2017, **6**(4): 15–26
DOI: 10.1515/ijicte-2017-0016

# Procedural Terrain Generation and Path Searching in Education

Pavla Grossmannová[1], Daniel Švábek[2]

[1]*Department of Informatics and Computers, Faculty of Science, University of Ostrava, Czech Republic {pavla.grossmannova@gmail.com}*

[2]*Department of Informatics and Computers, Faculty of Science, University of Ostrava, Czech Republic {daniel.svabek@osu.cz}*

**ABSTRACT**

This paper is primarily devoted to development of an educational desktop application, that describes terrain generating and pathfinding to students. The application is meant for lessons of Artificial Intelligence, that is one of compulsory optional subjects on Department of Informatics and Computers, University of Ostrava. Selected implemented algorithms enable to generate a terrain with adjustable parameters in three different implementations and it also enables pathfinding in three different ways. Used methods are compared according to different criteria for better understanding. Algorithms used in the application for terrain generating are fault method, hill algorithm and value noise. For pathfinding, there were used Dijkstra algorithm, A* and breadth-first search. The application enables camera movement around the terrain and graphical projection of costs of edges for pathfinding.

**KEYWORDS**

*Terrain generating, terrain editing, pathfinding*

## 1 INTRODUCTION

In today´s computer games, algorithms for terrain generating and pathfinding are used very often. Although this field is well known, there is still room for improvement, mainly because of time and space complexity or realistic terrain appearance.

Talking about available terrain generators, most of them provide procedural terrain creation followed by free editing. This ensures that terrain meets our requirements and we don´t have to create everything manually, which results in time saving. Most of current terrain generators doesn´t create real 3D landscape. In fact, it is only pseudo 3D terrain, sometimes called 2.5D. 2.5D is used in presented application. Noise functions are often used for terrain creating. However, you can use them for example to simulate handwriting or texture creation. As for pathfinding, we can see it not only in computer games, but also in the real world. These are, for example robotic vacuum cleaners looking for optimal routes or navigation systems.

Main reason of creation of the application was an attempt to give students an idea about possibilities in the field of terrain generating and pathfinding. Graphical output provides better understanding of used methods, thanks to their visual representation.

## 2 TERRAIN GENERATION POSSIBILITIES

There are many ways, how to deal with the generation of terrain. However, the methods try to do the same, creating something that resembles real terrain. The degree of similarity depends on purpose for which the terrain is created. In computer games, there is sometimes excessive realistic undesirable and landscape is often required to be adapted for fast processing and gamer´s walk-through of game.

The first option, how to create terrain is to generate it from curves. Unfortunately, in this case, it is difficult to simulate impression of randomness and the description of the terrain is also very complex. Generally, this approach isn´t used very often. The second way, and the more frequent, is "pure" procedural generation. Here are procedures using fractal geometry.

First mentioned method is fault method, which has a very simple procedure. First, a matrix of points is created, where each point has height set to zero. Subsequently, the area of matrix is divided into two parts, most often by a single line. The height of a part increases by some value and the height of the other is reduced by the same value. The procedure is then repeated, until we get desired appearance. We don´t need to work only with lines. Any shape can be used to divide area into two parts. Besides the variation, when we change height like in previous text, we can use other possibilities, to create a smoother surface. For this we use for example the sinus or the cosine functions (Fernandes, n.d.).

The second option for terrain creation is diamond-square algorithm. A condition of its use is to work with square array, but nothing prevents us from creating large square array and after procedure of creating terrain, we can "cut out" a rectangle or other non-square shape. The main principle is that we assign height values first to the centre of the square and then to the centres of sides of this square. Than we repeat process for newly created squares. Everything is repeated until the whole field is full of values of height.

Another here presented option is hill algorithm. It works by randomly choosing locations on which it creates hills. We can influence parameters of the hills such as location and size. If the new hill interferes with any of the previous, the final height of the collective points is sum of the height of point from an original hill and a new hill. We can affect the appearance of terrain by adjusting the range of random values or decreasing the maximum possible radius value with each new paraboloid or reducing it only for the certain number of new paraboloids (Nystrom, n.d.).

Value noise is another option. It is based on a composition of several grids together. Grids are generated with decreasing spacing between selected grid vertices. For the first octave, this spacing is the largest possible – grid size – meaning only four vertices are selected. Height of selected grid vertices is randomly generated from predetermined range and height of vertices between them interpolated. Appearance can be easily improved be creating one or more new octaves with a smaller spacing. The more octaves are created, the more it looks realistic. These octaves are eventually added together to build the final terrain. It should be noted, that every other octave has a higher frequency (smaller spacing) and a lower amplitude (smaller random height interval), otherwise summation of octaves wouldn´t have desired impact (Code.google.com, 2011).

Very popular technique is use of a Perlin noise, as it can be generally used in any number of dimensions. But we work most often only maximally with three dimensions. The main idea is that, for example for 2D, the function accepts parameters *x*, *y* and returns a single value, as well as in 1D or any other dimension. Returned value is always from <-1,1>. It should also be noted that for the same input values the function always returns the same output value (Biagioli, 2014). For more info about an improved Perlin noise see (Perlin, 2002).

Ken Perlin, author of Perlin´s noise, also created a simplex noise. Although the Perlin´s noise is abundantly used, it suffers from several ailments, that the simplex noise tries to remove. The main differences are lower computational difficulty and the fact that its generalization to higher dimensions is computationally less demanding than for original Perlin noise (Gustavson, 2005).

### Terrain editing and completing

Already created terrain can be modified by below mentioned methods to meet our ideas. We can, for example, try to increase valleys. This can be done simply by raising normalized values of height of each point to the power of two. This step affects more lowest values and higher values only a little bit. If larger valleys are needed, it is possible to use raising to the power of three, etc. (Nystrom, n.d.).

For the creation of islands is suitable hill algorithm. The goal is to prevent the hills from being placed in contact with the edge of the terrain. This will create an island. Of course, we can set the island´s minimum distance from the edge arbitrarily. With enough hills in the centre of area, it looks like the island.

Mountains generated by the above-mentioned methods, unlike the true mountains in nature, look like mountains even turned upside down. Only geologically young objects achieve this in the real world. Therefore, it is necessary to use for example erosion algorithms on the generated terrain to make the mountains different from the valleys. The main three groups of erosion used it terrain editing are thermal erosion, hydro erosion and wind erosion (Žára, 2004).

## 3 PATHFINDING

Finding paths in a map means finding paths in graphs by using graph algorithms. All points, that we can find during searching are represented as nodes of graph. Connections between nodes are represented as edges.

Well-known pathfinding algorithm is the breadth-first search. Graph is searched symmetrically. This means, that it spreads equally into all sides. It doesn´t favor any of his neighbors. So, we can say, that algorithm always extends into nodes lying in the distance *v* sooner than in distance *v+1*. Other well-known possibility is the depth-first search. It, as well as the breadth-first search, is uninformed method as it doesn´t take into account the location of target node and doesn´t use heuristic. This algorithm may not find the shortest path from start to end. This is ensured only if graph is tree-shaped and start node is root node (Kolář, 2000).

Dijkstra algorithm works with values of edges. It is used to find shortest path in weighted graphs. Weight of edge can´t be negative number. It is not necessary for the shortest path to have the least possible number of nodes, but sum of values of edges is always the least possible (Kolář, 2000). We can also use Bellman-Ford algorithm. This method can be used also for unweighted graphs, which is the main difference over Dijkstra algorithm.

A* algorithm is in the category of informed algorithms because, unlike those already mentioned, it uses a position of a target node. We can say, it looks for optimal path and explores less nodes than Dijkstra algorithm. Algorithm uses function *f(u)*, so-called heuristic function, which is for each node determined as follows:

$$f(u) = s(u) + c(u) \tag{1}$$

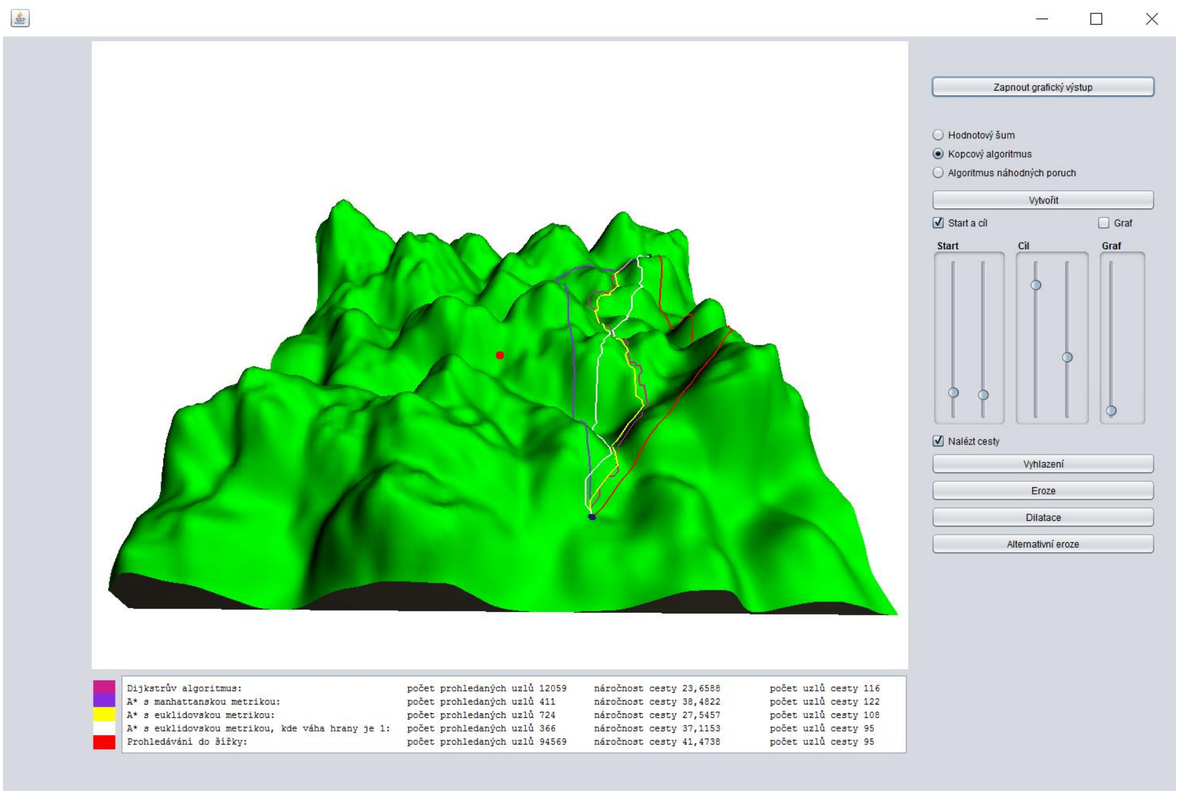
Value *s(u)* is distance between used node and start node. For this number we can use value calculated as in Dijkstra algorithm. Value *c(u)* is an estimate of distance from used node and the target. For the estimate of the distance we can use for example Manhattan metric or Euclidean metric (Kolář, 2000).

SMA* is abbreviation for Simplified Memory Bounded A*, it is an algorithm based on A*. Main significant difference, is that the algorithm has a limited memory size compared to A*. Therefore, it is necessary to define which nodes are stored in memory and which are removed (Russell, & Norvig, 1995).

## 4 APPLICATION

The application is written using the Java programming language, version 8. OpenGL (Open Graphic Library) was also used, accessed through the LWJGL (Lightweight Java Game Library). Application can be found at: http://www1.osu.cz/~svabek/TerrainApplication.zip

## Application environment



```
Dijkstrův algoritmus:                               počet prohledaných uzlů 12059    náročnost cesty 23,6588      počet uzlů cesty 116
A* s manhattanskou metrikou:                        počet prohledaných uzlů 411      náročnost cesty 38,4822      počet uzlů cesty 122
A* s euklidovskou metrikou:                         počet prohledaných uzlů 724      náročnost cesty 27,5457      počet uzlů cesty 108
A* s euklidovskou metrikou, kde váha hrany je 1:    počet prohledaných uzlů 366      náročnost cesty 37,1153      počet uzlů cesty 95
Prohledávání do šířky:                              počet prohledaných uzlů 94569    náročnost cesty 41,4738      počet uzlů cesty 95
```

**Figure 1** GUI

The application consists of three main parts. The first part is a window for terrain. The second part is section located on the right side, enabling to user interact with program. Last part is a text listing below the main window, where we can see the result values for each algorithm after pathfinding.

## Application control

The application is controlled by elements on the right side of the window, a keyboard and a mouse. The *Turn on graphic output* button displays the terrain created by the value noise. Switching between different terrains is done by selecting *Value noise*, *Hill algorithm* or *Fault algorithm*. When the *Create* button is pressed, the terrain is replaced by a new one. The *Start and finish* checkbox highlights two blue spheres that represent the start and finish of the path. Their starting position is in the upper left corner, unless the terrain is rotated differently. By selecting the *Graph* checkbox, a chart appears over the terrain. It represents difficulty of transition between nodes by its color. The sliders in the Start and Finnish sections allow user to set the starting position and destination for the pathfinding. In each section, one slider represents coordinate on the x-axis and the second on the z-axis. The slider in the *Graph* section deals with the y-axis. It increases and decreases the offset from the terrain of the chart as well as of the start and finish spheres and found paths, if they are visible. The *Find path* checkbox will search for paths between start and finish with use of predefined algorithms. These paths are displayed in the terrain and some of their values are listed in the text box below the terrain window. Found paths differ in color. How colors are assigned to individual algorithms can be seen next to the text box. The *Smoothing*, *Erosion*, *Dilatation* and *Alternative erosion* buttons are used to adjust the generated terrain. Through their proper combination we can achieve much nicer and more natural terrain than the initial terrain provides.

When user click on the window where the graphical output is displayed, it is possible to rotate, move and zoom the terrain. The *w*, *s*, *a*, *d* keypad keys make the camera move forward, back, left and right. The upper and lower arrow keys allow to move the camera up and down. The movement of the mouse controls the overall position of the camera, which rotates around the red dot, which is the center of the view. The mouse wheel controls zooming in and zooming away from the center of view.

## Specification of text output

The text output contains four columns. The first one specifies algorithm used for the path. The second one stands for number of scanned nodes. This value increases when algorithm finds unexplored node and changes the values of its variables. The third column represents difficulty of the path. It is the sum of difficulties of edges that creates path. The last column is number of nodes of path. This number indicates, how many nodes the path contains.

It is possible to have objection, that algorithm find the shortest path only if it explores all nodes. That is true, but for the testing purpose and specially to see savings in nodes exploring for A* versus Dijkstra algorithm, a variant of the so-called early termination was chosen. So, the search ends when the finish node is found.

## Graph

Each node in the graph is linked to nodes in its immediate vicinity. If we don´t take into consideration nodes on edges of space for terrain generating, then the number of neighbors is always eight. The color of the edge between two adjacent nodes indicates, how difficult it is for the search algorithm to use it. We can say that green means rather a flat surface and red means steep edge. This is based on used evaluation function (2).

$$y = \frac{1}{-\left(\left|b_y - a_y\right| + 1\right)} + 1 \tag{2}$$

Here $b_y$ stands for height value of node, $a_y$ stands for height of one of his neighbors. Since absolute value is always greater or equal to zero, $y$ is always positive number in the interval <0, 1). The $y$ value is than used for assign color to edge by command GL11.glColor3f(y, 1-y, 0). Function glColor3f accepts the red, green and blue values of color as parameters in range <0, 1>.

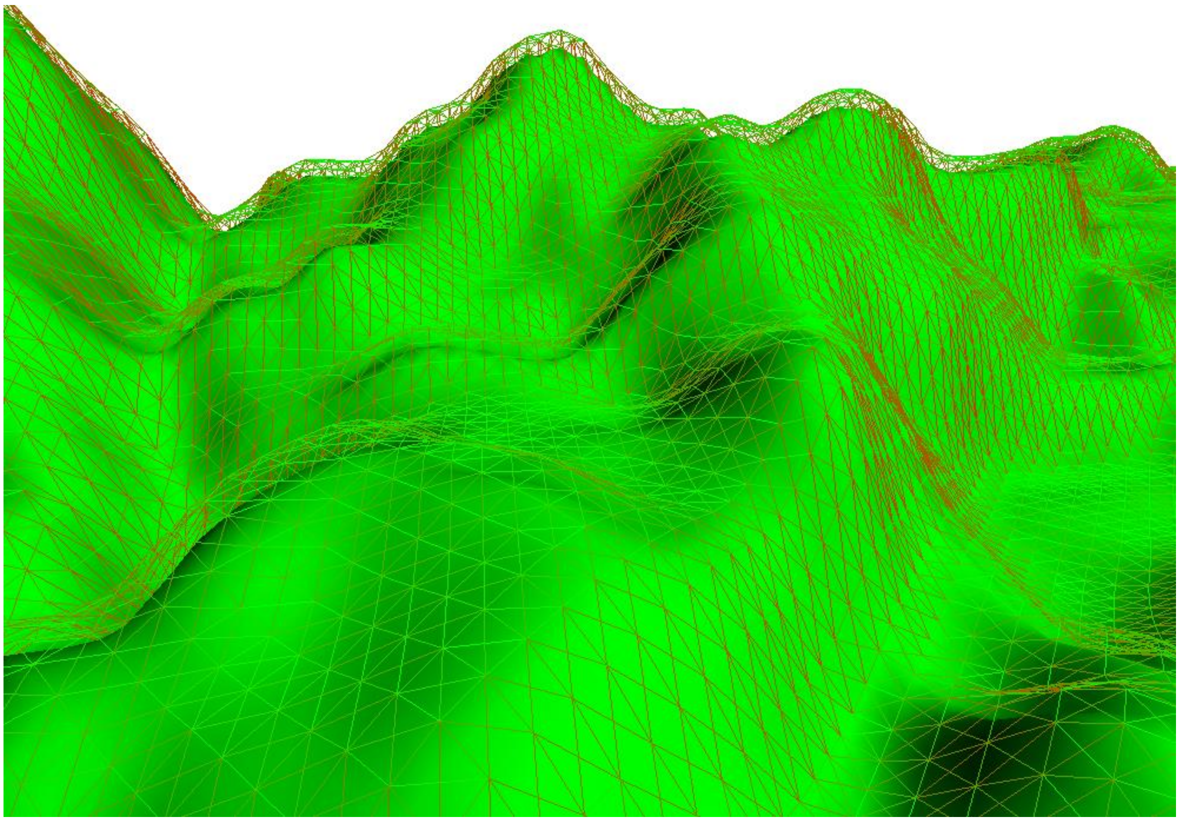


**Figure 2** Colored edges

## Generation of terrain

Application implements three methods: fault algorithm, hill algorithm and value noise. The reason for this choice was the fact, that these methods don´t provide the satisfying terrain unlike the Perlin´s noise. The main aim was to find out, if we can get a good-looking terrain from them thank to the follow-up adjustments. All three algorithms use a random number generator in their implementation to ensure that each terrain is original.

The terrain is a field of vertexes (129*129). Vertexes at the extreme edges are not rendered and serve only for calculations. Each vertex has value of its height. That´s why we´re talking about creating 2.5D terrain and not about the real 3D. We can never create caves or overhangs. Vertexes are grouped into strips of triangles and these are then rendered in the graphical output window. The following are examples of each algorithm without any modification.

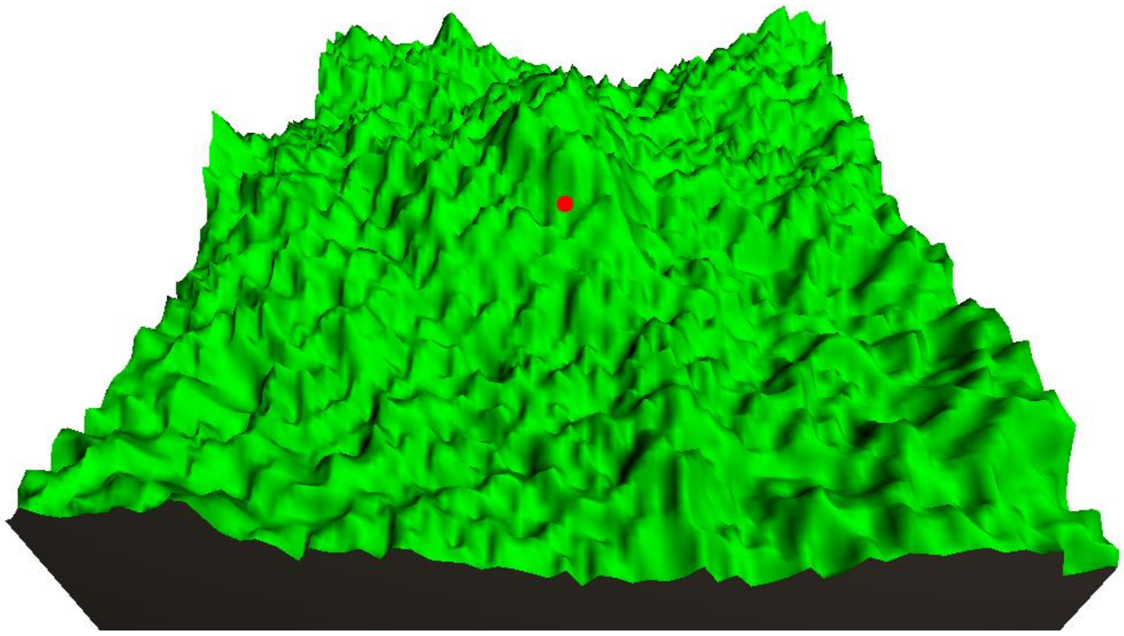


**Figure 3** Value noise

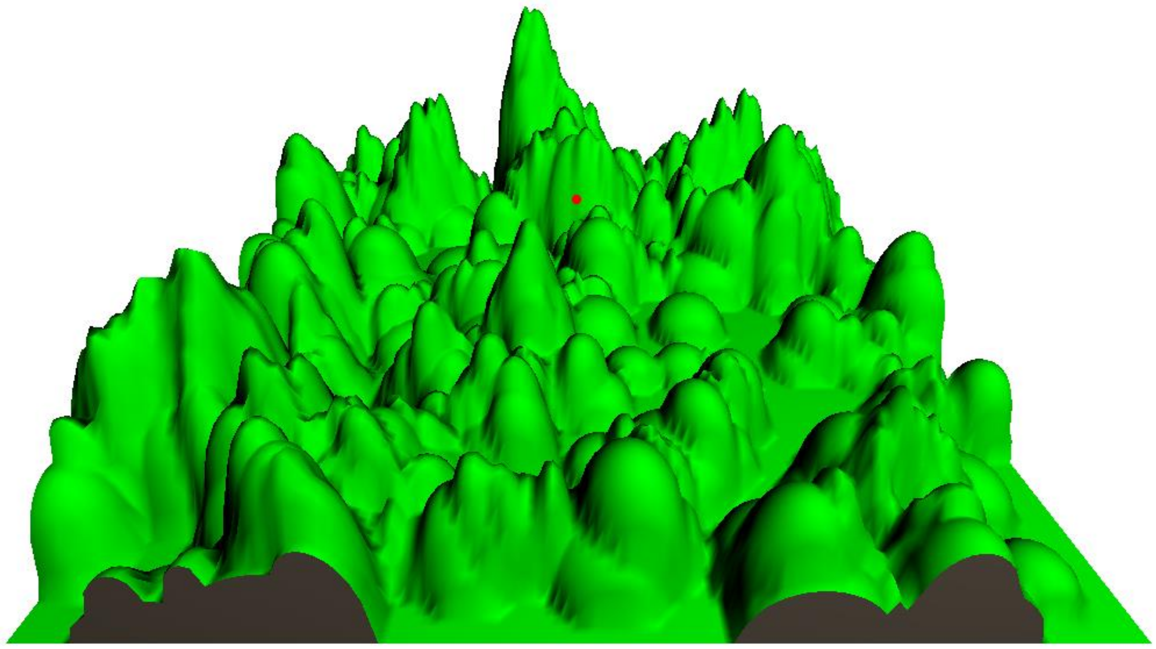


**Figure 4** Hill algorithm

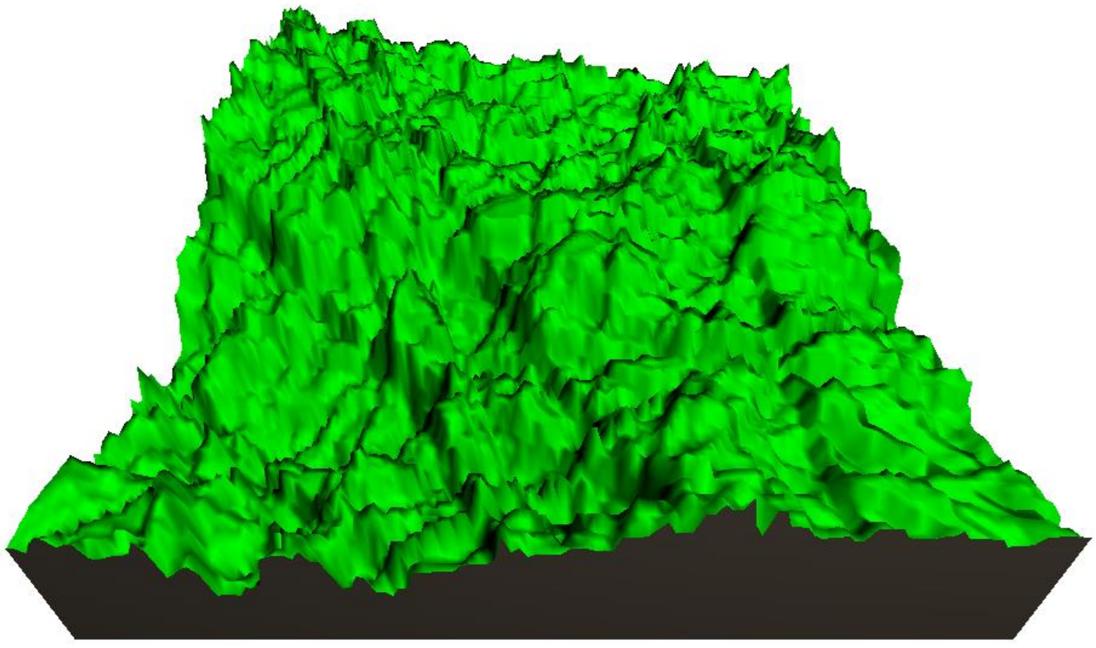


**Figure 5** Fault algorithm

## Editing of terrain

An appropriate combination of modifications creates a wide range of different shapes. All terrain adjustments implemented in program are based on a principle, that the new height value for vertex is created by height values of surrounding vertexes and its own height.

During smoothing we make an average of 8 neighbors and twice the middle vertex itself. For vertices that lie on the edge of field, we work with smaller neighborhood. The central vertex is counted twice to increase its influence, eventually reducing smoothing effect.

Erosion works only slightly differently. We again work with one vertex and its neighborhood. But now we add only values of height, that are smaller than height of central vertex, and twice the height of central vertex itself. The sum is again divided by the number of used vertices. This modification used multiple times results in creation of craters.

Dilatation of the terrain causes its visual swelling and aligning. The sum in this case includes vertexes whose height is greater than value of central vertex.

The last created option is an alternative erosion. Compared to the previous one, it uses randomness. The vertex height is used only if random value generated from the interval <0, 1> is greater than 0.8. Final value is equal to sum of used vertexes divided by their number plus one plus random value from <0, 1>.

To avoid unwanted visual artefacts or terrain behaviour, such as waves, it is necessary not to change values of vertexes immediately. Therefore, the second field of vertexes with the same size is created and newly calculated values are stored in it. Finally, these two fields are swapped.

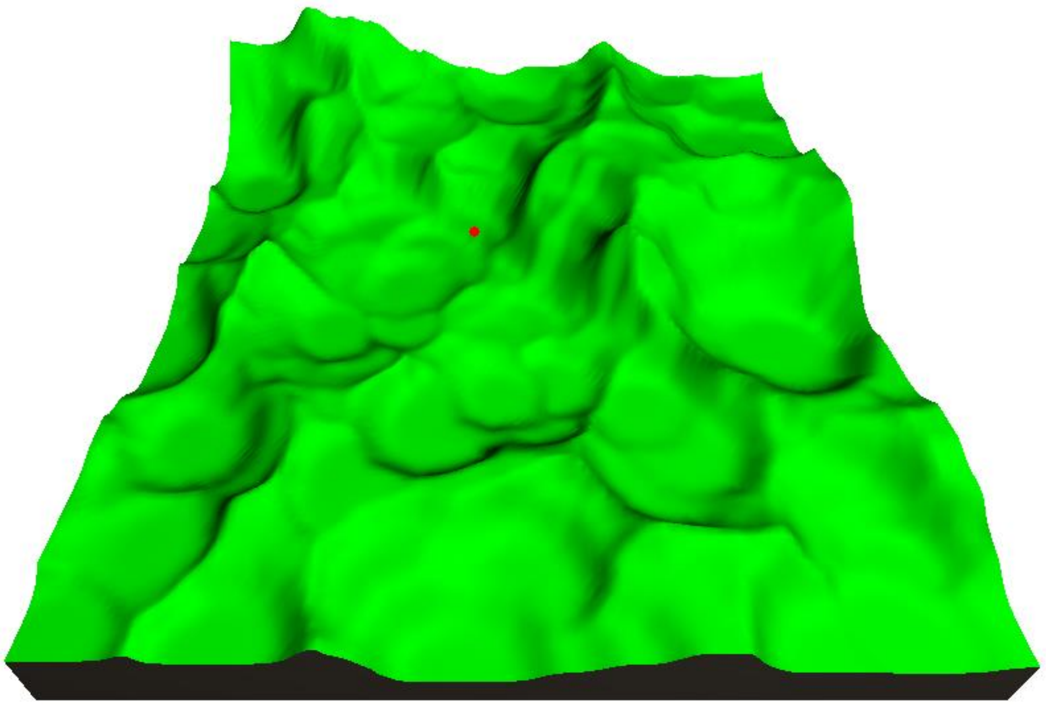


**Figure 6** Craters

In figure 6, we can see a multiple application of erosion on a terrain created by fault algorithm.

## Pathfinding

To demonstrate the algorithms that are looking for a path were chosen Dijkstra algorithm, A* in three modifications and breadth-first search. All these methods use a priority queue to store nodes while going through terrain. Nodes are ordered according to the rating function. The higher the value is, the further the nodes are inserted from the beginning of the queue. At the beginning, the queue always contains only the start node. Than nodes from his neighbourhood are added according to priority. First node in the queue is removed from it and his neighbours are explored. If these are not field's border nodes, then number of neighbours is always eight. To consider the fact that the diagonal transition is longer than the horizontal and vertical transitions, the diagonal values of difficulty are multiplied by $\sqrt{2}$. The edge evaluating procedure is similar to the procedure of colorful graph creation. Difference is that now even transition over flat surface is not zero. See equation (3).

$$y = \frac{1}{-(|b_y - a_y| + 1)} + 1 + 0.01 \tag{3}$$

Here $b_y$ is the height of the node and $a_y$ is the height of one of his neighbors. Value of 0.01 will guarantee that even a crossover over flat surface will not have a zero value of difficulty.

A* uses the approximate direction in which the target should be. Program shows three possible variants. The first of them uses the Manhattan metrics. The second is Euclidean metric that uses real value of edges and the third is Euclidean metric that uses edges evaluated by 1. The use of the A* results in decrease of explored nodes. The price for this reduction is loss of possibility to find only optimal path. This path can be the shortest one, but it is not ensured. If we want to compare the Euclidean and Manhattan metrics, then we need to say that Euclidean uses more demanding mathematical operations, such as the division. This results in its increased time complexity.

Function used by A* to insert a node into the right place consists of two values: the node´s shortest distance from the start, that is found by previous steps of the algorithm and the value calculated by one of three previously mentioned options. The latter value may be quite inaccurate and represents only an estimate.

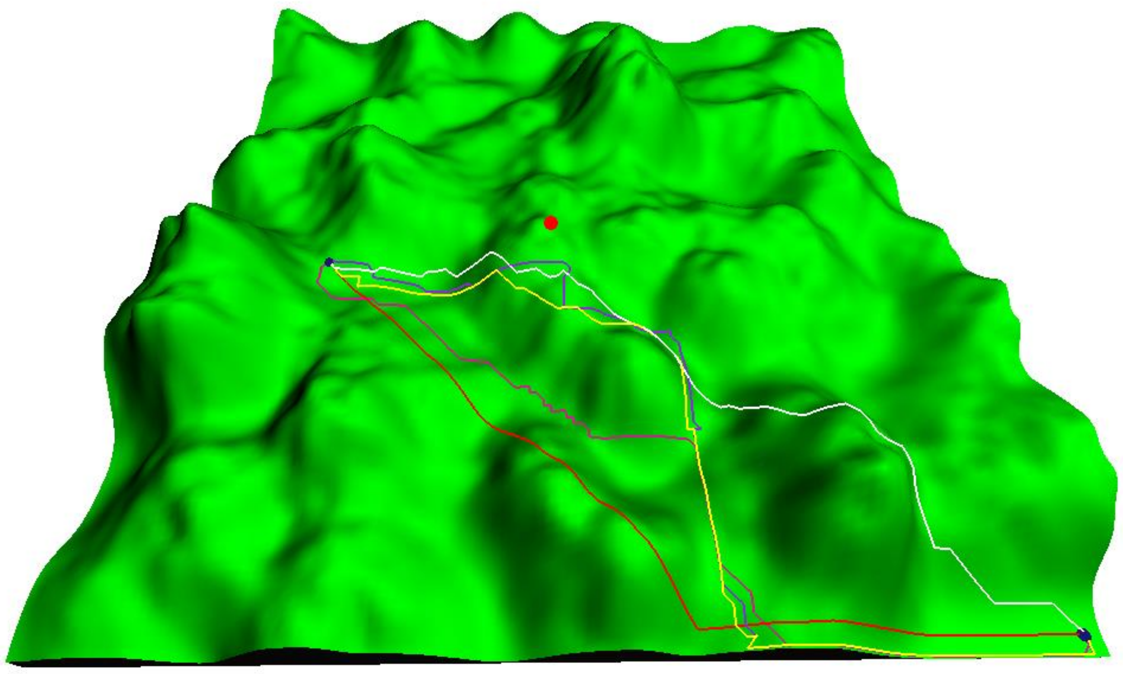


**Figure 7** Found paths

## Comparison of pathfinding algorithms

Pathfinding was executed twenty times on randomly generated terrains. The starting position was in the upper left corner and position of finish was in the bottom right corner.

Dijkstra algorithm always found the shortest path but with larger amount of explored nodes than A*. Due to same location of start and finish and the also fact, that breadth-first search doesn´t use value of edges, it had always same number of explored nodes and same number of nodes of path.

Terms from the two following tables are defined in previous text.

**Table 1** Pathfinding algorithms

| **Algorithm** | **Average number of explored nodes** | **Average difficulty of the path** | **Average number of nodes of path** |
|---|---|---|---|
| Dijkstra algorithm | 15 914.85 | 33.59 | 214.45 |
| A* with Manhattan metric | 578.85 | 67.71 | 242.60 |
| A* with Euclidean metric | 3 171.90 | 49.71 | 227.35 |
| A* with Euclidean metric, where value of edges is 1 | 634.70 | 64.67 | 134.00 |
| Breadth-first search | 127 510.00 | 67.67 | 127.00 |

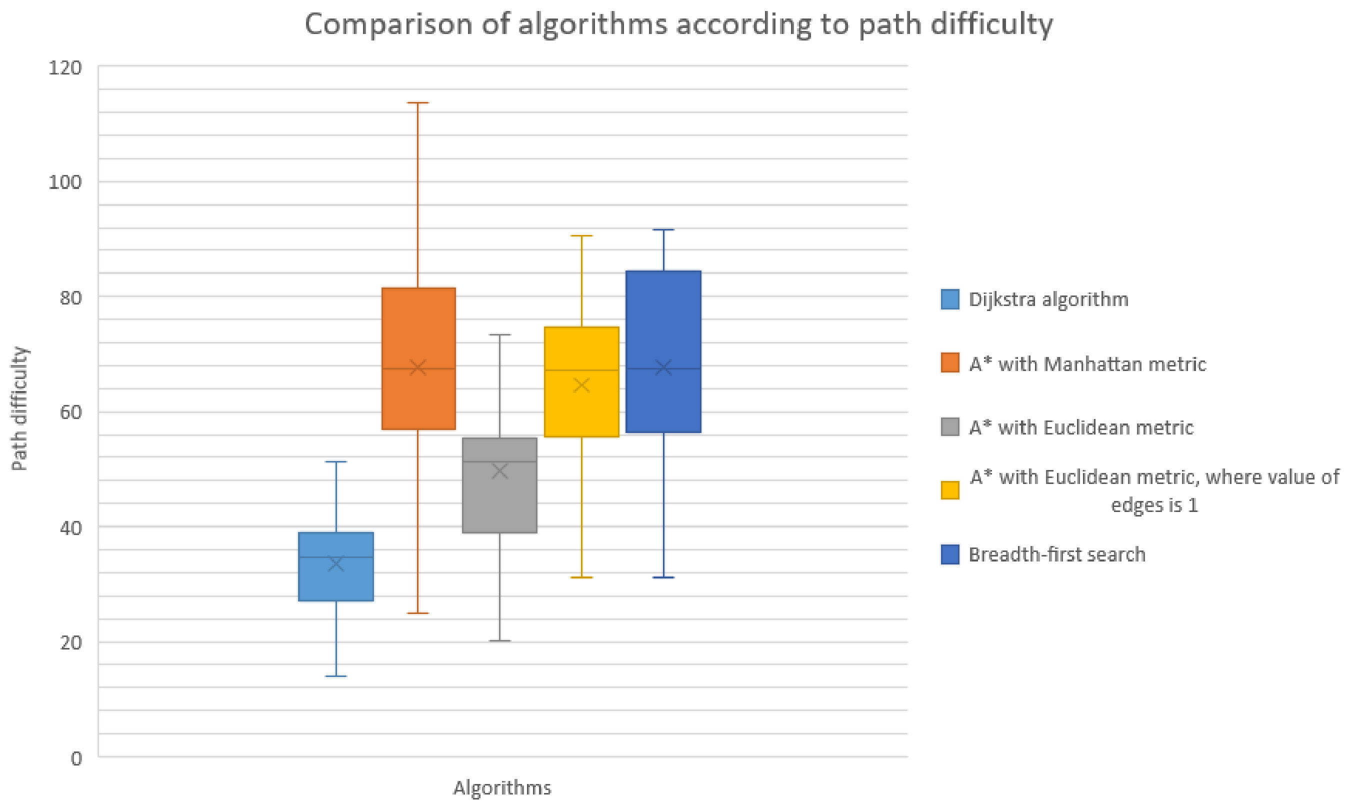


**Figure 8** Pathfinding algorithms according to path difficulty

## Comparison of method for terrain generating

Raw terrain generated by methods presented in the application is quite unrealistic. In the case of the fault algorithm, there are visible places where there was radical increase on the one side and decrease on the other side. This isn´t usual in real world due to erosion and other influences. Therefore, it is necessary to reduce visibility of these visual artefacts by some modifications. Also, the output of hill algorithm looks artificially. Only the value noise doesn´t have such visible visual artefacts.

According to us, the hill algorithm is best suited for editing, because it can handle the greatest amount of editing before its shapes goes flat. This flattening occurs in terrain generated by fault algorithm and value noise much earlier, because these terrains doesn´t have so big differences between heights of vertexes.

After applying modifications provided by the application, terrain becomes more realistic than before. Also, visual artefacts are removed. Suitable application of modifications can, according to us, leads to shapes that really resemble hills, mountains, craters and so on.

Algorithms can be compared according to overall difficulty of terrain. We can obtain this value as sum of all edge values that occur in terrain represented as a graph. Each of algorithms has been launched twenty-one times. Result are demonstrated in following tables.

**Table 2** Terrain generating algorithms

| **Algorithm** | **Average terrain difficulty** |
|---|---|
| **Value noise** | 28118.30 |
| **Hill algorithm** | 28932.99 |
| **Fault algorithm** | 24749.41 |

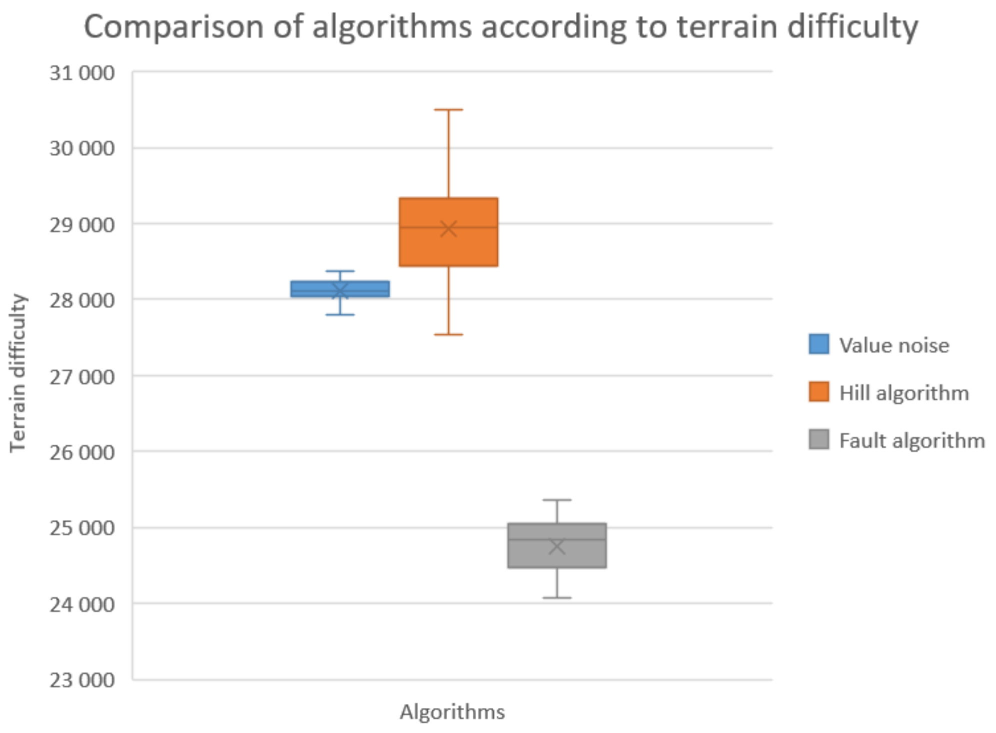


**Figure 9** Terrain generating algorithms according to terrain difficulty

## CONCLUSION

This paper deals with terrain generation and pathfinding in maps for educational purpose. To give students insight into this topic, application was created. This application shows some of mentioned methods for terrain generating, these are value noise, fault algorithm and hill algorithm. It is possible to edit raw terrains by smoothing, dilatation and 2 types of erosion. Pathfinding between 2 nodes is shown by Dijkstra algorithm, breadth–first search and A* with use of Euclidean and Manhattan metric. User can see the text output, that illustratively compares found paths per mentioned criteria. Moreover, the application contains graphical representation of edges between nodes of the graph, in which the path is searched for. These edges have colour that depends on their difficulty. It is possible to hoover camera over terrain. Implemented algorithms for terrain generation and pathfinding have been compared among to chosen evaluating criteria.

## ACKNOWLEDGEMENTS

We would like to thank to people that support creation of this paper, especially to RNDr. Martin Kotyrba, Ph.D. that teaches artificial intelligence on Department of Informatics and Computers, University of Ostrava. This collaboration provided better understanding of students needs and helped to create the fine educational application.

## REFERENCES

Biagioli, A. (2014). Understanding Perlin Noise. [online] Flafla2.github.io. Available at: http://flafla2.github.io/2014/08/09/perlinnoise.html [Accessed 1 Dec. 2017].

Fernandes, A. (n.d.). OpenGL @ Lighthouse 3D - Terrain Tutorial. [online] Lighthouse3d.com. Available at: http://www.lighthouse3d.com/opengl/terrain/index.php?fault [Accessed 30 Nov. 2017].

Gustavson, S. (2005). Simplex noise demistified. [ebook] Available at: http://weber.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf [Accessed 1 Dec. 2017].

Kolář, J. (2000). Teoretická informatika. Praha: Česká informatická společnost.

Nystrom, B. (n.d.). robot frog | 3d | terrain generation tutorial | Hill Algorithm. [online] Stuffwithstuff.com. Available at: http://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html [Accessed 1 Dec. 2017].

Nystrom, B. (n.d.). robot frog | 3d | terrain generation tutorial | flattening. [online] Stuffwithstuff.com. Available at: http://www.stuffwithstuff.com/robot-frog/3d/hills/flatten.html [Accessed 1 Dec. 2017].

Perlin, K. (2002). Improving noise. *SIGGRAPH 2002* Conference. Available at: http://mrl.nyu.edu/~perlin/paper445.pdf [Accessed 1 Dec. 2017].

Russell, S. and Norvig, P. (1995). Artificial intelligence: a modern approach. Englewood Cliffs, N.J.: Prentice Hall.

Žára, J. (2004). Moderní počítačová grafika. 2nd ed. Brno: Computer Press.

Code.google.com. (2011). Fractalterraingeneration - Value_Noise.wiki. [online] Available at: https://code.google.com/archive/p/fractalterraingeneration/wikis/Value_Noise.wiki [Accessed 1 Dec. 2017].