EUROGRAPHICS '91 / F.H. Post and W. Barth (Editors)
Elsevier Science Publishers B.V. (North-Holland)



# Using temporal and spatial coherence for accelerating the calculation of animation sequences

Eduard Gröller, Werner Purgathofer

Technische Universität Wien
Institut für Computergraphik
A-1040 Karlsplatz 13/186/2

**Abstract**

Ray tracing is a well known technique for generating realistic images. One of the major drawbacks of this approach are the extensive computational requirements for image calculation. When generating animation sequences frame by frame the computational cost might easily become intolerable. In the last years several methods have been devised for accelerating the computational speed by using spatial and temporal coherence. While these techniques work only under certain restrictions, a new approach is presented in this paper which leads to a considerable speed-up of the calculation process without putting any limitations on camera or object movement. In principle, the method is an extension of /ArKi87/, where rays are considered points in 5D space, by the time dimension. CSG is used for object description and has been modified correspondingly to allow easy use of coherence properties. The paper describes the theoretical background and the main concepts of a practical implementation.

Key words: Ray tracing, computer animation, ray space classification, temporal coherence, spatial coherence, acceleration

## 1. Introduction

Ray tracing is a technique where the color information for every pixel is calculated by casting rays through object space. Ray - object intersections are computationally by far the most expensive part in the image generation process and should be reduced or simplified as far as possible. Soon after ray tracing was introduced, spatial coherence properties were used to accelerate image calculation. The use of simple bounding volumes (boxes, spheres, plane sets, ...) led to a considerable acceleration of the ray tracing method. Subdivision methods turned out to be another very powerful optimization tool. /FuTa86/ and /Glas84/ examine object space subdivisions. /FuTa86/ describe a regular voxel subdivision while /Glas84/ uses an octree data structure for organizing the object space subdivision. In /Gerv86/ an image space subdivision method is presented and /ArKi87/ introduce ray space subdivision. Temporal coherence denotes similarities between consecutive frames of animation sequences. Most of the techniques that take advantage of temporal coherence properties work only under some restrictions. In /Hubs81/, one of the earliest works dealing with temporal coherence, a method is presented that works just for convex polyhedra. In /Hube88/ and /MuHi90/ methods are examined that work only for a fixed camera position. Some other methods are making a trade-off between

calculation speed and image quality (/Badt88/). Our goal was to develop a fast and general method (no restrictions on camera and object movement) for the calculation of animation sequences without loss of image quality.

## 2. Coherence in ray space

Starting out from the ideas presented in /ArKi87/ a new approach using spatial as well as temporal coherence is developed. Every ray that is generated during the calculation of an animation sequence is considered to be a point in an appropriate 6 dimensional ray space. In this space a ray is defined by its origin (3 coordinates), its direction (2 coordinates) and its time of generation (1 coordinate). The organization of ray space has to be done in such a way, that certain requirements are fulfilled. Rays close together in ray space (similar origin, direction and time of generation) should traverse similar regions of object space, so that coherence properties can be transfered from ray space to object space. In order to avoid as many as possible time consuming ray - object intersections the concept of the 6D ray space is used to eliminate intersection tests for groups of coherent rays, thereby distributing the cost of this eliminiation step over a number of different rays. The algorithm proceeds as follows: A 6D bounding volume B (a subset of ray space) is calculated, which contains every ray that might be generated during the calculation of the animation sequence. This bounding volume is then partitioned hierarchically into disjoint volumes or ray sets $B_i$. For every ray set $B_i$ a so called candidate set $C_i$ is calculated, which contains only that small number of objects of the whole object scene that might be intersected by rays of $B_i$.

Ray - object intersection is done in 2 steps:

1. finding a ray set $B_i$ that encloses the ray considered (ray classification)
2. ray intersection by using the small number of objects of the candidate set $C_i$

To ensure fast execution of step 1 the bounding volume B is partitioned into geometrically simple ray sets $B_i$ (hypercubes). Every ray set $B_i$ is associated with a region in 3D object space (a so called "beam"), namely with the set consisting of those points that could be reached by a ray of $B_i$. For a given ray set $B_i$ only those objects that lie at least partially inside the associated 3D beam are relevant for intersection tests. Candidate sets are therefore created by intersecting the whole object scene with these beams. This can be done efficiently, when the beams themselves have a simple geometrical shape (e.g. convex polyhedra). The 6D ray space has to be organized so that a simple ray set $B_i$ (hypercube) has a simple beam in 3D object space. Partitioning of the ray space is done "on the fly", candidate sets are created only when needed. Small ray sets (high spatial and temporal coherence) guarantee small candidate sets, but the cost of creating these sets is amortized only among a limited number of rays. So partitioning of a part of ray space is stopped either when the candidate set is small enough or the ray set has minimal size. An optimal trade-off between these two constraints (sizes of ray set and candidate set) can be found through experimental results.

## 3. 6D ray space

A ray is defined as a point in 6D space:

$$\text{ray } r = (x, y, z, u, v, t)$$

| | |
|---|---|
| x, y, z .... | origin of the ray |
| u, v ..... | ray direction |
| t ..... | time of generation |

A ray set $B_i$ is defined as a hypercube

$$B_i = [x_1,x_2]\times[y_1,y_2]\times[z_1,z_2]\times[u_1,u_2]\times[v_1,v_2]\times[t_1,t_2]$$

which consists of all those rays whose coordinates as 6D points lie in the 6 specified intervals. Using spherical angles (longitude, latitude) for determining the ray direction would have the undesirable consequence that a simple ray set $B_i$ has a beam with a non polygonal boundary. By using a so called direction cube, rays can be defined so that the 3D beams associated with 6D hypercubes are convex polyhedra.

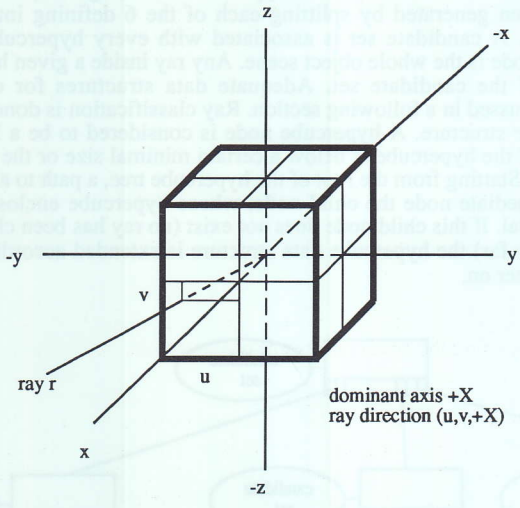


Figure 1: direction cube

All rays with the same origin are partitioned into 6 disjoint sets depending on the intersection of the ray direction with an axis aligned cube with sides of length 2 positioned at the ray origin. A ray direction is then specified by a "dominant axis" (corresponding to the intersected face of the cube) and the two intersection values u, v, $-1 \le u,v \le 1$ (see Figure 1). Ray space is therefore represented by six hypercubes, one hypercube for each dominant axis:

$$\begin{aligned}\text{ray space} = {} & R^3\times([-1,1]\times[-1,1],+X)\times T \cup \\ & R^3\times([-1,1]\times[-1,1],-X)\times T \cup \\ & R^3\times([-1,1]\times[-1,1],+Y)\times T \cup \\ & R^3\times([-1,1]\times[-1,1],-Y)\times T \cup \\ & R^3\times([-1,1]\times[-1,1],+Z)\times T \cup \\ & R^3\times([-1,1]\times[-1,1],-Z)\times T\end{aligned}$$

with $R^3$ . . . . . possible ray origins
$([-1,1]\times[-1,1],+X)$ . . . . . set of all ray directions with dominant axis +X
T . . . . . time interval

The six disjoint parts of ray space are stored in 6 distinct hierarchical data structures, one for every dominant axis. Whenever a ray is classified the relevant data structure is found by calculating the dominant axis of the ray in question.

## 4. Data structures for ray space

The ray space information is administered in 6 hypercube trees (Figure 2). Any hypercube node has up to 64 children generated by splitting each of the 6 defining intervals of the hypercube in the middle. A candidate set is associated with every hypercube node. The candidate set of the root node is the whole object scene. Any ray inside a given hypercube can only intersect objects of the candidate set. Adequate data structures for candidate set representation will be discussed in a following section. Ray classification is done by recursive traversal of the hypercube structure. A hypercube node is considered to be a leaf node (no further partioning done) if the hypercube is below a certain minimal size or the candidate set contains only few objects. Starting from the root of the hypercube tree, a path to a leaf node has to be found. At an intermediate node the child node, whose hypercube encloses the ray, is selected for further traversal. If this child node does not exist (no ray has been classified to lie in this part of ray space so far) the hypercube data structure is extended accordingly. Precise algorithms will be given later on.

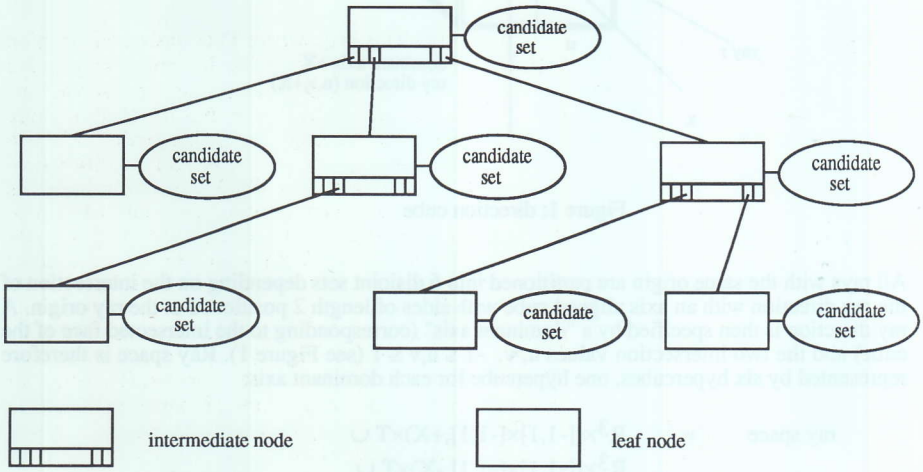


Figure 2: hypercube tree

When the relevant leaf node has been found, the ray - object intersection is done using the (hopefully) small candidate set.

## 5. Data structure for candidate sets

The object scene as well as the candidate sets are stored as CSG data. An object representation is defined as a binary tree where the leaf nodes correspond to geometrical primitives (cube, sphere, cylinder, cone, ...) whereas in the intermediate nodes set operations (union, intersection, difference) define how the objects represented by the two child nodes are combined. Storage efficient description of a moving object leads to an extension of the CSG model, so that the positions of an object over a period of time can be specified in a short and concise way (Figure 3).

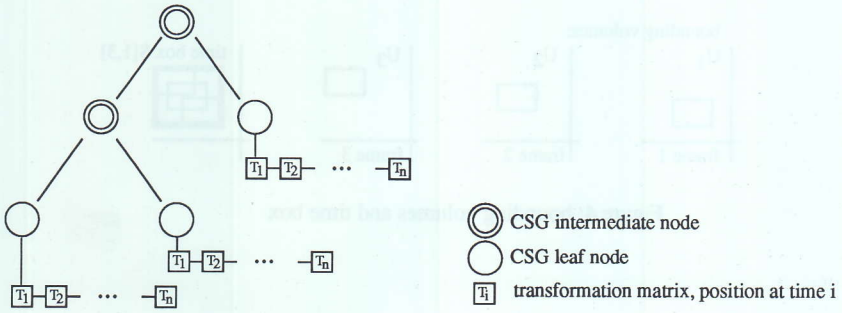


Figure 3: extended CSG tree

For every leaf node a list of transformation matrices $T_i$ is kept, where $T_i$ describes the position of the object at time i. When calculating frame i, matrices $T_i$ are used to determine the position of an object. In this data structure the object positions are stored for those specific discrete time instants the frames are calculated for. In some cases, e.g. no motion or uniform motion, more storage efficient data structures than lists are advisable. Storing all possible positions in one data structure allows the processing of the object not only for specific instants but for whole time intervals thereby making use of temporal coherence properties. Candidate sets are stored as extended CSG trees. When generating a candidate set only new intermediate nodes have to be created. Existing leaf nodes of the CSG tree can be reused by pointer reference. As leaf nodes usually require most of the storage space of a CSG tree, redundant storage of these nodes must definitely be avoided. One of the earliest optimization efforts concerning CSG dealt with bounding boxes and bounding spheres /Roth82/. For every node of a CSG tree bounding boxes can be calculated that enclose the represented object. Intersection tests are done with the simple bounding box first. Only if the bounding box is hit by the ray a much more time intensive ray - object intersection test has to be performed. How can this very useful method be incorporated into the concept of extended CSG trees? A straightforward approach of storing lists of bounding boxes, one bounding box for every time step i, in the intermediate nodes as well as in the leaf nodes would greatly increase the needed storage space. In our approach we use so called time boxes, an extension of bounding boxes by a temporal coordinate. A time box S[i,j] encloses the object during the time interval [i,j] (see Figure 4, time spheres could be used as well). Starting from bounding boxes $U_i$, $U_{i+1}$, ..., $U_j$ for time steps i, i+1, ..., j respectively, S[i,j] can be calculated easily. If the object is not moving during the interval [i,j], S[i,j] simply reduces to the usual bounding box. These time boxes are stored in the extended CSG tree (Figure 5) in the following way: For every leaf node a binary tree of time boxes S[i,j]

is calculated. The root of this binary tree represents S[1,n], the bounding volume that encloses the primitive element over the interval [1,n]. The leaf nodes are the simple bounding boxes $U_1$, ...,$U_n$ which are combined in a binary way to generate the bounding tree.

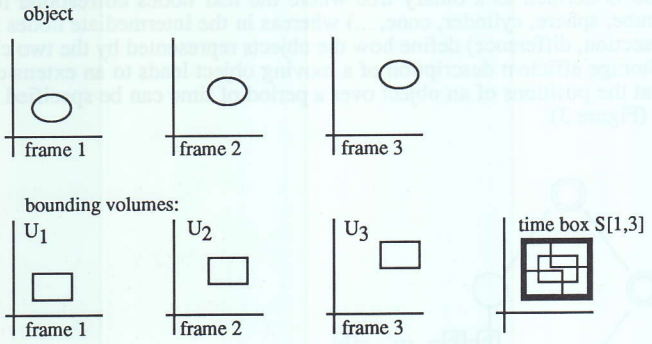


Figure 4: bounding volumes and time box

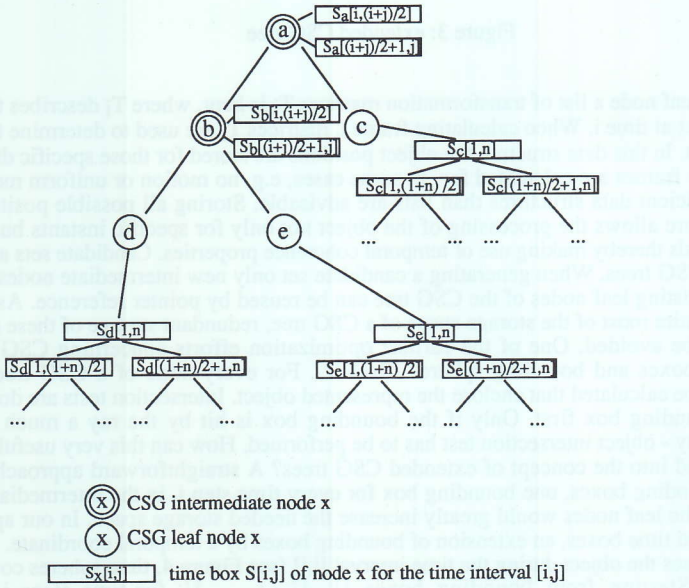


Figure 5: object description for time interval [i,j] as an extended CSG tree (matrices $T_i$ not shown), time boxes S[i,j]

As the leaf nodes of the candidate sets are stored only once, the entire bounding trees are kept. If an extended CSG tree represents an object for the time period [i,j] only two time boxes (i.e. S[i,(i+j)/2], S[(i+j)/2,j]) are stored at intermediate nodes. When the hypercube node, the extended CSG tree is associated with, has to be split, one of these two time boxes is used to generate the candidate set for the child hypercube which is defined for time interval either [i,(i+j)/2] or [(i+j)/2+1,j].

## 6. Algorithms

In a preprocessing step a 6D bounding volume is determined, enclosing all rays that might be relevant during the calculation of the animation sequence. Taking a box enclosure of all possible eyepoints (origins of primary rays), and the time box S[1,n] of the entire object scene (origins of secondary rays) gives a good estimation of all possible ray origins. Due to global lighting effects (reflection, transparency) no useful restrictions on ray directions seem apparent, all directions are considered possible (u, v $\in$ [-1,+1] for every dominant axis). The time interval is set to [1,n] if n frames shall be calculated. Using this information the six root hypercubes are generated. The candidate sets for these root nodes are extended CSG trees of the whole object scene. For every ray r, ray - object intersection is done in the following two steps (hq is a reference to the hypercube structure):

step 1:

```
candidate_set := ray_classification(r, hq)

ray_classification (r:ray; hq:hypercube_structure): CSG tree
begin
    determine the dominant axis of ray r

    traversal of the hypercube structure to find the smallest
    existing hypercube node hn that contains r

    if type of hn = leaf node
    then
        (* classification completed *)
        return (candidate set of hn)
    else
        (* hn has to be split *)
        return (generate_new_path(r, hn))
end

generate_new_path (r:ray; hn:hypercube_node): CSG tree
(* returns the candidate set of a newly generated hypercube leaf node *)
begin
    ray r determines which of the 64 possible children hn_child
    of hn has to be generated

    calculation of the candidate set of hn_child

    if hn_child or associated candidate set is small enough
    then
        type of hn_child := leaf node
```

```
            return (candidate set of hn_child)
        else
            (* further splitting necessary *)
            type of hn_node := intermediate node
            return (generate_new_path(r, hn_child)
    end
```

The calculation of the candidate set is done by intersecting the 3D beam of the child node hn_child with the candidate set of the parent hypercube node. For fast beam-object intersection the beam as well as the object are approximated by simple, easy to intersect, enclosing volumes. Intersection of the approximating volumes (instead of beam - object intersection) will result in slightly larger candidate sets. Depending on the different approximation volumes for beam and object there are different possibilities for this intersection step:

method a: 3D beam: approximated by simple axis aligned planes
object: approximated by bounding boxes

method b: 3D beam: approximated by simple axis aligned planes
object: approximated by bounding spheres

method c: 3D beam: approximated by a cone
object: approximated by bounding spheres

In method c the cone - sphere intersection can be calculated easily. A cone however is not as good as axis aligned planes for approximating the 3D beam. So with method c the candidate sets are calculated faster, but are usually larger than with method a or b. For further details see /ArKi87/.

step 2:

```
            intersection_ray_object (r, candidate_set)
            (* usual intersection of a ray with a CSG tree by recursive traversal *)
```

## 7. Extensions and modifications

Sometimes ray origins of primary rays lie far outside the bounding volume of the object scene. This leads to an unnecessarily large (and to a great extent empty) 6D bounding volume of the six roots (one for every dominant axis) of the hypercube structure. To ensure a more balanced partitioning of ray space only the 3D object bounding volume is used for the definition of the 6D bounding volume. Ray origins of primary rays outside the 3D object bounding volume are not considered. Ray classification has to be modified slightly for these rays. Because such a ray, as a 6D point, does not lie inside the 6D bounding volume the intersection point of the ray with the 3D object bounding volume is taken as the new ray origin. Ray classification is done with this modified ray as usual.

Adjacent primary rays are close together in ray space. First a quick test is done whether a ray lies inside the hypercube node found for the previous primary ray. This very often eliminates the need to traverse the hypercube structure.

Generally rays are calculated in temporal order. So parts of the hypercube structure that correspond to already processed time intervals are not needed anymore, valuable memory space can be reused.

No ray intersection test is done with candidate sets of intermediate hypercube nodes (Figure 2). Omitting these candidate sets leads to a considerable reduction in storage requirements. The candidate set of the root node is kept however. Whenever node splitting is performed, the candidate set for the new node is generated by intersecting the candidate set of the root with the 3D beam of the new hypercube. If it turns out during candidate set generation that the new candidate set is still too large and hence further splitting is necessary (the new hypercube node becoming an intermediate node as well) the candidate set need not be generated completely as it is not stored at the intermediate node anyway. Candidate set creation will, however, take longer as the whole scene is always taken as the starting point. If on the other hand candidate sets are stored in intermediate nodes, the generating step can start out from the (small) candidate set of the node to be split, at the expense of greater storage requirements.

## 8. Implementation and results

A test system was implemented /Reic90/ in Pascal on a VAX-cluster (5 VAX-Stations 2000, 1 VAX-Station 3200 with 3MB and 4MB user accessible memory respectively) using components of RISS (Realistic Image Synthesis System), a software package for the generation of realistic images developed at our department /GePu88/. Storage restrictions allowed only the processing of simple scenes. Experimental results show that even for simple scenes (≈30 objects, 10 frames, 200×200 resolution, maximal height of hypercube tree 4) a speed-up factor of about 2 can be obtained (see example pictures). For a finer partitioning of ray space (higher hypercube structure, smaller candidate sets) higher speed-up rates can be expected.

For a theoretical complexity analysis certain assumptions on the statistical distribution of the objects in the scene as well as of rays in 6D ray space would have to be made. The distribution of rays in 6D ray space, however, greatly depends on the varying camera and object movements in different animation sequences. So a general theoretical complexity analysis was not done.

The method presented uses temporal coherence properties without putting any restrictions on either camera or object movement. Images are generated with the same quality as a frame by frame approach would yield.

Example 1:*

object scene: 50 spheres with different surface properties
2 spheres moving
one light source, position of camera moving, 10 frames, adaptive oversampling
Resolution: 200×200
total number of rays: 2 672 939
primary rays: 1 102 900
speed-up: 45 %

Size of hypercube structure (depending on beam - object intersection method used):

| method | # of hypercube nodes |
|---|---|
| a: | 19 331 |
| b: | 18 104 |
| c: | 27 440 |

---

* See page 528 for Figure 6: Example 1 (frames 1 – 10)

average size of candidate set (max = 10):

| method | # of CSG nodes |
|---|---|
| a: | 7.25 |
| b: | 7.22 |
| c: | 8.55 |

primary ray is in the same hypercube as previous primary ray:

| method | # of rays |
|---|---|
| a: | 905 830 (82.13%) |
| b: | 906 761 (82.22%) |
| c: | 890 488 (80.74%) |

Example 2:*

object scene: 23 elementary objects
camera and blue plate moving, 10 frames, adaptive oversampling
Resolution: 200×200
total number of rays: 1 154 441
primary rays: 564 515
speed-up: 37 %

Size of hypercube structure (depending on beam - object intersection method used):

| method | # of hypercube nodes |
|---|---|
| a: | 3 634 |
| b: | 3 979 |
| c: | 3 446 |

average size of candidate set (max = 10):

| method | # of CSG nodes |
|---|---|
| a: | 5.31 |
| b: | 5.45 |
| c: | 5.79 |

primary ray is in the same hypercube as previous primary ray:

| method | # of rays |
|---|---|
| a: | 296 695 (52.56%) |
| b: | 294 916 (52.24%) |
| c: | 294 894 (52.24%) |

## 9. References

/ArKi87/ J. Arvo, D. Kirk, Fast Ray Tracing by Ray Classification, Computer Graphics, Vol. 21(4), July 1987, pp. 196-205.

/Badt88/ S. Badt Jr., Two algorithms for taking advantage of temporal coherence in ray tracing, The Visual Computer 4, 1988, pp. 123-132

/FuTa86/ A. Fujimoto, T. Tanaka, K. Iwata, ARTS:accelerated raytracing system, IEEE Computer Graphics and Application, April 1986, pp. 16-26

---

* See page 529 for Figure 7: Example 2 (frames 1 – 10)

/Gerv86/ M. Gervautz, Three improvements of the Ray-Tracing Algorithm for CSG-Trees, Computer and Graphics, Vol.10(4)1986, pp. 333-339

/GePu88/ M. Gervautz, W. Purgathofer, RISS-Ein Entwicklungssystem zur Generierung realistischer Bilder, Informatik Fachberichte 182, 1988, pp. 61-79

/Glas84/ A.S. Glassner, Space subdivision for fast ray tracing, IEEE Computer Graphics and Applications 4(10), October 1984, pp. 15-22

/Hube88/ S. Huber, Ausnützung zeitlicher Kohärenz beim Raytracing, Diplomarbeit am Institut für Praktische Informatik, TU Wien 1988

/Hubs81/ H. Hubschman, S. Zucker, Frame-to-frame coherence and the Hidden Surface Computation: Constraints for a Convex World, Computer Graphics, 15(3), August 1981, pp. 45-54

/MuHi90/ K. Murakami, K. Hirota, Incremental Ray Tracing, Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics, June 1990, pp. 15-29

/Reic90/ H. Reichardt, Ausnützung zeitlicher und räumlicher Kohärenz bei der Berechnung von Bildfolgen, Diplomarbeit am Institut für Praktische Informatik, TU Wien, 1990

/Roth82/ S. D. Roth, Ray Casting for Modeling Solids, Computer Graphics and Image Processing, Vol. 18, 1982, pp. 109-144

## Acknowledgment

We would like to thank H. Reichardt for valuable discussions and participation in the implementation of the methods described. This work was partly sponsored by the "Hochschuljubiläumsfonds" and by "Impuls GmbH".

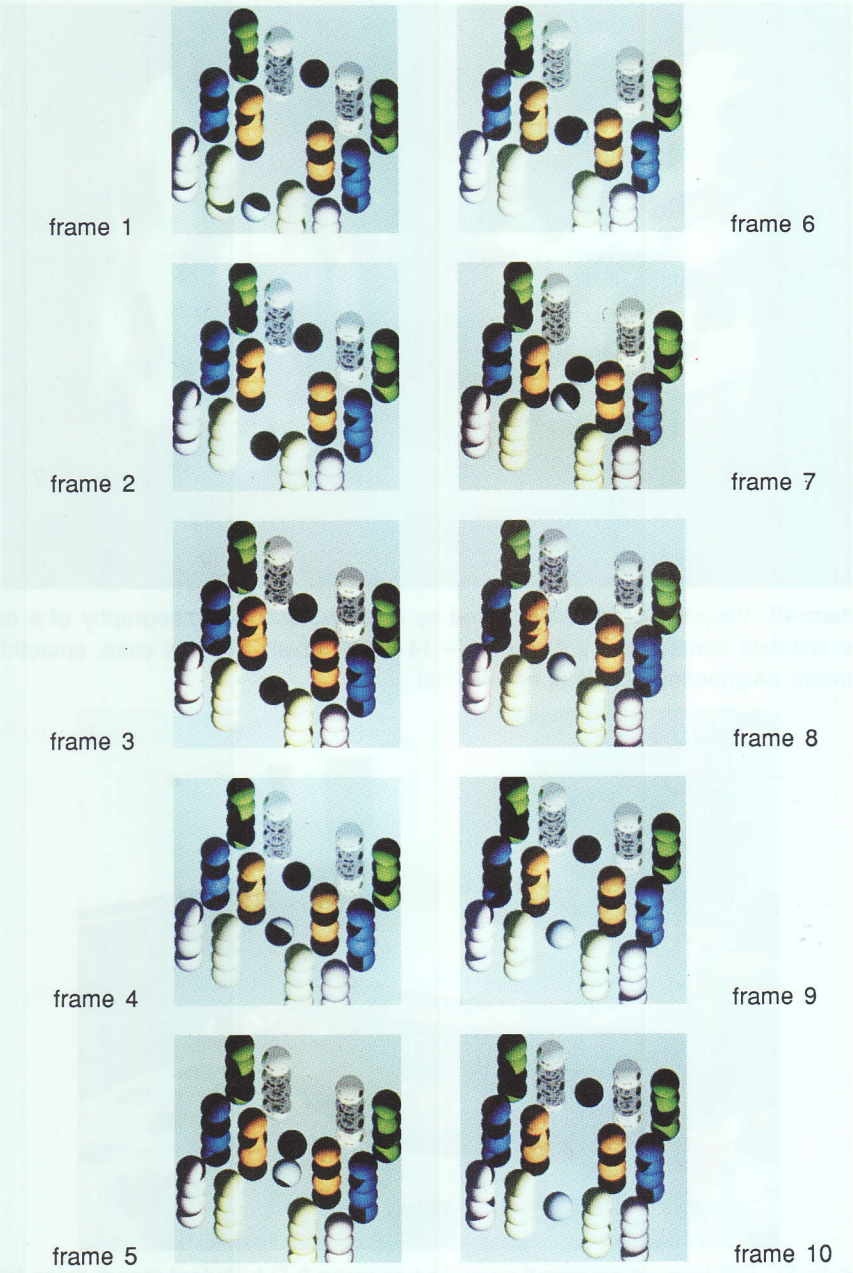


Figure 6: Example 1

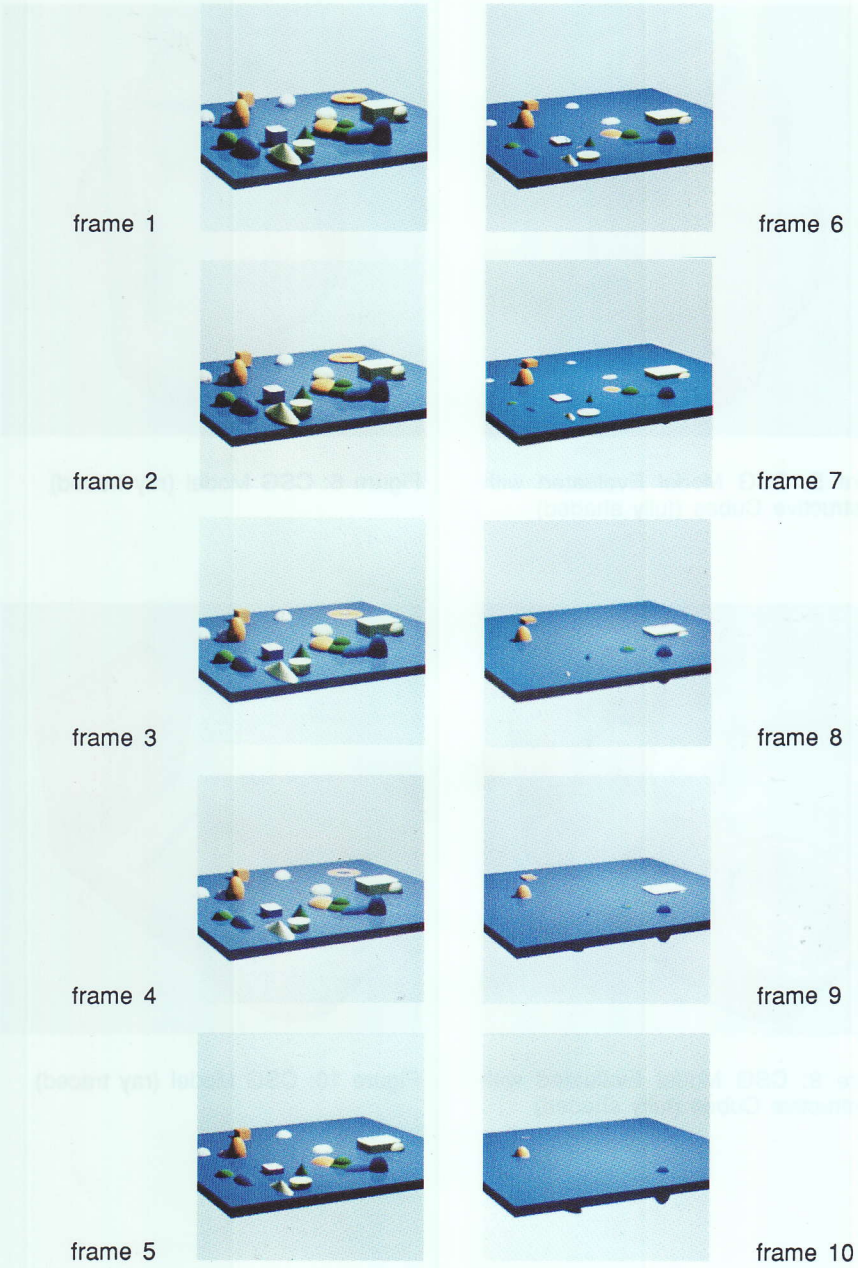


Figure 7: Example 2