# Adaptive Realtime Terrain Triangulation

Andreas Paul [*], Klaus Dobler [†]
Technische Universität München

**Abstract**

A system for realtime terrain triangulation is presented which allows fully user controllable and smooth level of detail transitions. To accomplish this as well as to offer the possibility of data compression, the terrain elevation data is hierarchically interpolated. The interpolation error, the so called hierarchical surplus, is used as a criterion to detect redundant data and serves as a way to control the error of the terrain approximation.

## 1 Introduction

The system is designed to be used by a flight guidance system applying computer generated synthetic vision for improving poor visibility guidance capabilities of aircraft. The basic idea of this guidance system is to offer a 3–dimensional computer generated pictorial representation of the terrain to the pilot, when the natural range of sight is reduced at night or due to bad weather conditions like fog or rain. (For further information see references [1], [2] and [3].)

In order to generate this synthetic vision of the surrounding terrain the first step is to determine the position and orientation of the aircraft using an integrated precision navigation system which couples data of a differential satellite navigation system (DGPS) and an inertial navigation system (INS) ([4]). Using this information in connection with a terrain database (DTED/DFAD) it is possible to generate the current out–the–window scene and present it to the pilot via a suitable display (Head Down, Head Up or Head Mounted Display).

The terrain databases available to us are DTED (Digital Terrain Elevation Data) and DFAD (Digital Feature Analysis Data). The part of DTED database

[*]Dipl.–Inform. Andreas Paul, Department for Engineering Applications in Computer Science and Numerical Programming
[†]Dipl.–Ing. Klaus Dobler, Institute of Flight Mechanics and Flight Control

currently used by our system consists of elevation data given in form of a regular grid with a mesh resolution of approximately 60m $\times$ 90m. Since we use a grid of $1025 \times 1025$ grid points the area covered is approximately 61.5km $\times$ 92.3km. The most detailed elevation data in DTED is given at a resolution of approximately 20m$\times$30m. The DFAD database contains information about areal, linear and point features. Areal features are given as polygons marking areas of certain surface qualities like water (=lakes), trees (=woods) or buildings (=towns). This kind of features is used to create a texture map that is drawn on the surface, thus coloring the elevation data from the DTED database. Linear features given as linestrips indicate objects like streets, rivers or railroad tracks. Finally point features mark the position of outstanding buildings such as churches, towers, power lines or bridges. The latter two kinds of feature are currently drawn in a separate pass from the terrain, therefore they do not directly relate to the problem of terrain triangulation and will not be further mentioned.

Given the intended use of the system as a realtime flight guidance aid certain design goals become obvious. First, since the system has to operate under realtime conditions a framerate of not less than 25 frames per second has to be achieved. The actual goal was set to be 30 frames per second. Next, the rendered terrain should be as close to reality as possible, in order to not alienate the pilot and to improve both his situational awareness and his guidance capabilities. The range of sight is set to be at least 25 km. To accomplish this at a framerate of 30 Hz, less important features of the landscape (=more distant features) have to be rendered at a lower level of detail than features very close to the aircraft. This level of detail has to be comfortably controllable, for example by setting an error margin $\varepsilon$ (given in meters of elevation) for features at various distances from the observer. Finally the system should include the possibility of compressing data as well as working directly with the compressed data in order to reduce disk space and memory requirements.

## 2 Procedure

To achieve these goals, especially data compression and the comfortably controllable level of detail, we decided to use a hierarchical interpolation approach for the terrain triangulation that was originally evaluated in [5] and is closely related to [6]. Other works dealing with hierarchical techniques for datacompression are

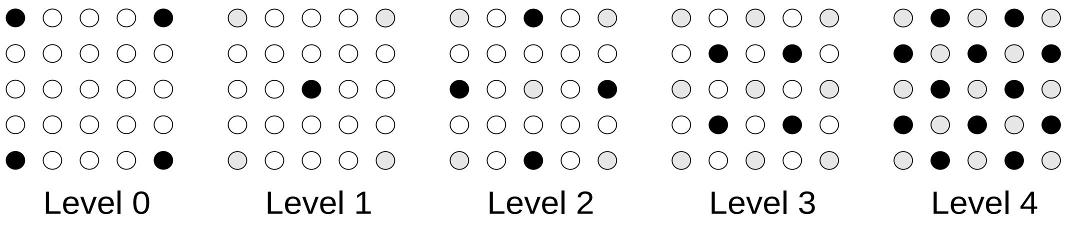


Figure 1: Hierarchical level of a $5 \times 5$ grid

[7], [8] or [9]. Unlike other methods for terrain triangulation as presentend e.g. in [10], our system is more suitable for realtime triangulation and comfortably controllable level of detail.

## 2.1 Hierarchical Interpolation

For a hierarchical interpolation of the data the regular grid of the DTED database is split into rectangular areas of $(2^n+1) \times (2^n+1)$ grid points. The next step is to implicitly subdivide those $(2^n+1) \times (2^n+1)$ grids into hierarchical levels (Figure 1 shows the levels for a $5 \times 5$ grid, hierarchical predecessors are marked grey).

Note that no actual splitting of the data is necessary, the access to the different level can be hardcoded into the algorithm that accesses the elevation data. This avoids the trouble of dealing with complicated, pointer based data structures such as lists or trees. Since the elevation data is stored in a linear memory segment cache and prefetch mechanisms usually used by workstations still have a chance to improve performance.

Of course hierarchical access to the twodimensional array means large jumps forward and backward within the memory which usually defeat every cache or prefetch strategies. But it is possible to reorganize the array in a hierarchical manner so that jumps only occur forward in memory, this way prefetch mechanisms can be reapplied. However this hierarchical organization of the array data has only been evaluated on a theoretical level.

Figures 2 and 3 show the immediate predecessor and successor relationship within the hierarchical structure.

In order to gain an indicator for the importance of each elevation value within the above described hierarchy, every value is interpolated linear of its hierarchical predecessors. Figure 4 shows which points are used to interpolate the values of each level.

The difference between interpolated and actual value, the hierarchical surplus

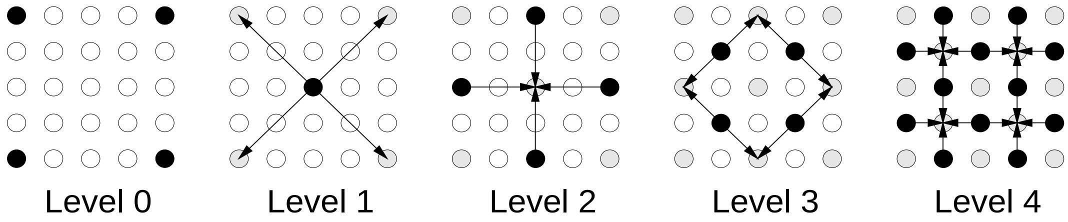


Figure 2: Hierarchical predecessors of each node

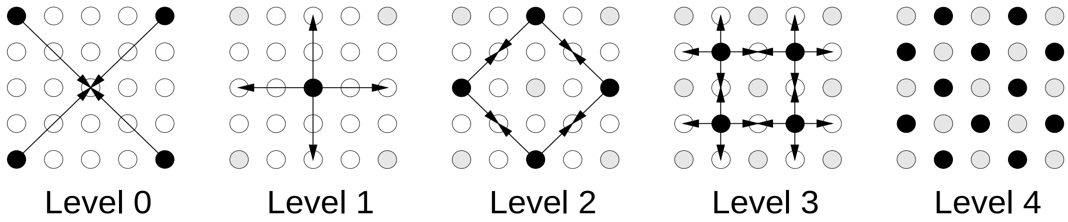


Figure 3: Hierarchical successor of each node

is stored for each node in a second array. This array of interpolation errors is then hierarchically traversed bottom–up level by level, storing the maximum of the currently examined node and its immediate hierarchical successors in each node. After that the highest interpolation error of the entire array can be found in the central node of the grid. Note that the computation of the interpolation errors and storing of the maximum of those errors can both be done in one pass at startup time.

## 2.2 Recursive Triangulation

One advantage and the most important reason why exactly this hierarchy was chosen is the fact that it allows a simple, straight forward, adaptive triangulation based on a classic divide&conquer scheme. The recursive algorithm used for the triangulation is basically as follows:

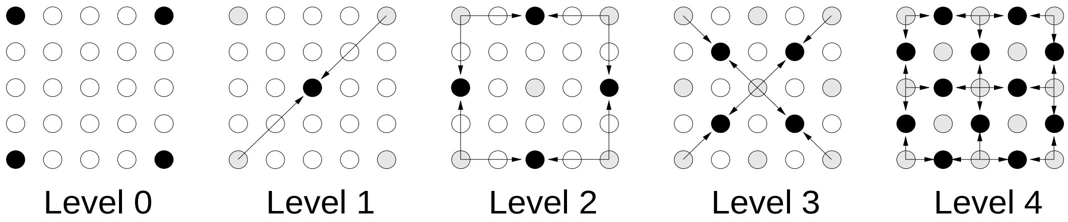


Figure 4: Predecessors of each node from which it is interpolated

```
examine_triangle(a,b,c)
{
    if(interpolation error at node (b+c)/2 < ε )
        draw_triangle(a,b,c);
    else {
        examine_triangle( (b+c)/2, a, b);
        examine_triangle( (b+c)/2, c, a);
    }
}
```

Figure 5 shows a sequence of triangles that are examined during an adaptive triangulation of a 5×5 grid. The function `examine_triangle()` is called two times. Once for the triangle between the northwest, northeast and the southwest corner of the array and once for the triangle of the southeast, southwest and northeast corner. Black grid points mark nodes that contain interpolation errors higher than a given error margin $\varepsilon$, white nodes contain interpolation errors below $\varepsilon$. Nodes that are examined at a certain level are marked by a circle. Figure 6 shows the resulting triangulation of the entire array. In this example the error margin $\varepsilon$ does not depend on the distance from the observer and is the same for all points.

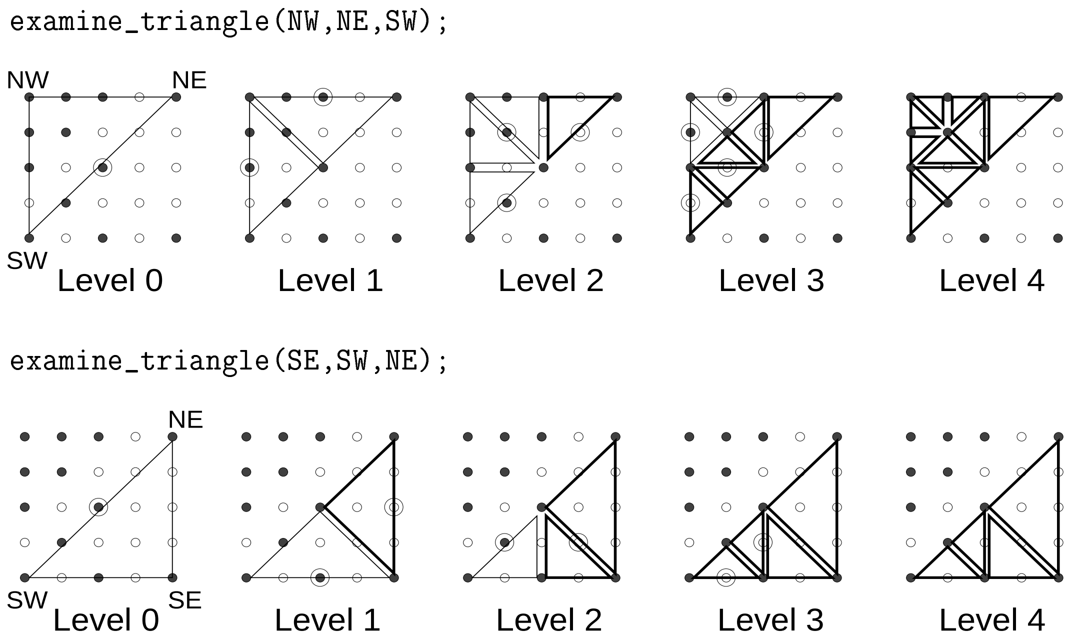


Figure 5: Sequence of triangles examined during a triangulation

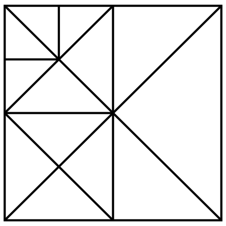

Figure 6: the resulting triangulation of the grid

In order to apply this triangulation scheme in our synthetic vision environment it is useful to supplement the function `examine_triangle()` with a termination rule that checks whether the currently examined triangle is within the observer's field of view. This way superfluous work can be avoided and most of the recursive function calls can be terminated at an early stage.

The method used to determine if a triangle is within the field of view is derived from a 3D–version of the well known Cohen–Sutherland line clipping algorithm ([11]) applied to triangles instead of lines. It requires a $4 \times 4$–matrix–vector multiplication for each examined node during the triangulation. These matrix–vector multiplications are the main computational overhead occurring at runtime. The alternative, not to check for visibility, would result in a much more significant overhead of work examining invisible triangles.

The next thing necessary for our synthetic vision environment is that the error margin $\varepsilon$ depends on the distance between the examined triangle and the observer. In order to achieve this $\varepsilon$ is simply replaced by a function `eps()` that is designed to depend on the distance between the observer and a gridpoint. The `if`–statement in `examine_triangle()` changes as follows:

```
if(error at node (b+c)/2 < eps( distance_to( (b+c)/2 )))
```

The intended consequence of this change is that different error margins will be applied to the triangles examined by the recursive calls to `examine_triangle()`. This means it is now possible that one triangle is recursively traced to a deeper level than the other. This situation will lead to T–junctions in the triangulation which result in so called *vertical holes* in the landscape. One simple way to avoid this is to mark the nodes of each examined and drawn triangle in a bitfield corresponding to the array of elevation data and supplement the `if`–statement in `examine_triangle()` as follows:

```
    if( error at node (b+c)/2 < eps( distance_to( (b+c)/2 ) )
        && node (b+c)/2 is not marked in the bitfield )
```

In order for this approach to work, the triangle that will be traced to a deeper level has to be examined first. So the problem now is to decide which of the triangles ((b+c)/2, a, b) and ((b+c)/2, c, a) to examine first.

We have not yet found a simple and save way to decide how deep a triangle will be traced. Several mechanisms were developed, but all of them showed to lead to vertical holes in the terrain.

To completely avoid the problem of vertical holes we have decided to use a two–pass approach for the triangulation. During the first pass the nodes of the triangles are marked in a bitfield instead of being drawn. The drawing takes place in a second pass where only the bitfield is examined. The two–pass version of `examine_triangle()`, that also incorporates the check for visibility follows:

```
examine_and_mark_triangle(a,b,c)
{
    if( triangle_is_not_visible(a,b,c) ) return();
    if( error at node (b+c)/2 < eps(distance_to((b+c)/2)))
        mark_nodes(a,b,c);
    else {
        examine_and_mark_triangle( (b+c)/2, a, b);
        examine_and_mark_triangle( (b+c)/2, c, a);
        }
}
examine_and_draw_triangle(a,b,c)
{
    if(node (b+c)/2 is not marked )
        draw_triangle(a,b,c);
    else {
        examine_and_draw_triangle( (b+c)/2, a, b);
        examine_and_draw_triangle( (b+c)/2, c, a);
        }
}
```

Some examples of what a useful `eps()`–function might look like are shown in figure 7. The left function is the one we currently use. Here the admissible error margin is zero within a distance of 1-5 km from the observer, which results in an exact representation of the terrain within that range. After that the margin of error increases linearly to a value of 600 m at a distance of 25 km. The

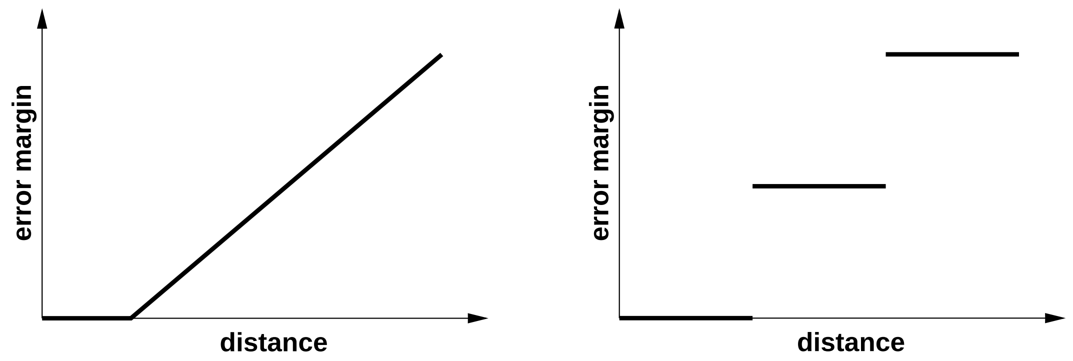


Figure 7: Examples of `margin_of_error()`–functions

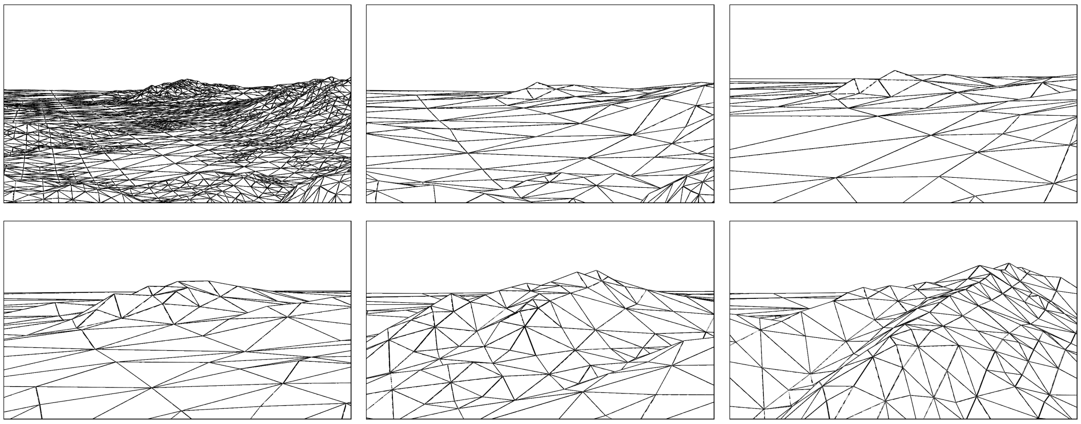

Figure 8: Observer approaching a mountain

right function could be used to simulate three separate level–of–detail areas. The advantage is that the triangulation of the terrain changes only at the borders of those areas which is under some circumstances more pleasing to the eye. However from those examples it should become clear that the level–of–detail is easily controllable and no constraints are imposed on the shape of the `eps()`–function.

Figure 8 shows a sequence of images as they occur when the observer approaches a mountain. The first and the second picture were created from the same point of view, only the first was triangulated at a much lower error margin in order to show what the landscape would look like if it was approximated exactly. The rest of the sequence shows that the closer the observer gets to the mountain, the more detailed the triangulation of the mountain becomes.

# 3 Results

The here described system was developed to run on a Indigo$^2$ Silicon Graphics Workstation with a R10000 CPU, 128MB of main memory and a Maximum IMPACT graphics board equipped with 4MB of texture memory. The designgoals of a range of sight of 25 km at a frame rate of 30 frames per second are met on this rather costly system. But since the program was written in C++ using the OpenGL and Motif libraries, it is fairly independent of the hardware it runs on. It is conceivable, that within a few years a low cost system using a multiprocessor board and a graphics board developed for the PC market supporting OpenGL on a hardware level can be developed. Under these circumstances the system will be affordable to owners of small private aircraft which was a design target from the very beginning of the project.

Besides the hierarchical reorganization of the array data previously mentioned some other ideas will be implemented in order to improve performance and image quality. One is to consider a local maximum in the elevation array a point of interest that must not be skipped during a triangulation. This way mountains would not pop up as they do now but only the slopes would become increasingly detailed when approaching a mountain. This event is much less detectable by the eye than a mountain suddenly popping up from a level horizon. This popping–up may sound dangerous in the context of a navigation aid for aircraft, but it occurs at a much too large distance, so it only causes an esthetical disturbance.

One idea to increase the performance of the triangulation is to use an incremental visibility check instead of the computational costly 3D–version of the Cohen–Sutherland algorithm currently used.

Another, as yet unconsidered problem is loading the next part of the database in flight while the aircraft is flying toward the edge of the terrain currently in memory. This obviously must not lead to any significant delay in terrain rendering. A slight reduction of the framerate may be acceptable, while a message to please stand by while new data is being loaded and processed is clearly unacceptable.

## References


[1] U. Rathmann. Künstliche Sicht. In Carl-Cranz-Gesellschaft, *CCG-Kurs LR 4.05 "Moderne Unterstütungssysteme für Piloten"*, volume 9. Oberpfaffenhofen, 1992.

[2] G. Sachs, K. Dobler, G. Schänzer, and M. Dieroff. Precision Navigation and Synthetic Vision for Poor Visibility Guidance. In *AGARD Flight Vehicle Integration Panel*, Lissabon, September 1996. AGARD.

[3] H. Möller and G. Sachs. Synthetic Vision for Enhancing Poor Visibility Operation. *IEEE Aerospace and Electonics Magazine*, 9:27–33, 1994.

[4] G. Schänzer S. Vieweg. Präzise Flugnavigation durch Integration von Satelliten-Navigationssystemen mit Inertialsensoren. In *DGLR-Jahrbuch 1992*, pages 171–177, 1992.

[5] T. Gerstner. Ein adaptives hierarchisches Verfahren zur Approximation und effizienten Visualisierung von Funktionen und seine Anwendung auf digitale 3–D Höhenmodelle. Master's thesis, Institut für Informatik, TU München, 1995.

[6] W.F. Mitchell. Adaptive refinement for arbitrary finite element spaces with hierarchical bases. *J.Comp.Appl.Math*, 36:65–78, 1991.

[7] K. Hiller. Datenkompression mit dem Dünn–Gitter–Verfahren. Master's thesis, Institut für Informatik, TU München, 1993.

[8] A. Frank. Hierarchische Polynombasen zum Einsatz in der Datenkompression mit Anwendung auf Audiodaten. Master's thesis, Institut für Informatik, TU München, 1995.

[9] A. Paul. Kompression von Bildfolgen mit hierarchischen Basen. Master's thesis, Institut für Informatik, TU München, 1995.

[10] M.H. Gross, O.G. Staadt, and R. Gatti. Efficient Triangular Surface Approximations using Wavelets and Quadtree Data Structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2), June 1996.

[11] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics – Principle and Practice*. Addison-Wesley Publishing Company, 2. edition, 1992.


*Some of the above references are available via the World Wide Web:*
`http://www5.informatik.tu-muenchen.de/publikat/publikat_e.html`