Automatic Tracking of Interactive Virtual Players by Cameras Using a Voronoi Freespace Representation

Martin Otten, Heinrich Müller

Forschungsbericht Nr. 807/2006
January 2006

Automatic Tracking of Interactive Virtual Players by Cameras Using a Voronoi Freespace Representation

Martin Otten, Heinrich Müller
Universität Dortmund
FB Informatik LS VII
D-44221 Dortmund
Germany
Tel.: +49 - 231 - 755 6134
Fax.: +49 - 231 - 755 6321
e-Mail: mueller@ls7.informatik.uni-dortmund.de

Forschungsbericht Nr. 807/2006
January 2006

Abstract

The problem of automatic camera control consists in continuously following a virtual player by a virtual camera in a virtual environment in order to show the player and its local environment in a suitable way. A particular challenge are interactively controlled players as they occur e.g. in 3D computer games. We present a data structure called Voronoi freespace representation (VFS-rep). The VFS-rep efficiently supports a class of camera control strategies based on local objective functions. Its main feature is the combination of a roadmap with a freespace representation. Besides collision avoidance and visibility estimation which use the freespace information, the roadmap of the VFS-rep helps if the camera has lost the player. In this case, the camera can move continuously along the roadmap to that branch of freespace to which the player disappeared. In this way, undesired discontinuous jumps of the camera to the new location of the player, which can be observed in games, become rare events, in particular in complex environments.

1 Introduction

The presentation of the views which are relevant for the players or observers is an important issue of computer games. We think of computer games which take place in a 3D virtual world[7, 16]. One or more players may interact with objects of the scene or with other players. For this purpose the real players are visually represented by virtual players which are controlled interactively by the behavior of the real player. Additionally to the players there might be observers who are not actively involved in the game but who are interested in the event. A well-known, net-based game of this type is *Half-Life*[18].

The visual link between the players or observers and the game is established by virtual cameras. The virtual cameras might be interactively controlled by the players or observers, or they might automatically present suitable views of the current configuration of the game. The latter is of interest for players who usually have to focus their attention on the game. Restricted net or server capacity, excluding simultaneous interactive access by all observers, might be another reason for automatic camera control.

One approach to having the relevant events of the games in view is to assign a virtual camera to the virtual player. The virtual camera follows the player automatically and yields views on the player and the player's current environment which fulfill prescribed requirements.

Technically, the problem of player tracking can be formalized as follows. Given are a scene S of obstacles, an online generated sequence P_i, $i = 0, \ldots,$ of locations of a player P, and a camera C. Wanted is a sequence of camera configurations C_i, $i = 0, \ldots,$ so that C_i is in the freespace of S and sees P_i in a suitable manner.

"Seeing in a suitable manner" is expressed by an objective or cost function c which depends on S, the location of P, and the location of C, and further parameters supplied by the *camera control engine* of the game. The further parameters in particular concern the view of C on P. But the distance of the camera path from the obstacles of the scene or the smoothness of the camera path might be influenced, too.

The main contribution of this paper is a data structure called *Voronoi freespace representation* (VFS-rep). A particular strength of VFS-rep is the possibility of calculating efficiently a freespace path between the current configuration of

the camera and the accompanied player efficiently. This is useful if visibility between the camera and the player is lost, or becomes minor. In this case, the camera can continuously follow the player along the freespace path in order to reach a new position with a better view. In this way, undesired discontinuous jumps of the camera to the new location of the player are rare events.

While calculation of the VFS-rep in a pre-processing step needs some time, the VFS-rep allows on-line tracking, including collision avoidance and visibility estimation, in real time and needs just a minor portion of the computing time of a typical game.

The following chapter 2 gives a survey on related work. Chapter 3 specifies the player tracking precisely, and outlines the basic approach on an example which is typical for a class of tracking approaches supported by the VFS-rep. Chapter 4 defines the VFS-rep and shows how it can be calculated efficiently. Chapter 5 presents basic algorithms of tracking, in particular concerning distance and visibility calculation, and shows how the VFS-rep can be used to minimize the objective function which is the core of the concept of player tracking. Chapter 6 compiles data of an empirical analysis of the presented solution. Chapter 7 concludes the paper.

2 Related Work

Halper et al.[11] give a good survey on the state-of-the-art of camera control in computer games, to which we refer instead of a recapitulation. They also work out the difference of the requirements on camera control in games to camera control in cinematography and computer animation. A particular difference of games is that the scene is influenced interactively so that a perfect planning in advance is not possible. Just estimated predictions on the behavior of e.g. the players might be used in order to optimize the camera behavior.

There are three types of data commonly used by camera control, which have, according to this difference, to be calculated on-line in real time: data about the freespace around the camera, for example the distance of the camera to the obstacles of the scene, information about visibility, in particular concerning the visibility between the camera and the player, and additional constraints which restrict the motion path of the camera.

In the approach by Halper et al.[11], the freespace and visibility are determined

on-line by using the possibilities of rendering libraries like OpenGL and the related graphics hardware. Constraints are defined by augmenting the given scene on-line by additional geometry not visible to the user.

This approach does not use a preprocessed explicit data structure concerning freespace, visibility, and constraints. Those data are determined ”on the fly”. This requires considerable computing power for more complex scenes. The computing resources are taken from the graphics hardware and thus might reduce the possibilities available for rendering. As an alternative for complex or large scenes, preprocessing-based approaches to visibility calculation known from interactive walkthroughs in virtual realities might be applied in order to diminish this problem[17].

A problem related to camera control is motion planning in robotics. A standard formulation of the problem of motion planning is: move a robot from a given start configuration into a desired goal configuration without collision with the surrounding scene. A quite general relation to camera control is that a collision-free path has to be found here, too.

Many solutions concern the version of static obstacles, that is the scene is static and just the robot is dynamic. In many games, the situation is quite similar: the environment is static and just the players change their positions. Approaches to motion planning in static scenes often consists of two phases: a *preprocessing phase* in which the scene is preprocessed in order to allow an efficient execution of the second phase, *path finding*. A good introduction into this topic is given by Latombe[13].

A first approach is to represent the complete freespace by a union of cells (regular/non-regular, adaptive/non-adaptive, hierarchical/non-hierarchical). A further possibility is to augment the cells with information about the distance to the closest obstacles. This leads to distance fields[9]. The cells define the vertices of a graph which are connected by an edge if the corresponding cells are neighboring. The second phase consists in finding a path in the graph from the cell of the source configuration to the cell of the goal configuration.

Another method are potential fields. In this case, the obstacles of the scene get repulsive fields, whereas the goal gets an attracting field. The superposition of the fields yields a force vector at every location in freespace. The robot reaches the goal configuration by following the force vectors.

Sometimes the freespace is reduced to a subspace or roadmap. Examples are

the medial axis and the visibility graph. The medial axis consists of all points of the freespace which have equal distance to at least two different points on obstacles. The visibility graph connects pairs of sampling points on the surface by an edge if they see each other. The medial axis and the visibility graph again define graphs which are traversed in the path finding phase.

These methods can be applied to camera control, too, in order to avoid collisions of the camera or to test for visibility. Drucker[5, 6] combines a visibility graph for global planning with some sort of potential approach for local planning. Li et al.[14] augment a rasterized cell representation of freespace by visibility information stored at every cell for rasterized viewing directions.

Our approach also is inspired by the methods of robot motion planning. It extends the medial axis roadmap to a representation of the complete freespace by a covering by cells. The advantage is that a basic roadmap for the camera path is available which helps if the view on the player gets lost. In this case, the camera can move along the roadmap to that branch of freespace to which the player disappeared. However, because of requirements on the view of the camera on the player, the medial axis as a roadmap is too restricted. In order to choose a suitable location, the information about freespace can be used. The information about freespace is also useful in order to calculate the visibility of the player with respect to the camera, and to avoid collisions of the camera with the scene.

3 Requirements and basic approach

Let us first recall the version of the camera problem treated in the following. The input consists of a scene S of obstacles, an online generated sequence P_i, $i = 0, ...,$ of locations of a player P, and a camera C. The output is an on-line real-time-generated sequence of camera configurations C_i so that C_i is in the freespace of S and sees P_i in a suitable manner.

The camera C is represented by a 3D orthogonal frame $\{\mathbf{o}_C, \mathbf{x}_C, \mathbf{y}_C, \mathbf{z}_C\}$ in the world coordinate system of S (Figure 1). $\mathbf{o}_C$ is the origin which serves as viewpoint. $\mathbf{x}_C, \mathbf{y}_C, \mathbf{z}_C$ are mutually orthogonal vectors among which $\mathbf{z}_C$ defines the view axis, and $\mathbf{x}_C, \mathbf{y}_C$ span the image plane.

The player P is represented by an orthogonal frame $\{\mathbf{o}_P, \mathbf{x}_P, \mathbf{y}_P, \mathbf{z}_P\}$ in space, for instance $\mathbf{o}_P$ as the center for the head, $\mathbf{z}_P$ as vector from $\mathbf{o}_P$ towards the face, $\mathbf{y}_P$ as a vector from $\mathbf{o}_P$ towards the top of the head.

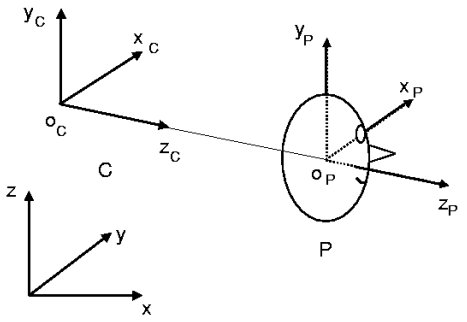

Figure 1: Definition of an ideal view of the camera on the player.

A reasonable heuristics for the camera view is to hold $\mathbf{z}_C$, $\mathbf{z}_P$, and $\mathbf{o}_P - \mathbf{o}_C$ co-linear and equally directed during the motion. In this way, the camera follows the head of the user from behind. Furthermore, the vector $\mathbf{u}_C$ is always perpendicular to the z-axis of the world coordinate system of S. The direction of $\mathbf{v}_C$ is chosen so that $\mathbf{v}_C$ has a positive component in direction of the z-axis of the coordinate system of S.

The relation between two consecutive camera configurations C and C^+, here $C := C_i$ and $C^+ := C_{i+1}$, is described by a transformation T, that is $C^+ = T(C)$. "Seeing in a suitable manner" is expressed by an objective or cost function c which depends on S, the next configuration P^+ of the player, the current configuration C of the camera, the unknown new configuration $C^+ = T(C)$ of the camera corresponding to P^+, and further parameters **c** supplied by the camera control engine. The parameters **c** allows the control engine to influence the camera behavior globaly. **c** comprehends for instance the desired distance between the camera and the player which might be changed dependent on the current location.

The goal of optimization is to find a transformation T_{opt} which minimizes c,

that is

$$T_{\text{opt}} = \arg\min_T c(S, P, P^+, C, T(C), \mathbf{c})$$

where the minimum is taken over all feasible transformations T. A transformation is feasible if the freespace constraint is satisfied, that is, $T(C)$ is in the freespace of S.

In the following simple example of a cost or objetive function, we assume T to be a rigid motion. The internal camera paramters defining the perspective mapping are held constant. The sample cost function consists of several components,

$$c(S, P, P^+, C, T(C), \mathbf{c}) :=$$

$$c_{\text{vis}}(S, P^+, T(C)) \cdot (c_{\text{dist}}(S, P^+, T(C)) +$$

$$c_{\text{angle}}(S, P^+, T(C)) + c_{\Delta\text{dist}}(S, C, T(C)) +$$

$$c_{\Delta\text{angle}}(C, T(C)) + c_{\text{safe}}(S, P, P^+, C, T(C)))$$

where

$c_{\text{vis}}(S, P^+, T(C)) = 1/\gamma$ where γ is the opening angle of a maximum view-cone at $\mathbf{c}^+$ with axis in direction of $\mathbf{o}_P^+ - \mathbf{o}_C^+$, so that no obstacle of S is between C and P in the cone. If $\mathbf{o}_P^+$ is not visible from $\mathbf{o}_C^+$, then γ is set to 0.

$c_{\text{dist}}(P^+, T(C), d_{\text{opt}}) =$ the difference of the desired distance d_{opt} between the camera and $\mathbf{o}_P^+$ and the actual distance $||\mathbf{o}_C^+ - \mathbf{o}_P^+||$.

$c_{\text{angle}}(P^+, T(C)) =$ the absolute angle between the desired direction $\mathbf{z}_P^+$ of the camera view on $\mathbf{o}_P^+$ and the actual view direction $\mathbf{o}_C^+ - \mathbf{o}_P^+$.

$c_{\Delta\text{dist}}(C, T(C)) =$ the absolute difference between $||\mathbf{o}_C{}^+ - \mathbf{o}_C||$ and $||\mathbf{o}_C - \mathbf{o}_C{}^-||$ where $\mathbf{o}_C^-$ is the camera location preceding $\mathbf{o}_C$.

$c_{\Delta\text{angle}}(C, T(C)) =$ the absolute angle between the vectors $\mathbf{o}_C{}^+ - \mathbf{o}_C$ and $\mathbf{o}_C - \mathbf{o}_C{}^-$ where $\mathbf{c}_C^-$ is the camera location preceding $\mathbf{c}_C$.

$c_{\text{safe}}(S, C, T(C), w) = w(||\mathbf{o}_C{}^+ - \mathbf{o}_C||)/d_{\text{free}}(\mathbf{o}_C{}^+)$ where $w(.)$ is a monotonous function controllable by the camera engine.

The model behind c_{safe} is that the faster the camera moves, expressed by $w(||\mathbf{o}_C{}^+ - \mathbf{o}_C||)$, the higher the distance $d_{\text{free}}(\mathbf{o})^+$ to the obstacles of the scene should be. The other functions demand that the player should be visible to the camera (c_{vis}), that the camera should hold a given distance and orientation to the player (c_{dist},c_{angle}), and that the camera has a certain inertia ($c_{\triangle\text{dist}}$, $c_{\triangle\text{angle}}$).

By applying the definition of the camera view given above, just the translational component of T remains as open parameter which can be used for minimization.

4 Calculation of the Voronoi freespace representation (VFS-rep)

The problem of calculation of the Voronoi freespace representation (VFS-rep) is to find, for a given 3D-scene S of polygonal obstacles, enveloped in a bounding volume, a Voronoi freespace representation of the freespace of S inside the bounding volume.

We solve the problem in three steps. The first two steps are *sampling of the obstacles of* S and *calculation of the freespace* on the resulting set of data points. The third step, *data reduction*, is not oblige, but usually improves the space and time requirements of the tracking phase significantly.

Before we start with the description of the solution, we recall briefly the definition of Voronoi diagrams – more details can be found e.g. in the survey by Aurenhammer[1] and the books by Edelsbrunner[8] or de Berg et al.[4]. Given a finite set of disjoint sites in d-dimensional Euclidean space, the *Voronoi region* of a site **s** is the region of all points in space being closer to **s** than to every other site. The *Voronoi diagram* is the decomposition of the space into Voronoi cells.

Figure 2, top, shows the Voronoi diagram in the case of 2D-points as sites. In the case of points, the Voronoi cells are convex polytopes (convex polygons in the plane). Their vertices are called *Voronoi points*, their edges *Voronoi edges*, and their faces *Voronoi faces*. Evidently, the boundaries of the Voronoi cells have maximal distance to the sites and coincide with the medial axis of the freespace between the sites.

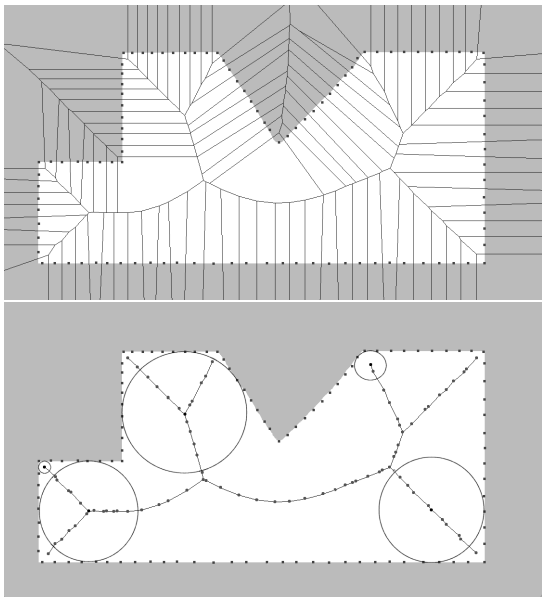

Figure 2: Approximate calculation of a medial axis of a polygonal scene, illustrated on a 2D-example. The first picture shows the sampling points and the resulting Voronoi diagram. The second pictures shows an approximation of the medial axis obtained by removing the Voronoi edges induced by closely neighbored sampling points. The white area indicates the region of interest inside a bounding volume which corresponds to the polygon. The medial axis outside this region is omitted. The circles indicate the property of the points on the medial axis to have equal distance to at least two sampling points.

4.1 Sampling of the scene

The goal of sampling is to replace the scene S of polygonal obstacles with a set of point obstacles P. In this way, the curved boundaries of the Voronoi cells are approximated by piecewise flat boundaries, as shown in figure 2, bottom. The advantage is that algorithms for point Voronoi diagrams are much easier to implement. A disadvantage is that a good approximation needs a large number of sampling points. However, as we will see, the efficiency of algorithms for

point Voronoi diagrams and the memory resources of today's PC make this approach practical also in three dimensions. The approach of approximation of Voronoi diagrams by point sampling has also been used by e.g. Geiger et al. [10].

A side effect of the sampling approach is that it harmonizes perfectly with point-based modeling[15]. If the scene is already represented by a cloud of points, the sampling step is not necessary. In particular, scene geometry acquired by highly resolved point-based 3D-scanners[3] may be used without surface reconstruction.

The point sampling has to satisfy the following

Sampling condition. Let d_s be a function on S which assigns a nonnegative sampling distance to every point on S. The sampling condition is satisfied if any point $\mathbf{q}$ on S has a neighboring sampling point of distance less than $d_s(\mathbf{q})$.

A particular type of sampling which exemplifies this general definition is *uniform sampling*. For uniform sampling, $d_s(\mathbf{q}) := d_0$ for all $\mathbf{q} \in S$ where $d_0 > 0$ is a constant. This reduces the sampling condition to the requirement that any point $\mathbf{q} \in S$ has a neighboring sampling point $\mathbf{p} \in S$ with distance less than the given bound $d_0 > 0$.

The bound d_0 defines a tolerance which has to be fulfilled in order that a region in space is considered as interesting freespace. This means that small openings or environments of concave corners are ignored. The value of d_0 defines the amount of tolerance. A possible choice is to make d_0 dependent on the size of the player.

Another example is *medial-axis adaptive sampling*. Medial-axis adaptive sampling is defined by $d_s(\mathbf{q}) := \max\{c_0 \cdot d_m(\mathbf{q}), d_0\}$ where $d_m(\mathbf{q})$ is the distance of $\mathbf{q}$ from the medial axis of S. and $0 < c_0 < 1$ and $d_0 > 0$ are given constants. d_0 plays the same role as for uniform sampling. The density of the sampling points is dependent on the distance from the medial axis, and thus is dependent on the extension of freespace in the environment of a surface region. If the freespace is narrow, the sampling density is high, and if there is much space, the points are sampled at a low density.

A difficulty with medial-axis adaptive sampling is that the medial axis usually is not known in advance. It is an interesting open question beyond the scope of

this paper to work out this sampling approach possibly based on cost-efficient estimates of the medial axis. In this paper, and in our implementation, we have used uniform sampling.

The triangles t of the input scene S are sampled independently as follows. First those edges of t of length $\geq 2d_0$ are subdivided. Then t is triangulated so that the subdivision points are included in the resulting triangulation. The procedure is iterated on the resulting triangles. The vertices of the resulting triangulation are the desired sampling points.

4.2 Calculation of the freespace

The VFS-rep consists of a spatial triangular mesh $V(S)$. $V(S)$ is an approximation of the medial axis of the original scene S, that is, of the surfaces of the Voronoi diagram of S which separate the Voronoi cells. The triangular mesh $V(S)$ needs not to be manifold, that is, edges with more than two incident triangles exist. In the 2D-analogue of figure 2, bottom, the polygonal chain of the medial axis corresponds to the triangular mesh in space, and the branching points of the medial axis correspond to non-manifold edges in space. $V(S)$ results by triangulation from Voronoi faces of the point-based Voronoi diagram which are specified later. For that reason we call these triangles *Voronoi triangles.*

Each vertex $\mathbf{v}_i$ of a Voronoi triangle v refers to a so-called *free distance value* $d_f(\mathbf{v}_i)$, $i = 0, 1, 2$. By barycentric interpolation, a distance value $d_f(\mathbf{v}) := \sum_{i=0}^{2} \mu_i \cdot d_f(\mathbf{v}_i)$ can be assigned to every point $\mathbf{v}$ of v, where the μ_i, $i = 0, 1, 2$, are obtained by resolving the equation $\mathbf{v} = \sum_{i=0}^{2} \mu_i \cdot \mathbf{v}_i$. The freespace $S_f(v)$ of v is defined as the union of all balls with center $\mathbf{m}$ on v and radius $d_f(\mathbf{m})$. Hence the free distance values have to satisfy the constraint that $S_f(v)$ has an empty intersection with scene S.

In the data structure of a VFS-rep, every Voronoi triangle refers to its three Voronoi vertices. Every Voronoi vertex refers to a list of its incident Voronoi triangles. The VFS-rep is calculated as follows.

1. Calculate the Voronoi diagram of the sampled scene of obstacles, including the input point of minimum distance of every Voronoi point

2. Remove of all Voronoi faces whose generating sampling points $\mathbf{p}_1$, $\mathbf{p}_2$

satisfy $d(\mathbf{p}_1, \mathbf{p}_2) < 2 \cdot d_0$, or which are outside of the bounding volume of the scene.

3. Triangulate the remaining Voronoi faces into Voronoi triangles.

4. Assign a free distance value d_f to every Voronoi vertex.

For the calculation of Voronoi diagrams in step 1, several efficient algorithms are known[8, 1, 4] The approach we use in our implementation is to lift the input points onto a hyper-paraboloid in 4D-space. The "bottom part" of the convex hull of the resulting points yields a Delaunay triangulation of the input points. The dual graph of the Delaunay triangulations is the Voronoi diagram. The convex hull can e.g. be calculated using the QHULL software[2].

In step 2, the generating points $\mathbf{p}_1$ and $\mathbf{p}_2$ of a face are the input points whose Voronoi regions share this face. The idea behind this choice of Voronoi faces to be removed is that faces induced by points $\mathbf{p}_1$ and $\mathbf{p}_2$ of distance $\geq 2d_0$ do not contain points on surfaces of S. The reason is that all points on such faces have a distance $> d_0$ to all sampling points, since $\mathbf{p}_1$ and $\mathbf{p}_2$ are closest sampling points by definition. According to the sampling condition, points without sampling point within distance d_0 are not on S.

On the other hand, it happens that Voronoi faces or parts thereof in freespace are lost. These faces, however, should usually be very close to the surface of S and thus are not relevant in our application.

The Voronoi faces are plane polygons. Thus the triangulation in step 3 can be performed straightforwardly. We have used the Delaunay triangulation[8, 4] which yields triangulations of the Voronoi faces since they are convex. An advantage of the Delaunay triangulation is that it should avoid "thin' triangles. This, however, is not necessarily correct for triangles at the boundary of the triangulation, and other approaches to triangulation might be used instead.

The free distance value $d_f(\mathbf{v})$ of step 4 is chosen so that the convex hull of the spheres of radius $d_f(\mathbf{v}_i)$ at the vertices $\mathbf{v}_i$ of a Voronoi triangle v, $i = 0, 1, 2$, is a subset of the freespace of S. For the case of uniform sampling, this constraint is satisfied by $d_f(\mathbf{v}_i) = ||\mathbf{p}_2 - \mathbf{p}_1||/2 - d_0$, $i = 0, 1, 2$, where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the sampling points inducing the Vononoi face from which v has emerged. The term $||\mathbf{p}_2 - \mathbf{p}_1||/2$ comes from the fact that $\mathbf{p}_1$ and $\mathbf{p}_2$ belong to the closest sampling points of every point $\mathbf{v}$ on v, and that $||\mathbf{v} - \mathbf{p}_1|| = ||\mathbf{v} - \mathbf{p}_2|| \geq ||\mathbf{p}_2 - \mathbf{p}_1||/2$. Hence this term yields an empty intersection with the sampling

points. d_0 is subtracted in the definition of $d_f(\mathbf{v}_i)$ in order that the resulting freespace $S_f(v)$ does not intersect S. If S would intersect, there would be a point on S of distance $\geq d_0$ to every sampling point, in contradiction to the sampling condition.

The definition of $d_f(\mathbf{v}_i)$ yields an approximation of the freespace by sets $S_f(v)$ of constant thickness each. This means that the function d_f is discontinuous. Furthermore, several values $d_f(\mathbf{v})$ need to be stored at every Voronoi point **v**, one for each incident triangle. A continuous d_f can be obtained by assigning the minimum of the values to **v**. Although this shrinks the freespace somewhat, we have used this representation. It has turned out that it is sufficient for our purpose.

4.3 Reduction of the Voronoi freespace representation

Uniform sampling of sufficient density causes a considerable number of triangles of the mesh $V(S)$ of the VFS-rep. The sampling density is chosen according to the requirements of the most narrow interesting regions of the freespace, but in this way it often exceeds the requirements of large regions of free space. In large regions, a lower number of larger Voronoi triangles would be sufficient. We achieve this goal by mesh reduction.

We reduce $V(S)$ by vertex elimination according to an approach inspired by the algorithm of Hoppe et al.[12]. In contrast to the original algorithm, we use just edge swapping for degree reduction. Vertex splittings or edge contractions are excluded since they would introduce new vertices. In the main phase of the algorithm, only vertices with manifold neighborhood are removed, that is, vertices without non-manifold incident edges. At the end, non-manifold vertices of a specific simple type are removed, too.

The energy function which controls vertex elimination in the algorithm by Hoppe et al. is replaced with a cost function based on the volume of the freespace. The decision on vertex elimination is based on the difference between the new volume of the resulting freespace and the old volume of the freespace of the replaced triangles. If the difference exceeds a given threshold, the operation is not executed.

The operation of removal of a vertex **v** starts with swapping of edges incident to **v**. The goal is to reduce the degree of **v** to three. If this goal is achieved, **v**

is removed and the resulting hole is triangulated. After every edge swapping, the free distance values d_f of the vertices of the involved Voronoi triangles are updated. The new values of d_f are chosen so that they yield a new freespace which is inside the original one. Several cases have to be distinguished for this purpose. The main parameters used are the distance between the original edge and the edge resulting by swapping, and the locations of the points on both edges between which the minimum distance is reached.

The mesh reduction algorithm begins by evaluation of the cost function for every vertex of the mesh $V(S)$. The costs of a vertex are calculated by tentatively eliminating the vertex from the mesh and calculating the difference between the new and the old local freespace volume. Then the vertices are arranged into a priority queue according to increasing costs. Vertices are eliminated in the order of the priority queue as long as their costs are less than a given threshold. After elimination of a vertex, the costs of involved neighboring vertices are updated, and the vertices are re-inserted into the priority queue according to their new costs.

5 Camera control

In the following we show how the VFS-rep of a given scene S can be used for camera control. We first describe how the the distance of the camera to the obstacles of S and how the visibility between the camera and the player can be calculated with the VFS-rep. Then we present a heuristics for minimization of the objective function c of chapter 3.

5.1 Distance calculation

The task of distance calculation is to find, for an arbitrary point **c**, a value $d_f(\mathbf{c})$ so that the ball centered at **c** with radius $d_f(\mathbf{c})$ belongs to freespace. $d_f(\mathbf{c})$ should be not too far from the maximum possible radius. If none exists, this fact should be reported, too.

During camera tracking, the following version of distance calculation is relevant. A predecessor $\mathbf{c}^-$ of **c** and some additional data are known. The additional data consist of a triangle v^- of the VFS-rep so that $\mathbf{c}^-$ is in the freespace of v^-.

For the solution of this "tracking version", two cases are distinguished. If **c** is not in the freespace of v^-, the triangles v' adjacent to v^- are determinded for which **c** is in the freespace of v'. If at least one is found, let $\overline{\mathbf{c}}$ be a point on v' closest to **c**. Then the maximum of the values $d_f(\overline{\mathbf{c}}) - ||\mathbf{c} - \overline{\mathbf{c}}||$ over triangles v' is reported as $d_f(\mathbf{c})$. If none is found, this fact is reported. It means, that **c** is not in freespace, or that the step size of tracking has been possibly too big.

If **c** is in the freespace of v^-, the same calulation is made for v^- instead of the adjacent v'.

At the beginning of tracking, a brute force initialization is performed. All Voronoi triangles v are tested for membership of **c** in their freespace. The test can be performed by checking a nearest neighboring point $\overline{\mathbf{c}}$ of **c** on v for whether the free distance $d_f(\overline{\mathbf{c}})$ of $\overline{\mathbf{c}}$ exceeds the distance between **c** and $\overline{\mathbf{c}}$.

5.2 Visibility calculation

The task of visibility calculation is to check for whether the line segment between the viewpoint $\mathbf{o}_C$ of the camera and the point of interest $\mathbf{o}_P$ of the player is completely in the freespace. It is assumed that a triangle v is given so that $\mathbf{o}_C$ is in its freespace $S_f(v)$.

The problem is solved by calculating a sequence of points $\mathbf{c}_i$, $i = 0, \ldots, m$, so that $\mathbf{c}_0 = \mathbf{o}_C$, $\mathbf{c}_m = \mathbf{o}_P$, $\mathbf{c}_i$ is in the freespace $S_f(v_i)$ of a triangle v_i of $V(S)$, and $\mathbf{c}_i$ is in $S_f(v_{i-1})$, for $i > 0$, too. The sequence is constructed by successively finding the triangles v_i, as follows. If $\mathbf{o}_P$ is in $S_f(v_{i-1})$ of the triangle v_{i-1} of the current point $\mathbf{c}_{i-1}$, the algorithm terminates. Otherwise, the neighboring triangle v of v_{i-1} with the farest intersection point **c** of the freespace $S_f(v)$ with the ray from $\mathbf{c}_{i-1}$ towards $\mathbf{o}_P$ is determined. Then $\mathbf{c}_i := \mathbf{c}$ and $v_i := v$. If the iteration has reached $\mathbf{o}_P$ after termination, visibility between $\mathbf{o}_C$ and $\mathbf{o}_P$ is reported. Otherwise, the exit point **c** of the ray with respect to the freespace of v_{i-1} is reported. In this case, the camera has lost the player. Then $\mathbf{c}'$ is used as the starting point of a search by the camera, which is described in chapter 5.3.

The component c_{vis} of the objective function of camera control in chapter 3 depends on the opening angle γ of a maximum viewcone in which the camera sees the player without occlusions caused by the scene. A possibility is to use the opening angle of the maximum cone with tip at $\mathbf{o}_C$ and axis on the line

segment between $\mathbf{o}_C$ and $\mathbf{o}_P$, which is completely in the freespace. However, the exact calculation of this cone is complicated. For that reason, γ is replaced with an other heuristic measure which is sufficient for the purpose. If visibility between $\mathbf{o}_C$ and $\mathbf{o}_P$ has been detected, we take the minimum of the estimated distances of the $\mathbf{b}_i$, $i = 1, \ldots, m$, to S, where $\mathbf{b}_i$ are the points calculated by the visiblity procedure. The estimated distance is taken as $d_f(\overline{\mathbf{b}}_i) - ||\mathbf{b}_i - \overline{\mathbf{b}}_i||$ where $\overline{\mathbf{b}}_i$ is a nearest neighbor of $\mathbf{b}_i$ on the triangle v_{i+1}.

5.3 Minimization of the objective function

We use a simple heuristics in order to find a translational component $T_{t,\text{opt}}$ of T_{opt} which yields an approximative minimum of the objective function c of chapter 3. The approach is to choose an optimal or at least a favorable solution $T_{t,j}$ for every component c_j, $j \in J := \{\text{vis}, \text{dist}, \text{angle}, \Delta\text{dist}, \Delta\text{angle}, \text{safe}\}$ of c, and take $T_{t,\text{opt}}$ as a weighted average of the $T_{t,j}$. If $T_{t,\text{opt}}$ yields a point outside the freespace, it is shortened so that the resulting point $\mathbf{o}_C^+$ stays in the freespace. A possibility is to take $\mathbf{o}_C^+$ half-way between $\mathbf{o}_C$ and the point at which the ray from $\mathbf{o}_C$ towards $\mathbf{o}_C^*$ leaves the freespace. The weights can be controlled by the camera engine.

If the value of the visibility component c_{vis}, evaluated at $\mathbf{o}_C^+$, is less than a given threshold provided by the camera engine, the procedure is iterated for $\mathbf{o}_C^+$. Otherwise, a procedure searching for the player is initiated.

The searching procedure calculates a shortest polygonal path p on the VFS-rep mesh $V(S)$ between $\mathbf{o}_C^+$ and $\mathbf{o}_P^+$. The second vertex $\mathbf{b}$ of the resulting path is taken for definition the translational component $T_{t,\text{vis}}$ contributing to the overal translational component by $T_{t,\text{vis}} := \mathbf{b} - \mathbf{o}_C$. The contribution of $T_{t,\text{vis}}$ is strengthened by increasing its weight if the player does not become (sufficiently) visible in subsequent steps. In this way, the camera is finally forced on the roadmap provided by the tirangular mesh $V(S)$. If the player does not become visible even then within a given time limit, the camera executes a jump to the player.

The shortest path is searched by the A^*-algorithm[13]. With exception of possibly the first and large point, the vertices of p belong to $V(S)$, and its edges are edges of $V(S)$. If $\mathbf{o}_C^+$ or $\mathbf{o}_P^+$, respectively, is not a vertex of $V(S)$, a vertex $\mathbf{v}_C$ of a triangle of $V(S)$ is selected which has the respective vertex in its freespace. The vertex with the shortest distance to $\mathbf{o}_P^+$ or $\mathbf{o}_C^+$, respectively, is take for this

purpose. The chosen vertex is the second or the last but one vertex, respectively, of p, and the rest of p is constructed as a shortest path between them.

We have not yet described the solutions chosen for all the component functions of c. They are

for c_{vis} see above,

for c_{dist} and c_{angle}

$$T_{t,\text{dist/angle}} := \mathbf{o}_C - (\mathbf{o}_P + d_{\text{opt}} \cdot \mathbf{z}_P)$$

where $\mathbf{z}_P$ is assumed to have length 1,

for $c_{\Delta\text{dist}}$ and $c_{\Delta\text{angle}}$

$$T_{t,\Delta\text{dist/angle}} := \mathbf{o}_C - \mathbf{o}_C{}^-$$

where $\mathbf{c}_C^-$ is the camera location preceding $\mathbf{c}_C$, and

for c_{safe}

$$T_{t,\text{safe}} := \mathbf{v} - \mathbf{o}_C$$

where $\mathbf{v}$ is that vertex of a freespace cell $S_f(v)$ of which $\mathbf{o}_C$ is a element, for which $w(||\mathbf{v} - \mathbf{o}_C||)d_f(\mathbf{v})$ is minimum.

6 Performance evaluation

We have implemented a camera control module based on this approach, within the *Half-Life* environment[18]. Half-Life is a network-based 3D computer game. It works according to the client-server concept (Figure 3). With Half-Life, a free software development kit is provided which allows to implement an own game logic in the server and the functionality of the clients. The camera engine is part of a client.

For Half-Life, several 3D game scenes are available. We have used some of them for evaluation of our solution of camera control: snark pit, stalkyard, rapid-core, datacore, frenzy, lambdabunker, subtransit, and undertow. Figure 4, top, shows the scene *datacore* from outside. In the following we use this scene as a typical example.

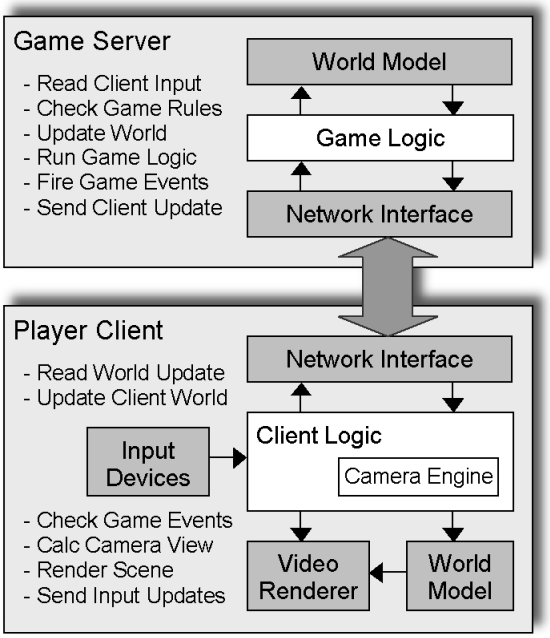

Figure 3: The architecture of the computer game Half-Life.

For the empirical analysis, a PC with an 800 MHz Celeron CPU and 384 MB RAM and Windows 2000 has been used. The program is written in C++.

Figure 4, middle, shows the result of the calculation of the VFS-rep based on uniform point sampling of *datacore*. A reduced VFS-rep is shown in Figure 4, bottom. The non-reduced VFS-rep is not shown because of its density of line segments. Figure 5 gives a closer view on the meshes.

Figure 6 compiles some statistical data of this calculation. The data show that mesh reduction has a considerable effect.

Sampling and calculation of the Voronoi diagram of resulting sampling points by QHULL[2] required 51 s. The subsequent mesh reduction has taken 1:05 h. Similar computation times could be observed for other scenes. The calculation times of mesh reduction, however, seem to be more sensitive to the combinato-

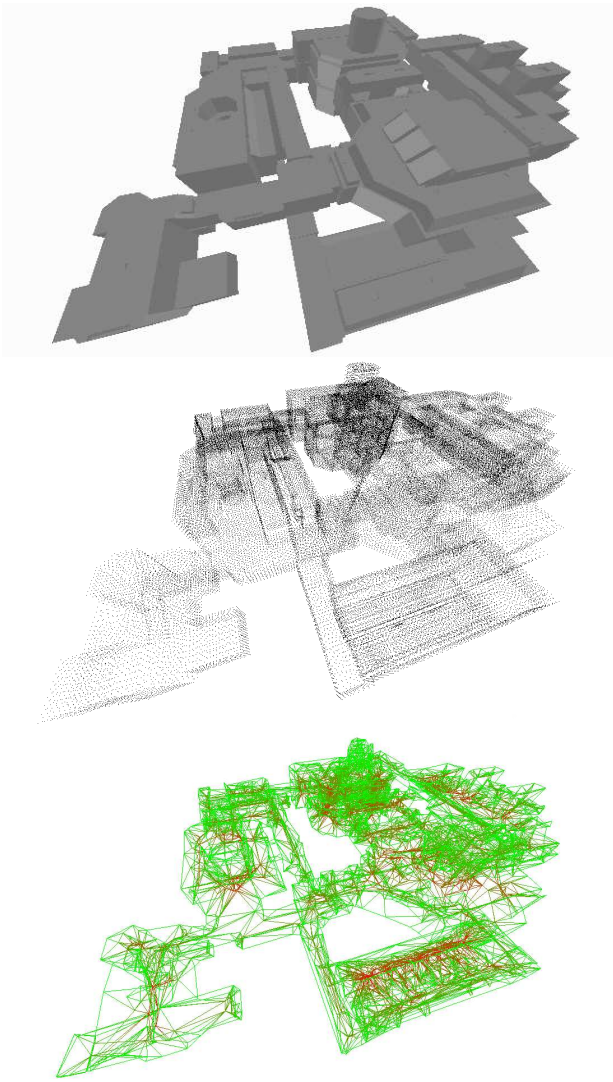

Figure 4: The test scene *datacore*. The first picture shows the scene from outside by its polygons. The middle picture shows the set of sampling points, and the third pictures visualizes the reduced VFS-rep by the edges of the resulting Voronoi triangles.

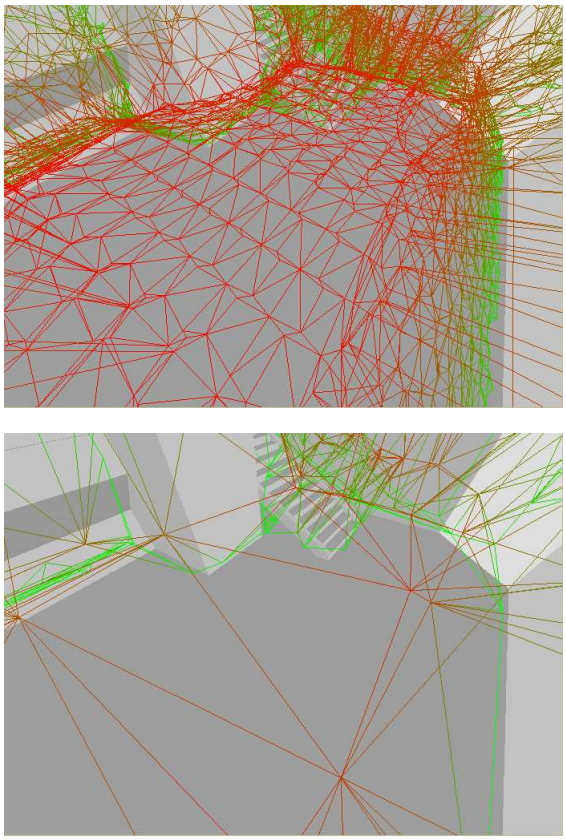

Figure 5: The test scene *datacore*. A closer view of the original and the reduced Voronoi mesh.

rial structure of the mesh than on its size, so that some variance can be noticed even on meshes on equal size.

At a first glance, this time requirement seems considerably. However, it should be noticed that preprocessing has to be invested for every scene just once. For that reason, we did not try to speed up the implementation.

The space requirements of the resulting VFS-rep data structure has been about

# polygons of the input scene	4 722
# sampling points	75 407
# Voronoi vertices	440 256
# inner Voronoi vertices	228 077
# inner Voronoi faces	119 283
# inner Voronoi triangles	339 867
# inner Voronoi vertices after reduction	5 853
# inner Voronoi triangles after reduction	14 046
time of sampling and Voronoi diagram calc./s	51
time of reduction/h	1:05

Figure 6: Quantitative properties of the scene *datacore*. ”Inner” means the the part of the mesh side the datacore hull.

440 KB.

During the game, just about 1% of the overall computing time of the game has been required by camera control.

Figure 7 shows three frames of a criticial situation in the datacore scene in which the VFS-rep has helped. The player disappears at a corner, but is found again by the camera.

7 Conclusions

We have presented an approach to automatic observation of a virtual player in a 3D computer game by a camera. The VFS-rep allows the camera to find the player on a continuous path if the view gets lost. We have demonstrated the usefulness of the approach by an implementation. While calculation of the VFS-rep in a pre-processing step needs some time, the VFS-rep allows on-line tracking, including collision avoidance and visibility estimation, in real time and needs just a minor portion of the computing time of the game.

The emphasis of the paper has been the presentation of the VFS-rep and the related algorithms. The VFS-rep is calculated from a point-sampling of the input scene. Thus this approach is particularly suited for point-based representations of the scene which is of relevance if scanned data and point-based rendering are

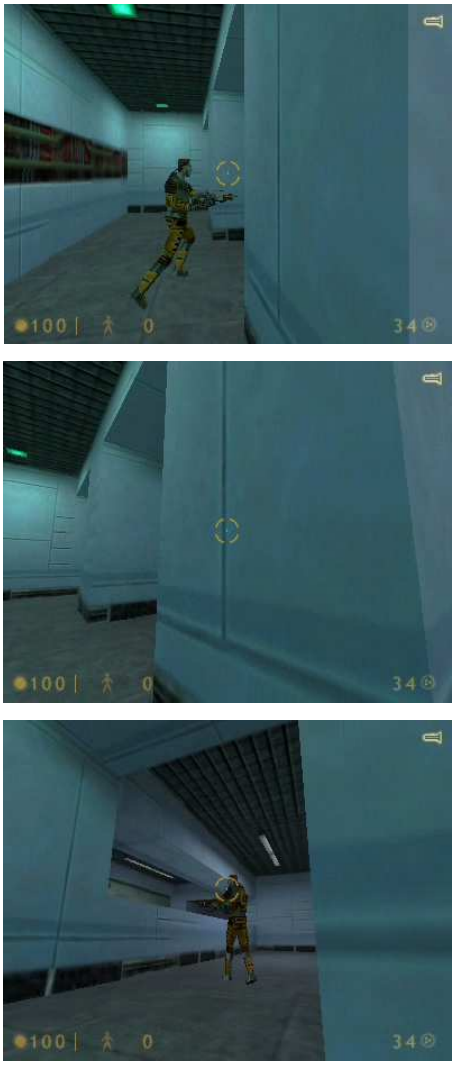

Figure 7: Example of a criticial situation in which the VFS-rep helps. The player disappears at a corner, but is found again by the camera.

used.

With respect to camera control, just the principle has been outlined, exemplified at a simple objective function and a simple heuristic solution. With this objective function, already good camera paths can be achieved if the free parameters are set properly. Additional work might be invested in this topic.

For example, other techniques of camera control, like e.g. that by Halper et al. [11] might be combined with the VFS-rep.

A more significant extension is to more than one player or additional moving objects in the scene. In this case more than just one frame P might be taken into consideration by the objective function. For this purpose, the actors or moving objects might be enveloped by a union of balls whose centers are the origins of frames. Then, for instance, collision between them and the camera can be avoided by expressing that the distance of the frame origins has to be kept at a certain distance from each of the ball centers. This might be achieved e.g. by some sort of repelling functions.

References

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. ACM Computing Surveys 23(3) 345–406, 1991

[2] C.B. Barber, D.P. Dobkin, and H.T. Hubdanpaa. The Quickhull algorithm for convex hulls. ACM Trans. on Mathematical Software 22(4): 469–483, 1996. Available from http://www.geom.umn.edu/software/qhull

[3] C. Bocchini, P. Cignoni, C. Montani, P. Pingi and R. Scopignio. A low cost 3D scanner based on structured light. Computer Graphics Forum 20(3):C-299–C-308, 2001

[4] M. de Berg, M. van Krefeld, M. Overmars, and O. Schwarzkopf. Computational geometry: algorithms and applications (2nd edition). Springer-Verlag, Berlin, 2000

[5] S. Drucker. Intelligent camera control for graphical environments. Ph.D. Thesis, Massachusetts Institute of Technology Media Lab. 1994

[6] S.M. Drucker and D. Zeltzer. CamDroid: A system for implementing intelligent camera control. In *Proc. ACM Symp. on Interactive 3D Graphics 1995*, pp. 139–144, 1995

[7] D.H. Eberly. 3D game engine design. Morgan Kaufman Publishers, San Francisco, 2001

[8] H. Edelsbrunner. Algorithms in combinatorial geometry. Springer-Verlag, Berlin, 1987

[9] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH 2000*, pp. 249–254, 2000

[10] B. Geiger and R. Kikinis. Simulation of endoscopy. In *Computer Vision, Virtual Reality and Robotics in Medicine*, Lecture Notes in Computer Science 905, pp. 277–281. Springer-Verlag, 1995

[11] N. Halper, R. Helbing, and Th. Strothotte. A camera engine for computer games: managing the trade-off between constraint satisfaction and frame coherence. Computer Graphics Forum 20(3):C-174–C-183, 2001

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proc. SIGGRAPH 1993*, pages 19-26, 1993

[13] J.C. Latombe. Robot motion planning. 3rd ed., Kluwer Academic Publishers, Boston, MA, 1993

[14] T.-Y. Li, T.-H. Yu, and Y.-C. Shie. On-line planning for an intelligent observer in a virtual factory. Computer Science Department National Chengchi University, Taipei, Taiwan, 2000. Available from www3.nccu.edu.tw/∼li/Publication

[15] H. Pfister, M. Zwicker, J. van Baar, M. Gross. Surfels: Surface elements as rendering primitives. In *Proc. SIGGRAPH 2000*, pages 335–342, 2000

[16] S. Rubin (ed.). AI game programming wisdom. Charles River Media Inc., Hingham, Mass., 2002

[17] S. Teller and C.H. Sequin. Visibility processing for interactive walkthroughs. In *Proc. SIGGRAPH 1993*, pages 61–69, 1993

[18] Valve Software. Half-Life. Available from http://www.valve-erc.com