William Raffe
Started: 17/11/2010

Overview of Evolutionary Terrain Prototype

Terrain Creation Parameters

User Specified Parameters:

Parameter Name	Typical Value	Description
Terrain Resolution	E.g. 129, 257, 513	Dictates the resolution of the terrains that will be produced and any sample terrains that are provided to the program
Square Number of Patches	2, 3, 4… 9, 10, etc.	This is the number of patches in a row/column in a generated terrain. This value squared gives the total number of patches. Using this number and the terrain resolution we can get the Base Patch Resolution. As the number of patches increases, the resolution of each patch decreases.
Overlap Percentage	Between 0 - 1	This percentage states how much of each patch is to overlap the patches around it when using the linear interpolation seam removal method. More precisely, this overlap amount is added to the Base Patch Resolution to get the Full Patch Resolution (i.e. the patch plus extra height-map data for an overlap region). In this way, we ensure that no matter what the value of Overlap Percentage is, there will always be Square Number of Patches in each row/column.

Dependent Parameters:

Parameter Name	Calculation
Base Patch Resolution	= terrainResolution / squareNumberOfPatches (rounded down)
Overlap Size	= basePatchResolution * overlapPercentage (rounded down)
Full Patch Resolution	= basePatchRes + overlapSize

Terrain Details:

Patches:

Patches are essentially small sample terrains. They are height-maps of a much smaller resolution that, when combined together, will form a full sized terrain with the dimensions equal to the Terrain Resolution parameter.

Height-map:

A height-map is a 2D matrix with dimensions equal to either the Terrain Resolution parameter (for a full sized generated terrain) or the Full Patch Resolution (for patches). Each value in this matrix specifies a height value between 0 and 1 that is used when rendering the terrain in either Unity or Matlab.

Patch-map:

A patch-map is a 2D matrix with dimensions equal to the Square Number of Patches parameter. Each value in this matrix is an ID that links to a unique patch within the Patch Database. When a height-map is generated, the values in the patch-map are used to query the Patch Database, extract the chosen patches to use their height-map data, sew the patches together (through overlapping), and then save the final full size terrain as a height-map.

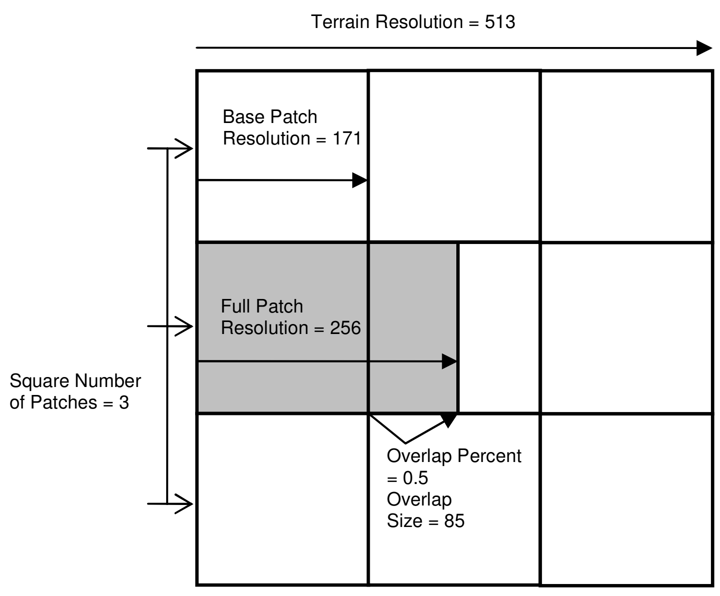

Terrain Resolution = 513
Base Patch Resolution = 171
Full Patch Resolution = 256
Square Number of Patches = 3
Overlap Percent = 0.5
Overlap Size = 85

Fig: Shows all important terrain values

Evolution Parameters

User Specified Parameters:

Parameter Name	Typical Value	Description
Sample Terrains	Array of height-map file names	Each member points to a height-map file. For now, a simple text based file representation is used where each value is a height-map value between 0-1 and the file is laid out in a matrix form with the same dimensions as the Terrain Resolution parameter and is delimited by white space and new line characters.
Population Size	3, 4, 5, 6… etc.	Specify how many terrains are generated and displayed per generation. For now, this has been fixed to a value of 8.
Number of Parents	2, 3, 4… etc	Specify the maximum number of parents that can be selected each generation to produce offspring. For now this value is fixed at 2. See the Evolution Details section below for what happens if less than 2 parents are selected.
Crossover Rate	Between 0 - 1	Given that two or more parents are chosen, this is a probability that crossover will occur on one specific patch. See Evolution Details below for more information.
Mutation Rate	Between 0 - 1	This is the probability that a specific patch will be mutated. See the Evolution Details below for more information

Evolution Details

Initial Population:

The initial population of a run can be provided by the user, loaded from defaults, or randomly generated. It is important to note that either user specified samples or the default terrains are needed to populate the Patch Database. When the program is first launched, patches are extracted from these sample terrains and put into a list/array/database so that they can be used throughout the execution of the program to generate terrains.

Fitness:

The fitness used here is Interactive Fitness. The user can select between 0 and numberOfParents terrains to be parents for the next generation. In the implementation so far, there are allowed a maximum of two parents per generation.

- No Parents: If no parents are selected from the current generation then the next generation is completely randomly generated and it is likely that no traits from the current generation will be passed to the next.
- One Parent: If only one parent is selected then the next generation will consist of terrains that are all mutated versions of that one parent.
- Two Parents: If two parents are selected then all the members of the next generation will be a result of crossover between those two parents. To ensure continuous exploration, once crossover has been done then mutation will also be carried out on each offspring.

It's also important to note that any parents that are selected are propagated to the next generation; that is to say that there will be one offspring that is an unmodified clone for each selected parent.

Crossover:

When two parents are selected, crossover is done to produce all of the members of the next generation. Crossover is done on a patch by patch basis; that is, each patch of each child is tested to see whether it under goes crossover. The process for crossover for each offspring/child when only two parents are involved is as follows:

1. First, out of the selected parents, one is chosen to be the template/base for the child. The child is now an exact clone of that parent. This parent can potentially be chosen randomly so that each child can any of the parents as its base parent (leads to less predictable offspring) or be fixed so that every offspring of that generation has the same base parent (depending on the Crossover Rate, this will lead to all offspring uniformly being more similar to one parent or the other).
2. Once a base parent is selected and its patch-map copied to the child, each patch in the child's patch-map is given a crossover probability.
3. If this probability value is less than the Crossover Rate parameter then the patch is swapped for the patch in the exact same position on the other parent's patch-map.

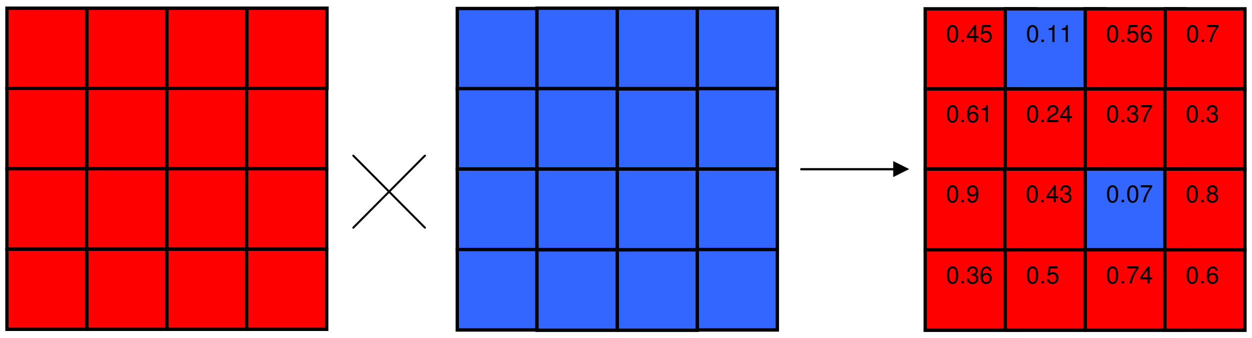


Fig: Shows the result of crossover between two parents, indicated as red and blue. This diagram shows patch-maps with a Square Number of Patches parameter value of 4. In this example, the Crossover Rate is set low at a value of 0.2. The red parent is selected as the base parent. The crossover probabilities are shown for each patch in the child on the right. If the crossover probability is less then 0.2 then the patch is switched for patch in the same position from the blue parent, otherwise it is kept as a red patch.

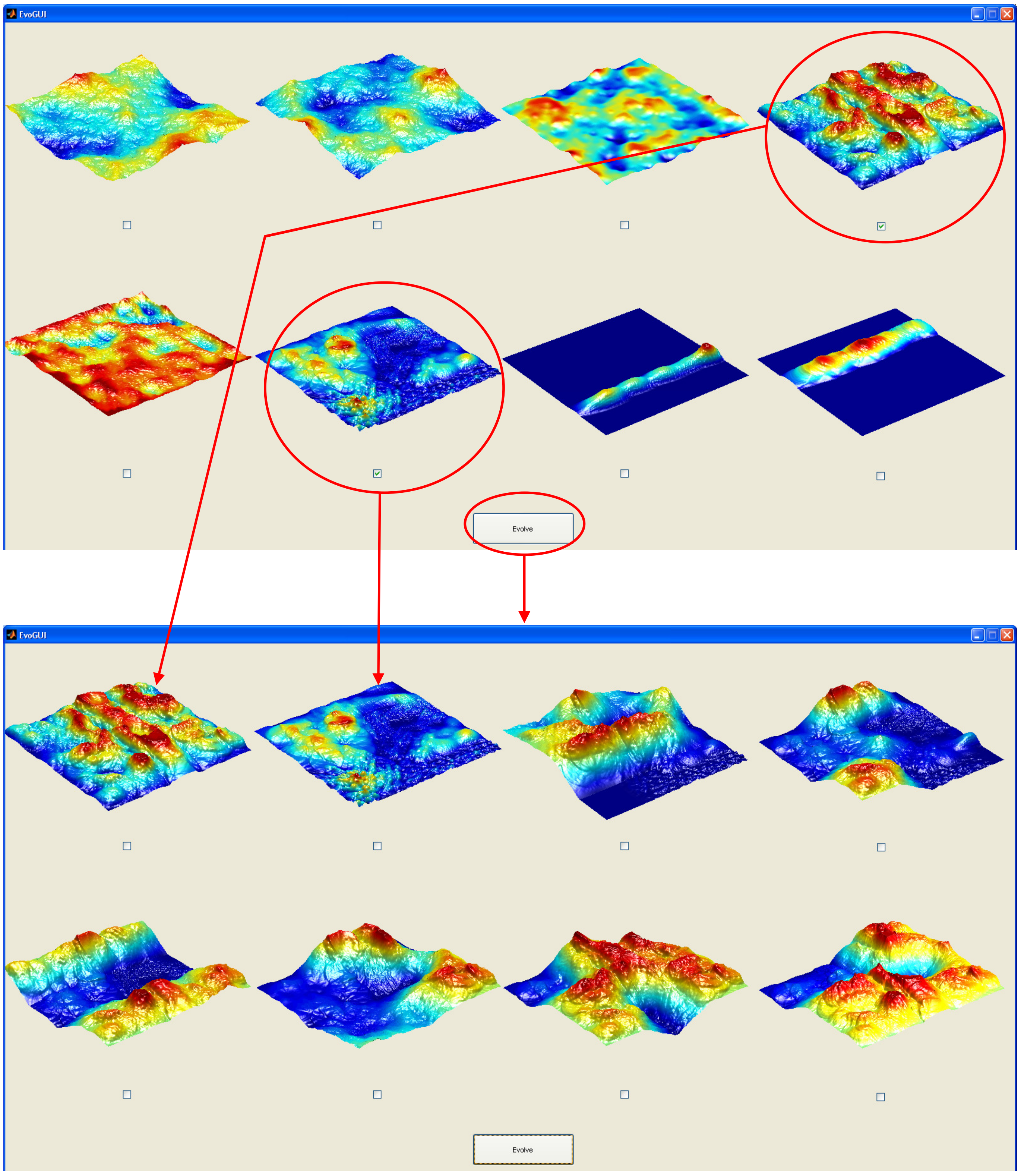
Mutation:

Mutation is carried out to introduce randomly selected patches into an existing patch-map. This process is done once a child is established, either after cloning the parent if only one parent is selected by the user or by after the crossover process if two parents have been identified. Again, this is a patch by patch algorithm that assigns each patch in the child's patch-map with a mutation probability. If this probability is less than the Mutation Rate parameter then the patch is removed and replaced with another that is randomly selected from the Patch Database. Note that if the Mutation Rate is equal to 0, every child will be a direct clone of the parent (if only one parent is used) and, conversely, if the Mutation Rate is equal to 1 then all children will be constructed completely randomly and have no similarity to the parents.

Implementation Evaluation

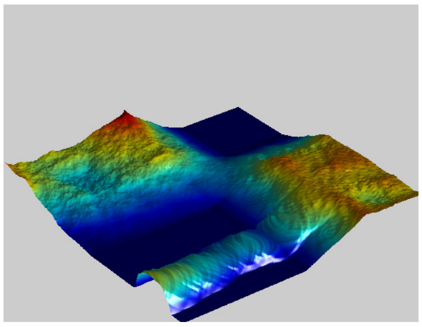
Overview of GUI:

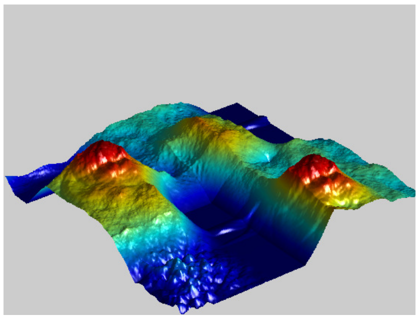
This program is fully implemented in Matlab. The GUI below shows how the user selects parents by clicking on the check boxes under each terrain and then presses the Evolve button to proceed to the next generation. As mentioned before, the use can select between 0 and numberOfParents parents (in this case, between 0 and 2 parents). In the next generation, the first few terrain in the top left are exact clones of the parents. The screen shot below also shows the default terrains provided in the implementation that were created in the Unity Game Engine terrain editor.

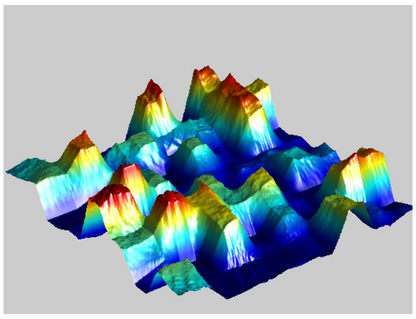


Changing the Square Number of Patches Parameter:

In the images below, the Square Number of Patches parameter is changed while all other parameters are kept constant. Randomly generated patch-maps are used for all of the terrains bellow. Terrain Resolution = 513, Overlap Percentage = 0.5.


sqrNumPatches = 2

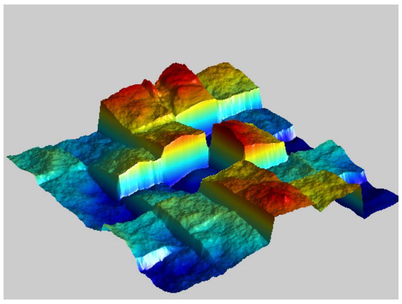

sqrNumPatches = 4

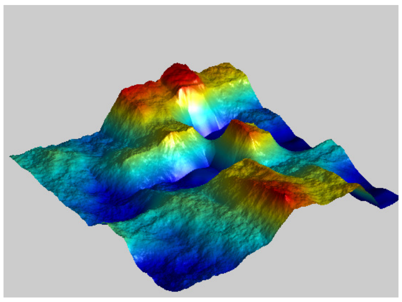

sqrNumPatches = 8

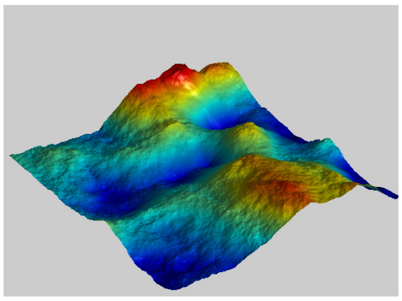
As can be seen from these images, increasing the number of patches results in more visible transitions between patches. Also, without any constraint for where patches should be placed, as more patches are used to represent the terrain there is a greater chance of discontinuation of features between two adjacent patches, resulting in jagged and illogical terrain.

Changing the Overlap Percentage Parameter:

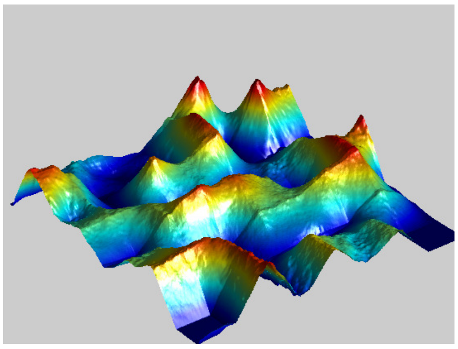
These images show the affect of changing the Overlap Percentage parameter while keeping all other parameters static. The same patch-map is used for all three of the terrains below. Terrain Resolution = 513, Square Number of Patches = 5.


overlapPercentage = 0.1


overlapPercentage = 0.5


overlapPercentage = 0.9

As predicted, overlapping patches more allows for more seamless flow between adjacent patches. However, overlapping the patches more does not solve all problems. For example, as we saw ealier, increasing the number of patches leads to discontinuous feature alignment between patches and an illogical terrain as a result. The image below shows what happens when you have a high overlap percentage value with a large number of patches.

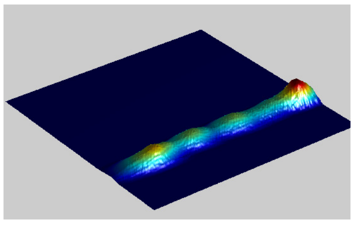
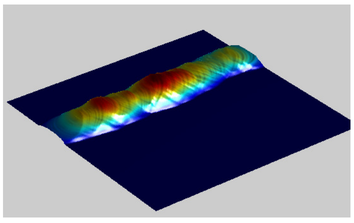

sqrNumPatches = 8, overlapPercentage = 0.9

While the peaks of the features look far more natural and there are smoother transitions between patches, there is still an irregular flow and no coherency among terrain features.

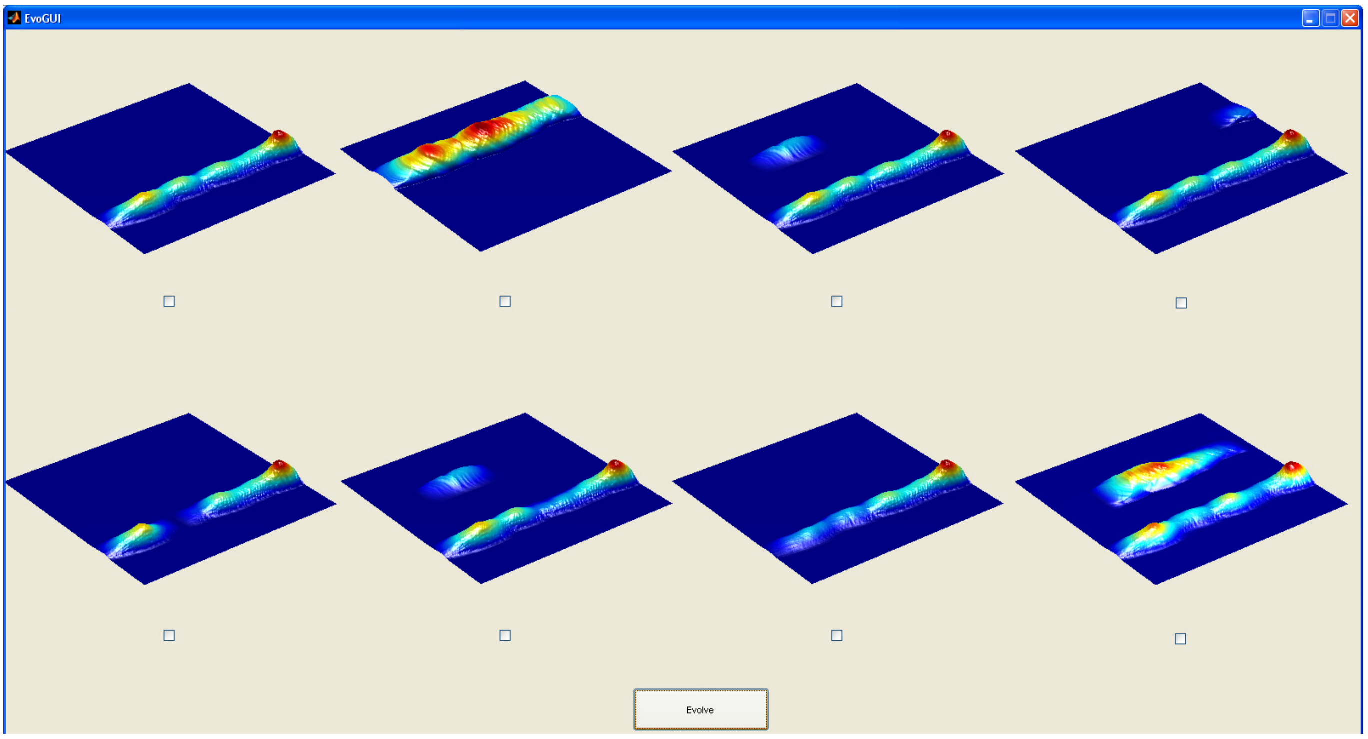
Changing the Crossover Rate Parameter:

The following experiment shows the results of changing the Crossover Rate and leaving all other parameters static. The two parents selected have mountain range features in different positions with zeroed height-map values surrounding them. These types of parents are chosen because the offspring will clearly show from which parent each patch is chosen for the mountain feature areas. In these results the Base Parent selection is fixed to be the first parent shown below; that is to say that every offspring will have an initial patch-map template of the first parent shown. Terrain Resolution = 513, Square Number of Patches = 5, Overlap Percentage = 0.9, Mutation Rate = 0.

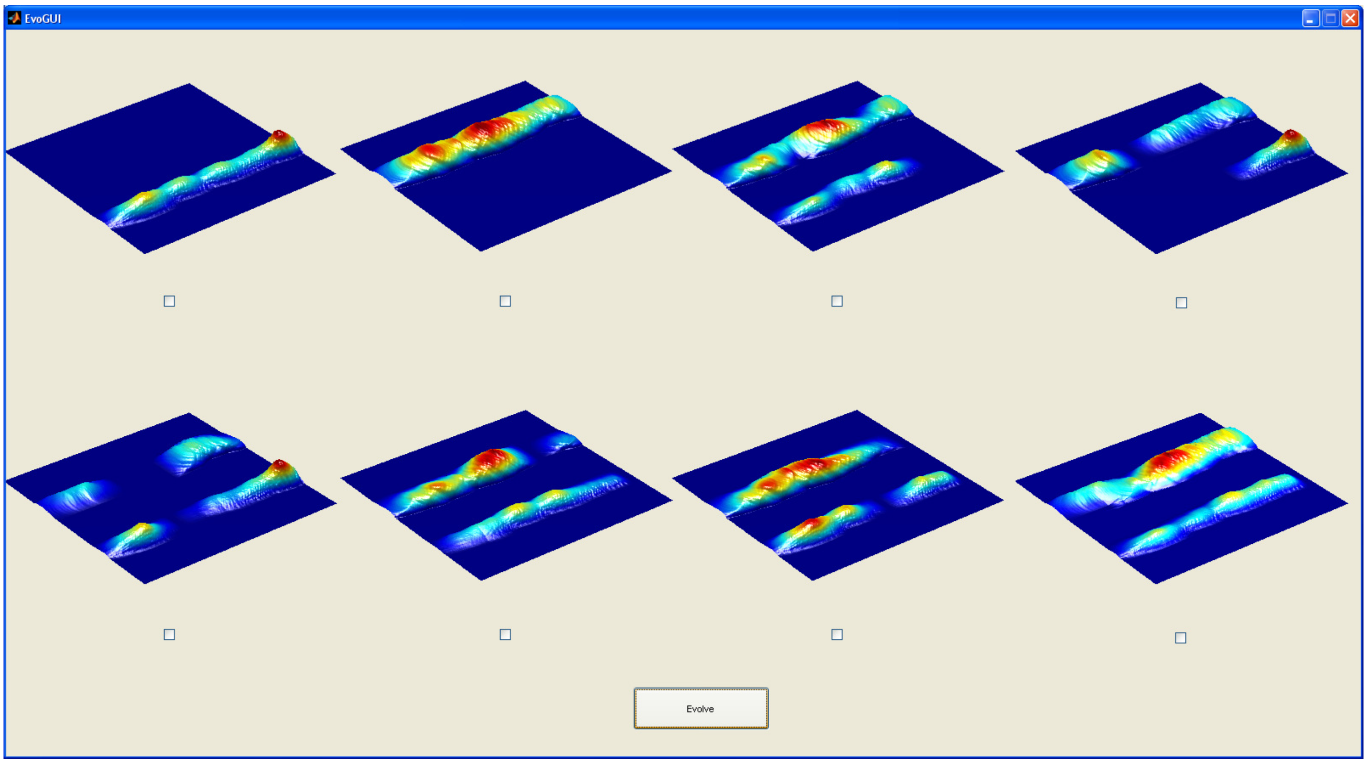
Parents:

 and 

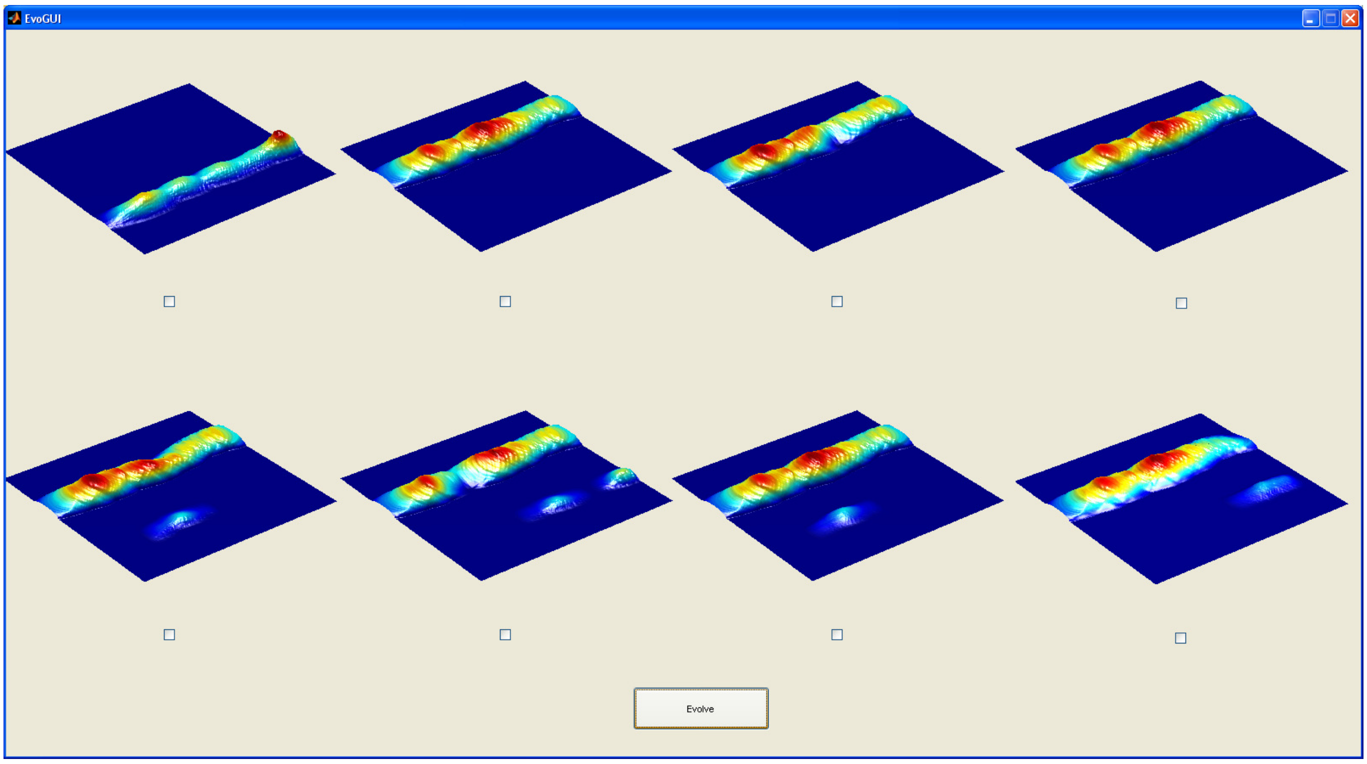
Crossover Rate = 0.1



Crossover Rate = 0.5



Crossover Rate = 0.9

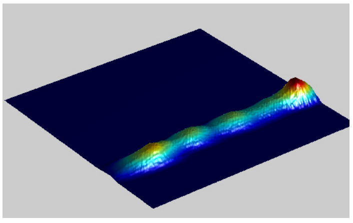


As expected, with a low Crossover Rate, it is less likely that crossover will occur so the patches from the base parent are kept, whereas when the Crossover Rate is high there is a higher chance of crossover occurring so the offspring mostly show traits of the other parent. If the Crossover Rate is in the middle we expect to see features from both parents apparent in the offspring or even no features at all if blank flat lying patches are selected from each parent.

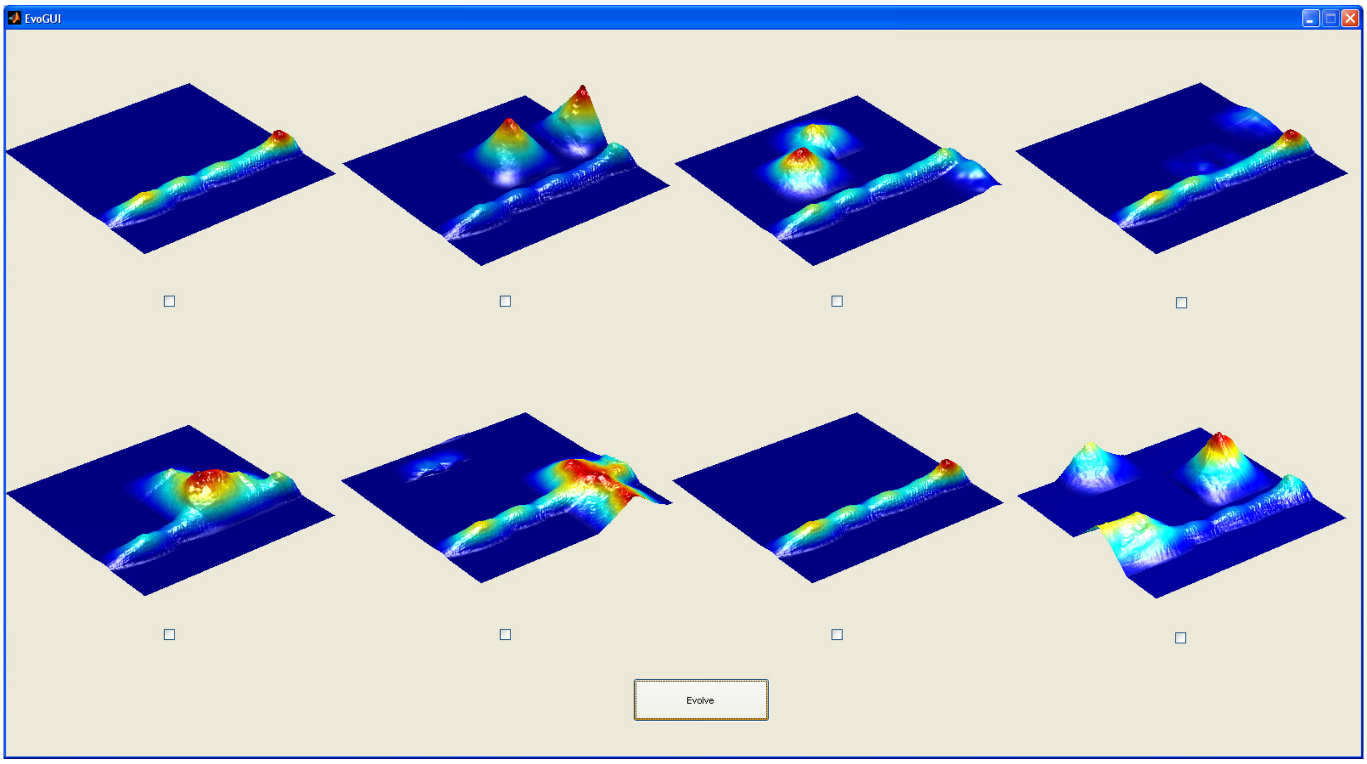
Changing the Mutation Rate Parameter:

Terrain Resolution = 513, Square Number of Patches = 5, overlap Percentage = 0.9, and because only one parent is being used the Crossover Rate is irrelevant.

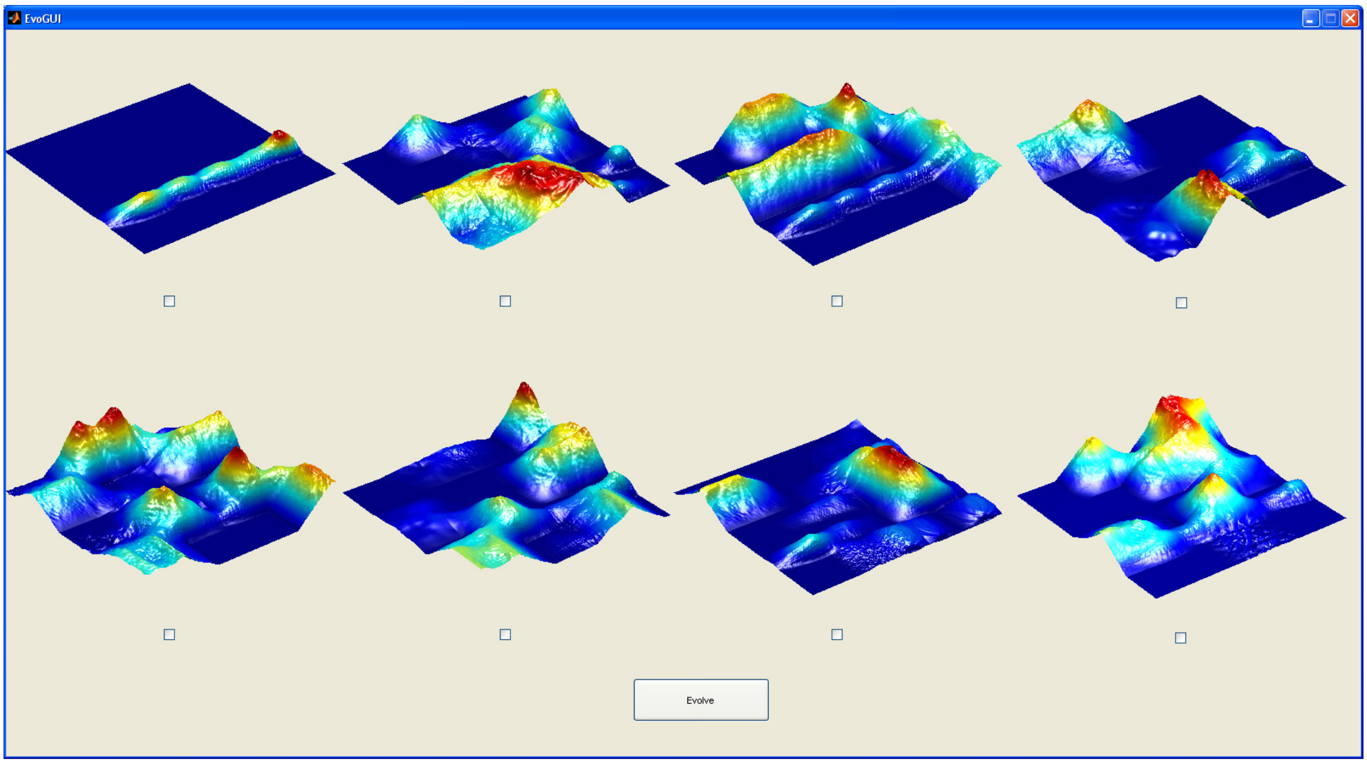
Parent:



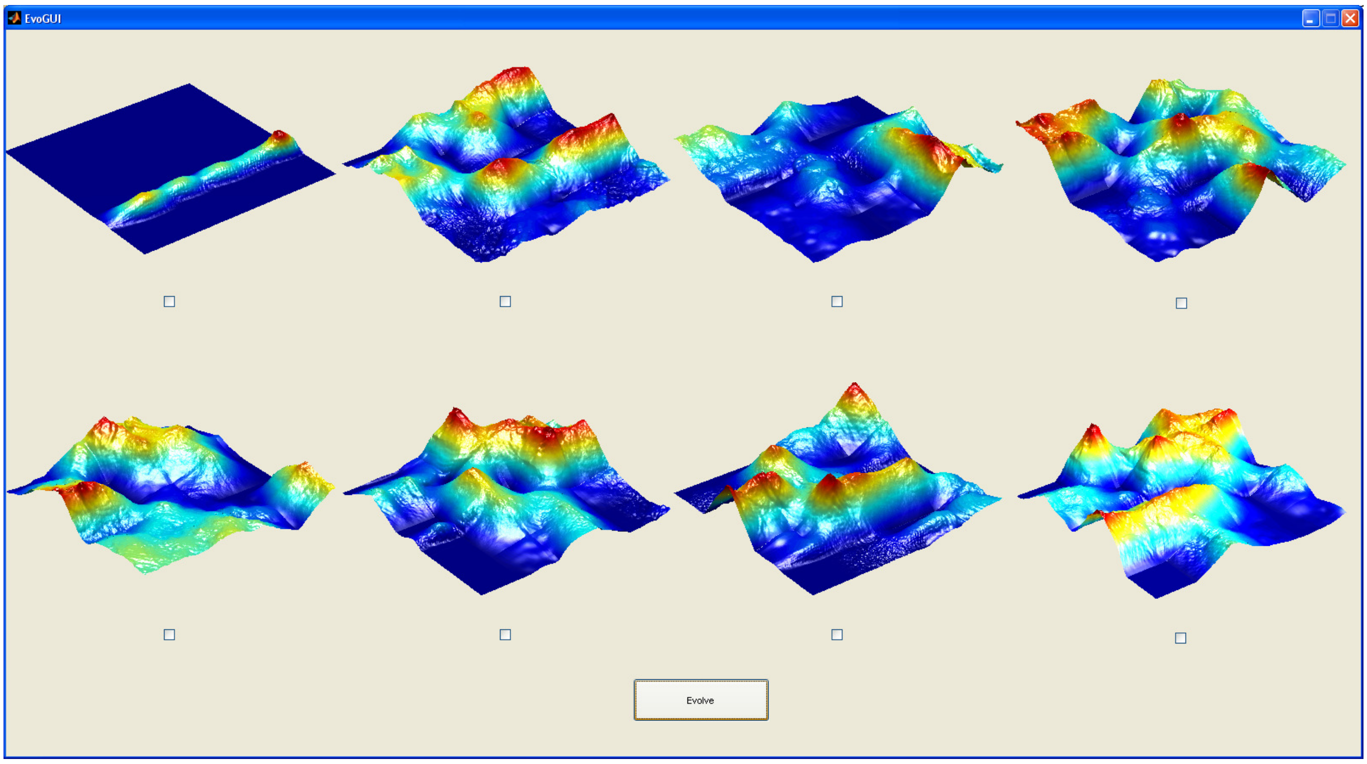
Mutation Rate = 0.1



Mutation Rate = 0.5



Mutation Rate = 0.9



These tests show that with a low Mutation Rate most of the offspring bare a close resemblance to the chosen parent. Setting the Mutation Rate at a middle value changes most of the offspring quite significantly but some of the mountainous feature patches and flat patches of the parent still remain. With a high Mutation Rate, none of the offspring bare any resemblance to the parent and could have in fact been randomly generated. Thus, to ensure exploitation and prevent over exploration, the Mutation Rate should not go above 0.5. In many cases, it would be most suitable to use a Mutation Rate value of around 0.2 or less.

Algorithm Efficiency:

While I have not formally experimented as of yet to gain solid values, it appears that the terrain creation algorithms work extremely quickly. Evolving eight patch-maps with crossover and mutation or simple randomization and then creating height-maps from those patch-maps only takes a matter of milliseconds.

The slowest parts of the program execution are loading in the sample terrains when the program first launches and then rendering the eight height-maps through the Matlab GUI interface. The loading of the sample terrains from file is a minor problem which may be fixed by storing the files as raw byte code rather than in a text file. The terrains are rendered in Matlab as three dimensional graphs with some lighting effects and it seems that Matlab struggles with performance when rendering eights highly detailed graphs (at least on my office desktop, I have not tested it in the games labs yet). This may be solved in the future by using this program on a more powerful computer or by exporting the code from Matlab and re-creating it in another, much lower level, language such as C++ that can plug directly into the DirectX graphics API or writing it completely in Unity. For the time being, writing this code in Matlab was the quickest way to get a functional solution framework in place, however if we want to extend the research in the years to come to handle object placement on the terrain then the code will need to be re-written into a game engine or work directly with the DirectX API.