Optimized Camera Ranking Algorithms for Real-time Image Based Rendering of Endoscopic Image Data

Marco Winter, Bernhard Then, Günther Greiner
Computer Graphics Group, University of Erlangen-Nuremberg
Am Weichselgarten 9, 91058 Erlangen, Germany
<marco.winter>,<guenther.greiner>@informatik.uni-erlangen.de

Abstract

Unstructured Lumigraph Rendering is a well-established and flexible image-based rendering technique to create novel views from a set of acquired images. However, the standard implementation may not provide a pleasing visualization when images are captured under suboptimal conditions. In our scenario, lightfield information is gained using images from endoscopes that observe the operating field during minimal-invasive surgery. Suboptimal lighting conditions and limited mobility of the endoscope lead to visual artefacts in the final visualization of the scene.

To resolve these problems, we propose a set of new algorithms for computing the ranking criterions that are used by the Unstructured Lumigraph for visualization. Using these algorithms, we get satisfying results for visualization of endoscopic image data captured under the aforementioned conditions. To achieve real-time visualization, the vital parts of our algorithms have been implemented using OpenGL and GLSL, thus making use of the features and speed of modern graphics hardware.

1. Introduction

The Unstructured Lumigraph (ULG) [2] represents one of the most popular image-based rendering techniques used so far. Its flexibility and stability makes it a good choice for surface visualization in a wide range of application areas, e.g. in medical scenarios. We use the Unstructured Lumigraph approach in computer assisted minimal-invasive surgery in order to visualize the surface of an operating field lying inside the human body, where image data is obtained from endoscopic video frames. Registration of the successive images is done by applying structure-from-motion algorithms and utilizing hardware-assisted tracking methods, like positioning robots or optical tracking systems [11, 13].

When acquiring such endoscopic images, there are two main problems that we have to deal with:

Local surface lighting: When capturing images from objects or scenes, the lighting condition of the environment in general is static and usually can be adjusted according to particular needs. In endoscopic investigations, the operating field is illuminated by an external light source whose emitted light is directed via an optical fiber system along the tube into the body. Because of the “spotlight” properties of the light, the part of the operating field that is in the focus of the endoscope is well lit, while the intensity falls off significantly at the surrounding area. This results in visual intensity artefacts when rendering these images using the ULG, as can be seen in figure 1. It also complicates scene depth reconstruction using structure-from-motion algorithms, as the visual information gets worse at the darker border areas of the images.

Spatial camera distribution: A convenient approach for image acquisition of an object is a regular sampling from the hemisphere or complete sphere around the object. However, in medical scenarios, the endoscope tube is inserted into the body through a small incision of usually 0.5–1.5 cm in diameter. Since this incision should not be enlarged by moving the endoscope, it limits the handling of the endoscope to only longitudinal and lever-like motion. So, image capturing can be done only inside the “operating range” of the endoscope’s lens. Under these conditions, the acquired camera images are distributed non-uniformly in space, with high density along the lens’ path and low density in lateral directions (see figure 2). When using a camera ranking approach as described in [6] and rotating the reconstructed lightfield, this camera distribution leads to visual “popping” artefacts, as can be seen in figure 3.

The visual artefacts described above are mainly caused by the visualization part of the ULG which is not designed to take care of these suboptimal image acquisition conditions. Therefore, we modified the visualization process by developing a set of new ranking algorithms for the Unstructured Lumigraph which are specially designed to address the issues discussed above.
Figure 1. The ULG visualization of a physical model representing a liver with gall bladder embedded in an organic looking environment. Due to different intensity distributions captured in the endoscope images, transitions between different images are clearly visible at the reconstructed liver model.

Figure 2. The camera path corresponding to figure 1 reconstructed from the movement of the endoscope. The green points show the lens’ positions and the gray frustums visualize the corresponding fields of view.

1.1. Previous Work

Image-based rendering is a now well-established technique that has seen several key concepts developed over time. Starting with the definition of the plenoptic function [1], the terms lightfield [8] and lumigraph [7] were coined for a dimensional reduction of complexity. In the following years real-time applications like view-dependent texturing [5], surface lightfields [14, 3] and the ULG were presented. The ULG approach may be seen as the most general regarding the requirements of the underlying lightfield and possible availability of proxy geometry. Therefore, many newer image-based rendering techniques are based on this approach.

The ULG introduces the notion of a so-called camera blending field. This field is created for each novel view and stores per-pixel contribution weights for all cameras to the new view. The weights are computed by considering criterions like viewing angle, image resolution and field of view of each camera. To use this information efficiently, the blending field is triangulated in 2D and weights are stored only at the vertices of the triangulated mesh (which is called the screen mesh). Intermediate weights are then interpolated during rendering using linear interpolation provided by graphics hardware.

In [6], a global camera preselection step is introduced that selects the $k$ best cameras ranked by similar criterions as used by the ULG. Only these cameras are used to create the blending field and the screen mesh for the novel view. Furthermore, the screen mesh is constructed as a heightfield and contains depth information from the $k$ best cameras. In contrast to the original approach, the blending weights at the screen mesh vertices are only based on the information about which camera provided the depth information at this vertex.

The approach developed by Vogelgsang et al. [10] combines both the camera preselection step from [6] and the weights computation for the blending field from [2] to get smoother visual results. Furthermore, they introduced the notion of depth meshes to store per-camera depth information. These meshes are rendered into a z-buffer from the novel view’s point of view to get a “depth print” of the scene. This z-buffer is then tessellated to get the screen mesh needed to store the blending field information.

Willems et al. [12] use a slightly different approach as they compute local blending fields per depth mesh instead of a global blending field.

1.2. Overview

In this paper, we introduce new ranking criterions and algorithms for reasonable camera selection in order to overcome the artefacts described in the previous section. As proposed in [10, 12], we consider the camera selection as a...
process divided into two separate steps: First, a camera pre-
selection step takes place. Here the \( k \) best cameras are se-
lected which make the best contribution to the novel view’s
image. Afterwards, the screen mesh and the weights for
the camera blending field of the novel view are computed,
which will be called the local selection step. In the follow-

ing sections, we describe new algorithms for both steps of
the ranking process.

Furthermore, we restrict ourselves to the usage of depth
meshes as it is done in [10, 13]. This is because depth
meshes usually are smaller in size and still easier to process
using hardware-accelerated interfaces like OpenGL or Di-
rect3D. Nevertheless, all algorithms described in this paper
can be modified to also make use of depth maps if needed.

In section 2, we first present methods for the second part
of the camera selection process, i.e. the local selection step.
The preceding camera preselection step will be discussed
afterwards in section 3. We will show some results of our
methods in section 4 and conclude our work in section 5.

2. Local Selection

In the local selection step, the screen mesh and the blend-
ing field are computed. Both are vital elements for the final
visualization, so we had to examine both algorithms to see
how their ranking algorithms had to be modified in order to
get satisfying results using our endoscopic image data.

In this step of the visualization pipeline, we were espe-
cially interested in addressing the artefacts resulting from
the lighting conditions of the operating field: because these
artefacts are based on per-pixel information of each origi-

nial image, solutions have to be found at the point where
this information is processed. In the following, we describe
methods how to handle the two described symptoms of sub-
optimal lighting conditions, i.e. badly reconstructed depth
information and visual intensity artefacts.

2.1. Screen Mesh Generation

The screen mesh can be created from depth meshes by
rendering them into a z-buffer and tessellating the buffer
contents [10]. However, depending on the quality of the
depth meshes, there are situations where this simple ap-
proach leads to undesired results: First, if the depth meshes
contain outliers, they can easily corrupt the overall result
(figure 4, left). Next, using cameras with higher angular
deviation, their depth meshes may occlude depth informa-
tion from other cameras (figure 5, left). We implemented
depth value selection methods for screen mesh generation
that address these problems.

The first problem can be solved by using the median
depth value of all rendered depth meshes. This will reduce
the impact of outliers and result in reliable values. To re-
solve the second issue, we compute weights for each used
camera based on viewing angle and resolution, similar to
the weights computation for the blending field in [2]: The
smaller the viewing angle between a used camera \( C^i \) and the
novel view \( C_v \), and the higher the resolution of \( C^i \), the bet-
ter the depth estimation should be that \( C^i \) provides. There
are several strategies how to use these computed weights:
One could simply use the best weighted camera for depth
estimation or use the “nearest” camera (in terms of provided
depth values) whose weight exceeds a certain threshold \( p \).

For real-time performance, the methods described above
have been implemented using graphics hardware and
OpenGL. Here, we make use of the OpenGL extension for
texture arrays [9] which enables us to bind a bunch of tex-
tures to a single texture unit. Using such a texture array,
each depth mesh is rendered into an own layer of this array
from the novel view’s point of view. At this point, we also
store the per-pixel weights that will be used by the weighted
selection method. This can be done by using RGB float tex-
tures.

Next, we use the transform feedback extension [9] that
allows us to write the output of a vertex shader directly into
a vertex buffer object. With this extension, we generate the
3D screen mesh as illustrated in figure 6: A GLSL vertex
shader is fed with the texture array containing the rendered
depth meshes and a set of vertices describing the screen co-
ordinates of the screen mesh. These coordinates are used by
the shader to fetch depth information from all layers of the
texture array.

Then, the shader performs one of the selection methods
described above: The median filter tries to find the median
depth value among all valid depth values, i.e. that are not

![Figure 4. Left: A depth mesh contains outliers and
occludes an otherwise satisfying reconstruction of the
surface. Right: Choosing the median depth value reduces
the impact of outliers.](image)

![Figure 5. Left: Camera \( C_2 \) occludes depth informa-
tion from camera \( C_1 \) (red line). Right: Due to angular
weights, camera \( C_1 \) is used to reconstruct depth
information at surface point \( p_1 \).](image)
lying on the far plane. The weighted depth shader evaluates the weights provided and selects the best depth information depending on these weights and the selected strategy.

Having performed depth value selection, the shader constructs 3D vertices by using the provided 2D coordinates and the selected depth values. These vertices are then written into a vertex shader written in GLSL.

The computation of the blending weights is done similarly to [2]. For each vertex \( v_i \) of the screen mesh and each contributing camera \( C_j \), we compute three weights \( w_{a,j} \), \( w_{r,j} \) and \( w_{f,j} \) which represent the angular, resolution and field of view penalty, respectively.

For computation of \( w_{a,j} \) and \( w_{r,j} \), we use the formulas

\[
w_{a,j} = \left(1 - \frac{\arccos \left( \frac{(C-v_i) \cdot (C_j-v_i)}{|C-v_i||C_j-v_i|} \right)}{\pi} \right)^{e_a}
\]

\[
w_{r,j} = \left( \frac{|C - v_i|}{\max(|C - v_i|, |C_j - v_i|)} \right)^{e_r}
\]

Here, \( e_a \) and \( e_r \) are user-controlled exponents that may be used to enforce epipole consistency if required (compare with [2]).

For computing \( w_{f,j} \), we use a Hermite spline interpolation to make the weight of camera \( C_j \) fall off smoothly at the edges of the camera’s frustum. We define \((s_x, s_y) \in [-1; 1]^2\) to be the screen coordinates of the vertex \( v_i \) in the image plane of \( C_j \), and \( h(a, b, x) \) to be the Hermite interpolation function that interpolates \( x \) using thresholds \( a \) and \( b \) \((a \leq x \leq b)\) as defined in (1), so the formula for computing the field of view weight can be defined as done in (2):

\[
h(a, b, x) = -2t^3 + 3t^2 , \quad t = \frac{x-a}{b-a} \quad (1)
\]

\[
w_{f,j} = h(-b,-a,s_x) \cdot (1 - h(a,b,s_x)) \cdot h(-b,-a,s_y) \cdot (1 - h(a,b,s_y)) \quad (2)
\]

Figure 7 shows two examples for \( w_{f,j} \), computed for different values of \((a, b)\) and visualized as heightfields.

Using these weights, we compute a combined weight \( w'_j \) using formula (3) where \( \alpha \) and \( \beta \) are user-controlled weighting variables. This way, we assure that the global weight is dominated by the field of view part and always falls off to zero at any vertex \( v_i \) that is not seen by camera \( C_j \). Using appropriate values for thresholds \( a \) and \( b \), the weight from (3) can be used to fade out dark border areas of the image of camera \( C_j \).

\[
w_{ij} = w_{f,j} \cdot (\alpha \cdot w_{a,j} + \beta \cdot w_{r,j})
\]

The final weight \( w_{ij} \) of \( C_j \) for the vertex \( v_i \) is computed by “normalizing” \( w'_j \) such that it sums up to 1 with all other weights for \( v_i \). To avoid blurry visual results, we only set the \( k_i \) best ranked cameras for \( v_i \) to have a final weight greater than zero, as also done in [2]. So if we define an order \( \sigma \in S_n \) such that \( w'_{\sigma(1)} \geq w'_{\sigma(2)} \geq \cdots \geq w'_{\sigma(n)} \), the final weight is computed with

\[
w_{ij} = \left\{\begin{array}{ll}
\frac{w'}{\sum_{i=1}^{k_i} w'_{\sigma(i)}} & \text{if } w'_j \text{ is among the } k_i \text{ best cameras ,} \\
0 & \text{otherwise.}
\end{array}\right.
\]

The computation of the blending weights is implemented as a two-step algorithm using GLSL. In the first step, the weights \( w_{a,j} \), \( w_{r,j} \) and \( w_{f,j} \) are rendered into the different slices of a texture array. As input for the shaders, we render the screen mesh points computed in the mesh generation step from the novel view’s point of view. Here, each point of the mesh maps to a texel of the current texture slice, and the weights are written to the different channels of the texture. Each ranked camera \( C_j \) renders its weights into slice \( j \), so in the end this step takes \( n \) passes (where \( n \) is the number of available cameras).

In the next step, a second shader uses the values of the texture array to compute intermediate values, in particular the weights \( w'_j \) of the cameras and the sums \( \sum_{i=1}^{k_i} w'_{\sigma(l)} \) of the \( k_i \)-best cameras. These values are written into new textures.

In the final lightfield rendering step, these textures as well as the texture array are used to blend the different camera images together. When rendering the screen mesh for camera \( C_j \), a shader uses the information stored in the textures to compute \( w_{ij} \) for each vertex \( v_i \) of the screen mesh. The final weight is used as an \( \alpha \)-value for texture blending.
3. Camera Preselection

Using all captured images to reconstruct a novel view can be an expensive task. Therefore, a preselection step is introduced that selects the \( k \) best fitting cameras according to some ranking criterions. Only these cameras are then used in the local selection step to compute the blending field.

One main criterion for an optimal camera preselection is that the selected \( k \) cameras provide a novel image which is little different from the one using all \( N \) available cameras. This criterion is not fulfilled by known ranking algorithms when dealing with our medical datasets, as could be seen in figure 3. This is mainly because these algorithms do not respect the need for a broad coverage of the scene by the selected cameras which is crucial for a consistent visual reconstruction. This problem is illustrated in figure 8 on the left: Camera \( C_6 \) which is used to reconstruct the scene information on the right side will at some point be replaced by Camera \( C_3 \) when using known ranking criterions. This change is accompanied by a sudden “appearance” of the area covered by \( C_3 \) and a disappearance of the one covered by \( C_6 \), thus leading to visual “popping” of scene areas.

![Figure 8. Left: A known ranking approach may select cameras \( C_2 \), \( C_5 \) and \( C_6 \) for scene reconstruction. When moving left, \( C_6 \) will be replaced by \( C_3 \). Right: Cameras \( C_1 \), \( C_4 \) and \( C_5 \) can be used to cover the whole scene as seen from novel view \( C \).](image)

To avoid these effects, we developed an algorithm which selects cameras by maximizing the coverage of the area that is visible by the novel view \( C \) (see figure 8, right). The algorithm works by discretizing the scene geometry into a coarse set of grid points \( g \), lying in world space. From the novel view's point of view, the point set is a regular grid of points (see figure 9). For each camera \( C_j \), the set of points which can be seen by both \( C_j \) and \( C \) in sufficient quality is determined. Then, a minimal set of cameras is chosen that covers all points sufficiently.

3.1. Point Set Generation

The point set is recreated every frame to form a regular grid of points. It is created similar to the screen mesh generation process: The depth mesh is rendered into a depth buffer whose content is afterwards used to reconstruct the 3D point set. However, in this case the depth meshes of all available cameras have to be processed instead of just the \( k \) best ones. If we use a huge amount of cameras, we might get a performance bottleneck when rendering all triangles of all depth meshes. Instead, we choose a local set of points from each depth mesh that represents the surface of the mesh.

Creating the local point sets can be done by simply using the vertices of the depth meshes. However, depending on how they were created, depth meshes may not have a uniform vertex density. Therefore, for each camera \( C_j \), we resample the depth mesh in a preprocessing step by projecting a regular grid of 2D points onto the mesh. Resampling is again done using an algorithm similar to the screen mesh generation process: The depth mesh is rendered into a depth texture of size \( g_x \times g_y \) to get a depth print of the mesh. We then render the grid which is of the same resolution using this texture to reconstruct the world coordinates of the grid points. Using a GLSL shader, we render the resulting 3D points into a vertex buffer object using the aforementioned transform feedback mechanism and discard the points that do not lie on the depth mesh. Since the resolution of the texture can be chosen quite small, e.g. \( 16 \times 16 \) grid points, rendering the local point sets can be done in real-time.

3.2. Camera Set Determination

Once the point set has been created, we need to find our smallest set of cameras that covers the scene. For each grid point \( g_i \), the cameras that can see this point are determined by projecting \( g_i \) on their image plane. For each camera \( C_j \) of those cameras, a weight \( w_{ij} \) is computed that tells how well \( C_j \) is suited to reconstruct the environment of \( g_i \). The weights can be computed using the methods described in section 2.2.
After this is done, our problem of finding a smallest set of cameras can be described as a classical Set Cover problem [4]: Let \( M \) be the number of grid points, \( N \) the number of all cameras, \( w_i = \max_{j \in S} (w_{ij}) \) and \( p \in [0; 1] \) a user-defined quality threshold. A camera \( C_j \) properly covers a grid point \( g_i \) if \( w_{ij} \geq p \cdot w_i \). Furthermore, let’s define \( S_j \) as the set of grid indices of the points that are covered by camera \( C_j \):

\[
S_j = \{ 1 \leq i \leq M | w_{ij} \geq p \cdot w_i \}. \tag{4}
\]

Then, we are looking for the smallest set of cameras \( T \), where \( T \subseteq \{1, \ldots, N\} \) such that \( \bigcup_{t \in T} S_t = \{1, \ldots, M\} \).

The Set Cover problem in general is \( \mathcal{NP} \)-complete. This problem cannot be relaxed in our case since we cannot reliably make any assumptions about the connectivity of the covered grid points \( g_i \), as can be seen in figure 10 (left). However, an algorithm of exponential run-time is not a pleasing option. Therefore, we implemented an algorithm which is similar to the standard approximation algorithm for the Set Cover problem [4], but only takes \( n \) selection steps (where \( n \) is the number of cameras to choose). The algorithm basically follows the greedy principle: In every step, we choose the camera \( C_j \) that covers most of the grid points which are still uncovered. The coverage of \( C_j \) is weighted using the quality criterion described in (4).

Depending on the quality threshold \( p \) and the number \( n \) of cameras to choose, there may be cases where a complete coverage of the scene is not possible. Figure 10 (right) shows such a case where only four cameras have been selected to reconstruct the scene. In such cases, it is desirable that the reconstruction focuses on the screen center of the novel view such that at least a consistent view of the core part of the scene is possible. So the weighting of a camera \( C_j \) also has to take the screen position of the grid points covered by \( C_j \) into account.

Let \( g'_i \in [0; 1]^2 \) the projection of the grid point \( g_i \) on the image plane of the novel view \( C \). Then we introduce a function \( d : [0; 1]^2 \rightarrow \mathcal{R} \) that weights \( g'_i \) according to its screen position. There are many possible implementations for \( d \), in our case we use the density function of the Gaussian distribution:

\[
d(x, y) = \frac{1}{2\pi\sigma^2} \exp \left[ -\frac{1}{2\sigma^2} \left( (x - 0.5)^2 + (y - 0.5)^2 \right) \right]
\]

Per default, we set \( \sigma = 0.5 \). The smaller \( \sigma \) is chosen, the higher is the penalty for a camera that represents the “border” of the novel view’s field of view.

There also might be cases where the scene can be covered by less than \( n \) cameras. In these cases, one approach could be to stop camera preselection at that point in order to reduce computational complexity at the local selection step. However, using the covering algorithm alone does not necessarily provide the best visual results when creating novel views, e.g. if areas of the scene are covered only by one camera with high angular deviation. Therefore, it is useful to select the remaining cameras from another ranking approach to add valuable information to the following reconstruction steps.

![Figure 10. Left: Camera C covers a set of grid points that are not necessarily connected directly. Right: Although the cameras are chosen correctly according to the weighting algorithm, the reconstructed scene may contain visible holes.](image)

**Algorithm 1 Camera preselection using the greedy principle, weighted with the distance to the screen center of the novel view.**

**Require:** \( n, N, M, S_1, \ldots, S_N \)

**Ensure:** \( T \)

1. \( U \leftarrow \{1, \ldots, M\} \)
2. \( T \leftarrow \emptyset \)
3. \( i \leftarrow 0 \)
4. while \( (U \neq \emptyset) \land (i < n) \) do
5. \( j \leftarrow \text{choose } j \in \{1, \ldots, N\}, \) such that \( \sum_{k \in (S_j \cap U)} d(g'_k) \) maximal
6. \( U \leftarrow U \setminus S_j \)
7. \( T \leftarrow T \cup \{j\} \)
8. \( i \leftarrow i + 1 \)
9. end while
10. while \( (i < n) \) do
11. \( j \leftarrow \text{choose } j \in \{1, \ldots, N\} \) from standard approach
12. \( T \leftarrow T \cup \{j\} \)
13. \( i \leftarrow i + 1 \)
14. end while

The whole approach described so far is summed up in algorithm 1. The first part describes the covering of the scene. If coverage is complete and there are still cameras to choose from, additional cameras are added using a standard approach (e.g. the original ranking approach from Evers-Senne and Koch [6]).

It must be pointed out that the set \( T \) may contain different cameras even in consecutive frames when changing the point of view, resulting in lots of visible “updates” of the
current view. Therefore, we use the current results of the algorithm only if the number of changed cameras in $T$ lies above a certain threshold.

### 3.3. Combination of Ranking Algorithms

As mentioned in the previous section, it is not guaranteed that covering the scene with as few cameras as possible provides a satisfying visual quality of novel views. Also, limited by the discretization of the scene by using a regular grid, it is not guaranteed that a scene reconstruction where all grid points are covered is closed (see figure 11). In these cases, we again want to assure that at least the screen center is correctly reconstructed. For that purpose, we may select cameras that specifically take care about the screen center before the main covering algorithm starts. This can in turn be done by applying the set of grid points.

Let $G_c \subseteq \{1, \ldots, M\}$ the set of grid indices of order $|G_c|$ whose projected screen positions in the image plane of the novel view $C$ are closest to the screen center. The order of $G_c$ can be small, like $|G_c| \leq 4$. Then for each camera $C_j$ and each grid index $i \in G_c$ we can compute a weight $w_{ij}$ saying how well $C_j$ is suited to reconstruct the environment of point $g_i$. Computing the weights can again be done using methods described in section 2.2. Then, the $n_c$ best ranked cameras are chosen to reconstruct the screen center. Finally, the coverage algorithm can be used to select the remaining cameras needed to cover the rest of the scene. It has to be clarified that it is up to the successive local selection steps which of these cameras are actually used to reconstruct the screen center.

As an alternative, it is also possible to use the original algorithm of Evers-Senne and Koch in order to select the $n_c$ best ranked cameras. Since this method tends to give high weights to cameras usable for reconstructing central parts of the scene, one approach could be to select the first $n_c$ cameras using this method and the remaining ones using the covering algorithm.

### 4. Results

Figure 12 shows some results of the blending weights computation of our local selection method. When setting $a = b = 1$ for the Hermite interpolation (which is equal to disabling it), the blending weights $w'_{ij}$ computed from $w_{a,j}$ and $w_{r,j}$ already provide results comparable to the original approach (compare with figure 1). As expected, there are also still intensity artefacts noticeable. These can be successfully reduced by also using the field of view weights $w_{f,j}$. The right side of figure 12 shows the result of the scene when setting $a = 0, b = 1$ for the Hermite interpolation.

In figure 13, the result of the coverage algorithm is compared to the original ranking approach from [6]. Due to the given camera path and the ranking algorithms of the original approach, the cameras selected for reconstructing the scene are clustered in the center of the screen. Using our approach, the resulting set of cameras provides an almost complete coverage of the same scene.

Table 1 shows performance results of the camera preselection process from a data set with 166 original views. The column “ESK (orig.)” shows the times using the implemented approach from Evers-Senne and Koch, “ESK (novis)” means a simplified algorithm that does not evaluate the visibility criterion. The columns “Coverage $(X \times Y)$” represent the results of our scene covering approach as described by algorithm 1, where $(X \times Y)$ is the number of grid points used for representing the scene. For our tests, we implemented a hardware-accelerated version of the method from Evers-Senne and Koch, where we used occlusion queries [9] to compute the weights for the visibility criterion. All measurements were done on a Intel Core2 4300 using a GeForce 8600 graphics processor.

It can be seen that our approach works quite independent of the number of cameras that are actually used to reconstruct the novel view, as does the reduced original approach. At small grid sizes, our approach is faster than the original one, although it does not reach the performance of the reduced approach. When choosing bigger grid sizes, our algorithm gets slower as it spends a lot of time solving the
By using depth meshes, the described algorithms could be implemented efficiently using graphics hardware and OpenGL. As an alternative, depth maps potentially provide more depth information than depth meshes, and can be used directly as input textures for the different ranking algorithms. Using depth maps for the methods described in this paper in general results in lower performance than using depth meshes. However, as graphics hardware experiences rapid improvements in speed and texture functionality, for future investigations it may be a reasonable decision to switch to depth maps as the main source for depth information.

### References