CEIG - Spanish Computer Graphics Conference (2015)
Jorge Lopez-Moreno and Mateu Sbert (Editors)

PREFR: a flexible particle rendering framework

S. E. Galindo[†1], P. Toharia[1], J. Lopez-Moreno[1], O. D. Robles[1], L. Pastor[1]

[1]Universidad Rey Juan Carlos, Spain

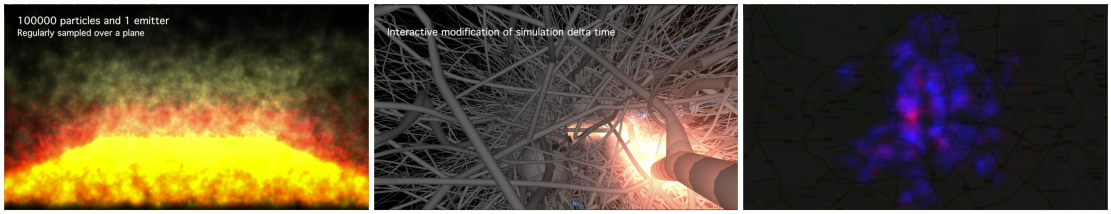


Figure 1: *Screen captures of the accompanying PREFR interactive demos (See supplementary videos). From left to right: Fire simulation (100k particles), neuronal activity simulation and GIS data visualization of Madrid city, reflecting a particular citizen and enterprises spatial interaction (See Section 4).*

Abstract

We present PREFR (Particle REndering FRamework): a first approach to a general-purpose particle rendering framework on the standard OpenGL architecture, designed with the goal of being easily configured by the user without compromising efficiency. In this paper, we analyze and discuss the performance of each stage involved in particle rendering in order to improve its efficiency for future versions with additional GPGPU computation steps or multicore parallelization techniques.

Finally, we show the potential of our particle engine by tackling two very different problems: The rendering of neuronal electrical impulses in physiological models of the human brain, and the visualization of emergent patterns for information analysis, specifically emphasizing structured information in a complex data set.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Particle systems—CG applicationsPerformance

1. Introduction

Particle rendering techniques have been extensively used in the fields of animation, simulation and information visualization. The most common approach is to develop *ad-hoc* solutions to tackle specific problems. This allows the programmer to obtain the best performance for each case. However, *ad-hoc* solutions lack the flexibility needed in more general scenarios, often requiring a great amount of development time. In this paper we present PREFR, a particle-rendering framework designed to be as general as possible while keeping high performance as a goal. This framework provides the programmer with an intuitive high level interface for building particle systems with a minimal effort, while allowing advanced programmers to go on a lower level of development and fine tune their applications in order to achieve the best performance possible. In section 3 we describe the architecture of the framework. In section 3.2 we analyze the performance of PREFR. Finally, in order to show the flexibility of our approach, we present three applications built with our framework which have a very distinctive nature both on the data and the type of particles.

† sergio.galindo@urjc.es

DOI: 10.2312/ceig.20151195

2. Related Work

One year after the first use of structured particle systems in the movie: *Star Trek II: The Wrath of Khan*, Reeves in his seminal paper [Ree83] introduced the concept of particle system: a method for modeling fuzzy objects such as fire, clouds, and water, which models an object as a cloud of primitive time-evolving particles that define its volume. He showed the potential of this method for stochastic modeling of irregular structures such as trees or grass in subsequent work with Blau [RB85].

Many game engines have developed their own custom solutions in order to edit and render large quantities of interactive particles [Roc14]. However, the architecture of a particle system manager has rarely been discussed in the literature [vdB00], with most academic papers focused on improving the rendering efficiency. For instance, Cantlay [Can07] proposed a method to render particles off-screen into a downsampled render target to avoid the overdraw and fill rate issues related to the rendering of large particles with alpha transparency. His method produced compelling results for low frequency effects such as smoke but it is not suitable for detailed rendering. In a similar line of work, Persson [Per12] proposed to trim particles with fitted polygons (instead of quads) to avoid unnecessary alpha tests. Over the last years, recent advances in the GPU pipeline have been successfully incorporated into particle systems, such as compute shaders [Tho14] (with a tile-based subdivision of the frame buffer), or interfaces with CUDA to handle the computation of particle dynamics [Sev13].

Starting with spots and stochastic textures, as proposed by van Wijk [vW91], the visualization of complex data has been a traditional application of particle systems, particularly 3D vector and flow field rendering in engineering applications [ZSH96]. In this field, the GPU has gained increased attention from the research community: from the GPU sorting algorithm used by Kruger et al. to depict 3D flow fields on regular grids [KKKW05] (which was on par with CPU in terms of speed), to the latest CUDA implementations, which outperform their CPU parallel counterparts in OpenMP [Sev13].

In the field of data mining, visual patterns can emerge from the clustering of multiple point samples and the associated shape. To this end, α-shapes were introduced by Edelsbrunner et al. [EKS83] in order to model the convex hull of a set of particles in a plane. Furthermore, Packer et al. [PBN*13] proposed a heuristic model to create hierarchical clusters based on α-shapes. In this work, we show how kernel functions (e.g. gaussian) mapped onto particles produce similar cluster visualizations. In this context, cluster recognition implies multiple perceptual cues: size, shape, variations, etc. requiring a configurable particle representation. For instance, Stenholt [Ste14] proved in a recent study that constant visual angle glyphs (constant screen-size particles) are better suited for the perception of cluster structures in 3D scatterplots.

In this paper we present a highly configurable particle system for data enhancement and visualization, analyzing its performance for further improvement in terms of rendering efficiency and editing capability.

3. PREFR framework

3.1. Framework architecture

In this section we discuss the architecture of the framework, including system requirements, design decisions and implementation details. Due to the previously mentioned lack of discussion in this aspect we propose the following approach.

PREFR has been designed taking into account flexibility and efficiency as the fundamental pillars of the architecture. Ease of use is also another important aspect when developing applications of a particle engine. A highly complex interface may provide additional control for elaborated interactions compared to simple ones, however, it is unclear which parameters or combinations of parameters should be exposed to each user profile (artist, data scientist, etc.) and thus they are out of the scope of this paper.

Thus regarding usability, the PREFR's design will be focused in simplicity on the integration, high degree of configuration and generality of cases. Even without adapted GUI, the actual framework allows to create visual effects and representations with a small amount of effort on the developer or end-user side. In Section 4 we show two very different scenarios based on a python scripting pipeline.

Functionality distribution is a crucial issue to tackle when designing a flexible and efficient system, as one usually opposes to the other, often in a difficult balance when looking for a particular behavior. PREFR has been designed with this in mind, providing an efficient framework, running on native OpenGL, for a wide range of applications.

In general, despite *ad hoc* solutions present a certain grade of modularity, they usually do not need an exposed set of operating modules to be exchanged or modified according to final application needs, ultimately leading to a certain coupling between these components. In contrast, this is one of the key aspects of PREFR, as the developed working base can be complete and simply exchanged with a brand new one crafted by the framework user. By this way, provided components can be easily reimplemented with low effort to meet the functionality of a specific application, adding or modifying new attributes or behaviors as needed.

PREFR is a particle engine framework focused on the actual particles processing, which are the main entity of the system i.e. it is all about the particles. The engine is created with a given budget of particles to be used in the simulation. Then, the desired set of them is distributed along the included components e.g. considering several emitters, each

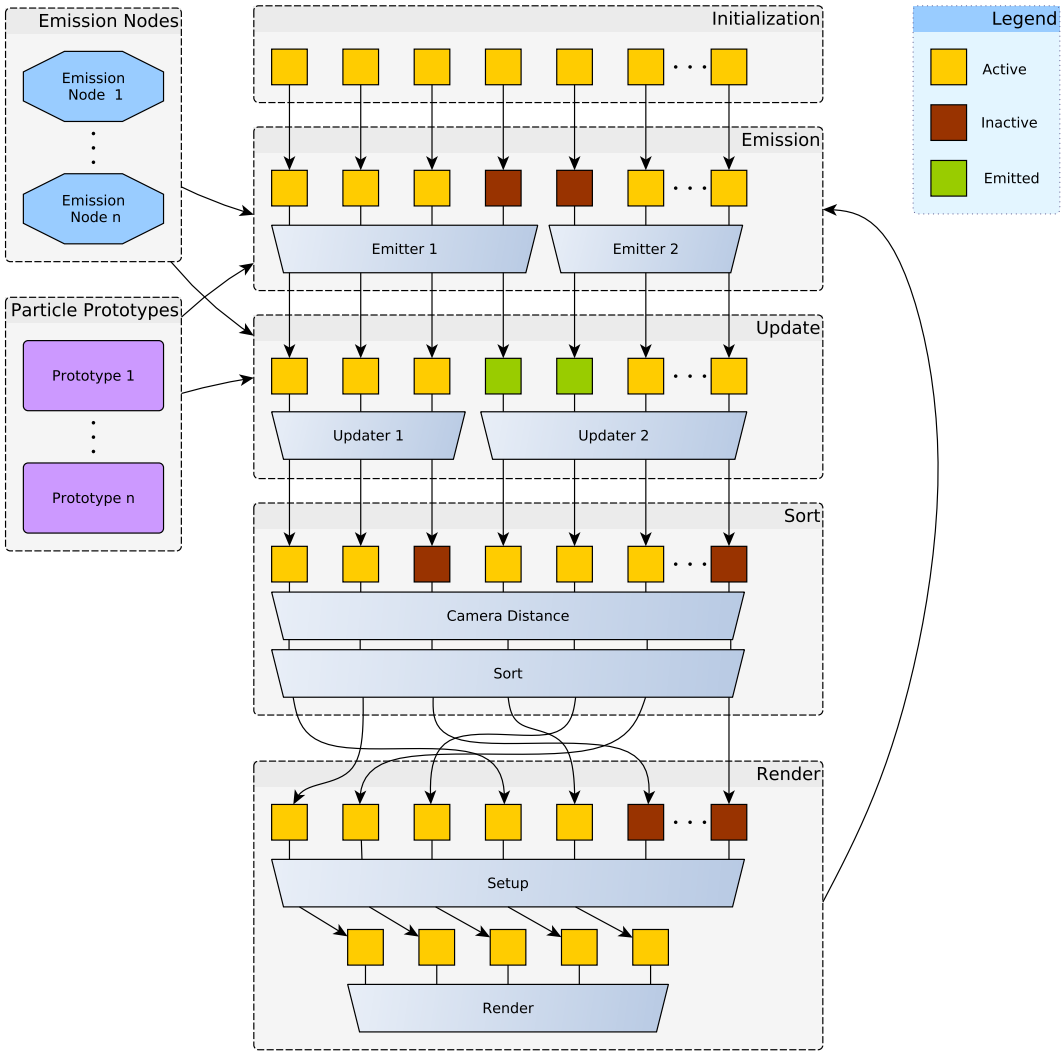


Figure 2: *System architecture and behavior overview.*

one performing a different behavior, any particles distribution can be made over them, even on a running simulation. This approach provides a considerable flexibility on the configuration of the system, as the same set of particles can be assigned into completely different ranges through the system stages' components, as it can be seen on Fig. 2.

It is important to note that the actual system stores dead particles as well, avoiding memory reservations and liberations which might lead to a substantial overhead.

3.1.1. Components

This section describes in more detail how we designed the system and how we come to decide functionality distribution and operating modules.

Regarding the classic pipeline of three-dimensional graphics applications, the render process is usually split into two main steps: Update and Render. The former consists on updating objects state for the elapsed time from the last frame or simulation step whereas the latter renders the current frame and displays the result on the screen. These stages can be subdivided when considering particle engines: emission and update. As we may observe that update operation concerns emission of new particles and the later refresh of alive ones' state; in the case of rendering we need to send the new or modified data to the graphics card and launch the render process, and depending on the nature of the chosen rendering method, it may be necessary to perform a sort operation.

Concerning the two subtasks mentioned compounding the update stage, as far as from the same data set different results might be desired, both emitter and updater modules are included as part of the framework, the first one being in charge of emission; i.e. the initialization of particles when they become “alive", being the second one in charge of updating their state through simulation and modifying in consequence their position, color, size, etc. These modules can be reimplemented for both initializing and updating in diverse ways according to application needs; e.g. if our program does not need to apply acceleration, we can override the functionality to save derived operations and improve application performance. These two modules can be accessed separately, referred herein on as “Update" components.

Taking into account the conservative render operation we described above, there is a renderer component which might be reimplemented for various render APIs such as DirectX or toolkits such as OpenSceneGraph. The renderer component is in charge of uploading related data to graphics card and afterwards, rendering the particles.

Particle engines usually allow to render diverse shapes or surfaces according to system needs, but the most commonly used is to render rectangles or triangles, in which several kinds of effects can be projected such as textures, colors, procedural image generation and so on. On the further text we will focus on rectangle-based (quads) rendering for it is both the most common and flexible, in the particular case of billboards, also known as impostors or in 2D, sprites. Billboards have the property of being always perpendicularly facing the camera, so they are rendered as squares or rectangles without taking into account the rotation in terms of vertices and fragments rendering. Our render approach is based on the instantiation feature proviced by OpenGL, which allows to save memory use and cost on the GPU. We also avoid to send dead particles as can be seen on Fig. 2 on the final stage.

When addressing particles rendering based on quads, as far as the desired result is more complex than colored rectangles moving along the screen it is necessary to set a transparency factor. This allows to create an infinite number of effects but with the counterpart it brings with it, which is the alpha blending, requiring to render objects in different and, in consequence, more inefficient way than when dealing with opaque 3D objects. The simplest solution, and the one discussed on this paper, is known as *Painter's algorithm*, consisting on rendering from farther to closer objects (from camera's point of view), so the closest objects are painted on top of farthest ones, as a real life painter would do when painting a canvas. In computer graphics this leads to a sorting operation concerning all transparent objects involved in scene. In the case of particle systems, which are expected to display thousands or even millions in nowadays real-time graphics applications, this becomes a significant time-consuming process, as sorting operation in a single core scales over $\mathcal{O}(n\ log\ n)$ computational cost. Over the last few years, GPGPU solutions have been released for the performance improvement of this particular issue.

The sorting operation input has been reduced to the operating minimum, which is particles IDs and distances, as the a sorted array of IDs is the only output. By this way, we reduce the consume of memory bandwidth and time spent on data transferring.

Considering sorting as a common problem, a sorter component is included on the system, for potential reimplementations or improvements. For this paper, the PREFR working base include a single core CPU and a CUDA Thrust! sorter components which after development have been easily exchanged with a single line of code. Performance issues will be discussed in detail later on.

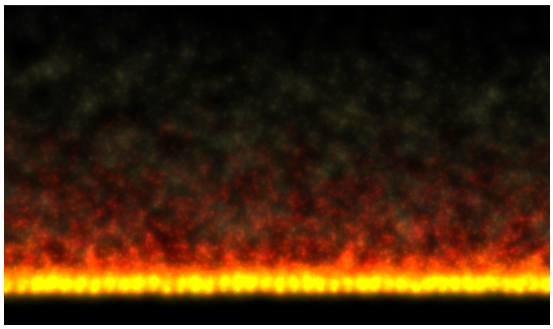
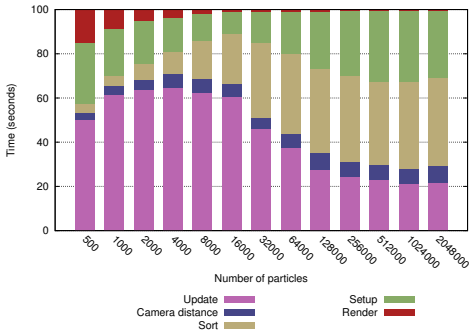


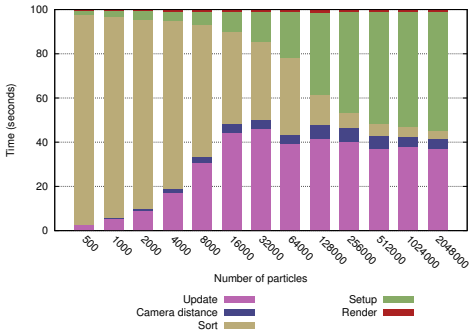
Figure 3: *Fire effect demo used for performance testing. In this study, beyond a small number of particles (>65k) GPU sorting outperforms the CPU implementation.*

Sorting operation also leads to two other tasks we externalized as reimplementable methods with future improvements in mind. First one is calculating particles distance to the camera, for the later sorting. This one, due to the strong coupling with the sorting operation is placed within the sorter component. The second one is the matching of the sorted distance array with their corresponding particle object containing its current position, size and color, placed within the renderer component due to its absolute dependence from render type and module.

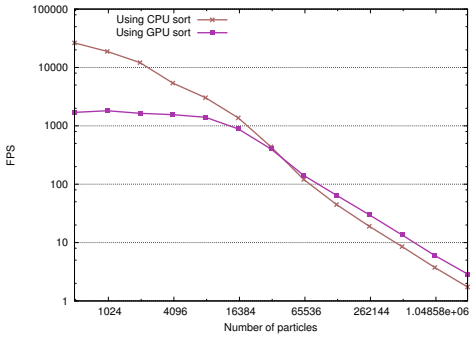
In order to define the way particles should look and behave like, we designed a particular component to allow this task. Particles aspect and behavior might be defined by a considerable number of parameters such as position, rotation, size, color, velocity, among others. However, we defined a minimum of parameters to determine their variations over time as lifetime, color, size and velocity, which allow to vary particles position, color and size along their entire life cycle. These attributes determining the behavior are stored in a particular module in charge of providing the corresponding values of each property for a given particle and life value, allowing to easily define the values obtained by particles along the simulation.



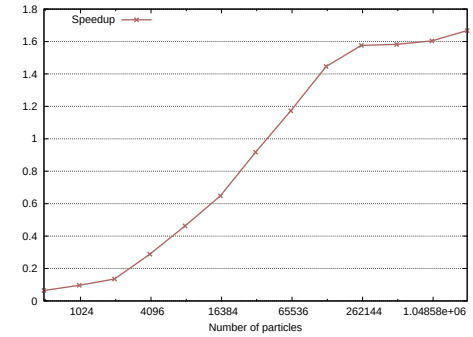
(a) Execution with sort operation on CPU. Percentage of execution time



(b) Execution with sort operation on GPU. Percentage of execution time



(c) Performance in FPS: CPU vs. GPU



(d) Speedup: CPU vs. GPU. At 65k GPU sorting outperforms the CPU version.

Figure 4: *Performance comparison, with increasing number of particles, between CPU and GPU versions.*

The current approach allow to define a series of life key points and their attribute values, which are linearly interpolated in the intermediate points. By this way, we may define a piecewise function with a low computational cost for each attribute.

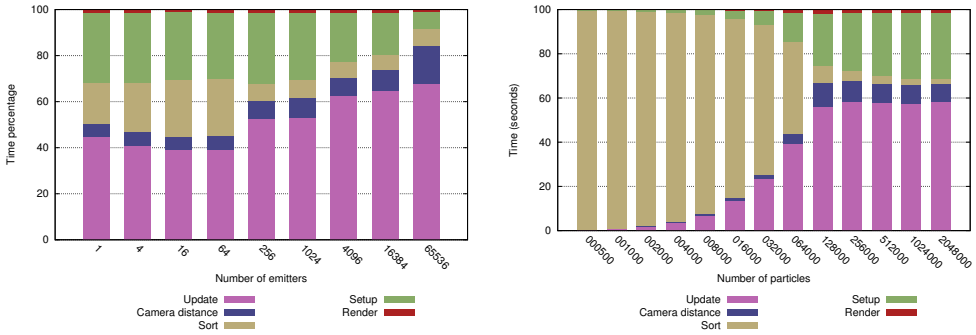
Emission nodes, as by their name may result confusing due to de resemblance to emitter are different but related components. Emission nodes are a connection of the particle engine with the space in the virtual world. Emission nodes are actually invisible but positioned objects over the scene, being responsible for determining particles initial position and velocity direction among other possible attributes. These two are the minimum we defined on the working base of PREFR, allowing us to emit targeted particles on a given position, sampled over lines, planes, spheres or even meshes.

Emission nodes may behave also as 3D objects in terms of movement, scale and so on, being all their attributes easily modifiable during runtime as shown in both use cases, in which we show how they move or change the color they induce to particles. Following this approach it is easy and efficient to create, for instance, fire torches all over a scene by simply adding nodes to the list, sharing the configuration with the rest of the desired emission nodes.

Once a global overview of the system has been given it can be observed in detail on Fig. 2. As described, the pipeline is summarized on the emission, update, camera distance calculation, sorting, setup and finally render, returning to emission stage on the next simulation frame. Prototypes and emission nodes are accessible for emitters and updaters. Particles life cycle also can be seen, which shows how a dead particle might be emitted to be later updated and rendered through the pipeline. Other particles might die on the frame, so they will not be sent to the graphics card.

3.2. Performance analysis

We have carried out different tests to evaluate the performance of the PREFR framework. All the experiments where run on an Intel i7-4820K CPU @ 3.70GHz with 16 GB of dual channel RAM Memory and an nVIDIA 770 GTX with 2GB of memory. We used CUDA Trust! running over CUDA 7.0. Tests have been run using the fire effect demo to present results taken from a good looking example, as shown in Fig 3.



(a) Increasing number of emitters. (b) Stressing the fill rate.

Figure 5: *Distribution of execution time (%) for the sort operation on GPU, varying fill rate and number of emitters.*

A first group of tests was executed in order to observe the impact of increasing the number of particles. Fig 4(a) shows for each operation the percentage of the total execution time when it is run only on the CPU. It can be seen that increasing the number of particles means that all operations spend more execution time, but specially for the sort one. In fact, the sort operation is the one that spends the biggest amount of time, giving us reasons to do a new test with an implementation in CUDA to be run on the GPU. Fig 4(b) shows the results obtained from this second experiment, and it can be seen that running the sort operation on the GPU greatly reduces the amount of time spent on it.

The benefits of this reduction of time in the sort operation can be seen in Figs. 4(c) and 4(d). In the former it can be seen that increasing the number of particles beyond 500,000 the implementation on GPU achieves a better FPS rate. In the latter we can see that having more than 1 million particles the implementation on the GPU yields an execution time 1.6 times faster than the implementation on the CPU.

More experiments have been run changing the number of emitters in the simulation. Fig. 5(a) shows the results collected running the simulation on the GPU. It can be noticed the robustness of the PREFR framework, since the amount of time spent in the sort operation decreases as the number of emitters increases.

A last set of experiments were run to evaluate the behavior of the PREFR framework stressing the fillrate. Once again, it can be seen in Fig. 5(b) that as the number of particles grows, the time spent in the sort operation operation is greatly reduced but with a small increase on time on the render operation due to fill rate and the CUDA and OpenGL context exchanges.

3.2.1. Comparison

A set of preliminary tests have been run over simple both OpenSceneGraph 3.2.1 and OGRE v1.9 examples consisting on an empty scene with a single emitter and 100.000 sorted particles, obtaining from stable simulation a raw performance of 7 and 12 FPS respectively, whereas PREFR performs 30 FPS using a set of CPU-based components.

Looking through both OGRE's and OpenSceneGraph's source codes, up to the authors' knowledge, both engine's introduce a significant overhead when addressing both sort and render operations. On one hand, this may be caused by the sorting generalization, including additional data which might slow down the operation. On the other hand, the render operation is also generic, integrated in both engines with other parts of the scene, leading to render particles one by one instead of using the batch scope addressed in PREFR.

4. Use cases

In order to show the flexibility and possibilities of the PREFR framework in this section we tackle two very different use cases, each requiring different approaches and a diverse and distinctive set of parameters. On one hand we present an example of displaying neuronal activity and thus we are using particles to present dynamic data and on the other hand we use particles to display Geographical Information System (GIS) data, in this case static information, and we show how to use our framework to gain insight on complex data by visual aggregation.

4.1. Neuron activity

In this use case we show a possible representation of neuronal activity information. The data used in this example comes from the Blue Brain Project and is composed of a synthetic microcircuit from the cortex area of the brain. This microcircuit has a column which is composed of 10 minicolumns and each minicolumn is composed of 100 neurons. From the whole microcircuit we chose a subset of neurons and posed a set of particles moving along the dendrites and following its trajectory to the soma in order to, although is

not a realistic simulation, mimic the electric impulses that may arise in the brain, as it can be seen on Fig. 6.

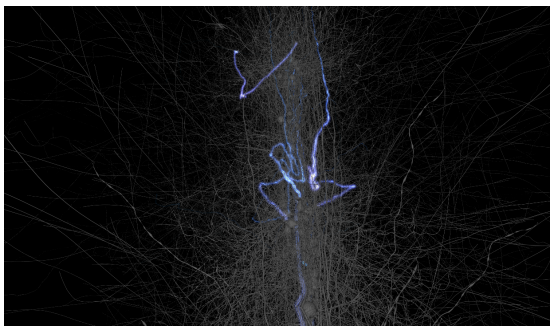


Figure 6: *Neuronal activity represented with particles*

There can be different approaches for presenting this activity information but we have chosen the one that has the best trade-off between the number of alive particles and the amount of additional computations besides the actual particle ones.

In the data used neurons are described in form of a set of connected nodes that form a tree for each of the their neurites (dendrites and axon). Therefore we computed the possible paths that go from dendrite leaf nodes to the soma and in order to make particles follow the trajectory of the neurites we placed an emitter at the beginning and move it in each step following the node chain of the neurite.

Each emitter has a number of particles, which can be user-configured, and has also a velocity which defines the distance it will be displaced from one step to another. In each step the emitter spreads particles with a lifespan greater than the time step which makes particles leave a trace of the impulse. Each particle is set to render a quad impostor which has its alpha channel decaying with respect to the center, thus achieving a blurring effect on the zones farther from the center of the impostor. For making the effect clearer we can easily configure the alpha decay function. Using the ability of PREFR to easily configure color and size with respect of the lifespan we can also fine tune the amount of trace, the size of the head of the impulse, etc., thus allowing the user to seek easily and interactively the final effect achieved.

4.2. GIS data

In this use case we show updated information from Madrid city about shops, establishments and properties as well as the activities carried out in each of them. This information is available at the Madrid City Council website [dcaMcc14], that keeps a list classified by categories of activities.

The approach we have chosen for presenting this information is based on setting up one particle per shop, establishment or property. The particle's color is chosen based on the category of activity, but taking into account their position in the color spectrum so as to be able to achieve some useful aggregation. The color has maximum intensity at the center of each particle, and its alpha follows an exponential function to color the surface covered by the particle. Finally, we have set different sizes for particles corresponding to different categories.

PREFR framework is flexible and configurable, so we have to say that the user can tune all the mentioned parameters to its convenience.

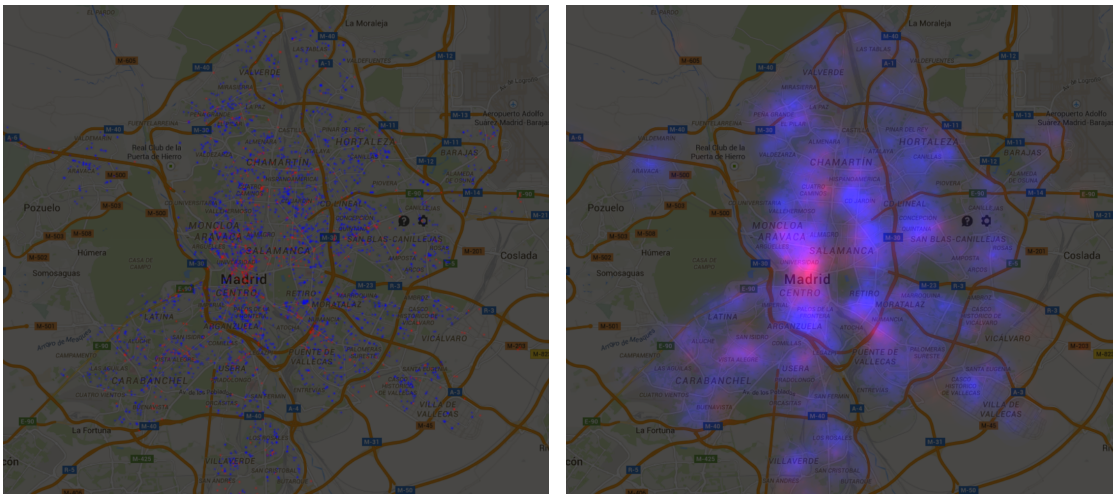
We have used the PREFR framework on information about the geographical distribution of schools including also pre-school and primary school (blue information in Fig. 7) as well as the distribution of places whose activities are not appropriate for under eighteens (red information in Fig. 7), like sex and gun shops, betting houses, etc.

Figure 7(a) shows the position of both types of activities without depicting their area of influence. It has been done using a point based representation of the particles. It can be seen that schools are spread throughout the city, while the other activities are less predominant but with an important concentration in the north part of the city center, as well as another noticeable presence south-east, under the area labeled as *NUMANCIA* in the map.

An increase in the particles' size allows to set up some influence area around each point, as we show in Fig. 7(b). It can be seen the appearance of some mauve areas as a result of the aggregation of red and blue colors. For example, in the north of the area labeled as *UNIVERSIDAD* in the map, there is one of these mauve regions, meaning that schools are close to places where the other activities take place. A discussion about the appropriateness of this situation can be easily settled.

Following this approach, we have also analyzed the geographical distribution of retail stores (blue data in Fig. 4.2) and malls (red information in Fig. 4.2), along with their regions of influence. As mentioned, to do that all particle parameters can be tuned up by the user to its convenience.

There are two noticeable red regions close to the city center. One slightly north of the center, and another one on the east part of a neighbourhood labeled *SALAMANCA* in the map (see Fig. 4.2). These are two populated (as well as quite popular) zones of the city, so it is normal to find this strong red areas surrounded by mauve and blue ones. But it can be easily seen that there are some red areas like the one south of the label *SAN FERMIN*, the one south east of *ATOCHA*, the one at *SANTA EUGENIA* and so on, isolated from blue ones. So it can be said that the influence of this shopping malls prevents from opening retail stores.



(a) Point based representation of particles (b) Particles with area of influence

Figure 7: *School vs bad influences*

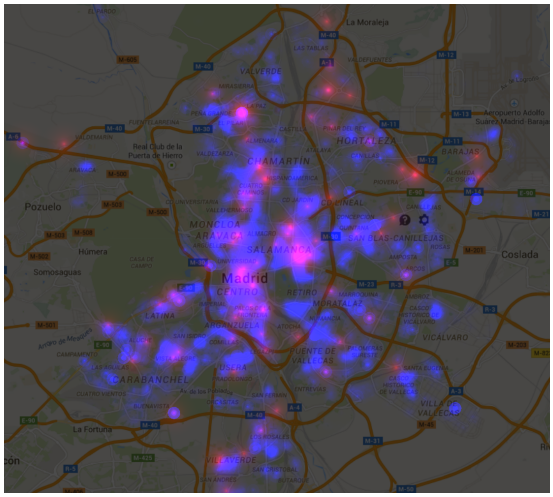


Figure 8: *Retail stores vs. malls*

5. Conclusions and future work

This paper presents PREFER, a particle rendering framework designed with two goals in mind: easiness of configuration and efficiency. Regarding the first issue, this paper presents two very different use cases, giving also an idea of the framework's flexibility for different application areas. Regarding this last issue, from our performance analysis we have identified parts of the pipeline which will be potentially moved to the GPU in future versions of this engine, increasing the overall performance of the system if high-end GPU units are available.

We also think that there are at least, two areas which require further research:

- The range of particle editing parameters exposed to the user. It should maximize expressiveness while keeping an intuitive and manageable set. Different dataset will require optimized controls which could even be inferred from the kind of visualization desired by the user.

- The mathematical functions or textures describing transparency at each particle have a direct impact on cluster perception and they should be studied and validated by means of psychophysical experimentation.

Finally we plan to publish our particle engine as an open source tool for visualization researchers.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 604102 (HBP), the European Research Council (ERC-2011-StG-280135 Animetrics), and the the Spanish Ministry of Economy and Competitiveness (Cajal Blue Brain Project, Spanish partner of the Blue Brain Project initiative from EPFL and grant TIN2014-57481). The work of Jorge Lopez-Moreno was funded by the Spanish Ministry of Science and Education through a Juan de la Cierva-Formacion Fellowship. The data used in the neuroscience examples has been provided from the Blue Brain Project.

References

[Can07] CANTLAY I.: Chapter 23: High-speed, off-screen particles. In *Gpu Gems 3*, Nguyen H., (Ed.). Addison-Wesley Professional, 2007. 2

[dcaMcc14] DATA CATALOG AT MADRID CITY COUNCIL O.: Censo de locales y sus actividades. http://datos.madrid.es/portal/site/egob, 2014. 7

[EKS83] EDELSBRUNNER H., KIRKPATRICK D., SEIDEL R.: On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on 29*, 4 (Jul 1983), 551–559. 2

[KKKW05] KRUGER J., KIPFER P., KONCLRATIEVA P., WESTERMANN R.: A particle system for interactive visualization of 3d flows. *Visualization and Computer Graphics, IEEE Transactions on 11*, 6 (Nov 2005), 744–756. 2

[PBN*13] PACKER E., BAK P., NIKKILA M., POLISHCHUK V., SHIP H. J.: Visual analytics for spatial clustering: Using a heuristic approach for guided exploration. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2179–2188. 2

[Per12] PERSSON E.: Graphics gems for games: Findings from avalanche studios(link). Siggraph 2012 course Advances in Real-Time Rendering in Games, 2012. 2

[RB85] REEVES W. T., BLAU R.: Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1985* (1985), pp. 313–322. 2

[Ree83] REEVES W. T.: Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2 (1983), 91–108. 2

[Roc14] ROCKENBECK B.: The infamous:second son particle engine, 2014. 2

[Sev13] SEVO I.: A million particles in cuda and opengl, 2013. 2

[Ste14] STENHOLT R.: On the benefits of using constant visual angle glyphs in interactive exploration of 3d scatterplots. *ACM Trans. Appl. Percept. 11*, 4 (Dec. 2014), 19:1–19:23. 2

[Tho14] THOMAS G.: Compute-based gpu particle systems. http://www.gdcvault.com/play/1020002/Advanced-Visual-Effects-with-DirectX, 2014. 2

[vdB00] VAN DER BURG J.: Building an advanced particle system, 2000. 2

[vW91] VAN WIJK J. J.: Spot noise texture synthesis for data visualization. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991), SIGGRAPH '91, ACM, pp. 309–318. 2

[ZSH96] ZOCKLER M., STALLING D., HEGE H.-C.: Interactive visualization of 3d-vector fields using illuminated stream lines. In *Visualization '96. Proceedings.* (Oct 1996), pp. 107–113. 2