# HW3D: A tool for interactive real-time 3D visualization in GIS supported flood modelling

Jan Bender
Universität Karlsruhe
jbender@ira.uka.de

Dieter Finkenzeller
Universität Karlsruhe
dfinken@ira.uka.de

Peter Oel
Universität Karlsruhe
peter.oel@web.de

## Abstract

Large numerical calculations are made to get a prediction what damage a possible flood would cause. These results of the simulation are used to prevent further flood catastrophes. The more realistic a visualization of these calculations is the more precaution will be taken by the local authority and the citizens. This paper describes a tool and techniques to get a realistic looking, three-dimensional, easy to use, real-time visualization despite of the huge amount of data given from the flood simulation process.

**Keywords:** CASA2004, computer graphics, virtual reality

## 1 Introduction

Flood catastrophes cause tremendous damage to people and their belongings. 21 people died in Germany in August 2002 during a flood of the river Elbe. Ten thousands had to be evacuated and thousands lost their home. The total damage runs into 15 billion Euros.

Today high sophisticated techniques are used to simulate a flood catastrophe. The Institute of Water Resources Management, Hydraulic and Rural Engineering (IWK) of the University of Karlsruhe developed a GIS-supported flood model for the German river Neckar [1]. The model is transferred to the water management administration of Baden-Württemberg with the aim to support the handling of flood-relevant questions, e.g. the classification of legally valid flood areas or the risk analysis. Possible inundation zones are calculated during the simulation process and the result can be visualized as overlay in topographic maps or ortho-photos. These visualizations are used by the local authority to prevent further damage. Informing the local inhabitants about the risk of possible floods is another aim of the visualization. The visualization tool was originally made for the flood data of the river Neckar but it can be used for every other river as well.

Two-dimensional map-based visualizations do not have the degree of immersion like three-dimensional visualizations have. Lifelike visualizations are needed to make people sensitive to the possible risk of a flood. Realistic looking visualizations of that purpose are typically made using large animation and rendering tools like 3D Studio Max (discreet) or Cinema 4D (MAXON). The result is a movie of a possible flood.

But there are many disadvantages using these tools. An expert is needed to handle the software and to create the movie from the given data. Until a movie is completed much time is spent to prepare the data and to render all the single images. This can take several hours for the entire animation. After a movie is finished the path of the camera is fixed. It is not possible to change the point of view arbitrarily without recomputing the whole movie.

The flood simulation tools are used at the local authority and should therefore be easy to use and flexible in adjusting to different datasets. The goal of this work is to build a three-dimensional real-time visualization tool that gives the user the possibility to navigate freely through every simulated flood scenario. The main objectives are realism, short loading times and real-time visualization despite of the really huge datasets, since the digital terrain model used by the flood simulation software is of very high resolution.

The following topics will be discussed in this paper. An approach in displaying flood periled areas in a realistic way will be shown in section 2. An overview of existing techniques for real-time rendering of huge terrain data sets will be given as well. The data used for the visualization will be described in detail in section 3. Afterwards in section 4 the methods which made the real-time visualization possible will be introduced. The following section 5 describes how the methods mentioned in the preceding section are combined with displaying large texture maps. Section 6 discusses the user interface of the visualization tool. In section 7 the performance of the developed software is measured. The paper ends with section 8 discussing future work.

## 2 Related Works

In this work the results of numerical flood simulation and its three-dimensional representation in real-time are combined. The goal of the work is to give the user a realistic impression of the area affected by flood. This includes textured buildings, aerial photos, textured land use and animated water. For the three-dimensional real-time rendering of the data the amount of triangles for displaying must be reduced. There are different techniques in reducing the amount of triangles.

One is to generate a static lower resolution mesh or triangulated irregular network (TIN) [2] which is always used for rendering. The generation of this mesh is done in a pre-processing step in which the original mesh is reduced and optimized for a given resolution. The advantage of this technique is that a reduced mesh has to be built only once. But while navigating through the scene the terrain always has the same resolution even when the viewpoint is close to a rough region. A moving viewpoint leads to the demand of a viewpoint dependent triangulation of meshes. The term level of detail identifies the algorithms aiming at this property.

One approach is to divide the source mesh into smaller areas and generate several multi-resolution triangulations. Dependent on the point of view a different level of detail of these areas will be shown (mentioned in [3]).

Hoppe [4] describes a continuous view dependent reduction of meshes for terrain rendering (progressive meshes). He uses TINs as well as geomorphing to avoid the phenomenon of vertex popping. As described in section 4 Hoppe's algorithm is not useful for this work.

Roettger et al. [3] present an algorithm for real-time generation of continuous levels of detail for height fields. As data structure they make use of a quadtree representation of the height field. As a limitation they assume the height field to be at size of $2^n + 1$. In this work a modified version of this algorithm is used which is described in section 4. Lindstrom and Pascucci [5] used also a quadtree for their algorithm

Lindstrom et al. [6] and Duchaineau et al. [7] both use a binary triangle tree (triangle bintree) for the mesh as data structure. Duchaineau et al. take a pre-processed bintree to built two priority queues for split and merge operations that maintain continuous triangulation. They apply a top-down approach in which the mesh is recursively refined until an error metric stops refinement. In contrast Lindstrom et al. use a bottom-up approach to simplify the triangle mesh until an error metric stops the simplification.

VRMLflood of Rinner and Fitzke [8] is an approach to achieve a realistic real-time visualization of flood periled areas. Due to VRML the user is able to access flood data over the internet but a plug-in for the user's browser is required. Usual plug-ins for browsers running on MS Windows are Cosmo Player (Silicon Graphics, Inc.), Cortona (Parallel Graphics) or Contact (Blaxxun). With VRMLflood it is possible to choose different water levels and watch the scene from different viewpoints. A water level is a plane that is determined by one z-value. In contrast to that in this work every water level is given as an elevation grid with a high resolution. The use of a grid provides more accuracy.

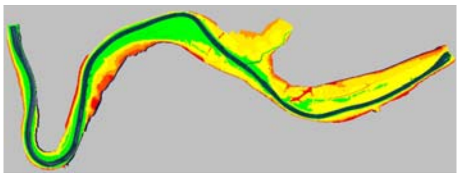

Figure 1: Top view on the terrain grid mesh

## 3 Visualization Data

This paper shows how to design a tool which allows the user to fly through a flood scene in real-time considering the special needs given from the flood simulation. The scene is a geometrical representation of the river landscape and consists of terrain with texture, water levels and buildings. Each dataset is georeferenced by using the Gauss-Krüger coordinate system. The data (except the texture images) is given in an ASCII-format.

The data used as terrain is a grid mesh. A z-value corresponding to the height of the landscape is given at each vertex. The z-value is a 32-bit floating point value. The top view on an example grid mesh is shown in figure 1. The grid consists of 3.108 x 3.869 vertices. Normally only the river surroundings are important for the flood simulation. For the other vertices there exist no z-values in the data. Only 1.686.538 vertices of the approx. 12 million vertices in figure 1 have a z-value. The resolution of the terrain grid mesh is relatively high because the flood simulation needs an exact representation of the landscape to get good results. Two neighbour vertices have the distance of 2 meters in the grid.

For the visualization of the water surface the user can load data for different water levels. The data for each water level is a grid mesh with the same resolution as the terrain mesh of the actual scene, also containing z-values corresponding to the water level for each vertex. The water level is not only a single z-value for the whole scene, since the flood can be blocked by a bank. For a large scene the absolute water level can change during the course of the river. After loading more than one water level the increase or the decrease of the water surface can be animated by interpolation.

The texture that is used for the terrain is normally an aerial photo or a map of the region. In general the texture needs more space than a modern graphics card can keep in its texture memory. For example, a map with a low resolution (one pixel per 1.25 meter) for the grid of figure 1 would have 4.972 x 6.189 pixels. With a colour depth of 32 bit the texture would need 123 megabytes of memory.

The flood simulation is of special interest for regions where people live or where important historical or industrial buildings are. For all the buildings in the flood region there exist floor plans given as polygon data. Using this data the developed software is able to generate the buildings for the flood scene. With the visualization tool the user can detect which buildings are affected by the flood and which are not. For the terrain grid mesh of figure 1 10.562 buildings had to be generated by an algorithm.

The amount of data that the software has to handle is huge. The next section will show how the real-time visualization of this data was realized.

## 4 Real-time Visualization

With an actual low end system it is not possible to render the data as described in the last section in real-time. The systems are too slow to handle that amount of data. The only way to reach a faster visualization is to reduce the given data.

The difficulty using a data reduction is to minimize the difference between the original data and the reduced data. The minimization of this error is a special challenge in the visualization of flood data. Every error in the elevation data of the terrain or the water can have the effect that the wrong areas in the terrain are flooded. The error at a point which is far away from the viewer is less disturbing than the error at a near point. This fact has to be considered when designing an algorithm to reduce the data.

Both the water and the terrain grid mesh are given as an ASCII-character-file with all vertices inside. One of the requirements was that the algorithm must not spend much time with preprocessing. Because of this the construction of a progressive mesh [9] with view-dependent refinement [10] was not possible. Instead of this a continuous level of detail algorithm was needed that works directly with the elevation grid.

The next sections introduce the methods used to reduce the data. Afterwards the results of these methods will be presented.

### 4.1 *Continuous Level Of Detail*

For the visualization tool a modified version of Roettgers continuious level of detail algorithm [3] is used to decrease the triangle count of the terrain and the water triangle mesh. It has the advantage that it does not need much time for pre-processing. The data structure of this algorithm is a quadtree.

A new triangle mesh is generated in two steps. First a matrix is build by recursively traversing the quadtree. This matrix has an entry for every vertex in the original grid and it represents the quadtree. For each node that is reached while traversing the quadtree a special value is set to the corresponding entry in the matrix. The value 0 is set if the actual node did not pass the view-frustum culling, i.e. that the whole sub-tree of this node is not visible and will not be displayed. If none of the vertices in the node has a z-value then the value is set to 1. Otherwise the node lies in the view-frustum and it has valid data for drawing. The entry in the matrix is set to the value 2 if the actual node should not be refined further and to the value 3 if more detail is needed. If one of the first three values is set then the corresponding part of the quadtree is not traversed further. To determine if a node should be refined further or not a cost function is used. The decision for a refinement depends on the distance between the viewer and the node and on the height difference in the node. For the reduction of the water data the second part is not needed because the water mesh contains no relevant height differences.

After the matrix is filled with the necessary values the second step is to draw the triangle mesh. In this step the quadtree is traversed a second time. The matrix is used to determine if a node should be refined further or if it should be drawn in the corresponding resolution. If the matrix has the value 0 or 1 then the node will not be drawn at all.

Triangle fans are used to draw a node in the triangle mesh. While drawing a node it is necessary to take a look at the resolution of its neighbour nodes. This can be done by using the matrix. If the resolution differs then cracks in the mesh will appear as shown in figure 2.

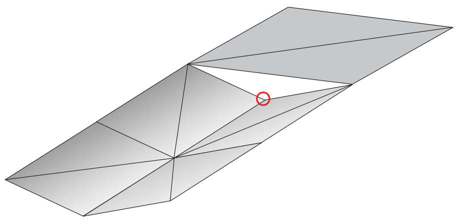
Figure 2: The problem of mesh cracks

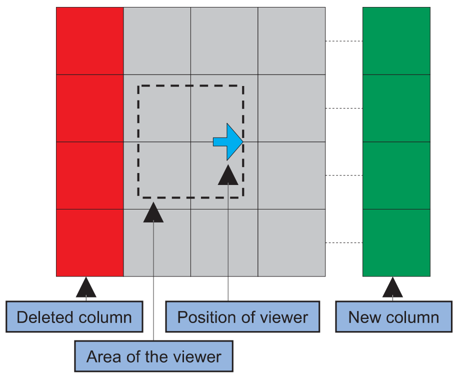

Figure 3: Tile management

To avoid this crack the vertex which is marked in figure 2 is skipped. The triangle mesh can be drawn without any cracks by skipping vertices as shown above if it is guaranteed that the level of detail of two neighbour nodes differs at maximum by one.

### 4.2 *Tile Manager*

The flood region which the software has to visualize has an arbitrary size. That means that even with using the continuous level of detail algorithm there are flood scenes which cannot be handled in real-time because they contain too much data.

The real-time condition can only be complied with if the amount of data that the system must handle has an upper bound. To limit the data only a part of the whole scene is drawn. While the viewer navigates through the flood scene the visible part has to be changed so that he never leaves it. To reach this goal the whole data of terrain, water, buildings and texture is partitioned in tiles with a fixed size.

Figure 3 shows the visible part of the scene consisting of 16 tiles. If the number of tiles that is used for the visible part is 4 to the power of n then the tile management can be easily combined with the quadtree of the continuous level of detail algorithm. The quadtree algorithm just needs to start at the n-th level. The arrow in the figure marks the position of the viewer. If he leaves the area in the centre of the visible part that is limited by the dashed line a new row or a new column of 4 tiles is loaded and on the other side a row or a column of tiles is deleted. Afterwards the viewer is again in the centre of the visible part. The limited area must have at least the size of one tile and it should not be bigger than four tiles for good results.

Terrain-, water- and building-tiles have exactly the same size and position. The partitioning of the terrain texture is a special case for the tile management. For drawing the scene a texture is needed that fits into the texture memory of the used graphics card and that has a resolution of $2^n \times 2^m$. The size of the texture tiles is chosen fulfilling the two conditions. In general the texture tiles have a different size and position than the tiles of terrain, water and buildings. The movement of the visible part of the texture depends just on the position of the terrain part and not directly on the position of the viewer.

With the tile management technique only a few tiles need to be kept in the main memory and to be drawn. The rest of the tiles can be swapped out in temporary files on the harddisk. These temporary files can also be used for reducing the time that is needed to load a scene. Every time the user wants to load data for terrain, water, texture or buildings the software checks if the temporary files for the chosen data set are already on the harddisk. If it has found the files then they are loaded instead of reading the original data and pre-processing it again. The speed-up is tremendous especially on loading terrain or water data because the grid meshes are given in an ASCII-character-file. A mesh with 3.108 x 3.869 vertices like the one in section 1 needs 75 megabytes of disk space for example. For the temporary files a more efficient file format is used and it is only necessary to load the 16 visible tiles. In the end the amount of data to load is about nine times less than with the original files. By using this technique the software needs only a few seconds to load a whole scene instead of a few minutes.

## *4.3 Results*

The terrain grid mesh shown in figure 1 has 3.108 x 3.869 vertices. Now the reduction of data obtained by the tile management, view-frustum culling and the continuous level of detail algorithm is measured.

As described in section 3 only 1.686.538 vertices of the grid mesh have a valid z-value. Without any reduction of the data the generated triangle mesh has 3.360.793 triangles. With an actual low-end system it is not possible to draw a mesh like this in real-time. Figure 4 shows the terrain mesh without any optimization.

The first reduction of the data is done by the tile management. The 16 tiles of the terrain mesh contain about 1 million vertices but only 444.411 of them have a valid z-value. The visible part of the real scene has a size of 2 x 2 kilometres. The corresponding triangle mesh consists of 878.196 triangles. So after the first reduction only 26 percent of the original data is left.

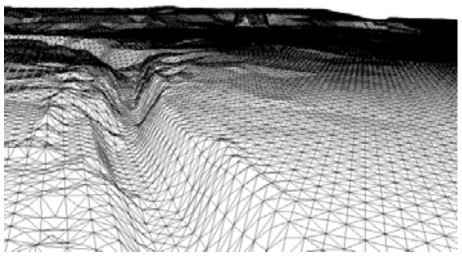

Figure 4: Triangle mesh without optimization

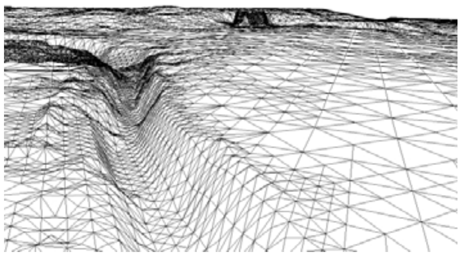

Figure 5: Triangle mesh with full optimization

As next step of the reduction view-frustum culling is executed. The actual camera frustum of figure 4 is used. After the reduction only the triangles that can be seen in this figure are left. After executing view-frustum culling there are 102.412 triangles left. Until now there is no error in the visible triangle mesh since none of the visible vertices was erased. After the two reduction steps only 3 percent of the original data is left without any approximation error.

The last step is to reduce the rest of the triangles with the continuous level of detail algorithm described above. The resulting triangle mesh is shown in figure 5.

After all optimization steps the triangle mesh consists of 9.771 triangles and it has an acceptable approximation error.

In the end the triangle mesh contains only 0,3 percent of the original data. This mesh can easily be drawn by an actual low-end system in real-time. The triangle mesh of a water elevation grid can be reduced even more. A water mesh contains no relevant height differences so there is no need for a very high resolution at a special vertex.

## 5 Realism

Section 4 describes how a real-time visualization of large data sets is realized. This section will show how the data sets can be presented in a realistic looking way.

### 5.1 *Terrain Texture*

To draw the surface of the terrain a triangle mesh is generated from an elevation grid. If all triangles have the same colour then the surface does not look like a realistic terrain. A better solution would be to give each vertex of the triangle mesh a colour that depends on its height. Afterwards the surface of a triangle is filled with interpolation.

The visualization with a texture as surface of the terrain provides the best results. For example aerial photos could be used as a texture to get a very realistic visualization. Another possibility is to use a map as texture to give important information about the region to the viewer.

In general the textures used for the terrain mesh do not fit in the texture memory. Because of that the textures are partitioned in smaller tiles. In the end there is no big terrain texture anymore. There are many texture tiles that fit together. The disadvantage of having more than one texture tile is that on each line where two neighbour texture tiles meet there appears

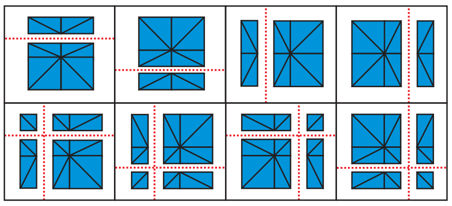

Figure 6: The division of triangle fans

a problem with the triangle mesh. If a texture tile ends in the middle of a triangle fan and another tile begins then there are triangles in the fan which have more than one texture. This is undesirable because every triangle should belong to exactly one texture tile. To solve this problem the affected triangle fan has to be divided.

Figure 6 shows all possible divisions of triangle fans that could appear. Dashed lines mark the borders of the texture tiles.

### 5.2 *Water*

The flood simulation software computes water grid meshes for different points in time. The increase or the decrease of the water level can be animated by interpolation over time. The interpolation of the whole grid mesh takes much computation time. A faster solution is to interpolate only the necessary vertices of the water grid mesh. For the real-time visualization of water the algorithm of section 4 is combined with interpolation. The z-value of a vertex is calculated when needed for the first time. In the end only the vertices are interpolated which the algorithm did not delete during the reduction.

For a realistic visualization of the triangle mesh a water texture is used. This texture is made transparent with alpha-blending. A transparent texture has the advantage that the user can see the terrain that is under water. If a map is used as terrain texture then the user can see directly which streets, buildings, etc. are affected by the flood. The geometry of the water must not be changed for generating waves on the surface. Because of that the motion of the waves is simulated by multi-texturing. The same water texture is used two times. Both textures are rotated continuously against each other. By adding the colour values of the two resulting images a new texture is generated for each frame. With this technique the viewer gets the impression that the water surface moves but without a certain direction.

While drawing the scene the depth-buffer of the graphics card is used to determine which parts of the terrain are flooded. In general the depth-values saved in the buffer are rounded. The size of the arising rounding error depends on the amount of bits per pixel used by the depth-buffer and on the depth of the view-frustum. According to [11] the loss of accuracy can be described with the following formula:

$$\log_2 \frac{z_{far}}{z_{near}} \qquad (1)$$

The terms $z_{near}$ and $z_{far}$ describe the depth of the near and the far clipping plane of the view-frustum. Figure 7 shows a screenshot that was made with a 16 bit depth-buffer. Lines of water appear over the terrain because of rounding errors in the buffer.

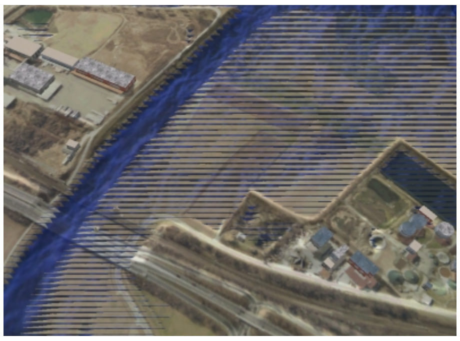

Figure 7: Problem with depth-buffer

To reduce the rounding errors which appear there are two possibilities. Either a graphics card which provides a bigger depth-buffer is used or the depth of the view-frustum is reduced.

The depth of the view-frustum depends on the near and the far clipping plane. To reduce the depth the clipping planes must move closer together. The view-frustum is intersected with the terrain surface. Afterwards the point of intersection with the minimum depth-value determines the position of the new near clipping plane. The new position of the far clipping plane is computed analogous.

## *5.3 Buildings*

In general flood simulation and visualization is made for inhabited regions. A local habitant will get a realistic impression more easily if he sees the three-dimensional buildings in the visualization. The data for the buildings are original data from the local land surveying office. For each building there are a two dimensional polygon representing its floor plan and attributes for additional information about the building.

The first step is to get a valid z-value for the polygon of the floor plan. This value can be computed with the elevation grid mesh of the terrain. For each vertex of the polygon the z-value in the terrain is determined by interpolation. The minimum of these values is used as the z-value of the whole polygon. The height of a building is estimated depending on its purpose. For example dwelling houses will be less in size than churches.

Dependent on the purpose of a building its 3D shape is automatically generated. This means that depending on the building's floor plan different roofs are created for dwelling houses, factories, churches etc. Factories receive a flat roof whilst dwelling houses and churches get a pitched roof. The algorithm for generating the pitched roofs is based on the work of Oswin Aichholzer and Franz Aurenhammer [12]. To achieve an even more realistic impression the buildings are textured depending on their purpose. Therefore different textures for doors, walls and windows are used. In figure 8 the left image shows how a church is constructed with triangles. The image on the right shows the same church drawn with textures for its walls, its windows and its roof.

Whole cities are created upon these buildings. But instead of creating these city maps procedurally like Yoav I. H. Parish and Pascal Müller [13] did the cities in this work are reconstructed based on real data. For each building its georeferenced position is given and therefore its exact placement on the terrain is known. This improves the visual impression of the cities and it is easier for the inhabitants to recognize their own city.

While the user navigates through a flood scene he can ask for the additional information that exists for a building by clicking with the mouse pointer on it. This information contains for example the purpose of the building.

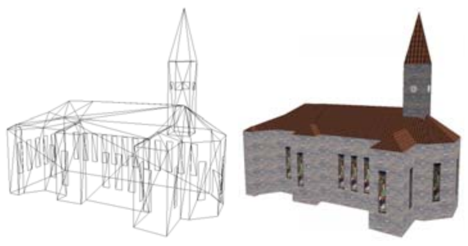

Figure 8: Model of a building

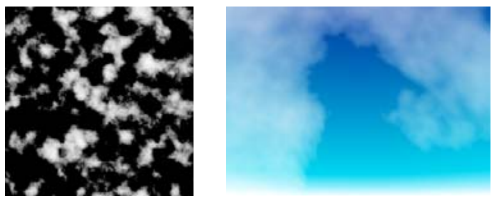

Figure 9: Cloud-texture with Perlin noise

### *5.4 Light And Sky*

The use of light in a scene is very important to get a realistic three-dimensional impression. By using light shading the contour of the terrain surface becomes visible.

For a realistic world a sky is needed in the flood scene. The sky should have a sun and clouds. The position of the sun is determined by the position of the light source. To draw the sun a plane with a corresponding texture is used.

The visualization tool uses a hemisphere to draw the sky. This sky dome is constructed with nearly 1.300 triangles. For the visualization the hemisphere is drawn two times. The first hemisphere is filled out with different blue tones to simulate the atmospheric effects. For the second hemisphere a cloud texture is generated with the Perlin noise technique [14]. The left image of figure 9 shows an example for a Perlin noise texture. The two hemispheres are combined with alpha-blending. The right image of figure 9 shows the resulting sky dome.

To simulate the motion of the clouds the sky dome rotates slowly around the z-axis.

## 6 User Interface

One of the main goals of this work is to develop a tool that is easy to use. After the user has loaded a new terrain dataset in the tool it will be immediately displayed. Now he can navigate freely through the scenario by keyboard and mouse inputs. Other datasets like texture, water levels and buildings can be added at any time to the scenario by loading the desired file.

The tool provides a map functionality for an easier navigation. Every georeferenced image of the displayed region can be loaded in an extra window and can serve as map. The position of the user is marked in this map. So the user always gets information about his current position. The map can also be used to jump immediately to a special point in the scenario.

After loading buildings the user can get information about every single building by clicking with the mouse on it. If the user has loaded more than one water data set then he is able to start an animation of the increasing or decreasing off water level. So he can watch which buildings, streets etc. will be affected by the flood.

The visualization tool provides a keyframe technique. The user can mark several positions of the camera during his flight through the scenario. The tool computes a control polygon for the marked points in order to generate a cubic B-spline for the flight path of the camera. This flight path is displayed in the map window and it can be used for an animated flight through the scenario. A flight path can be modified at any time by the user and he can save it to use it later again for example for a presentation. A video clip can be generated of a flight animation so a presentation of a flood model is possible an every computer. These video clips can be used by local authorities to inform the inhabitants of a possible flood.

## 7 Results

This chapter contains information about the memory requirements of the developed software and about its speed. In the end some screenshots will be presented.

For the measurements of memory requirement and speed a flood scene which contains the following components is used: a terrain grid mesh of 3108 x 3869 vertices, two water levels each with a grid mesh of 3108 x 3869 vertices, a 24 bit terrain texture with 4972 x 6189 pixels and 10562 buildings.

During the pre-processing the software generates a temporary file for each tile of the scene.

The following table shows the amount of memory that is needed by these files on the harddisk. Altogether the temporary files need about 88.32 MB of disk space. The central memory that is needed for this scene during runtime is about 30 MB. The measurements of speed were made on a PC with the following components: AMD Athlon XP 1800+, 512 MB PC-266 DDRAM and nVidia GeForce3 with 64 MB.

|  | Terrain | Water | Terrain Texture | Buildings |
|---|---|---|---|---|
| Number of tiles | 62 | 62 | 203 | 100 |
| Memory needed per tile | 793KB | 538KB | 5-40KB | 1-139KB |
| Total memory | 48MB | 32.5MB | 5.3MB | 2.52MB |

Table 1: Memory requirements of the temporary files

The speed and the number of triangles were measured at several different points of view in the flood scene. The resolution was set to 1024 x 768 pixels and the colour depth was 32 bit. All results are shown in the following diagram.

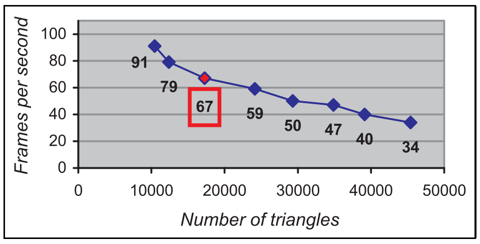

Figure 10: Measured rendering performance

A longer random flight through the whole scene was made. The flight went several times through each tile. To measure the number of triangles and the frames per second the parameters of the continuous level of detail algorithm were not changed to maintain a certain frame rate. The maximum number of triangles that was measured at a point was 45.401. At this point the algorithm rendered 34 frames per second. The average number of triangles was 17.259 and the average frame rate was 67. For a real-time visualization a rate of 25 frames per second is needed. So the software has fulfilled the real-time condition.

Figure 11 shows four screenshots of the developed visualization tool. Two images were made with a map as texture and for the other two an aerial photo was used.

## 8 Future Work

In future the algorithms for the real-time visualization will be optimized. Furthermore the visualization should become more realistic. Additionally to the polygons for buildings there exist polygons for special areas with information about their purpose. For example there exist polygons of forests or industrial areas. These polygons can be filled with a suitable texture or with suitable objects (e.g. tree objects for a forest area).

Another planned feature is the export of the geometry and texture data in a file which can be read by actual 3D software systems.

## References

[1] P. Oberle, S. Theobald, and F. Nestmann. GIS-gestützte Hochwassermodellierung am Beispiel des Neckars. *Wasserwirtschaft 90*, pages 368–373, 2000.

[2] D. C. Taylor and W. A. Barrett. An algorithm for continuous resolution polygonalizations of a discrete surface. In *Proc. Graphics Interface '94*, pages 33–42, Banff, Canada, May 1994. Canadian Inf. Proc. Soc.

[3] S. Roettger, W. Heidrich, Ph. Slusallek, and H.-P. Seidel. Real-Time Generation of Continuous Levels of Detail for Height Fields. In *Procceedings of WSCG '98*, pages 315–322, 1998.

[4] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the conference on Visualization '98*, pages 35–42. IEEE Computer Society Press, 1998.

[5] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proceedings of the conference on Visualization '01*, pages 363–371. IEEE Computer Society, 2001.

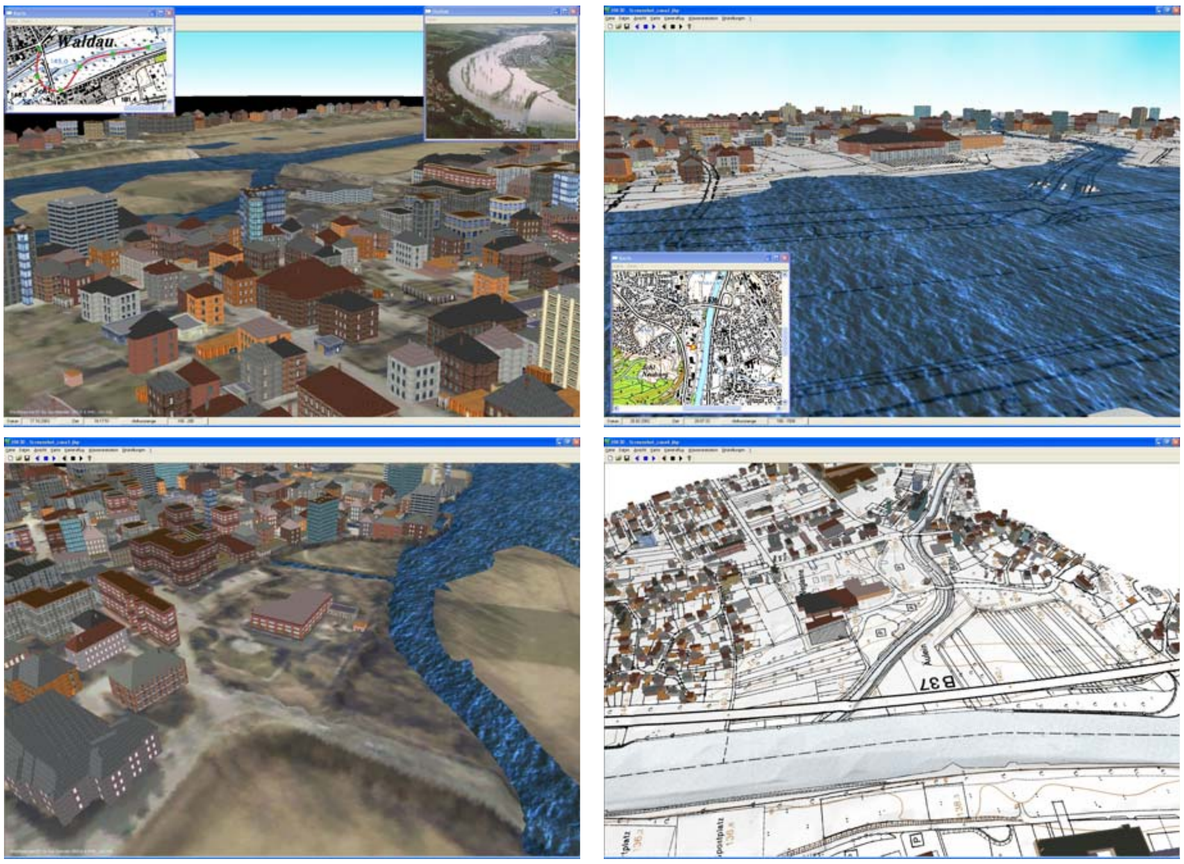

Figure 11: Screenshots

[6] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM Press, 1996.

[7] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88, 1997.

[8] J. Fitzke and C. Rinner. Visualisierung von Hochwasserszenarien mit VRML. In *Proceedings of Workshop Virtual GIS*, pages 28–29, 1998.

[9] H. Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[10] H. Hoppe. View-dependent refinement of progressive meshes. *Computer Graphics*, 31(Annual Conference Series):189–198, 1997.

[11] D. Shreiner, editor. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*. Addison-Wesley, 3rd edition, 2000.

[12] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Proceedings of the Second Annual International Conference on Computing and Combinatorics*, pages 117–126. Springer-Verlag, 1996.

[13] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM Press, 2001.

[14] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modelling: A Procedural Approach*. Academic Press, 1998.