Interpolation of Synthetic Image Sequences Using Displacement Vector Fields

Heinrich Müller[*], Michael Stark[†]

Abstract

A new approach, the pixel interpolation algorithm, for interpolating intermediate frames of a computer animation from keyframes is presented. The rendering technique used is full ray tracing. From the keyframes and the spatial input scene, the movement of groups of pixels between the keyframes is estimated and documented in a displacement vector field. From the displacement vector field, regions of the intermediate frames are interpolated. Regions of the intermediate frames not covered by interpolation are postprocessed by ray tracing. The displacement vector field is calculated by an efficient stable plane sweep algorithm in linear space and nearly linear time. An implementation applied on several examples showed typical savings of time against rendering frame by frame between 29 and 43%.

[*]Institut für Informatik, Universität Freiburg

[†]Fakultät für Informatik, Universität Karlsruhe

1 Introduction

In computer animation, typically 24 or 25 frames have to be computed for one second of motion picture. In the photo-realistic case, the calculation usually is performed frame by frame using renderers for single pictures. This neglects the fact that consecutive frames often differ only slightly.

The occurrence of temporal coherence between frames may be analyzed on two levels. *Temporal coherence in object space* considers the input scene. It occurs if the objects change their shapes, location, or optical properties only little from one frame to the next. *Temporal coherence in image space* is based on the image. It happens for example if the color of a pixel in a frame is nearly the same as in its neighbored frames. Methods on both levels were developed in the past. They are briefly surveyed in section 2.

Our *pixel interpolation method* presented in the following is a hybrid approach. It is based on full ray tracing, including shadows, reflection, and refraction. The key idea is to estimate the motion of groups of pixels from one keyframe to the next. The keyframes are rendered applying conventional ray tracing. During rendering, additional information about the scene is produced for motion analysis. Based on the predicted motion, the location and the color of a pixel in the intermediate frames are calculated from its location and color in consecutive keyframes.

The motion of the pixels is described by a *displacement vector field*. The concept of displacement vector fields is well known in image sequence analysis [**?**] [**?**]. However, while we have full information about the scene, but incomplete information about the images, in image analysis a full sequence of pictures is given, but the scene is unknown.

The details of the pixel interpolation method are given in section 3. In section 4, the deficiencies that might arise in the interpolated frames are analyzed. Section 5 is devoted to an efficient algorithm for the calculation of displacement vector fields. Results are presented in section 6.

2 The pixel interpolation algorithm

The pixel interpolation algorithm derives intermediate frames of an image sequence from selected keyframes. The keyframes are rendered using ray tracing. However, beside a picture additional information about the keyframes is provided by the renderer. The additional information helps estimating the motion of groups of pixels between the keyframes. The estimation of movement is then used for calculating the position and the color of the pixels in the intermediate frames. A part of the object space is interpolated if it is visible in consecutive keyframes. So, finding corresponding pixels in keyframes and computing their motion depends on information about the object space which is provided by the renderer. Pixels in the immediate frame which cannot be interpolated from the keyframes are finally calculated by ray tracing.

In summary, the interpolation process consists of the following three tasks:

1. Calculation of the visible parts of spatial primitives in a keyframe and parts of them visible in consecutive keyframes.

2. Calculation of the motion of parts visible in consecutive keyframes with respect to the image plane leading to the motion of individual pixels.

3. Calculation of the position and the color of a pixel in the intermediate frames.

The pixel interpolation algorithm works for changes in spatial geometry as well as for changes of viewing parameters or illumination. The interpolated frames include all optical effects known from ray tracing, like shadows, reflection, refraction, and texture-mapping. In particular for texture mapping, the algorithm is quite favourable since ray tracing of large textured surfaces is in fact replaced by image transformations.

To simplify the description of the algorithm, we will first focus on the rays of view. Then the method is extended to full ray tracing.

2.1 Interpolation for the rays of view

2.1.1 Calculating the visible parts of surfaces

The first task of the algorithm is to determine the visible parts of the object space in the keyframes, and those visible in several keyframes. We suppose the object space composed by surfaces. Each surface has its own inner coordinate system. For example, the parameters u and v of a parametric surface $F : \Re^2 \to \Re^3, F : (u, v) \mapsto F(u, v), (u, v) \in I$, may be used as inner coordinates. For other surfaces inner coordinates can be introduced artificially. For triangles, barycentric coordinates may be applied.

The surfaces may change their location and shape over time. The only invariant required is the inner coordinate system. Let F be one of the surfaces in object space. Then F_t denotes the surface at time t. The rays traced form the viewpoint through the middle of each pixel of the frame at time t defines a mapping of the discrete image plane onto the inner coordinate system (Fig. 1). Technically, the mapping is performed in two steps. First, the discrete image plane is mapped onto the surface in 3d space by ray-surface intersection. Then the inner coordinates are calculated for each 3d intersection point.

Let be

$$S(F_t) := \{ \quad (i,j,u,v) \mid \\ F_t \text{ is visible in pixel } (i,j) \text{ and the inner coordinates} \\ \text{of the intersection point on } F_t \text{ are } (u,v)\}$$

the relational representation of the mapping. Drawing for each element in $S(F_t)$ a point with coordinates (u,v) in the u-v-plane and interconnecting them by line segments according to the pixel neighborhood supplied by i and j (a line segment is introduced if the difference of exactly one index is one) results in a grid-like structure. For this reason, $S(F_t)$ is called a grid. It is important to recall that every vertex of a grid owns two kinds of coordinates, image coordinates and coordinates with respect to the inner coordinate system of a surface.

2.1.2 Calculating the commonly visible parts of a surface

To determine the commonly visible parts of a surface at two key-frames, at times a and b, grids $S(F_a)$ and $S(F_b)$ are calculated. The vertices of grid $S(F_a)$ which are covered by grid $S(F_b)$ induce a subgrid $I(F_a,F_b)$ of $S(F_a)$ corresponding to a part of surface F visible at time a as well as at time b,

$$I(F_a,F_b) := \{ \quad (i,j,u,v) \mid \\ (i,j,u,v) \in S(F_a) \text{ and } (i,j,u,v) \text{ is covered} \\ \text{by four elements of} S(F_b)\}$$

A point belonging to the grid at time a is called covered by the grid at time b if and only if there exist four vertices of grid $S(F_a)$ with the following properties:

- the four vertices are connected to a quadrilateral by line segments introduced above,

- the pixel coordinates of the four vertices form a 2×2 square in the image plane,
- the point is located inside the quadrilateral in the inner coordinate system of the surface.

2.1.3 Determining the displacement of geometry

In addition to the fact that an element of $S(F_a)$ is covered by $S(F_b)$, the position of the point within the corresponding quadrilateral can be computed. This is achieved by defining an inner coordinate system for the quadrilateral (Fig. 2). The chosen coordinates refer to bilinear interpolation within polygons. The coordinates μ,ν of point q within the quadrilateral with vertices p_1, p_2, p_3, p_4 are obtained as a solution of the equation

$$q = (1-\mu)((1-\nu)p_1 + \nu p_3) + \mu((1-\nu)p_2 + \nu p_4), \quad \mu, \nu \in [0,1].$$

The vertices of the quadrilateral also have image coordinates. From them, the position (i', j') of a pixel at time b which had the image coordinates (i, j) at time a can be computed. This leads to the geometric displacement vector field

$$\begin{aligned} D(F_a, F_b) := \{ \quad & (i, j, i' + \mu, j' + \nu) \mid \\ & (i, j, u, v) \in I(F_a, F_b) \text{ and} (i, j) \text{ has the in-} \\ & \text{ner coordinates} (\mu, \nu) \text{ w.r.t. the quadrilate-} \\ & \text{ral with minimal image coordinates } (i', j') \}. \end{aligned}$$

A pixel at position (i, j) in the keyframe at a has moved to position $(i' + \mu, j' + \nu)$ in the keyframe at time b. This motion is represented by its displacement vector $(i' - i + \mu, j' - j + \nu)$.

2.1.4 Calculating the displacement of color

The color C_b of a pixel moving from (i, j) to $(i' + \mu, j' + \nu)$ at its final position is calculated by

$$C_b = (1 - \mu)((1 - \nu)C_1 + \nu C_3) + \mu((1 - \nu)C_2 + \nu C_4).$$

C_1, C_2, C_3 and C_4 are the colors at the vertices of the quadrilateral at (i', j').

The original color C_a and the final color C_b define the color displacement vector (C_a, C_b) of the pixel. The color displacement vectors of all pixels make up the color displacement vector field.

2.1.5 Interpolation of intermediate frames

The intermediate frames are obtained by interpolation of individual pixels. The position of a pixel in an intermediate frame is calculated from its displacement vector. The most simple case is linear interpolation (Fig. 3),

$$(i'', j'') := \lfloor (i, j) + \frac{t - a}{b - a} \cdot (i' - i + \mu, j' - j + \nu) \rfloor.$$

Here (i'', j'') is the location of pixel (i, j) in the keyframe of time t.

The color of the pixel in the intermediate frame is calculated from its color displacement vector,

$$C_t := C_a + \frac{t - a}{b - a} \cdot (C_b - C_a).$$

2.1.6 Postprocessing of the interpolated frame

From the explanation of the pixel interpolation algorithm so far it is evident that there may exist areas in an intermediate frame which cannot be computed by interpolation. An example is a rotating sphere for which parts hidden at time a are coming up at the silhouette. Although visible at time b and partly visible at intermediate times t these parts are excluded from interpolation. The color of these pixels is calculated by ray tracing. Since these pixels are particularly delicate to aliasing it is important to perform proper anti-aliasing, for example by stochastic sampling.

2.2 Interpolation for full ray tracing

For full ray tracing not only viewing rays are cast, but also rays of reflection and refraction. The rays of view hitting a reflecting surface induce rays of reflection cast from the surface. The rays of reflection usually will in turn hit different surfaces. The same holds for rays of refraction.

The immediate idea to collect all intersections of a surface and treat them as a single grid is not feasible. A surface can be visible at different parts of the image by following different chains of rays. For example, a part of a surface may be seen following one ray of reflection. Simultaneously, the part may be seen following a chain of rays consisting of a ray of refraction and a ray of reflection. Both situations may cause different displacement vector fields which cannot be calculated from a single grid. There is still a second reason for not merging intersections with different history. The angle between a ray and the surface normal has a strong influence on the color contributed by the ray. Points of intersection lying close together on an surface may have quite different colors if the angles for the corresponding rays are highly different due to different histories. For these reasons, the history of ray, that is the chain of rays by which an intersection point is reached, is used as an indicator to separate intersection points into different grids. By this approach, errors leading to wrong corresponding points of intersection are practically avoided.

The set of grids arising for full ray tracing are organized into trees, called *grid trees* (Fig. 4). Each node of a grid tree corresponds to a single grid. Its root is a grid induced on a surface by the rays of view. A child of a node is a grid induced by the rays cast from the part of surface defined by the grid of the node, induced by the history of the node.

2.2.1 Problems with high order grids

Despite paying attention to the history, higher order grids cause additional problems compared with the grids of rays of view. Rays of view start from a single point in different directions. So a point in 3d space cannot be hit by more than one of these rays. This does not hold for high order rays. For example, the same point may be visible in different mirrors. One consequence of this observation is that the mapping of the discrete image plane is not simple, but can be self-overlapping (Fig. 5). Self-overlapping grids lead to problems because the quadrilateral covering a vertex in the grid matching process need not be unique. Thus the displacement vector of the corresponding pixel cannot be predicted uniquely (Fig. 6). Therefore self-overlapping parts of a grid have to be detected and removed before grid matching is done. This results in a certain loss of information for interpolation.

2.2.2 Displacement trees

Displacement trees generalize displacement vector fields. Given two grid trees to be matched, grid tree matching is performed by recursively traversing them. During the traverse, corresponding grids are searched and displacement vector fields are calculated from them like in the previous section. The nodes of a displacement tree consist of these displacement vector fields (Fig. 7).

2.2.3 Generalized interpolation of intermediate frames

Using displacement trees, intermediate frames including all optical effects of ray tracing can be interpolated now. This is done by building up a *color tree* for every pixel of an intermediate frame. The color trees are built by interpolating displacement vectors belonging to the hierarchy of displacement vector fields available from the displacement tree.

An intermediate frame is computed by sequentially processing the displacement tree. To build the trees for an individual pixel the displacement tree is traversed recursively. For each node (which is a displacement vector field) the same interpolation is carried out like for the simple case. However, there is one important difference. Every color entry for a pixel has an identification where it has to be put in the color tree.

After processing a complete displacement tree the correctness of color trees of the pixels has to be checked. First, the pixel color must include all reflections and refractions determined by the properties of the surfaces which are directly or indirectly visible in the pixel. Second, no additional color contributions to a pixel are allowed. Third, only one color tree is allowed to exist for a pixel. The only exception to this rule is the background. Any pixel whose color is determined by the background may be later overwritten by a new color belonging to a surface. All pixels offending against these conditions are excluded from interpolation.

The color of a pixel of a non-defect pixel of the intermediate frame is calculated by summing up the colors of its color tree. This assumes that the terms of the contribution of the rays of reflection and refractions are additive in the model of illumination used.

3 Deficiencies of interpolation

3.1 Interpolation of motion

The pixel interpolation algorithm is a heuristic method which approximates the intermediate frames. It is based on two assumptions. Any violation of these assumptions may induce deficiencies in the intermediate frames.

The first assumption is that the pixel motion is closely approximated by the underlying interpolation function. For the current implementation the pixel motion is assumed to be linear with respect to the image plane. A violation of this assumption may cause two different effects. A surface may be deformed or move to a position in an intermediate frame which is not correct. Furthermore, disjoint visible parts of a surface may intersect in an intermediate frame.

The second assumption is continuity of visibility. Continuity of visibility means that the visibility of a part in the two keyframes induces its visibility in an intermediate frame too. A situation in which this assumption does not hold is presented in the following example (Fig. 8).

The two keyframes contain two spheres. One sphere moves from the left top to the right bottom, whereas the other moves from the right top to the left bottom. Furthermore the motion of the two spheres is assumed to be linear with respect to the image plane, without any rotation. So, the pixel interpolation algorithm detects that for each sphere the visible parts from the two keyframes match completely. In addition the linear motion is detected without errors. The interpolation of the intermediate frame ends up with both spheres put in the center of the intermediate frame. In contrast the correct intermediate frame would contain only one sphere according to the position in 3d space.

This example is not very severe since it is correctly treated by the pixel interplation algorithm. The reason is that the interpolation of more than one surface to a pixel is detected by checking the color trees as described in the last section.

However, there exist other situations leading to serious deficiencies of the intermediate frame. For example, let us suppose that the motion cannot be determined for surface A. This may happen if the part of A is hidden by another surface in one of the keyframes or if A rapidly moves in itself, for example like a rotating sphere. Let be P a pixel of an intermediate frame which should show A. Evidently, interpolation is unable to assign A to pixel P. This is not severe. But let us now suppose that a surface B usually hidden at P by A can be correctly interpolated at P. Then the pixel interpolation algorithm would give P the color of B, having no chance to detect any collision of surfaces.

3.2 Interpolation of color

For the following discussion, recall that a pixel is really an area in 3d space projected onto the image plane. For the interpolation of color, it is important to know whether the size of this area has changed. For example, suppose a surface has changed its size from one keyframe to the next (Fig. 9). The change of size leads to a different refinement of the two grids. Before the process of grid matching starts, it has to be decided which grid should be the covering one. The choice is to cover the points of the finer grid by quadrilaterals of the coarser grid. In this case, more points are covered and more displacement vectors can be computed. If later more than one contribution will fall into a pixel of an intermediate frame, the color of the pixel is calculated by averaging the contributions. This leads to a better approximation of the pixel color and the information included in the two grids is used most effectively.

Like for the interpolation of motion, it is assumed that the change of color is closely approximated by the interpolation function used. In the current implementation change of color is assumed to be linear too. Hence interpolation of color is certainly not allowed if the change of color is not induced by a continuous change of illumination. This constraint is fulfilled by ambient and diffuse illumination components, but not by specular reflections from light sources or for sharp shadows. For example, a pixel shows an area which is in the shadow of a light source in the first keyframe, but not in the second. Then it would be wrong to simply interpolate the pixel color in an intermediate frame. This would lead to a blur of the sharp shadow over time. The same holds for highlights caused by specular reflections from light sources.

4 Computing displacement vector fields by plane sweep

The main task to be performed in calculating the displacement vector field $D(F_a, F_b)$ of a surface F between two keyframes F_a and F_b is *grid matching*. Matching two grids A and B is to find for each vertex p of grid A a cell in grid B containing p. The grids are assumed to be simple, non-self-overlapping.

The grid matching problem is related to the problem of finding the intersection points between two sets of disjoint line segments in the plane. The algorithm of Mairson and Stolfi [**?**] performs this task in time $O(m \log m + k)$, m the number of line segments of the grids and k the number of intersecting edges. Guibas and Seidel [**?**] gave an algorithm for intersecting two convex partitions of the plane in time $O(m + k)$. Although these algorithms may be adapted on grid matching we will present a new algorithm in the following. It takes care of the specialities of grid matching thus becoming more simple and avoiding unnecessary overhead.

4.1 Planesweep

Like the algorithms mentioned above, our solution uses plane sweep too. In the simple version sufficient in our case, a scan line is swept in parallel over the given grids. We choose the line in parallel to the v-axis and the sweeping direction along the u-axis. The sweep is simulated discretely by moving the line from event to event. Our events are the vertices of the grids. Starting with the leftmost vertex, the grids are scanned vertex by vertex from left to right. A subset of future events comprising the next event at least, is held in a data structure of events. In fact we have two data structures of events, E_A for grid A and E_B for grid B, giving at least the next event on each of both grids. The grids themselves are stored in data structures G_A and G_B. G_A and G_B supply the information for updating E_A and E_B during the sweep.

For our purposes we need a scan line data structure S_B holding all edges of grid B intersected at the current event. If a vertex p of grid A is the event, the at most two edges of grid B neighbored to p are quickly found in S_B. They help to test whether p is covered by grid B, and if so to calculate the displacement vector at p.

The test of covering is somewhat intricated since the grids may have holes. Covering is checked by the row and column indices of the end points of the neighbored edges found in S_B. In the case of covering only two arrangements are possible, the angle arrangement and the stripe arrangement (Fig. 10).

In the angle arrangement, either the initial or the final vertices of both edges coincide. By symmetry, both cases can be treated the same way. A proper coverage in an angle arrangement requires the difference of row and the difference of column of the indices of the two non-identical points having one of the combinations (-1,1), (1,-1),(1,1),(-1,-1). The first entry concerns the row difference, the second one the column indices. Only for these combinations, a covering is possible.

For the stripe arrangement, the differences of row and column of the initial and the final points are required to have one of the following combinations, (1,0), (0,1), (-1,0), (0,-1).

If the neighbored line segments satisfy the condition, it has to be checked whether the two other edges also do exist. If so, vertex p is covered by a quadrilateral of grid B, and the displacement vector can be calculated next. It is obtained by calculating the coordinates μ, ν of vertex p within the quadrilateral.

4.2 Data structures

In order to define the operations on the grid data structures G_A, G_B, the concept of chains is required. A horizontal chain is a connected sequence of edges induced by fixing the j-index of the vertices. For a vertical chain, the i-index is fixed. The vertices with one index fixed may induce more than one chain, since grids may have holes or may consist of more than one connected component. For monotonous chain, the u-coordinates of its vertices depend monotonously increasing or decreasing on the free index. The most left vertex of a monotonous chain is called its initial point.

The grid data structures has to supply all monotonous chains, accessible by their initial vertices. Further, for a given vertex, its next vertex to the right on a horizontal or a vertical chain has to be found. It is implemented by a linked data structure. The elements correspond to the vertices. The vertices are doubly linked vertically and horizontally. The details of building the grid data structures are omitted here.

The data structures of events, E_A and E_B, have to support three operations, INSERT, MIN and MINDELETE. E_A is initialized by the initial vertices of the horizontal chains of grid A. E_B is initialized by the initial vertices of the horizontal *and* the vertical chains of grid B. During the sweep, it also holds the end points of the edges currently intersected by the sweep line. The next event is found by MIN. If it is not the final vertex of a chain, its successor in the chain has to be inserted by INSERT, and the event itself is deleted by MINDELETE. Evidently, E_A and E_B can be implemented by a heap.

The operations of the scan line data structure S_B are INSERT, FIND, UPDATE, SWAP, and DELETE. It is implemented by a doubly linked linear list. S_B stores the edges currently intersected by the scan line according to increasing coordinates of the points of intersection with the scan line.

From the explanation of the data structures of events, we know that the end vertices of the edges of grid A currently intersected by the scan line are stored in E_A. For the first, we suppose a reference from each end point to one of its at most two neighbors in S_B. If the next event is a vertex p' of grid A succeeding to one of the vertices p in E_A on a horizontal chain, FIND is performed as follows. The edges in S_B referred to by p are accessed by the references. If they are neighbors of p' too, we are ready. Otherwise, the linked list is searched sequentially, starting with these edges. It can be supposed that the neighbors are not far from the neighbors of p. Then the references to the neighbors of p' are set.

Like for E_A there are references from the vertices in E_B on the edges in S_B. If a final vertex of a chain is found in E_B the corresponding edge is marked deleted and is removed from S_B. In fact, the element containing the edge is not set free until there are no references from E_A to the edge. Therefore each element of S_B owns a reference counter which may only change after a FIND operation. If the reference counter of the edge is 0, the edge is explicitly removed from S_B. This is performed by DELETE.

If an inner vertex of a chain is found in E_B at which two chains are crossing, the old edges of the chains are replaced in S_B by the new ones, properly arranged. This is the SWAP operation of S_B.

At inner vertices without crossing, the current segment of chain has to be replaced by its successor. This is performed by the UPDATE operation.

There remains the case of initial vertices, in both grid A and grid B. Locating them by a doubly linked linear list only may become inefficient. For this reason, the edges in S_B are additionally linked according to a balanced binary tree T. The keys of T are the edges arranged according to the points of intersection with the scan line.

For an INSERT of an initial vertex in E_B, the neighbors are searched in T. Then the corresponding edge is inserted in S_B as well as in T.

For a FIND of an initial vertex p in E_A, or a vertex without reference on an edge in S_B, the neighbors are searched in T. Then the references are set onto the neighbors, and the counters are updated.

4.3 Summary of the events

Now we are able to present the complete list of events of the plane sweep algorithm for grid matching, and the actions to be performed at these events.

Grid A.

Initial vertex.

- FIND the neighbored edges of the vertex in S_B, if any.
- Perform the test of covering and calculate the displacement vector if successful.
- INSERT the successor of the initial vertex in its chain into E_A.

Final vertex.

- FIND the neighbored edges of the vertex in S_B, if any.
- Perform the test of covering and calculate the displacement vector if successful.

Inner vertex.

- FIND the neighbored edges of the vertex in S_B, if any.
- Perform the test of covering and calculate the displacement vector if successful.
- INSERT the successor of the inner vertex in its chain into E_A.

Grid B.

Initial vertex only.

- INSERT the incident edges into S_B.
- INSERT the successor of the initial vertex in its chain into E_B.

Final vertex only.

- DELETE the incident edges from S_B.

Inner vertex only.

- UPDATE the current edge on the chain in S_B.
- INSERT the successor of the inner vertex in its chain into E_B.

Inner vertex and initial resp. final vertex simultaneously.

- UPDATE the current edge on the chain in S_B.

- INSERT resp. DELETE the incident edges in S_B.
- INSERT the successor of the inner resp. initial vertex in its chain into E_B.

Double inner vertex (crossing vertex).

- SWAP the edges at the crossing vertex in S_B.
- INSERT the successors of the inner vertex in the chains into E_B.

4.4 Performance

The main difference between our algorithm and the other algorithms is that any line segment intersection is avoided. The only operation to be performed is the comparison of a vertex against an edge, whether it is above or below. Less arithmetic operations are required for this test than for intersection. Another advantage is that intersection points are avoided as events. It is known for plane sweep algorithms that intersection point events may cause some overhead to avoid problems due to numerical inaccuracies. Our solution instead tolerates some dearrangement of the data structures. It may happen that the combinatorial arrangement in the data structures does not coincide with the geometric arrangement of vertices of grid A relative to edges of grid B, documented by misplaced references from E_A into S_B. The reason is that an edge of grid A may cross edges of grid B between its initial and its final vertex. However, if a vertex of grid A is coming up, its references are checked for correctness and are updated if necessary. The amortized overhead for checking and updating is proportional to the number k of intersections between edges of both grids. The total time complexity of the plane sweep algorithm is thus bounded by $O((m' + m) \cdot \log m' + k)$, m the total number of edges in both grids, and m' the number of initial vertices to be preprocessed into the E_A and E_B data structures. The space requirement is $O(m)$ which is linear in the number of edges and thus optimal.

4.5 Self-overlapping grids

For higher order rays, grids may be self overlapping. By additionally checking the edges of the same grid neighbored in the current position of the scan line for intersection, self-overlapping grids can easily be recognized.

5 Measurements

The pixel interpolation method was implemented in C on a Unix-workstation Sun 3/50 with 8 Mb of main memory. Fig. 11 depicts the structure of the overall process. It consists of three main parts, grid tree calculation, grid tree matching, and interpolation with post-raytracing.

The grid trees are established for both keyframes, 1 and n, separately by ray tracing. The data delivered by ray tracing, that is a ray trace tree for every pixel, are combined in a sorting step into the data structures of the grid trees. The main task to be performed here is in fact sorting. Since the amount of data to be sorted is big compared with the main memory available in the environment used, sorting is performed externally on disk.

The implementation of grid tree matching is straightforward, However, due to lack of main memory, the sweep was carried out using secondary memory too. Here benefit is derived by the property of sweeping that the size of the data structure documenting the state of the sweep line is small compared with the whole data set to be processed.

The interpolation is carried out for each of the intermediate frames 2 to $n-1$ in separate.

The program was applied on three image sequences. The keyframes were given in a distance of five frames, from which the intermediate frames are derived by interpolation. The interpolated frames were compared with corresponding frames obtained by full raytracing, visually and with respect to time requirements. The resolution was 576×780.

The first example (Fig. 12) shows a static scene the camera is moving in. The scene contains several mirroring and refracting objects, as well as three light sources two of which are relevant for shadows.

The second example (Fig. 13) consists of a static background scene and a few moving objects. The camera is fixed. Practically the whole scene is mirroring. It contains two light sources relevant for shadows.

The third example (Fig. 14) consists of a static scene with some rapidly moving objects. Additionally, the camera is moving. Some of the dynamic objects change their surface properties, increasing the number of mirroring objects. In fact this violates the assumptions of the pixel interpolation methods. Nevertheless the test was performed to see the effects which may result. The scene contains two light sources both of which are relevant for shadows. Since the objects are moving very fast, the distance between the keyframes was diminished.

Fig. 15 to 17 depict typical displacement vector fields of these examples. The geometric displacement is lowest in example 1 and highest in example 3.

Table 18 summarizes the results of measuring. It consists in three parts. The first part shows the times required by the several phases. The computation time of grid matching shows quite favorable compared with the time required to calculate a keyframe. Also, the other times but that for post-processing are neglectable. The crucial point is post-processing. Although a large portion of the area of the immediate frames (86%, 66%, 54%, respectively, for examples 1,2,3) can be interpolated, the savings in computation time is not more than 22%, 29%, and 22%. The explanation is that just the areas excluded from interpolation, namely contours of primitives, are those at which ray tracing is most expensive. This is since less primitives can be excluded from intersection tests at these parts by the data structures of the ray tracing process (hierarchically nested grids in our renderer). Further, more rays are required there for anti-aliasing. The effect is documented in the third part of the table by the number of primary rays required per pixel. In example 1, for instance, 7.27 rays are traced in the middle for every pixel in the post-processing phase, but only 2.09 for every pixel of a keyframe.

The second part shows the storage requirements of the ray tracing trees and the grid trees. Although we used secondary memory in our implementation, the numbers are so that they fit into the main memory of better equipped todays work stations.

References

[1] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185, 1981.

[2] R. S. L.J. Guibas. Computing convolutions by reciprocal search. In *Proceedings 2nd ACM Symp. on Comput. Geometry*, pages 90–99, 1986.

[3] H. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In *NATO ASI Series, Vol. F40, Theoretical Foundations of Computer Graphics and CAD*, pages 307–325. Springer-Verlag, Berlin, 1988.

[4] H.-H. Nagel. Displacement vectors derived from second order intensity variations in image sequences. *Computer Vision, Graphics & Image Processing*, 21:85, 1983.

Figure 1: Mapping the discrete image plane into the local coordinate system of a surface in the scene.

Figure 2: Calculation of the position of a point within a quadrilateral.

Figure 3: Linear interpolation of an intermediate frame (middle) from two neighbored key frames.

Figure 4: A grid tree.

Figure 5: Self-overlapping grids.

Figure 6: Two quadrilaterals covering a same vertex.

Figure 7: Calculating a displacement tree.

Figure 8: Violation of continuity of visibility.

Figure 9: Interpolation of color.

Figure 10: The angle arrangement (a) and the stipe arrangement (b).

Figure 11: Overview over the pixel interpolation process.

Figure 12: A frame of example 1.

Figure 13: A frame of example 2.

Figure 14: A frame of example 3.

Figure 15: A displacement vector field of example 1.

Figure 16: A displacement vector field of example 2.

Figure 17: A displacement vector field of example 3.

	Expl. 1	Expl. 2	Expl. 3
computation time			
raytracing a keyframe	12 h 52'	8 h 49'	7 h 30'
additional cost for raytrees	0.7%	1.7%	0.7%
sorting raytrees	0.2%	1.2%	0.4%
grid matching	1.8%	5.4%	2.3%
interpolating an intermediate frame	1.3%	2.7%	2.4%
postprocessing	96%	89%	94.5%
$\sum$ intermediate frame	9 h 14'	5 h 05'	5 h 21'
savings for intermediate an frame	29%	43%	29%
total savings	22%	29%	22%
disk space			
sorted raytrees per keyframe	15.2 Mb	25.6 Mb	11.0 Mb
displacement vector fields	14.5 Mb	23.3 Mb	9.3 Mb
raytrace statistics			
primary rays in keyframe per pixel	937511 2.09	718019 1.6	910386 2.03
primary rays in postprocesing per pixel	451695 7.27	399205 2.61	608797 2.94
postprocessed pixels	14%	34%	46%

Figure 18: Measurements.