Real-Time Lettering on 3D Signs with a 2D Font Engine

S. Heise[1] and J. Loviscach[2]

[1]Fachbereich 03: Mathematik/Informatik, Universität Bremen, Germany
[2]Fachbereich Elektrotechnik und Informatik, Hochschule Bremen, Germany

Abstract

Standard texturing shows a number of problems with 3D objects such as road signs, labels, or books. If letters are displayed at too large a scale, textures show blurred instead of hard edges; if letters are displayed at tiny sizes, textures appear either too pixelated or too blurry, but seldom well readable. In 2D as opposed to 3D, letters are created on demand in the required size by a sophisticated font rendering engine, a standard component of today's operating systems. A number of specific improvements such as hinting and RGB subpixel rendering are available. This paper demonstrates how these partially proprietary and patented 2D functions can be leveraged for 3D rendering. The price to pay is a loss in geometric precision, since typical 2D font rendering engines only handle affine transformations, which can merely approximate perspective projection. However, in most situations this is outweighed easily by the gain in clarity.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

1. Introduction

Graphics programming interfaces such as Adobe PostScript, Microsoft GDI+, and Sun Java2D do not only offer to draw letters in an upright position at arbitrary size: They also allow setting an affine 2D mapping to which the typeface characters are subjected. The basic idea proposed in this work is to use this affine mapping to approximate the transformation needed to display letters in 3D. Since this transformation usually involves linear perspective, there will be some amount of geometrical error. However, this error can be kept imperceptibly low in situations where readability is the issue: In such situations, characters appear at small sizes, so that a local approximation through an affine mapping works well, see Figure 1. In contrast to state-of-the-art GPU-based typeface rendering methods such as [QMK06], the presented method needs no special preprocessing and inherits complex techniques from the font engine, in particular hinting and RGB subpixel rendering (also known as Microsoft ClearType). As our experiments showed, in Microsoft's GDI+ both of these techniques also work with characters subjected to an affine mapping.

Road signs and similar items are of eminent interest in professional virtual reality applications such as driver training. Thus, we focus on planar objects. A standardized description is used to store the typographic data concerning which characters to place where on these objects. As a side-effect, storing not a texture image but an actual character string facilitates replacing the text, for instance for internationalized versions, for thousands of street names, or to create a 3D newspaper with daily news from the Web.

The 3D data of the camera, the 3D data of the text's ground plus the 2D data of the characters' locations are used to compute an optimal affine mapping for each character. The 2D font engine renders the text description into an offscreen bitmap, using these mappings for the characters. The offscreen bitmap is used as a texture, which is mapped onto the text's ground, precise to the pixel. This is easy because the screen coordinates, which are available in the pixel shader, act as texture coordinates. The method can be combined effortlessly with effects such as illumination, other textures, bump mapping, shadow generation.

This work is structured as follows: Section 2 summarizes related work and Section 3 introduces the different steps of our method. Section 4 details the per-character approximation of the perspective transform through an affine mapping, whereas Section 5 gives a criterion for when a single affine mapping suffices for a complete text string. Section 6 describes the 3D rendering and Section 7 reports our findings in benchmark tests. Section 8 concludes this paper.

Figure 1: *Compared to a standard MIP-mapped texture (upper lines), a 2D font engine delivers highly readable text even at tiny sizes (middle lines: pixelated, lower lines: with font antialiasing). This image uses large pixels deliberately to show the effect. ClearType, the best method, cannot be shown in this grayscale print.*

2. Related work

Well-readable font rendering on computer screens has been researched into for decades and is considered a standard feature in GUI-based operating systems. As opposed to vector graphics, standard typefaces are equipped with "hints" to preserve line widths and other characteristics even when the letters are only a handful of pixels high [Her93].

On most desktop computer systems, antialiasing is used by default. Whereas the characters' shapes look smoother, their readability tends to suffer, however, and only Microsoft's antialiasing method ClearType seems to achieve the readability of pixelated fonts [GTAE04]. ClearType employs the RGB subpixel patterns of today's standard displays to enhance the spatial resolution [BBD*00].

In standard 3D graphics, letters are treated like texture images, which leads to an omnipresent excessive blurring and on magnification also causes jaggies. In recent years, several works have proposed techniques to create vector graphics shapes on the GPU [LB05, KSST06, QMK06] or to improve the magnification of textures that contain hard-edged shapes [TC04, Sen04, RBW04, TC05, Lov06]. All of these methods are based on not storing standard images in the texture, but augmenting or even replacing an image by data that can be used efficiently to create lines or curved shapes. If addressed, strong minification is handled through standard MIP mapping. Thus, the work presented here is largely complementary to previous work on typeface rendering in 3D.

3. Overall strategy

From the magnitude of perspective distortion (for which we give an estimate), we can decide whether a sign's text can be rendered as one string with a single affine mapping or whether it is necessary to apply a different affine mapping for each character. In the latter case, we need the exact positions of all characters. These are determined on startup as floating-point numbers; kerning information, if present in the font, is included in the positions.

Every sign contains its own bitmap where it creates its pixel image using the affine approximation in 2D. During 3D rendering, the content of this bitmap is copied into a screen-sized texture, which is used for the shader. This texture is created once on startup and is then used for all signs. To accelerate the copy operations, only the content of each sign's bounding box is transferred.

4. Local affine approximation

We want to determine the best affine mapping around the point (x_0, y_0), which is the geometric center of a character on a sign's 2D drawing surface. Let M be a 4×4 matrix in homogeneous coordinates that describes the complete transformation from the 2D coordinates $(x, y, 0, 1)^{\mathrm{T}}$ on the letter's planar background to the normalized 3D viewing volume. The matrix M can be decomposed as

$$M = \begin{pmatrix} a_{11} & a_{12} & \cdot & b_1 \\ a_{21} & a_{22} & \cdot & b_2 \\ \cdot & \cdot & \cdot & \cdot \\ c_1 & c_2 & \cdot & d \end{pmatrix},$$

where A is a 2×2 matrix, $\mathbf{b}$ and $\mathbf{c}$ are 2D vectors, and d is a real number. The third row and the third column are not relevant for the considerations that are to follow.

A 2D point (x, y) will be mapped to the normalized screen space coordinates

$$\frac{A\begin{pmatrix} x \\ y \end{pmatrix} + \mathbf{b}}{\mathbf{c} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + d}.$$

The best local approximation at (x_0, y_0) through an affine mapping can be found by forming the derivative: A point (x, y) close to (x_0, y_0) will be mapped approximately to

$$\frac{A\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \mathbf{b}}{w_0} + \frac{A\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}}{w_0} - \frac{A\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \mathbf{b}}{w_0} \cdot \frac{\mathbf{c} \cdot \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}}{w_0},$$

where $w_0 = \mathbf{c} \cdot \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + d$.

This is the affine mapping set in the 2D engine, up to two minor changes: The y direction has to be inverted as it points downward in 2D; the output coordinates are normalized and

have to be scaled and offset from their range $[-1,1]$ to the actual range $[0, \text{screen width} - 1]$ and $[0, \text{screen height} - 1]$.

5. Uniform affine approximation

In situations of little perspective distortion, it is sufficient to apply a uniform affine mapping to the complete text, see Figure 2. This saves the time to compute and set the best mapping per character. To conform best to the correct shape of the text's bounding box, we do no longer use the derivative of the perspective transformation to form the affine mapping. Rather, we compute the centers of the bounding box's edges on the screen and create an affine mapping that maps the centers of the edges of the 2D bounding box to these points.

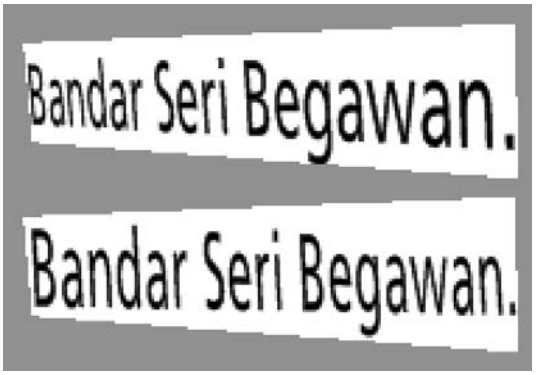

Figure 2: *Choosing the best matrix per character allows to handle perspective (top). The quicker solution is to use one affine mapping for all of the text. This will introduce a larger error, however.*

A criterion is required to decide when the distortion is low enough to allow using only a single affine mapping. To this end, we seek an estimate of the error introduced by an overall affine mapping as opposed to a per-character affine mapping. An estimate that can evaluated quickly is the second-order term of the Taylor expansion around the type's center $(x_0, y_0) = (0,0)$:

$$d^{-3}\,\mathbf{b}\left(\mathbf{c}\cdot\begin{pmatrix}x\\y\end{pmatrix}\right)^2 - d^{-2}A\begin{pmatrix}x\\y\end{pmatrix}\mathbf{c}\cdot\begin{pmatrix}x\\y\end{pmatrix}.$$

This is multiplied with the screen size; (x,y) is set to four points on the letters' bounding rectangle (top left, top mid, top right, mid right, to have four different directions). From this we get an estimated maximum error, measured in pixels. One can introduce a threshold for this value. When this is surpassed, optimal mappings should be determined on a character-by-character basis.

6. 3D rendering

Once the sign's pixels have been copied to a GPU texture, they have to rendered one-by-one into screen pixels. To this end, we employ the screen coordinates available from the position register introduced with Shader Model 3.0. There are no MIP levels, and the texture fetches use the quick nearest-neighbor sampling. For a further increase in speed, the colors could be stored as 8-bit gray scale instead of 24-bit RGB values if one does not apply ClearType.

Visible Pixels (average)	Frames per Second
75,000	330
150,000	200
300,000	150

Table 1: *For the benchmarks we used an animation containing twelve signs with ClearType.*

Even though we employ an unusual texture mapping, each sign's area is rendered as a 3D rectangle at its regular position. Thus, that there are no issues with falsely hidden or non-hidden surfaces. The colors of the typeface characters may be used like those of any regular texture: They may be combined with lighting, shadows, etc.

7. Results

A software prototype has been created in Microsoft XNA Game Studio, employing the languages C# and HLSL. Benchmarks were done on a 1800-GHz dual-core notebook computer equipped with an Nvidia 7400 GPU and running Microsoft Windows Vista. As the data show, rendering small signs with this method is a viable option, see Table 1.

Concerning the readability, the presented method easily outperforms standard texture mapping. The difference in quality is particularly high for small characters under oblique view, where a high degree of anisotropic sampling would be needed for standard textures. During our experiments, the lack of perspective distortion turned out to be only noticeable to the unwitting viewer if the fonts are viewed at large sizes under highly oblique angles, see Figure 3. This is no typical viewing condition for road signs or cockpit instruments and can thus be tolerated.

Using the 2D font engine for letters several tens of pixels large turned out to be not advisable due to two reasons: First, large letters often appear with a strong perspective distortion; second, the time to create these characters in the font engine and to transfer the large numbers of pixels to the graphics card lets this method become prohibitively slow. Thus, for large letters, standard textures or one of the GPU-based methods specialized on vector shapes should be used.

The switching from our method to such a method can happen through blending, so that for a certain range of sizes both methods will be used in parallel. Whereas we implemented blending our method to a standard texture for large sizes, we learned that this may actually not be necessary: Using the per-character affine approximation, the shapes of the letters

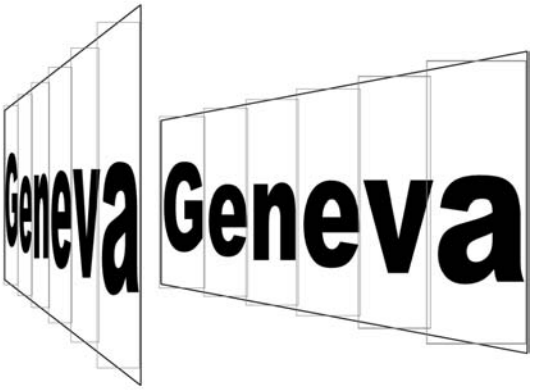

Figure 3: *There has to be considerable geometric mismatch—as visible from the gray bounding boxes of the characters—, before the per-character affine mapping produces odd-looking results.*

are matched so well that one can simply switch to the texture, without any intermediate range of blending, see Figure 4 and the accompanying video.

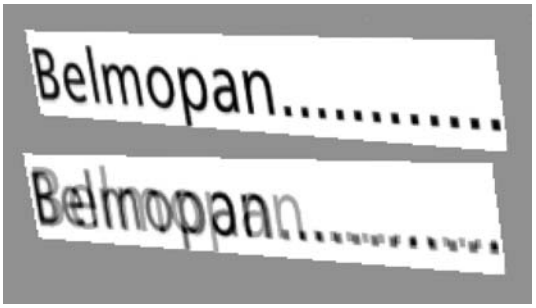

Figure 4: *To smoothen out the transition to texture-based fonts for large sizes, one may blend the texture with the result of our method over an intermediate size range. The result of the per-character approximation is so precise that blending it with the corresponding standard texture introduces only tiny changes (top). The approximation with a singe affine mapping (bottom) does not lend itself to such an approach.*

8. Conclusion and outlook

We have presented a method to produce clear lettering in 3D applications through leveraging a standard 2D font engine. It works particularly well for small font sizes.

The method can be extended to curved surfaces by taking the local tangent plane of each character into account. As opposed to the planar case, it may happen that some of the characters are hidden due to the curvature. At least for lettering on convex objects, this can be dealt with completely by per-character backface culling.

Since 2D graphics engines offer a host of high-quality antialiased drawing functions, it may make sense to also create vectors graphics similarly to the proposed font rendering. However, the elements of such a drawing may be very large and thus easily reveal the lack of perspective distortion. One option to overcome this problem is to adjust each Bézier path's anchor points instead of applying an affine mapping.

References

[BBD*00] BETRISEY C., BLINN J. F., DRESEVIC B., HILL B., HITCHCOCK G., KEELY B., MITCHELL D. P., PLATT J. C., WHITTED T.: Displaced filtering for patterned displays. In *SID Digest*. 2000, pp. 296–299.

[GTAE04] GUGERTY L., TYRRELL R. A., ATEN T. R., EDMONDS K. A.: The effects of subpixel addressing on users' performance and preferences during reading-related tasks. *ACM Transactions on Applied Perception 1*, 2 (2004), 81–101.

[Her93] HERSCH R. D.: Font rasterization: the state of the art. In *Visual and Technical Aspects of Type*, Hersch R. D., (Ed.). Cambridge University Press, 1993, pp. 78–109.

[KSST06] KOKOJIMA Y., SUGITA K., SAITO T., TAKEMOTO T.: Resolution independent rendering of deformable vector objects using graphics hardware. SIGGRAPH 2006 Sketches, 2006.

[LB05] LOOP C., BLINN J.: Resolution independent curve rendering using programmable graphics hardware. *ACM TOG 4*, 3 (2005), 1000–1009.

[Lov06] LOVISCACH J.: Rendering road signs sharply. In *Game Programming Gems 6*, Dickheiser M., (Ed.). Charles River Media, Boston, 2006, pp. 501–516.

[QMK06] QIN Z., MCCOOL M. D., KAPLAN C. S.: Real-time texture-mapped vector glyphs. In *Proc. of I3D 2006* (2006), pp. 125–132.

[RBW04] RAMANARAYANAN G., BALA K., WALTER B.: Feature-based textures. In *Eurographics Symposium on Rendering 2004* (2004), pp. 265–274.

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *SIGGRAPH/Eurographics Conference on Graphics Hardware 2004* (2004), pp. 65–73.

[TC04] TUMBLIN J., CHOUDHURY P.: Bixels: Picture samples with sharp embedded boundaries. In *Eurographics Symposium on Rendering 2004* (2004), pp. 255–264.

[TC05] TURINI M., CIGNONI P.: Pinchmaps: Textures with customizable discontinuties. *Computer Graphics Forum 24*, 3 (2005), 557–568.