Eurographics Symposium on Virtual Environments (2006)
Roger Hubbold and Ming Lin (Editors)

# Camera Setup Optimization for Optical Tracking in Virtual Environments

Philippe A. Cerfontaine[†1], Marc Schirski[1], Daniel Bündgens[1] and Torsten Kuhlen[1]

Virtual Reality Group[1]
RWTH Aachen University

**Abstract**

*In this paper we present a method for finding the optimal camera alignment for a tracking system with multiple cameras, by specifying the volume that should be tracked and an initial camera setup. The approach we use is twofold: on the one hand, we use a rather simple gradient based steepest descent method and on the other hand, we also implement a simulated annealing algorithm that features guaranteed optimality assertions. Both approaches are fully automatic and take advantage of modern graphics hardware since we implemented a GPU-based accelerated visibility test. The proposed algorithms can automatically optimize the whole camera setup by adjusting the given set of parameters. The optimization may have different goals depending on the desired application, e.g. one may wish to optimize towards the widest possible coverage of the specified volume, while others would prefer to maximize the number of cameras seeing a certain area to overcome heavy occlusion problems during the tracking process. Our approach also considers parameter constraints that the user may specify according to the local environment where the cameras have to be set up. This makes it possible to simply formulate higher level constraints e.g. all cameras have a vertical up vector. It individually adapts the optimization to the given situation and also asserts the feasibility of the algorithm's output.*

Categories and Subject Descriptors (according to ACM CCS): G.1.6 [Numerical Analysis]: Optimization Constrained optimization, Global optimization, Gradient methods, Simulated annealing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Virtual reality I.4.1 [Image Processing and Computer Vision]: Digitization and Image Capture Camera calibration I.4.8 [Image Processing and Computer Vision]: Scene Analysis Motion, Tracking

## 1. Introduction

The number of methods and systems using multiple cameras for 3D reconstruction has drastically increased over the past few years [Fau93] [KH94] [FZ98] [Pol00] [KNZI02]. The variety of this kind of systems ranges from low cost hardware like simple webcams trough commodity hardware [Tsa87] up to expensive high performance infrared tracking systems. There need to be at least two cameras in the system and in practice there may be a lot more, like in motion capturing for instances. This makes it difficult to find the optimal alignment for all cameras. Since all these systems live from the fact that the view frusta of their cameras overlap, it is extremely important to find the best possible alignment. The quality of a tracked or captured sequence, in terms of maximal visibility and minimal occlusion, is directly connected to the quality of the camera setup. If the setup is constructed incorrectly, the tracking sequence quality will be poor. In the best case you will simply have to do the whole capture process again and in the worst case of online tracking the missing data cannot be recovered. Recapturing a whole sequence can be arbitrarily expensive, considering the amount of time, manpower, materials and money spent on certain movie sequences nowadays [Mar99]. For all tracking scenarios it is preferable to guarantee an optimal underlying camera setup from the start.

† corresponding author, email: cerfontaine@rz.rwth-aachen.de

## 2. Related work

Although there are different techniques to calibrate a single camera or a set of cameras to estimate certain parameters like the current alignment [Tsa86] [Zha98] or lens distortion [BBV01], those techniques often used in Augmented Reality (AR) applications [GCHO02] do not focus on how to position several cameras to get the best volume coverage. There are publications that focus on multi-camera systems like [SMP05, AW05, SWI06] while others consider systems with only one camera [HS97]. Our goal is to optimize the alignment of a multi-camera system by using methods from the mathematical minimization theory like the ones described in [Hro02]. We had to adapt those techniques from the continuous case to our discrete approach of the problem.

Due to the considerable processing power of modern programmable graphics hardware, a variety of approaches for leveraging this power have been introduced lately, leading to the (not quite precise) acronym GPGPU for general purpose computation on graphics hardware. The applications range from very graphics-centric uses like ray-tracing [PBMH02] over computational fluid dynamics [HBSL03] to very generalized numerical algorithms [KW03]. An overview of general purpose computations on GPUs can be found in [OLG*05]. We take advantage of the additional processing resources on the graphics card to accelerate the camera visibility test.

## 3. Overview

The following sections gradually explains how our algorithm works. First we are going to start by stating our approach of the problem and explain why we chose this method. Then we will specify how to compute the quality and score for a given camera setup. Afterwards we will discuss the optimization techniques that we use, and finally we will present the achieved results and our conclusions.

## 4. Camera setup optimization

### 4.1. Specifying the problem

The aim is to track a moving object within a certain volume without any data loss. This means that all positions within this volume need to be visible by at least two cameras at all times, when we neglect occlusion problems. The question is how to position these cameras, such as to reach this total coverage. In fact, this is the necessary condition for 3D reconstruction of positions within the volume. The supreme goal would be the reconstruction inside the complete volume. Although achieving this goal may seem to be impossible, e.g. in the case where the volume to be tracked is bigger than the sum of all camera-view-frusta, the goal will always be to minimize the unseen part of the volume or the part only covered by one camera. Positions within the desired tracking volume are classified as traceable once they are seen by at least two cameras of the specified setup.

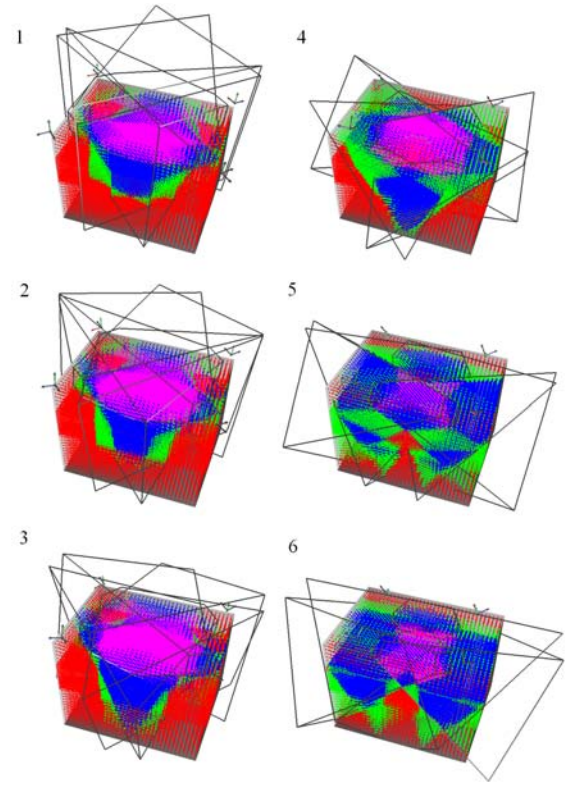


**Figure 1:** *Optimization step sequence for a five sided CAVE with four cameras. The camera positions were constrained to remain on the open ceiling of the CAVE. Notice the increased point sample rate in head height to improve the head-tracking robustness in the CAVE.*

#### 4.1.1. Volume of interest

First of all we need to describe the volume inside of which we want to perform 3D position reconstruction. In order to do so we are going to specify a completely arbitrary list of positions, that we will henceforth call volume of interest. This approach yields maximum flexibility in case of fancy shaped tracking volumes or certain regions of interest inside the volume which, have increased importance.

We simply can describe complex volumes by sampling them without even taking care of the sample's ordering. Or we could describe the volume by testing a uniform grid of positions against a set of implicit functions we provide. We can even specify volumes that are completely disjoint in space without any additional effort.

Stressing the importance of a certain region inside the volume of interest becomes almost trivial since we may simply increase or duplicate the number of positions in this specific area of the volume. The use of a weighting mechanism in the scoring function is another possibility, the only thing we

have to do in that case is to add an importance weight to every position.

In some cases where the volume of interest has a high symmetry like a platonic solid for instance it may seem trivial to find the optimal solution for the camera alignment problem through an analytic approach. As long as complications like parameter constraints, local restrictions and specific regions of interest remain unconsidered this is acceptable. But even then the analytic approach needs to be adapted and re-evaluated for every change in the volume of interest. In contrast, our algorithm only needs to be restarted and then it automatically optimizes the multi-camera system's alignment to fit the needs of the user.

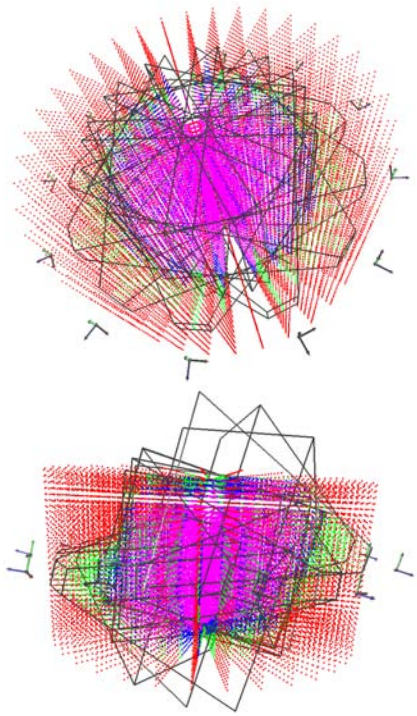

**Figure 2:** *Optimization for a circular setting with central region of interest*

#### 4.1.2. Camera setup

The second important part of the tracking problem are the cameras. To be able to optimize the setup, our algorithm needs to know about the cameras we use. In fact, optimizing the camera setup means optimizing the individual camera parameters. Thus for the gradient based steepest descent approach in section 4.3.1 we need initial values for the parameters we want to optimize, and fixed values for constant parameters. In order to concretize the parameters we will name them to allow a better comprehension. The necessary parameters to decide whether a point in space is currently seen by a camera are its position and orientation as well as those parameters describing the viewing volume of the camera device. We considered two possible scenarios for the viewing volume: First the classical view-frustum and second a view-cone for lenses with huge radial distortion, e.g. fish-eye-lens webcams [BBV01]. In this paper focus on the frustum scenario where in addition to the position and orientation of the camera we need the six culling planes for maximal flexibility. This leaves us with twelve parameters per camera, which we need to specify and optimize. Usually the parameters concerning the view-frustum of the camera can be looked up in the camera's manual since they are a direct consequence of the current camera's lenses and settings. Finding some field of view angles and focal distances is sufficient to set up the corresponding view frustum. For homogeneous camera setups with identical cameras we have to check the frustum parameters only once, while with heterogeneous camera systems we need to collect these parameters for each camera type.

The six remaining parameters concerning the camera's initial position and orientation will have to be given separately for each camera. There are various ways to gather them, e.g. measuring an existing setup with a laser rangefinder device. Another more convenient way would be to use the algorithm described in [SMP05] to compute them automatically from camera footage of known regular calibration shapes. Since these are the parameters we want to optimize, the measurements just need to be approximate and not done a hundred percent accurate. Providing an initial camera setup may seem complex, but also has the important advantage to give us feasibility certitude for the computed optimized camera setup. Since the initial setup is supposed to be feasible, the outcoming solution, representing the closest local minimum, is probably more realistic than a solution which is far away from it. Now we will explain how to evaluate a given camera setup's quality before we go into further details.

### 4.2. Measuring the camera setup quality

Once the camera parameter values are available, we need to evaluate the impact of altering them on the setup's quality in terms of coverage of the specified volume of interest. In order to do this we compute an $N \times M$ visibility matrix $v$ where $N$ is the number of cameras and $M$ is the number of volume samples. This matrix tells us whether the $i^{th}$ camera sees the $j^{th}$ position or not. Every position is tested against each camera, and the outcome is written to the visibility matrix $v[i][j] \in \{0,1\}$. Having this matrix computed allows us to specify a scoring or evaluation function taking the matrix as input. Depending on the given requirements and scenario it is possible to have various evaluation methods. Probably the most common for tracking would be to count the number of positions seen by at least two cameras.

Evaluating the quality of a given camera setup is called

a query, i.e. computing its number of traceable positions $\sharp positions_{traceable}$, can be split up into three distinct processes:

1. For every camera $i$, compute the visibility $v[i][j] \in \{0,1\}$ of every point $j$.
2. For every point $j$, add up the visibility scores over all cameras $i$, i.e. $v_j = \sum_{i=1}^{N} v[i][j]$.
3. For every point $j$, threshold $v_j$ with the desired minimum number $k$ of cameras seeing point $j$ and sum up the results, i.e. $\sharp positions_{traceable} = \sum_{j=1}^{M} thresh_k(v_j)$ with $thresh_k(n) = 1$ for $n \geq k$ and $thresh_k(n) = 0$, otherwise.

A setup achieving an increased number of traceable positions for this evaluation method has a better coverage of the specified volume of interest. But if heavy occlusion problems are encountered during the tracking process it may become necessary to increase the number $k$ of cameras necessary for a position to be classified as traceable. For the evaluation this simply means counting the number of positions seen by at least three, four or even more cameras thus enhancing the robustness against marker occlusion during motion capture or simple head tracking for instances. The visibility matrix also offers the possibility to measure a camera's impact on the whole camera setup. To do so we simply sum up the number of positions seen by this camera or count the positions seen by the camera which account for the number of traceable positions.

In some cases it might prove useful to use viewing cones instead of view frusta for the visibility test; in fact, cheap camera devices like most webcams suffer from radial distortion problems. This means that the corners of an image provided by such a camera are distorted. So the pixels in the corners do not necessarily yield the objects one would expect when casting a ray through those pixels on the image plane. Therefore it might be better to completely discard the image's corners during the 3D reconstruction process. Although there are techniques [BBV01] [Wil94] for computing image filters and transformations to cope with radial distortion, it is often the case that these kinds of transformations cannot be applied in real-time scenarios due to performance problems. In such cases, we may skip the corners and keep only the central, inner circle of the image by swapping the classical view frustum test with a view cone test in our evaluation method. We would still have to transform the considered position in space to the camera's coordinate system and check if the distance from the camera's origin to the point lies within near and far plane range. We also need a simple scalar product to check whether the angle between the view axis of the camera and the direction vector from the camera to the point is smaller than half the opening angle of the cone in order to decide wether the point is seen.

Because the scoring function of our evaluation method only depends on the computed visibility matrix, we may simply alter the way we compute it without having to adapt the score function or even the complete optimization mechanism based on it. The camera setup's quality or score can be expressed as the remaining fraction of unseen or not traceable positions. Hence zero would be an already perfect setup and one would be the worst camera setup where not even a single position in the volume of interest fulfils the desired criterions. However, having a means to evaluate and measure the camera setup's quality enables us to change camera parameters and to compute the impact on the final score.

#### 4.2.1. Shifting the workload to the GPU

As the first two steps of the evaluation method presented in the previous section consist of largely identical operations being executed on a large number of data objects, i.e. points, they can easily be modeled as SIMD (*Single Instruction - Multiple Data*) operations (see [Sie79]). This in turn allows for a mapping to the computational model employed in modern GPUs. The final step is a reduction operation, which is more difficult to be executed efficiently on graphics hardware [Har05]. Luckily, modern GPUs offer an asynchronous query mechanism, which provides information about the number of drawn fragments for a given time interval, and which can be used for the final summation step.

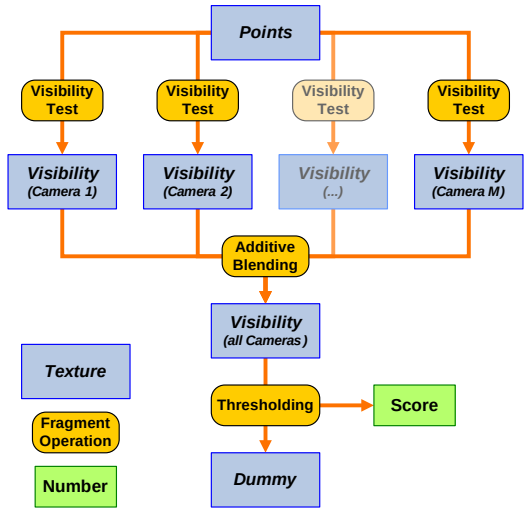


**Figure 3:** *Order of operations and data flow for GPU-based camera setup evaluation.*

To employ GPUs for the computation of the given problem, data elements and computations are mapped to graphical primitives and vertex or fragment shaders, respectively. We store the point coordinates in the color components of 32bit floating-point RGBA textures. For the results of the visibility check as well as the summation process, we rely on 8bit textures with a single color channel. This allows a threshold with a desired visibility of up to 255 cameras. For now, we use *pbuffers* for render-to-texture functionality. For the computations, we exclusively use fragment shaders operating on texels. As usual in GPGPU applications, we execute

computations by drawing a single quad over the render target.

This leads to the following order of operations for the evaluation of a given camera setup (see Figure 3):

1. For every camera, the fragment shader for the visibility test is parameterized with the camera's transformation matrix. It takes the point texture as input and writes its result into a texture associated with its camera. By having every camera associated with a dedicated texture, the results of the visibility check can be re-used for a future setup evaluation if the camera has not moved.
2. The contents of all camera textures is added into the summation texture through additive blending.
3. An occlusion query is started, which will count the number of fragments drawn until the end of the query.
4. The fragment shader for thresholding is parameterized with the desired minimum visibility. It takes the summation texture as input and writes a fragment for every point with at least the desired visibility count and discards the fragment otherwise.
5. The occlusion query is finished, resulting in the number of fragments drawn, i.e. points passing the threshold in the previous step.
6. The results of the occlusion query are retrieved.

Note, that the graphics pipeline is quite deep on modern hardware. In combination with its ability to execute commands asynchronously with the calling program, this allows the graphics hardware and the main processor to work in parallel, thus leading to a more efficient use of available computing resources. Nevertheless, retrieving the result of an occlusion query forces the GPU and the CPU to synchronize. In order to avoid unnecessary stalling, we execute steps 1 to 5 for several camera setups before retrieving the results of the query, whenever possible. This minimizes synchronization overhead and maximizes parallelization of GPU and CPU.

### 4.3. Optimizing the camera alignment

#### 4.3.1. Discrete, gradient based steepest descent

As the problem we are trying to solve is discrete, a camera either sees a position or not, we neither have a function nor a system of equations we could try to minimize or solve. But we have a score that indicates how close we are to the final goal. Our approach takes the output from this scoring function and tries to minimize it in the same way a solver for non-linear systems of equations would minimize the residual of the system to be solved.

Knowing all parameters for all cameras in the setup gives us the possibility to modify them and evaluate the impact on the final score. By increasing or decreasing a parameter value like the camera position on the X-axis we can compare the obtained scores and decide whether it is better to move the camera in positive or negative X direction. Having the scores for the initial, increased and decreased camera position enables us to compute a gradient for the score function $sco(pos_X)$ with respect to this parameter. We either compute the gradient $G(pos_X)$ as forward, backward or central differences.

The gradient vector for all parameters indicates the *direction* of greatest improvement from the current camera setup state. Moving the whole setup into that direction would thus yield the best increase in terms of tracking quality within the volume of interest with respect to the specified score function. Moving the whole system, i.e. modifying all parameters at once, is called an *aggregate step*, while altering only one parameter at a time is called a *single step*. To achieve faster convergence from the initial camera setup towards the local minimum of the score function, we use aggregate steps as long as they improve the setup's quality. Once the quality would have been worsened by an aggregate step we switch to single step mode and modify the parameter with the greatest gradient first. If improvements can not be obtained even through repeated single steps, the local minimum of the score function has been reached. In some cases it might prove useful to diminish the stride to see if the minimum can be approached even more accurately.

#### 4.3.2. Simulated annealing

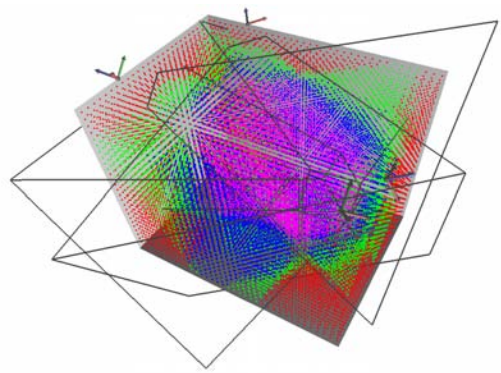

**Figure 4:** *Camera setup optimization using simulated annealing with positional constraints to the upper edges of the CAVE.*

In addition to the steepest descent gradient based algorithm described in the previous section, we implemented a simulated annealing variant [Hro02]. This implementation has the advantage to be independent of initial parameter values since it uses a random uniformly distributed sampling of the domain of the score function. Therefore the stride for each variable parameter also becomes obsolete. The only similarity to the gradient based variant is the score function and the fact that this function is discrete. This discreteness is somehow special since simulated annealing is typically used for minimizing continuous functions. Another advantage of

simulated annealing is that it does not necessarily get stuck in the closest local minimum. Depending on the current *temperature* of the system it has a certain probability to leave an already good state to find a better one, even if it has to go through worse states before. The temperature is an indicator for the likelihood to change the current state for an inferior one. The greater the temperature, the higher is the probability for such an exchange. In general the starting temperature is high and then gradually decreases during the optimization process. This so called cooling procedure progressively stabilizes the whole system.

#### 4.3.3. High level parameter constraints

Sometimes it is useful to restrain the optimization algorithm to the local environment where the cameras have to be placed. For instances, if we plan to use the camera footage for a movie, like the *bullet time* effect in the first Matrix film [Mar99], it would be appropriate to keep a vertical up vector during the capture process. Sometimes, due to spatial, technical or camera mounting limitations, the camera parameters can only take certain values within a valid interval. To integrate such constraints into the optimization, we adapted the way we modify the camera's parameters. The first thing that makes up a parameter is its variability. In fact some parameters like the six camera frusta values might be constant. Such parameters can be skipped from the optimization using a simple flag which reduces the amount of degrees of freedom in the system. Next we might want to specify an interval of values that the parameter may take. When using support stands to mount the cameras they can only be placed at certain heights between the minimum and the maximum range of the stand. The last property we integrated is interval cyclicity, meaning that as for angles between $-\pi$ and $\pi$ radians defining the camera's orientation we can cycle through the defined interval. You could also imagine an ordered list of intervals with valid parameter values giving you full flexibility to constrain your setup's optimization. When experimenting with different parameter constraints the benefits of the GPU implemented visibility test in terms of speedup are a real asset.

## 5. Results and use cases

Since the number of possible scenarios for our optimization algorithm is quite large, we took several common use cases and focused on those for our analysis. The first case we examined was the one displayed in Figure 1. We parameterized a five sided CAVE with four cameras that had to remain on the open top-side. We also increased the sample rate in head height to enhance the head-tracking reliability.

The camera's position and orientation are represented using an OpenGL style cartesian coordinate system where the viewer looks into negative Z-axis direction and the camera up vector points to the Y-axis. The camera view frustum is drawn as a wire frame to keep its inside visible.

The employed solving method was the steepest descent approach and the corresponding convergence curve is shown in Figure 5. The Y-axis shows the normalized score $\in [0,1]$ and the X-axis indicates the number of visibility tests, called queries, that were necessary so far. We intentionally opted for this runtime indication and comparison method, since it is independent of the underlying hardware and the chosen visibility test (CPU or GPU). It also has the advantage of being applicable for both the steepest descent and the simulated annealing algorithms. As depictured in Figure 5 the curve is continuous and monotone, which matches our expectations as steepest descent is a greedy approach that only takes parameter values that improve the setup's quality for the next step.

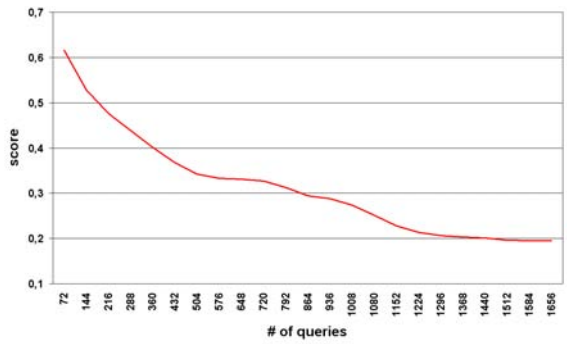


**Figure 5:** *Steepest descent convergence curve.*

The second scenario we examined had even harder constraints than the first one. The cameras in the CAVE had to remain on the upper edges because of a mounting rail construction. This time we used the simulated annealing implementation and came to the solution in Figure 4 and the curve in Figure 6. The curve is not monotone anymore since the algorithm has a certain probability of exchanging its current camera setup for a lower-quality one. This probability is determined by the *temperature* mentioned in the simulated annealing section. As the temperature drops during the cooling process this probability decreases and the convergence curve smoothes. A remarkable feature of this use case is the symmetry of the final setup obtained in Figure 4. The random sampling employed here would have suggested a more unsymmetrical solution.

Comparing the steepest descent and the simulated annealing algorithms is not as easy as it may seem because the outcome strongly depends on the initial parameter values for the steepest descent algorithm, while simulated annealing is completely independent of it. Another complicating factor is that simulated annealing is not deterministic, and thus the results may vary from time to time. The overwhelming amount of degrees of freedom including the number of cameras, all their parameters and constraints, as well as the infinite possibilities related to the volume of interest, make it difficult to

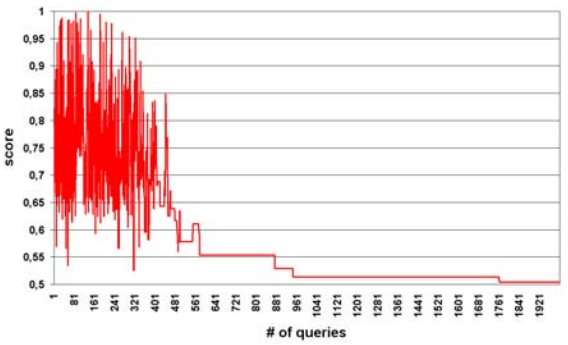


**Figure 6:** *Simulated annealing convergence curve.*

produce a fair, complete and exhaustive comparison. In general, the simulated annealing algorithm takes more time to evaluate than steepest descent but may find better settings, especially when the initial values for steepest descent are meager. Regular, symmetric volumes of interest are more suitable to the gradient approach so that simulated annealing is more likely to find a better solution with irregular volumes.

As third case we considered the circular setup with the central region of interest from Figure 2. Here you can notice that the analytical solution to place the cameras on the outskirts of the volume is not necessarily the best solution. In the lower part of the figure is an intermediate step of the steepest descent algorithm were you can see how the cameras on one side are gradually lowered and turned downwards while the cameras on the other side do exactly the opposite to minimize the untraceable part of the volume. The algorithm was able to considerably increase the traceable volume because the initial setup was far from covering the desired volume.

### 5.1. GPU acceleration benefits

For measuring the results of shifting the computational load of the camera setup evaluation to the GPU, we compared the execution time of an evaluation request including score retrieval running on a CPU and a GPU. This evaluation is called query in Figures 5 and 6. The test machine consisted of a 3.2GHz Pentium 4 equipped with an NVIDIA GeForce 6800 Ultra. All fragment shaders were implemented using GLSL.

Using the CPU turned out to be faster than using the GPU for small numbers of points to be evaluated, e.g. a volume with 16x16x16 samples resulting in 4096 points. However, our measurements showed a significant speed-up by employing the GPU with a rising number of points and cameras. Evaluating setups with 1 to 4 cameras with 100k to 1M points resulted in an acceleration factor of 6 to 8. For extreme setups with 36 cameras and over 2M points, we even achieved a speed-up factor of 9, which is a major asset when experimenting with different initial parameter values and/or constraints for the gradient based steepest desent method.

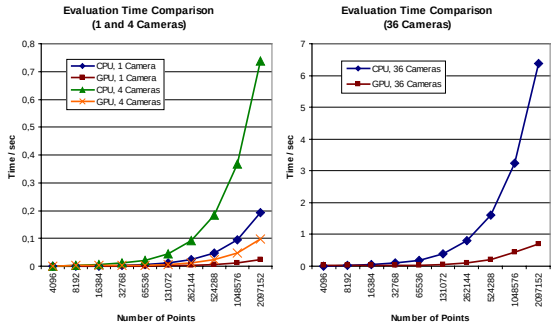


**Figure 7:** *Benchmark results for CPU-based and GPU-based evaluation of a camera setup.*

## 6. Conclusions and future work

We developed an algorithm to improve the tracking reliability in virtual environments like CAVEs for instances. This algorithm is also applicable to any multi-camera setup used for 3D-reconstruction. Examples include motion tracking systems or modern special effect movie scenes. It could also be used in the conception of camera based security and surveillance systems to analyze and optimize area coverage, although this use case has not been evaluated yet.

The employed optimization methods were able to significantly increase the tracking volume of the given camera setups, see Figures 5 and 6. A score of 50% may seem rather poor when minimizing towards 0 in the case of Figures 4 and 6. Considering the positional constraints applied to the cameras, their viewing angle and the tracking volume it is still a convincing result.

The GPU-based query implementation demonstrated the GPU's supremacy over the CPU in SIMD scenarios again. Although runtime is not really crucial for this application it is always preferable to have the desired results as soon as possible. Especially when we want to improve the precision of our optimization, we have to increase the position sample rate to get a better approximation of a continuous volume. In such cases the computation lasts for several minutes on a standard CPU. The sample rate for the screen shots in Figures 1, 2 and 4 were intentionally low to grant the reader a better visibility. The number of positions is limited by the maximum texture and memory size of the graphics card but should be sufficient for most applications.

We will conduct further investigations on the optimizations's impact on real life tracking sequences in various virtual environments. An alternative implementation using a genetic optimization algorithm is also planned. The different

optimization strategies will be compared in order to decide whether there is one superior approach or if the algorithms are complementary with respect to the considered environment.

## References


[AW05] ALLEN B. D., WELCH G.: A general method for comparing the expected performance of tracking and motion capture systems. In *12th ACM Symposium on Virtual Reality Software and Technology (VRST)* (Monterey, California, November 2005).

[BBV01] BRÄUER-BUCHARDT C., VOSS K.: A new algorithm to correct fish-eye- and strong wide-angle-lens-distortion from single images. In *ICIP* (2001), pp. 225–228.

[Fau93] FAUGERAS O.: *Three-Dimensional Computer Vision*. MIT Press, 1993.

[FZ98] FITZGIBBON A., ZISSERMAN A.: Automatic 3d model acquisition and generation of new images from video sequences. In *European Signal Processing conference (EUSIPCO '98)* (Rhodes, Greece, 1998), pp. 1261–1269.

[GCHO02] GIBSON S., COOK J., HOWARD T. L. J., ORAM D.: Accurate camera calibration for off-line, video-based augmented reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2002)* (Darmstadt, Germany, September 2002).

[Har05] HARRIS M.: Mapping computational concepts to gpus. In *GPUGems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, 2005.

[HBSL03] HARRIS M. J., BAXTER W. V., SCHEUERMANN T., LASTRA A.: Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 92–101.

[Hro02] HROMKOVIC J.: *Algorithmics for Hard Problems*, second ed. Springer, 2002.

[HS97] HEIKKILA J., SILVEN O.: A four-step camera calibration procedure with implicit image correction. In *IEEE Computer Vision and Pattern Recognition* (1997), pp. 1106–1112.

[KH94] KUMAR R., HANSON A.: Robust methods for estimating pose and a sensitivity analysis. In *CVGIP-Image Understanding* (1994), vol. 60, pp. 313–342.

[KNZI02] KURAZUME R., NISHINO K., ZHANG Z., IKEUCHI K.: Simultaneous 2d images and 3d geometric model registration for texture mapping utilizing reflectance attribute. In *Fifth Asian Conference on Computer Vision (ACCV)* (January 2002), vol. I, pp. 99–106.

[KW03] KRUEGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 908–916.

[Mar99] MARTIN K. H.: Jacking into the matrix. *Cinefex 79* (October 1999), 66–89.

[OLG*05] OWENS J. D., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A. E., PURCELL T. J.: A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports* (2005), pp. 21–51.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (July 2002), 703–712. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[Pol00] POLLEFEYS M.: *3D Modelling from Images*. Tutorial notes, in conjunction with ECCV 2000, Dublin, Ireland, June 2000.

[Sie79] SIEGEL H. J.: A model of simd machines and a comparison of various interconnection networks. *IEEE Transactions on Computers C-28*, 12 (December 1979), 907–917.

[SMP05] SVOBODA T., MARTINEC D., PAJDLA T.: A convenient multi-camera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments 14*, 4 (August 2005). To appear.

[SWI06] STATE A., WELCH G., , ILIE A.: An interactive camera placement and visibility simulator for image-based vr applications. In *Engineering Reality of Virtual Reality 3D (Imaging, Interaction, and Measurement) 18th Annual Symposium on Electronic Imaging Science and Technology/SPIE* (San Jose, CA, January 2006).

[Tsa86] TSAI R.: An efficient and accurate camera calibration technique for 3d machine vision. In *IEEE Conference on Computer Vision and Pattern Recognition* (Miami Beach, Florida, 1986), pp. 364–374.

[Tsa87] TSAI R.: Metrology using off-the-shelf tv cameras and lenses. In *IEEE Journal of Robotics and Automation* (August 1987), vol. 3, pp. 323–344.

[Wil94] WILLSON R. G.: *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Department of Electrical and Computer Engineering,Carnegie Mellon University, January 1994.

[Zha98] ZHANG Z.: A flexible new technique for camera calibration. Technical Report MSR-TR-98-71, December 1998.