

Real-time Cartoon-like Stylization of AR Video Streams on the GPU

Jan Fischer, Dirk Bartz

WSI-2005-18
September 2005

Graphisch-Interaktive Systeme
Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
e-mail: {fischer, bartz}@gris.uni-tuebingen.de
WWW: <http://www.gris.uni-tuebingen.de>

Real-time Cartoon-like Stylization of AR Video Streams on the GPU

Jan Fischer Dirk Bartz

WSI/GRIS - VCM
University of Tübingen, Germany

e-mail: fischer@gris.uni-tuebingen.de

Abstract

The ultimate goal of many applications of augmented reality is to immerse the user into the augmented scene, which is enriched with virtual models. In order to achieve this immersion, it is necessary to create the visual impression that the graphical objects are a natural part of the user's environment. Producing this effect with conventional computer graphics algorithms is a complex task. Various rendering artifacts in the three-dimensional graphics create a noticeable visual discrepancy between the real background image and virtual objects.

We have recently proposed a novel approach to generating an augmented video stream. With this new method, the output images are a non-photorealistic reproduction of the augmented environment. Special stylization methods are applied to both the background camera image and the virtual objects. This way the visual realism of both the graphical foreground and the real background image is reduced, so that they are less distinguishable from each other.

Here, we present a new method for the cartoon-like stylization of augmented reality images, which uses a novel post-processing filter for cartoon-like color segmentation and high-contrast silhouettes. In order to make a fast post-processing of rendered images possible, the programmability of modern graphics hardware is exploited. We describe an implementation of the algorithm using the OpenGL Shading Language. The system is capable of generating a stylized augmented video stream of high visual quality at real-time frame rates. As an example application, we demonstrate the visualization of dinosaur bone datasets in stylized augmented reality.

1. Introduction

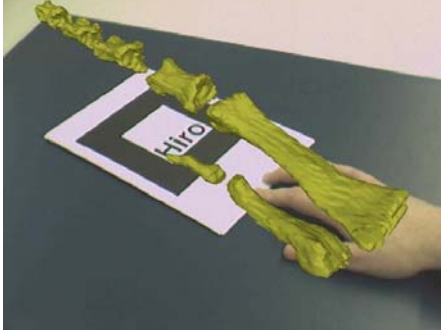
Augmented reality (AR) has become a widespread method for enriching the user's environment with virtual

objects [3]. In video see-through augmented reality, a digital video camera continually acquires images of the real surroundings. Graphical objects are then drawn over the camera image, which is displayed as a background image plane. In order to achieve a correct spatial positioning and orientation when rendering the virtual objects, tracking techniques like vision-based marker tracking are normally employed [15].

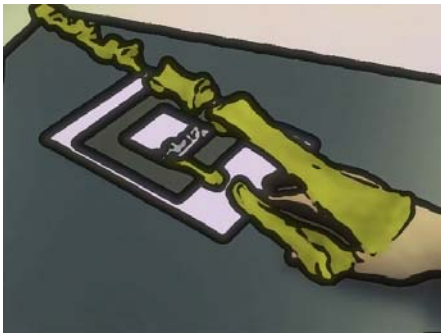
In conventional augmented reality systems, the graphical objects are rendered over the camera image using standard real-time graphics algorithms. Low level software libraries like OpenGL [20] or high level scene graphs based on them are often used for this task. The real-time rasterization methods which constitute the core of such renderers rely on simplified assumptions for illumination and shading. During system setup, manually placed virtual light sources are provided as input for the lighting calculations. Simple interpolation methods like Gouraud shading then spread the computed brightness values over the graphical models. The resulting renderings tend to look artificial, and they stand out from the background image. This visual discrepancy between real and virtual scene elements in augmented reality is illustrated in Figure 1(a). Here, the virtual dinosaur bone model can be easily distinguished from the real background image.

We have recently presented a novel paradigm for generating augmented video streams. Our *stylized augmented reality* technique attempts to create similar levels of realism in both the camera image and the graphical objects. The cartoon-like stylization mode, which we have described, produces augmented images composed of mostly uniformly colored regions, which are enclosed with black silhouette lines [6]. Since the same type of stylization is applied to both the background image and the virtual models, they become much more difficult to distinguish. This can lead to an improved immersion into the augmented environment.

Our previously published stylization method uses a specific pre-processing filter for the camera image and an adapted non-photorealistic rendering (NPR) procedure for



(a) Conventional augmented reality



(b) The new cartoon-like stylization method for AR

Figure 1. Model of a dinosaur bone displayed in augmented reality. The stylized augmented reality video stream is generated in real-time.

the virtual objects. It relies primarily on image processing and object space computations performed on the CPU. In this paper, we describe a new algorithm for the cartoon-like stylization of AR images. This new method is a specialized post-processing filter, which is applied to the augmented image after the overlay of virtual objects. We present an implementation of the new algorithm using vertex and fragment shaders for the programmable GPUs of recent graphics cards. This allows for a straightforward and efficient design of the algorithm. The new method produces a stylized augmented video stream of a better visual quality at a significantly higher frame rate compared to the previously described approach. In particular, it all but eliminates flickering silhouettes in the processed video, which were often generated by the previous algorithm.

Some types of application exist for which the stylized augmented reality technique may not be well suited. In fields like medicine or security-critical scenarios, the fidelity of the displayed camera image is very important. Our approach, which reduces the visual realism of the generated video stream, can therefore not be utilized in such a context. However, we believe that applications in many other fields can benefit from the blurred barrier between real and vir-

tual. These include entertainment, education, training, and research on human perception and presence in augmented environments.

In the remainder of this paper, we give a brief summary of previous work in Section 2. Section 3 contains the description of our new non-photorealistic image filtering algorithm. Subsequently, Section 4 elaborates on some details of our implementation. Experimental results obtained with our algorithm are discussed in Section 5, and Section 6 describes the visualization of dinosaur bones as an example application for our system. Finally, we give some concluding remarks in Section 7.

2. Related Work

An approach that is complementary to our method of applying stylization to AR images is the attempt to improve the realism of virtual objects. This way, a better visual correlation to the camera image can also be achieved. Research has been done into methods of analyzing the real illumination conditions in an augmented reality setup. Examples of this approach include the work of Kanbara and Yokoya on analyzing the distribution of real light sources, which is then used for adapting the representation of graphical objects [14]. Their method requires a special marker and mirror ball to be visible in the camera image for computing the environment light map. A similar technique for utilizing an acquired environment illumination map is proposed by Agusanto et al. [2]. In their system, a mirror ball and special camera are used in a specific procedure for determining the lighting conditions in the scene beforehand.

Another method for making virtual objects appear more realistic is to add shadowing to the AR image. This creates the impression that shadows are cast from virtual objects on physical surfaces. Haller et al. describe an algorithm for computing such shadows in augmented reality [11, 10]. A similar technique is used for a user study on the effects of shadowing in AR by Sugano et al. [22]. As a drawback of these methods, a model of the geometry of the surfaces and objects in the real world is required. This model needs to be generated beforehand and is assumed to remain static.

Our new algorithm for generating a stylized augmented video stream is based on a non-photorealistic image filter. Non-photorealistic and artistic rendering and image processing have been areas of very active research for several years. Strothotte and Schlechtweg have published a good survey of methods used in the field [21]. Another overview over various NPR techniques is given by Gooch and Gooch [8]. One example of an algorithm for the cartoon-like stylization of photographs is the work presented by DeCarlo and Santella [4]. Their technique uses a combination of color segmentation and edge detection, which partly inspired our approach. However, this method requires several

minutes for processing an input image. An algorithm for semi-automatic conversion of a real video sequence into a cartoon-like video has been presented by Wang et al. [25]. This method produces results of good visual quality, but it is an offline algorithm and computationally too expensive for real-time applications. Moreover, a certain amount of user interaction for the specification of semantic regions in some video frames is necessary. Hertzmann and Perlin have presented a method for the non-photorealistic processing of video streams using an artistic painterly style [13].

A system that integrates non-photorealistic rendering into augmented reality has been presented by Haller and Sperl [10, 12]. However, their system applies artistic rendering techniques only to the virtual objects, whereas the background camera image is displayed in its original, unprocessed form.

Our stylized augmented reality concept can be considered to be related to the fields of mediated and diminished reality (e.g. see [9, 17]). These approaches also aim at applying various filters to the user’s vision of the real world instead of merely superimposing graphical models.

3. Description of the Algorithm

We present a new algorithm for the stylization of an augmented video stream. For each frame, a standard augmented reality pipeline first generates an output image containing the camera image with overlaid virtual objects. This original AR frame is rendered using the graphics hardware and resides in its local frame buffer memory. A post-processing filter is then applied to it, which is executed by the graphics processing unit (GPU). An overview of the approach is shown in Figure 2.

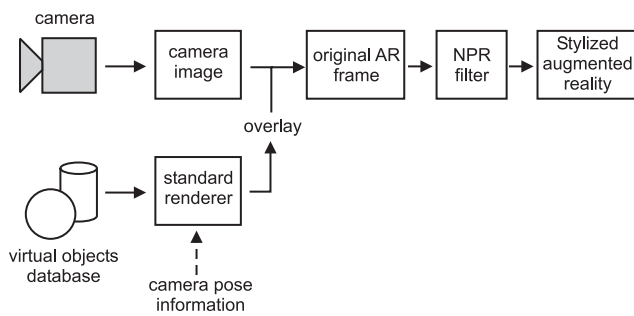


Figure 2. Overview of the new stylized augmented reality pipeline.

The stylization filter consists of two steps. In the first step, a simplified color image is computed from the original AR frame. The simplified color image is made up of mostly uniformly colored regions. A non-linear filter using a photometric weighting of pixels is the basis for this

computation. The photometric filter is applied to a shrunk version of the input image. This way, a better color simplification is achieved, and the required computation time can be reduced. Several filtering iterations are consecutively applied to the image. The repetition of the filter operation is necessary in order to achieve a sufficiently good color simplification.

The second stage of the non-photorealistic filter is an edge detection step. The simplified color image is the primary input for this operation. This way, the generated silhouette lines are located between similarly colored regions in the image, which is an approximation of a cartoon-like rendering style. To a lesser degree, edges detected in the original AR frame are also taken into account when drawing the silhouette lines. The higher resolution of the original image compared to the shrunk color image can contribute some additional detail to the edge detection result. Figure 3 shows an overview of this filter stage.

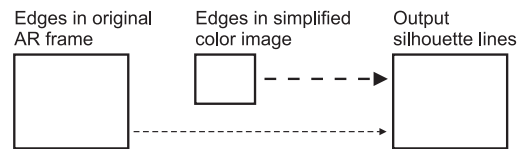


Figure 3. Edge detection results from the original image and the shrunk color image are combined.

In typical setups, most of the input for the edge detection step is taken from the simplified color image. It consists of mostly uniformly colored regions generated by the photometric filter. Therefore, edges detected in the simplified color image typically correspond quite well to the outer boundaries of physical or virtual objects.

Finally, the simplified color image is combined with the edge detection results. The color image is enlarged to the size of the original input image. The combined responses of the edge detection filters are drawn over the enlarged image as black lines. A specific weight function is used for computing a transparency for the detected edge pixels, which produces a smooth blending over the color image.

3.1. Generation of Simplified Color Image

A shrunk version of the original AR frame is rendered into the local frame buffer of the graphics card. This is done by drawing a rectangle textured with the original image. The texturing process is configured so that a smoothly scaled version of the image is produced. User-definable parameters *shrunkImageWidth* and *shrunkImageHeight* specify the dimensions of the new image. The non-linear filter is then applied iteratively

by using the output image of the last iteration as input texture for the next filtering step.



(a) Original AR frame



(b) Simplified Color Image

Figure 4. Generation of the simplified color image for an original AR frame. In the augmented reality scene, a virtual plane model is overlaid over the camera image. (Parameters: $shrunkImageWidth=240$, $shrunkImageHeight=180$, $\sigma_p=0.025$, $numFilterSteps=7$)

Our non-linear filter is inspired by bilateral filtering, which is a widespread method for creating uniformly colored regions in an image [23]. The bilateral filter algorithm combines geometric and photometric weights when adding up pixels in the neighborhood of the currently regarded pixel. While the geometric factor gives a greater weight to pixels closer to the current location, the photometric weight suppresses the influence of pixels with very dissimilar color values. We have found the photometric weight to be sufficient for our application. Ignoring the geometric weight simplifies the algorithm and reduces the computational complexity. Moreover, this simplified non-linear fil-

ter produces very good results.

In addition to disregarding the geometric weight, we have also modified the filter so that the photometric weight only depends on the actual color of each pixel. Each pixel is converted into the YUV color space before the filter is applied. In the YUV color space, the Y component represents the brightness of a pixel, while U and V are the chrominance (color) components [18]. For computing the weight of each pixel in the neighborhood, our filter only takes the U and V coordinates into account.

We denote the original RGB image function as \mathbf{f} and the corresponding color coordinates in YUV space as \mathbf{f}_{UV} . The non-linear filter computes the simplified RGB image \mathbf{h} using the following equation:

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \sum_{\xi \in \Omega_{\mathbf{x}}} \mathbf{f}(\xi) s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) \quad (1)$$

In Equation 1, \mathbf{x} is the currently regarded point in the output image. A weighted sum is computed over image points ξ in the neighborhood $\Omega_{\mathbf{x}}$ of \mathbf{x} in the input image. We use a quadratic image area as neighborhood for the summation. The weight $s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x}))$ depends on the similarity of values in the color channels $\mathbf{f}_{UV}(\xi) - \mathbf{f}_{UV}(\mathbf{x})$. In our algorithm, s is a Gaussian function:

$$s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) = e^{-\frac{1}{2} \left(\frac{|\mathbf{f}_{UV}(\xi) - \mathbf{f}_{UV}(\mathbf{x})|}{\sigma_p} \right)^2} \quad (2)$$

Note that s is a function of the absolute value of the color difference $\mathbf{f}_{UV}(\xi) - \mathbf{f}_{UV}(\mathbf{x})$ (Equation 2). The standard deviation σ_p of the Gaussian function determines the properties of the color simplification and can be chosen by the user as a parameter for our algorithm. In order to maintain the overall brightness of the image, the weighted sum is divided by the normalization factor $k(\mathbf{x})$, which is computed as shown in Equation 3.

$$k(\mathbf{x}) = \sum_{\xi \in \Omega_{\mathbf{x}}} s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x})) \quad (3)$$

The effect of this non-linear filter is that an averaging of pixels only occurs in places where nearby pixels have similar colors. In such places in the image $s(\mathbf{f}_{UV}(\xi), \mathbf{f}_{UV}(\mathbf{x}))$ is large. If near the currently regarded pixel colors are present which are far away in color space, they are not taken into account. Thus strong edges in the image are preserved.

We use a small local neighborhood of 5×5 pixels for the weighted summation. As described above, the non-linear filter is applied several times. For each filtering step, the resulting image from the previous iteration is used as input. The number of color simplification iterations performed by the algorithm, $numFilterSteps$, can be chosen by the user. Figure 4 shows an example of a simplified color image computed for an augmented reality frame.

3.2. Adaptive Edge Detection based on Intensity and Color Contrasts

After the simplified color image has been generated, the edge detection step is performed. We use the Sobel edge detection filter for computing the partial derivatives of color channel values along the x-axis and the y-axis [7]. Here again, we convert the pixels into the YUV color space before the edge detection step. We denote the image function of the simplified color image as \mathbf{S} , consisting of the channels (S_Y, S_U, S_V) . Correspondingly, the original AR frame \mathbf{A} contains the YUV channels (A_Y, A_U, A_V) .

For each of the color channels of both images, two partial derivatives are calculated. In the case of the Y component, these are the derivatives $\frac{\partial S_Y}{\partial x}$, $\frac{\partial S_Y}{\partial y}$, $\frac{\partial A_Y}{\partial x}$, and $\frac{\partial A_Y}{\partial y}$. The U and V color channels are processed accordingly. Based on the partial derivatives, gradient magnitudes $(|\nabla S_Y|, |\nabla S_U|, |\nabla S_V|)$ are computed for the simplified color image, and $(|\nabla A_Y|, |\nabla A_U|, |\nabla A_V|)$ for the original AR frame.

An edge detection response is then calculated for each pixel using the gradient magnitudes. This response value is obtained through the weighted averaging of the local contrast in the intensity (Y) and color (U,V) channels. The relative weight of the intensity and color contrasts is determined by the parameter $\alpha \in [0; 1]$. Equation 4 shows the computation of the edge detection response for the simplified color image, $edge_{(S)}$, and the original AR frame, $edge_{(A)}$. Using this method, the edge detection process can generate responses in locations with homogeneous intensities, where the color channel gradient is large. The user can emphasize intensity contrasts or color contrasts for locating silhouette edges by adjusting the value of α . Edge detection responses computed for the example AR scene in Figure 4 are shown in Figure 5.

$$edge_{(S)} = (1 - \alpha) \cdot |\nabla S_Y| + \alpha \cdot \frac{|\nabla S_U| + |\nabla S_V|}{2} \quad (4)$$

$$edge_{(A)} = (1 - \alpha) \cdot |\nabla A_Y| + \alpha \cdot \frac{|\nabla A_U| + |\nabla A_V|}{2}$$

The two edge detection responses are then combined for determining the final silhouette intensity in the full-resolution output image. For every pixel position (x_o, y_o) in the output image, we denote the corresponding coordinates in the simplified color image as (x_s, y_s) . The parameter β introduced in Equation 5 specifies the relative influence of the simplified color image and the original AR frame for silhouette detection. Note that the output coordinates (x_o, y_o) are also used for accessing the edge detection responses of the original AR frame because it has an identical image resolution.



(a) $edge_{(S)}$



(b) $edge_{(A)}$

Figure 5. Edge detection responses for the original AR frame and simplified color image shown in Figure 4. (Parameter $\alpha=0.3$. Images have been brightened for better clarity.)

$$I_o(x_o, y_o) = (1 - \beta) \cdot smoothstep_{s_0}^{s_1}(edge_{(S)}(x_s, y_s)) \quad (5)$$

$$+ \beta \cdot smoothstep_{a_0}^{a_1}(edge_{(A)}(x_o, y_o))$$

As shown in Equation 5, each of the two edge detection responses is filtered with the *smoothstep* function. This function is provided by the shading language used for the implementation (see below). It returns a value of zero for edge detection responses below the threshold s_0 (a_0), and a value of one for responses above s_1 (a_1). Between the two thresholds, smooth Hermite interpolation is used (see [16] for a complete definition). The parameters s_0 , s_1 , a_0 and a_1 are specified by the user. They determine the minimum edge detection response necessary for generating a silhouette, and how steeply the silhouette intensity increases.

The combined edge detection response I_o is computed for every pixel location (x_o, y_o) in the output image. The final output image is then generated as follows: For every

output pixel, a corresponding simplified color image pixel is looked up with an interpolated texture access to $S(x_s, y_s)$. This pixel is then rendered at (x_o, y_o) , possibly with a silhouette edge blended over it. The silhouette edge intensity is computed as the factor $(1 - I_o)$, which is used for scaling the values in the RGB color channels of the output pixel. This way, the output pixel is dark, if a large combined edge detection response has been computed. The resulting output image is a magnified version of the simplified color image with black silhouette lines rendered over it. This is illustrated in Figure 6, which shows the final output image generated for the original AR frame in Figure 4(a).



Figure 6. The final output of the non-photorealistic filter (Parameters: $s_0=0.054$, $s_1=0.064$, $a_0=0.3$, $a_1=0.7$, $\beta=0.3$)

4. Implementation Details

We have implemented the non-photorealistic image filtering algorithm using the OpenGL Shading Language [19]. The shading language makes it possible to execute the code of the non-photorealistic filter on the graphics processing unit (GPU). All necessary computations are performed on data which are stored in the local memory of the graphics card. This eliminates the need for a time-consuming read-back of graphics memory contents.

The implementation of the algorithm uses three different textures stored in graphics card memory:

- During program startup, a one-dimensional exponential texture is defined. This exponential texture contains a sequence of function values computed as shown in Equation 2. The non-linear filter in the color simplification step looks up the photometric weights for neighboring pixels in this one-dimensional texture.

This way, an explicit evaluation of the Gaussian function for every pixel is unnecessary. The exponential texture is updated whenever σ_p is changed by the user.

- For each frame, the original AR image is copied from the frame buffer into a separate texture. This original AR texture is then used for rendering a scaled-down version of the image. Moreover, this texture is later accessed as image function **A** by the edge detection filter for computing the edge detection response $edge_{(A)}$. Standard OpenGL functionality is used for copying the frame buffer content into the texture memory (`glCopyTexSubImage2D()`¹, see [20]).
- The scaled-down version of the original AR image is also stored in a separate texture. This texture is repeatedly overwritten with the results of the iterations of the non-linear filter. Finally, it contains the simplified color image. Since this texture serves as buffer for intermediate images generated by the filtering passes, we also refer to it as the *multipass texture*. Again, the scaled-down original image and the filtering passes are copied into the texture using `glCopyTexSubImage2D()`.

Each of the textures is bound to a separate texture unit of the GPU. This way, they can be accessed simultaneously from the shaders. The filtering passes are performed by rendering 2D rectangles into the OpenGL back buffer. Before each rectangle is rendered, the respective image filtering shaders are activated. The final result image is again rendered into the back buffer, overwriting data from intermediate passes before the buffer is displayed to the user.

Both the color simplification and the edge detection stages are implemented as a pair of vertex and fragment shaders. The actual image filtering is performed in the fragment shaders, which require a large number of texture lookups. In order to achieve real-time performance for the non-photorealistic filter, a scheme for pre-computing texture coordinates in the vertex shaders is used. This way, no nested loops and vector multiplications are necessary for generating texel addresses in the neighborhood of the currently processed texture coordinate. A detailed description of this technique has been given by Viola et al. [24]. As mentioned in Sections 3.1 and 3.2, the non-photorealistic filter converts pixels into the YUV color space. This conversion is performed by multiplying RGB vectors with a

¹As an alternative solution, a technique for directly rendering into texture memory could be used. However, render-to-texture is currently not supported in the software environment that we use for development. We have run comparative benchmarks which showed that the `glCopyTexSubImage2D()` calls in our implementation do not have any measurable impact on the frame rate. This is due to the limited memory requirement of the camera image and the small number of actual copying operations.

constant matrix, which is an operation that can be computed very efficiently on the GPU.

5. Results

We have tested our new algorithm for generating stylized augmented video streams with various AR scenes. The software containing the implementation of the algorithm provides functionality for importing 3-d models in standard file formats. Vision-based marker tracking using the ARToolKit library delivers the pose information necessary for correctly rendering graphical objects (see [15]). The user can translate, rotate and scale the virtual models in relation to the marker coordinate system. Moreover, material parameters and textures can be assigned to the models. A comprehensive user interface for choosing the parameters of the stylization algorithm is also provided.

Figure 7 shows images of three test scenes. In each column of images, the first row contains the original AR frame, and the result of the stylization algorithm is shown in the second row. A virtual Santa Claus model is the virtual object in Fig. 7(a) and 7(d). In Fig. 7(b) and 7(e), a virtual Moka Express coffeemaker is located over the ARToolKit marker. The graphical model of a DC10 plane is displayed in the AR scene shown in Fig. 7(c) and 7(f).

For rendering these example images, the adaptive edge detection was performed with a strong emphasis on the simplified color image. The factor for edges in the original AR frame (parameter β in Equation 5) was set to values smaller than 0.5. This way, thick silhouette lines between large, homogeneously colored image regions were generated. The selected size of the multipass texture, which is also the size of the final simplified color image, typically was less than half of the dimensions of the original AR frame. Therefore, some detail was removed from the image, and uniformly colored regions were created. As illustrated in Figure 7, real and virtual scene elements look very similar in the stylized output video frames.

In Table 1, frame rates of our stylized augmented reality system measured for different algorithm parameters are listed. These measurements were made on a computer system with an Intel Pentium 4 Xeon processor running at 2.66 GHz and a graphics card based on an NVidia GeForce FX 6800 chipset. The webcam used in the AR system delivers video images with a resolution of 640 by 480 pixels. The first column in the table lists the selected resolution of the multipass texture (*shrunkImageWidth* x *shrunkImageHeight*). In the second column, the number of non-linear filter iterations (*numFilterSteps*) is listed. The third column contains the measured runtimes of the non-photorealistic filter in milliseconds. Finally, column four shows the overall system frame rate including the entire augmented reality pipeline.

Table 1. Runtimes of the non-photorealistic filter and overall frame rates for different algorithm parameters.

| Resolution (multipass tex.) | Filtering iterations | Post-processing (msecs) | Overall fps |
|-----------------------------|----------------------|-------------------------|-------------|
| 240x180 | 5 | 19 | 27.84 |
| 240x180 | 7 | 25 | 24.22 |
| 240x180 | 9 | 30 | 21.51 |
| 400x300 | 5 | 44 | 16.68 |
| 400x300 | 7 | 59 | 13.37 |
| 400x300 | 9 | 74 | 11.12 |

We have found a multipass texture resolution of 240x180 texels to be sufficient for generating an output video stream of good visual quality in most cases. Moreover, no more than 7 filter iterations are normally necessary. Assuming that the graphical models contained in the AR scene do not consist of an excessive number of polygons, our system can generate stylized augmented video streams at 25 fps or more in a typical setup.

6. Example Application

We demonstrate the visualization of dinosaur bones as an example application for our stylized augmented reality method. Figure 8 shows three different bone segments rendered in an AR scene using our stylization algorithm. The datasets were generated from the actual bones of a *Plateosaurus*. They were acquired with a computed tomography (CT) scanner. Subsequently, the volume datasets created by the scanning procedure were converted into a polygonal representation, which is imported into our augmented reality system.

In the output video stream generated by our stylization method, real and virtual objects look very similar. This improves the immersion in the augmented environment. When viewing the dinosaur bones, a convincing experience is created for users of the stylized augmented reality system. Such an application could be used for instance in a museum setting and for educational purposes.

7. Conclusions

We have presented a new method for generating stylized augmented reality images. Since the non-photorealistic filter is designed as a post-processing step, it can easily be combined with any augmented reality rendering system. Its GPU-based implementation is fast and delivers real-time

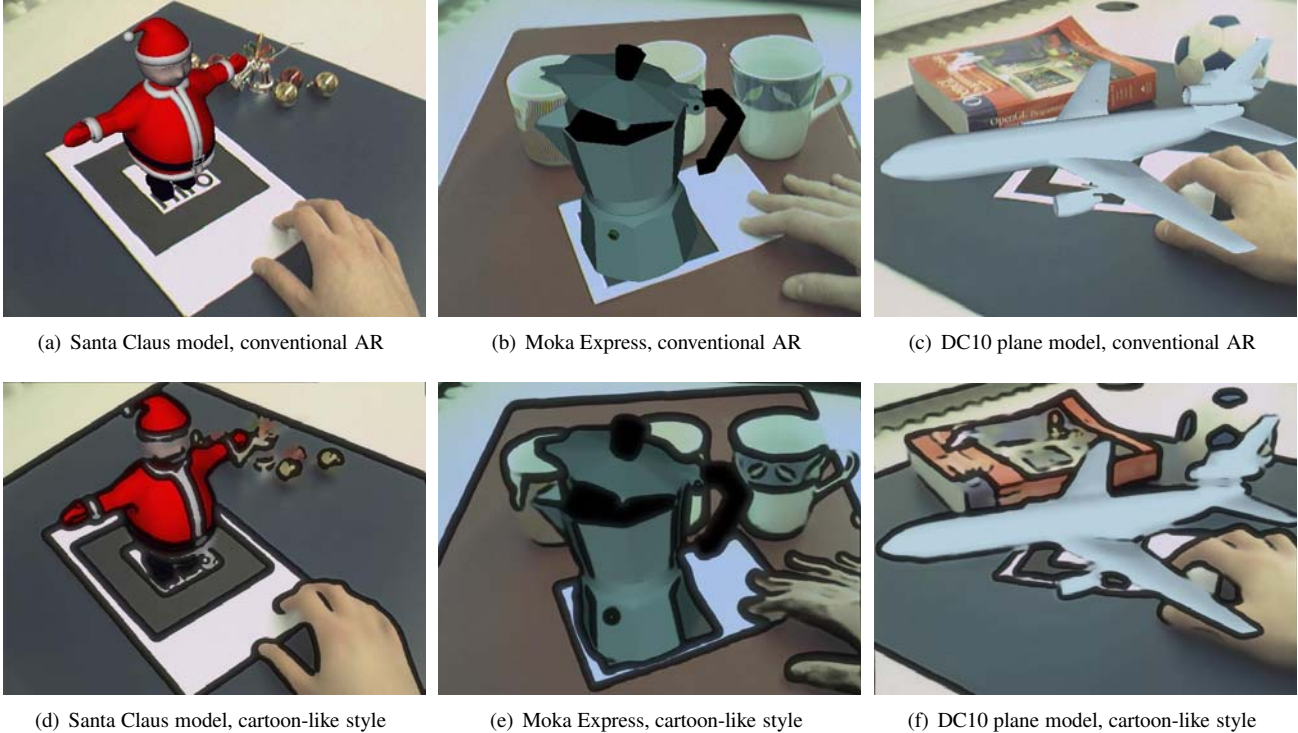


Figure 7. Three example scenes illustrating the effect of our new cartoon-like stylization for augmented reality. In each column, the top image shows conventional AR rendering. The bottom row contains the stylized versions of the respective video frames.

frame rates.

The algorithm consists of a color simplification step and an adaptive edge detection method, which are computed on a per-frame basis. They do not take temporal coherence in consecutive images into account. Depending on the illumination conditions in the surroundings and the quality of the video frames delivered by the camera, the parameters of the algorithm need to be adapted in order to obtain a good separation of uniform regions in the image. However, good results are normally generated with a constant parameter set as long as a similar type of augmented scene is viewed. Only if the video acquisition settings of the camera change significantly or if the lighting in the observed environment varies strongly, the algorithm parameters have to be corrected. In our experience, a near-constant setup of the camera image filter delivers acceptable results under most circumstances. However, very large or extremely small color contrasts between regions in the observed scene can cause the algorithm to produce unsatisfactory stylized images.

In order to achieve the objectives of stylized augmented reality, it is essential that the real background image and the virtual models look similar. We have found that the cartoon-like stylization produces a similar level of realism for both layers in the AR image. Since real and virtual scene el-

ements become much more difficult to distinguish, the immersion in the augmented scene is improved. While it could be argued that our method is based on effectively degrading the visual quality of the camera image, we think that enough information is preserved to be useful for many applications. For these scenarios, which do not require a high fidelity rendering of the camera image, our method can be considered a realization of the principle of “functional realism” as described by Ferwerda [5].

The new paradigm of *stylized augmented reality* can be useful for applications in entertainment, education and art. Due to the adapted degrees of realism in the real and virtual environments, a more convincing experience can be created for the user. Augmented reality games, art projects and interactive courses could benefit from this advantage. As a future development, various types of non-photorealism can be used in stylized augmented reality, resulting in different experiences for the user.

Acknowledgments

We would like to thank Ángel del Río for his support during the experiments and for proofreading this paper.

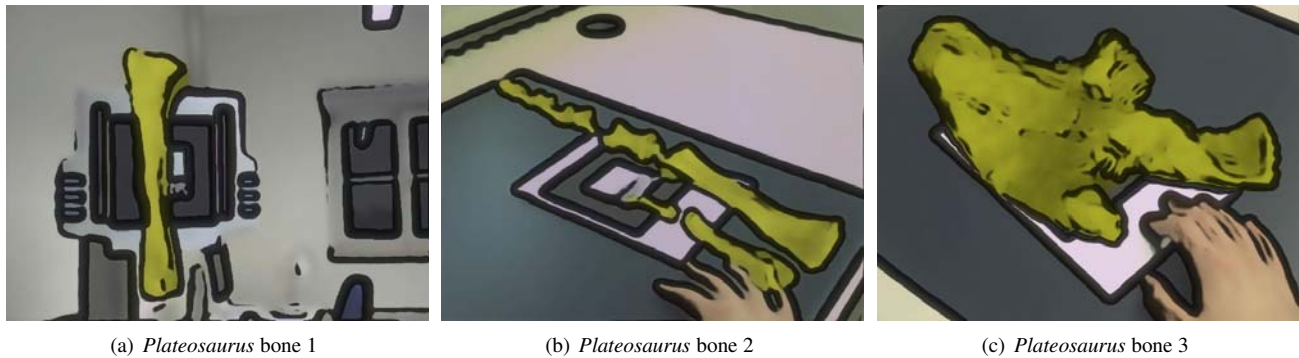


Figure 8. Visualization of dinosaur bones in augmented reality. Due to the blurred barrier between real and virtual in the stylized output video, a better immersion in the augmented environment is created.

The volume datasets containing the scanned dinosaur bones were provided by Prof. Dr. Hans-Ulrich Pfretzschner and Heinrich Mallison from the research group Vertebrate Paleontology of the Institute for Geosciences of the University of Tübingen.

Some of the graphical models used in our experiments were downloaded from the 3D Cafe website [1].

This work has been supported by project VIRTUE in the focus program on "Medical Robotics and Navigation" (SPP 1124) of the German Research Foundation (DFG).

References

- [1] 3D Cafe's Free Stuff. <http://www.3dcafe.com/>, 2005.
- [2] K. Agusanto, L. Li, Z. Chuangui, and N. Sing. Photorealistic Rendering for Augmented Reality using Environment Illumination. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 208–216, October 2003.
- [3] R. Azuma. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.
- [4] D. DeCarlo and A. Santella. Stylization and Abstraction of Photographs. In *Proceedings of ACM SIGGRAPH*, pages 769–776, July 2002.
- [5] J. Ferwerda. Three Varieties of Realism in Computer Graphics. In *Proceedings SPIE Human Vision and Electronic Imaging*, pages 290–297, 2003.
- [6] J. Fischer, D. Bartz, and W. Straßer. Stylized Augmented Reality for Improved Immersion. In *Proceedings of IEEE Virtual Reality (VR)*, pages 195–202, March 2005.
- [7] R. Gonzalez and R. Woods. *Digital Image Processing*. Prentice-Hall, 2nd edition, 2002.
- [8] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A K Peters, 2nd edition, 2001.
- [9] R. Grasset, J. Gascuel, and D. Schmalstieg. Interactive Mediated Reality (poster). In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, page 302, October 2003.
- [10] M. Haller. Photorealism or/and Non-Photorealism in Augmented Reality. In *ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry (VRCAI)*, pages 189–196, June 2004.
- [11] M. Haller, S. Drab, W. Hartmann, and J. Zauner. A Real-time Shadow Approach for an Augmented Reality Application using Shadow Volumes. In *ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 56–65, October 2003.
- [12] M. Haller and D. Sperl. Real-Time Painterly Rendering for MR Applications. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (Graphite)*, pages 30–38, June 2004.
- [13] A. Hertzmann and K. Perlin. Painterly Rendering for Video and Interaction. In *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 7–12, 2000.
- [14] M. Kanbara and N. Yokoya. Geometric and Photometric Registration for Real-Time Augmented Reality (posters and demo session). In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, page 279, September 2002.
- [15] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proceedings of IEEE and ACM International Workshop on Augmented Reality (IWAR)*, pages 85–94, October 1999.
- [16] J. Kessenich, D. Baldwin, and R. Rost. The OpenGL® Shading Language (v1.10). <http://www.opengl.org/documentation/oglsl.html>, 2004.
- [17] S. Mann and J. Fung. VideoOrbits on Eye Tap devices for deliberately Diminished Reality or altering the visual perception of rigid planar patches of a real world scene. In *International Symposium on Mixed Reality (ISMAR)*, March 2001.
- [18] W. Pratt. *Digital Image Processing*. John Wiley & Sons, 2nd edition, 1991.
- [19] R. Rost. *OpenGL Shading Language*. Addison-Wesley Publishing Company, 2004.

- [20] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 4th edition, 2003.
- [21] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics - Modelling, Rendering, and Animation*. Morgan Kaufmann Publishers, 2002.
- [22] N. Sugano, H. Kato, and K. Tachibana. The Effects of Shadow Representation of Virtual Objects in Augmented Reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 76–83, October 2003.
- [23] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *International Conference on Computer Vision (ICCV)*, pages 839–846, 1998.
- [24] I. Viola, A. Kanitsar, and M. Gröller. Hardware-Based Non-linear Filtering and Segmentation using High-Level Shading Languages. In *Proceedings of IEEE Visualization*, pages 309–316, 2003.
- [25] J. Wang, Y. Xu, H. Shum, and M. Cohen. Video Tooning. In *Proceedings of ACM SIGGRAPH*, pages 574–583, August 2004.