Texram: A smart memory for texturing

Article *in* IEEE Computer Graphics and Applications · June 1996

DOI: 10.1109/38.491183 · Source: IEEE Xplore

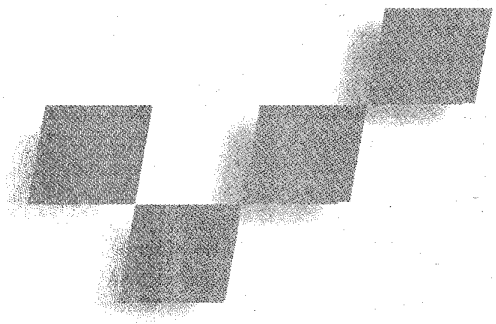
CITATIONS
78

READS
828

3 authors, including:

Andreas Schilling
University of Tuebingen
85 PUBLICATIONS **1,611** CITATIONS
SEE PROFILE

Wolfgang Straßer
University of Tuebingen
153 PUBLICATIONS **5,335** CITATIONS
SEE PROFILE



Texram: A Smart Memory for Texturing

Andreas Schilling, Günter Knittel, and Wolfgang Strasser
University of Tübingen

Logic-embedded memory is an emerging technology that combines high transfer rates and computing power. Texram implements this technology and a new filtering algorithm to achieve high-speed, high-quality texture mapping.

During rasterization, mapping textures onto objects is a problem of determining a screen pixel's projection on the texture (which we call its *footprint*) and computing an average value that best approximates the true pixel color. In real-time environments, where fast rasterizing units issue several tens of millions of pixels per second, the hardware expenses for texture mapping become substantial. System designers must therefore choose and adapt algorithms very carefully. Thus, the straightforward approach of taking the mean of all texture pixels t (or texels for short) inside the footprint for the screen pixel's color $C(x, y)$

$$C(x,y) = \frac{1}{M} \cdot \sum_{m=1}^{M} t_m \qquad (1)$$

or, more generally, of defining a filter kernel h and convolving it over the texture $t(\alpha, \beta)$

$$C(x,y) = \iint (h(x-\alpha, y-\beta) \cdot t(\alpha,\beta)) \mathrm{d}\alpha \mathrm{d}\beta \qquad (2)$$

can be excluded from further discussion because of the high computing costs.

Summed-area tables[1] attempt to simplify and speed these operations. Instead of the color value, each cell of a summed-area table holds the sum of all values in a certain region—usually the rectangle defined by the position of the cell and the origin, as indicated in Figure 1.

Given the bounding box of the footprint, $C(x, y)$ is then approximated by accessing the table four times and performing the following operation:

$$C(x,y) = T4 - T3 - T2 + T1 \qquad (3)$$

However, since a pixel's footprint is not rectangular, but can be considered as a quadrilateral in the general case, a potentially large number of texels within the bounding box contribute without reason to the pixel color. Glassner proposed a solution that incrementally removes rectangles within the bounding box to best approximate the footprint at the cost of increased computing times.[2]

Summed-area tables are not well suited to a direct hardware implementation for two reasons:

- If the color components are 8-bit quantities, a 1,024 × 1,024 summed-area table requires entries as wide as 28 bits for each color component.
- Any given pixel requires four random accesses, which limits the achievable texturing speed.

Parallelization schemes for summed-area tables show substantial disadvantages. Replicating the entire map four times leads to unacceptable memory capacity requirements. Interleaving the table across four memory banks (similar to the scheme shown in Figure 2 for level 0) would require rounding the bounding box dimensions to even numbers. This would further reduce texturing accuracy, especially for small footprints.

Another approach is to create a set of prefiltered textures, which are selected according to the level of detail (the footprint size) and used to interpolate the final pixel color. The most common method is to organize these maps as mipmaps as proposed by Williams.[3] In a mipmap, we denote the original texture as level 0. In level 1, each entry holds an averaged value and represents the area of 2 × 2 texels. We continue this until reaching the top level, which has only one entry holding the average color of the entire texture. Thus, in a square mipmap, level n has one fourth the size of level $n-1$.

The shape of the footprint is assumed to be a square of size q^2, where, for example,

$$q = \max\left(\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right) \qquad (4)$$

as suggested by Heckbert.[4] In Equation 4, u and v denote texture, and x and y denote screen coordinates.

0272-17-16/96/$5.00 © 1996 IEEE

The mipmap is accessed by the texture coordinates u, v of the pixel center and level λ, which in the general case is a function of $\log_2 q$. For example, if λ has an integer part λ_I and a fraction λ_F, λ can be written as

$$\lambda_I = \lfloor \log_2 q \rfloor \quad \text{and} \quad \lambda_F = \frac{q}{2^{\lambda_I}} - 1 \qquad (5)$$

Nearest neighbor sampling is inadequate, however, because of severe aliasing artifacts. Instead, the levels λ and $\lambda + 1$ are accessed and bilinearly interpolated at u, v. The final pixel value is linearly interpolated from the results in both levels according to λ_F.

Mipmapping is a reasonable candidate for a hardware realization because of its regular access function. If we design the memory to deliver all eight texels for a trilinear interpolation by a single access, texturing can potentially keep up with fast rasterizer units. We can accomplish this by having eight independent memory banks and a conflict-free address distribution as shown in Figure 2. Furthermore, we can reduce data traffic between the rasterizer unit and the texture system by performing the trilinear interpolation as well as all bank-address calculations locally.

At this point it becomes obvious that the ideal solution is a highly integrated memory device that incorporates all needed arithmetic units for fast mip-mapping. Logic-embedded memories have been shown to provide a quantum leap in performance in other areas such as z-buffering[5,6] or main memory caching. Taking advantage of the potentially high speed of such memory devices, we can alleviate the deficiencies arising from the square footprints. We discuss our novel additive approach for image enhancement below in "Footprint assembly."

However, a memory chip design is far too expensive to be limited to a single function. Thus, we designed Texram to provide hardware support for texture, environment, and reflectance mapping. Detail maps are a new feature that allow for magnification without image degradation. The design also addresses multimedia applications, such as video mapping and image warping, as well as volume-rendering acceleration.

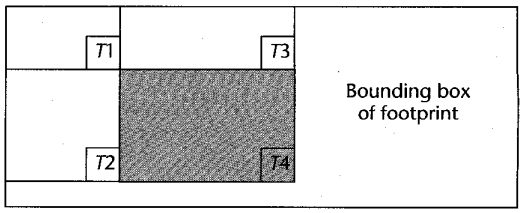
Features

The following sections describe Texram's features and underlying algorithms.

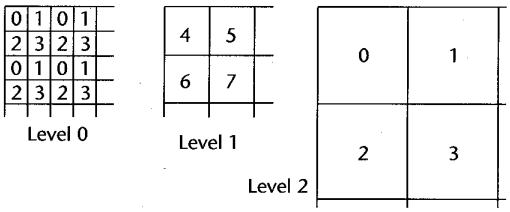
Texture mapping

Unless explicitly stated otherwise, we use the word "texture" here as a synonym for any image or structure to be mapped onto an object. Basically, the chip handles 2D textures of 8-bit quantities in that it has a single trilinear interpolator. It can be used in four different configurations:

- *Luminance mapping* uses a single Texram in the system. The texture is considered to hold gray values only.
- *Index mapping* reduces the number of different colors in a picture in an operation called *color quantization*. It is used in graphics or video systems that provide only a small number of bits per pixel, where pixel values do not directly represent a certain color but are used instead as pointers into a color look-up table. The table holds a small subset of colors that best represent the colors of the original picture. Several algorithms have been developed to construct this set.[7,8] Applied to texture mapping, the texels are used as indices into a color look-up table, and the pixel color is obtained by post-look-up interpolation. For each pixel, eight indices are read out of the memory and passed to an on-chip color look-up table that has 256 entries for red, green, blue, and opacity α (32 bits total) and acts as an eight-port memory. The color components are fed into the trilinear interpolator and returned to the graphics system one at a time.
- *True-color textures* assign a 2×2 texel space one RGBα-quadruple. Each pixel requires four accesses, and the maximum texture size per chip is reduced to one fourth. Color components are again handed out sequentially.
- *Single-color maps* treat the texture as a single color component and require three (RGB) or four (RGBα) devices in the system. This is the most powerful configuration, offering the maximum texture size with no degradation in speed.



1 Summed-area table.



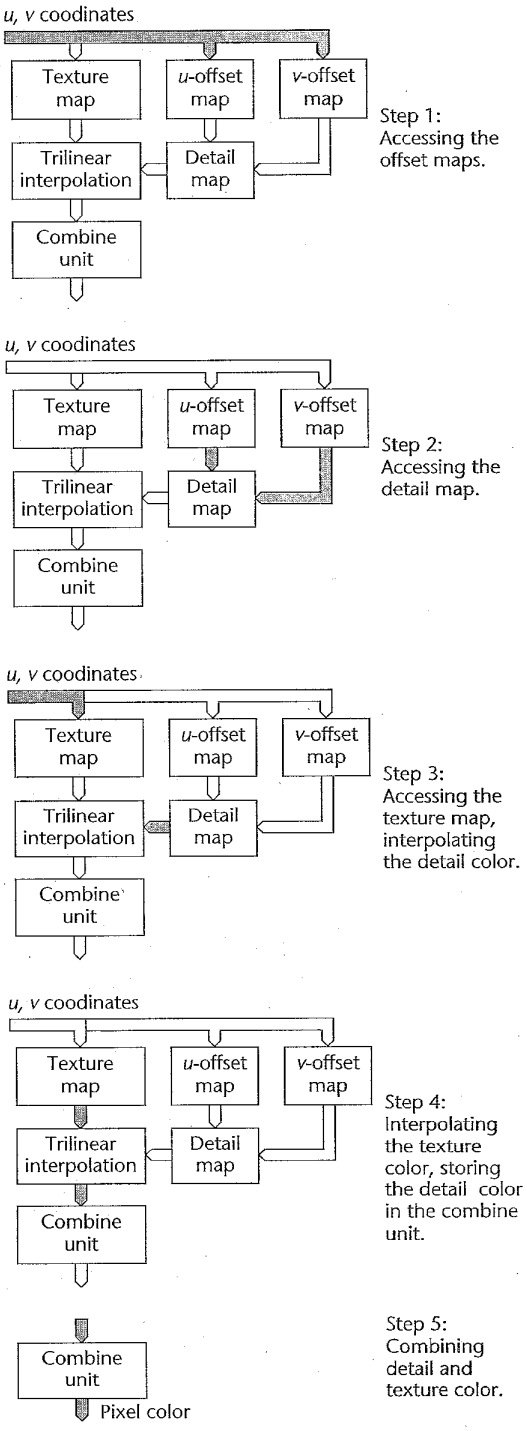
2 Assignment of texels to memory banks.

Larger textures can be distributed across multiple Texrams, and small texture patches can be replicated (tiled) over the object surface. For this reason, Texram includes a set of OpenGL-compatible border operators, such as wraparound, clamp, and fixed-border color.[9] Texture patches with distinct structures are smoothly replicated by *mirroring*, that is, the tile is mirrored about its borders.

We consider an interpolated texture value T to contribute to the diffuse portion C_d of a pixel's intensity, which is finally computed by

$$C_d = \left(I_d \mid k_T \mid 1\right) \cdot T \qquad (6)$$

where the term in parenthesis denotes a selection of either I_d, the shaded intensity of the pixel delivered by the rasterizer; k_T, a constant taken from an internal reg-



3 Using detail maps.

ister used to modify the same texture for different objects; or 1, yielding T unmodified.

Environment mapping

The environment of a given scene is projected onto the six faces of the unit cube, and mapped to each pixel using the reflected sight vector as a pointer. Accordingly, Texram controls six environment maps, each again organized as mipmaps. The design assumes that environment map coordinates and the map identifier are generated by the rasterizer on a per-pixel basis. Mapping both texture and environment onto a surface requires two Texram accesses. We consider the interpolated environment intensity E to be the specular reflected part of a pixel's intensity. The specular reflectance coefficient k_s is taken from an internal register, so we can extend the illumination model to

$$C = \left(C_d \mid I_d\right) + k_s \cdot E \qquad (7)$$

The selection says that we can map the environment directly onto the object as produced by the rasterizer.

Reflectance mapping

Instead of using a constant specular reflectance coefficient for the entire object, we can associate a reflectance map holding $k_s(u, v)$ to a texture map. This is an efficient method to model materials like wrinkled leather, varnished wood, and embossed paper. If using the interpolated reflectance coefficient R, the illumination model turns into

$$C = \left(C_d \mid I_d\right) + R \cdot \left(E \mid I_s\right) \qquad (8)$$

The last selection permits the use of any externally generated specular intensity component I_s. The reflectance map is accessed at the same coordinates as the texture map. Using the technique described later in "Map linking," this causes no additional overhead for the rasterizer.

Detail maps

Magnification at a large scale reveals the block structure of discrete pixel maps. Loading appropriate maps for each level of detail causes substantial delays in the rendering process if the magnification scale varies greatly over a single object (for example, a long wall seen in perspective) or if the object moves rapidly. In most cases, however, a small set of microstructures might be sufficient to characterize the whole texture.

Thus we introduce detail maps, which can be assigned to any texture or reflectance map. Level 0 of any map can be considered as the top level of a detail mipmap, which in turn has four levels. Each texel in level 0 covers an area of 16×16 texels in level -4 and is assigned a pointer into the associated region of the detail map. By this indirection, details can be attached to texels in the most flexible way. Physically, these pointers are two 8-bit offsets stored separately in so-called detail offset maps. Thus, the maximum detail map size is 256×256.

Detail maps, if present, are involved whenever a negative λ is received. Texture coordinates are used to access the detail offset maps, yielding the detail region coordinates of the nearest neighbor texel. Using these addresses, the detail mipmap is accessed and the color value is trilinearly interpolated. The original texture map is accessed in a third step if

- $-1 < \lambda < 0$, and thus texels from the texture are needed for the interpolation, or
- the pixel color from the detail map is to be modified using the texel entry to achieve an even higher realism.

Figure 3 shows the simplified processing sequence.

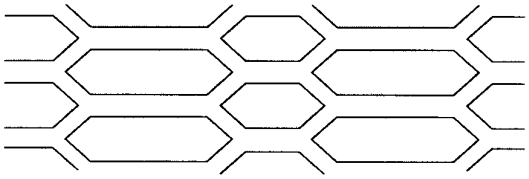
As an example, let's consider a texture of ceramic tiles and mortar joints as shown in Figure 4. As in reality, the tiles may differ slightly in color, but there might be only two micro-structures needed to magnify the texture. Thus, the detail map needs only structures for ceramic tiles, mortar, and borders of the two materials.

Figure 5a shows a magnified view of level 0 of the texture, together with the detail map offsets assigned to each texel. Figure 5b shows the entire detail map, having 80 × 64 entries in level –4. To get the coordinates of the resample location in the detail map (shown as a cross mark in Figure 5), the fractions of u and v are multiplied by 16, sign extended, and added to the entries in the detail offset maps. The remaining fractional bits are used for trilinear interpolation.

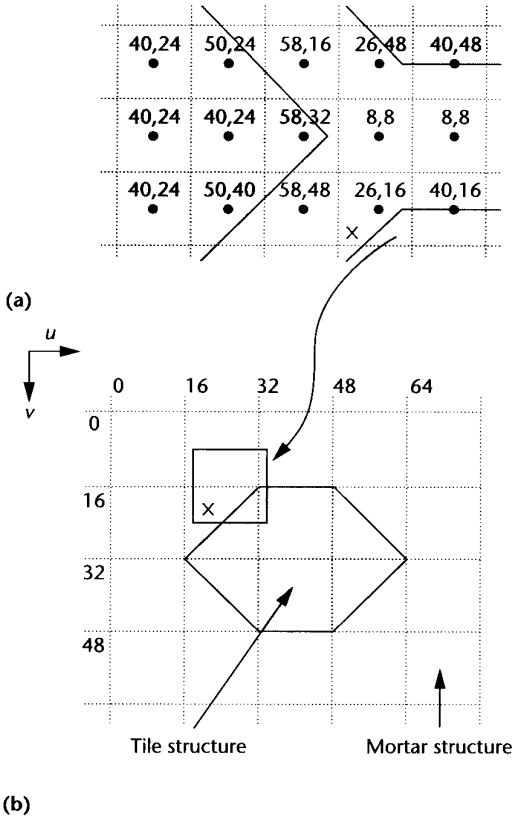
Thus, we can consider the detail map as a "pattern collection" from which—in the ideal case—we can construct the complete magnified texture from translations alone. The indirection of detail map coordinates makes it possible to magnify irregular textures without causing a coarse staircase structure on the screen.

Figure 6 shows another example of a texture and detail map combination. In this example, the detail texture has been extracted manually from the original image. However, the detail texture block selection as well as the assignment of blocks to the texels can be done automatically using suitable algorithms from texture analysis.

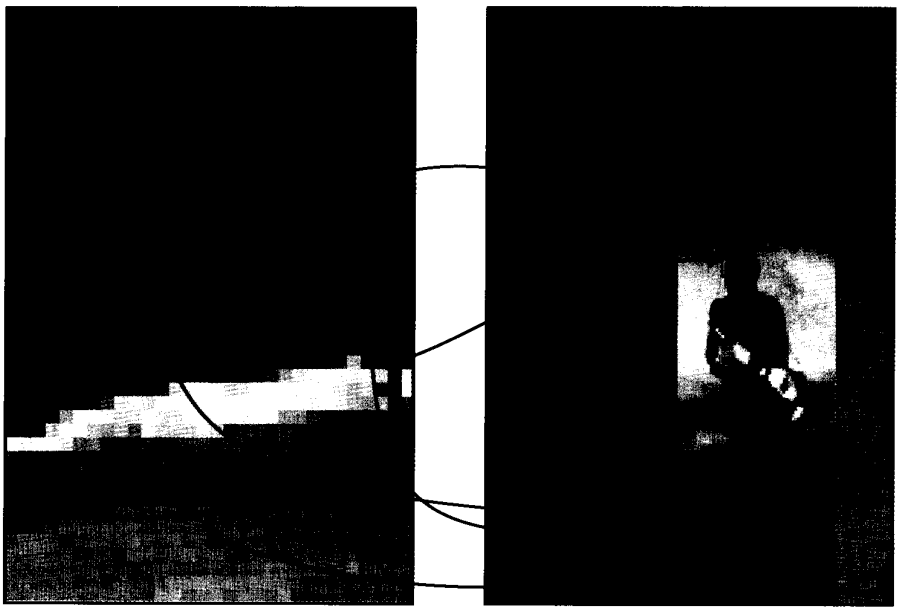
The chip takes the original texture color into account by computing



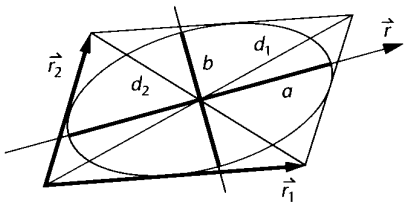
4 Texture example.



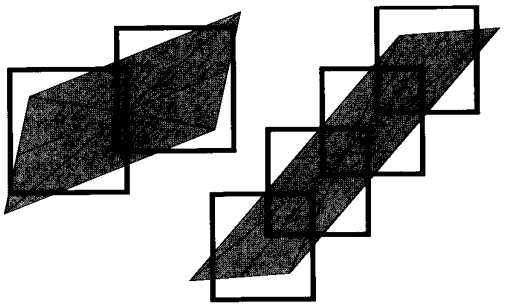
5 (a) Level 0 of the texture and (b) associated detail map.



6 Application of detail maps: Texture on the left, detail map on the right. Arrows represent pointers formed by the u- and v-offset maps.



7 Footprint geometry.



8 Footprint assembly.

9 Mapping examples: (a) mipmapped scene; (b) same scene rendered with footprint assembly.

$$C = \left(k_a + \left(1 - k_a\right) \cdot T\right) \cdot D \qquad \textbf{(9)}$$

where D is the interpolated detail map color and k_a is taken from an internal register. It is thus possible to modify the detail color with the color of the overlying texel and therefore to use the same detail map for differently colored regions of the texture.

Since offset maps have no lower resolution maps associated with them, the detail maps can in most cases be stored instead of the low-resolution mip-map levels of the offset maps. This makes efficient use of memory resources.

Footprint assembly

As a novel way of mapping textures onto surfaces, we introduce *footprint assembly*, the approximation of the pixel's projection on the texture by N square mipmapped texels. $N = 2^m$ for practical reasons, so the texels can be summed up and shifted right m places to give the final texture color. The sequence of coordinates is generated internally, so this kind of texturing is still very fast for a reasonable m. To avoid unacceptable computing times, however, the user can define an upper limit for m.

We neglect the perspective deformation of a pixel's footprint on the texture and consider as the general case a parallelogram given by

$$\vec{r}_1 = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \\ \frac{\partial v}{\partial x} \end{bmatrix} \quad \text{and} \quad \vec{r}_2 = \begin{bmatrix} \frac{\partial u}{\partial y} \\ \\ \frac{\partial v}{\partial y} \end{bmatrix} \qquad \textbf{(10)}$$

where u, v are the texture coordinates and x, y are the screen addresses of the pixel.

The pixel center $\vec{p}$ in the texture map is the intersection point of the diagonals, and we want to find the direction $\vec{r}$ in which to step from the pixel center to best approximate the footprint. Ideally, this is given by the main axis a of the ellipse, as shown in Figure 7. However, the computational expenses for finding the ideal direction (see the sidebar) are too high for real-time operation.

Thus, we propose using the larger of $\vec{r}_1$ and $\vec{r}_2$ as marching direction $\vec{r}$

$$q = \min(|\vec{r}_1|, |\vec{r}_2|, d_1, d_2) \qquad \textbf{(11)}$$

as edge length and

$$N = \frac{\max(|\vec{r}_1|, |\vec{r}_2|)}{q} \qquad \textbf{(12)}$$

rounded to the nearest power of two as the number of square mipmapped texture elements for the footprint. A difference vector $\Delta\vec{r} = (\Delta u, \Delta v)$ is constructed according to

$$\Delta u = \frac{r_u}{N} \quad \text{and} \quad \Delta v = \frac{r_v}{N} \qquad \textbf{(13)}$$

A sequence of sample points $\vec{p}_n$ is generated by

$$\vec{p}_n = \vec{p} + \frac{n}{2} \cdot \Delta\vec{r}, \text{ where } n = \pm 1, \pm 3, \pm 5, \ldots \qquad \textbf{(14)}$$

Figure 8 shows two examples. Thus, to use this mode, the rasterizer transfers u, v and λ during the first access, and $\Delta\vec{r} = (\Delta u, \Delta v)$ and N during a second access to the Texram. The Texram autonomously generates N sample locations and subsequently returns the averaged pixel color to the rasterizer. The advantage of this method is twofold:

- the rasterizer can perform the above calculations sequentially using its hardware units for perspective texture mapping multiple times, while
- the Texram assembles the previous pixel.

The enhanced image quality justifies the increased rendering time, as Figure 9 shows. Figure 9a, created by mipmapping, shows vanishing details towards the background. Figure 9b shows the same scene generated by footprint assembly with N limited to 16. The image in Figure 9b required an average of 2.3 mipmap accesses per pixel.

Video mapping

Modeling natural phenomena like fire or a water surface in real time or providing a realistic background scene like a view from a window on a busy street is nearly impossible using the surface-approximation approach. On the other hand, because it does not matter what particular scene we see and no interaction is intended, we can conveniently use image sequences captured by camera and recorded on tape.

Basically, video mapping differs from texture mapping in two ways:

- the map changes dynamically at a high rate, and therefore
- the mipmap must be generated on the fly.

The first point further implies that we must provide a separate serial pixel port through which the continuous stream of video data can enter the chip without slowing down the rasterizer accesses. For the same reason, the mipmap generation must be performed by dedicated arithmetic units.

The integration of video data opens up the wide field of multimedia applications, such as real-time zooming or rotating, image warping, and image morphing.

Real-time mipmap generation

Real-time mipmap generation is compulsory for video mapping and useful for any texture map. We use a 2×2 box filter for this purpose. For video mapping we must consider that in most cases the video source delivers interlaced images. Thus, if the actual frame consists of odd screen lines, we have to read out the corresponding even lines from the memory array to compute a line in level 1. Entries in level λ are computed as soon as all needed entries in level $\lambda - 1$ are available. They are then stored in small FIFO memories, from where they are passed to shift registers connected to the page registers of the memory banks (see "Chip architecture" below). Thus, shortly after the chip receives the last texel of level 0, it completes the generation of the mipmap. Figure 10a (next page) shows the circuitry needed to compute level 1 from both interlaced and noninterlaced sources, and Figure 10b shows the unit needed to compute level λ from the entries of level $\lambda - 1$.

Volume rendering

Using the memory arrays and the hardware units for footprint assembly and mipmapping differently, the chip

How to find the main axis of the ellipse inside the pixel footprint

Referring to Figure 7 in the main text, the derivative of the coordinate transformation from pixel space into texture space is defined by

$$\vec{r}_1 = M \cdot \vec{e}_x = M \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \vec{r}_2 = M \cdot \vec{e}_y = M \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

and thus,

$$M = [\vec{r}_1 \ \vec{r}_2] \tag{2}$$

The ellipse is defined by

$$\vec{x}^T (M^{-1})^T M^{-1} \vec{x} = 1 \tag{3}$$

$(M^{-1})^T M^{-1}$ is symmetrical and can be diagonalized:

$$(M^{-1})^T M^{-1} = R \cdot \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2} \end{bmatrix} \cdot R^{-1} = N \tag{4}$$

Its eigenvectors

$$\vec{v}_1 = R \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \vec{v}_2 = R \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5}$$

give the directions of the main axes of the ellipse. N has the eigenvalues

$$\lambda_1 = \frac{1}{a^2} \quad \text{and} \quad \lambda_2 = \frac{1}{b^2} \tag{6}$$

We can save the calculation of M^{-1} by using $M^{-1} \det(M)$ instead, which can be derived from M by reordering the matrix coefficients without division.

$$N' = N \det(M)^2 \tag{7}$$

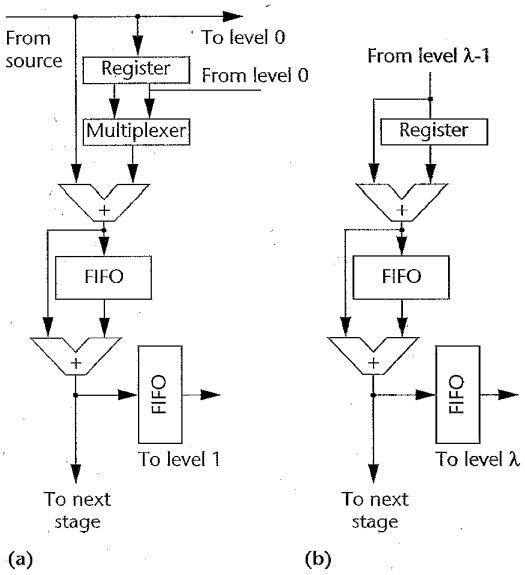
has the same eigenvectors as N:

$$\vec{v}_{1'} = \vec{v}_1 \quad \text{and} \quad \vec{v}_{2'} = \vec{v}_2 \tag{8}$$

with the eigenvalues

$$\lambda_{1'} = b^2 \quad \text{and} \quad \lambda_{2'} = a^2 \tag{9}$$

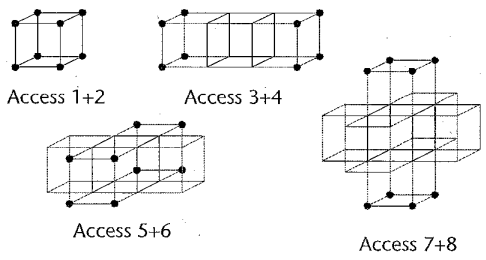
The main axis of the ellipse has the direction of the eigenvector with the smaller eigenvalue.



10 Mipmap generation circuitry: (a) stage 0 and (b) stage $\lambda - 1$.

11 An RGBα-data set of 1 million samples.

12 Skin and tissue blended with bone.



13 Access types for rendering original data.

turns into a complete ray-casting machine for volume rendering. As such, it can generate evenly spaced resample locations along an arbitrarily oriented viewing ray, read the eight neighboring voxels by two consecutive accesses, and trilinearly interpolate the function value at each resample point. (Note that only the four larger memory banks holding level 0 are used for volume rendering.) Using pre-shaded and pre-classified data sets (that is, data sets containing RGBα), the remaining step for the visualization is just α-blending, which can easily be done by the host or by one or more digital signal processors (DSPs) for real-time operation.

In a four-chip configuration, RGBα-datasets can have up to 1 million samples. As an example, see the data set of a human head in Figure 11, which consists of a stack of 64 computed tomography (CT) scans of 128×128 samples.

In Figure 12, the same view is shown with the skin and tissue translucently rendered over the bone. Both images were created by sending 128×128 rays through the data. The resulting images were bilinearly interpolated to a 512×512 screen resolution.

For a more flexible visualization or interactive segmentation, the chip can operate on the original samples and return both the interpolated function value and the local gradient for each resample location. In this mode, eight accesses are performed for each resample location as shown in Figure 13. After accesses 1 and 2, all data is available for trilinearly interpolating the function value. Accesses 3 and 4 yield all additional values needed to compute the x components of the gradient at the original sample locations. These quantities are then fed into the trilinear interpolator to give the local x gradient at the resample location. The y and z components are computed the same way after accesses 5 through 8. The special address interleaving scheme ensures that each access is nonconflicting. In other words, the Texram can read all four associated voxels out in parallel. In a four-chip configuration, original data sets can be as large as $256 \times 128 \times 128$ 8-bit voxels.

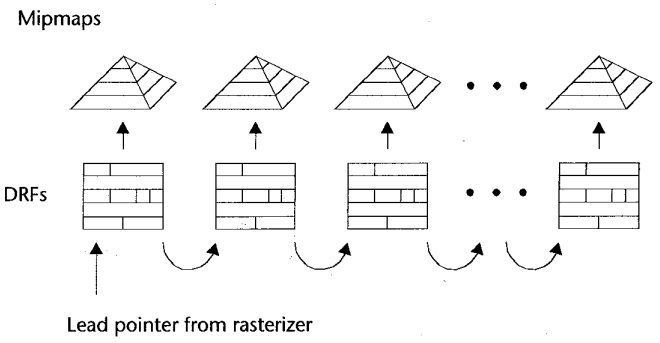
Map linking

To handle up to seven maps involved in coloring a single pixel, we use a linked list of map description register files (DRFs). In the general case, a texture can be linked to the u, v-offset map, finally pointing to the texture detail map. This map in turn points to a reflectance map linked to offset and detail maps followed by the environment map. Even a video map can be linked to an environment map to model, for example, a TV screen reflecting a light source.

Thus, there are seven DRFs on chip. Each DRF holds the physical offset of the map on the memory array, along with its size and type, links, border operation and color, information about whether or not new texel coordinates are to be received prior to an access, and various constants for the evaluation of the illumination equation as explained earlier in the discussions of environment and reflectance mapping and detail maps. DRF number 7 is special in that it holds six offsets and sizes for the environment map.

Thus, the set of DRFs can be considered as micropro-

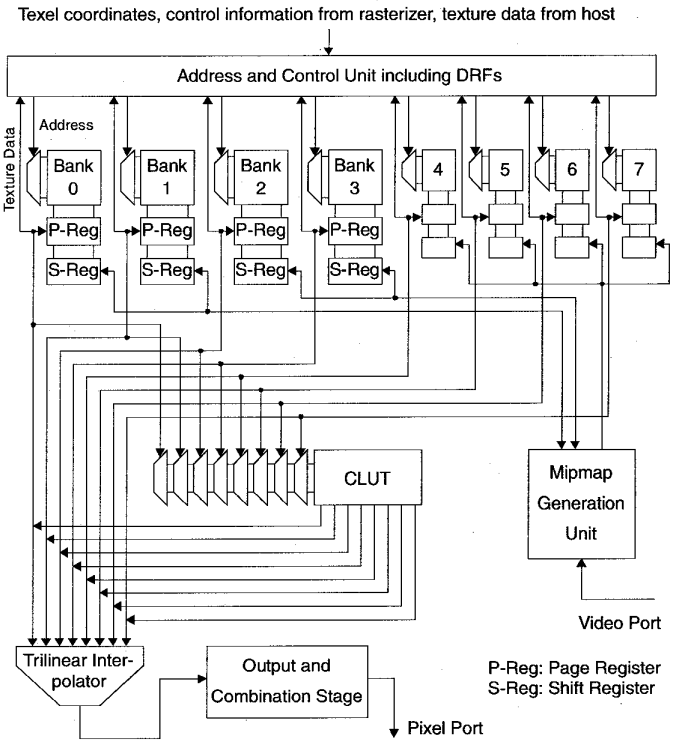
gram storage. The rasterizer transmits the pointer to the leading DRF along with each pixel, and the Texram proceeds down the chain until a new parameter is needed or the pixel is finished.



14 Map linking using description register files.

Chip architecture

One chip provides storage for a 1,024 × 1,024 single-component mipmap. The capacity of the memory arrays sum up to 11,239,424 bits, and thus Texram uses 16-Mbit DRAM technology. The memory system consists of four large arrays of 274 × 8,192 bits, holding the even levels, and four small arrays of 69 × 8,192 bits for the odd levels. A pipelined control unit, which encloses the DRFs, generates all addresses and controls internal operation as well as data flow to and from the rasterizer and the graphics system. The trilinear interpolator is designed for a 6-bit fraction of the texture coordinates. A simplified schematic of the chip appears in Figure 15.



15 Architecture of the Texram chip.

Any random access to a DRAM array takes place in two steps: First, the addressed row (or page) must be loaded completely into a page register (row access), from where the desired data item can then be accessed in a second step (column access). If the following memory cycle refers to the same row, the row access can be skipped, since the data still exists in the output register (page-mode access). To ensure that most accesses are page-mode cycles, we place the texels of rectangular regions into one page (64 × 64 texels in level 0).

The textures are loaded from the host via the texel coordinates port. After specifying a start address, four texels can be written into the chip per access. Consecutive addresses are generated internally.

Performance

Because of the pipelined architecture, Texram performance depends only on the memory access time. In page mode, an access takes 30 nanoseconds, which results in a peak performance of 33 million textured pixels per second. A page fault causes a delay of another 30 ns. We achieved this small value (about one fourth of conventional DRAMs) by exploiting the fact that data read from memory is not modified, which means that the memory can be prepared for the next read access immediately after the actual one. Thus, as a rather pessimistic example, if a scanline averages eight pixels and two page faults, we get a sustained performance of about 27 million textured pixels per second.

These figures apply to luminance, index, and single-color mapping. The performance is about one fourth in the case of all four color components on a single chip.

For footprint assembly the performance further scales down by the average number of mipmap accesses per pixel. Additionally, the number of page faults per scanline will increase.

Access to a linked map will probably cause a page fault. Using a detail map in conjunction with a texture map will take about 240 ns per pixel, which results in an output rate of about 4 million pixels per second. In the extreme case, linked texture and reflectance both visit their detail maps before accessing the environment map, which reduces the generation rate to about 1.8 million pixels per second.

Finally, volume rendering of RGBα data sets takes two memory accesses per resample location. Thus, in a four-chip configuration, rendering a 64 × 128 × 128 data set

takes about 70 milliseconds, assuming one resample location per volume element. Trilinear reconstruction and gradient estimation of original data sets take eight accesses per ray point, with an increased likelihood for page faults. Simulations show an average rendering time of 0.28 seconds for $256 \times 128 \times 128$ data sets in a parallel four-chip system.

Future work

The techniques discussed so far allow us to map images on smooth surfaces. Most real objects, however, show geometrical structures such as dimples or wrinkles on their surface. *Bump mapping* is a technique to display such surfaces without having to model the structures geometrically. Thus, the visual appearance of the objects is greatly enhanced without causing the number of surface elements to explode.[10]

The surface structures are generated by perturbing the surface normals according to angular deviations stored in a so-called *bump map*. Using these de-adjusted surface normals, the illumination calculations produce reflections as if the structures had been modeled explicitly.

However, we postponed bump mapping to the next generation design, since its computing demands exceed the current technological possibilities. Our approach is as follows:

- Create a local coordinate system $\vec{n}, \vec{e}_1, \vec{e}_2$ from the interpolated normal vector $\vec{n}_I$ issued by the rasterizer and a "main direction" $\vec{h}$. The computation of the two vectors perpendicular to $\vec{n}$ is an expensive computation, since it requires normalizations and outer products:

$$\vec{n} = \frac{\vec{n}_I}{\left|\vec{n}_I\right|} \quad \vec{e}_1 = \frac{\vec{h} \times \vec{n}}{\left|\vec{h} \times \vec{n}\right|} \quad \vec{e}_2 = \vec{n} \times \vec{e}_1 \tag{15}$$

- Calculate the new normal vector $\vec{n}_B$ from $\vec{n}$ and the map entries b_1 and b_2:

$$\vec{n}_B = \vec{n} + \vec{e}_1 \cdot b_1 + \vec{e}_2 \cdot b_2 \tag{16}$$

- Calculate the reflected view $\vec{v}_R$ vector from the unit view vector $\vec{v}$ and the new normal vector $\vec{n}_B$. The reflected view vector need not be normalized.

$$\vec{v}_R \cdot \left|\vec{n}\right|^2 = 2 \cdot \vec{n} \cdot (\vec{n} \bullet \vec{v}) - \vec{v} \cdot \left|\vec{n}\right|^2 \tag{17}$$

- Calculate the environment map coordinates from the reflected view vector by dividing by the largest component of the vector. The effort is comparable to the effort for the perspective correct texture coordinate generation.

A dedicated smart memory device for bump mapping, located between the rasterizer and the Texram, would maintain the high rendering speed, since the environment coordinates could then be modified in parallel with texturing.

Conclusion

Integrating arithmetic units and large memory arrays on the same chip and thus exploiting the enormous internal transfer rates provides an elegant solution to the memory access bottleneck of high-quality texture mapping. Using this technology, we can not only achieve higher texturing speed at lower system costs, we can also incorporate new functionalities such as detail mapping and footprint assembly to produce higher quality images at real-time rendering speeds. Environment and video mapping are also integrated on the Texram, which therefore represents an autonomous and versatile texturing coprocessor.

Logic-enhanced memories might become the computing paradigm of the future, not just in graphics applications. Technological advances will foster this trend by providing an ever increasing amount of memory capacity and chip space for arithmetic units. As the ultimate solution, we can expect a complete 3D graphics pipeline including all memory systems integrated on a single chip. ■

Acknowledgments

This work was done in the Monograph project, supported by the CEC's ESPRIT program. We would like to thank our project colleagues M. Malms, J. Haas, and J. Binder from the IBM Development Laboratory in Böblingen, Germany, for their support and profound technical knowledge.

References

1. F.C. Crow, "Summed-Area Tables for Texture Mapping," *Computer Graphics* (Proc. Siggraph), Vol. 18, No. 3, July 1984, pp. 207-212.
2. A. Glassner, "Adaptive Precision in Texture Mapping," *Computer Graphics* (Proc. Siggraph), Vol. 20, No. 4, Aug. 1986, pp. 297-306.
3. L. Williams, "Pyramidal Parametrics," *Computer Graphics* (Proc. Siggraph), Vol. 17, No. 3, July 1983, pp. 1-11.
4. P. Heckbert, "Texture Mapping Polygons in Perspective," NY Inst. Tech. Computer Graphics Lab, Tech. Memo #13, Apr. 1983.
5. M.F. Deering, S.A. Schlapp, and M.G. Lavelle, "FBRAM: A New Form of Memory Optimized for 3D Graphics," *Computer Graphics Proc.* (Siggraph 94), ACM, New York, 1994, pp. 167-174.
6. G. Knittel and A. Schilling, "Eliminating the Z-Buffer Bottleneck," *Proc. European Design and Test Conf.*, Paris, IEEE Computer Society Press, Los Alamitos, Calif., 1995, pp. 12-16.
7. M. Gervautz and W. Purgathofer, "A Simple Method for Color Quantization: Octree Quantization," in *New Trends in Computer Graphics*, N. Magnenat-Thalmann and D. Thalmann, eds., Springer-Verlag, New York, 1988, pp. 219-231.
8. P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics* (Proc. Siggraph), Vol. 16, No. 3, 1982, pp. 297-307.
9. Silicon Graphics, *OpenGL Reference Manual*, Addison-Wesley, Reading, Mass., 1992.
10. A. Watt and M. Watt, *Advanced Animation and Rendering Techniques*, Addison-Wesley, Reading, Mass., 1992, pp. 199-202.

Andreas Schilling *is engaged in teaching and research on computer graphics and vision at the University of Tübingen, Germany. His research interests include architectures for real-time rendering, antialiasing, texture mapping, and computer vision. Schilling received his MS in physics in 1988 and his PhD in computer science in 1995, both from the University of Tübingen.*

Günter Knittel *is a PhD candidate at the Computer Graphics Lab (WSI/GRIS) of the University of Tübingen. His research interests include VLSI architectures and hardware systems for computer graphics, visualization, and parallel processing. Knittel received a master's degree (Dipl.-Ing.) in electronics engineering from the Technical University of Berlin. He is a member of the IEEE Computer Society.*

Wolfgang Strasser *is full professor of computer science and adjunct professor of mathematics at the University of Tübingen. His graphics group (WSI/GRIS) consists of about 20 researchers working in the area of graphics system design, graphics hardware, visualization, rendering, and geometric modeling. He is a member of IEEE, Eurographics, and the Gesellschaft für Informatik.*

Readers may contact the authors at the University of Tübingen, WSI/GRIS, Morgenstelle 10, D-72076 Tübingen, Germany, e-mail {andreas, knittel, strasser}@gris.informatik.uni-tuebingen.de.