Hardware Accelerated Texture Extraction for a Photo-Realistic Predictive Display

Tim Burkert, Jan Leupold, Georg Passig

Institute for Real-Time Computer Systems
Technische Universität München, D-80333 Munich, Germany
Email: tim.burkert@rcs.ei.tum.de

Abstract

Predictive displays have proven their suitability to compensate time delays in the visual feedback of teleoperation applications. The presented work describes a *photo-realistic* predictive display that relies on a 3D polygonal scene model and camera images. Photo-realism is obtained using computer graphics techniques, especially the mapping of textures acquired from camera images. This work is focused on the extraction and processing of these textures. They are acquired by assigning areas of camera images to polygons of the scene. Occlusions in the scene are determined and marked in the textures. Artifacts due to inaccurately localized cameras are removed by discarding uncertain color information. Unknown areas in the textures are finally filled by interpolation. The algorithms are performed as much as possible directly on the graphics card for acceleration.

1 Introduction

The effect of time delay in the control loop between the operator and a robot is a well known problem in telemanipulation scenarios [6]. Lane et al. [11] verified in a case study that time delays greater than one second significantly decrease the operator's performance. But even lower lags of about 300 msec already lead to a "move and wait" strategy [5].

Among other approaches, like for example semi-autonomous tele-operators [12], predictive displays are considered as a viable method to compensate time delays [1]. Several variations of predictive displays have been suggested, which superimpose computer graphics on camera images (augmented reality) [10, 13], or completely replace them by synthetic images (virtual reality) [4, 7].

In this work the latter approach is used for the proposed predictive display. A geometric scene model of the robot and its environment is acquired [2] and continuously updated. A predicted view of the scene is generated from this model and presented to the operator with no noticeable time delay.

Photo-realism is required for immersive telepresence applications. To achieve photo-realism, computer graphics techniques, especially the mapping of textures acquired from camera images, are used. This paper is focused on the extraction and processing of these textures. They are acquired by assigning areas of camera images to polygons of the scene. Occlusions in the scene are determined and marked in the textures. Artifacts due to inaccurately localized cameras and model inherent inaccuracies are removed by discarding uncertain color information. Unknown areas in the textures are finally filled by interpolation.

Another photo-realistic display that is focused on an image-based approach with uncalibrated and a-priori unknown setup is for example proposed in [8]. Carranza et al. extract textures for 3D-video of human actors [3]. Artifacts are removed off-line by checking the visibility of vertices from several slightly displaced views.

Section 2 introduces the embedding telemanipulation scenario. The process of texture extraction is explained in section 3. The paper closes with results in section 4 and a conclusion in section 5.

2 Embedding System

2.1 Overview

The presented work is part of a teleoperation scenario as illustrated in figure 1. A human operator controls a far-off robotic manipulator to fulfill a certain task. A straightforward implementation is to

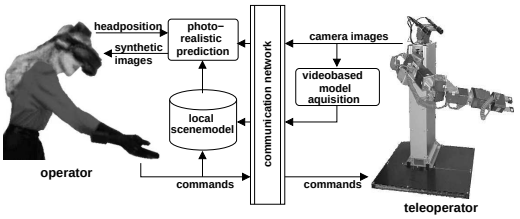

Figure 1: Teleoperation scenario with a photo-realistic predictive display

measure actions of the operator, such as motions of his hand and head by pose-tracking devices. These actions are mapped into commands that are transmitted over the communication network and are executed by the robot. To give visual feedback of the robot, its environment, and the execution of the issued commands to the operator, video images are captured by cameras on the robot. These images are sent back over the network. Both transmissions involve a delay composed of network latency and data transmission time. The effect for the operator is a noticeable time delay between his actions and a visible change in the remote scene. To compensate this delay a predictive display is used. Instead of camera images, a synthetic view of the remote scene is displayed to the operator. This synthetic view simulates the view of the remote scene as it would look like after the execution of the command.

2.2 Scene Model

The proposed predictive display relies on a polygonal 3D model of the remote scene, as well as information about the kinematics and dynamics of the robotic manipulator.

Two scene models are used that represent the environment of the robot at different points in time. The process of texture extraction relies on the model M_t that corresponds to the remote scene at the time the camera image was taken. The display for the operator is based on a second model M_d. The vertices in M_d represent the predicted scene based on the operator's head and hand position.

3 Texture Extraction

The measured position and orientation of the operator's head define his current viewpoint in M_d. The actual rendering of the predicted scene is performed using a standard rendering algorithm with texture mapping. Since the textures are extracted from camera images they implicitly contain scene lighting. The following subsections are focused on the acquisition of these textures.

Processing of textures involves highly parallelizable computations. In contrast to the CPU, the *Graphics Processing Unit* (GPU) is optimized for such operations. Not even by using architecture extentions like MMX, ISSE or 3Dnow! the CPU can reach similar throughput. Above all, handling textures on the CPU leads to heavy transfer of image data across the AGP which then easily becomes a bottleneck. Therefore all described algorithms are designed to run as far as possible directly on the graphics card for acceleration.

A drawback of execution on the GPU is that possible operations are limited to those offered by the API, here OpenGL. The following subsections show how the aspired tasks can be realized in spite of this restriction.

Texture extraction is performed periodically to cope with illumination changes. The necessary input data is:

- the camera image
- the corresponding scene model M_t
- the intrinsic camera parameters $C_t(c_x, c_y, s_x, s_y, f)$
- the camera position and orientation P_t

3.1 Extraction of Perspective Textures

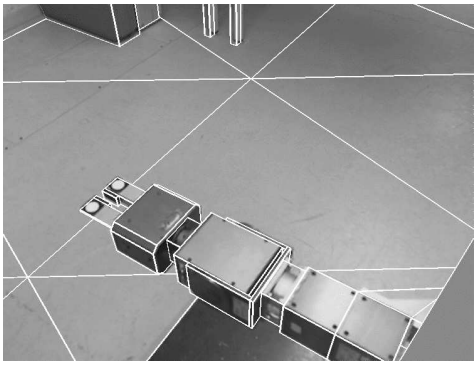

Figure 2: Camera image with overlaid scene model

In a first step the camera image is copied into the framebuffer. Now areas in the camera image are as-

signed to polygons. Using C_t and P_t the projection (x, y) of each vertex of M_t is calculated (fig. 2).

Based on these image coordinates a bounding box is constructed for every polygon. This rectangle contains all color information for this polygon available in the camera image. It is now stored as a texture (`glCopyTexImage2D`). Giving the offset (x_o, y_o) of the bounding box relative to the image origin, the normalized texture coordinates (u, v, w, q) of a vertex can be calculated by

$$u = \frac{x}{t_x s_x} + \frac{c_x - x_o}{t_x} \quad (1)$$

$$v = \frac{y}{t_y s_y} + \frac{c_y - y_o}{t_y} \quad (2)$$

$$w = 0 \quad (3)$$

$$q = \frac{z}{f} \quad (4)$$

with t_x, t_y being the texture size in pixels. q contains the perspective distortion of the camera projection for correct interpolation across the texture during rendering.

3.2 Occlusion Handling

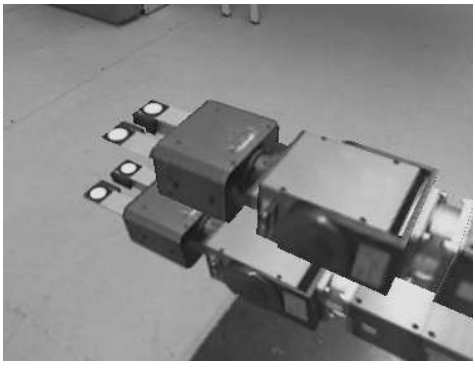

Figure 3: Rendered image from different viewpoint without prior occlusion detection

So far all parts of the camera image within the projection of the corresponding polygon were used as texture. But what happens if parts of this polygon are occluded by others? Simply using the method described before, areas in the image that belong to the occluder will appear in the occludee's texture. While rendering new synthetic views the occluder's surface appears copied onto the polygon farther away (fig. 3). To avoid such artifacts, parts of the textures that actually belong to other polygons are marked as invalid in the alpha channel. Valid texels are coded with an alpha value of 1 and invalid texels with 0.

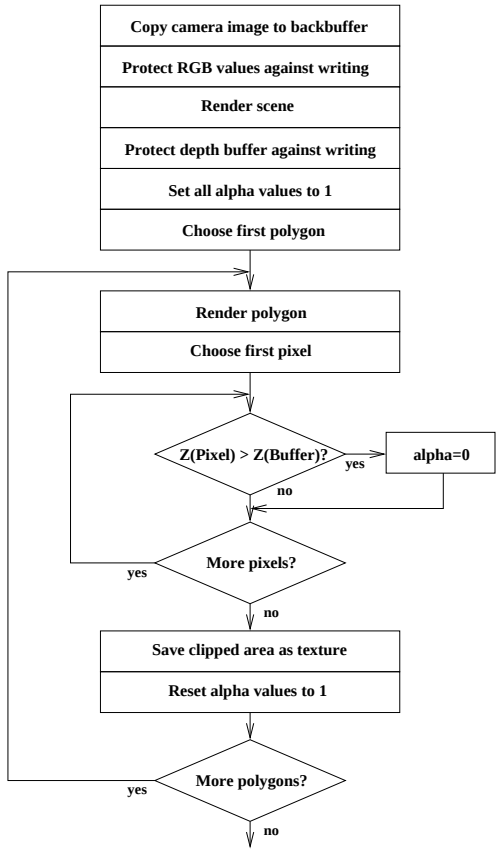

Figure 4: Occlusion detection

The depth buffer of a 3D hardware offers the functionality necessary to realize this on the graphics card. The algorithm is implemented as follows.

As mentioned before the camera image is loaded into the graphics memory, to be precise it is loaded into the color buffer. Now the color buffer is protected against writing and the whole scene is rendered. As a result the depth information for each pixel is calculated and stored in the depth buffer. For the following steps the depth buffer is also protected against writing. The alpha value for every pixel is initialized to 1, marking every pixel as valid.

In a next pass every polygon is rendered sepa-

rately once again. This time the determined depth for each pixel is compared to the stored depth value. A stored depth value smaller than the one of the polygon indicates that this part of the polygon is occluded by another one. Therefore the alpha value for such pixels is changed to 0. The content of the previously determined bounding box is stored as a texture including the alpha values. After resetting all alpha values to 1 the next polygon is processed.

For all rendering passes C_t is used for projection. Figure 4 summarizes the presented method.

3.3 Removing Artifacts

In the previous sections a perfect camera registration and a perfectly modeled remote environment were assumed. In reality the camera position will usually not be known precisely enough. In addition the model is always an approximation. Therefore some pixels in the camera image may be assigned to inappropriate polygons. This leads to artifacts while rendering the scene. For polygons that have a common edge the artifacts are hardly noticeable, because they are next to each other from every point of view. But when occlusions occur, some color information may happen to appear at a completely wrong position when viewed from a different point of view during rendering. Wrong color information in the *occluding* polygon is not very eye-catching, because it always appears at polygon borders. But within the *occluded* polygon wrong color information can appear anywhere. This is especially disturbing when a completely different color is displayed within a regularly textured area (fig. 5).

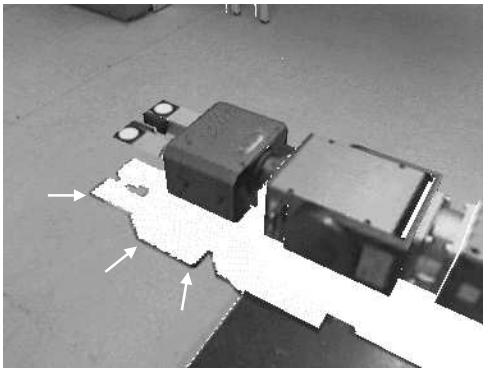

Figure 5: Artifacts in rendered image

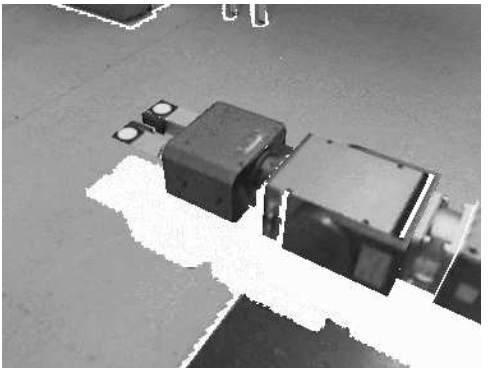

Figure 6: Artifacts removed by enlarging occlusions

Therefore a method for removing artifacts by discarding uncertain color information is used. Such artifacts always occur next to areas that were marked as occluded (see sec. 3.2). The solution is to enlarge the detected occlusions (fig. 6) which corresponds to a manipulation of the alpha mask. This mask is scaled down by the factor of two in both dimensions using bilinear interpolation offered by texture mapping. Four texels are merged to one, resulting in $\alpha \leq \frac{3}{4}$ if at least one original texel was occluded. The new mask is again enlarged to the original size. Since again bilinear interpolation is employed, all texels that were influenced by at least one occluded texel satisfy $\alpha \leq \frac{15}{16}$. These texels are then set to $\alpha = 0$. The *maximum* enlargement is $1 \rightarrow 16$ texels for a single occluded texel. To achieve wider enlargement the algorithm is repeated several times always using the determined alpha mask as new input. An example is shown in figure 7. The resulting alpha mask is fused with the color texture for further processing.

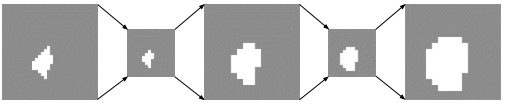

Figure 7: Progressive enlargement of occluded areas

Very small textures are excluded from occlusion enlargement. They may otherwise lose their whole color information.

3.4 Normalization

Simply saving the textures perspectively distorted together with the their texture coordinates results in two problems.

First, subsequent camera images are usually not recorded from the same point of view. What perspective should be used if two textures for a polygon are to be merged (see sec. 3.5)?

Secondly, the position of the camera relative to the polygon also specifies the size of the textures. This can lead to very large textures even for small polygons when the camera is close to them. This must be avoided because of limited texture memory.

As a solution the distorted textures are only stored temporarily. Each perspective texture is projected onto its polygon from a point of view in the normal direction towards the polygon with the perspective correction as described in 3.1. Figure 8 illustrates this mapping. Orthogonal projection does

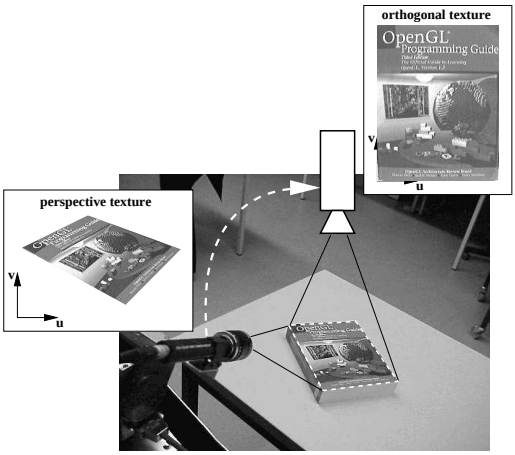

Figure 8: Texture normalization

not change the length ratios. This leads to the resulting view being directly proportional to the polygon's size. The resulting reprojected textures for one polygon will always have the same shape and size no matter from where the camera image was taken. These orthogonal views are now stored as textures. Since there is no more perspective distortion in the normalized textures, the texture coordinates can be reduced to (u, v) for further processing. A maximum size for textures is defined to avoid memory overflow. Every texture larger than this threshold is scaled to the maximum size.

3.5 Texture Fusion

Whenever a new camera image is transmitted from the robot to the operator, the new color information has to be merged with the textures extracted from previous pictures. First the stored texture is used to render the polygon from an orthogonal view. While rendering the texture alpha values are tested so that only valid pixels are drawn. Then a normalized

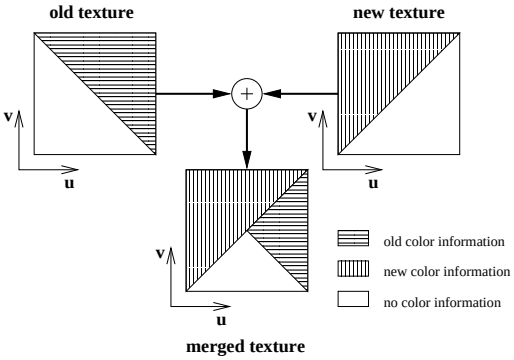

Figure 9: Texture fusion

temporary texture is extracted from the new camera image as previously described. This texture is now employed to render the polygon in a second pass while testing the alpha values once again. The resulting texture is stored again for rendering synthetic views. It contains all color information from the current camera image plus the holes filled as far as possible with old information (see fig. 9).

3.6 Hole Filling

Even after texture extraction and fusion from many camera images there may always be parts of polygons that have never been seen. Handling of artifacts reduces the available color information even more. Such disturbing "holes" in the textures must be hidden from the operator. They are therefore filled by interpolating the color information from their borders. This operation is not available in OpenGL. It is reproduced as follows.

First for every polygon a texture pyramid is built. In each step the side lengths are divided by two. The alpha-value of the subsequent step is calculated from the four involved texels by

$$\alpha_{i+1} = \begin{cases} 1, & \text{for } \sum_{j=0}^{3} \alpha_{i,j} > 0 \\ 0, & \text{for } \sum_{j=0}^{3} \alpha_{i,j} = 0 \end{cases} \quad (5)$$

The color ist only defined for $\alpha_{i+1} = 1$ and is determined by

$$\underline{C}_{i+1} = \frac{\sum_{j=0}^{3} \alpha_{i,j} \underline{C}_{i,j}}{\sum_{j=0}^{3} \alpha_{i,j}} \quad (6)$$

This yields an interpolation of only *valid* texel colors. The resulting texture is the input for the next interpolation until the result only contains one pixel (fig. 10, left). This texel is the interpolation of all valid texels[1].

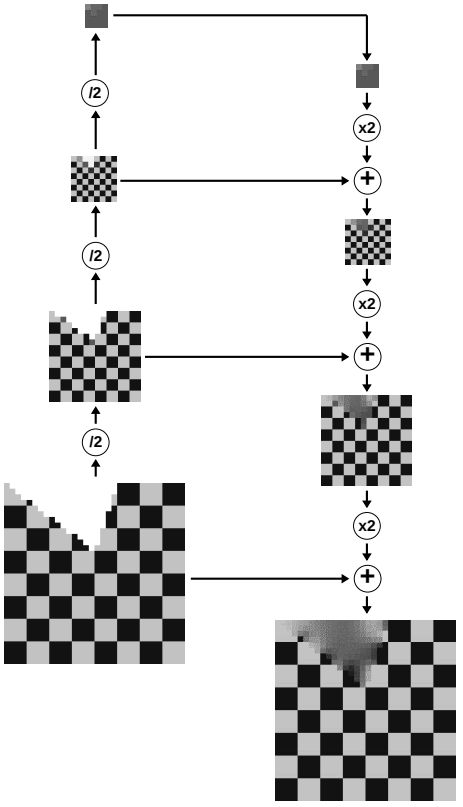

Figure 10: Texture pyramid, left: construction, right: collapse

[1] It is not really the *bilinear* interpolation of *all* valid texels.

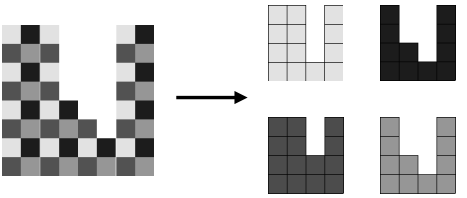

Figure 11: Subsampling of a texture

The equations above can not directly be mapped to OpenGL commands, because in the graphics pipeline the interpolation occurs *before* the alpha test. Therefore during interpolation it is not possible to distinguish between valid and invalid texels. Instead the texture is subsampled four times, so that each resulting subtexture contains one of the texels to interpolate for the corresponding texel of the next step (fig. 11). The weighted interpolation is then performed using the *NVIDIA register combiners* [9].

Once all subtextures are available the pyramid is collapsed (fig. 10, right). The smallest texture (1x1 texel) is enlarged to 2x2 texels. All valid texels of the next step (2x2) overwrite the content of this texture. The result is enlarged to 4x4 texels and so on until the original size is reached.

4 Experiments

4.1 Setup

For the measurement of the operator's movements, the information of an *Ascension Flock of Birds* magnetic sensor system is used. Sensors are mounted in a gripper and on the *Sony HMD800* head-mounted display that shows the currently predicted view. The position of the user's hand controls a 7-joint modular robotic system in an anthropomorph, right-hand configuration. A pan-tilt unit completes the mechanic system. Two *Sony DFW-V500* firewire cameras provide the required images. The camera system can also be mounted on a tripod to allow translational movements. Latencies of the transmission channel are simulated by adding a delay to the unidirectional control flow from the tracking system to the manipulator and the pan-tilt head.

The computer graphics calculations are performed on a standard PC with an *AMD Athlon* Processor with 1GHz and 512MB DDR-RAM. The

graphics card is based on an *NVIDIA Geforce 4 Ti4600 Processor* with 128MB memory. OpenGL 1.3 is used as graphics API on a Linux system.

4.2 Results

Photo-realistic scene prediction without noticeable time delay can already be shown in the system when the display is based on a manually acquired model (like the model in fig. 14). Most of the algorithms concerning memory and calculation intensive textures are executed directly on the graphics hardware. That way reasonable execution times are achieved even on consumer graphics cards. The predicted scene is rendered with about 100 frames per second. The time for extracting textures depends on the model. For the example scene with about 2,000 polygons the average processing time is 500ms. The maximum texture size is set to 256 x 256 texels with 32 bit per texel. In figure 5 artifacts due to modeling errors were shown. Using hole filling for the occluded areas leads to even more disturbing artifacts (fig. 12). If hole filling is performed after the occlusions were enlarged (fig. 6), the color interpolation gives the desired results (fig. 13).

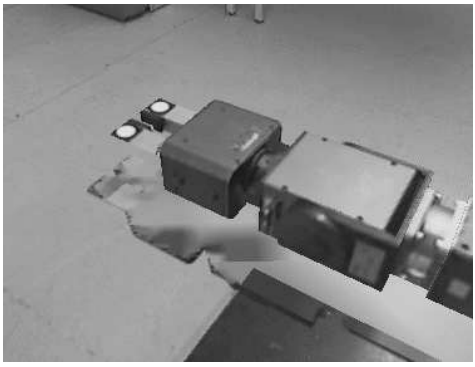

Figure 12: Hole filling without removing artifacts

An example of scene prediction after texture extraction from two images is shown in figure 14 (here without hole filling). For the overlapping areas of the two camera images texture fusion was applied.

Figure 15 shows a predicted view after 20 camera images were processed. As an example of a change in the scene the robot arm has been moved, as commanded by the operator. Tests of a real tele-

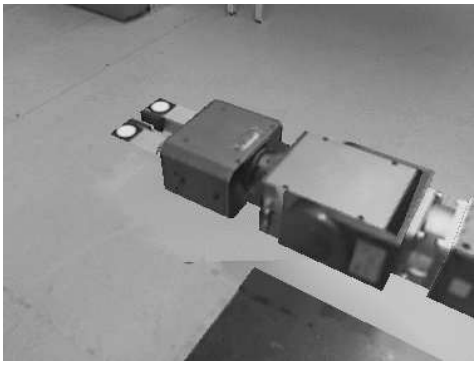

Figure 13: Hole filling after removing artifacts

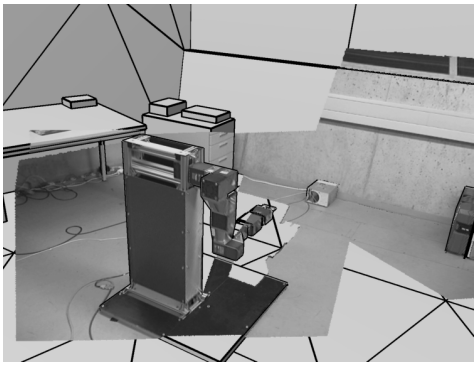

Figure 14: Scene prediction with textures from two camera images

manipulation scenario were successfully conducted as shown in figure 16 from the operator's point of view.

5 Conclusion and Future Work

This paper described the concept and an implementation of hardware accelerated texture extraction for a photo-realistic predictive display. Fast processing and rendering is possible even on low-cost 3D graphics cards using OpenGL.

A method for facilitating texture update in parallel to rendering is already developed and implemented. Closed-loop interaction with continuous texture update will be shown once the implementation of the two model states is finished.

A vendor-independent solution for the weighted interpolation using a fragment program is in progress.

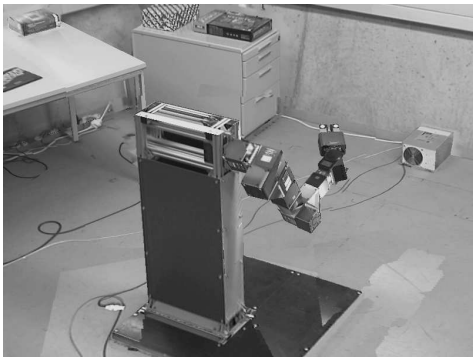

Figure 15: Densely textured scene prediction with changed model (robot arm moved)

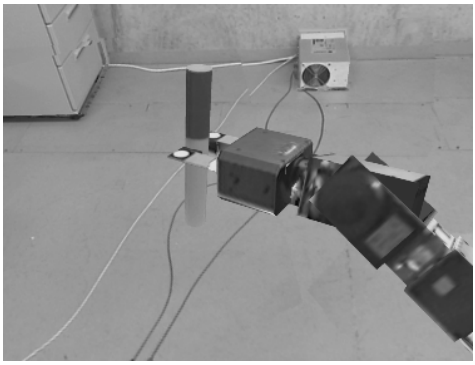

Figure 16: Grasping an object from the operator's point of view

6 Acknowledgement

This work was supported in part by the German Research Foundation (DFG) within the Collaborative Research Centre SFB 453 on "High-Fidelity Telepresence and Teleaction".

References

[1] Antal K. Bejczy, Steven Venema, and Won S. Kim. Role of computer graphics in space telerobotics: Preview and predictive displays. In *Cooperative Intelligent Robotics in Space*, volume 1387 of *Proceedings of the SPIE*, pages 365–377, November 1990.

[2] T. Burkert, J. Leupold, and G. Passig. Model Acquisition from Stereo Vision for Photo-Realistic Scene Prediction. Third IEEE International Conference on Humanoid Robots, October 2003.

[3] J. Carranza, C. Theobalt, M. A. Magnor, and H. Seidel. Free-viewpoint video of human actors. In *SIGGRAPH 03 Conf. Proc.*, 2003.

[4] Kostas Daniilidis and Christopher Geyer. Omnidirectional Vision: Theory and Algorithms. In *Proc. 15th Int. Conf. on Pattern Recognition*, volume 1, pages 89–96, 2000.

[5] Stephen R. Ellis, Mark J. Young, and Bernard D. Adelstein. Discrimination of changes in latency during head movement. In *8th International Conference on Human-Computer Interaction (HCI'99)*, 1999.

[6] William R. Ferrell. Remote manipulation with transmission delay. In *IEEE Transactions in Human Factors in Electronics, 6(1)*, 1965.

[7] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl. ROTEX – The First Remotely Controlled Robot in Space. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'94)*.

[8] Martin Jägersand. Image-based predictive display for high d.o.f. uncalibrated telemanipulation using affine and intensity subspace models. *Advanced Robotics*, 14(8):683–701, February 2001.

[9] Mark J. Kilgard. *NVIDIA OpenGL Extension Specifications*. NVIDIA Corporation, 2003.

[10] Won S. Kim. Virtual Reality Calibration and Preview/Predictive Displays for Telerobotics. *Presence*, 5(2):173–189, 1996.

[11] J. C. Lane, C. R. Carignan, B. R. Sullivan, D. L. Akin, T. Hunt, and R. Cohen. Effects of time delay on telerobotic control of neutral buoyancy vehicles. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, Washington, 2002.

[12] Matthew R. Stein. *Behaviour-Based Control for Time-Delayed Teleoperation*. Dissertation, University of Pennsylvania, 1994.

[13] G. Thomas, T. Blackmon, M. Sims, and D. Rasmussen. Video engraving for virtual environments. *Electronic Imaging*, 1997.