to process some well known complex scene and to compare the processing time with other methods. Figure 6 shows a simple test scene with two light sources. The subdivided scene contains 5930 elements.

Results of the few experiments are illustrated in Table.1.

The test scenes have been prepared by hand and our future work will be focused on the design of a better user-friendly input model. We plan to experiment with model converters; they will supply additional BSP sorting information. An ideal solution would be to implement a compact 3D scene modeller that would dynamically build and maintain

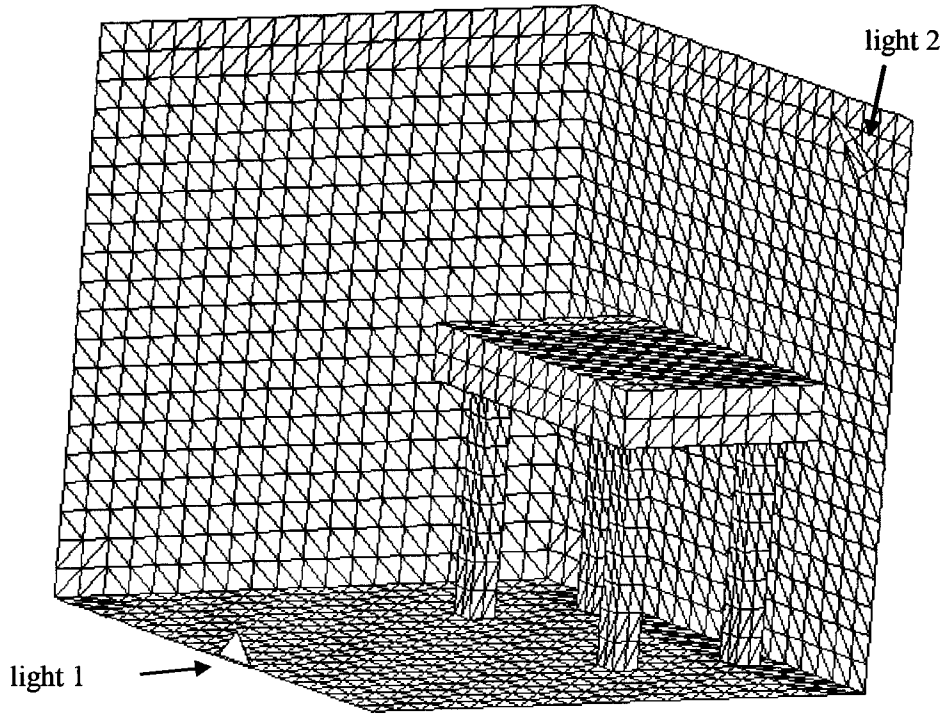

Figure 6: Simple test scene - 2 light sources

hemicube resolution	# elements	time/100 iterations [s]
150 x 150	5930	31.23
300 x 300		52.05
600 x 600		121.34

Table 1: Results for simple scene - SGI R4000

models with additional sorting information and pass them to the radiosity render. The tree of regional BSP trees needs to be thoroughly investigated and evaluated, both theoretically and practically.

References

[1] Baum, D.R., Mann, S., Smith, K.P., and Winget, J.M. *Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions* Computer Graphics, Vol. 25 No. 4, July 1991.

[2] Cohen, M.F., and Greenberg, D.P. *The hemi–cube: A Radiosity Solution for Complex Environments* Computer Graphics, Vol. 19 No. 3, July 1985.

[3] Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F. *Computer Graphics, Principles and Practise, 2nd Edition* Addison-Wesley, Reading, Massachusetts, 1990.

[4] Thibault, W.C., and Naylor, B.F. *Set Operations on Polyhedra Using Binary Space Partitioning Trees* Computer Graphics, Vol. 21 No. 4, July 1987.

A Pseudocode

Assumptions: The patch acting as actual energy source was selected. Painter's algorithm starts in the patch centre and walks through the sorted scene to obtain patches in correct order.

```
void GoSTree (pointer_to_regional_tree Node)

{
 if (first walk through left subtree, then right subtree)
  {
   if (next left node is inner node) GoSTree (Node->left subtree);
    else /* we are in leaf - project region */
        GoRegion (Node->region);
   if (next right node is inner node) GoSTree (Node->right subtree);
    else GoRegion (Node->region);
  }
 else  /* same in reverse order */
 { if (next right node is inner node) GoSTree (Node->right subtree);
    else GoRegion (Node->region);
  if (next left node is inner node) GoSTree (Node->left subtree);
   else GoRegion (Node->region);
  }
}

void GoRegion (pointer_to_region Region)
{
 if (region is partialy visible from given point)
   GoRegionTree (Region->node);
}

void GoRegionTree (pointer_to_BSP_tree BSP_Node)
{ /* traversing BSP tree */

  if (node is empty) exit;

   if (patch is front face as seen from given point)
    {
     GoRegionTree (BSP_Node->back);
     Action for all patches/elements
     belonging to this node of BSP tree.
     GoRegionTree (BSP_Node->front);
   }
   else {
     RGoRegionTree (BSP_Node->front);
     RGoRegionTree (BSP_Node->back);
    }
  }
}
```