Image Synthesis with Image-space Finite Elements
OTKA Project Report, PD — 104710

Principal investigator: László Szécsi PhD

September 28, 2016

1 Research objectives

The objective of the research was to investigate image synthesis algorithms that construct an image using atomic components different from the pixels of classic optical simulation. This approach does not only include illustrative, non-photorealistic, or artistic rendering techniques, but also synthesis of artificial visual stimuli and particle-based modeling and simulation approaches.

2 Overall progress

On one part, research carried out during the project went along the path outlined in the original proposal, and for the other part, it extended to include applications in experimental biology. In terms of the stated practical goals, the work was fruitful — advances in several areas have been made, international cooperation has been established, several students and a PhD student have been attracted to the research — but management of time and dissemination was not sufficiently stringent to entirely deliver the high level of publications that we anticipated. In particular, the GPU-based retina stimulation project, carried out in cooperation with international partners, absorbed inordinate amount of time and effort, but suffered delays beyond our immediate control, with a major journal publication (Nature Scientific Reports) due, but not submitted for publication yet.

3 Results

3.1 Component-based visualization engine

A framework for the implementation and testing of new visualization algorithms was developed in the early stage of the project. This component-based engine exploits new language capabilites of C++ to realize a strongly property-based *object model*, instead of a monolythic hierarchical object-oriented scheme. Convoluted GPU-programming tasks can be scripted with relative ease. The framework was the basis of the development of most results detailed in this document, and will remain a useful tool for any future research within the group.

As part of this effort, our paper[4] presented an algorithm for ordering state change operations—including shader context changes and input/output bindings—necessary to render a frame in an interactive application. We expanded on the context of the *render queue*, but instead of sorting renderable primitives only by material, we propose a flexible framework organizing draw calls into a tree, where items that share any of the conceivable render states are grouped together in a cost-optimal way, minimizing the number of CPU-

GPU communication instances required execute them.

Further publications may emerge as students combine these ideas with upcoming graphics technologies (projects on DX12 and Vulcan are ongoing), but it will not be in focus for the principal researcher.

4 Time-of-flight

An early branch of investigation carried out under the umbrella of this grant was using time-of-flight information in PET reconstruction[31]. This allows better reconstruction of activity from measurement data (see table 1 for an example), albeit at an increased computation cost.

4.1 Non-photorealistic rendering

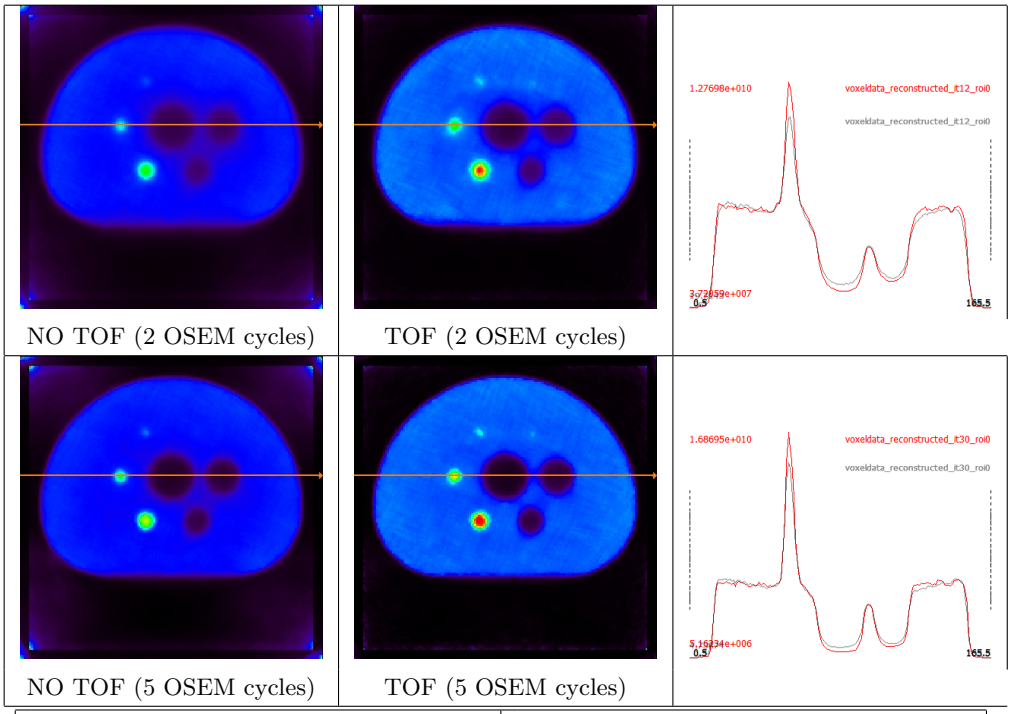
Non-photorealistic rendering (NPR) includes a vast range of techniques from CAD through educational illustrations[8;6] to artistic paintings[19] and animation[7;35]. One common aspect is that they do not aim to produce a picture by optical modeling of human vision, accurately reproducing the visual stimuli reaching the eye, but also involve the other main organ of human vision — the brain — into the simulation. The way to achieve this is to impose new rules and limitations, not allowing pixel colors to be found independently. The rules can be seen to define image space finite elements. The task is to place these elements in the image so that it allows the human viewer to perceive the virtual model and enhance specific features.

As plenty of high quality off-line solutions for NPR have been published before, including a our paper on hatching for motion picture production[35], in the OTKA project we focused on real time algorithms. Existing ones were held against a much poorer quality standard than production solutions, and our motivation was to bridge this gap.

4.1.1 Self-similar hatching algorithms

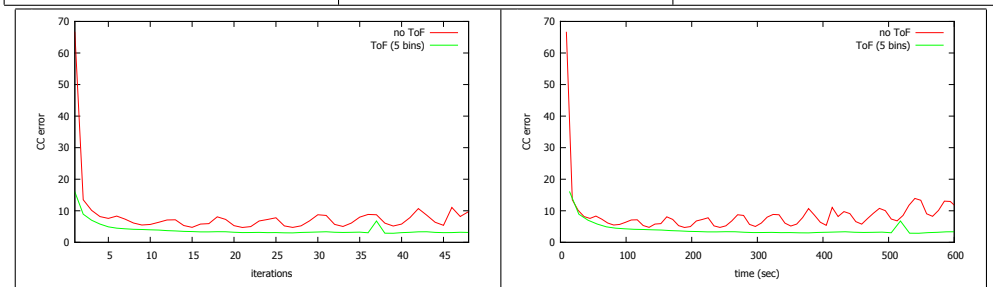
Hatching is one of the basic artistic techniques that is often emulated in stylistic animation. Hatching strokes should appear hand-drawn, with roughly similar image-space width, dictated by pencil or brush size, but they should also stick to surfaces to provide proper object space shape and motion cues. Both properties must be maintained in an animation, without introducing temporal artifacts. In particular, when surface distance or viewing angle is changing, object-space density of strokes should adapt without the strokes flickering or drifting on the surface, while presenting natural randomness inherent in manual work[12;1].

In our papers[30;26;28] we presented *recursive procedural tonal art maps* (RPTAM), a single-shader rendering technique that fulfils the above criteria with less limitations than previous techniques. In particular, strokes are preserved as constructing elements (as opposed to texture harmonics[5]), infinite zooming is possible (as opposed to a finite LOD set[18]), and single-pass local shading is sufficient (as opposed to global geometry processing and hidden stroke removal[35]). The key idea is that we place strokes in texture space, at pre-generated seed locations exhibiting a self-similar pattern, allowing for smooth transitions between any texture scalings. Figure 1 depicts recursively nested seed sets, and figure 2 illustrates our proposed algorithm for generating them, which is similar to the chaos game method of IFS attractor generation. We also have shown how choosing an initial point with coordinates constructed as a recurring quaternary de Bruijn sequence can ensure evenly distributed seed sets, in the sense that the elemental interval property is ensured. Actually, the relation between the newly invented method and de Bruijn sequences was not discovered at the time of our initial publication[26], but was published later in[28]. This meant that polinomial



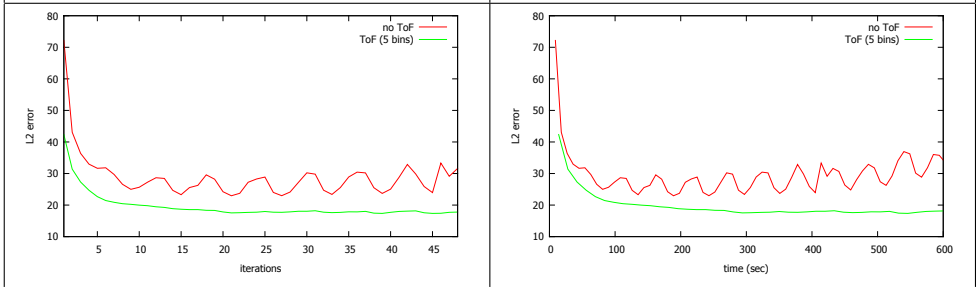
NO TOF (2 OSEM cycles) | TOF (2 OSEM cycles)

NO TOF (5 OSEM cycles) | TOF (5 OSEM cycles)



CC error versus iteration count.

CC error versus reconstruction time.



L2 error versus iteration count.

L2 error versus reconstruction time.

Table 1: Geometry only reconstruction with and without time-of-flight (TOF).

time algorithms known for de Bruijn sequence generation are applicable in our seed set generation process.

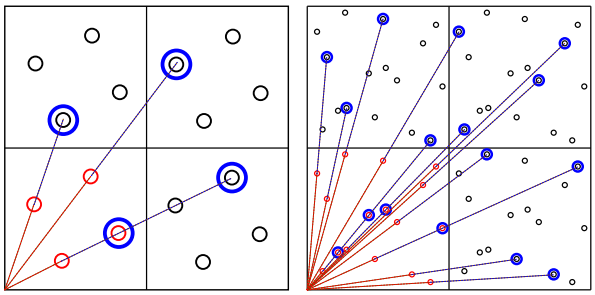


Figure 1: Seed sets of 4 and 16 elements with the recursive nesting property. Large circles indicate seeds, small circles are the seed pattern repeated on a 2×2 grid. The dense pattern scaled up from any corner gives the sparse pattern.

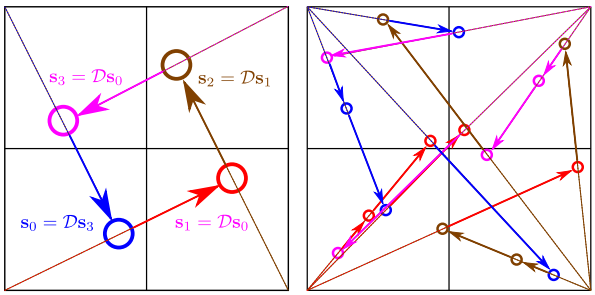


Figure 2: Operator $\mathcal{D}$ projects seeds to double their distance from the nearest corner. Seed sequences are generated by repeating this operation. Seeds are colored to show the corner used for the scaling.

Our method, like dynamic solid textures[5] and Tonal Art Maps[18], provides impeccable temporal coherence. Dynamic solid textures can produce *binary style* rendering to approximate hatching, but our method can work with stylized hatching strokes. Tonal Art Maps are equivalent to our approach in quality, but they do not offer infinite zooming. TAMs can be edited manually, or generated automatically by randomly inserting new strokes, and rejecting or clipping those colliding with existing ones. The TAM generation method is a lengthy trial-and-error search, while polynomial time methods exist for the generation of de Bruijn sequences. Both methods target the same quality criteria of uniformly distributed strokes, but our method guarantees uniformity in a well-defined geometric sense, not only as an stopping criterion for a random process. More importantly, as opposed to TAMs, seed sets do not need to be re-generated if artistic parameters like stroke length or stroke texture change, even allowing these to be animated. We can fade strokes simultaneously (producing rendering similar to TAMs), but also individually, which is a unique feature among texture-based hatching methods.

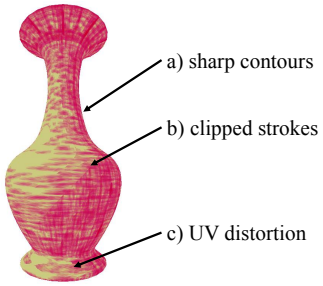


Figure 3: TAM and RPTAM suffer from stylistic inconsistencies including: a) strokes clipped at object silhouettes, b) strokes clipped or faded for density control, and c) strokes distorted by anisotropic UV mapping.

Figure 3 shows stylistic artifacts that arise with both TAM and RPTAM as a result of their texture-space approach. In both methods, density control is performed on the pixel level. This means that the decision weather a stroke should appear can be different for different parts of a stroke. In we take a binary decision, some strokes are clipped before entering an area that should be less densely hatched. If

the decision is smooth, e.g. implemented by blending between textures with different line densities, then strokes will fade out. Neither case is consistent with the requirement that the image is constructed using strokes matching an artist's pencil or brush, in consistent style. Similarly, as strokes are applied to object surfaces in texture space, they are clipped at object silhouettes (or UV-parametrization discontinuities) in image space. This is not possible in hand-drawn art, and makes the renders appear artificial.

One concept for adding overdrawn strokes was to render an inflated geometry and reconstruct clipped stroke parts with a localized filtering operation. This approach turned out not to deliver robust results, despite considerable researcher and student effort spent on exploring it.

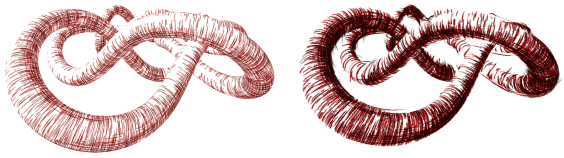


Figure 4: Texture-space methods like TAM (left) must fade or clip strokes at object boundaries and for density control. TAMISS (right) restores stylistic coherence by fitting new strokes in image space.

In our papers[21;27;29] we proposed *Tonal Art Maps with Image Space Strokes* (TAMISS), a hybrid technique that combines the robust visibility testing and density control of TAM or RPTAM with the stylistic freedom of image space stroke extrusion (figure 4). The idea is to assign unique IDs to all TAM strokes, perform rasterization of surfaces with TAM, producing fragments marked with stroke IDs, and fit a curve on each set of fragments sharing the same ID. The curves can be extruded to image space strokes in proper style, while visibility and density control has already been taken care of by TAM.

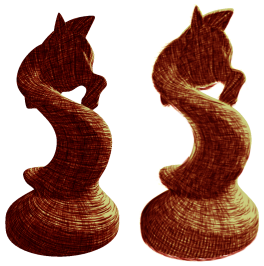


Figure 5: Knight model rendered with RPTAM (left) and TAMISS (right).

We discribed a fitting process using Ordinary Least Squares[10], solving the arising system of equations with the Conjugate Gradient Method[17]. We provided a GPU implementation capable of solving the task in real time, regardless of the scene complexity.

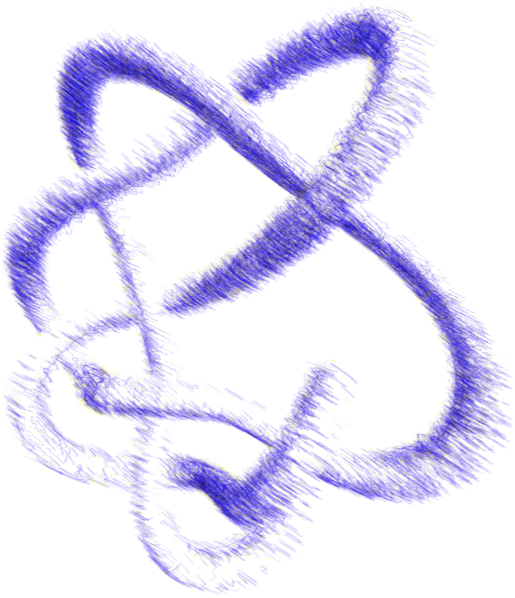


Figure 6: Torus knot model rendered in a pen-like style. All lines are uniquely randomized.

The TAMISS solution[21] won the Best Poster Award at the Eurographics conference, and the extended version[29] has been accepted to STAG

2016. Whether it will be invited as a Q1 journal publication in Computers and Graphics is pending.

A remaining limitation of both our RPTAM and TAMISS methods is that, like any texturing, they require a two-dimensional parametrization of the 3D surface, i.e. texture coordinates. The properties of this mapping may strongly influence the quality of the result. In order to sidestep this problem, we pursued the idea of extending the de Bruijn sequence-based method of seed point generation to three dimensions. Even though this resulted in an elegant piece of theory, the point set projected to the 2D surface turned out not to meet the original requirements of consistency, meaning this avenue of thought had to be abandoned, and the necessity of 2D parametrization accepted. We found that for RPTAM or TAMISS, the parametrization needs to be close to conformal, and without discontinutities. There exist high quality methods that achieve such a parametrization, but we believe a specialized solution exploiting the self-similarity characteristics is possible. Ongoing research, in cooperation with faculty collegaues involved in computer geometry is expected to address this open question.

4.1.2 Outlines

Outlines are used in both artistic and technical depictions to emphasize discontinuities. Drawn outlines include *silhouettes* at image-space boundaries of object surfaces (separating front- and back-facing parts in object space), and *creases* at discontinuities of the surface normal. Outlines often must be rendered as wide, textured, semi-transparent strips without seems, folds or flickering.

There are two well known approaches to outline rendering. The first one works in image space with the use of normal and depth maps[16]. Edge pixels — those which lie near discontinuities in these maps — can be found using edge detection filters. What level of image-space discontinuity warrants outline edges must be adjusted by fine-tuning filter parameters and applying mask textures[20]. Object-space consistency of outlines during animations is also subject to those parameters. However, the main problem with this approach is the excessive texture access bandwidth and the absence of real scalability in line features. The other approach works in world space and generates new triangle strip geometry to visualize the outlines. In this case we do not need to search on per pixels basis.

We investigated the visibility problem of crease outlines in stylistic and engineering rendering. We offered two different, consistent formulations that lead to artistic and technical drawing styles. We developed real-time, flicker-free GPU algorithms for both problems.

Compositing object-space outlines with surfaces in a 3D scene is a major challenge. It can be seen as a hidden line removal problem that can be solved geometrically in **object space**[2], which is expensive, even if accelerated by an image-space lookup[15], or using image space **depth testing**[11], where filtering must be used to alleviate instabilities. One technique we proposed is based on the latter approach. The outline, rendered as a wide strip, will be considered visible in all its width where its centerline is not hidden. This is similar to an artist painting strokes on paper, lifting the brush roughly where outlines would go behind objects. We will refer to this approach as the *Wide Outlines with Approximate Rendering Technique*, or *WO/ART*.

In technical drawings, enhancing feature edges must not modify shape contours or interfere with exact occlusion. Drawing the polygon wireframe[3] only where the object itself is visible—thus, practically, onto its surface—is such a technique. We extend this from polygons to non-planar, potentially self-occluding

smoothing groups of arbitrary topology to get on-surface crease edge rendering. We will refer to this approach as *Wide Outlines with Precise Rendering of Occlusions*, or *WO/PRO*.

We proposed a solution[25] that can render outlines with a performance similar to regular incremental triangle mesh rendering, but meeting quality standards set by offline NPR methods. The particular contribution of this work is a real-time outline rendering algorithm that does not use image processing filters, renders antialiased, continuous, textured, alpha-blended outlines of any width without seems or folds, and provides flicker-free animation.

The goal is to provide a solution which can replace costly edge detection filters with a more flexible, geometry-aware method in real-time applications, and offer a faster alternative in stylistic rendering.

Triangle adjacency computation could be necessary either for identifying edges between front- and backfaces, or to link segments of silhouettes crossing from one triangle to another. This can be performed once at mesh loading time. The resulting *triangle adjacency buffer* contains a triplet of integer triangle indices for all mesh triangles. For manifold meshes, all entries are valid, if triangles adjoint at creases are considered adjacent.

Crease halfedge identification can be performed during triangle adjacency computation. Whenever two spatially aligned halfedges are found, if their vertices are not identical, we have found two crease halfedges. Halfedges form one or more closed loops, and can be organized as such. There could be various representations for the result, but it is usually required that the halfedge loops can be traversed and the vertices along crease outlines enumerated. In our solution, we need to know for every triangle, which crease halfedges are adjacent. If we are not interested in triangle adjacencies over creases, this information can be stored in the triangle adjacency buffer, storing the crease halfedge ID instead of the neighboring triangle ID. The ID ranges must be separate, easily assured if crease halfedge IDs start from the number of triangles in the mesh.

Silhouette segment identification is view dependent, and must be performed in every frame.

We argue against Markosian-style silhouette segments, aligned on triangle mesh edges. First, they are prone to backtracking. Second, finding adjacent segments would require a varying-number-of-steps search over edges sharing the end vertices of the silhouette edge. A data structure listing adjacent edges for all vertices would also be required. While this would provide a clear-cut situation where all outline segments are edges of the triangle mesh, which could be beneficial for hidden segment removal and provide a better alignment of outlines with triangle mesh renderings in low-polygon-count cases, but burdens the outlines with artistically undesirable triangulation artifacts.

Segments of Herzmann and Zorin style silhouettes are easily linked together, and also to crease halfedges, using the triangle adjacency buffer. All segments are assigned an ID identical to the ID of their triangle. The adjacency buffer entries for the two edges the silhouette crosses readily contain the IDs of the adjacent outline halfedges, be they silhouettes or creases.

Clipping crease halfedges to silhouettes is necessary when using Herzmann and Zorin silhouettes because a crease edge might be partially back-facing by that definition. Clipping the halfedges and linking them to the silhouette edges produces closed halfedge loops around visible on-screen areas.

Linking edge segments into continuous halfedge loops is necessary if we wish to render outline strokes as wide triangle strips. An ordered, traversable list of halfedges also allows for smoothing and parametrization.

Hidden halfedge removal must be performed to avoid drawing outlines behind closer surfaces.

Outline parametrization can be performed along the halfedge loops. We have to take care to ensure animation coherence. Thus, parameters are stored for all crease halfedges and triangles, and only slightly adjusted from frame to frame.

Outline stylization includes smoothing, stroke breakdown and vectorization, converting the halfedge loop into textured triangle strips acceptable as artistic strokes. Switchbacks and small glitches in the outline should be removed, and loops should be separated where they turn at sharp angles to produce separate strokes.

WO/ART can be implemented as a two-pass method. The first pass renders solid surface geometry, also producing a depth buffer with a small depth slope bias. The second pass renders crease halfedges—augmented with adjacent vertices along the crease halfedge loop—as four-control-point patches. The *constant hull shader* computes and outputs crease normals at vertices, and samples visibility using *comparison filtering* along the three-segment line strip, producing an output *visibility bitmask*. The number of samples depends on the screen-space size of the halfedge, but we limited it to 64 to fit the bitmask into two integers. The tessellator is set up to convert the patch to a quad strip with a linear tessellation factor along the v axis roughly corresponding to the sample density. The *domain shader* positions the strip vertices offsetting them along the interpolated crease normals, at a distance proportional to their visibility. Vertex visibility is computed by averaging relevant bits of the visibility mask. Completely hidden parts will have zero width, the strip will taper off where visibility vanes, and numerical inconsistencies are hidden by averaging multiple visibility samples. The pixel shader applies texturing with alpha-blending.

These results have only been published in a preliminary poster form at the Eurographics conference[25]. The WO/ART solution will be submitted as a journal paper to strongly evolving outlet Periodica Polytechnica on behest of its general editor as part of an effort to fill the journal with quality articles and help elevate it to gain an impact factor.

4.1.3 Image space hatching

In our papers[34;14] we presented a screen space hatching algorithm that provides time coherent placing of hatching lines relative to object surfaces. While with screen space techniques we can easily achieve consistent image space hatching density, it is hard to make hatching lines express surface features, and to make them follow the underlying geometry. Drawing individual textured lines can provide high quality results, but their direction and amount of bending should be calculated according to the 3D geometry. We proposed a method that combines the illumination gradient with curvature based line direction calculation to support a wide variety of objects. To achieve surface position coherency during animation we use image space velocity maps to move the individual hatch lines. We use rejection sampling and low discrepancy sequences to filter out high density areas where the flow accumulates lines, and to fill in the vacant areas.

4.1.4 Hatching for natural phenomena

In our papers[32;33] we presented a highly parallel algorithm for the stylized, real-time display of fluids and smoke. We use metaballs to define a fluid surface from a particle-based fluid representation, but instead of the costly complete reconstruction of this surface, we only trace the motion of random seed points on it. Hatching strokes are extruded along the lines of curva-

ture. We proposed methods for hidden stroke removal and density control that maintain animation consistency.

4.2 Real-time ray casting of self-similar procedural geometries

The basic question asked in this line of research is: what kind of geometry can be ray traced with an iterative, branching-free algorithm? This is a critical property for efficient GPU parallelization. We aim to provide a geometric model with this motivation, and explore what kind of application such a limited model may have.

Complex geometries, like those of plants, rocks, terrain, or even clouds are challenging to model in a way that allows for real-time rendering but does not make concessions in terms of visible detail. In our papers [22;24] we proposed a procedural modeling approach, called KRS, or *kernel–reflection sequences*, inspired by iterated function systems. The model is composed of kernel geometries defined by signed distance functions, and reflection transformations that multiply them. We showed that a global distance function can be evaluated over this structure without recursion, allowing for the implementation of real-time sphere tracing on parallel hardware. We also showed how the algorithm readily delivers continuous level-of-detail and minification filtering. This, combined with the smooth shading and texturing techniques we also describe, eliminates aliasing and achieves automatic, continuous level-of-detail.

We proposed several techniques to enhance modeling freedom and avoid conspicuous symmetries. *Conformal* transformations, which include geometric inversion, map spheres and planes to spheres and planes. If symmetric geometry is subjected to such a transformation, mirror planes are mapped to spheres, eliminating symmetry. An unbounding sphere of the symmetric geometry is mapped to an unbounding sphere of the transformed geometry. This makes it possible to find unbounding spheres for a conformally transformed KRS, even if different KRS levels are subject to different transformations. For that reason, sphere tracing geometries defined by signed distance functions under conformal transformations are of interest.

Given a world space *probing* sphere, we can verify if it is an unbounding sphere simply by transforming it to kernel space, by always taking the mirror test decision on the sphere center (which is known now), and checking the radius against the distance. Note that this relaxes the requirement on kernel geometries that they have to be defined by distance functions, as it is enough to merely support a binary intersection test with a sphere.

The sphere tracing process becomes a trial-and-error search finding unbounding spheres, not entirely unlike numerical root finding methods employing *binary search*. Therefore, we call this algorithm *binary sphere tracing*. It can be seen as an extreme version of sphere tracing with *over-relaxation* [13], with no option to revert to classical sphere tracing near surfaces, and without the possibility to accept an unbounding sphere that is larger than what we speculated on. These restrictions are necessary for admitting conformally transformed geometry.

Following up on the work of Bisi and Gentili on the quaternion algrebra of Möbius tranformations, we derived a set of formulas for transforming spheres, perfomring ray–sphere intersections in quaternions, and transforming surface normals.

We also proposed a GPU load balancing scheme for best utilization of computing power. To prove that the model can be used to realize various natural phenomena in uncompromising detail and extents, without obvious clues of symmetry, we implemented real-time ren-

dering of aquatic and terrestrial surface formations and vegetation.

The paper on this research was accepted for STAG 2016, with an invitation for publication in the Q1 journal Computers and Graphics possible.

4.3 Retina stimulation

This project[23;9] emerged as an international cooperation with Peter Hantz at the Friedrich Miescher Institute in Swizerland. Aimed at retina research crucial for understanding retina pathologies and restoring vision artificially, the results were (and are) destined to appear in a high impact factor biology or neuroscience journal. However, several factors delayed this. First, the effort kept extending to provide a research tool as generic as possible, surveying the entirety of visual stimulation techniques used over several fields of visual science, including psychophysics and optogenetics. Second, our research partner's professional and personal relations at the FMI have deteriorated over time, resulting in him and the institute parting ways. Dr. Gnther Zeck from the NMI (Institute for Sciences and Medicine) of the University of Tübingen joined the international cooperation, providing the necessary laboratory background, but at the moment, crucial measurement results remain missing that prove our solution's superior applicability in the field.

Both of the above mentioned institutes conduct experiments with rodent retina. Visual stimulus patterns have to be projected onto the specimen obtained from animals bred for the purpose. The stimulus patterns range from the very simple to the computationally demanding. Temporal synchronzation requirements are difficult to achieve with the existing equipment, and the solutions must support the purposes of biology researchers lacking any programming or graphics background.

Light stimulation with precise and complex spatial and temporal modulation is required in several research fields like visual neuroscience, optogenetics, ophthalmology, and visual psychophysics. We devised an intuitive and flexible stimulus generating framework (GEARS — GPU-based Eye And Retina Stimulation), which offers access to GPU computing power, and allows interactive modification of stimulus parameters during experiments. Furthermore, it has support for driving external equipment, as well as for synchronization tasks, via USB ports.

GEARS supports real-time operations like tone mapping, histogram equalization or contrast stretching, filtering operations like edge enhancement by double-Gaussian kernels, image sharpening, as well as further operations in real or Fourier space. If a large batch of random numbers is required in every frame, GPU-based parallel random number generation is also possible.

We proposed a new computational workflow model that is more inclusive than any of previous solutions, some of which are restricted to polygon rendering with precomputed textures.

In most light stimulus software developed up to this time, simple usage and flexibility were conflicting demands. A graphical user interface (GUI) has inherent limitations and requires permanent development. In contrast, if customizing the software is implemented through an application program interface (API), the user has to possess deep programing skills. In order to avoid drawbacks of APIs and GUIs, a visually aided scripting interface (VSI) has been developed. This is an integrated, Python-based script editor to write, modify and combine intuitive stimulus components, with custom code completion and call tips, providing instant documentation, as well as component and parameter listing. The VSI offers instant visual presentation of the editable features of the stimulus elements. Moreover, it shows the time flow of the stimulus sequence prepared for

execution.

In GEARS, all stimuli are constructed as a combination of previously defined components. Our approach is inspired by modern game engine systems, where aspects of game world entities are represented by components that can be freely combined. Most of the earlier solutions utilized a low number of shaders, while stimuli were implemented by setting appropriate shader parameters. This structure cannot guarantee the flexibility and efficiency that shaders tailored for specific tasks would provide. Moreover, the continuous demand for new stimuli will inevitably exceed the limitations of previously written shaders, while new ones cannot be implemented without resorting to GPU programing.

In GEARS, creating custom-made shaders for specific tasks does not need GPU programing expertise, because of the core software mechanism that dynamically generates graphical (GLSL) shaders from SBC combinations. This modular structure is the key to the flexibility and simplicity, and at the same time the high computational power of GEARS. A large library of parametrizable components is provided, which covers the needs for displaying practically all of the stimuli we identified in the literature.

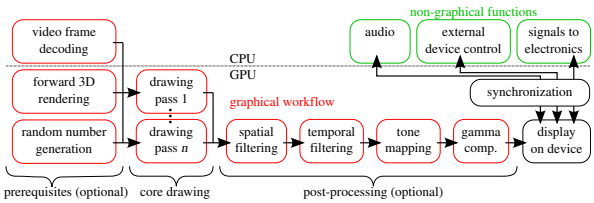


Figure 7: Generation procedure of a single stimulus frame. *Prerequisites*, *core drawing* and *post processing* phases are each composed of passes marked by red boxes.

Figure 7 shows the proposed workflow for rendering a single frame of a stimulus. The nucleus of the workflow is called *core drawing*, which is a set of passes designed to produce an intermediate version of a stimulus image, which may be further processed before being displayed on the screen. The core drawing may be preceded by operations like video decoding, random number generation, and/or 3D OpenGL rendering, and can be followed by post-processing steps like spatial and temporal filtering, or gamma compensation.

GEARS offers a feature which is unique within the set of similar software, namely the possibility for real-time spatial and temporal filtering of the rendered frames. Filtering with large kernels is computed more efficiently in the frequency domain, whilst for filterings with small ones, the spatial domain method demands less resources. This capability is important in opening the possibility of a system identification approach to acquire models of retina cell behavior. For filtering in the Fourier space, the kernel can directly be given in the frequency domain, offering a convenient way to specify low-pass, band-pass, high-pass, or anisotropic filterings.

During temporal filtering the pixel values of the stimulus frames are regarded as *discrete time signals*. Filtering can be performed either by convolution with a temporal one-dimensional filter kernel, or, if the kernel is smooth enough to be well approximated by a composition of complex exponentials, a linear time invariant (LTI) signal processing operation is also suitable. The LTI state representation can be directly given, which is essential if a theoretical system realization is being investigated, where the differential equations governing the system are known.

Realizations of temporal filtering funtionality include *rectangular*, *triangular*, *Hamming*, and *Hann* windows, *exponential* attenuation, and a mixture-of-Gaussians model for the approximation of retina *cell* response.

Another unique capability is that of real-time dynamic tone mapping operations with

linear (contrast stretching) and sigmoidal transfer functions, as well as histogram equalization of picture sequences or videos.

Our computational model has been shown to be able to reproduce results from literature, and measurements for new experimental schemes made possible by the extended computational pipeline were also carried out. These include measuring the response to random white noise stimuli with various temporal and spatial kernels modeling behavior or specific retinal cells, natural or artificially manipulated. Similar measurements with simple shapes, moving or stationary, also occured. Measurmetns of natural videos with various real-time processing options are under evaluation currently. These results validate our proposed framework as a useful tool in retina research. However, until these measurements were all processed and the biological implications laid out by our partners, publication was not quite possible. The paper is not yet submitted to Nature Scientific Reports, but partners believe that is going to happen in the upcoming weeks. Thus, the research would conclude with a high IF journal publication.

References

[1] Zainab AlMeraj, Brian Wyvill, Tobias Isenberg, Amy A Gooch, and Richard Guy. Automatically mimicking unique hand-drawn pencil lines. *Computers & Graphics*, 33(4):496–508, 2009.

[2] A. Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference*, pages 387–393. ACM, 1967.

[3] J.A. Bærentzen, S.L. Nielsen, M. Gjøl, and B.D. Larsen. Two methods for antialiased wireframe drawing with hidden line removal. In *Proceedings of the 24th Spring Conference on Computer Graphics*, pages 171–177. ACM, 2008.

[4] D. Bányász and L. Szécsi. Optimizing state changes in rendering engines. In L. Szirmay-Kalos, editor, *Hungarian Computer Graphics and Geometry Conference (GRAFGEO)*, pages 116–123, 2014.

[5] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. Dynamic solid textures for real-time coherent stylization. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 121–127. ACM, 2009.

[6] S. Bruckner, S. Grimm, A. Kanitsar, and M.E. Gröller. Illustrative context-preserving volume rendering. In *Proceedings of EUROVIS*, volume 2005, pages 69–76, 2005.

[7] J. Collomosse and J.E. Kyprianidis. Artistic stylization of images and video. *Tutorial at Eurographics*, 2011.

[8] B. Csébfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. In *Computer Graphics Forum*, volume 20, pages 452–460, 2001.

[9] P. Hantz, Á'. Kacsó, G. Zeck, and L. Szécsi. Interactive light stimulus generation with high performance real-time image processing and simple scripting. *Frontiers in Neuroscience*, (41), 2016.

[10] F Hayashi. *Econometrics*. Princeton University Press, 2000.

[11] T. Isenberg, N. Halper, and T. Strothotte. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. In *Computer Graphics Forum*, volume 21, pages 249–258. Wiley Online Library, 2002.

[12] Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Piché, and Victor Ostromoukhov. Hatching by example: a statistical approach. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 29–36. ACM, 2002.

[13] Benjamin Keinert, Henry Schfer, Johann Korndrfer, Urs Ganse, and Marc Stamminger. Enhanced Sphere Tracing. In Andrea Giachetti, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014.

[14] Z. Lengyel, T. Umenhoffer, and L. Szécsi. Realtime, coherent screen space hatching. In L. Szirmay-Kalos, editor, *Hungarian Computer Graphics and Geometry Conference (GRAFGEO)*, pages 131–137, 2014.

[15] L. Markosian and J.F. Adviser-Hughes. *Art-based modeling and rendering*. Brown University, 2000.

[16] M. Nienhaus and J. Doellner. Edge-enhancement-an algorithm for real-time non-photorealistic rendering. *Journal of WSCG*, 11(2), 2003.

[17] Jorge Nocedal and Stephen J Wright. *Conjugate gradient methods*. Springer, 2006.

[18] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 581–581. ACM, 2001.

[19] R. Sayeed and T. Howard. State of the art non-photorealistic rendering (NPR) techniques. *Eurographics UK: Theory And Practice Of Computer Graphics*, pages 89–98, 2006.

[20] J. Shin. A stylised cartoon renderer for toon shading of 3d character models. Master's thesis, University of Canterbury, UK, 2006.

[21] Lszl Szcsi, Marcell Szirnyi, and gota Kacs. Tonal Art Maps with Image Space Strokes. In Luis Gonzaga Magalhaes and Rafal Mantiuk, editors, *EG 2016 - Posters*. The Eurographics Association, 2016.

[22] L. Szécsi. A geometry model for logarithmic-time rendering. In L. Szirmay-Kalos, editor, *Hungarian Computer Graphics and Geometry Conference (GRAFGEO)*, pages 21–28, 2014.

[23] L. Szécsi. Gpu pattern generation for retina stimulation experiments. In *Képfeldolgozók és Alakfelismerők Társaságának 10. országos konferenciája*, pages 388–393, 2015.

[24] L. Szécsi, Z. Bendefy, and Á'. Kacsó. Kernel-reflection sequences. In *Smart Tools and Apps in computer Graphics 2016*, pages 53–62. EUROGRAPHICS, 2016.

[25] L. Szécsi, B. Hajagos, and T. Umenhoffer. On depth-testing wide outlines. In Miguel Chover and A. Augusto de Sousa, editors, *Girona*, pages 15–16, 2013.

[26] L. Szécsi and M. Szirányi. chapter Recursive Porcedural Tonal Art Maps, pages 57–66. Vaclav Skala - Union Agency, 2014.

[27] L. Szécsi and M. Szirányi. chapter Tonal Art Maps with Image Space Strokes, pages 155–159. BME Iranyitastechnika es Informatika Tanszék, 2015.

[28] L. Szécsi and M. Szirányi. Dinamikusan generált textúra alapú vonalkázás. In *Képfeldolgozók és Alakfelismerők Társaságának 10. országos konferenciája*, pages 687–702, 2015.

[29] L. Szécsi, M. Szirányi, and Á'. Kacsó. Tonal art maps with image space strokes. In *Smart Tools and Apps in computer Graphics 2016*, pages 39–44. EUROGRAPHICS, 2016.

[30] L. Szécsi, M. Szirányi, and T. Umenhoffer. Improving texture-based npr. In L. Szirmay-Kalos, editor, *Hungarian Computer Graphics and Geometry Conference (GRAFGEO)*, pages 138–148, 2014.

[31] L. Szécsi, L. Szirmay-Kalos, Gy Egri, and G. Patay. Binned time-of-flight positron emission tomography. In *KEPAF 2013*, pages 340–350, 2013.

[32] L. Szécsi and F. Tükör. Hatching animated implicit surfaces. In L. Szirmay-Kalos, editor, *Hungarian Computer Graphics and Geometry Conference (GRAFGEO)*, pages 124–130, 2014.

[33] F. Tükör and L. Szécsi. chapter Hatching for Metaball Surfaces., pages 89–98. 2014. TU Vienna.

[34] T. Umenhoffer, Z. Lengyel, and L. Szécsi. Screen space features for real-time hatching synthesis. In Czúni Lászlo, editor, *Képfeldolgozók es Alakfelismerők 9. országos konferenciája*, pages 82–94, 2013.

[35] T. Umenhoffer, L. Szécsi, and L. Szirmay-Kalos. Hatching for motion picture production. In *Computer Graphics Forum*, volume 30, pages 533–542, 2011.