# Parallel Visualization of the Sloan Digital Sky Survey DR6

Balázs Domonkos Kristóf Ralovich
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Magyar tudósok krt. 2., H-1117, Budapest, Hungary
domonkos@iit.bme.hu, kristof.ralovich@gmail.com

**ABSTRACT**

The new generation astronomy digital archives cover large area of the sky at fine resolution in many wavelengths from ultraviolet through optical and infrared. For instance, one of these projects the Sloan Digital Sky Survey is creating a detailed catalog covering more than a quarter of the sky with images measured with five different filters. The size of the data set can be measured in terabytes. These archives enable astronomers to explore the data for their research. However, virtually walking through these huge data sets also enables to visualize the beauty of the Universe and raises problems which can be interesting for people related to computer graphics. In this paper we present a technique for parallel visualization of large-scale scattered astrophysical data that has wide-spectrum photometric property. Our method performs sort-last parallel particle rendering using hierarchical, static data distribution; and its performance scales up linearly by increasing the number of the rendering nodes. It also enables setting the color matching function in the rendering phase and as well as altering the distance calculation formulae that calculates spatial coordinates from the redshift – all interactively.

**Keywords:** Graphics Systems, Distributed/Network Graphics

## 1 INTRODUCTION

Up till now, the Sloan Digital Sky Survey (SDSS) is one of the largest astronomical survey ever undertaken. When completed, it will provide detailed optical images covering more than a quarter of the sky and a three dimensional atlas of about a million galaxies, quasars, and stars. As the survey progresses, the data is released to the scientific community and the general public as well. The latest release to date (SDSS Data Release 6) has been announced in June 2007. The amount of gathered and processed photometric and spectroscopic data exceeds 10 terabytes. This data contains detailed imaging and spectroscopic description of more than 800 000 astronomical objects.

This data is indisputably a treasury for the astrophysicists for checking the validity of numerous models related to the origin and evolution of the Universe and to the fundamental characteristics of the galaxy population. However, this huge data set is interesting *for itself*, too. The photometric images of the astronomical objects with aid of spectroscopic data can be visualized in three dimensions interactively in order to show the structure and the beauty of the observed part of the Universe. Moreover, it is possible to alter the color matching functions that maps the original photometric data to pixel colors in the rendering phase. On the other hand, the visualization system can be designed to enable varying the distance calculation algorithm and tuning its parameters also during the image synthesis.

However, such amount of data fairly exceeds the *memory capacity* of a recent graphics hardware. To overcome this limitation while keeping the advantage of hardware accelerated rendering that produces acceptable frame rates, the rendering has to be decomposed to run in *parallel* utilizing the cumulative processing power of multiple computer nodes. First, the data have to be distributed among the nodes, then the visualization of the partial data is performed in parallel, and finally the rendering outputs have to be composited. Both image-order (ray casting) and object-order methods (splatting or particle rendering) exist for rendering scattered data. In our work we have investigated the latter approach.

Parallel rendering necessarily raise the issue of load balancing that is originated in *data distribution strategy*, especially when the memory is the bottleneck of overall visualization task. Image-space partitioning is not feasible when using such a huge data set since it requires all nodes to be able to render potentially any part of the dataset. Because of the size of the whole data set exceeds the capacity of the system memory of a render node only object-space distribution is appropriate for interactive rendering. In case of particles representing astrophysical objects with photometric data, the rendering cost of a particle is inversely proportional to the square of its distance. When simply partitioning the data set into axis-aligned blocks according to



*WSCG 2008 conference proceedings, ISBN 80-903100-7-9*
***WSCG'2008, February 4 – 7, 2008***
*Plzen, Czech Republic.*

the number of the rendering nodes, the rendering cost per node does not necessarily decreases linearly by increasing the number of nodes (Fig. 3). However, when the data set is partitioned by distributing the leaves of a *space partitioning tree* built on the data set, a linear scale-up can be guaranteed.

In this work we used images and numerical data of more than 800 000 objects over about 8 000 square degrees of the sky for *sort-last* [8] *parallel particle visualization* using *k*D-tree data partitioning and sort-independent blending. The purpose of our rendering scheme is to support interactive visualization of such data sets. This paper summarizes our experiments and suggestions.

## 2 RELATED WORK

One of the most popular architectures is Chromium, a parallel implementation of OpenGL that allows flexible sort-first parallel rendering. Distributed particle-based simulation and rendering that uses Chromium and MPI was investigated by Smith [13]. A distributed scene graph library (Aura) was developed and compared to Chromium for parallel particle rendering by Schaaf et al. [16]. A system for real-time animation and rendering of large particle sets using GPU computation including inter-particle collisions and visibility sorting was presented by Kipfer et al. [6]. Taylor et al. discussed a parallel implementation of the visualization of galaxy formation simulation running in a grid environment using a decentralized peer-to-peer approach [15].

From the application point of view, Rosner et al. have created a movie from the SDSS Data Release 4 data set walkthrough [12]. Subbarao et al. have made a three dimensional model of the galaxies and quasars found by the SDSS. They visualized 250 000 galaxies and 40 000 quasars including the cosmic microwave background radiation. Their model is interactive, which means one can fly around in it exploring both galaxies close up and the large scale structure of the Universe [14]. The Extragalactic Atlas of the Digital Universe visualization program by Hayden Planetarium can render the whole SDSS Data Release 6 data set [4]. For the preceding movie and the applications Partiview was used which is an interactive open-source tool from the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign [9].

## 3 PREPROCESSING THE SDSS DR6 DATA SET

The rendering scheme of ours can be divided into three main stages. First, the data used for rendering is downloaded from the SDSS servers and *preprocessed* to meet the requirements of the graphics hardware. This long process that have to be performed once is detailed in this section. Before starting the effective rendering an *initialization step* has to be performed during application startup in which the geometry and the textures are computed (Sec. 4). This is followed by the visualization step in which the rendered frames are produced and the user inputs are handled (Sec. 5).

The SDSS Data Release 6 data is distributed via the *Catalog Archive Server* (CAS) which is an SQL database that contains the measured spectroscopic properties of the astrophysical objects, and the *Data Archive Server* (DAS) which is a file server storing the outputs of the imaging pipelines. From now on we will refer these as the structural (or spectroscopic) and image (or photometric) data, respectively. For creating our data set, we have queried all the records that has accurately measured spectroscopic data (redshift, viewing angle, etc.) from the SQL database and then retrieved the photometric data for these objects from DAS; i.e. the corresponding image taken by the SDSS telescope for every single object.

DAS contains images of the emitted spectrum of galaxies, quasars and stars recorded with five different filters. We preferred to keep the possibility of post-shading the objects. That means, one could interactively modify the color matching functions either to enhance a small frequency domain or to get a comprehensive view of the whole spectrum. On the other hand though, it is possible to handle these five color channels on the GPU at the cost of multiple textures and a more complex logic in the pixel shader, it is reasonable to choose a trade-off between the performance and the accuracy. In our solution the photometric data were transformed from the five-channel UGRIZ color space (ultra violet, blue-green, red, far red and near infrared pass band filters [3]) to four-channel images that have the same extent in the frequency domain but fit better to the 4-wide SIMD architecture of the graphics hardware.

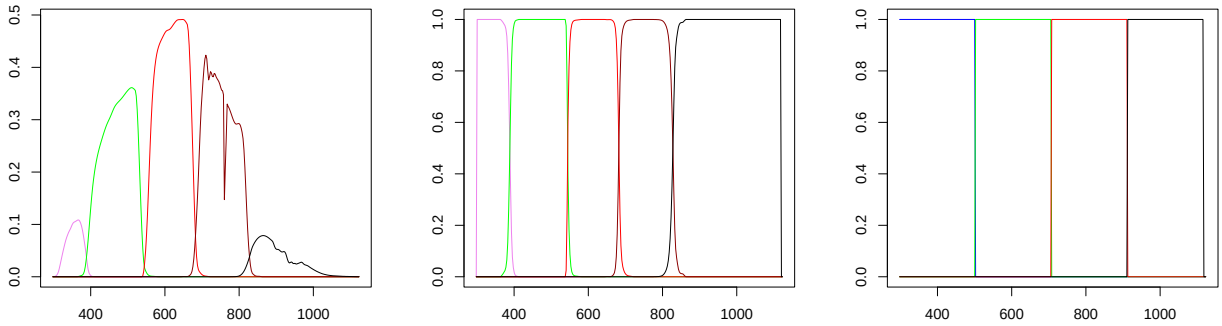
The original $f_i(\lambda)$ *color matching functions* illustrated in Fig. 1 (a) are described on the SDSS web site while the pixel values $c_i$ are known from the downloaded images for each filter $i$. However, the originally measured $\Phi(\lambda)$ spectrum cannot be calculated from these quantities. We treated $\Phi(\lambda)$ as a constant $\Phi_i$ for each filter:

$$c_i = \int_\lambda \Phi(\lambda) f_i(\lambda)\, d\lambda = \Phi_i \underbrace{\int_\lambda f_i(\lambda)\, d\lambda}_{F_i} \quad (1)$$

From these $\Phi_i$ values an *estimated spectrum* can be calculated using the weight functions $w_i(\lambda) = f_i(\lambda) / \sum f_i(\lambda)$ (Fig. 1 (b)):

$$\Phi_{est}(\lambda) = \sum_j w_j(\lambda)\Phi_j = \sum_j w_j(\lambda)\frac{1}{F_j}c_j \quad (2)$$

The new pixel values can be computed refiltering this estimated spectrum with the new four-channel color




(a) Original UGRIZ filters. (b) Filter weights $w_i(\lambda)$ (c) Our 4-channel filters (stored in RGBA format in our rendering system)

Figure 1: Transformation from 5-channel to 4-channel filters. The abscissa is the wavelength in nanometers and the ordinates show the transmission of the filter for figures (a) and (c) and the weighting values for figure (b)

matching functions $f_i'(\lambda)$ . We applied simple box filters (Fig. 1 (c)) partitioning the spectrum into four intervals with equal extent between $\lambda_0$ and $\lambda_1$, $\lambda_1$ and $\lambda_2$, and so on:

$$\begin{aligned} c_i' &= \int_\lambda \Phi_{est}(\lambda) f_i'(\lambda)\, d\lambda = \int_{\lambda_i}^{\lambda_{i+1}} \Phi_{est}(\lambda)\, d\lambda = \\ &= \int_{\lambda_i}^{\lambda_{i+1}} \sum_j w_j(\lambda) \frac{1}{F_j} c_j\, d\lambda = \\ &= \sum_j \underbrace{\frac{1}{F_j} \int_{\lambda_i}^{\lambda_{i+1}} w_j(\lambda)\, d\lambda}_{C_{i,j}}\ c_j \end{aligned} \tag{3}$$

According to (3), the transformed color vector $\mathbf{c_i'}$ can be efficiently calculated multiplying the 4-by-5 matrix $[C_{i,j}]$ by the input color vector $\mathbf{c}$.

The original-scale images are resampled to $32 \times 32$ smaller images also in the preprocessing step. This infers only marginal information loss, since the vast majority of the images did originally fit into this size. In order to reduce the size of the data stored offline, every image is compressed using the lossless DEFLATE algorithm.

## 4 INITIALIZATION STEP

In the following sections the preliminary computations are introduced that precede the rendering steps. First, during application startup the structural data is read in, and spatial coordinates are calculated from the redshift values based on a given parametrized cosmological distance model (Sec. 4.1). Then the objects are distributed among the rendering nodes based on their position (Sec. 4.2). The next section explains how the spatial structure of the data set is calculated that has to be distributed.

### 4.1 Distance Measures in Cosmology

The small-scaled concept of distance between two points in our immediate environment cannot be extended to cosmological scales. Since the distances

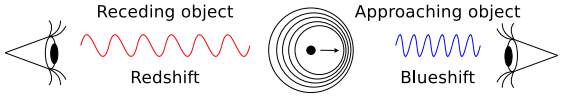



Figure 2: Redshift and blueshift in wavelength due to the relative motion

between comoving objects are constantly changing in the expanding Universe, and since the Earth-bound observers look back in time as they look out in distance, many *distance measures* can be defined [5]. They are often based on observable quantities such as the wavelength shift of a receding galaxy or the luminosity of a distant quasar. However, the concept of "distance measurement" can be treated more generally. For instance the time elapsed since the emission and the observation of the photons (lookback time) can be considered as distance measure as well.

The dominant motion in the Universe is the expansion described by Hubble's Law. It states that the observed velocity of a distant galaxy away from us is proportional to its distance, where the proportion coefficient $H_0$ is called *Hubble constant*. it is currently believed to be about 77 km/sec/Mpc. The symbol "Mpc" denotes mega parsec which is approximately $3.09 \cdot 10^{16}$ meters.

Light from moving objects appears to have different wavelengths depending on the relative motion of the source and the observer. An observer looking at an object that is moving away receives light that has a longer wavelength than it had when it was emitted. For optical wavelengths this means that the wavelength of the emitted light is shifted towards the red end of the electromagnetic spectrum. More generally, any increase in wavelength is called *redshift* Conversely, a decrease in wavelength is called blueshift (Fig. 2).

Redshift $z$ can be calculated as the Doppler shift of its emitted light resulting from radial motion:

$$z \equiv \frac{\lambda_o}{\lambda_e} - 1, \tag{4}$$

where $\lambda_e$ is the emitted and $\lambda_o$ is the observed wavelength. The cosmological redshift is directly related to

the *scale factor* $a(t)$ of the Universe, which is a function of time and represents the relative expansion of the Universe. For redshift $z$

$$1+z = \frac{a(t_o)}{a(t_e)} = \frac{1}{a}\,, \tag{5}$$

using the normalization $a(t_0) = 1$ and $a \equiv a(t_e)$ where $a(t_e)$ is the scale factor when the photons were emitted, and $a(t_o)$ is the scale factor at the time they are observed.

**Distance Measures**

The small comoving radial distance $\Delta D_{CMR}$ between two nearby objects in the Universe is defined as the distance between them which remains constant when the two objects are moving with the Hubble flow [5]. Generally, the **comoving radial distance** $D_{CMR}$ of two objects is computed by integrating the infinitesimal $\Delta D_{CMR}$ contributions between nearby events along a radial ray [18]:

$$D_{CMR} \;=\; \int \frac{c}{a}\,dt = \int_{\frac{1}{1+z}}^{1} \frac{c}{a\dot{a}}\,da, \tag{6}$$

where $c$ is the speed of light and $\dot{a}$ is the time derivative of $a$. The **light travel time** $D_{LTT}$ is calculated similarly: [18]

$$D_{LTT} \;=\; \int c\,dt = \int_{\frac{1}{1+z}}^{1} \frac{c}{\dot{a}}\,da\,. \tag{7}$$

The *mean mass density* $\rho$ of the Universe and the value of the *cosmological constant* $\Lambda$ are dynamical properties of the Universe which affect the time evolution of the metric [17] [5]. They can be converted into dimensionless density parameters by [11]

$$\Omega_M \;=\; \frac{8\pi G\rho_0}{3H_0^2} \quad \text{and} \quad \Omega_\Lambda = \frac{\Lambda c^2}{3H_0^2}, \tag{8}$$

where $G$ is Newton's gravitational constant. There are two additional density parameters: the *radiation density* $\Omega_r$ and the *curvature term* $\Omega_k = 1-\Omega_M-\Omega_\Lambda-\Omega_r$ [18].

Using the Newtonian approximation to capture the dynamics of the Universe $\dot{a}$ can be substituted by $H_0\sqrt{X(a)}$ with [18]

$$X(a) \equiv \frac{\Omega_M}{a} + \frac{\Omega_r}{a^2} + \Omega_\Lambda a^2 + \Omega_k\,. \tag{9}$$

This enables to calculate (6) and (7) from redshift $z$

$$D_{CMR} \;=\; \frac{c}{H_0}\int_{\frac{1}{1+z}}^{1} \frac{1}{a\sqrt{X(a)}}\,da \;\text{ and} \tag{10}$$

$$D_{LTT} \;=\; \frac{c}{H_0}\int_{\frac{1}{1+z}}^{1} \frac{1}{\sqrt{X(a)}}\,da\,. \tag{11}$$

The **angular diameter distance** $D_A$ can be calculated directly from $D_{CMR}$ as follows: [5]

$$D_A \;\equiv \frac{R}{\Theta} = \tag{12}$$

$$= \;\frac{c}{H_0\,(1+z)} \cdot \begin{cases} \frac{1}{\sqrt{\Omega_k}}\sinh\left(\frac{H_0\sqrt{\Omega_k}}{c}\,D_{CMR}\right) & \text{for } \Omega_k > 0 \\ D_{CMR} & \text{for } \Omega_k \approx 0 \\ \frac{1}{\sqrt{\Omega_k}}\sin\left(\frac{H_0\sqrt{\Omega_k}}{c}\,D_{CMR}\right) & \text{for } \Omega_k < 0 \end{cases}$$

The **luminosity distance** $D_L$ is related to the angular diameter distance [18]:

$$D_L \equiv \sqrt{\frac{L}{4\pi S}} = (1+z)^2 D_A\,. \tag{13}$$

We do the *numerical evaluation of the integrals* (10) and (11) using the mid-point rule with ten million panels. Instead of evaluation for each object, they are sorted by ascending redshifts and the distance integrals are evaluated for all objects in a single pass through the sorted redshifts. Moreover, $D_{CMR}$ and $D_{LTT}$ values are calculated in parallel while calculating $D_A$ and $D_L$ does not need any iterative calculation only evaluation of explicit formulae (12 and 13). The total time cost of the calculation for the whole data set is under a second on a 2 GHz AMD64 processor.

## 4.2 Data Distribution

The data set is partitioned among $N$ rendering nodes by distributing the astrophysical objects. The distribution is based on the spatial coordinates of the objects that are calculated in the preceding section. It is achieved as a result of building a $k$D-tree over the whole data set – constrained by the fact that all except one of the leaves of the tree must contain $N$ particles – and uniformly distributing the contents of the leaves (Fig. 4). This is more favorable than simple chopping the scene into axis aligned blocks according to the number of rendering nodes. The former approach guarantees practically linear scale-up in the rendering frame rates since the data set partitions have uniform spatial distribution. The scale-up is worse for the latter one when only the particles per node ratio is reduced by adding more nodes to the system but the particles are assigned to the nodes as a spatially centralized way (Fig. 3). Unfortunately, the other side of the coin is that the $k$D-tree distribution cannot be efficiently used with sort-dependent blending operators, since each node generates images not for a convex volume but for any part of the space; and the complete ordering of the object images would be required. Even so, when the scalability and the load balancing strategy has great importance a space partitioning tree aided data distribution can be preferred.

As a final step of the initialization the spatial data is uploaded to the graphics hardware and the interactive visualization is started.

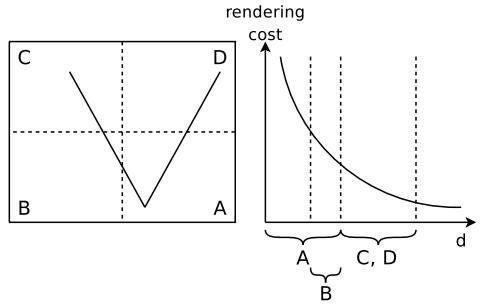



Figure 3: Rendering using block partitions. The rendering cost of a particle is proportional to the area of its projection on the camera image, thus it is inversely proportional to the square of its distance. The rendering cost is *not* decreasing linearly with the increasing number of nodes. The load is not balanced well among the nodes therefore the overall rendering time is dominated by the most loaded node. (Nodes B, C, and D have to wait until Node A completes the rendering.)

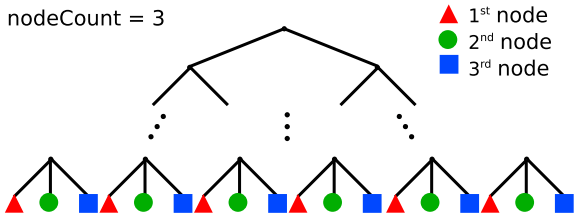



Figure 4: Distributing the contents of the leaves of the data splitting *k*D-tree during initialization.

# 5 RENDERING

The following subsections discuss the sort-last [8] parallel rendering process in detail. The rendering is accomplished separately on each node of the cluster, while the final parallel compositing of the partial images is performed as a co-operation of the nodes.

## 5.1 View Frustum Culling

The most obvious way of visualizing such number of astrophysical objects is by the means of a particle system. But since in our case each particle has a corresponding unique texture – derived from the image recorded by a telescope – the major issue is the high memory requirement instead of the large number of the particles.

In order to avoid unnecessary rendering *view frustum culling*was applied using a *k*D-tree space partitioning scheme. The data distribution hierarchy described in Sec. 4.2 sports also a straightforward way to cull invisible geometry: the tree is traversed from the root node, an intersection test is performed between the viewing frustum and the axis aligned bounding box (AABB) of the children nodes. If an AABB turns to be outside the viewing frustum [2] its descendants do not have to be processed thus all the belonging points can be culled.

During the construction of the tree it can be assured, that the resulting tree is well balanced by choosing the position of an axis aligned splitting plane as a median of the corresponding coordinate of the objects. Since a balanced tree can easily be represented as an array of its nodes, a simple linear vertex buffer [10] is capable of storing the positions for all the points. The additional advantage of using vertex buffers is they are stored in the graphics memory requiring to upload them only once.

## 5.2 Batch Rendering

Sending the image of each particle by itself to the OpenGL rendering system would result in too many API calls (not to mention that OpenGL cannot handle so many textures objects concurrently) thus frittering away the well-known performance potential of batch rendering large parts of the visible particles. Our strategy for avoiding this situation is packing sets of individual particle images into larger textures, so called *texture atlases* (e.g. OpenGL square textures with size of $ATLAS\_SIZE = 512$). The atlases are filled with the images of the particles using the fast `glTexSubImage2D` function replicating a tile pattern (*IMAGE_SIZE* was 32 in our case). Rendering all the particles corresponding to an atlas can be performed with a single `glDrawArrays` function call. To make the GPU able to recall which part of an atlas belongs to an actual particle, a 2D offset is calculated and assigned as a vertex attribute. Moreover, it is worth using multiple atlases in a round-robin fashion in order to defer synchronization between the CPU and the GPU.

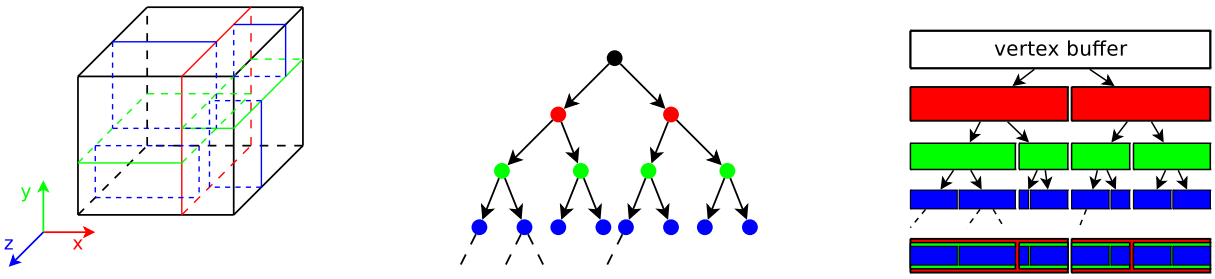
This technique seems to exploit the asynchronous operation of the CPU and GPU, keeping both of them busy. On the other hand, the high traffic generated by texture uploads causes the bus to become the performance bottleneck but on our cluster configuration this setup yielded the highest frame rates. See Sec. 7 for other possible approaches.

From the number of images $t = \lfloor \frac{ATLAS\_SIZE}{IMAGE\_SIZE} \rfloor^2$ that fit into one atlas we can express the number of rendering passes $n = \lceil \frac{p}{t} \rceil$ required to visualize the number of visible particles $p$ returned by view frustum culling.

## 5.3 Color Matching and Blending

The fixed function OpenGL pipeline is replaced with a pair of CG vertex and fragment shader programs [1]. Point sprites are used to visualize the particles, that is for each astrophysical object a textured point primitive is rendered. The following vertex attributes are assigned to the points (see the Cg snippet below): `position` is the position of the particle, `texCoord0` is the texture coordinate generated by rasterization while `texCoordOffset` is the 2D offset into the atlas.

```
struct VertexInput {
  float4 position       : POSITION;
  float2 texCoord0      : TEXCOORD0;
  float2 texCoordOffset : TEXCOORD1;
};
```




(a) *k*D-tree space subdivision scheme. (b) Balanced *k*D-tree nodes. (c) Mapping tree nodes to vertex buffer sequences.

Figure 5: Space partitioning and scene representation using a balanced *k*D-tree.

The vertex shader is quite simple, it passes the texture coordinates through and in addition adjusts the proper point size for the sprite considering its distance from the eye. The pixel shader is where the actual texturing and color adjustment takes place. Although, particle sorting could be performed fast on the GPU, when using final compositing of images of *k*D-trees sorting the huge number of partial images before blending is unfeasible. Therefore, we gave up the order-dependent part of the *over* operator that should be applied to capture the attenuation of a distant object's light obscured by a closer one; and kept only the order-independent additive part using the following blending equation:

$$\begin{aligned} PixelColor &= (IncomingColor * IncomingAlpha) \\ &+ (DestinationColor * 1.0)\,, \end{aligned}$$

where $IncomingAlpha$ is the product of the average of the incoming color channels (Fig. 1(c)) and the intensity attenuation factor. The value of this factor is constant 1.0 until the rendered size of the particle reaches the size of a pixel; then it falls proportionally to the subpixel area of the particle's image.

### 5.4 Final Compositing

In the last phase of the image synthesis the partial images generated by the rendering nodes are transferred through the interconnection network from one node to another. For compositing, we applied the *parallel pipeline* compositing algorithm [7] consisting of two stages. The images to be composited are divided into $N$ framelets, which is the number of the compositing processes. In most implementations, $N$ equals the number of the rendering processes as well since every node both renders and composites. In the first part of the algorithm these framelets flow around through each node in $N-1$ steps, each consisting of a compositing and a communication stage. After $N-1$ steps each processor will have a fully composited portion of the final frame. The framelets are collected for an external display node or for an internal node in the second part in one step. The clear benefit of this compositing scheme is that the amount of data transferred on the network in one step is independent of the number of compositing processes.

## 6 RESULTS

For our experiments we used a Hewlett-Packard "Scalable Visualization Array" consisting of four rendering nodes. Each node has a dual-core AMD Opteron 246 processor, 2 GBytes of memory, an nVidia Quadro FX3450 graphics controller with 256 MBytes graphics memory, and an InfiniBand network adapter.

The initialization step (different distance calculations, space partitioning, and loading all the images) could be performed under a minute for the whole data set. The color matching functions, the distance calculation models, and the parameters of these could be altered during the visualization.

To illustrate the scalability of our rendering system, configurations of one up to four rendering nodes were investigated for different subset of the SDSS DR6. One of the nodes displayed the final output on a $800 \times 600$ viewport. The average frame rates and their standard deviation calculated for 500 frames are illustrated in Table 1. The rendering results are presented in Fig. 6. For creating these images comoving radial distance was applied with $H_0 = 77$ km/sec/Mpc, $\Omega_M = 0.27$, $\Omega_\Lambda = 0.73$, and $\Omega_r = 7.02\,10^{-5}$; according to [18].

It is hard to make any valuable comparison between the results presented by other interactive approaches (e.g. [4], [12], or [14]) and our achievements. This is because – according to our best knowledge – other interactive simulations do not use unique images for every visualized particle. The other factor limiting the direct comparison is that Partiview-based visualizations [9] lack support for distributed computation and programmable graphics pipeline.

All the source code of the tools were used for downloading and preprocessing the SDSS data files as well as the final data set ($\approx 7.3$ GB at full resolution with compression) are available from the authors upon request. Comments and corrections are highly appreciated.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper we have demonstrated that using hierarchical object-space partitioning a large-scale scattered astrophysical data set can be efficiently visualized in a

| Nodes | 1% (85 MB) | 5% (425MB) | 20% (1.66GB) | 50% (4.15GB) | 100% (8.3GB) |
|---|---|---|---|---|---|
| 1 | $8.04 \pm 0.03$ | $1.87 \pm 0.02$ | $0.46 \pm 0.00$ | N/A | N/A |
| 2 | $12.13 \pm 1.06$ | $2.95 \pm 0.09$ | $0.83 \pm 0.01$ | $0.35 \pm 0.00$ | N/A |
| 3 | $12.99 \pm 1.52$ | $3.84 \pm 0.58$ | $1.11 \pm 0.02$ | $0.49 \pm 0.01$ | N/A |
| 4 | $14.20 \pm 1.32$ | $4.10 \pm 0.61$ | $1.52 \pm 0.01$ | $0.65 \pm 0.01$ | $0.34 \pm 0.01$ |

Table 1: Scalability results for the average frame rate when rendering increasing subsets of the SDSS DR6 data set. All test cases was measured on a $800 \times 600$ viewport. The images were downsampled to 8-bit color depth and downscaled to $32 \times 32$. The N/A sign indicates that the test case cannot be measured due to the lack of memory capacity of our nodes. The size of a texture atlas was $512 \times 512$.

distributed rendering environment using sort-last parallel particle rendering. The performance of our test system scales up approximately linearly by increasing the number of the rendering nodes. As an extra feature in order to support interactive demands, it also enables setting the color matching function in the rendering phase and as well as altering the distance calculation formula that calculates spatial coordinates from the redshift.

The disadvantage of our approach is that it *does not support efficient sort-dependent blending* for compositing the partial images, so thus the light attenuation cannot be taken into account. However, when dealing with huge data sets the scalability of the rendering frame rates has great importance. This is true especially when the amount of photometric data to be rendered on a node exceeds of the capacity of the texture memory and therefore multiple rendering passes are required within a frame; for instance for a practical data set: the SDSS DR6. In these circumstances, the space partitioning tree aided data distribution can be preferred.

As our measurements reflect, increasing the number of rendering nodes results in a near linear frame rate improvement letting us conclude that more nodes would render faster or be able to handle even larger data sets efficiently.

However, our current configuration was evidently bandwidth limited. If the graphics cards were equipped with more memory (e.g. one gigabyte would be comparable to the size of the image data handled by a node) a completely different storage method could be relevant. Some parts of the image data could be kept in the graphics memory and be accessed orders faster than continuous uploads. This would necessitate the administration where the particle images are stored.

On the other hand, applying level of detail on the particles – like replacing textured point sprites not greater than a pixel with an appropriately colored point – could also impact performance positively.

## ACKNOWLEDGEMENTS

This work has been supported by Hewlett-Packard and the Hungarian National Office for Research and Technology. We are also grateful to Dávid Koronczay for his remarkable guidance in astrophysics.

## REFERENCES

[1] K. M. J. Fernando R. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Professional, 2003.

[2] K. H. Gil Gribb. Fast Extraction of Viewing Frustum Planes from the World-View-Projection Matrix. `http://www2.ravensoft.com/users/ggribb/plane20extraction.pdf` [Online; accessed 18-10-2007].

[3] J. Gunn, M. Carr, C. Rockosi, and M. Sekiguchi. The Sloan Digital Sky Survey Photometric Camera. *Astronomical Journal*, 116:3040, 1998.

[4] Hayden Planetarium. `http://haydenplanetarium.org/` [Online; accessed 20-09-2007].

[5] D. W. Hogg. Distance measures in cosmology. *ArXiv Astrophysics e-prints*, May 1999.

[6] P. Kipfer, M. Segal, and R. Westermann. UberFlow: a GPU-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM Press.

[7] T.-Y. Lee, C. S. Raghavendra, and J. B. Nicholas. Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):202–217, 1996.

[8] S. Molnar, M. Cox, and D. Ellsworth. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[9] National Center for Supercomputing Applications. Partiview. `http://virdir.ncsa.uiuc.edu/ partiview/` [Online; accessed 20-09-2007].

[10] D. S. OpenGL Architecture Review Board, M. Woo, J. Neider, and T. Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. OpenGL. Addison-Wesley Professional, 2005.

[11] P. Peebles. *Principles of Physical Cosmology*. Princeton Series in Physics. Princeton University Press, Princeton, U.S.A., 1993.

[12] D. Rosner, R. Landsberg, and J. Frieman. SDSS Movie, 2005. `http://astro.uchicago.edu/ cosmus/projects/sloanmovie/` [Online; accessed 20-09-2007].

[13] C. Smith. Distributed Rendering of Particle Systems. Technical report, 2003.

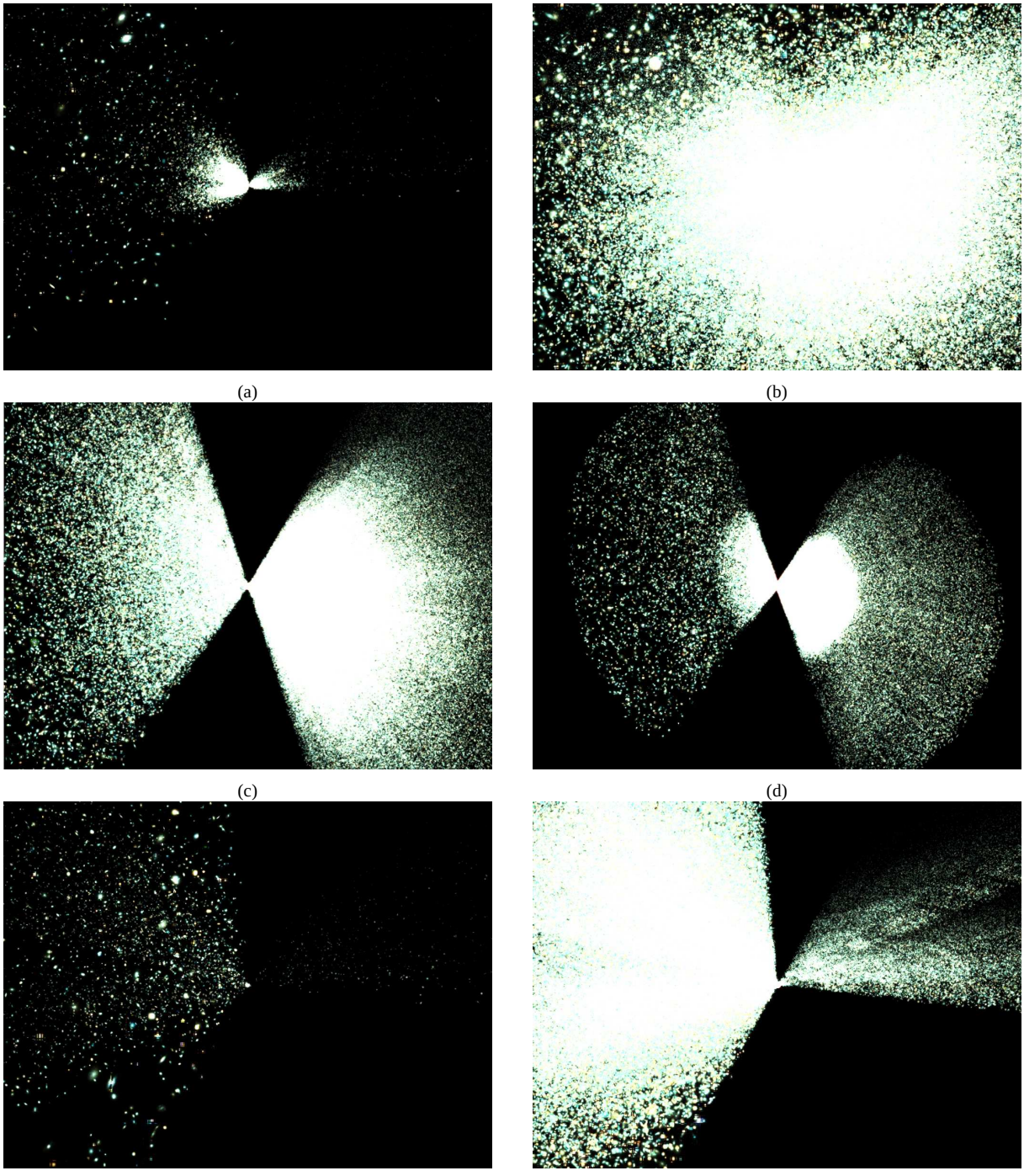
[14] M. Subbarao, D. Surendran, and R. Landsberg. SDSS Galaxy Fly-through. `http://astro.uchicago. edu/cosmus/projects/sloangalaxies/` [Online; accessed 20-09-2007].

[15] I. Taylor, M. Shields, I. Wang, and R. Philp. Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 16.1, Washington, DC, USA, 2003. IEEE Computer Society.

[16] T. van der Schaaf, M. Koutek, and H. Bal. Parallel Particle Rendering: a Performance Comparison between Chromium and Aura. In *Eurographics Symposium on Parallel Graphics and Visualization Proceedings*, pages 137–144, 2006.

[17] S. Weinberg. *Gravitation and Cosmology : Principles and Applications of the General Theory of Relativity*. Wiley, July 1972.

[18] E. L. Wright. A Cosmology Calculator for the World Wide Web. *ArXiv Astrophysics e-prints*, 118:1711–1715, Dec. 2006.



(a) (b) (c) (d) (e) (f)

Figure 6: Rendering results of the parallel particle visualization. The whole SDSS DR6 data set was rendered. The images were downsampled to 8-bit per channel color depth and downscaled to $32 \times 32$. (a) Close-up of the center of the data set. (b) Seeing through the center from greater distance. (c) Large-scale structure of the data. (d) Large-scale structure from greater distance. (e) Rendering 1 percent of the data set from a spectacular view. (f) Rendering the whole data set from the same position.