Large-Scale Structure in the Universe

Technical Report CS03-16-00
Department of Computer Science
University of Cape Town

Carl Hultquist, Sameshan Perumal, Patrick Marais and Tony Fairall

Abstract

Cosmologists are currently researching the theory of *large-scale structures*, which are in essence groups of neighbouring galaxies. Recent “galaxy-hunts” have resulted in data for hundreds of thousands of galaxies being made publicly available, and it has become infeasible to isolate large-scale structures by hand from this data. Furthermore, it is difficult for cosmologists to visualise such structures by simply observing the galaxies that comprise the structure; they need a graphically rendered system in which they can change their viewpoint and observe the structure from any position desired.

We present a system identifies large-scale structures from datasets of galaxy information, and then displays the data using OpenGL in such a way that the user can “fly” through space in realtime, observing not only a single structure but the entire dataset and how structures are positioned relative to each other.

1 Introduction

A new theory is being developed by cosmologists that is hoped to provide an explanation about how the universe has developed and how it will continue to develop. This involves what are known as *large-scale structures*, which are simply sets of neighbouring galaxies that have an impact on each other[1]. Cosmologists are interested in the topology of these structures, and in the overall topology that these structures impose on the universe[2].

Until recently, there has been very little data on galaxies that allows cosmologists to accurately determine clusters; and the data that has been available has been sufficiently small to allow for structures to be identified by hand. However, massive “galaxy hunts” in the past few years have culminated this year[3] to produce vast galaxy databases. Most notably the Sloan Digitial Sky Survey [Kron et al. 2003] and 2dF Redshift Survey [Peterson et al. 2001] contain information pertaining to approximately 133988 and 102235 galaxies respectively.

These large datasets are cumbersome to process by hand, and a means of easily identifying and visualising large-scale structures using a computer is desirable. We propose such a system, which identifies large-scale structures from sets of galaxy data and allows the user to “fly” through the space, visualising the structures from any viewpoint. Furthermore, we make use of a novel and cheap method of achieving depth perception, allowing objects to appear at different distances from the user for a more realistic interaction.

[1]Typically by means of gravitational forces

[2]That is, the overall pattern — or lack thereof — inherent in large-scale structures

[3]2003

2 Background

The work required to produce such a system falls into two major categories, namely *structure identification* and *rendering and user interface*. As such, we consider related and background work in two distinct sections.

2.1 Structure identification

This section of the work serves as the data source for the user interface, and is responsible for processing galactic data in order to identify large-scale structures and generate visualisations of these in 3D. These visualisations are intended to assist astronomers in investigations into large-scale structures formed by galaxies - the work of [Humphreys and Bruce 1989] suggests that such visualisations aid understanding.

The crux of this section revolves around the technique of *sphere percolation* around galaxies to identify structures within galactic data. A large-scale structure can be identified using the technique presented in [Fairall 1997]: for each galaxy in the dataset, expand a sphere around its position. Expand the radius of all spheres until some maximum value is reached, or intersection with another sphere occurs. Each set of intersecting spheres then forms a structure, as indicated in Figure 1. A

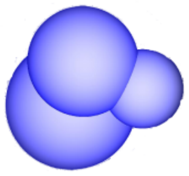


Figure 1: Graphical illustration of sphere percolation for structure identification.

problem with most galactic datasets is that data density decreases with distance from Earth. This is due to the interference caused by closer galaxies, which obscure those further out. A refinement to this scheme allows the percolation radius (maximal sphere radius) for a given galaxy to be individually set in order to compensate for this. This detection algorithm is vastly simplified through the use of a Minimum Spanning Tree (MST) - in that case, only edges originating from a given galaxy need be considered to determine membership of neighbours in a structure.

A convex hull is an efficient, fast way to generate a surface that encloses a volumetric data set, such as that created by large-scale structure identification. A convex hull of a set of points S in 2D is formally defined as the intersection of all convex sets containing S [M. de Berg and Schwarzkopf 2000] [Day 1990]. Intuitively, it may be thought of as wrapping a rubber band around the point set - this idea is conveyed in Figure 2. Extrapolating this idea into 3D produces polygonal surfaces that contain all points in a volume. Although fast and efficient, a weak point of this technique is that is unable to correctly handle cavities or concave surfaces. Hence, the **G** in Fig. 2 looks like an ellipsoid.

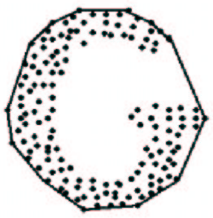


Figure 2: A 2-dimensional convex hull of a set of points

A much better solution uses a volume specific surface extraction technique like Marching Cubes [Lorensen and Cline 1987]. It is specialised for volumetric data, and is able to handle concavities. However, it is computationally expensive, and as such may not always be appropriate. One other problem is that point data must be converted to volumetric data in some form to achieve this. Many methods exist to do this, but an interesting solution uses Constructive Solid Geometry (CSG) [Requicha 1980] to obtain a solid object that provides this volume information. CSG uses simple objects together with the boolean difference($-$), intersection($\cap$) and union($\cup$) operators to define complex solid objects. In the case of this work, a typical CSG result would be similar to Figure 1.

2.2 Rendering and user interface

The rendering portion of our system is reponsible for rasterising galaxies and large-scale structures in an efficient manner. As such, considerations such as LOD (level-of-detail), billboarding and user-defined hardware programs needed to be addressed. A system that provides a similar tool showing visualisations of galaxies and other celestial phenomena is Partiview [American Museum of Natural History 2003a], provided by the Hayden Planetarium [American Museum of Natural History 2003b]. To our knowledge, Partiview does not, however, identify and display large-scale structures.

LOD methods provide an effective means of increasing efficiency by simply rendering objects with less detail as they move further away from the viewer. In this way, less geometry is sent to the hardware for objects that contribute less to the overall view, without actually sacrificing detail[4]. An example of various levels of detail can be seen in Figure 3. This is an extensively well researched topic, and details of various approaches can be found in [Clark 1976], [Funkhouser and Séquin 1993], [Hoppe 1996], [Garland and Heckbert 1997] and [Southern and Gain 2002].

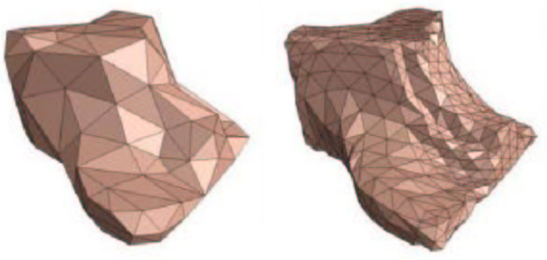


Figure 3: An example of varying LOD in a model. The figure on the left shows a simplified form of the model on the right.

Billboarding [McReynolds et al. 1999] is a useful method

[4]Due to the object being further away, it is rendered with fewer pixels and hence can be represented using fewer polygons to give the same detail that is visible

of realistically rendering an object using a single, textured polygon. The process simply calls for the polygon to be oriented in such a way so that it directly faces the viewer (see Figure 4). Such a technique is useful in providing a realistic rendering of galaxies.

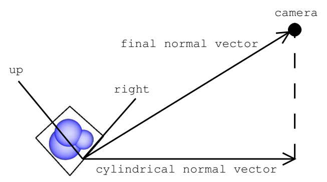


Figure 4: An illustration of the billboarding process. The polygon representing the object is simply rotated so that it faces the viewer.

Recent graphics hardware[5] allows for user-specified programs to be run efficiently on the hardware. This provides a means to dynamically adjust either vertex data (before rasterisation) or resulting pixel colours (after rasterisation). Such techniques can often be used to bypass certain features of the standard graphics pipeline that are not utilised by an application, or to develop more complex changes that can be executed more quickly on the graphics hardware than on the CPU. With OpenGL, such enhancements are available through *extensions*, of which there are several that are particularly useful and worth considering for use[6].

Triangle strips are yet another means for increasing rendering efficiency. These are natively supported by OpenGL, and can have a maximum improvement of 67% ($N+2$ vertices sent to graphics hardware as opposed to $3N$).[7] Generating good triangle strips is an ongoing area of research, and some methods can be found in [Evans et al. 1996], [El-Sana et al. 1999] and [Stewart 2001].

In addition to efficient rendering, we have also considered the problem of an effective user interface. In particular, defining an intuitive interface for the user to "fly" through space is difficult. Many modern three-dimensional graphical interfaces have a notion of gravity with associated meanings for "up" and "down", and often the interface exploits this fact. In space, however, there is no such grounding, and so these interfaces are not suitable. [Shoemake 1992] and [Chen et al. 1988] describe some popular techniques for interfaces that use a mouse.

Finally, we have explored means for conveying depth information to the user, allowing them to see in "true 3D" and perceive the distance of objects from the viewpoint. [Steenblik 1987] describes how a method called *chromastereoscopy* can be used to sacrifice colour for depth information. [Cruz-Neira et al. 1993] describe another method using polarised shutter glasses that are synchronised with the display unit so as to display a different image to each eye.

3 Approach

The system that we have designed provides an efficient and dynamic solution, both in terms of structure identification, rendering and user-interface.

An important part of indentifying structures is standardising astronomical coordinates, which use velocity to indicate position. Additionally, the creation of an MST significantly simplifies the task of identifying structures. Since galactic data is for all intents static, the above two aspects have been addressed using offline pre-processing for a given dataset, in the form of a cross-platform wxWindows GUI application [Smart et al. n. d.]. For the creation of the MST, Prim's algorithm [Prim 1957] was chosen due to the maximally dense nature of our graph (every galaxy is potentially connected to every other galaxy). Kruskal's algorithm [Kruskal 1956] was innappropriate since all $N(N-1)$ edges have to computed, sorted and stored, which is infeasible for large datasets. Prim's allows computation of edges on demand, without requiring storage.

The Marching Cubes technique mentioned previously was not used, since it is inappropriate to the data we are working with. After investigations, it proved infeasible to manually code the algorithm, and most other implementations expect data in the form of volumetric slices, such as those obtained from medical imaging. Various alternative solutions were then investigated, including voxel representations and a custom surface fitting technique.

The choice to use the Visualisation ToolKit [VTK 2003] to achieve a similar effect stems from the maturity, stability and appropriateness of the code available. Additionally, VTK provides built-in support for CSG to create the volumetric data used by the Contouring filter, which creates a surface around volumetric data.One shortcoming of VTK is that it is intended as a complete solution, from data source to rendering, and as such provides no mechanism whereby data can be directly extracted by an external application - specific code had to be written to address this problem.

With regards to structure visualisation, three methods

[5]For example, nVIDIA's GeForce3 and better

[6]Namely `ARB_multitexture`, `ARB_vertex_program`, `ARB_vertex_buffer_object` and `NV_register_combiners`

[7]Where N is the number of triangles in the mesh being rendered

have been implemented. They address the issues of speed and accuracy by attempting to achieve a balance. The first is very fast, but lacks adequate detail for useful investigation by end users. The last is significantly slower, but provides very detailed results, that are much more useful in investigations. These options are:

- **Wire-tree visualisation of structure MST:** This basically entails providing a means to render the edges and vertices of the structures local MST, using lines and points. This method has numerous advantages: it is incredibly quick to create; it provides a good overall view of the form of the structure; it is implicitly supported by the existence of the local structure MST. This visualisation is not suitable for investigations into the forms of structures since it does not convey any volume information, and in particular, is unable to reproduce the cavities and doughnut-like shapes that are possible using the sphere percolation technique.
- **Convex-Hull:** This visualisation represents a middle ground between the above technique and Contouring. Essentially, it should be possible to wrap a 3D convex hull around the points in the structure to capture some sense of the volume and extent of it. This technique still suffers from being unable to handle concavities or holes in the generated solid. It is good for quickly locating identified structures to verify positional and approximate volumetric requirements of the user.
- **Contouring:** Uses the vtkContour filter, to produce a surface around the structure that respects concavities and holes in the solid. This visualisation suffers from slow execution time, and as such could violate the interactivity requirement. However, the added detail gained is invaluable, hence its inclusion. A further advantage is that the sampling detail can be adjusted, allowing for simple, quick approximations to be generated. The quality of the visualisation can subsequently be refined to produce complicated, detailed structures. This flexibility greatly increases the interactivity level (at low detail) without diminishing the effectiveness and accuracy of the results (at high detail).

The Contouring approach is volume based, and as such is an $O(n^3)$ algorithm[8], hence there is a large delay associated with execution of the Contour Filter with high qulaity settings. A mechanism was therefore added to allow structures and their visualisations to be saved, thereby reducing most computation to a single, intensive run which need not be subsequently repeated.

[8] n here refers to the grid density, which can be altered dependent on requirements

The user interface allows for a variety of rendering techniques to be used, allowing the user to tune settings according to their individual system setup. The following rendering options are supported:

- **Various methods for rendering galaxies**. Galaxies can be rendered as either points (using `GL_POINTS`) or as billboards, or not at all. In the case of billboards, a selection of images of galaxies have been chosen to randomly texture the galaxies.
- **Support for `ARB_vertex_buffer_object` extension**. This allows vertex data to be stored in the memory of the graphics hardware, resulting in significant speedups when actually rendering.
- **Support for `ARB_vertex_program` extension**. This is used in conjunction with billboarding, and allows the computation of billboard rotations to be performed as a vertex program on the graphics hardware.
- **Discrete LOD of large-scale structures**. As the viewer approaches large-scale structures, the detail with which the structures are rendered increases[9]. Conversely, as the viewer moves away from a structure, it becomes "smaller"[10] and hence requires fewer polygons to achieve the same detail. We have implemented this using a preset number of discrete LOD's, and the LOD to use is decided based on the viewer's distance from the centroid of the structure.
- **Per-pixel shading of large-scale structures**. Anomalies with the standard OpenGL Gouraud shading were noted, whereby placing a diffuse light near the centroid of a polygon would result in the polyon being poorly lit (whereas, in fact, the interior — near the centroid — should have been well lit). This is simply a result of the use of Gouraud shading. To overcome this — and hence provide more realistic results — we implemented a simple per-pixel diffuse lighting scheme that makes use of the `NV_register_combiners` extension.
- **Chromastereoscopy**. For further realism, we have implemented *chromastereoscopy* support for both galaxies and large-scale structures. This allows a pair of inexpensive glasses to be used which adjust the focal length of varying colours of light, making reds appear close, greens further away, and blues the furthest away. Thus the user can obtain a real sense of distance with regard to the placement of galaxies and structures relative to their viewing position.

[9] That is, the number of polygons used to draw the structure is increased

[10] In terms of the screen-space that it takes up

"Flying" has been implemented using a new system (see Figure 5), whereby three vectors are necessary to uniquely determine the user's position and view. Firstly, a displacement vector $\boldsymbol{P}$ is used to define the user's position in space[11]. A *viewing* vector, $\boldsymbol{V}$, then defines the *direction* in which the user is looking and finally an *up* vector, $\boldsymbol{U}$, defines the direction currently corresponding to "up" for the user[12].

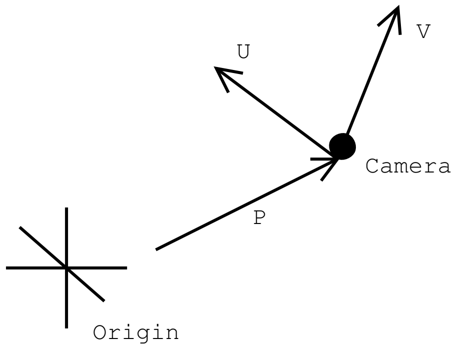


Figure 5: Our approach to motion calculations. Three vectors ($\boldsymbol{P}$, $\boldsymbol{V}$ and $\boldsymbol{U}$) are used to encode the user's position and viewing direction. $\boldsymbol{P}$ maintains their position relative to the origin, $\boldsymbol{V}$ stores the direction in which they are viewing the scene, and $\boldsymbol{U}$ stores the vector that is currently the user's sense of "up" (i.e. determines rotation about the $\boldsymbol{V}$ vector).

User interactions and their effects on $\boldsymbol{P}$, $\boldsymbol{V}$ and $\boldsymbol{U}$ are then defined as follows:

- **Look "up" and "down"**. $\boldsymbol{V}$ and $\boldsymbol{U}$ are adjusted by rotating them about $\boldsymbol{V} \times \boldsymbol{U}$. $\boldsymbol{P}$ remains unchanged. This is effected by moving the mouse forwards (up) and backwards (down).
- **Look left and right**. $\boldsymbol{V}$ is adjusted by rotating it about $\boldsymbol{U}$. $\boldsymbol{P}$ and $\boldsymbol{U}$ remain unchanged. This is effected by moving the mouse left and right.
- **Move forwards and backwards**. $\boldsymbol{P}$ is adjusted by "sliding" it along the vector $\boldsymbol{V}$. That is, $\boldsymbol{P_{new}} = \boldsymbol{P} + \alpha\boldsymbol{V}$. $\boldsymbol{V}$ and $\boldsymbol{U}$ remain unchanged. This is effected by holding down the left (forwards) and right (backwards) mouse buttons.
- **Strafe left and right**. $\boldsymbol{P}$ is adjusted by "sliding" it along the vector $\boldsymbol{V} \times \boldsymbol{U}$. That is, $\boldsymbol{P_{new}} = \boldsymbol{P} + \alpha(\boldsymbol{V} \times \boldsymbol{U})$. $\boldsymbol{V}$ and $\boldsymbol{U}$ remain unchanged. This is effected by pressing the left and right arrow keys on the keyboard.

[11]Relative to the origin

[12]That is, in screen co-ordinates $\boldsymbol{U}$ points towards the top of the screen

Furthermore, we have developed a cross-platform user interface using wxWindows [Smart et al. n. d.] and OpenGL [Silicon Graphics Inc. 1995]. This allows for an interface native to the platform on which the application is run to be used, enhancing the overall usability of the application.

4 Results

From the perspective of Structure Identification and Visualisation, the results are very encouraging. Expert users, represented by Dr. T. Fairall of the Astronomy department of UCT, defined the following as important goals that our system has achieved:

- Correctly identifying standard, recognised structures (eg. Virgo, Coma and Fornax clusters)
- Illustrating the *"finger of god"* effect apparent in most clusters.
- Conveying the *"foamy"* texture created by large-scale structures.

Coordinate transformations are working correctly - verified against accepted values, as well by visual comparison of resultant datasets to the actual night sky. The time taken to identify structures is acceptable, since on average approximately 10 comparisons must be made for each galaxy, which reduces the computational expense from $O(n^2)$ to $O(n)$. The most striking feature of identification is that most of our local large-scale structures (cz < 7500 km/s) which are identified correspond to previously established structures, as determined by astronomers.

The results of the visualisation of these structures is mixed, though still promising. The **wire-tree** representation, although simple, is very effective at conveying the overall form of a given structure. Additionally, there is almost no computational cost (edges are simply extracted from the global MST to form the local MST).

The **Convex Hull** representation is less effective - sphere like objects, with very little distinguising features seem to be characteristic of all visualisations of these types. This form is effective at quickly locating structures (much more visible than the wire-tree representation) and it does succeed in providing a good estimate of the total volume of the structure. This is important in altering the percolation radius to ensure that relevant galaxies are not omitted and irrelevant ones are excluded. Since convex hull generation is fairly quick, the use of this representation to fine-tune percolation parameters is an effective technique.

The Contour Filter and CSG combined representation proved to be very effective. The ability to selectively alter the level-of-detail[13] of the generated surface allows compromises between speed and accuracy to be made. The use of variably sized spheres, dependent on the distance to the furthest neighbour, greatly increases the quality of the visualisation, and is able to capture the sponge- or thread-like quality of large-scale structures. Additionally, the preservation of cavities allows empty areas of space within a structure to be easily determined.

Rendering efficiency is good, with a "typical" setup being able to run at approximately 30 FPS[14] on fairly common hardware[15]. Here, typical denotes the rendering of both large-scale structures and galaxies. Table 1 shows some configurations with more detailed results. High quality configurations, which render the best quality and most realistic results[16] run at a more moderate 10 to 12 FPS. All preparation optimisations — discrete LOD's, vertex programs, chromastereoscopy lookup tables, and hardware buffers — are generated extremely quickly, with the slowest being LOD generation at approximately 0.3 seconds per structure (for 15 levels of detail).

Furthermore, all rendering optimisations and structure identification settings can be modified dynamically by the user, allowing for performance to be adjusted according to the user's unique setup.

Finally, our new interface for motion in space compares favourably to other techniques. Feedback from one expert user was particularly positive, with the comment that our interface is preferred to that offered by [American Museum of Natural History 2003a].

Figures 6 and 7 show screenshots of the resulting application with structures and galaxies.

Configuration	**FPS**	**Verts/s**	**Tris/s**
Galaxies rendered as points, LOD optimisations for structures, OpenGL lighting, no chromastereoscopy	29.91	2241067	395859
Galaxies rendered as points, LOD optimisations for structures, OpenGL lighting, with chromastereoscopy	25.01	1873925	331007
Galaxies rendered as billboards, no structures rendered	24.99	3521538	1760769
Galaxies not rendered, structures rendered with no LOD optimisations and with OpenGL lighting	9.37	2372064	790688

Table 1: Some overall results of specific configurations tested. In particular, these results demonstrate that our use of chromastereoscopy does not have a large impact on rendering efficiency, and also that LOD optimisations cause a significant boost in rendering efficiency. changes.

5 Conclusion

In conclusion, we have achieved all of our goals by developing a system that:

1. Efficiently and correctly identifies large-scale structures.
2. Allows for quality of the resulting structures to be controlled by the user.
3. Efficiently renders galaxies and large-scale structures.
4. Allows the user to "fly" through space to obtain whatever view they desire.
5. Allows for rendering settings to be cutomised to suit the user's requirements.

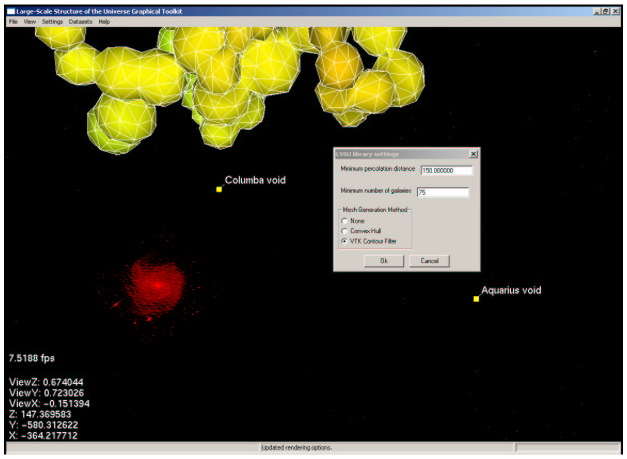
Our results also show that the system is both usable and efficient enough to operate on a common hardware setup. In particular, our numerical results in Table 1 show that the system runs suitably well on a modest hardware specification, and also that certain optimisations (such as discrete LOD usage) have a very favourable impact on performance. Coupled with the fact that rendering settings are customisable, our implementation is dynamic and can be adapted to work at high speed for the hardware that it is run on. This

[13]This a facility provided by VTK, and determines the granularity of the mesh on which the volume data is sampled.

[14]Frames per second

[15]Pentium 4 2GHz, GeForce 2 MX440, 512Mb RAM. Whilst 512Mb of RAM is fairly high, we do not believe that lower amounts will pose an issue. The average memory footprint of our system is approximately 30 to 40 Mb.

[16]Namely, using billboards for galaxies and chromastereoscopy for depth perception


Large-Scale Structure of the Universe Graphical Toolkit
File View Settings Datasets Help
Columba void
Aquarius void

LSSU library settings

Minimum percolation distance: 150.000000

Minimum number of galaxies: 75

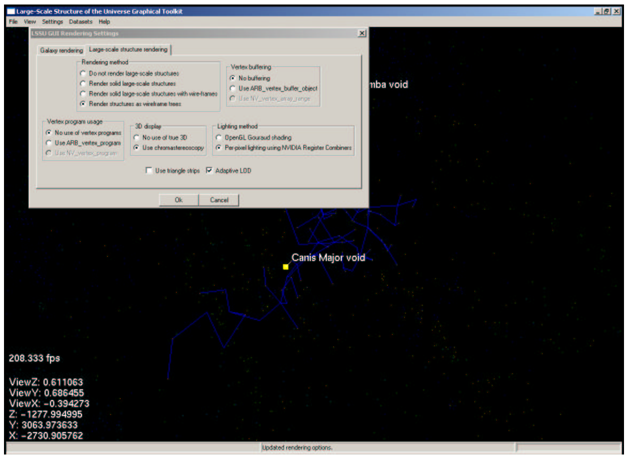
Mesh Generation Method: None / Convex Hull / [✓ VTK Contour Filter]

```
7.5188 fps

ViewZ: 0.674044
ViewY: 0.723026
ViewX: -0.151394
Z: 147.369583
Y: -580.312622
X: -364.217712
```

Updated rendering options.

Figure 6: A screenshot showing structures with triangle wireframes, a billboarded galaxy, and a dialog for library settings.


Large-Scale Structure of the Universe Graphical Toolkit
File View Settings Datasets Help
Galaxy rendering
Large-scale structure rendering
Ok
Cancel
mba void
Canis Major void

LSSU GUI Rendering Settings

Rendering method: Do not render large-scale structures / Render solid large-scale structures / Render solid large-scale structures with wire-frames / [✓ Render structures as wireframe trees]

Vertex buffering: [✓ No buffering] / Use ARB_vertex_buffer_object / Use NV_vertex_array_range

Vertex program usage: [✓ No use of vertex programs] / Use ARB_vertex_program / Use NV_vertex_program

3D display: No use of true 3D / [✓ Use chromastereoscopy]

Lighting method: OpenGL Gouraud shading / [✓ Per-pixel lighting using NVIDIA Register Combiners]

Use triangle strips [✓ Adaptive LOD]

```
208.333 fps

ViewZ: 0.611063
ViewY: 0.686455
ViewX: -0.394273
Z: -1277.994995
Y: 3063.973633
X: -2730.905762
```

Updated rendering options.

Figure 7: A screenshot of an underlying minimum spanning tree representation of a large-scale structure, with a dialog for adjusting rendering settings.

concept is further supported in our structure identification process by allowing the user to alter the quality and algorithm used to identify structures (as mentioned in Section 4), offering a completely customisable system that the user can adjust to suit their requirements.

In addition, we have developed a novel interface for the user to navigate around the universe, and have added chromastereoscopy which allows for a more realistic experience by providing depth information to the user.

6 Future work

There are still a number of features that could be extended in this project. We have identified the following key extensions:

- **User-defined dataset modification**. Our current system generates a datafile as a pre-process to generate data necessary for structure identification. Whilst the user can use this technique to generate their own datafiles, this is separate from our main solution and also does not cater for interactive changes.

- **Enhanced rendering**. Our most recent implementation does not display a large-scale structure when the user is inside it. This is both for rendering efficiency purposes and to allow the user to still see all the data outside of this structure. One means of improving this would be to make the structure which the user is inside slightly transparent, allowing them to perceive both the structure and the data outside of it. Continuous LOD, with a structure such as a progressive mesh, would also be beneficial to overall appearance and would avoid "popping" of structures as the user approaches or moves away from them.

- **More efficient rendering**. For example, the use of efficient triangle strips, culling of entire structures (that lie behind the user), and the incorporation of newer graphics hardware features.

- **True 3D**. Whilst chromastereoscopy provides a cheap and easy means of providing depth information, one has to sacrifice colour in the process. Incorporating some other means of 3D (such as the use of stereoscopic pairs or a shutter-glass system) would allow for a more realistic experience.

- **Improved galaxy rendering**. Since our solution was developed specifically for the study of large-scale structures, little attention was given to the rendering of galaxies. Consequently, our galaxies are not completely realistic and, in particular, the small range of textures used can cause galaxies to appear the same. A more realistic means of rendering galaxies — that actually takes the galaxy's real look into account, say — could produce a more compelling visualisation.

References

American Museum of Natural History, 2003. Partiview. `http://haydenplanetarium.org`.

American Museum of Natural History, 2003. The Hayden Planetarium. `http://www.haydenplanetarium.org`.

Chen, M., Mountford, S. J., and Sellen, A. 1988. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, 121–129.

Clark, J. H. 1976. Hierarchical geometric models for visible-surface algorithms. In *Proceedings of the 3rd annual conference on Computer graphics and interactive techniques*, ACM Press, 267–267.

Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. 1993. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, 135–142.

Day, A. M. 1990. The implementation of an algorithm to find the convex hull of a set of three-dimensional points. *ACM Transactions on Graphics (TOG) 9*, 1, 105–132.

El-Sana, J., Azanli, E., and Varshney, A., 1999. Skip Strips: Maintaining Triangle Strips for View-dependent Rendering. IEEE Visualisation '99.

Evans, F., Skiena, S. S., and Varshney, A. 1996. Optimizing triangle strips for fast rendering. In *IEEE Visualization '96*, R. Yagel and G. M. Nielson, Eds., 319–326.

Fairall, A. 1997. *Large-Scale Structures in the Universe.* No. ISBN 0-471-96252-X. Wiley-Praxis.

Funkhouser, T. A., and Séquin, C. H. 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics 27*, Annual Conference Series, 247–254.

Garland, M., and Heckbert, P. S. 1997. Surface simplification using quadric error metrics. *Computer Graphics 31*, Annual Conference Series, 209–216.

Hoppe, H. 1996. Progressive meshes. *Computer Graphics 30*, Annual Conference Series, 99–108.

Humphreys, G. W., and Bruce, V., 1989. Visual cognition: Computational, experimental, and neuropsychological perspectives. Lawrence Erlbaum Associates, Hove.

Kron, R., et al., 2003. Sloan Digital Sky Survey. `http://www.sdss.org`.

Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, vol. 7, 48–50.

Lorensen, W. E., and Cline, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, 163–169.

M. de Berg, M. van Kreveld, M. O., and Schwarzkopf, O. 2000. *Computational Geometry: Algorithms and Applications.* No. ISBN: 3-540-65620-0. Springer-Verlag.

McReynolds, T., Blythe, D., Grantham, B., and Nelson, S. 1999. *SIGGRAPH '99 Advanced Graphics Programming Techniques Using OpenGL (course notes).*

Peterson, B., et al., 2001. The 2dF Galaxy Redshift Survey. `http://www.mso.anu.edu.au/2dFGRS/Public`.

Prim, R. C. 1957. Shortest connection networks and some generalizations. In *Bell System Technical Journal*, vol. 36, 1389–1401.

Requicha, A. G. 1980. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys (CSUR) 12*, 4, 437–464.

Shoemake, K. 1992. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92*, 151–156.

Silicon Graphics Inc., 1995. OpenGL - High Performance 2D/3D Graphics. `http://www.opengl.org`.

Smart, J., et al. wxWindows Home. `http://www.wxwindows.org`.

Southern, R., and Gain, J. 2002. *Creation and Control of Real-time Continuous Level of Detail on Programmable Graphics Hardware.* The Eurographic Association and Blackwell Publishers.

Steenblik, R. 1987. The Chromastereoscopic Process: a Novel Single Image Stereoscopic Process. In *Proceedings of SPIE: True 3D Imaging Techniques and Display Technologies.*

Stewart, A. J. 2001. Tunneling for triangle strips in continuous level-of-detail meshes. In *Proceedings of Graphics Interface*, B. Watson and J. W. Buchanan, Eds., 91–100.

VTK, 2003. The visualisation toolkit. http://www.vtk.org.