Non-Photorealistic Shading in an Educational Game Engine

Bert Freudenberg · Maic Masuch

Institut für Simulation and Graphik
Otto-von-Guericke-Universität Magdeburg, Germany
{bert|maic}@isg.cs.uni-magdeburg.de

An Educational Game Engine

At the University of Magdeburg we use game programming as an on-going topic in practical courses for computer science students. The 3D game engine we use is "educational" in at least two senses: Firstly, it's used by students for their projects. Secondly, we used it for a virtual walk-through that was publicly shown in a museum exhibition, educating visitors about archaeology and architecture.

One of the interesting questions to ask when using a piece of technology for teaching is if one should solely focus on the state of the art as it stands, or if one should encourage thinking outside of the box. While the former certainly is easier to accomplish, and necessary to lay the foundation for getting known to the field, we try to do the latter, too. In the context of gaming, we try to identify areas that are not yet widely explored by commercial games. One such area is the visual style of games.

Visual Styles For 3D Games

Of course, it would be misleading to state that games could not be differentiated by their visual style. This is each game company's art department's job, after all. However, a certain similarity in visual appearance emerged over the last years, largely due to the uniformity in graphics hardware features. Everyone who wanted their game to run as fast as possible had to use the same static light-map lighting, same multi-texturing, same matrix skinning as everyone else.

This has changed with the availability of programmable graphics hardware. Now the game developer is able to create a much wider variety of rendering styles with much less effort than with the fixed-function graphics boards. But what do we do with that new flexibility, if not struggling for more movie-like photorealism?

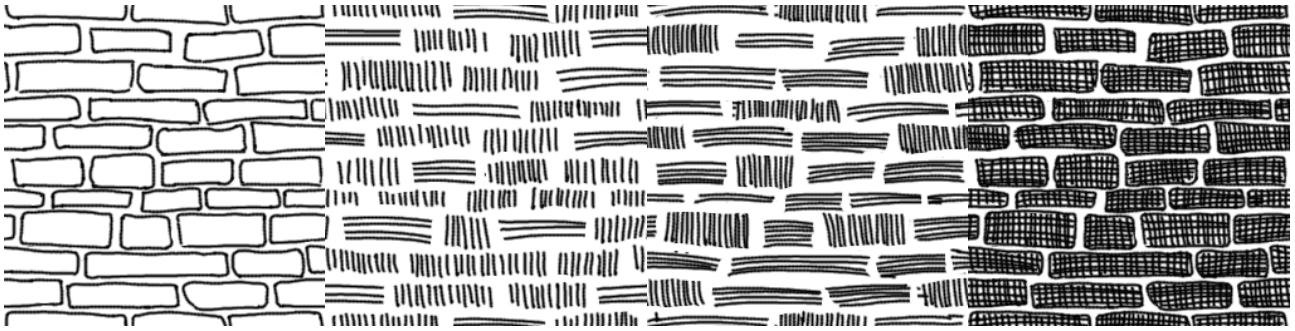
Non-Photorealism

Many ideas for alternative rendering styles can be found in the area of non-photorealistic rendering research [7]. Most of the work done to date is computationally too expensive for interactive use, but real-time methods are being developed, and hardware is always getting faster, too.

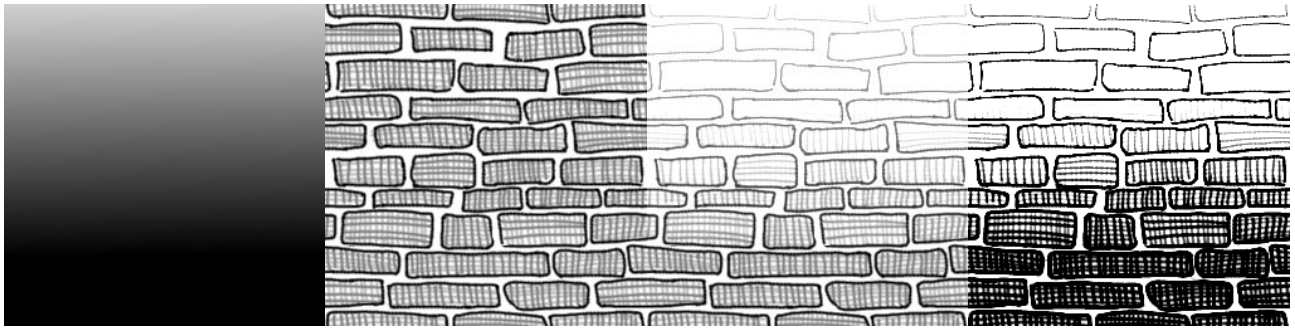
For real-time non-photorealistic shading we came up with a simple, yet expressive approach [1]. Our goal was to have "spatially distributed shading," that is, brightness variation is not expressed as varying intensity for each pixel, but rather the ratio of the number of dark and

light pixels per area is adjusted. This form of shading is common in traditional pen-and-ink drawings, engravings, or wood-block printing. Existing real-time approaches do not integrate easily with traditional rendering engines, but require modifying the renderer itself [4, 5].

We use gray-scale textures that contain strokes of varying intensity on white ground. In the image row below, the first three images are three layers from a brick-like stroke texture, while the fourth image shows all three layers at once. To assemble a "stroke texture", each layer is assigned a gray value and they are blended on top of each other. The result is shown as T in the lower image row.



At runtime, the texturing stages are configured to add the actual intensity I from lighting to these stroke textures T. This makes lighter strokes become white due to color clamping. The remaining gray-scale strokes are darkened by scaling them towards zero: $1 - c(1 - V)$ for a constant $c > 1$ and $V = I + T$ where T is the stroke texture and I is the intensity.



I T $I + T$ $1 - 4(1 - (I + T))$

In this example, we used a scaling factor of $c = 4$ which maps well to the color scaling available in current graphics boards. In OpenGL terms, this corresponds to texture environment extensions like ARB_TEXTURE_ENV_COMBINE. Because of the equality $4(1-(I+T)) = 4((1-I)-(T))$ we can do this computation in a single texture stage that is set up to SUBTRACT two sources, one of which is PRIMARY_COLOR using a ONE_MINUS_SRC_COLOR mapping $(1 - I)$ and the other is TEXTURE with plain SRC_COLOR mapping (T). The SCALE is set to 4.0 for this stage. A second texture stage is needed to invert the result, for example by using the REPLACE mode for the PREVIOUS source using a ONE_MINUS_SRC_COLOR mapping.

This simple setup still allows to produce a variety of rendering styles just by changing the textures. For example, a random noise texture yields an effect not unlike "stippling" in traditional illustrations. Additional advantages are:

- Easy-to-produce stroke textures: The textures can either be generated by a program (like in [5]), or they are drawn by hand. When drawing manually, we use a painting program to draw successive layers and do the compositing into a single stroke texture.
- Works out-of-the-box with conventional shading: The only difference to "photorealistic" shading is the texture stage setup. Lighting, be it static light-maps, vertex lighting, or

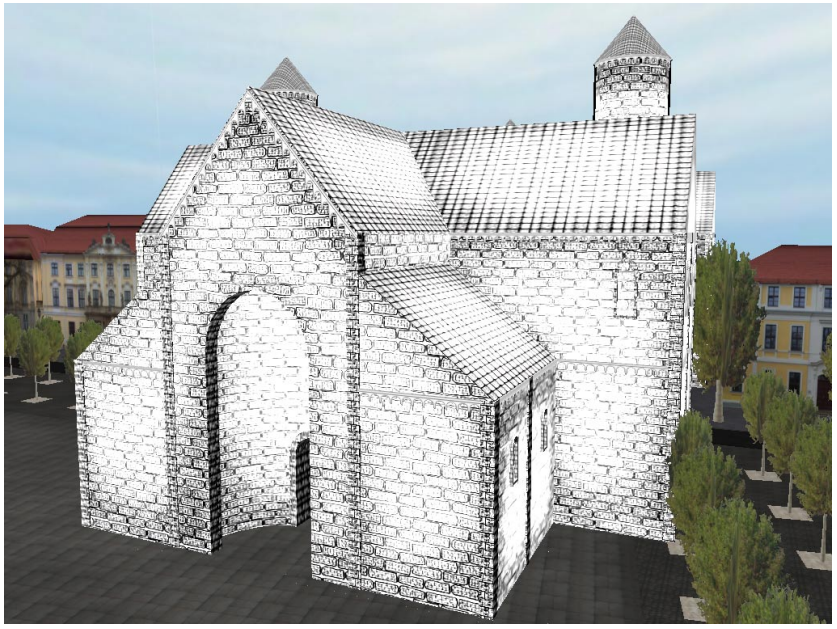
dot-product bump mapping, can be used as before (provided there are enough texture stages).

- Inherent frame-to-frame coherence: Unlike approaches that use random strokes for each frame, our technique produces a steady, undisturbing experience. It can be easily combined with mip-mapping techniques to reduce moirée patterns [3].
- No runtime overhead: Since only the texture stages have to be reconfigured, our shading runs as fast as conventional multi-texturing. No additional computations are necessary.

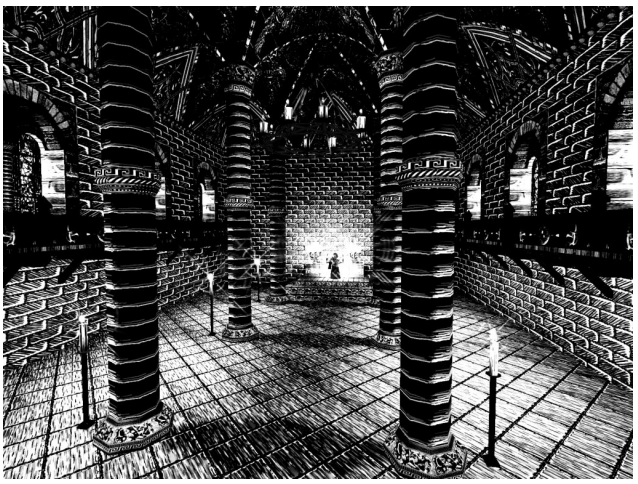
Since the stroke textures technique only supports shading surfaces, additional techniques have to be employed. For example, to render silhouette edges, a very simple technique is to render all back-facing triangles in wire-frame mode [6].

Application

We used the stroke texture shading technique in a couple of student's projects. One was a virtual reconstruction project that was exhibited in 2001 in the Museum of Cultural History, Magdeburg [2]. It involved presenting a reconstruction of a building whose only remnants were some base walls and shades in the ground. Archaeologists considered it to be misleading to show the building in a photorealistic style. Rather, we chose to model today's surroundings photorealistically, while the building itself was shown in pen-and-ink style. There is no dynamic lighting here, rather a hand-painted "indication map" was used to darken surfaces towards the edges:



Another project involved converting a photorealistic game world into pen-and-ink-style. We started with a game engine demo level that some of our students build during an internship at V_2O - visions to order GmbH. All textures were converted to gray-scale, all lights were set to white. After the texture shaders were setup for stroke map rendering we identified the textures that had to be redone (the windows, for example, were simply reused). The textures were drawn, the lighting was tuned, but not a single line of C++ code had to be written (see images on next page). We usually demonstrate this application on a GeForce2Go accelerated laptop with a 1600×1200 screen, which runs at roughly 20 frames per second, but is actually limited by CPU speed due to the dynamically updated light maps.



Conclusion

We have had a good experience using non-photorealism in a game engine. The results are both satisfying on their own and educating for our students. They learned to use existing hardware in non-usual ways. Since we have not developed our own 3D game engine yet (we are using the commercial "Shark 3D" engine by Munich based Spinor GmbH), the minimally invasive nature of the stroke texture shading method was very helpful. The ease of adopting this technique in an existing 3D application hopefully leads to a wider use of non-photorealistic real-time techniques in the industry.

References

[1] Bert Freudenberg. Real-Time Stroke Textures. In *SIGGRAPH 2001 Conference Abstracts and Applications*, page 252, 2001.

[2] Bert Freudenberg, Maic Masuch, Niklas Röber, and Thomas Strothotte. The Computer-Visualistik-Raum: Veritable and Inexpensive Presentation of a Virtual Reconstruction. In *VAST2001: Virtual Reality, Archaelogy, and Cultural Heritage*, Glyfada, Greece, 2001.

[3] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. *Computer Graphics Forum: Proceedings Eurographics 2001*, 20(3):184–191, 2001.

[4] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, pages 13–20, June 2000.

[5] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. *Proceedings of SIGGRAPH 2001*, pages 579–584, August 2001.

[6] Ramesh Raskar and Michael F. Cohen. Image precision silhouette edges. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 135–140. ACM SIGGRAPH, April 1999.

[7] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann, San Francisco, 2002. ISBN: 1-55860-787-0. 472 pages.