AN EFFICIENT ALGORITHM FOR RAY CASTING OF CSG ANIMATION FRAMES

Andrea Sanna and **Paolo Montuschi**
Dipartimento di Automatica e Informatica,
Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino (Italy)
E-mail: sanna@polito.it

ABSTRACT

This paper presents a new algorithm to generate ray casted CSG animation frames. We consider sequences of frames where only the objects can move, in this way, we take advantage of the high screen area coherence of this kind of animation. A new definition of bounding box allows us to reduce the number of pixels to be computed for the frames after the first. We associate with each box a CSG sub-tree encapsulated and two flags denoting if the box has changed in the current frame and if it will change in the next frame, respectively. We show with two examples the advantages of our technique when compared with an algorithm which entirely renders each frame of an animation. For the test sequences the intersections with CSG objects may be reduced about up to one fifth, while the rendering may be computed up to four times faster.
Keywords: animation rendering, bounding box computation, constructive solid geometry.

1 INTRODUCTION

The computer animation is a new and very attractive research field and rendered animation is used into a broad spectrum of applications: video games, virtual reality, CAD, movies and so on.
The main drawback of most rendering algorithms is the high computation time required to render the images. In particular, this problem becomes crucial when complex rendering algorithms such as ray tracing [Glass89b] are used. On the other hand, shadow, reflection and refraction optical effects can be handled. One possible acceleration may come by tracing only the primary rays; this methodology is called ray casting and it has the drawback of being not able to handle reflection and refraction rays. Several methodologies have been proposed to speed up the ray tracing process of static images; a survey of these techniques can be found in a chapter written by Arvo and Kirk in [Glass89b].
The algorithms based on coherence properties are of particular interest to speed up the computation of animated sequences. The term coherence in computer graphics usually denotes similarities between items or entities and it may be used to describe properties locally constant [Foley90a] [Groll95a]. Several kinds of coherence have been defined, for instance, the frame coherence denotes that two successive frames are more likely similar if the "distance" in time between them is small. This and other properties can be used to speed up the ray tracing process of an animation; we review the main algorithms based on coherence properties in section 2.
Two main problems have to be addressed in the rendering process of an animation: the frame descriptions have to be computed and each frame of the sequence has to be rendered. In this paper we assume that the descriptions of the scenes have been already computed and are available to the rendering process; we focus our efforts to address the second issue. We propose a new ray casting algorithm for CSG animation frames. The Constructive Solid Geometry (CSG) method [Foley90a] describes an object by means of a binary tree where the leaves are primitive solids (spheres, cylinders and so on) and the nodes of the tree are operators (union, intersection and difference). The root of the CSG tree represents the whole scene. Our algorithm is based on the bounding box group theory presented in

[Sanna95a]; a new definition of bounding box allows us to reduce the number of pixels to be computed for the frames after the first.

Two new flags are associated with each primitive CSG object denoting if the object itself has changed in the current frame and if it will change in the next frame, respectively. These flags, called current and future flags, are inherited by the bounding box encapsulating the object at the leaf level and are used to find the pixels to be traced in each frame. In general, it is very likely that two frames close in time are similar; this is employed by compression algorithms such as MPEG which may provide high compression ratios with low loss of quality. Moreover, it has been proved in [Crock87a] that sequences where the changes between following frames affect only small portions of screen have a high screen area coherence; this is usually true when consecutive frames do not require a change in viewing position. We will show with two examples that this property may be employed to speed up a ray casting algorithm; the intersections with primitive CSG objects may be reduced to about one fifth with respect to the number performed with a ray casting algorithm which entirely renders all frames of a animation; the computation overall delay may be reduced by more than four times.

A survey of previous works is given in section 2. In section 3 we explain the basic idea of our technique and show in the details the steps of the algorithm as well as the definitions of the CSG operators. Results and remarks can be found in sections 4 and 5, respectively.

2 PREVIOUS WORK

Several techniques to speed up the rendering process are based on coherence properties. Two surveys of these methodologies can be found in [Foley90a] and [Groll95a]. Coherence properties allow us of reusing calculations made for one part of the environment or picture for other nearby parts, either without changes or with incremental changes that can be computed more efficiently than computations from scratch. Several kinds of coherence have been defined: spatial coherence, image coherence, ray coherence, temporal coherence, frame coherence and so on. *Spatial coherence* properties of the scenes are used to reduce the number of ray/object intersection tests. Two methods are used; the first one is based on hierarchies of bounding volumes while the second one on a recursive space subdivision [Glass89b]; the aim of these methodologies is to avoid expensive and unnecessary tests. *Image coherence* assumes that neighboring pixels may have very similar colors since just a few objects affect a small portion of image. This property is employed in [Amana84a] to reduce the number of objects to be considered for intersection with a cone traced from the observer. Moreover, it is very likely that rays with similar origin and direction have the same behavior; this similarity, called *ray coherence*, may be used to classify the rays. In [Arvo87a] and [Heckb84a] it is shown how the ray classification is used to speed up the ray tracing. For an effective rendering of an animation *temporal* and *frame coherence* are very important; they are an index of how two frames close in time are similar. These kinds of coherence are employed in [Badt88a] and [Green89a], respectively. Under the assumptions of constant viewing position and small changes between two following frames, it is likely that only small portions of screen have changed from the previous frame. Crocker exploits this property (*screen area coherence*) in modeling solid objects and proposes an efficient scan line algorithm for fast preview of CSG models [Crock87a]. The scan line is performed only in the part of screen where a change has been detected from the previous frame. In [Adels95a] a method to generate ray traced animation frames by re-projection is proposed. This approach is divided in three steps: re-projection of the points from the base frame to the inferred frame, verification of the re-projected points to check if they are illuminated or in shadow and finally, reflection, refraction and specular highlights phenomena are added to the inferred frame. It is worth to point out that several techniques based on coherence properties produce "not exact" images since they use some approximations in order to reduce the amount of computation.

An algorithm not based on coherence properties has been presented in [Glass88a], where static objects are rendered into a 4D space-time environment instead of rendering moving objects in 3D. A unique structure of space-time bounding entities can be computed before the rendering process starts. In this way, when a ray has to be tested for intersections, just the boxes belonging to the same region of space and time of the ray are considered. With the Glassner's approach just one hierarchy of boxes is computed instead of computing a hierarchy for each frame of the animation.

3 THE ALGORITHM

3.1 Basic Idea

The goal of this paper is to provide an efficient algorithm to exactly render animation frames modeled with the CSG technique by means of a ray

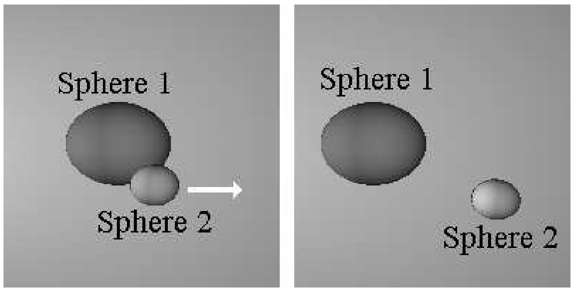

Figure 1: Sphere 2 is moving.

casting approach. The primary rays are traced to identify the objects visible from the observer, while the shadow rays are traced to determine if the points hit by the primary rays are either in shadow or illuminated. If we assume that only the objects can move, most of the area of the screen does not change from one frame to the next [Crock87a]. We introduce two new flags associated with each primitive CSG object and, hence, added also to the bounding boxes surrounding the objects themselves. These two flags denote if one or more attributes of an object (position, shape, color and so on) have "changed" in the current frame and if they will change in the next frame, respectively. Current and future flags can help us to identify the pixels to be traced in the frame under test. If an object has the current flag set to TRUE it means that at least one of its attributes has changed from the previous frame and therefore, the pixels affected by the object have to be retraced. The projection of the bounding box of the object allows us to identify the pixels to be considered and hence the flags are added to the bounding boxes, too. Moreover, if an object has the future flag set to TRUE, it means that at least one of the attributes of the object will change in the next frame; therefore, the pixels affected in the current frame will have to be retraced in the next frame. Let us consider for example the left part of Fig. 1; two spheres are depicted, where the first one is a static object while the second one is moving and in the next frame of the animation will be placed in the scene as shown in the right part of Fig. 1. Only a portion of the second frame should be retraced since the points of the object 1 not covered by the sphere 2 are unchanged. We can identify these points using the current and future flags. In this case, the future flag of the object 2 will be TRUE in the left part of Fig. 1, while the current flag of the same object will be TRUE in the right part. The problem is more complex to determine the shadows. If a point of an object is visible from the observer, we have to compute its illumination; in order to do this, we trace a ray starting from the point toward each light source (shadow rays). If we determine a plane of projection for each light (as if the lights were virtual observers) and we project the bounding boxes of the objects over these planes, we can find the changes seen by the light sources in the same way we find the changes seen by the observer on the plane of view. The projections of the bounding boxes on the planes found for the lights define several areas on these planes where the shadow rays are casted. The current and the future flags indicates if these areas have changed in the current frame and if they will change in the next frame, respectively. If a shadow ray is casted trough an area having the future flag set to TRUE, it means that the illumination of the point under test could change in the next frame. On the other hand, if we want to determine if the illumination of point belonging to a static object has changed in the current frame, we need to check, for each light, if the shadow rays for that point "falls" into areas of the planes of the lights having the current frame set to TRUE. When a shadow ray falls into an area having the current flag set to TRUE, the contribution of the considered light to the illumination of the point has to be recomputed. **??**. consider itself; frame

Since the areas on the plane of view and on light planes are found by projecting the bonding boxes instead of the objects themselves, it is possible that several points are retraced even if it is not necessary. In order to reduce this possibility, the bounding entities of the objects have to be computed as tightest as possible; to tackle this issue we improve and modify the bounding box group theory presented in [Sanna95a] since it provide the tightest encapsulation of the root of a CSG tree by starting from the bounding boxes of the leaves.

3.2 Definitions and symbols

In this section definitions and notation used in this paper are provided. The bounding box group theory was presented in [Sanna95a] and it was proved that a more effective encapsulation of a static CSG model may be obtained by relaxing the constraint of having just one bounding box at each node of the CSG tree. If a group of boxes can be handled at each CSG node, the encapsulation of the root (which is itself a group of boxes) is as tightest as possible. If we consider an CSG node and the bounding box group associated with it, we can say that a box belonging to the group encapsulates a part of the CSG sub-tree defined at the node itself. The definitions provided in this section take into account the new current and future flags.

Bounding box. We define a bounding box (bb) as a rectangular prism whose edges are parallel to the co-ordinate axes. A box consists of four different data: 1. corner points ($cp()$) 2. the CSG sub-tree encapsulated ($st()$) 3. the current change flag ($cf()$) 4. the future change flag ($ff()$). For instance, a bounding box a could be described as follows: $a = \{cp(a), st(a), cf(a), ff(a)\}$. The CSG sub-tree encapsulated at the leaf level is a primitive CSG object.

Inner box. An inner box is a prism which encapsulates only points belonging to primitive CSG objects.

Bounding box group. A group of bounding boxes (BBG) consist of an unordered list of bounding boxes; the bounding box groups are denoted with the upper cases, whereas the boxes belonging to a group are denoted with the lower cases. For instance, a bounding box group A consisting of n boxes could be described as follows: $A = \{bb_1, \dots, bb_n\}$. The bounding entities at each CSG node consist of two BBGs, a first group of bounding boxes and a second group of inner boxes, i.e. $\{BBG, IBG\}$.

Image-plane-bb. Given a bounding box, a projection plane and a center of projection, the image-plane-bb is the smallest rectangle on the projection plane that contains the perspective projection of the bounding box.

Symbols. The symbols $\cup$ and $\cap$ denote union and intersection operations, respectively; the symbol $-$ is used to denote difference operations. The subscript *in* is used to distinguish bounding boxes from inner boxes. The symbol *.or.* is used to indicate the logical or operation between two flags.

3.3 Basic operators

For each node of a CSG tree a BBG has to be built; this operation starts at the leaf level where each primitive CSG object is surrounded by a box (a box is a particular case of BBG). The boxes at the leaf level have to be composed according to the CSG operators in order to obtain a BBG for the root of the tree. In this section, we show how the basic CSG operators presented in [Sanna95a] are modified in order to consider the current and future flags.

Union: The union U of two BBGs $A = \{a_i, \dots, a_m\}$ and $B = \{b_j, \dots, b_n\}$ is simply obtained as linkage of the two lists $D = \{a_i, \dots, a_m, b_j, \dots, b_n\}$.

Intersection: The intersection I of two BBGs $A = \{a_i, \dots, a_m\}$ and $B = \{b_j, \dots, b_n\}$ is obtained by considering all the possible pairs having an element a from A and an element b from B. Each pair produces a bounding box having $(cp(a) \cap cp(b), st(a) \cap st(b), cf(a).or.cf(b), ff(a).or.ff(b))$. The result I is the set of these bounding boxes produced by each pair. The intersection between the corner points allows us to compute the 3D space shared by two elements of a pair and the intersection between the CSG sub-trees of the elements the portion of the whole CSG tree encapsulated by the resulting box. If the intersection between the corner points is the empty space there is no a resulting box.

Complement: Before to define the complement of a BBG is necessary define the complement of a single box. The complement of a bounding box a is, in general, a bounding box group such as its elements surround all the space not encapsulated by a. The complement of a can be denoted with $\overline{a}$; each element of $\overline{a}$ has the same CSG sub-tree as well as the same flags as a. At this point, the complement of a BBG is defined as:

- the complement of each element (bounding box) belonging to the group;
- the intersection of the groups obtained by the previous step.

The notion of complement allows us to define the difference operator.

Difference: The inner boxes have to be considered for the difference operation, too. Given two bounding entities $A = \{BBG_A, IBG_A\}$ and $B = \{BBG_B, IBG_B\}$, the difference $D = A - B$ can be computed as: $D = \{BBG_A, IBG_A\} * \overline{IBG_B}$. The operator $*$ is separately applied over the two groups of A: $\{BBG_A * \overline{IBG_B}, IBG_A * \overline{IBG_B}\}$. The operator $*$ applied over two BBGs $A = \{a_i, \dots, a_m\}$ and $B = \{b_j, \dots, b_n\}$ is obtained by considering all the possible pairs having an element a from A and an element b from B. Each pair produces a bounding box having $(cp(a) \cap cp(b), st(a) - st(b), cf(a).or.cf(b), ff(a).or.ff(b))$. The operators $*$ and $\cap$ differ since they subtract and intersect the CSG sub-trees, respectively.

3.4 Steps of the algorithm

We assume that the current and future flags are associated with each primitive CSG object in the frame descriptions. All the objects of the first frame have the current flag set to TRUE since they have to be entirely rendered in the first time. For each frame, the primitive objects are encapsulated into 3D bounding boxes; the current and future flags of an object are inherited by its box, therefore, the current flag indicates if the box of an object has changed in the current frame, while the future flag denotes if the box will change in the next frame. For each frame of an animation, the bounding box group of the root of the CSG tree is built with the rules shown in section 3.3; with this step a 3D group of boxes is obtained. The bounding boxes are projected on the plane of view using the observer as center of projection; a set of overlapping image-plane-bbs is obtained. The overlapping rectangles are split into non intersecting areas. In general, the decomposition of two overlapping rectangles leads to four new rectangles plus a rectangle obtained as intersection of the overlapping original rectangles. Each new area is associated with the CSG sub-tree of the original rectangle encapsulating it, while the intersection area is associated with the union of the two original CSG sub-trees. On the other hand, the flags of the new areas are equal to the ones of the original rectangles, while the flags of the intersection area are computed as logical OR operation between the original flags. This step produces a set of non overlapping rectangles of the observer.

For each light a plane of projection has to be found, then, the 3D boxes at the root level are projected using the lights as centers of projection; the same operations described above for the rectangles of the observer have to be also performed for the lights. After these operations we obtain a list of non overlapping rectangles for each light. Two matrices have to be defined (current and future flag matrices); the first one is used as current matrix (MAT1) while the second one (MAT2) as future matrix in the odd frames and viceversa in the even frames The size of the two matrices depends on the frame resolution since each entry is a bit denoting if the corresponding pixel has to be traced in the current and in the next frame, respectively. The two matrices are initialized setting all the elements to zero. At this point, the first frame is rendered with the following methodology:

1. MAT1 becomes the "current" matrix and MAT2 the "future" matrix. The rectangles of the observer having the current flag set to TRUE are considered and the corresponding locations in the current matrix are set to one. All rectangles of the first frame have the current flag set to TRUE.

2. MAT1 is analyzed and a ray, when an element is found to one, is traced through the corresponding pixel. The rectangle containing the pixel is searched in the list of the observer and the intersection test is performed only with CSG sub-tree associated with the rectangle itself. If the future flag of the rectangle is set to TRUE the corresponding locations in MAT2 are set to one. When an intersection point with the scene is found, the illumination has to be determined; a line starting from the intersection point and ending to each light source is computed. The intersection with the plane of projection of each light is found as well as the rectangle of each light enclosing this intersection. The shadow tests are performed by considering only the CSG sub-trees associated with the rectangles previously found. If one of these rectangles has the future flag set to TRUE, then, the pixel for which we are computing the intensity might "change" in the next frame, therefore, the corresponding entry in MAT2 has to be set to one. Moreover, to speed up the determination of the intersection points over the planes of projection of the lights, we define, for each light, a matrix of size depending on frame resolution (shadow matrix); each entry of this matrix is a pair of integer values representing the coordinates of intersection points on the plane of the light.

3. MAT1 and MAT2 are exchanged.

4. For the frames following the first we perform the same operations as described above. Moreover, when a rectangle of the observer has the current flag set to FALSE we analyze its pixels. For each pixel we consider the shadow matrix of each light and we check if the intersection point obtained by tracing a ray through the pixel "falls" into a rectangle of the light having the current flag set to TRUE. If either the result of this test is affirmative or the old position stored in the shadow matrix does not fall in any rectangle of a light, then, the location into the current flag matrix corresponding to the pixel under test is set to TRUE. If we trace a ray and we are not able to find a rectangle of the observer containing the corresponding pixel, then, the intensity of the pixel is set to background color.

4 EXAMPLES

In this section, we compare the performance of the proposed algorithm with an algorithm that entirely renders each frame of an animation. We have developed two software packages in C; the first one is based on the proposed algorithm (PA) while the second one is based on a classical ray casting approach (CA) which entirely renders each frame of an animation. CA may be thought as the same as PA when the first frame has to be rendered; the overhead to manage the current and future flag matrices as well as the shadow matrices is not introduced for CA. In order to be fair the same "rendering engine" has been used for both programs, that is, the same light model and the same intersection routines have been adopted.

Two sequences are shown:

1. The first example (ten frames) shows the modeling of a simple bike; this animation has been built with up to 56 primitive CSG objects and three

point light sources (Fig. 2).
2. In the second example (sixteen frames) three spheres strike several objects placed over three columns; this animation has been built with up to 42 primitive CSG objects and two point light sources (Fig. 3).
In Tables 1 and 2 we list the intersection tests with primitive CSG objects, the pixels analyzed and rendering times for the example 1 and 2, respectively. In the first column we report the number of the frame, while in the next three columns the statistics for the proposed algorithm (PA) are presented, that is, number of intersection tests, pixels analyzed and rendering time; the last three columns concern CA. The statistics have been obtained using a 133 MHz Pentium Pro with 64 Mbytes of RAM and each frame has been rendered with a resolution of 640x480 pixels in true color. Computation times include the delays to store the frames on disk.

5 REMARKS

A comparison between PA and CA shows that intersection tests and computation time can be reduced more than 3.5 times for example 1. A larger advantage is obtained for example 2 where less than 20% of the intersection tests have to be performed, thus leading to a reduction of 4.5 times of the rendering time
The overhead due to the management of flag and shadow matrices may be considered negligible for sufficiently complex scenes. The overhead can be observed by comparing the rendering times for the first frame; in the worst case (example 1) eleven seconds are required and they represent about the six per cent of the whole computation time for the first frame.
The performance of our algorithm decreases with the number of lights placed into the scene. When we evaluate the shadows, it is sufficient that just one light contribution has either changed or that it will change in the next frame to re-compute all light contributions. When the number of light sources grows up is more likely that the intensity has to be computed also for static points of the observer. This drawback could be overcome considering in a separate way the contribution of each light source, in this way, only a partial illumination might be re-computed for the points contained into rectangles of the observer having the current flag set to FALSE. The performance of our algorithm depends on the complexity of the considered animation since we obtain better speed up factors when the moving objects affect small parts of screen (i.e. when a high screen area coherence can be detected). Therefore, we may obtain better results for an animation consisting of a large number of small CSG primitive objects where just a few of them can move in each frame of the animation.
Our algorithm needs more memory than CA meanly because of the need to store the current and future flag matrices and the shadow matrix of each light. All matrices depend on the frame resolution; one bit is sufficient for each entry of the flag matrices, therefore, with a resolution of 640x480 pixels 75 kbytes of memory are needed for the two matrices. An entry of a shadow matrix is made of two integer values; if we assume that an integer is represented in memory with 4 bytes and we consider a resolution of 640x480 pixels, we need about 2 Mbytes for each light. If the memory requirement were too onerous, the matrices related to the lights could be eliminated at the cost of increasing the computation time. In fact, in this case the intersection tests between shadow rays and projection planes of the lights should be computed for all pixels of each frame.
It is worth to observe that our algorithm needs that a plane of projection could be found for each light source; for this reason, light sources can be placed only outside the objects. This could be a drawback for a ray tracing approach but not for ray casting applications since reflection and refraction rays are not handled.

6 CONCLUSION

In this paper we present a new algorithm to improve the ray casting of CSG animation frames; we tackle the problem to speed up sequences with constant point of view and moving objects. Our methodology aims to identify the pixels of a frame unchanged from the previous frame; in this way, a set of rays may be not retraced. This technique is based on a new definition of bounding box having two flags inherited by the surrounded object. These flags indicate if the object, and hence its box, has changed in the current frame and if it will change in the next frame, respectively. Our strategy is not constrained neither to objects with a specific shape or having specific motion laws nor to sequences having an high frame to frame coherence.
With practical examples we have shown the advantages of the proposed algorithm when it is compared with a methodology that entirely renders each frame of an animation. For the best example the computation time is reduced more than four times as well as the intersection tests with CSG primitives.

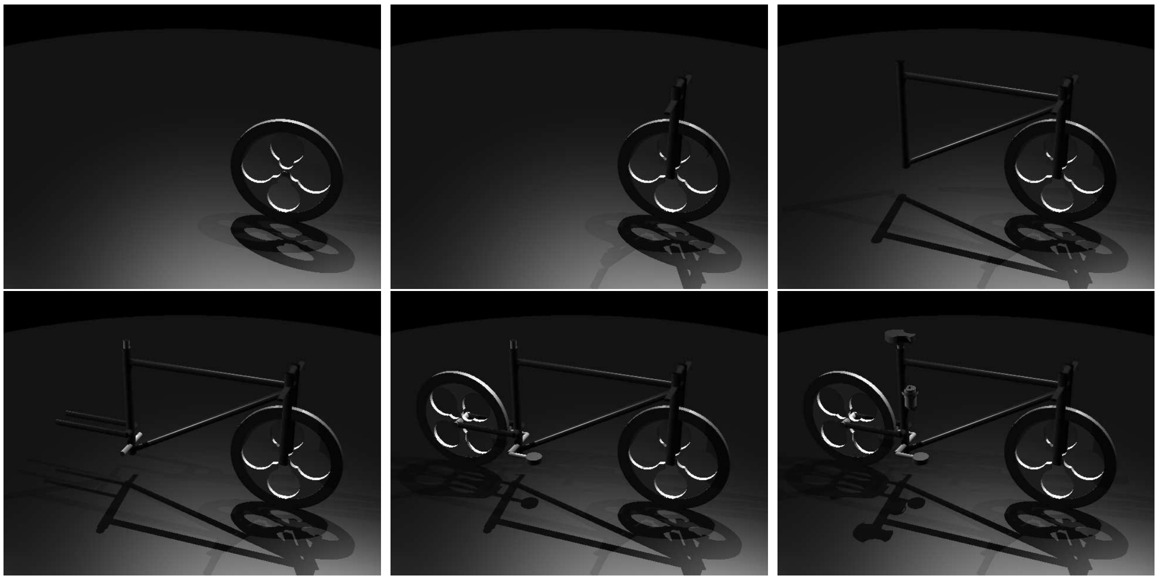

Figure 2: six out of ten frames of example 1.

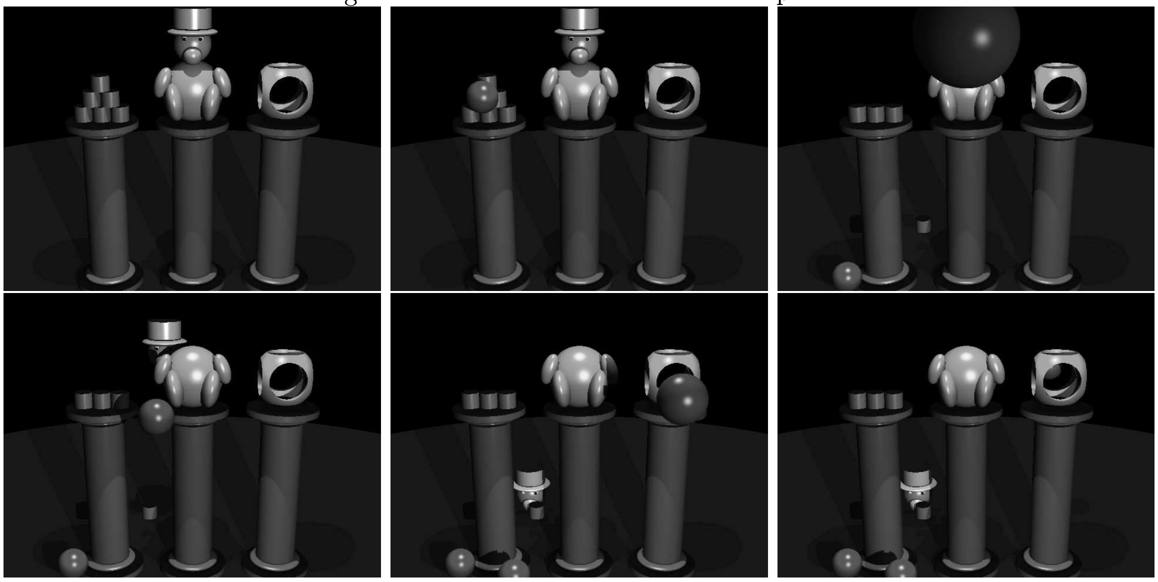

Figure 3: six out of sixteen frames of example 2.

Our methodology can be used whenever a good image quality and short computation times are required; for instance, when a fast preview of a ray traced animation is required or when an animation of CAD models has to be computed.

References

[Adels95a] S. J. Adelson and L. F. Hodges, *Generating Exact Ray-Traced Animation Frames by Reprojection, IEEE* Computer Graphics & Applications, May 1995, pp. 43-52.

[Amana84a] J. Amanatides, *Ray Tracing with Cones,* ACM Computer Graphics (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 129-135.

[Arvo87a] J. Arvo and D. Kirk, *Fast Ray Tracing by Ray Classification,* ACM Computer Graphics (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 196-205.

[Badt88a] Sig Badt Jr. *Two Algorithms for Taking Advantage of Temporal Coherence in Ray Tracing,* The Visual Computer, No. 4, 1988, pp. 123-132.

[Crock87a] G. A. Crocker, *Screen-Area Coherence for Interactive Scanline Display Algorithms, IEEE* Computer Graphics & Applications, Sep. 1987, pp. 10-17.

Table 1: Statistics for example 1.

Frame	PA Int. tests	PA Pixel Traced	PA time [sec]	CA Int. tests	CA Pixel Traced	CA time [sec]
1	2,494,606	307,200	173	2,494,606	304,125	162
2	705,529	23,406	41	2,624,071	304,125	169
3	560,149	19,124	33	2,658,333	304,125	171
4	2,306,736	149,340	144	3,097,541	304,125	196
5	482,002	62,465	41	3,171,344	304,125	205
6	1,452,019	58,273	87	4,267,696	304,125	263
7	707,314	37,922	48	4,285,156	304,125	269
8	477,432	16,575	31	4,336,464	304,125	278
9	149,339	10,312	15	4,354,992	304,125	285
10	105,998	5,574	11	4,368,000	304,125	290
Total	9,441,124	690,191	624	35,658,203	3,041,250	2288

Table 2: Statistics for example 2.

Frame	PA Int. tests	PA Pixel Traced	PA time [sec]	CA Int. tests	CA Pixel Traced	CA time [sec]
1	1,680,849	222,709	123	1,680,849	222,709	119
2	213,206	47,544	24	1,736,182	253,228	134
3	122,717	20,119	16	1,650,183	225,865	125
4	191,998	23,545	20	1,701,827	224,413	125
5	180,675	25,520	19	1,698,670	232,637	123
6	166,194	25,558	18	1,656,405	220,923	118
7	179,779	33,080	18	1,654,926	236,535	128
8	145,416	18,368	15	1,618,999	221,785	122
9	158,097	16,707	15	1,638,589	220,923	124
10	208,799	25,413	17	1,642,268	218,822	117
11	269,969	31,609	23	1,605,962	214,668	114
12	276,763	47,972	20	1,624,323	236,213	120
13	245,851	28,287	24	1,594,025	215,150	112
14	259,501	25,929	23	1,640,781	214,668	113
15	266,079	22,527	17	1,677,974	214,668	116
16	279,379	41,163	20	1,652,555	214,668	115
Total	4,845,272	656,050	412	26,465,518	3,587,875	1925

[Foley90a] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics Principles and Practice,* Addison-Wesley, 1990.

[Glass88a] A. S. Glassner, *Spacetime Ray Tracing for Animation, IEEE* Computer Graphics & Applications, March 1988, pp. 60-70.

[Glass89b] A. S. Glassner, *An Introduction to Ray Tracing,* Academic Press, 1989.

[Green89a] S. A. Green and D. J. Paddon, *Exploiting Coherence for Multiprocessor Ray Tracing, IEEE* Computer Graphics & Applications, Nov. 1989, pp. 12-26.

[Groll95a] E. Gröller and W. Purgathofer, *Coherence in Computer Graphics,* Technical Report, 1995, TR-186-2-95-04, Institute of Computer Graphics 186-2, Technical University of Vienna.

[Heckb84a] P. S. Heckbert and P. Hanrahan, *Beam tracing Polygonal Objects,* ACM Computer Graphics (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 119-127.

[Sanna95a] A. Sanna and P. Montuschi, *Spatial Bounding of Complex CSG objects,* IEE Proc.-Comput. Digit. Tech., Vol. 142, No. 6, Nov. 1995, pp. 431-439.