A Transform, Lighting and Setup ASIC for Surface Splatting

Simon Heinzle, Olivier Saurer, Sebastian Axmann, Diego Browarnik,
Andreas Schmidt, Flavio Carbognani, Peter Luethi, Norbert Felber, Markus Gross
ETH Zurich, Switzerland, Email: simon.heinzle@inf.ethz.ch

***Abstract*— This paper presents the first ASIC implementation of the transform, lighting and setup stages of the elliptical weighted average (EWA) surface splatting algorithm, a high quality method for anti-aliased rendering of point sampled objects in computer graphics. The algorithm has been integrated on a small core size of 8.15 mm^2 in a 180 nm process using massive resource sharing. It achieves the high throughput of 2.94 million points per second at the operating frequency of 147 MHz, with a power dissipation of 300mW.**

I. Introduction

In recent years the polygonal complexity in computer graphics has exploded, while the number of screen pixels has only grown moderately. This trend has lead to an ever and ever increasing ratio of polygons to screen pixels. As a result, tiny triangles often fall between sampling points and get lost during rasterization, thus causing unwanted aliasing artifacts. Therefore, triangles may not be the optimal rendering primitive in all cases.

Moreover, constructing optimal triangle meshes from models is very costly. Current and next-generation 3D camera systems, for example, are able to capture three dimensional scenes very similar to photographs, by combining a two dimensional color sampling with a depth sampling of a scene. Immediate, high-quality rendering of the captured scene is a key issue for mobile 3D camera devices. However triangulating the aquired point samples combined with traditional polygon rendering leads to very slow frame rates. Therefore, directly displaying the point samples generated from such 3D cameras could be a better option.

Elliptical weighted average surface splatting [1] addresses the issue of high-quality rendering of point samples as an alternative to triangle based rendering. It also comes with implicit anti-aliasing under minification and it does not suffer from deficiencies stemming from very small points. In EWA splatting, a point sample is represented as a disk-shaped colored region with a fuzzy influence on neighboring points.

The first hardware accelerated geometry system for computer graphics has been presented in [2], a relevant follow-up work focused on triangle rasterization [3]. But also other rendering paradigms, such as ray tracing, have been investigated for hardware architectures [4]. However, the predominant rendering technique on the graphics processor market is still triangle rasterization, due to its simplicity, matureness and good characteristics in terms of memory bandwidth.

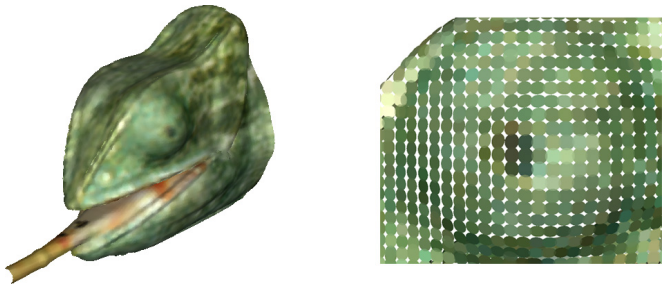

Fig. 1. Left image shows the head of a chameleon model, rendered using EWA surface splatting. The right image shows a close-up of the eye, with reduced point radius to illustrate the point samples.

Although implementations of EWA surface splatting for commercial, general purpose graphics processors (GPGPUs) exist [5], they still suffer from incompatibilities in the rasterization stage. Therefore, they exhibit poor performance when considering the massive silicon resources present in such graphics accelerators. In the recent publication [6], a dedicated hardware architecture for EWA surface splatting has been introduced. While the rasterization unit of the system in [6] was implemented as a custom ASIC design with fixed point arithmetic, the floating point computations of the transform, lighting, and setup stages were committed to four digital signal processors (TigerSHARC ADSP-TS201S by Analog Devices).

This paper presents the first ASIC implementation for the transform, lighting, and setup stages of the architecture [6]. It features full floating point arithmetic and exploits much less resources in terms of chip surface and power dissipation than the signal processing units of [6], while achieving similar performance.

II. EWA Surface Splatting

This section briefly reviews the concept of EWA surface splatting, which is essential for understanding the hardware design. Point samples – or splats – are represented by their position $\mathbf{c}$ in space, two tangent axes $(\mathbf{u}, \mathbf{v})$ spanning an elliptical reference system, and a diffuse color $\mathbf{d}$. Splats can be considered as overlapping elliptical disks in space.

More specifically, the tangent axes span an elliptical Gaussian reconstruction kernel on top of the disk. The kernel leads to a smooth blending of overlapping splats for high quality results. To avoid the problem with anti-aliasing under minification, a band-limiting pre-filter is applied to each splat.

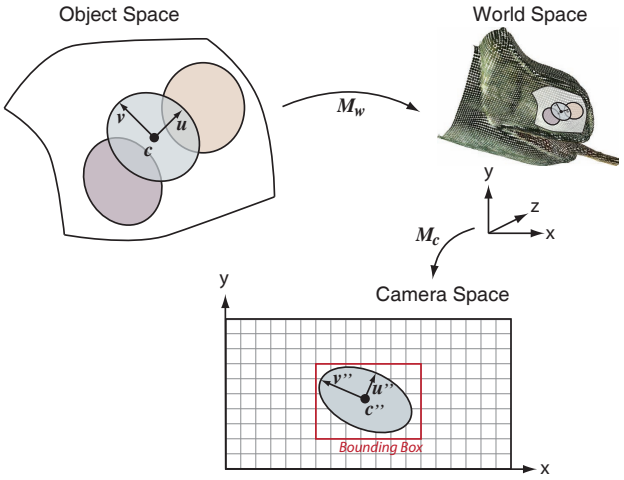

Fig. 2. Mapping of point samples from object space to screen. Point samples are defined in the geometric space of their object. The points are first transformed to world space using the object-to-world-space transformation matrix $\mathrm{M_w}$, that is positioned in the scene. Finally the points are transformed into the coordinate system of the camera and projected onto the screen using the world-to-camera-space projection matrix $\mathrm{M_c}$.

This filter is also Gaussian, hence it can easily be convolved with the blending Gaussian.

A. Transform and Lighting

In the Transform and Lighting stage, similar to triangle-based rendering, all splats are transformed into the camera space and the lighting equation is evaluated on each splat. See Figure 2 for an illustration. The concepts of the different coordinate systems as well as its transformations are similar to triangle rendering pipelines [7].

A model is represented by the point samples $\mathcal{S} = \{(\mathbf{c}_i, \mathbf{u}_i, \mathbf{v}_i, \mathbf{d}_i)\}$. First, the splats are transformed from their object space to world space, using the homogeneous transform matrix $\mathrm{M_w}$. Then the lighting equation is evaluated per splat, where the angle between light source direction and splat normal defines the amount of light hitting the surface. All point samples are then transformed from world space into camera space, using the homogeneous transform $\mathrm{M_c}$.

An overview on the arithmetic complexity of the transform and lighting stage can be found in Table I.

B. Splat Setup

After all points are transformed to camera space, in a first step the tight, axis aligned bounding box is evaluated. This bounding box surrounds the splat and limits its extent in screen space (see Figure 2). The camera-space ellipse, represented by the tangent axes $\mathbf{u}''$ and $\mathbf{v}''$, is then expressed using the general quadratic equation of a conic section:

$$f(x,y) = a\,x^2 + 2b\,xy + c\,y^2 \leq 1. \qquad (1)$$

Finally, the pre-filter is applied. An overview on the arithmetic complexity of the splat setup stage can be found in Table I.

C. Splat Rasterization

The rasterizer finally draws the ellipse into the rendering memory. For each pixel inside the bounding box, the rasterizer evaluates Equation 1 with $f(x - \mathbf{c}''_{i,x}, y - \mathbf{c}''_{i,y})$. This result defines a relative distance of the pixel to the center of the ellipse. If $f \leq 1$, then the pixel is inside the ellipse and the influence of the splat onto this pixel is weighted using the filtered Gaussian reconstruction kernel, and accumulated in case of overlapping splats.

Due to the overlap of the splats, multiple splats can contribute to a single pixel with different weights. Therefore, all pixels need to be normalized by the sum of weights at the end of a rendering cycle. The overlap is also one of the most relevant differences to opaque polygon rendering, where only one polygon contributes to a pixel.

III. Implementation

A high-level overview of the architecture can be found in Figure 3 while a photograph of the chip is shown in Figure 4.

Configuration and splat data are sent to the input block using a handshake direct memory access (DMA) protocol: if the input buffers and the chip are ready, the sender is then allowed to send bursts of up to 512 words. The input buffers include a dual-port RAM of size 512 × 32 bit.

A parameter register bank stores constant scene data: the screen space filter size, cut-off radius, screen width and height, the World to Screen 4 × 4 homogeneous transformation matrix and the camera position.

The "World to Screen" block computes the transformation of all splats, the "Lighting & Culling" block evaluates the

Transform and Lighting	
$\mathbf{c}'_i = \mathrm{M_w}\mathbf{c}_i$	Object to world transformation
$\mathbf{u}'_i = (\mathrm{M_w^{-1}})^\top\ \mathbf{u}_i$	
$\mathbf{v}'_i = (\mathrm{M_w^{-1}})^\top\ \mathbf{v}_i$	
$\mathbf{n}'_i = \mathbf{u}'_i \times \mathbf{v}'_i$	Determine normal
$\mathbf{v}_i = \mathbf{c}_{\mathrm{camera}} - \mathbf{c}'_i$	Determine viewing direction
$\mathbf{v}'_i \cdot \mathbf{n}'_i \leq 0 \rightarrow$ discard	Back-face culling: discard points facing away
$\mathbf{d}'_i = \frac{\mathbf{c}_i \cdot l_i}{\|\mathbf{c}_i\|\|l_i\|}\mathbf{d}_i$	Evaluate lighting equation
$\mathbf{c}''_i = \mathrm{M_c}\mathbf{c}'_i$	Transform from world to camera space
$\mathbf{u}''_i = (\mathrm{M_c^{-1}})^\top\ \mathbf{u}'_i$	
$\mathbf{v}''_i = (\mathrm{M_c^{-1}})^\top\ \mathbf{v}'_i$	
Splat Setup	
$r_{i,d} = o\sqrt{u_{i,d}^2 + v_{i,d}^2} + f$	Extent of screen space splat. o is cut-off radius, f is screen space filter size
$\mathrm{bbmin}_{i,d} = r_{i,d} - v_{i,d}$	Determine the bounding box
$\mathrm{bbmax}_{i,d} = r_{i,d} + v_{i,d}$	
$\mathrm{M} = \begin{pmatrix} u_x & v_x \\ u_y & v_y \end{pmatrix}$	Parameters of quadratic equation (conic matrix)
$\begin{pmatrix} A & B \\ B & C \end{pmatrix} = \mathrm{MM}^\top + \begin{pmatrix} f & 0 \\ 0 & f \end{pmatrix}$	Extend quadratic equation by screen space filter size
$\mathrm{Q} = \frac{1}{AC+B^2}\begin{pmatrix} C & -B \\ -B & A \end{pmatrix}$	Determine conic matrix
$\mathbf{x}^\top \mathrm{Q} \mathbf{x} = 1$	Quadratic equation, with $\mathbf{x} = (x,y)^\top$

TABLE I
Arithmetic operations.

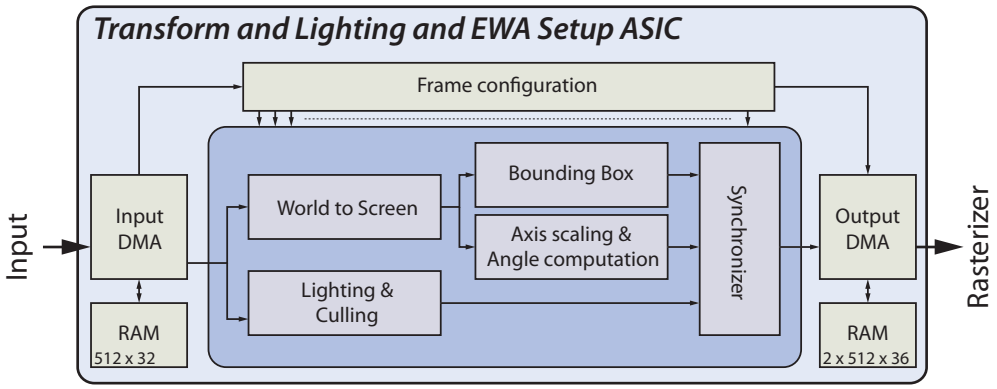

Fig. 3. High level overview of the architecture.

lighting equation, and discards splats facing away from the camera. The "Bounding Box" block determines the screen space extent of a splat. To save bandwidth communication to a subsequent rasterizer ASIC, the quadratic equation is transformed in the "Axes scaling & Angle computation" block for better accuracy using fewer bits.

A synchronizer gathers all the information from the separate blocks and stores them into output block using two dual-port RAMs of size 512×36. The output block uses the same DMA protocol as the input block.

A. Axes Scaling and Angle Computation for Better Accuracy

The parameters of Equation 1 have quadratically inverse influence on the size of the ellipse. This makes it hard to control the error of the ellipse in fixed point representation.

To control and reduce the error, the explicit properties of the ellipse (length of the main axes and rotation) are determined and transmitted. This way, the error is limited effectively: using an 8 bit fixed point representation for the rotation angle, the error is at most $0.35°$. For the length of the axes, a custom floating point format with 8 bits exponent and 7 bits mantissa has been used, to limit the error relatively to the size of the splat.

The axis scalings are determined by the eigenvalues of the matrix Q, defined in Table I. The rotation of the ellipse is determined by the angle of an eigenvector of the matrix Q to the x-axis. Note that the angle computation requires the evaluation of an arcus tangent, the implementation of which will be discussed in Section III-C.

B. Resource Sharing

Unfortunately, EWA rasterization setup computation cannot be bound numerically and needs to be carried out in floating point precision. The proposed implementation uses the IEEE 754 single precision number format for all calculations. The isomorphic implementation of the algorithm would require a massive amount of floating point units: 106 adders, 137 multipliers, 12 multiplication inverses and 4 inverse square roots.

However, the algorithm exhibits potential for arithmetic reuse at a high level. For example, the "World to Screen" function performs similar computations on the position and the tangent axes. Additionally, the "Bounding Box" block performs similar computations on the separate dimensions. Applying high-level resource sharing, the number of final floating point units used in the design reduces to 46 adders, 70 multipliers, 5 multiplication inverses and 2 inverse square roots.

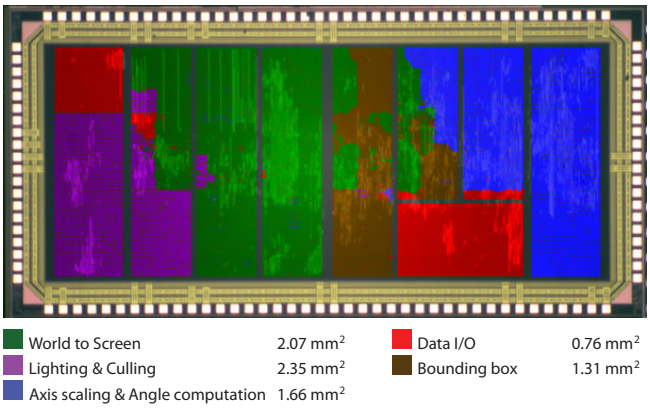

Fig. 4. Die photograph highlighting the different blocks of the architecture.

C. Arcus Tangent Implementation

The arcus tangent is a highly non-linear function. For high accuracy, a polynomial approximation would require a large amount of hardware, whereas iterative algorithms such as the Cordic, would need many iterations.

As the output accuracy of the arcus tangent for this system is fixed to 8 bit, a look-up table approach has been preferred. The symmetry of the arcus tangent function, i.e. $\arctan(-x) = -\arctan(x)$, allows the reduction of the look-up table by half. Naively storing the arcus tangent using the x values as indices and the function values as data leads to large errors, especially near $x = 0$, where the arcus tangent function is steeper (see Figure 6). The better solution is to use an inverse look-up table approach: the function values of the arcus tangent are used as indices into the look-up table, whereas the associated x values are stored as data.

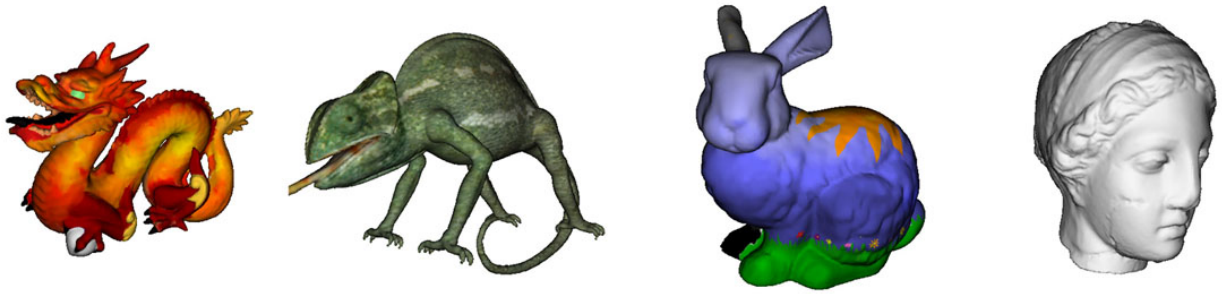

Fig. 5. Four sample renderings.

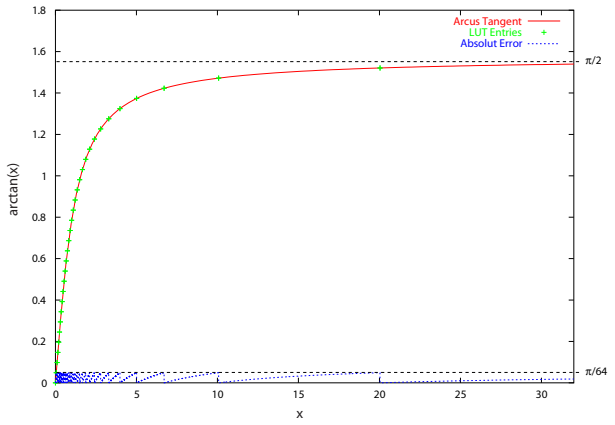

Fig. 6. The arcus tangent approximation, illustrated for 32 LUT entries. The absolute error is always below $\frac{1}{32}\frac{\pi}{2}$.

To find the best function value corresponding to a given x, a binary search algorithm has been adopted. By using the inverse look-up table of size 128, the maximum absolute error of the output is constantly bound by $\frac{1}{128}\frac{\pi}{2}$, An illustration for a inverse look-up table of size 32 is given in Figure 6.

D. Design for Testability

Various test modes have been used. The DMA I/O circuits can be bypassed to feed data directly to the computational units. Furthermore, built-in self test circuits for fast RAM testing performing different write and read patterns are included. The design is also fully scan-chain testable.

IV. Results

The chip has a core size of 1.887 mm $\times$ 4.322 mm in a 180 nm technology and is located in a PGA 144 package. Pad supply is 3.3V, core supply is 1.8V.

The chip operates at a maximum frequency of 147 MHz, with a throughput of 2.941 million splats per second and 300 mW power dissipation. This performance is very close to the peak performance of 3 million splats per second achieved by the implementation of [6] using four digital signal processors, however using significantly fewer resources. The work of [5] achieved a peak performance of 26 million splats per second on a NVIDIA GeForce 6800 GPU, however using a much bigger die size: this work and the rasterizer ASIC from [6] combined only represent 6.58% of the GPU die area, see Table II. A few sample renderings can be found in Figure 5.

Chip	Process	Area	Frequency
NVIDIA GeForce 6800 Ultra (estimate)	130 nm	287 mm^2	400 MHz
Rasterizer ASIC from [6]	250 nm	25 mm^2	196 MHz
This work	180 nm	8.16 mm^2	147 MHz

TABLE II
DIE SIZE COMPARISON.

V. Conclusion

This paper presents the first dedicated VLSI implementation of the transform, lighting and rasterization setup stages for EWA surface splatting. The achieved performance is similar to [6], but it exploits significantly fewer resources. Compared to a GPGPU implementation of EWA surface splatting, the performance is very good considering the resources used. The prototype proves that EWA surface splatting is indeed well suited for hardware architectures. It further gives evidence that an integration into traditional triangle pipelines as presented in [6] would be a valuable addition for GPUs.

References

[1] M. Zwicker, H. Pfister, J. V. Baar, and M. Gross, "EWA splatting," *IEEE Trans. on Vis. and Comp. Graph.*, vol. 8, no. 3, pp. 223–238, 2002.

[2] J. H. Clark, "The geometry engine: A VLSI geometry system for graphics," in *ACM Computer Graphics (Proc. ACM SIGGRAPH 1982)*. ACM, 1982, pp. 127–133.

[3] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The triangle processor and normal vector shader: a VLSI system for high performance graphics," in *ACM Computer Graphics (Proc. ACM SIGGRAPH 1988)*. ACM, 1988, pp. 21–30.

[4] S. Woop, J. Schmittler, and P. Slusallek, "RPU: a programmable ray processing unit for realtime ray tracing," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 434–444, 2005.

[5] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's GPUs," in *Proc. of EG Symp. on Point-Based Graphics*. Eurographics, 2005, pp. 17–24.

[6] T. Weyrich, S. Heinzle, T. Aila, D. Fasnacht, S. Oetiker, M. Botsch, C. Flaig, S. Mall, K. Rohrer, N. Felber, H. Kaeslin, and M. Gross, "A hardware architecture for surface splatting," *ACM Transactions on Graphics (Proc. ACM SIGGRAPH 2007)*, vol. 26, no. 3, pp. 90–11, 2007.

[7] T. Akenine-Möller and E. Haines, *Real-Time Rendering*, 2nd ed. A. K. Peters Ltd., 2002.