Animating architectural scenes utilizing parallel processing
F. Van Reeth, E. Flerackers, W. Lamotte

Applied Computer Science Laboratory, Limburg University Center, Universitaire Campus, 3590 Diepenbeek, Belgium

ABSTRACT

The ever increasing demand for realism and complexity in synthesizing architectural animated sequences makes the development performant rendering algorithms and -platforms a necessity. In order to answer this necessity, we take recourse to parallel processing techniques. More precisely, networks of transputers are utilized to implement parallel rendering algorithms. Hence, we can build animation systems which can linearly be expanded in performance, simply by adding new processors to them. Especially for traditionally expensive rendering techniques, a linear performance improvement is measured in systems with up to 50 processors.

INTRODUCTION

Looking at the history of computerized architectural visualizations, one can distinguish several eras. In the beginning, two-dimensional plotter drawings were all one could ask for. Later on, three-dimensional vectorized wire-frame drawings came to be, followed by visualizations in which more sophisticated colored renderings were to be used. Currently, state-of-the-art architectural visualization is requiring the availability of three-dimensional computer
animation with an ever increasing quest towards photo-realism. The fact that a lot of the naturalistically looking surface materials which are needed in architectural scenes take up a considerable amount of computing power in order to be generated, makes animation system designers search for more performant rendering solutions. This is especially true when making animated sequences, as the display standards require the production of at least 25 images for every second of animation.

The research group in our laboratory took the decision of going in the parallel processing direction in order to increase the rendering performance. By utilizing parallel processing techniques, it becomes possible to achieve high performance ratings on the one hand, while still staying within software solutions on the other hand (i.e., without requiring special purpose hardware). As many timeconsuming rendering techniques are too complicated (or too expensive) to be implemented in special purpose hardware, this seems a good approach.

In casu, we utilize processors of the transputer family to realize the parallel implementations. On these transputers, a complete three-dimensional computer animation system, incorporating an object modeler, a motion specification subsystem, as well as a renderer, has been implemented. It turns out that many of the calculation intensive rendering techniques can be implemented successfully: procedurally generated 3D texture-, reflection-, bump- and transparency mapping techniques (for rendering types of stone, wood, marble, water, clouds, metals, bricks, etc.), ray tracing algorithms (for generating realistic reflection, transparency and shadowing effects), as well as radiosity methods (for generating realistic images of diffusely reflecting surfaces). An overview of our realizations, together with some relevant references, will be given in this paper.

THE ANIMATION HARDWARE

The existing image synthesis architecture in the laboratory consists of a parallel network of transputers. The transputer of the current generation is a 32-bit parallel processor sustaining 20 MIPS / 1.5 MFlops at 25 MHz. Each of its four serial links allow bidirectional synchronized communication with other
members of the transputer family at 20 Mbit / sec. Special purpose logic on chip is provided for rapid context switching in the scheduling and descheduling of the parallel processes running on the processor.

The entire hardware platform of a graphics system in the lab consists of the following components: (i) a PC, utilized as a host for booting the network and for housing the I/O equipment (mouse, graphical tablet & stylus, scanner, video recorder interface, etc.); (ii) host transputer boards (with transputers having up to 32 MB of local memory) used for connection with the PC and for storing the database as well as application specific programs (modelling, motion specification, programming environment, etc.); (iii) a farm or pipeline of transputers having typically 2 to 4 MB of local memory, utilized for executing the calculation intensive rendering algorithms; (iv) two graphics boards, which respectively deliver a non-interlaced high-resolution signal for displaying the user interface and an interlaced true color PAL video signal for displaying the rendered images.

OBJECT MODELLING

In general, the making of an architectural animation can have two very distinct intentions: it can be made for structural analysis purposes, or it can be made for demonstrational purposes. Depending on this purpose, the underlying model representation is of large importance. In the former case, exact mathematical representations through higher order curved surfaces might be necessary, while in the latter case, a polygonal approximation of objects often will suffice. Although we already have been experimenting with bicubic patches to represent curved surfaces exactly (Lamotte [13]), our main attention currently is directed towards representing objects to be animated with a boundary representation consisting of polygonal faces. In our parallel ray tracing algorithm, a procedural representation can also be applied. (Note: As more and more commercial three-dimensional modelling packages become available, it is moreover a necessity to have an "open" file format with respect to object representation, so models created on divers modelling packages can be animated on our animation system; cf. the ray tracing figure later on.)

No matter what representation is chosen, objects should be
represented in such a way that they can be easily created and manipulated by human operators. Appropriate tools have to be offered, enabling animators, designers and engineers to define and shape objects according to their specific wishes. A good user interface, with multiple simultaneous views, is very important in order to reduce the amount of time it takes to create the desired object models: the benefit of having a performant parallel rendering system can indeed disappear completely if a cumbersome user interface on top of it doesn’t allow the models to be specified quickly. On the one hand, the available tools should be accessible to a novice animator/user of the system via a graphical interface, while they also should be accessible to an experienced user through a fast "short-cut driven" interface. On the other hand, a continuous visual feedback has to be available alerting the user in case something should have gone wrong. This implies it is necessary to have the ultimate high-quality rendering techniques available in the modelling stage.

Objects in our system are structured as a hierarchical composition of more elementary objects, enabling advanced possibilities during the motion specification. Each of these elementary objects has a reference to its parent object, and consists of a number of groups, being sets of polygons with identical coloring, shading and/or texturing attributes.

At the user interface level, we developed a modeller in which hierarchical objects can be created, deleted, repositioned, shaped, etc. in an interactive fashion. Starting from 2D primitives (polygons, splines, ...) simple 3D objects can be created, e.g., by transformational sweeps. Two examples of elementary transformational sweep methods, are those in which objects are created by rotating and linking primitive surfaces around an axis or by translating and linking primitives in a certain direction. Through parameterization and combination of these primitive transformations (e.g., scaling and/or translating the 2D primitive while rotating around an axis), more sophisticated objects can be created. Although these more sophisticated might not always be used in the creation of buildings, they can conveniently be utilized in the creation of "entourage" objects. Figure 1 gives snapshot of our object modelling screen in which such an object, in casu a car to be used in an animation, is being modelled.
Aside the transformational sweep techniques, constructive solid geometry (CSG) tools can also be exploited very well in architectural modelling. These tools enable creation of models consisting of the union, the difference, or the intersection of other two-dimensional or three-dimensional models. Concretely, we implemented a variant of the polygonal CSG operations found in (Laidlaw [12]).

MOTION MODELLING

Having a two-dimensional screen, controlling the motion of three-dimensional object models in a three-dimensional virtual world is a non-trivial task. Aside the objects, the virtual camera itself is often also to be animated: this is especially true in architectural computer animations, as the virtual camera is frequently the main (or even only) moving item.

In order to enable an operator to create sophisticated motion, we developed a user-friendly graphical motion specification system, conform the object modelling subsystem with multiple simultaneous views. Several important key issues support the user-friendliness: (1) the utilization of hierarchies, (2) the manipulation of four-dimensional curves, and (3) real time
By defining objects in a hierarchical structure, it becomes much more easy to specify local motion. We can take the example of a fork-lift: by defining the fork in a co-ordinate axis relative to the vehicle itself, the fork can be moved in the vertical direction of its local co-ordinate system, while the vehicle itself is following a certain path towards its destination in the co-ordinate system of the world.

For specifying smooth motion of objects, lights and cameras over three-dimensional paths, we provided the operator with an implementation of four-dimensional splines. These splines are explicitly represented as three-dimensional objects in the scene, in which editable control points are foreseen for manipulation of position and orientation. Figure 2 shows a screen snapshot of our motion modelling subsystem, in which a fly-through of a radiosity shaded (cf. rendering section) room is prepared. The path itself is represented as a spline curve, while the control points have a cube representation (see the top-left window, as well as the window below it, in which a more distant view is given). The virtual camera is represented as small pyramidal shape. In the bottom left window, two-dimensional editable projections of the splines are represented; besides the x-, y-, and z-components of the motion paths, the superimposed time-dimension can also be edited accurately here. Each desired motion path (and according motion speed) can thus be created by simply inserting and repositioning control points on the curves. An important issue is that the effect of a changement on a curve in any particular window, is automatically displayed in the other windows on the screen.

Last but not least, we implemented the facility to display in real time (25 frames/s) a part of the animation on the screen in any desired rendering technique. This so-called real time preview is very important in an animation environment, as it offers the opportunity to get an exact impression of the animation movements without having to make a video recording. The cycle between animation specification and verification can be much faster and more precise in this way. The preview is realized by (i) calculating rapidly a number of frames at lower resolution (one third of the resolution), (ii) storing the
frames in the memory of the worker boards, and (iii) finally dumping them in real time to the video RAM via the transputer links.

Figure 2. Snapshot of our motion specification subsystem.

**RENDERING**

**General issues**

In our rendering system a parallel Z-buffer algorithm as well as a parallel ray tracer are implemented. In the ray tracer as well as in the Z-buffer, parallelism is introduced by dividing the overall screen into screen regions and by distributing the pixel values to be calculated among the available processors. The m x n resolution of the screen regions depends on the rendering algorithm at issue. For ray tracing, this resolution is n x n (with n=4 or 8 or 16), whereas in scan conversion algorithms the horizontal screen region resolution equals the overall resolution. The exploitation of coherence is one of the main factors in the resolution determination.

The amount of work - opposed to the amount of communications - to be done at the pixel level, is crucial for maintaining a linear ratio "performance / number of processors". The load balance in the ray tracer can be kept optimal relatively easily, as (1) rays are fired in the scene independently of one another, and (2) the number and the complexity of the calculations to be performed per pixel are rather large. The more simple the processing per pixel
becomes, the more complicated it becomes to keep the processors usefully busy. Hence, the optimizing of parallelization of a Z-buffer algorithm is already a bit more difficult. Through introduction of some buffering processes a good distribution of the rendering pipeline (transformation - clipping - perspective division - shading - scan conversion - image storage) over the available processors, it is possible to maintain an almost linear expandability.

Within the basic rendering algorithms, some other facilities are integrated also. First of all, parallel algorithms to conquer anti-aliasing (Crow [6]; Van Reeth [18]) are incorporated. Aside the anti-aliasing algorithms, a number of parallelized methods are available to enhance the surface structure of 3D objects via texture mapping techniques (Heckbert [9]). Appropriate illumination models (Hall [8]) are foreseen in the shading process, enabling the simulation of several lighting effects. In the succeeding paragraphs, the most important realizations with respect to architectural visualization are elucidated.

Ray Tracing
In the ray tracing algorithm, parallelism in general can be introduced by dividing the problem in image space, or by dividing the problem in object space. In the latter methodology, object space is subdivided and distributed to different processors. Each of the processors consequently performs the amount of processing necessary for the subdivided space parts which they control. The difficulty with this approach consists of keeping the processors usefully busy. Indeed, processors controlling large parts of non-empty object space receiving relatively little rays may become idle, unless some non-trivial balancing scheme is realized. In the former methodology, the screen is subdivided into rectangular regions, and the work to be done for rendering these regions is distributed among the processors. In order to have a good load balance, relatively small regions should be assigned to the different processors in an interleaved fashion, so the "more time consuming" parts of an image are likely to be encountered by more processors. Given an interleaved distribution, one has a choice between realizing a static distribution in which the processors are always given the same regions, or a dynamic distribution in which the processors are given regions depending on their work load.

We follow the screen space division methodology: the screen is
subdivided into rectangular regions of \( n \times n \) pixels \((n = 4, 8 \text{ or } 16)\) that have to be ray traced in parallel by the different workers. The master transputer keeps track of the regions that yet have to be processed and distributes the work to be performed. Initially each processor gets a screen region, so each processor starts working. After having rendered the pixels within the region, the processor forwards the result and asks for the next region to be calculated. Hence, processors that coincidently have to process less complicated regions will simply be processing more regions in comparison to the processors that have to calculate more complicated regions (rather than going into an idle waiting state). Appropriate buffering and priority techniques are necessary in order to avoid unnecessary elapsed waiting times (Van Reeth [19]).

Additional optimizations, of which the exploitation of coherence (Kaplan [11]) is the most important, have been incorporated in the ray tracer. (Indeed, the parallelization of an efficient algorithm is always preferable above parallelizing a less efficient one). In casu, we implemented a parallel version of the voxel structure presented in (Amanatides [1]). In this approach, space is subdivided uniformly, and a ray is intersected only with objects lying in the subdivision currently being pierced by the ray.

An important additional issue to take into account when implementing a parallel ray tracing system concerns the distribution of the object database. As each of our workers has 2MB DRAM at its disposal, it is not uncommon that the entire database organized into the voxel structure can be copied to each of the workers (Note: We expect moreover that in the near future even 4MB or more for every processor will be a very acceptable amount of memory, even when using several dozens of processors). If this is the case, the scene is broadcasted to each of the workers prior to the start of the actual ray tracing and consequently no further scene transmission is necessary. In case the database organized into the voxel structure doesn't fit into the memory of the workers, scene transmission becomes the problem to be tackled. Given the fact that the ray tracing process has a non-uniform work distribution on several levels (on the intersection level, e.g., the intersection of a ray with a bi-cubic patch (Lamotte [13]) takes up far more time than that of ray tracing spheres, whereas on screen level the amount of work varies almost inherently depending on screen content), transmission upon request is the methodology
Visualization and Intelligent Design

We followed. In order to make this transmission upon request efficient, appropriate data structures have to be defined. Concretely, we utilize the voxel structure, mentioned in the previous paragraph, not only for subdividing space and thus speeding up the intersection process, but also for subdividing the database and thus speeding up the scene transmission process (as only objects in the neighborhood of the ray are transmitted). Details of this implementation can be found in (Van Reeth [19]). Average anti-aliased rendering timings of scenes (of a few thousand polygons) fitting in the workers' memories on average take less than five minutes to be rendered on a 13 processor network. For scenes not fitting in the workers' memories, a small overhead has to be taken into account. We finalize this paragraph by referring to Figure 3, in which a city model (taken from Autodesk 3D Studio) is ray traced.

![Image](image-url)

Figure 3. A ray traced image of buildings in the mist.

Radiosity

During the last years, a substantial amount of research regarding realistic image synthesis has been focused on the realisation of global illumination models. Two approaches for solving the illumination problem in diffusely reflecting environments can be recognized. The first one is based on extensions to the traditional ray tracing algorithms (Cook [5], Kajiya [10]). The second category incorporates radiosity algorithms (Cohen[4], Goral [7], Nishita [15]). One of
the major advantages of the radiosity approach is that it results in a solution independent from the position of the viewer.

Although the early radiosity methods gave rise to the most realistically rendered images of that period, they suffered from a storage and time cost that were $O(n^2)$ - with $n$ the number of elementary surface patches -, so they were not practical to be used for high complexity scenes. With the introduction of progressive radiosity algorithms, however, the performance has drastically improved (Cohen [3], Wallace [20]).

A next step upwards with respect to improving performance is to incorporate parallel computations to solve the radiosities. As the computations of the form factors take up most of the time, it could be worthwhile to parallelize them. Several approaches involving coarse-grained parallelism can be found in the literature (Baum [2], Puech [16], Recker [17]). They are based on parallel computation of the form factors with a hemi-cube like technique to determine visibility and illumination.

We implemented a parallelisation of the form factor computations based on ray tracing methods. The calculation of the radiosities (given the assumption of global Lambertian diffuse illumination) narrows down to solving a set of equations in which the relation of energy leaving each surface element and arriving on each surface element is expressed:

$$B_{dAi} \ dAi = E_{dAi} \ dAi + \rho_{dAi} \ \int B_{dAj} \ F_{dAj-dAi} \ dAj \ \text{(1)}$$

where $B_{dAi}$ is the radiosity (energy per unit area) of differential area $dAi$; $dAi$ is a differential surface area $i$; $E_{dAi}$ is the energy emission of differential area $dAi$ (i.e. a light source); $\rho_{dAi}$ is the reflectivity of differential area $dAi$ (a value between 0 and 1 indicating which fraction of the arriving light is reflected); and $F_{dAj-dAi}$ is the form factor from $dAj$ to $dAi$ (a value between 0 and 1 indicating which fraction of light leaving $dAj$ arrives at $dAi$). As the number of differential areas is infinite, the number of radiosities to be calculated is also infinite. Hence, a discretized version of the above integral
equation is used. In this approach the scene is discretized into finite areas rather than differential ones, resulting in equation (2):

$$ B_{Ai} = E_{Ai} A_i + \rho_{Ai} \sum_j B_{Aj} F_{Aj-Ai} A_j \quad (2) $$

Using the reciprocity relationship $F_{Ai-Aj} A_i = F_{Aj-Ai} A_j$, this becomes:

$$ B_{Ai} = E_{Ai} + \rho_{Ai} \sum_j B_{Aj} F_{Ai-Aj} \quad (3) $$

Utilizing the ray tracing methodology, one has to compute the form factor from a lighting area $A_2$ to a differential area $dA_1$ at each vertex. For the case in which one subdivides the area $A_2$ uniformly in $n$ parts, the following form factor (including a visibility term $\delta_i$) is to be computed (Wallace [20]):

$$ dF_{A2-dA1} = dA_1 \sum_{i} \delta_i \frac{\cos \theta_{1i} \cos \theta_{2i}}{\pi r_i^2 + A_2 / n} \quad (4) $$

The radiosity $B_1$ at a vertex 1 received from an area $A_2$ is consequently given by (Wallace [20]):

$$ B_1 = \rho_1 B_2 A_2 \sum_{i} \delta_i \frac{\cos \theta_{1i} \cos \theta_{2i}}{\pi r_i^2 + A_2 / n} \quad (5) $$

As equation (5) has to be calculated for each vertex in the scene, and this for several dozens of progressive iterations, it is obvious a lot of computation is involved here. Based upon a variant of our parallel ray tracing methodology, these equations can be solved very quickly: for scenes like the one in figure 4, on average a few seconds are needed per iteration on a 13 processor system. We again refer to other material (Lamotte [14]) for the implementational details of this approach, as they go behind the scope of this paper. Figure 4 shows a snapshot of a radiosity shaded office room.
Texture mapping

In the creation of naturally looking indoor and outdoor sceneries, the utilization of mapping techniques (Heckbert [9]) is of main importance. Several mapping categories can be defined: (i) texture mapping, for defining structured surface color (e.g. for generating marble); (ii) reflection mapping, for giving a metallic look to surfaces; and (iii) bump mapping for changing the surface structure of materials (e.g. making it look more roughly). In each of these categories, one can have procedurally defined mappings, as well as a bitmapped approach (in which often scanned-in images are used). We incorporated both mapping methodologies in the parallel shading software. Figures 4 and 5 show respectively an indoor and an outdoor scene in which texture mapping is heavily used. The anti-aliased images took respectively 2.5 and 4 minutes to be rendered on a 13 processor network.

CONCLUSIONS

We have shown that the implementation of parallel rendering techniques can be a viable solution for solving the problem of ever increasing demand for higher realism in the making of architectural animations. It turns out that especially
Figure 5. An indoor scene, rendered with several marble textures

Figure 6. Outdoor scene with texture mapped trees, clouds, gravel and lawn
the complex computational algorithms offer great potential in the area of parallelisation. The next large milestone with regard to demonstrational architectural visualisation will probably be found in the direction of real-time image synthesis; we are convinced that parallel processing technologies can't be omitted in the approach of that milestone.

ACKNOWLEDGEMENTS

The help of all the colleagues in the laboratory who contributed in the realization of the animation- and rendering algorithms is greatly appreciated. This work is partly funded by a grant from the Belgian National Fund for Scientific Research (NFWO).

REFERENCES


