Frame-to-Frame Coherent Animation with Two-Pass Radiosity

I. Martín, X. Pueyo

Institut d'Informàtica i Aplicacions
Universitat de Girona
Girona, SPAIN

D. Tost

Departament de Llenguatges i Sistemas Informàtics
Universitat Politècnica de Catalunya
Barcelona, SPAIN

Abstract

This paper proposes an efficient method for the production of high quality radiosity solutions which makes use of the a-priori knowledge of the dynamic properties of the scene in order to exploit temporal coherence. The method is based on a two-pass strategy that provides user-control on the final frame quality. In the first pass, it computes a coarse global solution of the radiosities along a time interval and then, in the second pass, it performs a frame-to-frame incremental gathering step using hardware graphics accelerators. The frame production computing cost is thus reduced because the method takes advantage of frame-to-frame coherence by identifying the changes produced by dynamic objects, and by decoupling them from computations that remain unchanged. The input data is a dynamic model of the environment through a period of time corresponding to the same camera recording. The method proceeds by incrementally updating two data structures: a space-time hierarchical radiosity solution for a given interval of time, and a hierarchical tree of textures representing

the space-time final illumination of the visible surfaces. These data structures are computed for a given viewpoint, either static or dynamic. The main contribution of this work is the efficient construction of the texture tree by identifying the changes produced by dynamic objects and by recomputing only these changes.

1 Introduction

Giving a dynamic behaviour of the objects, as well as providing interactive navigation through the environments greatly improve the perception of realism. A further step in realistic image synthesis is thus the production of animations using global illumination models. Up to now, progress in this field has been limited by the high computational cost of realistic animation synthesis.

In static environments, the energy transport between scene elements remains constant. Navigation through such scene poses the problem of obtaining accurate image quality as the viewer comes closer to specific regions and moves further away from other ones. Walkthrough radiosity algorithms thus require adaptive refinements and coarsening of the energy transport solution. In dynamic environments, the movement of an object causes changes in the visibility between scene elements, and therefore it modifies the energy transport. However, the extent of these changes is generally limited. Computing each frame from scratch does not seem a reasonable choice since successive frames are not too much different one from the other. In order to take advantage of frame-to-frame coherence in dynamic environments, it is necessary to identify changes and to update the previous solution.

There have been several attempts to characterize the visibility changes produced by moving objects in order to incrementally compute the form factors, as well as to update the energy transport. These methods, briefly overviewed in section 2, provide means of interactively manipulating complex radiosity scenes. However, they all focus on updating the global radiosity solution. Global finite-element solutions with reasonable refinement thresholds are sufficient in many interactive applications. However, high-quality realistic rendering, such as it is often needed in animation, requires a final

per-pixel gathering pass in order to provide better visual appearance. The cost of this second pass often ranges from twice to ten times the global pass, making the use of acceleration techniques even more necessary. Up to now, no specific solution has been proposed to speed-up the final gathering step. It is the goal of the work presented herein.

This paper presents a method for the efficient production of photo-realistic animations using radiosity, which can be outlined as follows. With a user given error level, it reconstructs the continuous global illumination function of the environment over a time interval, as it is seen along the camera path. The motion of the objects and the camera is supposed to be known a priori, an hypothesis which, although being restrictive, seems reasonable, taking into account that, in the design stage of an animation sequence, simple local illumination models are generally sufficient. On the basis of this assumption, an incremental two-pass strategy is designed, which computes two data structures: a hierarchical spatio-temporal radiosity solution of the environment, and a spatio-temporal hierarchical texture tree that represents the final illumination of the visible surfaces. These data structures have the minimum size in order to produce a given frame n, and they are updated when frame $n+1$ is requested.

The paper is structured as follows. First, the previous work is reviewed in Section 2. Section 3 presents a brief overview of the method along with a review of a previously published two-pass method for static scenes that is used as the starting point of our work. Next, the method is depicted in detail in Section 4. Section 5 explains how frame-to-frame coherence is used within the method to speed-up computations. Results of the experiments performed are presented in Section 6 to analyze the behaviour of the proposed method. Finally, Section 7 presents the conclusions and the future work.

2 Background

Several radiosity methods have been proposed that deal with dynamic objects in the context of progressive strategies [1, 2, 3, 4], as well as of hierarchical solutions [5, 6, 7]. Most methods suppose an a-priori knowledge of the movement [8, 5] although others

allow restricted interactive changes [7].

A common idea of these papers is to provide means of identifying changes in the visibility as well as in the energy transport. In finite element methods, the visibility tests are embedded in the form factor calculations. The a-priori knowledge of the movements allows the construction of swept volumes that restrict the part of the scene that is affected by the motion. In an early paper [8], the form factors between patches that do not interact with moving objects were identified using this strategy, computed in a preprocess and stored. In addition, modified form factors were incrementally computed by using reprojection techniques. First restricted to 2D [9], and later extended to 3D [10], a mechanism for the prediction of changes in form factors has been proposed, which is based on a geometrical data structures called *visibility complex* and *visibility skeleton*.

In the progressive radiosity context, an incremental computation of the radiosity solution was proposed in [1, 2]. At each frame, moving patches first shoot a negative energy to undo their effect in the previous frame, and then they shoot their current radiosity. This mechanism, called redistribution, has proven to be valuable for interactive indoor design where the frame rate can be low. A similar technique called re-propagation stores a dynamic array called Shadow Form Factor List, which allows us to exactly re-propagate radiosity only on the parts of the scene affected by changes [3]. Although it allows near real-time animations, it is restricted to one change per frame. In addition, as it is based on a fixed initial meshing, artifacts may appear due to a coarse refinement.

In [5], the hierarchical radiosity is extended to dynamic environments by updating the hierarchy of links between static objects affected by moving occluders. Links can be refined, coarsened and temporally occluded. As the moving objects are known in advance, the link events are easily predictable. However, links involving a dynamic patch are recomputed from scratch. The main limitation of this approach is that it cannot handle light source movements. A similar idea is described in [6], which introduces shadow links with information on occluders that allow dynamic events to be easily identified. An approach towards interactive changes in clustering-based hierar-

chical radiosity is presented in [7]. It keeps a line space hierarchy associated to the links between elements of the scene, whose traversal provides a fast identification of the links that have changed and an accelerate solution of the modified system. The major drawback of the line-space hierarchy is its huge memory consumption. This problem is addressed in [11], which propose a dynamic management of shafts storage. In [12] and [13], the idea of developing general frameworks to deal with space-time hierarchical radiosity was proposed.

A different approach from the previously cited is to compute a radiosity solution for a time interval rather than frame-to-frame updating the solution. This is done in [14] by using a global Monte Carlo method, which removes the necessity for performing repeated light transports. In [15], the 4D space (3D space+time) is represented by a set of discrete samples which are eventual views of the scene. Any camera path can then be reconstructed from these samples. This method has proven to be efficient, although choosing the optimum set of samples is an open problem.

In summary, significant advances have been made towards using temporal coherence in radiosity and, specifically, in hierarchical radiosity. However, these previous works deal with the global solution of the system. For high quality rendering, this global solution is not sufficient unless unaffordable refinement levels of the solution are reached. On the contrary, high quality images require a final gathering step [16]. Different two-pass strategies have been developed in the literature [17, 18, 19, 20, 21], but none of them has been adapted to a dynamic environment. Analyzing the cost of the two steps, it is clear that the local pass cost is considerably higher than the global one: from twice to ten times more expensive depending on the type of scene and on the gathering computations. Therefore, the acceleration of the local pass is crucial for reducing the overall computational cost.

Furthermore, temporal coherence is by no means less important at the frame level than globally. In classical key-frame animation, the interpolation between key-frames is precisely based on frame-to-frame coherence. In a recent paper [22], this coherence is taken into account to render walkthrough animation sequences, by combining inexpensive Image-Based Rendering techniques and ray-tracing calculations. An Ani-

mated Quality Metric is proposed to guide the decision of which of the two rendering techniques must be used.

This paper addresses high quality rendering of dynamic radiosity environments. The main difference with previous work is the use of frame-to-frame coherence in the gathering step, which significantly reduces its computational cost. As mentioned above, frame-to-frame coherence has only been applied to the global pass. Our method provides means of obtaining high quality images that could not be obtained only with a global pass.

In the context of two-pass radiosity animation, temporal coherence should be used in the local pass to adaptively reconstruct the illumination function at the pixel level, trying to minimize the number of computations while keeping a high image quality. The pixels could be either replicated from the previous frame, interpolated between two key-frames or actually recomputed. In [23] a two-pass radiosity algorithm for static scenes has been proposed. This method allows an adaptive reconstruction step, which computes pixel intensities either through an actual gathering process or by simple interpolation. This strategy, which fits the requirements explained above, will be extended to cope with dynamic environments in the next sections.

3 Overview of the Method

As mentioned above, the goal of the proposed method is to efficiently reconstruct the radiosity function of a dynamic environment as a moving observer along a given period of time sees it. The changes in the scene due to object motion or camera changes affect not only the illumination across the surface but also the perception of the illumination. The illumination across a surface varies because the form factors change. Form factor modifications are caused by changes in the relative position of the two elements and by the motion of possible blockers in between. The variation of the illumination function across a surface is related to the global energy balance of the environment.

Moreover, geometric changes also influence the viewer's perception, which may vary even if the illumination across the surface remains constant. As the viewer comes closer

to a surface, the reconstruction function requires a higher level of refinement. On the other hand, simpler approximations become sufficient if the viewer moves away from the surface. At the extreme, when the surface exits from the viewing frustum, it no longer requires a final gathering.

The method herein proposed is based on an incremental two-pass strategy. We argue that the former type of changes (illumination variation) should be detected and processed mainly in the global pass, whereas modifications of the perception are mostly related to the second pass. However, unless very deep refinements of the solution are performed, the global pass fails at detecting small changes that may be perceptually noticeable. Thus, the local pass may need to recompute some energy transfer at higher image precision level.

As exposed above, the method is conceived as an extension of the two-pass method presented in the background section [23]. This method produces high quality radiosity images and it stores the results as textures, which are extended to the temporal dimension. Thus, the hierarchical structure computed to represent an animation sequence is made up of nodes that contain a space-time representation of the radiosity function. Each node is stored as a temporal sequence of textures of the same size that can be considered as a texture movie for a given patch in a given interval of time. The key of the method is to use frame-to-frame coherence to minimize the computations of these texture movies by only recomputing the changes produced by the dynamic behavior of the scene objects.

3.1 Two-Pass Method for Static Scenes

The method presented in [23] uses a hierarchical radiosity global pass (GP) in order to compute a global illumination solution that guides a second pass (SP) which builds the details of the illumination function. The global pass provides at the end a set of links for each polygon of the scene. This set of links represents the contribution of the scene to that polygon. The key of the method presented here is to identify in the gathering step the subset of these links that should be recomputed with higher accuracy (*bad*

links). The remaining links, whose contributions do not need to be recomputed, can be used directly (*good* links). Graphic hardware accelerators are used to recompute the *bad* links without introducing an unacceptable computing overhead. Different criteria can be proposed to classify links [19]. Specifically, a link is classified as *good* if the error estimation is within given bounds. This means that the illumination corresponding to that link is accurate enough. A conservative implementation of this definition is to classify as *bad* links those corresponding to direct illumination onto visible patches, and *good* links the remaining ones.

No modification has been introduced in the basic GP algorithm and the original brightness-weighted oracle is used [24]. As the second pass analyzes the interactions at the polygon level it is necessary to keep all the links corresponding to the same polygon-to-polygon interaction grouped. Therefore, in contrast to the classical HR method, in the global pass when a link is refined it is not removed and the whole link hierarchy is stored.

Once the global solution has been computed the illumination of the visible surfaces has to be reconstructed. The final goal of this step is to produce a texture for each visible patch that represents accurately its diffuse illumination and then to project all the visible patches using texture mapping to render the final image.

The method determines the visible patches, at which level of the hierarchy the textures are computed, and the size of these textures to guarantee that radiosity is computed at pixel level. Then, for each patch that has a texture associated, the links arriving at that patch are classified into two sets (*good* and *bad* ones).

The energy carried by *good* links is rendered into an auxiliary texture using linear interpolation (Gouraud shading). The result is accumulated into the polygon texture. Then *bad* links are traversed and the illumination for each one is computed in the auxiliary texture and then accumulated into the polygon texture. This process is performed as follows: first for each bad link of the polygon, the illumination is computed at each texel without performing visibility computations; next, the visibility is computed and stored in a texture of transmittances whose texels represent the fraction of the shooter energy that actually reaches the corresponding texel; finally, the auxiliary texture is

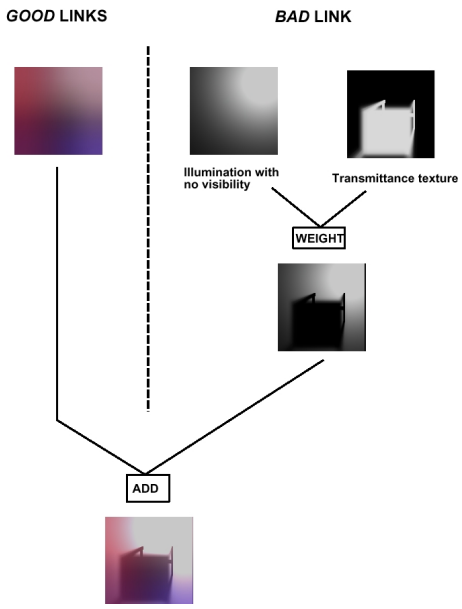

Figure 1: *Texture computation for a polygon with one bad link. Good links are rendered as usual while the bad link is rendered using the transmittance texture.*

multiplied by the transmittance texture giving as a result the illumination due to the shooter polygon. The result is accumulated into the polygon texture.

Transmittance Texture Computation

The main feature of the method is that the transmittance texture is computed using the graphics hardware accelerator, which makes the second pass very fast. The way of taking advantage of the hardware accelerator is to use the frame-buffer as the transmittance texture and to compute the visibility using projective shadows. This process is done in three steps:

1. Selection of a set of N random sample points on the shooter polygon corresponding to the *bad* link being recomputed. These sample points will be used to approximate the visibility in a similar way to the one used in hardware-driven soft shadow computation [25]. The number of sample points that are required to accurately approximate the visibility depends on several factors: the size of the shooter polygon, the distance between the shooter and the receiver, and the rela-

tive location of the link occluders. As the last factor is very difficult to evaluate, the method takes only into account the two first factors. Concretely, the number of points is made proportional to the maximum form factor between the shooter and the texels.

2. For each sample a projective transformation is computed taking the sample as the viewpoint and the polygon as the base of the viewing pyramid. The 4×4 homogeneous transformation matrix computation is described in [26].

3. All the scene polygons in the viewing pyramid are projected using the projective transformation of step 2. The transmittance texture texels are incremented if the sample and the texel are mutually invisible. After all the samples have been processed the transmittance texels have a value in the range $[0, N]$. A value of 0 means that all the sample points are visible from that texel, while a value of N means that none of the sample points is visible from that texel.

As mentioned above, this transmittance texture is used to weight the auxiliary texture giving as a result the illumination due to the shooter polygon. Illumination from *bad* links and *good* links are finally added resulting in the polygon illumination (Figure1).

The texture size is such that at least one texel covers one image pixel, ensuring that radiosity is computed at pixel level in the gathering step. In [23] it is shown that the memory requirements for the textures representing the radiosity of the visible patches is approximately one order of magnitude higher that the memory required for the image.

4 Space-Time Hierarchical Representation

The two-pass method described in the previous section produces a hierarchical representation of the radiosity as textures for the visible patches. The natural extension is to add the temporal dimension to the texture representation. This new structure is made of temporal texture spans that represent the radiosity function for a given patch

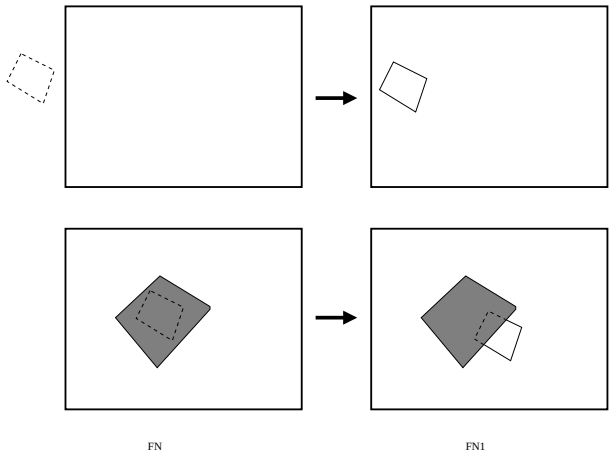

Figure 2: *Top row: a patch enters the view frustum and a new texture movie is initialized. Bottom row: a hidden patch becomes visible and a new texture movie is initialized.*

over an interval of time. These spans are called texture movies as they are made of a sequence of consecutive images (textures) of the same size.

This hierarchical representation of radiosity is computed from a coarse classical hierarchical radiosity solution that guides the two-pass temporal computations as in the static case. This global solution also has to be extended to the temporal dimension and it has to be efficiently recomputed. The problem of recomputing hierarchical solutions due to dynamic changes in the scene has been successfully addressed [7, 11] and it is not the goal of this work.

4.1 The Texture Movie Hierarchy

The construction of this structure is driven by the changes in the projection of the scene patches onto the camera frame-buffer along time. As textures are only created for visible patches, the creation, destruction and resizing of these textures are events that depend on how a patch is projected onto the camera frame-buffer. The fraction of the patch that is visible has to be taken into account, and the area of the camera frame-buffer covered by the visible part of that patch, for a given instant of time. The fraction of the patch that is visible determines if the patch is represented by one texture

at the top level or if it is represented at lower level by a hierarchy of textures. Also, the area covered in the camera frame-buffer by the patch determines the size of texture that guarantees the desired image quality. All these events are used as a criterion to perform the spatio-temporal subdivision of the patches into texture movies.

The criteria for the texture-movie interval computation are detailed:

1. Visibility of a patch. The changes in the visibility of a given patch between frames can produce the creation or termination of texture movies:
   - Create a new interval when a patch enters into the view frustum and it is not hidden by another patch, or when a hidden patch within the view frustum becomes visible (Figure 2).
   - Close a created interval when a visible patch leaves the view frustum, or when a visible patch starts to be hidden by another patch (Figure 3).
2. The patch is represented at a different level of the hierarchy. There are two events that can cause this change:
   - The visible portion of a patch changes and radiosity is represented at a higher or a lower level. This event is depicted in Figure 4. When a patch is leaving the view frustum the visible fraction of that patch becomes smaller, so if it is represented one level down the patch hierarchy (textures at one level

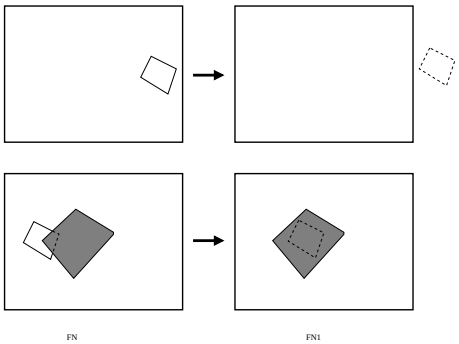

Figure 3: *Top row: a patch leaves the view frustum and the texture movie that has represented its radiosity is closed. Bottom row: a visible patch is hidden and the texture movie that has represented its radiosity is closed.*

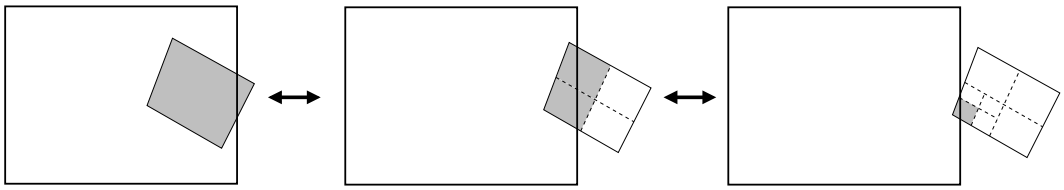

Figure 4: *The texture representing the radiosity of a polygon can be set at different levels depending on the visible part of that polygon.*

down are one fourth of the size of the upper texture) some of the children patches may be totally outside the view frustum and then storing of the texture is avoided. When a patch is entering into the view frustum and it is not represented at the top level, all the children become visible and it is more efficient to represent them as one texture at one level up the hierarchy.

- The projected area of a visible patch changes and the corresponding texture has to be resized. In this event, there are no visibility changes but a modification of the texture movie resolution to maintain homogeneous image quality.

Switching the level of the texture in the hierarchy could lead to popping artifacts. However, the texture size is computed to ensure that all the image pixels are covered by at least one texel at any frame, therefore the transition between levels is rather smooth. The artifacts are hardly noticeable in a static image, and even less in animation. Another source of artifacts could be abrupt changes in the link classifications. In order to avoid this problem the classification of a link does not change while the receiver patch is visible.

4.2 The Global Solution Space-Time Hierarchy

The global pass constructs a hierarchical representation of the illumination function over the 4D space. In contrast to the classical hierarchy [24], this structure keeps the history of links, i.e. links are not deleted when refined, and all the links of the hierarchy that sometimes carry radiosity during the considered time interval are stored. With

each link, a set of time intervals is stored which represent the lapses of time in which the link is active. A link is considered as active during a time interval if it is used for the energy transport during this interval.

As time passes, three possible events may happen: visibility changes, geometrical factor modifications or energy transport variations. The oracle function is sensible to these three types of events that provoke a refinement or coarsening of the links. The evolution of the hierarchy through time is thus a natural way of representing the key-instants of the links evolution.

In order to take better advantage of this information, in addition to the time intervals, information on the type of event that has produced the refinement or coarsening is also stored. Namely, a visibility flag indicates that the modification is caused by a visibility change.

Moreover, following the strategy proposed in [23], links are labeled in their different intervals as being *good* or *bad*. A *good* link at a given instant is a link that provides an approximation of the illumination which does not require a final gathering step. In the global data structure, for the *good* links, the form factors associated to the activity intervals are also stored. These form-factors are considered to vary linearly along the activity interval.

A similar approach is used for storing a time-space representation of the radiosities. The patch hierarchy is augmented with a list of intervals that store a linearly varying representation of the radiosity of that patch.

The time intervals associated to links and patches will be used in the second pass to recompute the radiosity of visible patches. Thus, it is not necessary to store time information for all links and patches, but only for patches that contribute to the radiosity of visible patches and for links that transport radiosity to visible patches. This restriction highly reduces the memory consumption due to temporal data associated to the global solution.

For the global pass data structure construction, temporal coherence can be exploited by using the approach presented in [7, 11]. This work introduces the line-space hierarchy method that allows to adaptively control the interactions that have changed,

recompute them, and redistribute the energy changes to converge to a new solution. This approach has not been fully implemented because the use of a coarse global solution makes not critical the energy transport recomputations. Temporal coherence is used to efficiently identify the links that change from one frame to the next using a clustered representation of the scene and shaft culling. Thus, only a small set of links is recomputed. Then, energy transport is performed from scratch. The temporal data is updated comparing the previous solution with the new solution.

4.3 The Incremental Approach

The main problem faced for the temporal data structures presented in the previous section is memory storage. If the two data structures (global pass and texture hierarchy) have to be constructed to represent a whole animation sequence, then the storage needs become unaffordable. As presented in Section 3.1, the memory required to store an image as a texture hierarchy is one order of magnitude the size of the image. For an image size of 352x240 pixels, the texture memory required is about 2.5 Megabytes. Storing one second of animation (24 frames) would need approximately 60 Megabytes. A short sequence of 10 seconds would consume 600 Megabytes, which is unacceptable even for high-end machines.

These requirements force the use of an incremental approach that computes only a minimum part of these hierarchies. The method proposed herein is an incremental algorithm that updates the data structures as frames are produced. The pseudo-code corresponding to the main loop is presented in Table 1. In this algorithm, two processes are performed simultaneously:

1. The first one is performed in the `computeNewInterval` function and it is driven by the `timeIntervals` variable. This process determines the extent of the texture movie intervals, but it does not compute them. It starts at the time instant that it receives as an input parameter and it computes the texture movie intervals by evaluating the changes in visibility described in Section 4.1. These changes are evaluated by projecting the scene patches onto an item-buffer that allows to

compute the projected area of each patch. Then, the item-buffer is traversed and the visibility of the patches is updated.

To update the interval structure, the top patches are traversed and the extent of corresponding texture movies are updated with the following criterion:

- If the patch is not visible, close all existing intervals corresponding to this patch and its descendants.
- If the patch is visible then its projection onto the item buffer is analyzed:
  - If the patch is suitable to be represented by a texture at its level, then the optimum texture resolution is considered:

```
generateSequence(iniTime, endTime, numFrames)
{
  frameComputed = 0;
  timeIntervals = iniTime;
  timeGenerated = iniTime;
  stepFrame = (endTime - iniTime) / numFrames;

  thereAreFramesToCompute = true;
  while (thereAreFramesToCompute)
    {
      timeIntervals = computeNewInterval(timeIntervals);

      canComputeCurrentFrame = true;
      while (canComputeCurrentFrame)
        {
          if (!generateAndRenderFrame(timeGenerated))
            canComputeCurrentFrame = false;
          else
            {
              frameComputed++;
              timeGenerated += stepFrame;
            }
        }
      if (frameComputed >= numFrames)
        thereAreFramesToCompute = false;
    }
}
```

Table 1: *Pseudo-code for main loop*

* If the patch was visible in the previous frame, then the resolution computed is compared with the resolution computed for the previous frame. If they differ, the previous interval is closed and a new one is created with the new resolution.
* If the patch was not previously visible, a new interval is initialized with the computed resolution.

– If the patch has to be considered at a lower level of the patch hierarchy, then the interval corresponding to the actual patch (if any) is closed and the whole criterion is applied to the children. In order to apply the criteria, the children must be projected onto the item buffer to determine their projected area and to compute their visibility and texture size.

It should be noted that this process does not effectively compute the texture movies, but the extent of the intervals. This is, it determines the space-time subdivision of the texture hierarchy for a given interval of time. The texture movies are effectively computed in the second process, as they are needed to produce successive frames. In Section5 the use of frame-to-frame coherence to compute the texture movies is explained in more detail.

This process finishes when all the patches traversed have at least one interval that has been closed. As explained before, an interval is closed when the patch becomes invisible, it is represented at other patch level, or it reaches a maximum predefined length.

2. The second process tries to generate as many frames as possible with the intervals determined in the first process. Four actions are performed each time a frame is requested in call `generateAndRenderFrame(timeGenerated)`:

   - First, all the texture movies whose maximum time is lower than `timeGenerated` are deleted, because they will not be used anymore.
   - Second, the patch hierarchy is traversed to find which intervals contain `timeGenerated`, because they will be needed for the production of the frame. Some of these intervals have not been yet computed because it is the first

time that they are requested. To compute these intervals, the global solution must be able to answer all the queries needed to perform the computations: radiosity of the visible patches, form factors of the links, ... These queries must be answered for the interval of time corresponding to the union of time intervals of the texture movies that have to be computed.

- Update the global pass for the interval of time computed in the precomputation step.
- Generate the textures corresponding to `timeGenerated`.

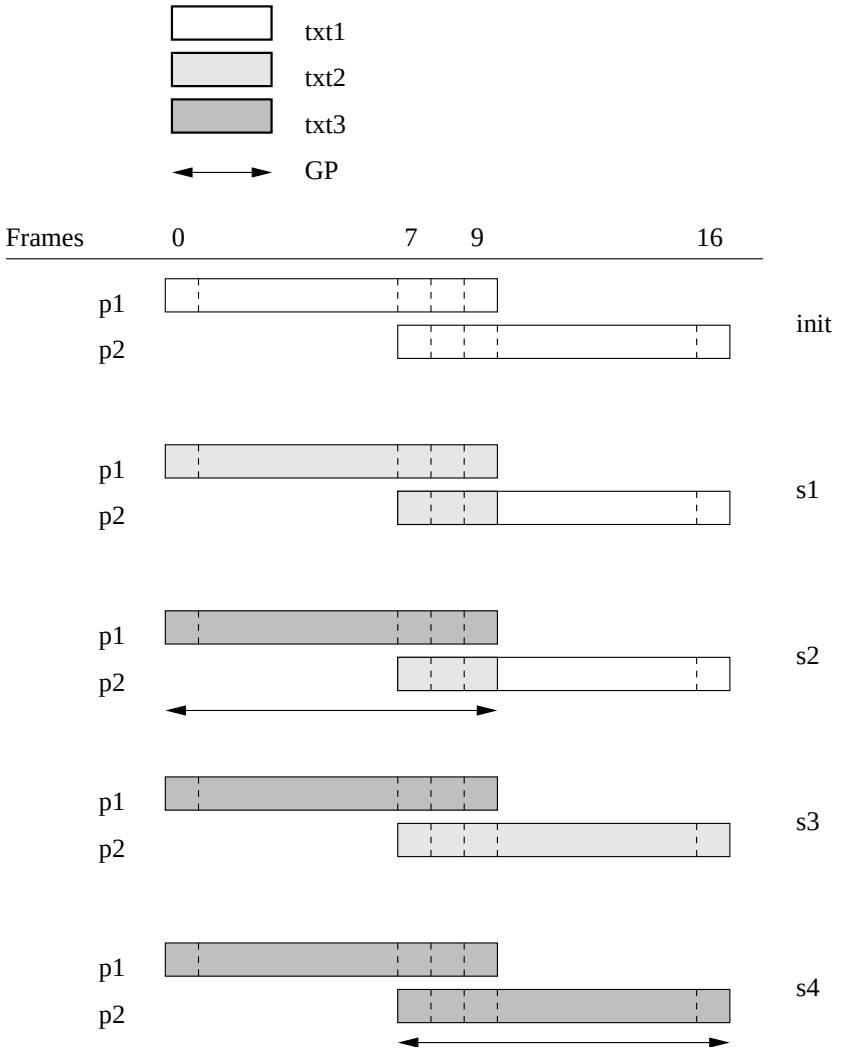

Figure 5: *Texture movie computation on two patches over an interval of time.*

Figure 5 shows an example of how the global solution and the texture movie hierarchy is incrementally updated following the criteria exposed above. The figure represents a 17 frames interval with two patches P_1 and P_2 that are visible in the sub-intervals $[0, 9]$ and $[7, 16]$ respectively. There are 5 main steps in the computation of the sequence:

1. Frame 0 is requested and the first action is to compute the interval extent for the patches until the extent of all visible patches in frame 0 has been fully computed. Only P_1 is visible at frame 0 so the extent computation is performed in the interval $[0, 9]$.

2. The next step computes all the texture movies corresponding to patches visibles at frame 0. Again, only patch P_1 is visible so its texture movie is computed. The global pass is updated to contain information for interval $[0, 9]$.

   Frames $[0, 6]$ are computed and rendered using texture movie of patch P_1.

3. Frame 7 needs patch P_2 interval to be completed. The first action is to complete interval extent computation.

4. The next step computes all the texture movies corresponding to patches visibles at frame 7. Patch P_1 has been already computed so texture movie for patch P_2 is computed. The global pass is updated to contain information for interval $[7, 16]$.

   Frames $[7, 16]$ are computed and rendered using texture movie of patches P_1 and P_2.

5 Frame-To-Frame Coherence For Texture Movie Computation

As mentioned above, the local pass constructs a representation of the animation in terms of a set of texture-movies that are associated to patches at different levels of the hierarchy. The level of the hierarchy for a texture is determined by following the criterion proposed in [23]. The length of the movie is determined by changes in

visibility (patches entering or leaving the image and being occluded) and projected size (zooming in and out). Each time a frame n is requested, new texture movies might be computed to ensure that all the visible surfaces have a texture representation at some level for time T_n.

Texture movies are computed by examining the links transporting energy to the corresponding patch during the associated time interval. Links are traversed up and down the hierarchy, and grouped into *good* and *bad* sets. For simplicity links are classified at top level (shooter is a top-level patch), and thus all the links down the hierarchy have the same classification. Each *bad* link is recomputed performing a gathering step, as shown in [23], thus generating a layer of the texture movie. These layers are accumulated, and finally the layer corresponding to all the *good* links is computed using the global solution data and accumulated in the texture movie. This mechanism is the same as the one used in the static method described in Section 3.1.

The key feature of this second pass is that layers corresponding to static *bad* links (receiver and shooter are static) can be efficiently computed by separating dynamic changes from static ones. When computing the gathering step for a *bad* link, the most

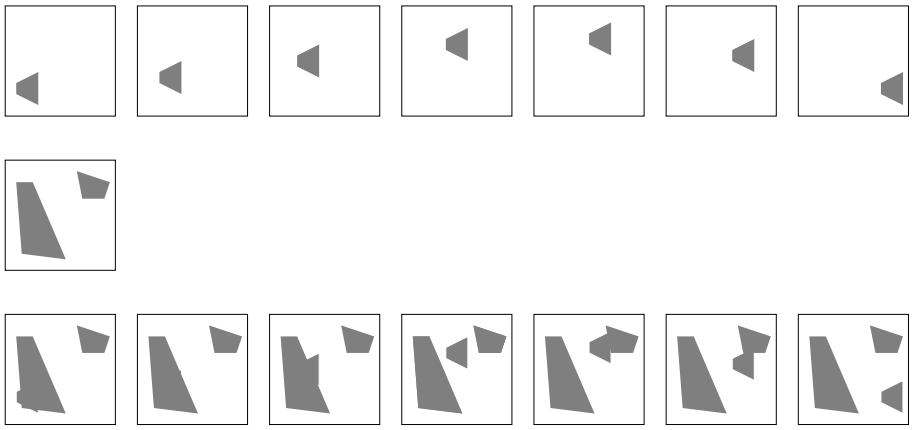

Figure 6: *For computing transmittance textures in a texture movie, dynamic blockers are recomputed once per frame (top row). Static occluders are computed once per movie (middle row). Finally, transmittance textures are merged to yield the transmittance texture movie for the corresponding link and time interval.*

expensive operation is the transmittance texture computation (visibility). Using a clustered representation of the scene and shaft culling, blockers are separated into static ones and dynamic ones in a fast previous step. For a N key-textures sequence, only one transmittance texture is computed for the whole layer corresponding to the static blockers. Then, N transmittance textures are computed for the dynamic blockers. Finally, transmittance textures are merged and used to compute the layer weighting the unoccluded illumination (Figure 6).

Merging transmittance textures cannot be done by simple averaging because of the kind of information that they represent. Figure 7 shows the meaning of transmittance values and which values can yield the merging. Exact merging is not possible when taking into account only the transmittance textures values, because there is geometrical information lost in the process. In our method, we use a simple approach that allows a fast merging and gives reasonable quality shadows. Given the transmittance values T_s and T_d corresponding to the same texel, the following operator is applied (Figure 7):

$$T = (\max(T_s, T_d) + \min(1, T_s + T_d))/2.$$

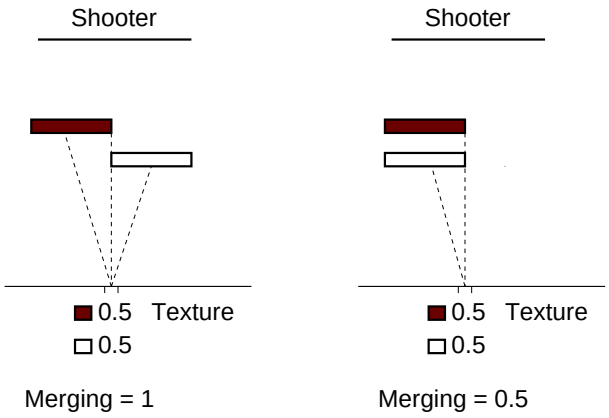

Figure 7: *Transmittance merging from different set of blockers. The same values to merge can yield very different results depending on the blockers location. In the left case, two values of* 0.5 *yield a total* 1 *(full occlusion) because the two occluders do not overlap. In the right case, the opposite configuration is shown.*

where

$$T_i \in [0, 1]$$

One of the main problems of the local pass is memory consumption. Texture movies are expensive to store in memory because of their size. The maximum size texture movie processed in our tests is a sequence of 20 textures of 1 Megabyte. With this size, several sequences of this size computed at the same time can require too much memory for conventional computers. To overcome this problem, a disk cache for the textures has been implemented. The user can control maximum texture memory. Local disk is used to temporally store the textures that do not fit in memory. As shown in the result section, the overhead for this approach is very small, while allowing the use of an arbitrary amount of textures. An alternate solution would be texture compression. However, at the moment, we have not been able to find any compression scheme faster than disk caching and without artifacts.

6 Results

The method proposed in this section has been implemented using the SIR framework [27], and the timings have been measured in an Intel Pentium III 500 MHz processor. The animations have been computed at 352x250 pixels, and the dynamic properties of the scene have been modeled with A|W Maya and exported to MGFE file format that makes them readable by the SIR framework.

The results show the speed-ups achieved by using temporal coherence with regard to the static method presented in Section 3.1. Both methods use hierarchical radiosity without any clustering in the global pass solution, and thus, they suffer from the initial quadratic number of links problem. However, the fact of using hierarchical radiosity without clustering adversely affects more the dynamic approach than the static one. Thus, if the global pass were a clustering method, the benefits of the dynamic approach would be greater. A more detailed explanation of this fact is presented in Section 6.1, along with some experimental estimations of the benefits of using clustering.

There are three groups of tests performed to evaluate the method. In each of these tests, there is a set of parameters that are modified to measure the degree of coherence of each configuration:

- The first group of tests use a modified standard test scene (Pete Shirley's table and four chairs scene) and a single dynamic object. The parameters modified are the camera movement and the number of dynamic lights. This test is aimed at showing how the camera movement and dynamic lights affect the degree of coherence that is exploited by the method.
- The second group of tests use a similar scene. The parameter modified is the number of dynamic objects. This test shows how the increasing number of dynamic objects in a scene affects the coherence.
- The third test is an animation in a scene completely different from the previous ones. It consists on a very partitioned environment with multiple lights. There is a single moving object that is followed by the camera. This test shows the performance of the method for a non-homogeneous lighting distribution.

The first test uses a modified standard test scene (Pete Shirley's table and four chairs scene). The animations show a dynamic object (a very simplified space car) traversing the scene under different conditions. As the objects moves, the shadows that it projects onto the other surfaces change and vice-versa, yielding to recomputation of some links. In addition, as the object approaches or recedes from the camera, its texture diminishes or increases. The second pass may thus require a recomputation of its associated texture.

Along with the movement of the object, two other parameters change in the animations: the camera (static camera, center of pan animated, center of pan and eye animated) and the lights. Specifically, the camera can be static, or the center of pan can move while the viewing position remains constant, or both the center of pan and the viewing point move. The light sources can be either static or dynamic.

These two parameters influence the performance of the method because they affect the degree of coherence of the images. The camera movement causes all the visible

textures to potentially change their projection onto the image. Indeed, the visible polygons can enter into the viewing frustum or disappear from it, or they may be zoomed in or out, therefore reducing the time interval in which coherence can be exploited. Moving light sources means that their associated *bad* links cannot be efficiently computed because all the blockers must be considered as dynamic, and thus transmittances are computed from scratch for each frame.

Six animations have been computed by combining the object's movement and the two mentioned parameters. Table 2 shows timing and memory usage of the animations. All the animations are 400 frames long (16 seconds approximately). The scene has 398 static polygons and 16 dynamic polygons. The six sequences computed correspond to the same animation (one dynamic object crossing the scene) but changing the camera movement and the presence of dynamic light sources. All the animations have been computed with the texture disk cache enabled.

For each animation, the total time and the average time spent per frame are shown. The time per frame is really low, around 11 seconds. In order to better evaluate the speed-up of the proposed method, it has also been compared to the strategy proposed in [23], of which it is a natural extension to 4D. Thus, the frames have also been computed statically, each of them from scratch, using the static method. The speed-up has been measured as an average ratio between the time spent per frame in the static method and the dynamic one. It should be noted that the static method already supposes a great speed-up compared to classical hierarchical radiosity (one order of magnitude). Therefore, animations of the same quality computed with hierarchical radiosity would be extremely expensive (from one to two orders of magnitude). Finally, the maximum memory used during the computations is shown. This memory includes all the data structures involving the method: textures, patches, links, etc.

The speed-up variation in the animations corresponds to the different levels of coherence related to the changes in the sequence conditions. The presence of dynamic light sources produces smaller ratios because coherence cannot be used when computing texture layers associated to dynamic *bad* links. Camera movement also decreases the efficiency, because visible surfaces change from one frame to another, and then less

		Total	Per Frame	Speed-Up	Memory
Four Static lights	Static camera	7011 sec	17.52 sec	5.7	82 M
	Dynamic camera (center)	8693 sec	21.7 sec	3.31	138 M
	Dynamic camera (center and eye)	7392 sec	18.47 sec	3.03	130 M
Two static lights and two dynamic lights	Static camera	6023 sec	15.05 sec	3.44	82 M
	Dynamic camera (center)	6573 sec	16.12 sec	2.12	137 M
	Dynamic camera (center and eye)	6573 sec	16.43 sec	2.17	130 M

Table 2: *Timings for the six animation sequences computed in the first series of tests.*

coherence can be used. It should be noted that absolute times are greater for static camera animations. This is due to the fact that the point of view of the static camera animations comprises the entire scene and then, textures must be generated for all the visible polygons. However, the camera with center of pan and eye movement exhibit the lower absolute times, because significant intervals of the animation show very few visible polygons, and fewer textures have to be computed.

Figure 8 shows, for each animation, one frame corresponding to the same time instant. The left columns correspond to the scene with four static light sources, and the right columns show frames for the scene with two static light sources and two dynamic ones. The top row corresponds to animations computed with a fixed camera. The middle row corresponds to animations computed with a camera having a dynamic center of pan (pointing to the dynamic object). Finally, the bottom row corresponds to the full-animated camera following the same path as the dynamic object. Full MPEG-1 animations can be found in www: `http://ima.udg.es/~imartin/coherence/index.html`.

The second series of tests uses a modified standard test scene (Pete Shirley's three chairs scene). The animation is a long path that traverses the scene affecting almost all the interactions in the environment. This path is traversed by an increasing number of dynamic objects (cubes). The coherence decreases as the number of dynamic object grows because of the increasing number of links that must be recomputed in the global

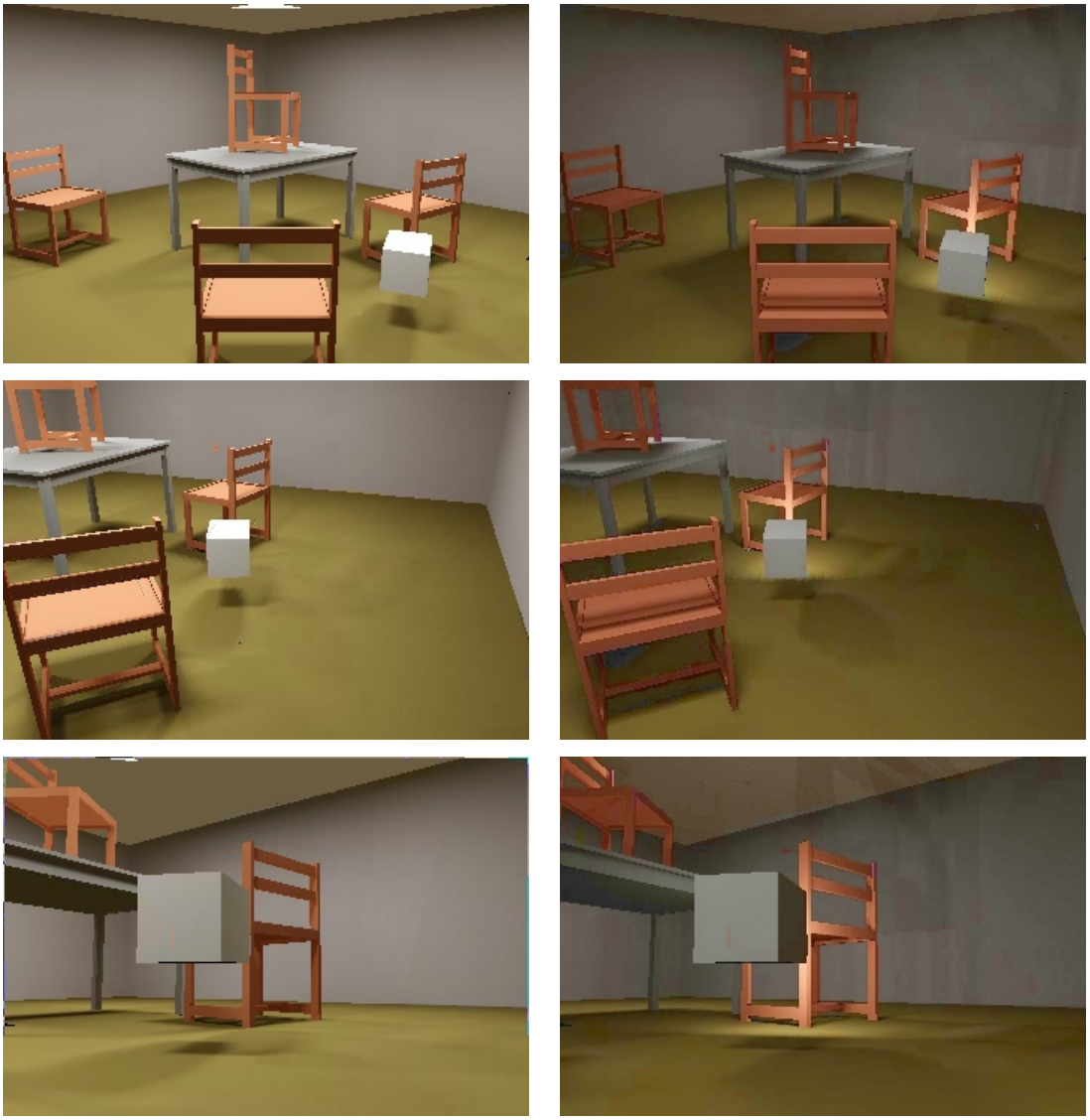

Figure 8: *Frames corresponding to the same time in six different animations. The left column corresponds to animations with four static light sources, and the right column corresponds to animations with two static light sources and two dynamic ones. The top row corresponds to a static camera, the middle row corresponds to a camera with an animated center of pan, and the bottom row corresponds to animations with a full-animated camera.*

pass, and because of the increasing number of objects that cast dynamic shadows that must be recomputed each frame. The camera remains fixed in all the animations to better appreciate the influence of the dynamic objects.

Six animations have been computed with this scene. Each one has 1, 2, 4, 8, 16, and 32 dynamic objects respectively.

All the animations are 400 frames long (16 seconds approximately). The scene has 262 static polygons, and each dynamic object has 6 polygons. The six sequences computed correspond to the increasing number of dynamic objects. All the animations have been computed with the texture disk cache enabled.

For each animation, the total time and the average time spent per frame are shown. The speed-up of the method is measured as in the previous test.

Number of dynamic objects	Total	Per Frame	Speed-Up	Memory
1	1646	4.11	5.17	28 M
2	2200	5.49	4.78	33 M
4	3232	8.08	4.32	39 M
8	4905	12.25	3.41	51 M
16	9348	23.37	2.52	68 M
32	21437	53.59	1.68	85 M

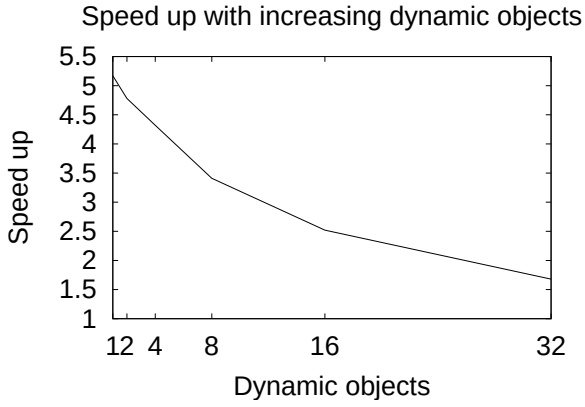

Table 3: *Results for the second series of tests. The table shows the performance of the method on the same scene with increasing number of dynamic objects. The columns show: time for computing the whole animation (Total), time per frame (Per Frame), speed-up of the method compared with the static computation of the frames (Speed-Up), and the maximum memory used (Memory).*

The speed-up variation in the animations is shown in Table 3, and it corresponds to the different levels of coherence related to the changes in the sequence conditions. The more dynamic objects in the scene, the more computation time needed and the less coherence can be exploited. However, the speed-up decreases slowly and the method is still capable to exploit coherence with a high number of dynamic objects. This is mainly due to the fact that most of polygons that cast shadows are static (chairs).

Figure 9 shows one image of each animation. Dynamic objects are uniformly distributed over the scene so the number of links affected by these objects is nearly proportional to the number of dynamic objects in the scenes tested. Table 4 shows the average number of links that are affected from frame to frame. As expected, this average grows with the number of dynamic objects and, if these objects are distributed uniformly, it converges approximately to the total number of links.

Number of dynamic objects	Total	Max	Min	Average (%)
1	97345	5440	24253	9682 (9.9%)
2	108455	12109	72485	31939 (29.4%)
4	114187	19534	88289	76141 (66.68%)
8	129262	93909	110953	101557 (78.5%)
16	159334	122778	140605	130410 (81.8%)
32	260032	212802	227305	213981 (82.3%)

Table 4: *Results for the second group of tests. The table shows how links are affected by dynamic objects. The second column (Total) shows the maximum number of links used during the sequence computation, the third one (Min) shows the minimum number of links updated from frame to frame, the fourth one (Max) shows the maximum number of links updated from frame to frame, and the last one (Average) shows the average number of links updated from frame to frame (total and per cent).*

The third test is an animation on a highly partitioned scene. It has a single dynamic object that is followed by the camera along a path that traverses most of the scene. The animations show how the method performs in a sequence where lighting conditions

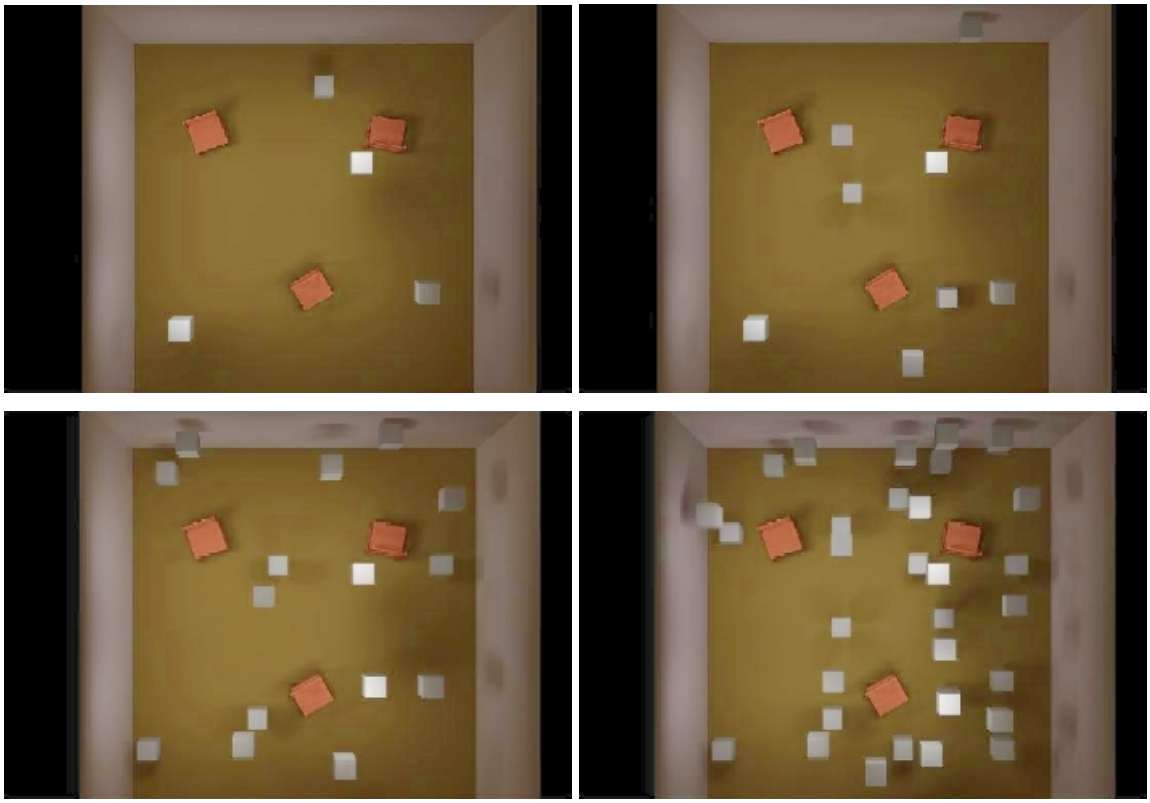

Figure 9: *Images from four animations of the second test. The scenes show how the increasing number of object fill the room homogeneously.*

and visible surfaces vary rapidly. Table 5 shows the timing of this animation and the speed-up of the method with respect to statically computed frames.

The results show how coherence decreases because of the fast changes in the visible surfaces, and because most of the shadows correspond to the dynamic object and then they must be recomputed from frame to frame. Figure 10 shows four frames of the animations that illustrate the different lighting conditions that the method deals with.

6.1 Performance considerations

Although speed-ups showed by the presented results are valuable, the actual implementation suffers from an unnecessary (theoretically) overhead. Since the global pass is based in classical hierarchical radiosity, the method has to cope with a quadratic number of links. As an example, the scene used in the first series of tests (∼500 poly-

	Total	Per Frame	Speed-Up	Memory
Cubic Star Wars	4462	11.15	2.33	51 M

Table 5: *Results for the third test. The table shows: time for computing the whole animation (Total), time per frame (Per frame), speed-up of the method compared with the static computation of the frames (Speed-up), and the maximum memory used (Memory).*

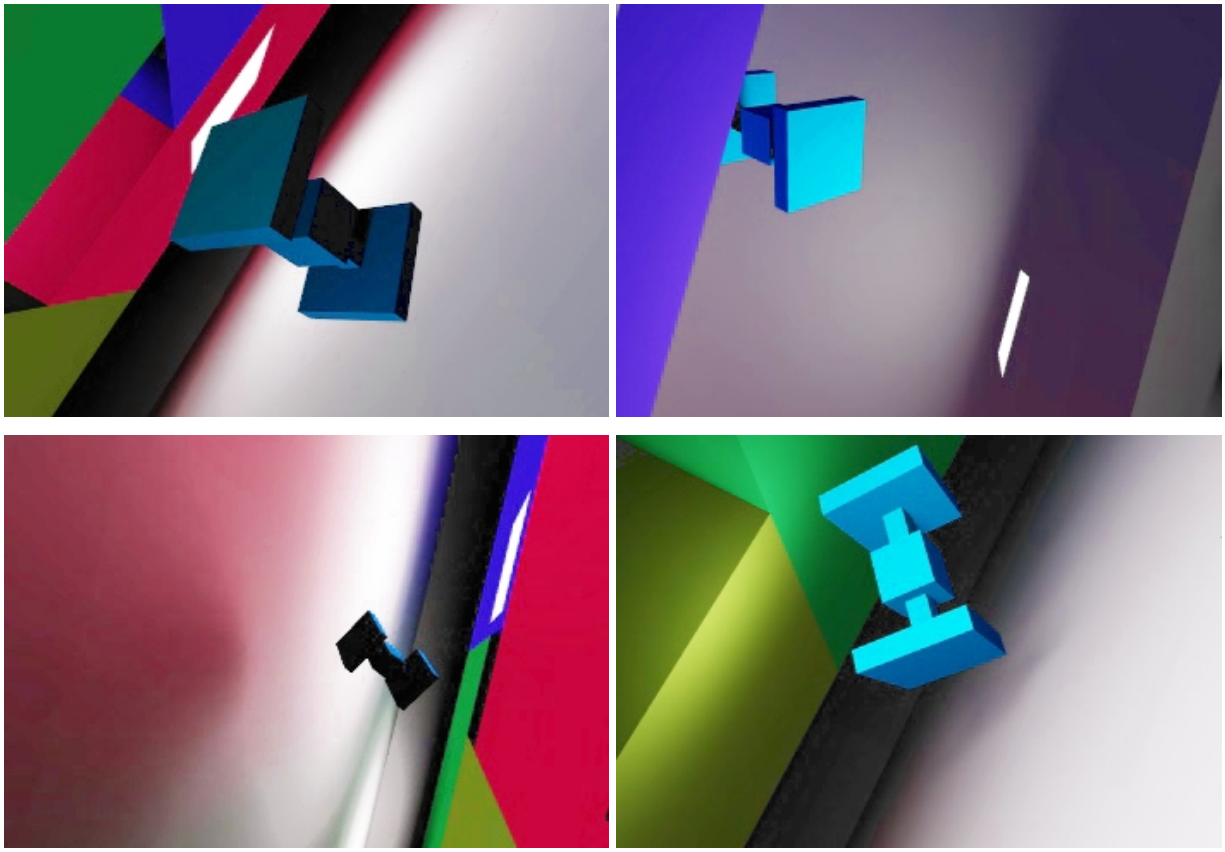

Figure 10: *Four frames of the animation corresponding to the third test.*

gons) produces a global solution that uses about 200000 links. In [28] a test scene of 23000 initial polygons is used for clustering computations, yielding 250000 links approximately. This comparison shows that the results provided in this paper could be enhanced with the use of clustering radiosity, since the overhead in the number of links introduces several significant performance penalties:

- The most obvious drawback is the extra visibility computations associated to the

links. The quadratic initial number of links means that a huge number of rays must be traced to compute an initial solution even for simple scenes.

- Second, the use of classical hierarchical radiosity produces a high spatial density of links. For example, in the chairs scene, each patch of one chair has a bunch of links corresponding to other chair. In clustering radiosity, this bunch of links could be probably substituted by one link between that patch and a cluster representing the other chair. This bunch of links is concentrated in a thin space region, so, if a dynamic link traverses this region, all the links are likely to be affected and then they will have to be recomputed.
- The last and more subtle penalty is related to texture movie computations. Each time such a movie is computed, the global pass must be set to each of the key times corresponding to each sample of the movie. This means an entire traversal of the link structure to set the appropriate form factors. The overhead in the number of links makes this traversal too expensive compared to the size of the scene used.

In order to quantify this overhead a test has been performed to measure the computational overhead of using classical hierarchical radiosity. The assumption made to allow such a test is that the number of links used for animations presented in the first set of tests (a single moving object) would be one order of magnitude smaller with clustering. This means that all global pass related operations can be reduced to a 10%. Table 6 shows the corrected results.

	Previous Speed-Up	Corrected Speed-Up
Dynamic camera and dynamic lights	2.17	3.9
Static camera and static lights	5.7	6.86

Table 6: *Corrected speed-up for two sequences of the first set of tests.*

The profiling of the executions has shown that about 25% of the total time of the sequence computations is due to the update of the links from frame to frame, and to

the traversal of the link hierarchy to set the form factors. This percentage is valid for this concrete test, and would eventually increase with larger scenes due to the quadratic nature of the number of links. The use of clustering would greatly improve this overhead. Specifically, it would provide a method to produce the minimum global solution needed to drive the second pass.

7 Conclusions and Future Work

In this paper we have presented a method to exploit temporal coherence in the computation of animations with hierarchical radiosity. The concept of hierarchy is extended to the temporal dimension. An animation sequence is represented as a hierarchy of spatio-temporal nodes which represent the radiosity over a given surfaces in a given time interval. The main contributions are:

- A new representation of space-time radiosity is introduced. The illumination of a given scene over a given time interval is represented as a hierarchy of spatio-temporal nodes. Each node stores a texture movie that represents the illumination of the corresponding patch over a continuous interval of time.
- A refinement criterion for this hierarchical structure is introduced that takes into account the projection of the visible surfaces onto the camera frame-buffer and thus, it guarantees an homogeneous visual quality.
- Finally, an incremental construction of this data structure is presented that makes this method usable. The high storage requirements of the spatio-temporal representations make an incremental approach unavoidable. The data structure that stores the sequence is maintained as small as possible for producing the current frame.

The use of classical hierarchical radiosity imposes a limit on the size of the scenes because of the quadratic cost of the global pass. Morevover, the fact that only a coarse solution is needed in the global pass makes this drawback more obvious. Using clustering radiosity [29] would dramatically reduce the cost of the global pass for larger

scenes, and the method should only be slightly modified to cope with links with clusters acting as shooters.

In addition, taking into account that the global pass is not directly used for rendering, a specific oracle could be designed to better drive the second pass, and refine more finely the links that will no further be recomputed in the second pass.

Finally, the oracles of the second pass could be improved by introducing animation criteria following the work done in [22].

References

[1] S. E. Chen, "Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System," in *Computer Graphics (ACM SIGGRAPH '90 Proceedings)*, vol. 24, pp. 135–144, August 1990.

[2] D. W. George, F. X. Sillion, and D. P. Greenberg, "Radiosity Redistribution for Dynamic Environments," *IEEE Computer Graphics and Applications*, vol. 10, pp. 26–34, July 1990.

[3] S. Muller and F. Schoeffel, "Fast Radiosity Repropagation for Interactive Virtual Environments Using a Shadow-Form-Factor-List," in *Fifth Eurographics Workshop on Rendering*, (Darmstadt, Germany), pp. 325–342, June 1994.

[4] X. Pueyo, D. Tost, I. Martin, and B. Garcia, "Radiosity for dynamic environments," *The Journal of Visualization and Computer Animation*, vol. 8, no. 4, pp. 221–231, 1997.

[5] D. A. Forsyth, C. Yang, and K. Teo, "Efficient Radiosity in Dynamic Environments," in *Fifth Eurographics Workshop on Rendering*, (Darmstadt, Germany), pp. 313–323, June 1994.

[6] E. Shaw, "Hierarchical radiosity for dynamic environments," *Computer Graphics Forum*, vol. 16, pp. 107–118, June 1997.

[7] G. Drettakis and F. X. Sillion, "Interactive update of global illumination using a line-space hierarchy," in *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, vol. 31, pp. 57–64, 1997.

[8] D. R. Baum, J. R. Wallace, M. F. Cohen, and D. P. Greenberg, "The Back-Buffer Algorithm: An Extension of the Radiosity Method to Dynamic Environments," *The Visual Computer*, vol. 2, pp. 298–306, September 1986.

[9] R. Orti, S. Riviere, F. Durand, and C. Puech, "Radiosity for Dynamic Scenes in Flatland with the Visibility Complex," in *Computer Graphics Forum*, vol. 15, pp. C237–C248, September 1996.

[10] F. Durand, G. Drettakis, and C. Puech, "The Visibility Skeleton : A powerful and efficient multi-purpose Global Visibility Tool," in *Computer Graphics Proceedings, Annual Conference Series, 1997 (ACM SIGGRAPH '97 Proceedings)*, pp. 89–100, 1997.

[11] F. Schoffel and A. Pomi, "Reducing Memory Requirements for Interactive Radiosity Using Movement Prediction," in *Tenth Eurographics Workshop on Rendering*, pp. 233–242, 1999.

[12] I. Martin, X. Pueyo, and D. Tosti, "A framework for animation in global illumination," Tech. Rep. IIiA 98-23-RR, Institut d'Informatica i Aplicacions, Universitat de Girona, Girona, Spain, October 1998.

[13] C. Damez and F. Sillion, "Space-Time Hierarchical Radiosity," in *Tenth Eurographics Workshop on Rendering*, (Granada, Spain), pp. 243–254, Jun 1999.

[14] G. Besuievsky and M. Sbert, "The Multi-Frame Lighting Method: A Monte Carlo Based Solution for Radiosity in Dynamic Environments," in *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, (New York, NY), pp. 185–194, Springer-Verlag/Wien, 1996.

[15] J. Nimeroff, J. Dorsey, and H. Rushmeier, "Implementation and analysis of an image-based global illumination framework for animated environments," *IEEE*

Transactions on Visualization and Computer Graphics, vol. 2, pp. 283–298, December 1996.

[16] M. C. Reichert, "A Two-Pass Radiosity Method Driven by Lights and Viewer Position," m.Sc. thesis, Ithaca, NY, January 1992.

[17] H. E. Rushmeier, "Extending the Radiosity Method to Transmitting and Specularly Reflecting Surfaces," m.Sc. thesis, Ithaca, NY, 1986.

[18] F. Sillion and C. Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection," in *Computer Graphics (ACM SIGGRAPH '89 Proceedings)*, vol. 23, pp. 335–344, July 1989.

[19] D. Lischinski, F. Tampieri, and D. P. Greenberg, "Combining Hierarchical Radiosity and Discontinuity Meshing," in *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pp. 199–208, 1993.

[20] W. Sturzlinger, "Optimized local pass using importance sampling," in *WSCG 96 (Fourth International Conference in Central Europe on Computer Graphics and Visualization)*, vol. 2, (Plzen, Czech Republic), pp. 342–348, University of West Bohemia, 1996.

[21] C. Ureña and J. C. Torres, "Improved irradiance computation by importance sampling," in *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)* (J. Dorsey and P. Slusallek, eds.), (New York, NY), pp. 275–284, Springer Wien, 1997. ISBN 3-211-83001-4.

[22] K. Myszkowski, P. Rokita, and T. Tawara, "Perceptual-Informed Accelerated Rendering of High Quality Walkthrough Sequences," in *Tenth Eurographics Workshop on Rendering*, pp. 13–26, 1999.

[23] I. Martin, X. Pueyo, and D. Tost, "A two-pass hardware-based method for hierarchical radiosity," *Computer Graphics Journal (Proc. Eurographics '98)*, vol. 17, pp. C159–C164, September 1998.

[24] P. Hanrahan, D. Salzman, and L. Aupperle, "A Rapid Hierarchical Radiosity Algorithm," in *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, vol. 25, pp. 197–206, July 1991.

[25] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, "Fast Shadows and Lighting Effects using Texture Mapping," in *Computer Graphics Proceedings, Annual Conference Series, 1992 (ACM SIGGRAPH '92 Proceedings)*, pp. 249–252, 1992.

[26] M. Herf and P. S. Heckbert, "Fast Soft Shadows," in *ACM SIGGRAPH '92 Visual Proceedings, Technical Sketch*, p. 145, 1996.

[27] I. Martin, F. Perez, and X. Pueyo, "The SIR rendering architecture," *Computers & Graphics*, vol. 22, no. 5, pp. 601–609, 1998.

[28] M. Stamminger, P. Slusallek, and H.-P. Seidel, "Bounded radiosity - illumination on general surfaces and clusters," *Computer Graphics Forum (Eurographics '97 Proceedings)*, vol. 16, no. 3, pp. C309–C317, 1997. Available from http://www9.informatik.uni-erlangen.de/eng/research/pub1997.

[29] B. Smits, J. Arvo, and D. Greenberg, "A Clustering Algorithm for Radiosity in Complex Environments," in *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pp. 435–442, 1994.