Multi-Sensor Panorama Fusion and Visualization

Karsten Scheibe[1], Martin Scheele[1], and Reinhard Klette[2]

[1]Optical Information Systems, German Aerospace Center, Berlin, Germany
[2]Center for Image Technology and Robotics, The University of Auckland
Auckland, New Zealand

Abstract. The paper describes a general approach for scanning and visualizing panoramic (360°) indoor scenes. It combines range data acquired by a laser range finder with color pictures acquired by a rotating CCD line camera. The paper describes coordinate systems of both sensors, specifies the fusion of range and color data acquired by both sensors, and reports on three different alternatives for visualizing the generated 3D data set. Compared to earlier publications the recent approach also utilizes an improved method for calculating the spatial (geometric) correspondence between laser diode of the laser range finder and the focal point of the rotating CCD line camera. Calibration is not a subject in this paper; we assume that calibrated parameters are available utilizing a method as described in [12].
Keywords: panoramic imaging, range finder, rotating line camera, sensor fusion, 3D visualization.

1 Introduction

Laser range finders (LRFs) have been used for close-range photogrammetry (e.g. acquisition of building geometries) for several years, see [11, 13]. An LRF which is based on the frequency to distance converter technique has a sub-millimeter accuracy for sensor-to-surface distances which are between less than one meter and about 15 meters, and an accuracy of 3 to 4 mm for distances less than 50 meters. It also captures intensity images. However, we are interested in true-color surface textures.

In earlier publications [7] we demonstrated how to fuse LRF data with pictures (i.e., colored surface texture) obtained by a rotating CCD line sensor (which we call the *camera* in this paper). Both devices are independent systems and can be used separately. To be precise, three CCD lines (i.e., for the red, green and blue channel) capture a color picture, and the length of these lines is in the order of thousands of cells (pixels), producing pictures of several gigabytes during a 360° scan.

The fusion of range data and pictures for 3D scene rendering is a relatively new development; see e.g. [9] for combining range data with images acquired by a video camera. Combinations of panoramic images [1] and LRF data provide a new technology for high-resolution 3D documentation and visualization. Fusion of range data and panoramic images acquired by a rotating line camera has been

discussed in [7, 10]. Calibration of range sensors [7] and of rotating line cameras [8] provide necessary parameters for this process of data fusion. In this paper we assume that parameters have been calibrated, and we do not elaborate on calibration details, but provide a brief indication of related calibration needs.

The main subject of this paper is a specification of coordinate transformations for data fusion, and a discussion of possible ways of visualizations (data projections). Applications are the generation of orthophotos, interactive 3D animations (e.g., for virtual tours), and so forth. Orthophotos are pictorial representations of orthogonal mappings of textured surfaces onto specified planes (also called *orthoplanes*). High-accuracy orthophotos are a common way of documenting existing architecture. Range data mapped into an orthoplane identify an *orthosurface* with respect to this plane.

Note that acquired range data (for one LRF viewing position) provide (limited) 2.5 D surface data only, and full 3D surface acquisitions can only be obtained by merging of data from several LRF viewpoints.

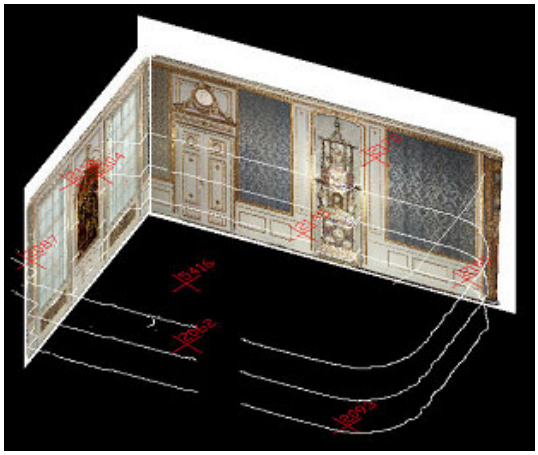

Fig. 1. A (simple) 3D CAD model consisting of two orthoplanes.

The approach in [7] addresses multi-view data acquisition. It combines several LRF data sets with several camera data sets by mapping all data into specified orthoplanes. This simplified approach utilizes a 2.5 D surface model for the LRF data, and no complex ray tracing or volume rendering is needed. This simplified approach assumes the absence of occluding objects between LRF (or camera) and orthosurface. In a first step we determine the viewing direction of each pixel of the camera (described by a formalised sensor model) towards the 2.5 D surface sampled by the LRF data. This can be done if both devices are calibrated (e.g., orientations of the systems in 3D space are known in relation to one world coordinate system) with sufficient accuracy. Requirements for accuracy

are defined by the desired resolution in 3D scene space. Orientations (i.e., affine transforms) can be specified using control points and standard photogrammetry software. The 2.5 D model of orthosurfaces is generated by using several LRF scans to reduce the influence of shadows. More than a single camera viewpoint can be used for improved coloration (i.e., mapping of surface texture). Results can be mapped into several orthoplanes, which can be transformed into a unified 3D model in a second step. See Figure 1.

In this paper we discuss a more advanced approach. For coloration of an extensive 3D point cloud, generated from several LRF data sets, we use captured panoramic images obtained from several camera scans. This requires an implementation of a complex and efficient raytracing algorithm for an extremly large data set. Note that this raytracing cannot assume ideal correspomndences between points defined by LRF data and captured surface texture; we have to allow assignments within local neighborhoods for identifying correspondences between range data and surface texture. Figure 2 illustrates this problem of local uncertainties.

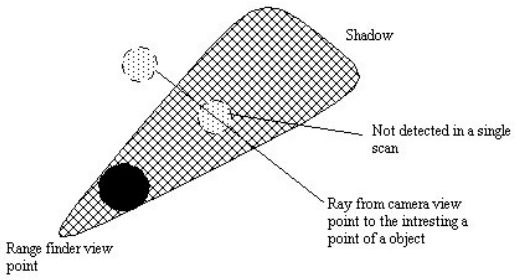

Fig. 2. Raytracing problem when combining one LRF scan with data from one camera viewpoint.

There are different options to overcome this problem. A single LRF scan is not sufficient to generate a depth map for a complex 3D scene. Instead of fusing a single LRF scan with color information, followed by merging all these fused scans into a single 3D model, we prefer that all LRF scans are merged first into one unified depth representation of the 3D scene, and then all camera data are used for coloration of this unified depth representation. Of course, this increases the size of data sets extremely, due to the high resolution of LRF and camera. For simplification of raytracing, the generated 3D points can be first used to create object surfaces by triangulation, applying standard routines of computer graphics. This can then be followed by raytracing, where the parametrization obtained in triangulation (including routines for simplification) reduces the size of the involved sets of data.

LRF and camera will have different viewpoints or positions in 3D space, even when we attempt to have both at about the same physical location. A simple approach for data fusion could be as follows: for a ray of the camera map the picture values captured along this ray onto a point P calculated by the LRF if P is the only point close (with respect to Euclidean distance) to this ray. An octreee data structure can be used for an efficient implementation. However, this simplified approach never colorizes the whole laser scan, because surface edges or detailed structures in the 3D scene always create very dense points in the LRF data set.

As a more advanced approach assume that we are able to arrange that the main point of the LRF and the projection center of the camera are (nearly) identical, and that orientations of both rotation axes coincide, as well as of both optical axes. Then processing of the data is straightforward and we can design rendering algorithms that work in (or nearly in) real time. Intensity (depth) data of the LRF will be simply replaced by color information of the camera. No ray tracing algorithm is necessary for this step because occlusions do not need to be considered. The result is a colored 3D point cloud in world coordinates. Nevertheless, to model the data it is necessary to triangulate the LRF points into a mesh (because LRF rays and camera rays do not ideally coincide). A triangulation reduces the number of points and makes it possible to texture the mesh. Note that using this approach the same shadow problem can occur as briefly discussed above for single LRF scans.

This more advanced approach requires to transform the panoramic camera data into the LRF coordinate system. In order to cover the 3D scene completely, several scans are actually required from different viewpoints, which need to be merged to create a 3D mesh (also called *wireframe*). Points obtained from one LRF scan are connected with points obtained from another LRF scan. In this case the advantage of unique ray-to-ray assignments (assuming aligned positions and directions of LRF and camera) is lost. It is again necessary to texture a 3D wireframe by data obtained from different camera viewpoints (i.e., a raytracing routine is again required). We describe a time-efficient raytracing approach for such a static texturing situation in this paper. We report about advantages of applying independent LRF and camera devices, and illustrate by examples (e.g., results of the "Neuschwanstein project").

The Neuschwanstein project is directed on a complete 3D photogrammetric documentation of this Bavarian castle. Figures in this paper show the Thronsaal of this castle as scanned from the viewpoint (i.e., LRF and camera in about the same location) about at the center of the room. Of course, a more complete photogrammetric documentation used more viewpoints to reduce the impact of "shadowed areas". The paper describes all transformations and algorithms applied in this process.

2 Coordinate Systems

LRF and camera scans are in different independent coordinate systems. To fuse both systems it is necessary to transform the data into one primary reference system, called the world coordinate system.

Rays of the panoramic camera are defined by image rows i (i.e., this is the horizontal coordinate) and pixel position j in the CCD line (i.e., this is the vertical coordinate). Similarily, we identify rays of the LRF by an index i and a constant angular increment φ_0 which defines the absolute horizontal rotation angle $\varphi = i \cdot \varphi_0$, and an index j and an angle increment ϑ_0 which defines the absolute vertical angle $\vartheta = j \cdot \vartheta_0$. Note that these absolute angles are also the same for the panoramic camera. However, the possible range of vertical angles of the camera is typically reduced compared to that of a LRF; the possible range of horizontal angles of the LRF is typically reduced compared to that of a panoramic camera.

2.1 LRF

A LRF scans in two dimensions, vertically by a deflecting mirror and horizontally by rotating the whole measuring system. The vertical scan range is 310° (which leaves 50° uncovered), and the horizontal scan range is 180°. The LRF scans overhead, therefore a whole sphere will scanned with one 180° moving. Fig. 3 depict a LRF raw data set and the not calibrated image.

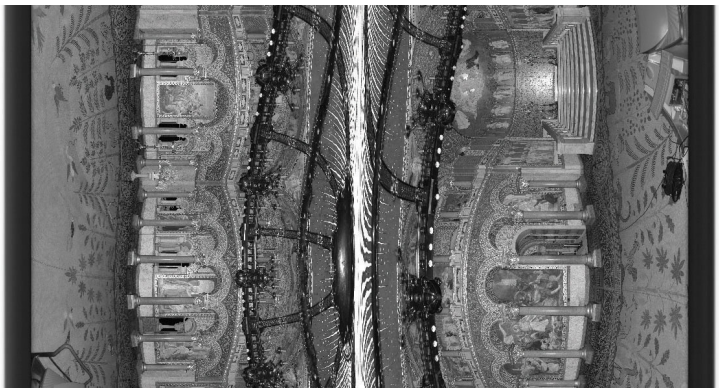

Fig. 3. Raw data of the not calibrated LRF image.

Rays and detected surface points on these rays (which define the LRF data set) can be described in a polar coordinate system. According to our application of the LRF, it makes sense to transform all LRF data at one view point into a normal polar coordinate system with an horizontal range of of 360° and a

vertical range of 180° only. At this step all LRF calibration data are available and required.

The photogrammetry specify for rotating measuring devices, e.q theodolite systems, how to measure the errors along the rotating axis. They called vertical and horizontal collimation errors. The pole columns describe the column around the zenith, the highest point in the image. To determine the collimation errors, typically the to measured point will dedicated by two sites. That means the point will measured in two steps, first measured in site one, than the both rotation axis will turned by 180° and the point will measured again[5]. Fig. 4 depict the optical Z-axis an orthogonal axis to the corresponded horizontal rotation axis and the tilt-axis, the vertical rotation axis K.

The horizontal and vertical collimation errors are calculated by determining the pole column (this can be done in the LRF image based on two rows [layers] and identical points at the horizon). This provides the offset to the zenith and to the equator (i.e., the horizon). Secondly the horizontal collimation error can be calculated by control points along the equator. The vertical collimation error can be determined based on these results. As an important test we have to confirm that the zenith is uniquely defined in 3D space for the whole combined scan of 360°.

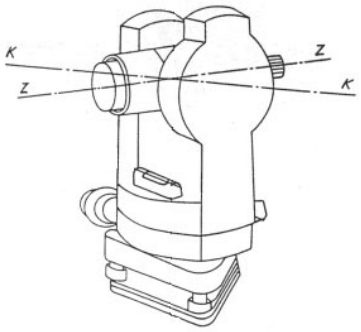

Fig. 4. Theodolite with two axes: the terms 'Zielachse' and 'Kippachse' in (German) photogrammetry specify the optical Z-axis and an orthogonal K-axis. A range finder measures along a variable Z-axis, which may be effected by horizontal (i.e., along the Z-axis) or vertical (i.e., along the K-axis) errors.

Each point in the LRF coordinate system is described in polar or Cartesian coordinates as a vector $\overrightarrow{p}$, which is defined as follows:

$$
\begin{aligned}
p_x &= R \cdot \sin\vartheta \cdot \cos\varphi \\
p_y &= R \cdot \sin\vartheta \cdot \sin\varphi \\
p_z &= R \cdot \cos\vartheta
\end{aligned}
$$

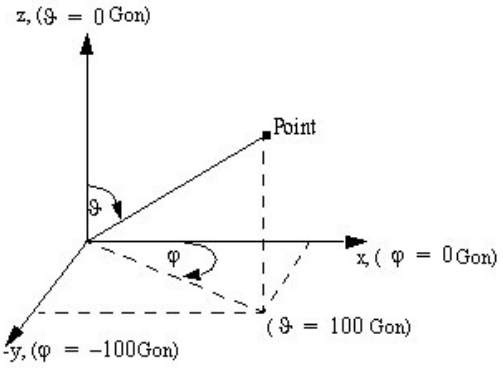

Fig. 5. Range finder xyz-coordinate system: the Z-axis of Fig. 4 points towards p, and is defined by slant ϑ and tilt φ.

The orientation and position with respect to a reference vector $\overrightarrow{r}$ in the world coordinate system is defined by one rotation matrix $\mathbf{A}$ and a translation vector $\overrightarrow{r_0}$:

$$\overrightarrow{r} = \overrightarrow{r_0} + \mathbf{A} \cdot \overrightarrow{p} \tag{1}$$

All coordinate systems are right hand systems. The laser scanner rotates clockwise. The first scan line starts at the positive y-axis in the LRF system at the horizontal angle of 100gon[1] The rotation matrix combines three rotations around all three axes for the right hand system:

$$\mathbf{A} = \mathbf{A}_\omega \cdot \mathbf{A}_\phi \cdot \mathbf{A}_\kappa \tag{2}$$

The resulting matrix $\mathbf{A}$ is then given as

$$\begin{pmatrix} C\varphi \cdot C\kappa & S\varphi \cdot S\kappa & S\varphi \\ C\omega \cdot S\kappa + S\omega \cdot S\varphi \cdot C\kappa & C\omega \cdot C\kappa - S\omega \cdot S\varphi \cdot S\kappa & -S\omega \cdot S\varphi \\ S\omega \cdot S\kappa - C\omega \cdot S\varphi \cdot C\kappa & S\omega \cdot C\kappa + C\omega \cdot S\varphi \cdot S\kappa & C\omega \cdot C\varphi \end{pmatrix}$$

where κ, ϕ, ω are the rotation angles around the z-, y-, and x-axis, respectively, and C stands short for the cosine and S for the sine.

2.2 Camera

The panoramic image sensor is basically a rotating CCD line sensor. Three CCD lines are mounted vertically and also rotate clockwise. The scanned data are stored in cylindrical coordinates. In an ideal focal plane each pixel of the

[1] The unit gon is defined by $360^\circ = 400$gon.

combined (i.e., all three color channels) line is defined by the vector $\overrightarrow{r_d}$. The rotation axis of the camera is incident with the main point of the optics. The focal plane is located at focal length f, without any offset $\overrightarrow{\Delta}$. Scans begin at the horizontal angle of 100gon. We have the following:

$$\overrightarrow{r_d} = \begin{pmatrix} r_{dx} \\ r_{dy} \\ r_{dz} \end{pmatrix} = \begin{pmatrix} 0 \\ f \\ j \cdot \delta \end{pmatrix} \tag{3}$$

The used CCD line had a length of approximately 70mm and it had 10,296 pixels with a pixel size $\delta = 7\mu m$ indexed by j. Each scanned surface point is identified by the camera rotation $\mathbf{A}_\varphi$. In analogy to the LRF, a reference vector (in world coordinates) for the camera coordinate system is described by the rotation matrix $\mathbf{A}$ as follows:

$$\overrightarrow{r} = \overrightarrow{r}_0 + \mathbf{A} \cdot \lambda \cdot \mathbf{A}_\varphi \cdot \overrightarrow{r}_d \tag{4}$$

λ is an unknown scale factor of the camera coordinate system (for the 3D scene). If the LRF and the camera systems have the same origin, then λ correspondents to the distance measured by the laser scanner. We also model the following deviations from an ideal case:

- The CCD line is tilted by three angles regarding the main point ($\mathbf{A}_I$).
- The CCD line has an offset vector regarding the main point ($\overrightarrow{\Delta}$).
- The optical axis is rotated regarding the rotation axis ($\mathbf{A}_O$).

These deviations are depicted in Figure 6 and described in the following equation:

$$\overrightarrow{r} = \overrightarrow{r_0} + \lambda \cdot \mathbf{A}\mathbf{A}_\varphi\mathbf{A}_O \left(\mathbf{A}_I \begin{pmatrix} 0 \\ 0 \\ j \cdot \delta \end{pmatrix} + \begin{pmatrix} \Delta_x \\ f + \Delta_y \\ \Delta_z \end{pmatrix} \right) \tag{5}$$

For the calculation of calibration parameters $\mathbf{A}_{opt.}$, $\mathbf{A}_{in}$ and the offset $\overrightarrow{\Delta}$, see [7].

3 Fusion

Fusion of the data sets starts with transforming both coordinate systems (i.e., LRF and camera coordinate systems) into one world coordinate system. For this step the orientation of both system needs to be known. A transformation of LRF data into the world coordinate system is then simple because all required parameters of the equation are given. It is only necessary to determine the orientation to the world coordinate system as shown in Equation 1. For the camera data the parameter λ is unknown, which can be estimated by Equations 1 and 5. By applying all parameters of the interior and external orientations to the vector $\overrightarrow{r_d}$ the following simplified equation results:

$$\overrightarrow{r} = \overrightarrow{r_0} + \lambda \cdot \overrightarrow{r_d} \tag{6}$$

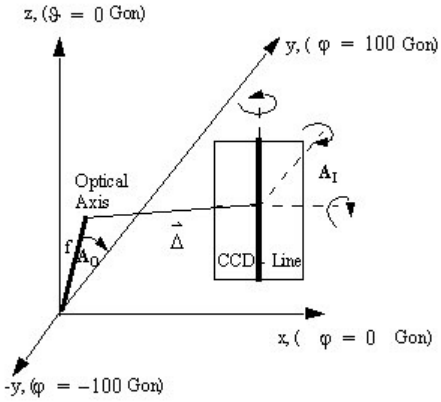

Fig. 6. Rotating line camera xyz-coordinate system: the effective focal length f defines the position of an image column (i.e. position of the CCD line at this moment) parallel to the z-axis, with an assumed offset Δ for the center of this image column.

and therefore

$$\begin{pmatrix} r_x - r_{0x} \\ r_y - r_{0y} \\ r_z - r_{0z} \end{pmatrix} = \lambda \cdot \begin{pmatrix} dx \\ r_{dy} \\ r_{dz} \end{pmatrix} \tag{7}$$

and λ f.i.

$$\lambda = \frac{r_y - r_{0y}}{r_{dy}} \tag{8}$$

But the ray between a pixel and a 3D point in the LRF data set can be disturbed by an object, and a raytracing routine has to check whether the LRF point can be colored properly.

We applied an LRF and camera setup which allows to center both main points in such a way that we are able to map any LRF point or camera ray into the world coordinate system. Equations (1) and (5) reduced by the term $\overrightarrow{r_0} + \mathbf{A}$ are described in the following equation:

$$\overrightarrow{p} = \lambda \mathbf{A}_\varphi \mathbf{A}_O \left(\mathbf{A}_I \begin{pmatrix} 0 \\ 0 \\ j \cdot \delta \end{pmatrix} + \begin{pmatrix} \Delta_x \\ f + \Delta_y \\ \Delta_z \end{pmatrix} \right) \tag{9}$$

By applying all parameters of the interior orientations to the vector $\overrightarrow{r_d}$ the following simplified equation results. $\overrightarrow{r_d}$ now describes the viewing direction of each pixel like in an ideal focal plane as shown in Equation (4), and we obtain the following:

$$\overrightarrow{p} = \lambda \cdot \mathbf{A}_\varphi \cdot \overrightarrow{r_d} \tag{10}$$

Note that λ corresponds to the distance R of the LRF to the scanned point. $\mathbf{A}_\varphi$ contains the rotation angle φ and represents an image column i. The transformed vector represents the image row j and the number of the pixel in the CCD line. Therefore each point in the LRF coordinate system has an assigned pixel value in the panoramic image. Figure 7 depicts a flipped open sphere. Horizontal coordinates represent angle φ and vertical coordinates the angle ϑ of the LRF coordinate system.

Fig. 7. Panoramic image data have been fused in a subwindow of the shown range image. (The figure shows the Thronsaal of castle Neuschwanstein.)

4 Visualization

4.1 Projection

All projections are implemented with OpenGL. Generally, OpenGL is an interface which stores all transformations in different types of matrixes. All other important information can be saved in arrays (e.g., object coordinates, normal vectors and texture coordinates). The rendering engine multiplies all matrices to a transformation matrix and transforms each object coordinate by multiplying the current transformation matrix with the vector of the object coordinate. Different kinds of matrixes can be stored in stacks to manipulate different objects by different matrices. The main transformation matrix $\mathbf{M}_T$ is give as follows:

$$\mathbf{M}_T = \mathbf{M}_V \cdot \mathbf{M}_N \cdot \mathbf{M}_P \cdot \mathbf{M}_M \tag{11}$$

$\mathbf{M}_V$ is the view port matrix, which is the transformation to the final window coordinates. $\mathbf{M}_N$ is the normalization matrix of the device coordinates, $\mathbf{M}_P$ the projection matrix and $\mathbf{M}_M$ the matrix to transform model coordinates (e.g., a rotation, scaling, or translation).

4.2 Central Projection

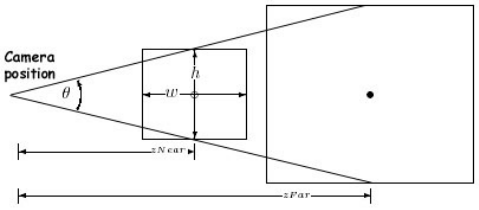

Fig. 8. Central projection of objects in the range interval $[Z_N, Z_F]$ into a screen (or window) of size $w \times h$.

Now we consider central projection of an object or scene on a perspective plane. The actual matrix for projection is the matrix $\mathbf{M}_P$ and results for following dependencies Fig. 8 and Equ. 12. In Figure 8 the clipping planes are drawn as zFar the clipping at the fare point which we will used as symbol Z_F and the clipping plane zNear in the nearest point to the camera used as symbol Z_N. The clipping planes can be used as a boundary box to select the scene depth.

$$\mathbf{M}_p = \begin{pmatrix} \cot\frac{\theta}{2} \cdot \frac{h}{w} & 0 & 0 & 0 \\ 0 & \cot\frac{\theta}{2} & 0 & 0 \\ 0 & 0 & \frac{Z_F+Z_N}{Z_F-Z_N} & -\frac{2\cdot Z_F \cdot Z_N}{Z_F-Z_N} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{12}$$

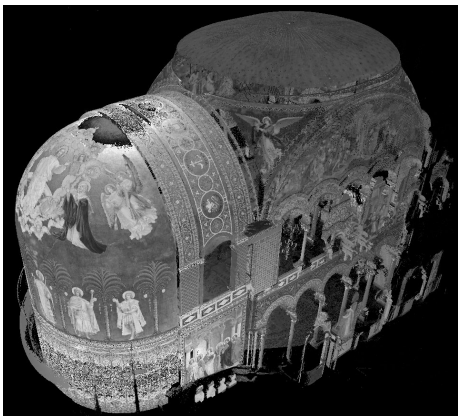

Fig. 9. Central projection of the same hall shown in Fig. 7.

All matrixes in OpenGL can be set comfortably by functions. The figure 9 depict a 3D model rendered central perceptively based on image data of Fig. 7. In this case the colored information are ignored.

4.3 Orthogonal Projection

The orthogonal projection is the projection of each point orthogonally to the plane independent from the viewing point. Figure 10 and Eq. 13 depict the dependencies.

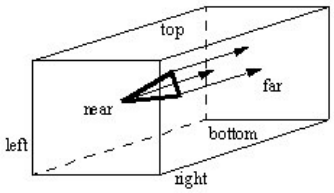

Fig. 10. Orthogonal parallel projection: the screen (window) can be assumed at any intersection coplanar to the front (or back) side of the visualized cuboidal scene.

$$\mathbf{M}_P = \begin{pmatrix} \frac{2}{R-L} & 0 & 0 & \frac{R+L}{R-L} \\ 0 & \frac{2}{T-B} & 0 & \frac{T+B}{T-B} \\ 0 & 0 & \frac{2}{F-N} & \frac{F+N}{F-N} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

In the mostly case high resolution orthophotos as final product stored in an common file format independent from the resolution of the viewport of OpenGL are necessary. The first step is to determine the attitude parameter of the orthoplane. This can be done in the 3D model. A correction of the attitude can included in this step. This means that in the most cases a ceiling or a panel, the xy-plane, or a wall, the xz-plane, is parallel to the corresponding plane in the world coordinate system. The Eq. 1 expand by the parameter $\mathbf{A}_{ortho}$ the attitude of the orthoplane and a factor for the resolution t is shown in the following Eq. 14. In this case are both systems are already fused to on common image. The "3D" coordinates with the appendant color information are established.

$$\overrightarrow{o} = t \cdot \mathbf{A}_{ortho} \cdot (\overrightarrow{r_0} + \mathbf{A} \cdot \overrightarrow{p}) \tag{14}$$

o_x and o_z is the position in the orthoplane. If is necessary o_y can saved as the altitude in the orthosurface. The digital surface model (DSM) can be used to generate orthophotos from independent cameras or if necessary to scan without the mechanical fixes, described in the Introduction. The Fig. 11 and Fig. 12 depict the dependencies and a grey coded surface model.

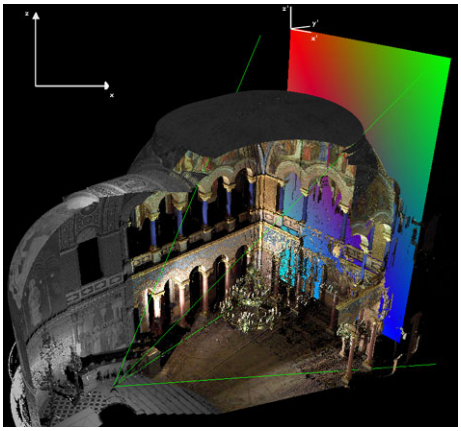

Fig. 11. A defined ortho plane 'behind' the generated 3D data.

Fig. 12. Gray-value encoded and orthogonally projected range data of those surface points which are in 2 meter distance to the defined (see Fig. 11) orthoplane.

4.4 Stereo Projection

The model view can be modified by changing the matrix $\mathbf{M}_V$, so that the 3D object can rotate or translate in any direction. The camera view point also can modified. It is possible to go into the 3D scene and looking around. Further more it is possible to render more than one viewpoint in the same rendering context and create anaglyph stereo pairs by this way. There are a couple of methods of setting up a virtual camera and rendering two stereo pairs, many methods are strictly incorrect since they introduce vertical parallax. An example of this is called the *toe-in method* (Fig.: 13), while incorrect it is still often used because a correct asymmetric frustum method requires features not always supported by rendering packages [3].

In the *toe-in* projection the camera has a fixed and symmetric aperture, each camera is pointed at a single focal point. Images created using the *toe-in method*

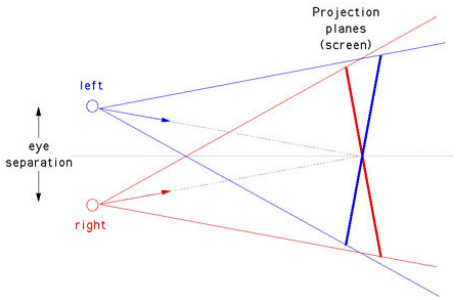

Fig. 13. (Incorrect) toe-in stereo projection.

will still appear stereoscopic but the vertical parallax it introduces will cause increased discomfort levels. The introduced vertical parallax increases out from the center of the projection plane and is more important as the camera aperture increases.

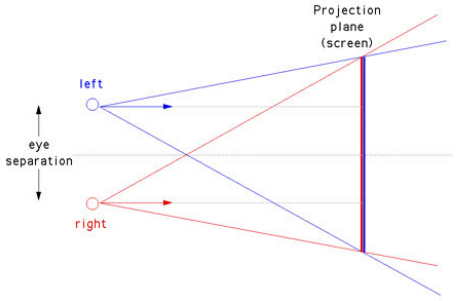

Fig. 14. Correct stereo projection based on asymmetric camera frustums.

The correct way to create stereo pairs is the "non symmetric frustum" method. It introduces no vertical parallax. It requires a non symmetric camera frustum, this is supported by some rendering packages, in particular, OpenGL.

4.5 Triangulation

In fig. 9 the measured points are shown. The high point density makes the pointcloud look like a surface. But the single points become visible, when viewing a close-up of the object. An other disadvantage of this representation is that modern graphic adapters with it is 3D acceleration only support a fast rendering

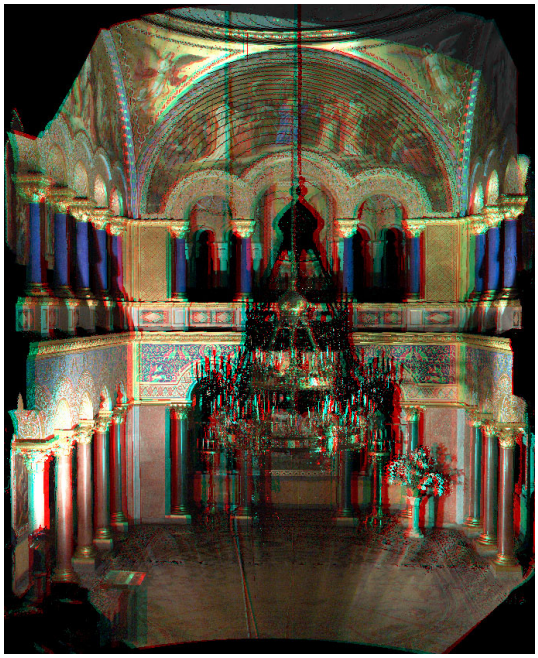

Fig. 15. Correct stereo projection of the same hall shown in Fig. 7; the anaglyph uses red for the left eye.

of triangles and triangle stripes. Polygons will tesselated[2] by the graphic adapter. To render triangles it is necessary to triangulate the single points to triangles.

Because of this, the pointcloud is converted in a triangle mesh. Therefore, a initial, dense mesh is generated. For the generation the incremental algorithm proposed by Bodenmueller [2] is used. Originally, this approach is developed for online proccessing of unorganized data from hand-guided scanner systems. But the method is also suitable for the processing of the laser scanner data we used, because it uses a sparse, dynamic data structure which can hold larger data sets and it is able to generate a single mesh from multiple scans. The following work flow explain the principal way for the triangulation.

- tinning of points (density check)
- normal approximation (local approximation of the surface)
- point selection (insert point, depend by normal and density)
- estimation of the euclidian neighborhood relationships
- neighborhood projecting to tangent plane (P[3D -> 2D])

[2] Tessellation is the splitting of polygons into triangles.

- new local triangulation (Delaunay[3])

A second important relationship is the connectivity between the triangles. This basic relationship is important for the most algorithm for example the calculation of triangle strips, shadows or to meshing the pre triangulated points. The following section described a fast way to do this.

4.6 Connectivity

The connectivity means, which polygon is connected by an other polygon. In the standard computer graphic and in the most cases it is not necessary to improve the algorithm to calculate the connectivity, because the most models have only a lots of polygons and it is a static pre calculation, mostly done by the initialization of the object. It is very easy to check every edge of a polygon with every all edges of all other polygons. In our case we have many millions of polygons. By implementation of the connectivity algorithm based on the Gamasutra's article[6] more than one hour was needed to calculate the connectivity. The idea is to hash the point indices to one edge index. The Fig. 16 illustrate how to hashing the edges. Every edge have two indices n, m. Important is by sorting the indices the first column represent the low index n and the higher is m. Every pair n, m have a uniqueness z. by pushing the n value in the higher part of a register and m in the lower part. Now we can sort the first column of our structure by z. Only one loop is enough to set the dependencies. If one row i and row $i+1$ have the same z the dependencies are directly given by the second and third column of our structure. The row three and four in Fig. 16 must have the same z and the connectivity is given by column two and three: Triangle one, side three is connected by triangle two, side one. With this algorithm we need for the same scan before only 10 seconds.

[3] In mathematics, and computational geometry, the Delaunay triangulation, for a set P of Points in the plane, is the triangulation DT(P) of P such that no point is inside the circumcircle of any triangle in DT(P).

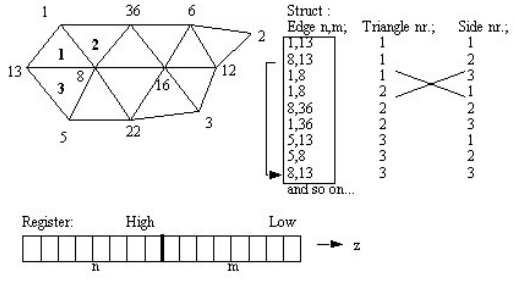

Fig. 16. Fast connectivity calculation of triangles.

4.7 Light and shading

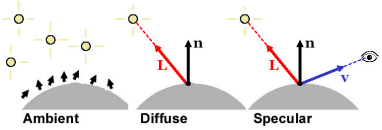

Fig. 17. Common light sources supported in OpenGL.

For a better visualization and a realistic impression it is necessary to shade a scene, calculating shadows, reflections and other light features. Shading means the surface shading of objects, not shadow calculations itself. Especially complex detailed structures have better depth impression, if they are shaded. Shading surfaces calculate how much light the surface is reflecting. Three kind of light contributions will supported. Ambient, diffuse and specular light. Fig. 17 depict the common light sources contributions. Ambient light is the generally background illumination, independent from the position of the light source and the view. Every object can have different material properties. The ambient properties is compared to the color of an material. A black surface will not reflect any color. In OpenGl is this a simplified calculation. The factor of a model property will multiply with the factor of the setting light source. If we have a white surface Ambient(1,1,1) and a blue light source Ambient(0,0,1) the reflected light by the surface will be blue. A green material Ambient(0,1,0) and a blue light source will reflect nothing, it seems to be black. The generally ambient factor in the global light model properties must be set to zero. Otherwise the reflected light from a green surface by a blue light source is a little bit green. This will simulate a light continuum. When we throwing light by a blue lamp to a green object it will always shining a little bit green. Sometime it‘s comfortable to set the color directly, but setting the material properties is more dynamical, if we switch the light sources. Diffuse light works also like the ambient light but additional the

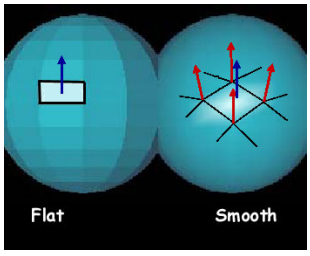

Fig. 18. Shading models in OpenGL.

dot product of the vertex[4] normal and the normalized vector from the vertex to the light source will considert.According to Lambert the effective light intensity is attenuated by the cosine of the angle between the direction of the light and the direction from the light to the vertex being lighted. The specular light source contribution is the product of the material specular reflectance, the light's specular intensity, and the dot product of the normalized vertex-to-eye and vertex-to-light vectors, raised to the power of the shininess of the material. Further more all three light source contributions are attenuated equally based on the distance from the vertex to the light source and on light source direction, spot exponent, and spot cutoff angle. This means the given factor, raised to the power of a spot exponent. Thus, higher spot exponents result in a more focused light source. If the light is positional, rather than directional, its intensity is attenuated by the reciprocal of the sum of a constant factor, a linear factor multiplied by the distance between the light and the vertex being lighted, and a quadratic factor multiplied by the square of the same distance. OpenGl supports generally two kinds of shading models, flat shading and smooth shading, also called gouroud shading. Flat shading of a vertex will set the diffuse and specular contributions to the whole vertex, no interpolation. In the smooth shading model every point of a vertex get a normal vector, this is the interpolated vector of the normals of all connected verticies. Fig. 18 illustrate the differences and Fig. 19 depict the unshaded and shaded Thronsaal.

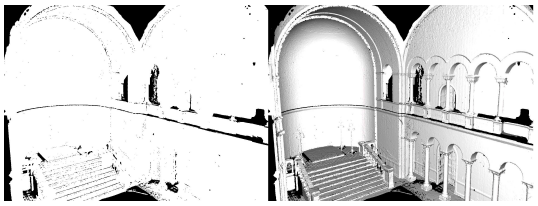

Fig. 19. Unshaded (left) and shaded (right) Thronsaal.

4.8 Shawdows

Shadow calculations help the viewer to manage the scene faster. The brain get a better and faster depth impression. One the other hand the scene looks more realistic. In the computer graphic are a couple of publications, how to render a scene with shadows. In the game industry many tricks was created to calculate really fast shadows. One example is the mirroring of objects to a plane, most to the floor. But this are only possible planar shadows. More difficult are volume shadows. In this case all objects in a shadow volume get shadows. The idea is to

[4] Vertex is a geometrical primitive (Triangle, Quad, Polygon etc.).

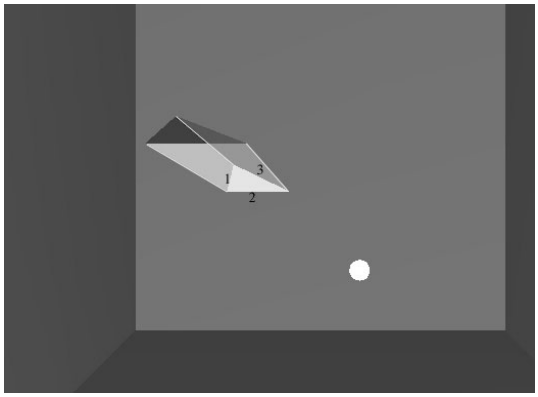

Fig. 20. Volume shadow by using the stecil buffer.

mask the scene by a special buffer. Verticies will only rendered if a special buffer is confirmed to a function (LESS, GREATER, EQAL etc.). Modern graphic adapter supports such buffer in hardware, the depth buffer and the stencil buffer are some of this. The next following steps describe how to use the stencil buffer for a fast volume shadowing. This approach was first introduced by F. Crow in 1977[4].

Render all verticies in the color buffer in consideration of the depth buffer. The color buffer is the final buffer we really see. The depth buffer represent the z-value (depth) of every rasterized pixel. Only one pixel in the same row and column, that with the nearest z-value to the view point must be drawn. The following next steps describe the shadwow algorithm.

- Render all front faces to the stencil buffer (stencil function increment)
- Render all back faces to the stencil buffer (stencil function decrement)
- Blend shadow over the whole scene where stencil is not equal

The culling function specifies, whether front- or back-facing facets can be culled. Front and back is dependent to the order of the vertex points, counter clockwise or clockwise, mostly counter clockwise. Fig. 20 depict the workflow. Our triangle are counter clockwise, therefore the drawn rectangle on edge one and three are also counter clockwise. This both rectangle increment the stencil buffer in the first pass. In the second pass the drawing is set to clockwise and the stencil function to decrement the stencil buffer. Only the rectangle on edge two will drawn (decrement the stencil buffer). Now the stencil buffer is masked correctly. Finally a rectangle over the whole view will be blended. Everywhere the stencil buffer is not equal a shadow will be blended. The result is the projected shadow and shadowed objects in the shadow volume. To increase the performance we can calculate the object silhouette. This can be done by calculate the connectivity of all faces, like explain in the section connectivity. Further more the visibility of a face must be calculated, because the faces of a volumetric object e.g. a cube are always connected. In this case the algorithm draw only not connected edges

of a face or when the face itself is not visibly. Of course, first the algorithm can check is the object itself visible or not to increase the performance.

4.9 Texturing

Texturing, or texturmapping is the mapping of our triangles with images. Each point coordinate (x,y,z) becomes a texture coordinate (i,j). Since we have small triangles it is not necessary to rectify the images. For bigger triangles first it is necessary to rectify the mapping part of the image, because the images are not in the same coordinate system and are perspectively. Secondly we must check that the mapped triangle was seen by camera. This is the discussed raytracing step in the introduction. Actually a typical raytracing check the camera ray with each triangle for collisions. A optimized data structure and octtrees increase the performance of the algorithm. The following approach describe how to mapping the texture very easily. This approach use the distance from every triangle to the camera position of the to mapped texture. Eq. 4 reduced by the term $\mathbf{A}\cdot\lambda\cdot\mathbf{A}_{\varphi}\cdot\overrightarrow{r}_d$ gives the current camera position $\overrightarrow{r}_0$. The orientation is not needed in this step. The distance results by Eq. 1 and the camera position to

$$\sqrt{(r_x - r_0x)^2 + (r_y - r_0y)^2 + (r_z - r_0z)^2}. \tag{15}$$

Now all triangles, sorted by the nearest distance, will textured and masked in the image. If a new texture coordinate already masked in the image they will not mapped. This is also a statical procedure, all texture coordinates will pre calculated and stored in a list.

5 Concluding Remarks

This paper introduce a algorithm, how to fuse laser scanning data with images of a rotating line camera. The coordinate systems of both systems and the transformation of both data sets in one common reference system (world coordinate system) are described.

The visualization of the data and different possibilities of projection are shown. For a more realistic view some light effects and shadow calculation are discussed. A fast connectivity algorithm as an base for many calculations in the computer visualization is introduced.

For fusing data of single scans, it seems to be very easily to do this with mechanical fixes and assumed that the main point of the laser scanner and the optical projection center of the panorama camera are identical. In this case the laser point and camera assignment are directly given. But our experiences was, that for bigger models with many scans we must resign our approach. The common way is, first to calculate the 3D model by using different laser scans and then mapping the color information. Triangulation and meshing of this huge unorganized pointclouds is a separate step. Many publication about this procedure are available. By using the approach from Tim Bodenmüller we

generate a initial dense mesh. In future work we must more simplify this mesh. Errors in the mesh e.g. holes must be found automatical.

Texturemapping with the panoramic data are shown. The radiometric problems by mapping different textures must also research in future work. A homogeneous lightning with different camera placements during the data acquisition is very difficult. Shadows caused by real light must be found automatical and must be modified radiometrical or masked for not using.

Acknowledgment: The authors thank R. Reulke for ongoing collaboration on the discussed subjects. Thank B. Strackenburg for supporting the projects.

References

1. R. Benosman and S.B. Kang, editors. *Panoramic Vision: Sensors, Theory, and Applications*. Springer, Berlin, 2001.
2. T. Bodenmueller and G. Hirzinger. Online Surface Reconstruction from Unorganized 3D-Points for the DLR Hand-guided Scannersystem. *Eurographics2004*, In Press.
3. P. Bourke. see website http://astronomy.swin.edu.au/ pbourke/stereographics/stereorender/
4. F.C. Crow. Shadow algorithms for computer graphics, parts 1 and 2. *Proc. SIGGRAPH*, pages 242–248 and 442–448, Vol. 11-2, 1977.
5. F. Däumlich and R. Steiger, editors. *Instrumentenkunde der Vermessungstechnik.* H. Wichmann Heidelberg 2002 pages 205–211
6. A. Lee. see website gamasutra.com/features/20000908/lee_01.htm *Gamasutra Game Developer Site.* Last visit: 30.01.2004
7. F. Huang, S. Wei, R. Klette, G. Gimmelfarb, R. Reulke, M. Scheele, and K. Scheibe. Cylindrical panoramic cameras - from basic design to applications. In D. Kenwright, editor, *Image and Vision Coumputing New Zealand*, pages 101–106, 2002.
8. F. Huang, S. Wei, and R. Klette. Calibration of line-based panoramic cameras. In D. Kenwright, editor, *Image and Vision Coumputing New Zealand*, pages 107–112, 2002.
9. F. Kern. Supplementing laserscanner geometric data with photogrammetric images for modelling. In J. Albertz, editor, *Int. Symposium CIPA*, pages 454–461, 2001.
10. R. Klette , G. Gimel'farb, S. Wei, F. Huang, K. Scheibe, M. Scheele, A. Brner and R. Reulke, On Design and Applications of Cylindrical Panoramas. In N. Petkov and M. Westenberg, editors, *Proc. CAIP*, pages ??–??. Springer, Berlin, 2003.
11. W. Niemeier. *Gebäudeinformationssystem.* Vol. 19, *DVW-Schriftenreihe*, chapter "Einsatz von Laserscannern für die Erfassung von Gebäudegeometrien", pages 155–168. Wittwer, Stuttgart, 1995.
12. R. Reulke, A. Wehr, R. Klette, M. Scheele, and K. Scheibe. Panoramic mapping using ccd-line camera and laser scanner with integrated position and orientation system. *Image and Vision Coumputing New Zealand*, pages 72–77, 2003.
13. A. Wiedemann. Kombination von Laserscanner-Systemen und photogrammetrischen Methoden im Nahbereich. *Photogrammetrie Fernerkundung Geoinformation*, pages 261–270, 2001.