Data Visualization Using Hardware Accelerated Spline Interpolation

Petr Kadlec
kadlecp2@fel.cvut.cz

Marek Gayer
xgayer@fel.cvut.cz

Pavel Slavík
slavik@fel.cvut.cz

Czech Technical University
Department of Computer Science and Engineering
Karlovo náměstí 13
121 35, Praha 2, Czech Republic

ABSTRACT
We present our method of real-time rendering of grid-structured data. The method uses bicubic spline interpolation running on common current graphics hardware to map the values to a texture. It allows us to render isolines of the visualized data and dynamically change the visualization parameters. We have implemented the method and incorporated it into a system of real-time simulation and visualization of combustion processes in pulverized coal boilers. We have compared the visual quality of produced images with the previously used visualization methods.

Keywords
Real-time visualization, vertex and fragment programs (shaders), hardware accelerated rendering, splines, isolines.

1. INTRODUCTION
Many research projects and applications demand real-time simulation and visualization of various processes. Real-time visualization offers many significant advantages:

- The results are obtained quickly.
- The user can easily get an outline of the simulated process.
- It allows interaction with the simulation model.
- Changes in modeling parameters have immediate visualization response.
- Real-time visualization results are easily understandable and attractive, which is important in education and demonstration of the simulated processes.

Because the real-time visualization is clearly desirable, it is unfortunate that the traditional view regards the speed of the simulation as a contradiction to the visualization quality. In other words, we have to choose whether the CPU will spend time calculating the model, or displaying it. And if we do want to have an outstanding visualization, the speed of the simulation must be sacrificed. Luckily, today, we have an option – if we utilize capabilities of current graphics hardware, we may achieve both visualization speed and quality.

A typical example of a visualization method is a Cartesian grid containing values computed by the simulation. We want to display the simulated values in a way that helps us to see the most important features of the modeled situation.

Splines, isolines
The simulation (or whatever source for data we have) yields raw data values. We have to convert them to some visual attributes, usually color. This mapping can be direct, or indirect, using texture. Both methods work quite well when the grid values are almost linear. But even if this is the case, the way the rendered values are interpolated [Hec91a] may introduce artifacts into the resulting image. A typical example of such artifact is a triangular pattern, which occurs at those grid cells that are not “planar”, i.e. linear in both grid dimensions. In order to eliminate this weakness, we do not use the conventional linear interpolation; we apply spline (cubic polynomial) interpolation instead. Because we require such nonstandard mapping technique, we cannot use the

standard texture mapping equations that are implemented in graphics libraries and hardware. We would have only one option – to write a custom software renderer – in case we could not utilize modern graphics accelerators.

If we use a texture to convert the data values to color, we immediately have one more capability at hand – isolines.

One of the best methods to show up the features of a model in many applications is isoline drawing. This simple notion has an amazing effect for understanding the model. (That is probably the reason why the isolines are used so widely [Gil81a].) There are numerous algorithms to determine isolines of a value array [Kre96a], [Las92a]. Their main drawback is apparent: they are not designed to work in real time. Today's hardware may be able to perform similar algorithms in real time, but at the price of CPU computing power dedicated solely to a single visualization feature. If we are willing to exchange the accuracy of these methods for the performance, we can use simpler methods that offer only an approximation of the isolines, but with far smaller computation requirements.

Our method belongs to such category. We use a precomputed texture that is mapped to the displayed grid in such a way that the texture coordinate(s) correspond with the value stored at the respective points. In the resulting image, the isolines appear at those points where the interpolated texture coordinate(s) match the isoline value.

Again, if the texture mapping used would apply linear interpolation, the isolines would appear quite jagged. But as we can use the capabilities of modern graphics accelerators, we are able to perform the interpolation using splines.

Modern graphics hardware

Modern graphics hardware offers more processing power than the graphics hardware used in the past. More triangles per second, higher frame rates, more textures, higher resolution, … all these improvements are immediately available to any user that posses a modern graphics accelerator. The programs just run faster in the moment you replace the hardware. Although it is of course good to have more processing power available, this is not the most important effect of the advancements. The best innovation is the programmability of graphics accelerators. An accelerator now allows us to change its mode of operation in a variety of ways. We can change the transformation and lighting calculations done by the accelerator, and we can also change the texturing calculations. And whatever program we pass to the accelerator, it runs directly on the graphics processor, separately and simultaneously with the CPU.

So that by making use of these features, we can write custom texture mapping program for graphics hardware, performing spline interpolation instead of linear and send the program into the accelerator to be run on it. Using this interpolation, the isolines rendered by the aforementioned method are (although still equally fast and simple) far superior to the isolines rendered using the linear interpolation. The same is true about the overall quality of the rendered grid. All features of the image are smooth and the blocky appearance caused by linear interpolation of non-linear data has gone.

2. OUR WORK

In the process of rendering grid-structured data, we have to answer three questions:

- How should we convert the raw data into some form that is visually representable,
- How to interpolate the values between the grid points,
- How to render the resulting data with both speed and quality.

Our method addresses all these issues: The conversion from the raw data into visualizable parameters is done using textures. When the visualized data represents one-dimensional values (like temperature), the texture used is just one-dimensional (we could use two- or three-dimensional textures to visualize structured data). Data values in the grid are interpolated using spline (bicubic) patches. This interpolation is simple enough to be very fast, and still results in high quality images. The third topic is quite a problem for traditional methods that suffer from the contravening requirements of speed and quality.

A typical modern graphics accelerator contains a Graphics Processing Unit (GPU), which is a programmable processor capable of doing all computations required for traditional rendering. It is worth emphasizing that the GPU is a standalone hardware device that is capable of doing calculations completely independently of CPU. This results in a desirable new level of parallelism that allows us to relieve the CPU of some low-level repeated work and let it do the "hard" computations for the simulation.

The GPU is programmable in a number of ways: The two basic features are vertex programs and fragment programs (also called vertex and pixel shaders). A vertex program is a replacement for traditional graphics transformation and lighting calculations. A fragment program is a replacement for traditional texturing computations (see Fig. 1). We can use only vertex, or only fragment program, but most possibilities arise from using them in cooperation.

In general, the vertex and fragment programs can do any computation the GPU is capable of; but there are not only advantages (such as the added parallelism): the GPU has limited precision (this is not an issue for our use); the computation results are retrieved with difficulties (we display the results directly, so that we do not need to retrieve them explicitly); and the vertex (fragment) program must be executed for each vertex (fragment, i.e. generally said, each pixel), so that repeated calculations should be left to the CPU.

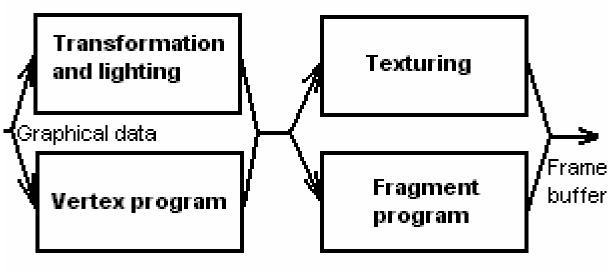

Figure 1. Vertex and fragment programs in the rendering pipeline

In our method, we use vertex and (optionally) fragment programs to do the spline interpolation and value-to-texture lookups. The basic rendering core looks like this:

```
for every grid cell do
    precompute spline coefficients
    for every pixel in cell do
        compute the spline value
        convert the value to color
```

The important point to note here is that the "for every pixel" loop executes entirely on the GPU.

We chose Catmull-Rom spline interpolation [Cat74a], because it is easily used to do interpolation on 2D Cartesian grid (creating a bicubic patch). The spline coefficients are computed only once for a whole grid cell, so this computation is done in the main program, on the CPU. These coefficients are sent to the vertex and/or the fragment program (as program local parameters), which do the actual interpolation (in addition to the traditional viewing transformation computations, etc.).

The interpolation may be done with varying precision: We can evaluate the spline patch at every pixel (using the fragment program), or only at selected points (using the vertex program) and then interpolate between these points only linearly (see fig. 2). This choice results in a tradeoff between the visualization quality and speed. If we decide to interpolate only at selected number of points, we can also choose the number of interpolation points. As the visualization proceeds in real-time, we may let the user choose the visualization parameters at run-time and change them dynamically. (Although for a single frame, the choice is fixed, i.e. we do not use any adaptive subdivision or similar techniques.) According to our experience (see section 4), the per-pixel interpolation is slightly slower, but the difference in quality is almost negligible, when the number of subdivision points is high enough.

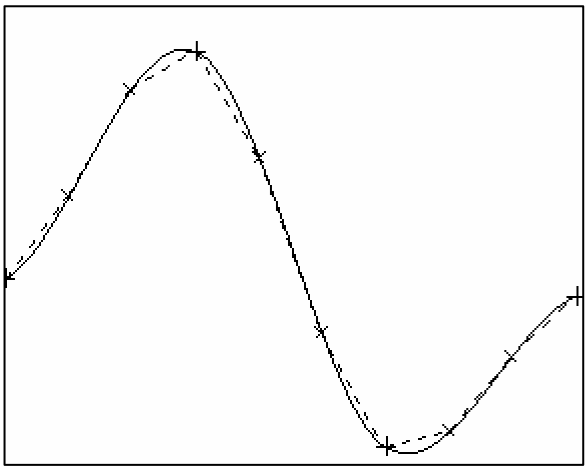

Figure 2. 1-D illustration of spline evaluation at discrete points

After the pixel value is determined using the spline interpolation, we convert it to color. For that purpose, we use a precalculated texture containing colors corresponding to the range of data values. By properly choosing the texture, we very simply achieve rendering of isolines (see Fig. 9). The basic idea is to convert chosen values to the isoline color (e.g. black) instead of the color normally corresponding to the value. This way, the isoline appears in the picture at each point where the interpolation yields the respective value. Because of the spline interpolation, the isolines are smooth. The advantage of the use of textures is also that texture mapping is very fast on current graphics hardware. We can also dynamically change the texture and thereby choose palette, isoline equidistance and other parameters in the way that suits the visualized data at the exact moment. The drawback of the isoline drawing method is that the isolines may be blurry in situations when gradient is small, and, in extreme (the whole data grid equal to the isoline value), the whole picture could get black (as one big "isoline").

3. IMPLEMENTATION

We have implemented the presented techniques as a library in the standard ANSI C++ language, using the OpenGL API. Because of this choice, our implementation may be used on a wide range of computer systems, as the OpenGL is a broadly supported industry standard. We have used GLUT as the windowing interface, because it is also supported on a majority of platforms. All these choices ensure that our implementation is easily portable to other systems.

Because the OpenGL standard has been specified some time ago, most of innovations in current graphics hardware are supported through use of OpenGL extensions [OGL03a]. OpenGL version 2.0 (which should contain many of these extensions as a part of the basic API) is expected in near time horizon, but it will probably not be widely supported from the beginning. So we decided to use various OpenGL extensions for interfacing the new technologies (such as vertex and fragment programs). The library uses extensions for vertex and fragment programs, vertex arrays, and point sprites. We implemented support for the standardized ARB extensions (ARB_vertex_program, ARB_fragment_program, etc.) and for nVidia versions (NV_vertex_program, NV_fragment_program, etc.) of these extensions. Our implementation is capable of choosing at run-time those extensions that are supported by the host system. The implementation is easily extensible to support any other extensions that have similar interface.

Hardware Support

The mentioned extensions are today supported on many low-cost graphics accelerators. The vertex programs are supported (in hardware) on the most of common graphics hardware sold today (e.g. even an old nVidia GeForce2MX has hardware support for vertex programs).

The fragment program hardware support has been not so broad in the past, but recently, new models of graphics cards have begun to support the fragment shaders also, even in the low-end models (e.g. nVidia GeForce FX5200).

It is theoretically possible to run the program on a system without hardware support for the technologies used. But as the method is designed for hardware support, it would make no sense, because all advantages of the method would be overruled by the emulation overhead.

4. APPLICATION AND TESTS

At the Czech Technical University at Prague, we are developing an application for real-time simulation and visualization of combustion processes. Currently, our system (called My Pulverized Coal Combustion – MyPCC) is based on simple Fluid Simulator [Gayer02], particle system and simplified combustion and heat transfer engine [Gayer03]. Until now, we were unable to obtain real-time, high quality visualization of about 40 various computed volume characteristics (such as temperatures, velocities and mass fluxes). For such visualization, we were using simple approach of linear color interpolation of quads, supported and rendered fast by current modern graphics accelerators.

Incorporating the spline-interpolation methodology to our application not only proved the idea and proposes the above-described methodology is correct. It also discovered and approved wide enhancement in the quality of graphics output compared to old original linear interpolation method. We also used the MyPCC application [Gayer03] for testing the performance of different visualization methods on various platforms with different overall and graphics performance.

Picture quality enhancements

The difference between only common linear interpolation version and the spline interpolation method is typically visible on every picture, even at the first sight (see Fig. 2 and Fig. 3). In every picture, the visualized areas come smoother, edginess of the margins between areas of passage of major value differences are avoided and the picture is more realistic and attractive to viewer.

Significant differences are being visible on places with large values differences, such as on Fig. 4 and Fig. 5. Probably the most significant enhancements are being visible when visualizing details of an area inside the boiler using higher zoom levels (see Fig. 6 and Fig. 7). Also, a dramatic gain in visual quality is achieved in cases, where (due to demand of high simulation speed) low-resolution grids – e.g. 20 to 40 cells – are used.

The overall visual quality enhancement in all cases is significant.

Performance discussion

The considerable advantage of our method is that it can be used on almost every today’s common graphics accelerator (which supports vertex shaders). Thus, the high quality visualization output can be achieved even on the old and very cheap GeForce2 MX family of cards. On today’s new graphics accelerators, with improved vertex and pixel shaders, we get even better performance.

The following Table 1 summarizes the overall performance measured in drawing frames per second (FPS) when visualizing the test grid consisting of 10000 cells using the above described methods. The tests have been run using resolution 1024 by 768 pixels in true color, full screen mode. No simulation or other visualization was running during the measurements.

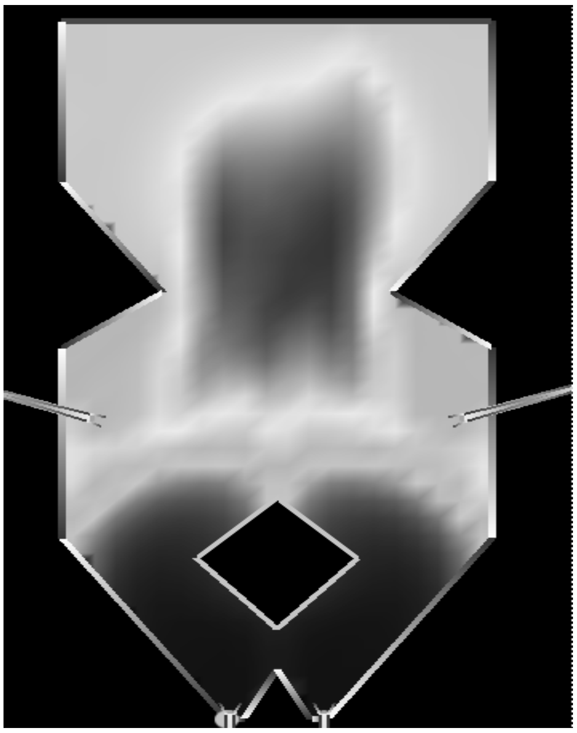

Figure 3. The original picture or boiler area temperatures, generated by linear interpolation of OpenGL quads

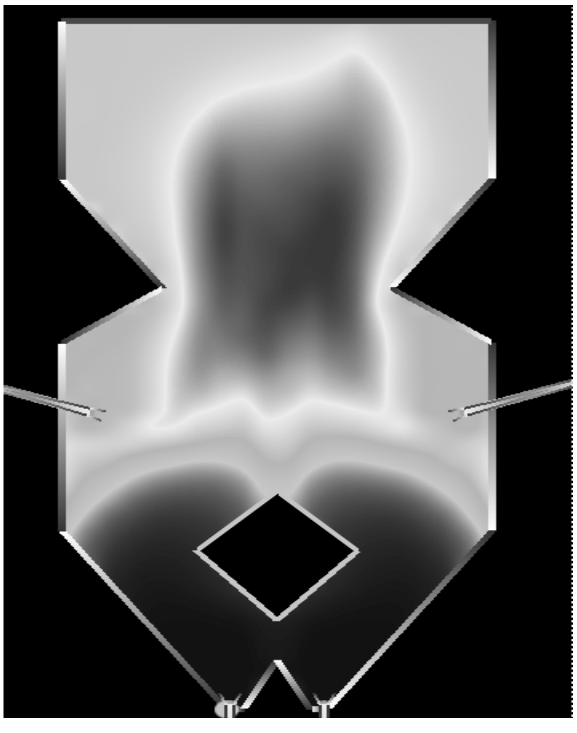

Figure 4. The visual enhancement is typically visible on every picture of the visualization of cell characteristics with any zoom level.

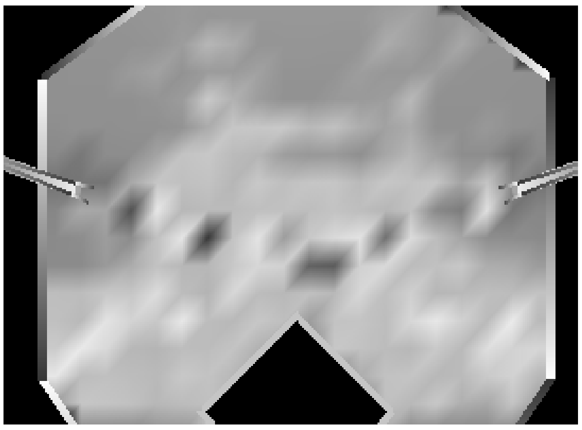

Figure 5. Original visualization of combustible masses inside the boiler area

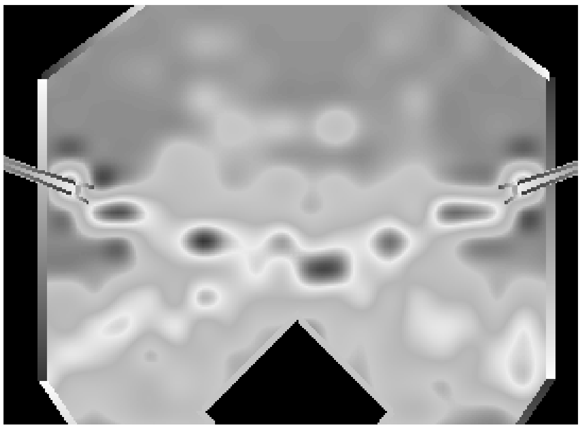

Figure 6. Visualization of the mass using the spline interpolation method

Figure 7. The original visualization output suffered considerably in picture quality when using high zoom levels

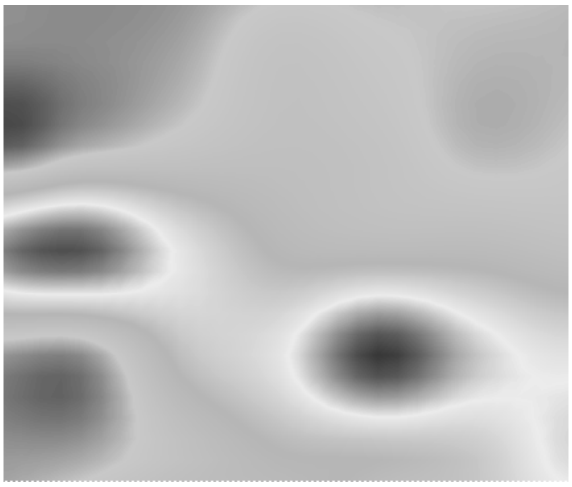

Figure 8. Even with high zoom levels, the spline interpolation method gives acceptable, realistic and attractive visual output

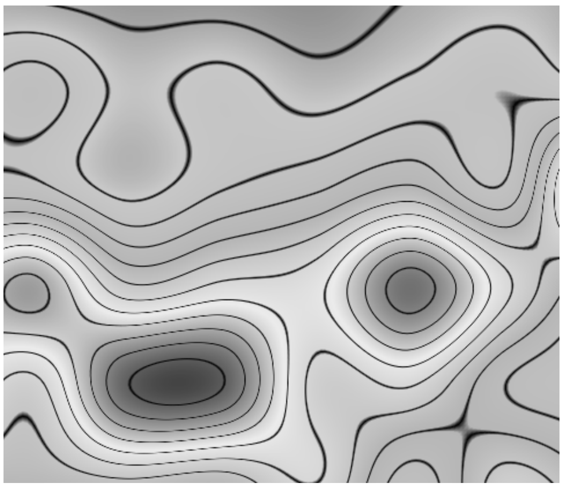

Figure 9. Using the pre-calculated texture palette concept, we can easily visualize isolines, with no additional performance cost over the basic spline interpolation.

The performance of original concept of real-time color linear interpolation (LI), maintained by drawing quads, and spline interpolation methods (SI) are being compared. The following systems were being used to gather results:

- GF2MX – nVidia GeForce 2 MX 440, AMD Athlon 1333 MHz, 133MHz FSB
- GF4TI4600 – nVidia GeForce 4 TI 4600, AMD Athlon XP 2000+, 333MHz FSB
- GF-FX – nVidia GeForce FX 5600, Intel Pentium 4, 2 GHz, 266MHz FSB

Although the performance of the spline interpolation does not seem to approach the simple color interpolation technique, it is not slower in orders of magnitude (with the exception of software pixel shader emulation needed for spline interpolation on GeForce 2 MX), and thus it is still suitable for real-time visualization. Our spline interpolation method gives results that are comparable to high-quality visual outputs of common systems, such as the well-known CFD package FLUENT 6 [Fluent03]. In contrast to this and other similarly based systems, our system is able to offer scalable performance to visualize tens of data frames per second with hardware-accelerated spline-interpolation.

System/FPS	LI Quads	SI Vertex	SI Pixel
GF2MX	57	27	< 1 *
GF4TI4600	95	56	20
GF-FX	148	110	68

** This card does not natively support pixel shaders. We had used software emulation (NV30 emulator) developed by nVidia to measure this feature instead.*

Table 1 - A comparison of graphical performance on different graphics cards.

Vertex shaders versus pixel shaders

We have implemented the spline interpolation with both vertex shaders (computing the spline value only at discrete points) and pixel shaders (computing the spline value at every pixel). After testing of visualizations of various characteristics and areas of boiler, we have realized that the gain in visual quality when using per-pixel interpolation in comparison with vertex-based interpolation is very low, while requesting significant performance. Thus, we recommend using vertex shader interpolation instead of pixel shaders as the default technique for accelerated spline interpolation.

5. CONCLUSIONS, FUTURE WORK

Our method achieves real-time rendering of grid-structured data using pre-calculated texture. We interpolate the data using spline interpolation that runs directly on the graphics accelerator, therefore leaving the CPU to compute the simulation in parallel. The texture used may be changed dynamically, which results in a possibility of interactive adjustments of visualization parameters. The method allows us to display isolines of the displayed data at no added cost. The isoline displaying may be enabled or disabled by simply choosing the texture.

We have implemented the techniques and used the implementation to improve quality of visualization in the "My Pulverized Coal Combustion" system. The success of the implementation proves the significant contribution of this hardware-accelerated technique for maintaining real-time, high-quality visualization of the cell characteristics. This concept can be used in similar applications regarding scientific, isocontour based visualization of computed or simulated data. Although the new method is somewhat slower than common simple visualization using linear interpolated quads, it is still suitable for real-time visualization, with dramatic enhancement of visual quality.

In the future, we plan to extend the method in the following ways:

- Although the method itself is in no way limited to 2D, all our tests were done in 2D. We intend to test the method in 3D visualization.
- We would like to test further interpolation methods, other than the Catmull-Rom interpolation.
- We plan to implement support for ATI versions of OpenGL extensions in order to support older ATI accelerators.
- A similar approach may be potentially used to visualize and render more-dimensional data, such as vector fields. We plan to test this approach and develop a method using vertex and fragment programs to display vector data.

6. REFERENCES

[Cat74a] E. Catmull, R. Rom, A class of local interpolating splines, Computer Aided Geometric Design, Academic Press, 1974

[Fluent03] Fluent, Inc. – Flow modeling software and services, Corporate website http://www.fluent.com/

[Gayer02] M. Gayer, P. Slavík, F. Hrdlička, Interactive System for Pulverized Coal Combustion Visualization with Fluid Simulator, Proceedings of the Visualization, Imaging, and Image Processing. Acta Press, Anaheim, pp. 288-293, 2002

[Gayer03] M.Gayer, P. Slavík, F.Hrdlička, Interactive Educational System for Coal Combustion Modeling in Power Plant Boilers. Proceedings Eurocon 2003 – Computer As a Tool. IEEE Slovenia Section, vol. 2, pp. 220-224, 2003

[Gil81a] P. P. Gilmartin, The interface of cognitive and psychophysical research in cartography, Cartographica 18(3): pp. 9–20, 1981

[Hec91a] Paul. S. Heckbert, Henry P. Moreton, Interpolation for Polygon Texture Mapping and Shading, State of the Art in Computer Graphics: Visualization and Modeling, Springer-Verlag, 1991

[Kre96a] M. van Kreveld, Efficient methods for isoline extraction from a TIN. Int. J. of GIS, 10, pp. 523–540, 1996

[Las92a] Michael J. Laszlo, Fast generation and display of iso-surface wireframes, CVGIP: Graphical Models and Image Processing, v.54 n.6, pp.473–483, 1992

[OGL03a] OpenGL® Extension Registry, Silicon Graphics, Inc. website, http://oss.sgi.com/projects/ogl-sample/registry/