# Automatic Terrain Generation with a Sketching Interface

Anna Puig-Centelles
Universitat Jaume I,
Castellón, SPAIN
apuig@uji.es

Peter A. C. Varley
Universitat Jaume I,
Castellón, SPAIN
varley@uji.es

Oscar Ripolles
Universitat Jaume I,
Castellón, SPAIN
oripolle@uji.es

Miguel Chover
Universitat Jaume I,
Castellón, SPAIN
chover@uji.es

**ABSTRACT**

Virtual environments should offer the user a deep interactive experience with both large worlds to explore and a higher degree of perceived realism. The main goal of our work is to provide the final user with an easy-to-use accurate terrain generation application, which allows non-professional users to design their own desired terrain. In this paper we consider the creation of islands to be used in computer games. We introduce a simple terrain algorithm and we also consider its integration into a sketching application. The application will offer both a 2D and a 3D representation of the terrain, in order to simplify the interface and provide the user with more interactive feedback about the island that has been designed. Our framework offers real-time algorithms for both creating and modifying terrain features, thus improving the final results with more realism and greater customization by the user.

**Keywords:** Terrain Generation, Sketching Inteface, Islands, Game Environment.

## 1 INTRODUCTION

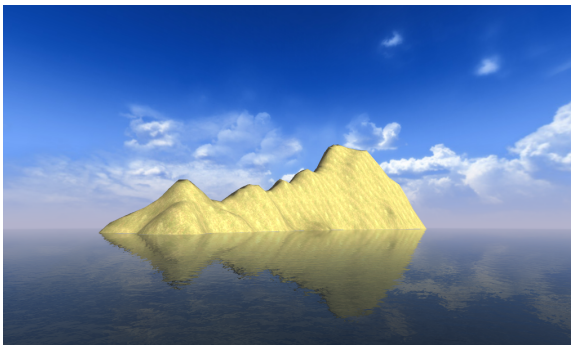


Figure 1: Example of a terrain obtained with our framework and imported into the Torque Game Engine.

In recent years, computer graphics have undergone an intense evolution as new graphics hardware offers a final image quality that was unimaginable just a few years ago. As a result, interactive graphics applications, such as computer games or virtual reality environments, now include more complex scenes offering very detailed environments. Terrain is therefore a key element that can lessen the sense of realism if it is not addressed correctly.

Terrain generation is a research area which has been active for many decades. The growing power of modern computers has made them capable of producing increasingly more realistic scenarios. Synthetic terrain generation is a process which creates elevation values throughout a two dimensional grid. The need for highly realistic scenarios often involves developing algorithms that can generate more realistic terrains with more user control over the final terrain that is created. Different terrain generation techniques that are capable of offering very realistic artificial terrains have been reported in the literature. Nevertheless, not many applications provide enough user control. On the contrary, those applications that provide user interaction are often too difficult to control.

Sketching is a tool that is well suited to the design of architectural elements and it provides the user with a considerable amount of control over the created elements. Research has produced prototype tools for interpreting sketches of abstract polyhedra [23, 11]. However, less work has focused on sketching the underlying terrain or extracting it from a photograph. Thus, buildings are often considered as the *foreground* and are taken into account properly, whereas terrain is seen as *background* which is often ignored.

Our aim is to develop convenient and simple ways to create computer models of terrain. In this paper we address the problem of creating models of islands for use in computer games. Our goal is similar to the idea given in [12], where the authors show that relatively simple algorithms can provide non-professional users with fast, successful results.

In this work we describe a terrain creation algorithm for islands based on *heightmaps*, which are regularly-spaced two-dimensional grids of height coordinates. The elevation of the terrain is automatically calculated from the coastline sketched by the user, who can also create hills and apply perturbations in order to achieve a more realistic and irregular terrain. Once again, it is



 ISBN 978-80-86943-94-7

important to mention that our aim is to provide the user with more control over the terrain that is generated.

Furthermore, we also consider the integration of our algorithm into a sketching application. This application combines a 2D representation window and a 3D displaying window. First, the user creates a silhouette of the island in the 2D window and then, the user will be able to modify the terrain appearance in the 2D and the 3D windows. The terrain thus obtained will be outputted as a heightmap that may then be imported into a game engine.

This work is organized as follows. Section 2 contains the state of the art in terrain generation and sketching freeform surfaces. Section 3 describes our terrain generation algorithm. After that, Section 4 analyzes our sketching application. Later, Section 5 depicts our results and discusses a usability test. Finally, Section 6 presents our conclusions and future work.

## 2 RELATED WORK

In this section we analyze the different approaches that currently exist for terrain generation. After that we will take into account the different software tools which are available for creating artificial terrain. Finally, we will give some basic ideas on sketching and its application to our purposes.

### 2.1 Terrain Generation

The literature offers a wealth of research on synthetic terrain generation. Techniques can be grouped into three different categories:

**Procedural Approaches.** This category includes methods in which the terrain is generated automatically. These methods can be further separated into fractal techniques and physically-based techniques.

The most popular procedural approach is fractal-based terrain generation, which is efficient but difficult for users to control. It is possible to find a review of recent fractal approaches in [2].

Physically-based techniques simulate the effects of physical processes such as erosion by streams [8] or wind [24]. A recent technique that combines a non-expensive fluid simulation with an erosion algorithm is presented in [1]. It also supports effects like dissolving or sedimentation of material in the process of erosion.

Fractal landscape terrain generation and physical erosion simulation are both approaches that add terrain details through procedural refinement. Nevertheless, modifying their parameters to obtain a desired terrain may be a painstaking task.

Another proposal appears in [15], where the authors provide an alternative method for terrain generation that employs a two-pass genetic algorithm approach to produce a variety of terrain types using only intuitive user inputs. The process is efficient but very difficult for a user to control.

**Real Terrain Information.** This approach groups the techniques from the **Geographic Information Systems** (GIS), where elevation data come from real-world measurements [25]. Similarly, another source of information could be the study of the extraction of terrain from photographs [3]. All these approaches have the advantage of offering highly realistic terrains in very little time, but with little user control.

**User Defined Approaches.** This is the most flexible type of technique, in which a human artist creates the terrain manually, using an image editing program, 3D modeling software, specialized terrain editor programs or the editors that are included in game engines.

The authors of [10] allow the user to control the terrain generation process by receiving as input an image generated by an image editing program in order to perform the terrain generation. In [26], patches from sample terrain (represented as a height field) are used to generate new terrain and the synthesis is guided by a user-sketched feature map that specifies where terrain features occur in the resulting synthetic terrain. Later, [22] introduced a simple interface for sketching heightmaps of islands. This application was very simple but, although it offered a good amount of user control, it was difficult to use and the obtained terrain was not completely customizable.

### 2.2 Terrain Software

In this section we introduce some terrain tools for simulating artificial environments. There is a wide range of software available. In Terragen [19] and Terraineer [20], the user sets parameters and the program creates a pseudo-random landscape which meets those parameters. Terraineer offers the possibility of experimenting with different height generation algorithms. World Machine [17] additionally includes modeling of physical weathering processes. In all of these programs, terrain is modeled and imported/exported as a heightmap.

Further user interaction is offered in Nem's Mega 3D Terrain Generator [5], where the user is initially presented with a flat piece of terrain and has various options for modifying it. L3DT [21] is another software that generates artificial heightfields and exports its data to multiple formats.

### 2.3 Sketching

There have been some recent developments on the automated interpretation of freeform surfaces from sketches, but they either interpret the drawing as being that of a single solid object [7] or leave the freeform surface floating in mid-air as a patch, without continuing to the horizon in the manner of a landscape [18, 6].

However, interpreting a terrain sketch as a floating free-form patch presents some problems. Qin et al [18]

require the user to draw a grid of quadrilaterals to represent the surface, and a trained neural network is also needed to interpret it. In Kaplan and Cohen's approach [6] the boundary of the patch must be clear in the original input, either by virtue of its obvious contrast with the background or by being specified by the user. Their approach also requires user intervention to resolve ambiguities. The problem of boundary constraints can be overcome if we restrict ourselves to sketching islands. The boundary constraints for islands are simple: an island has a coastline, that is, a continuous bounding curve.

More recently, a system for designing freeform surfaces with a collection of 3D curves has been proposed in [13]. They create objects within an easy interface, which is based on drawing simple lines. By using a similar simple sketching interface, [27] introduce an over-sketching application for feature-preserving surface mesh editing. This application allows simple yet realistic mesh deformations to be obtained.

# 3 OUR TERRAIN GENERATION

The method that we present here for generating islands is based on the use of heightmaps and allows users to define and modify the coastline of the island. Furthermore, they can also create and reshape any number of hills, which will interact each other and with the existing terrain. Finally, we offer the possibility of applying filters to give the final terrain a more realistic appearance.

## 3.1 Reshaping the Coastline

The user can draw the silhouette of the island freely, but it will be necessary to delineate a continuous curve by sketching a closed shape, as shown in Figure 8. The shape of the coastline can be changed by redrawing, starting and ending at points near the existing coastline and drawing a continuous curve in any direction between those two points. Thus, it is possible to apply different operations in order to modify the existing coastline.

The user may decide to cut a piece of the island off. As a consequence, the terrain will be split into two areas. Depending on the direction of the cut, the algorithm will decide which one of these areas is to be rejected. The direction of the cut is understood as being the direction running from the initial to the finish point. The rejected area will be the one on the left-hand side of the cut. Figure 2 presents an initial coastline and the silhouettes obtained after performing cuts with the same start and finish points but following different directions.

Furthermore, the user can also add new pieces to the existing area. Again, the algorithm will behave differently depending on the direction of the sketched drawing. If the line has been sketched clockwise, the new area will be added to the existing one. In contrast, if the line has been performed in an counterclockwise direction, then this new area will be maintained and the old one will be rejected. In Figure 3 we can see an example of these possible ways of modifying the area by adding or subtracting a piece of terrain.

We must note that the algorithm will differentiate between cut and supplement operations by testing whether the line goes through the terrain area or not. In those cases in which a line is used to perform more than one operation, each point where the line intersects the silhouette will be interpreted as the finish and start points of the consecutive operations. In Figure 4 we depict an area that is being modified by two consecutive operations: the first one consists in an external clockwise supplement and the second one is a curved internal cut that is rejecting the piece on its left-hand side.

## 3.2 Updating the Terrain Height

Every time the coastline silhouette is modified, the terrain algorithm has to react adequately to those changes and recalculate the height of the terrain in order to offer a smooth continuous surface.

Since we are simulating the terrain of an island, we must take into account the level of the sea. We have to ensure that every single point within the sketched coastline is above sea level. As a consequence, when the coastline changes, it may be necessary to modify the elevations of some onshore points. Ideally, points close to parts of the old coastline which remain unchanged should also remain unchanged, but points close to parts of the new coastline should be elevated above sea level regardless of their previous height. Therefore, if we reshape the coastline then we have to check whether all the points contained inside the island have the appropriate height.

In order to obtain the new height values, we take into account the distance from each point to the nearest piece of new coastline $D_n$ and to the nearest piece of old coastline $D_o$, both scaled to the range 0 to 1.

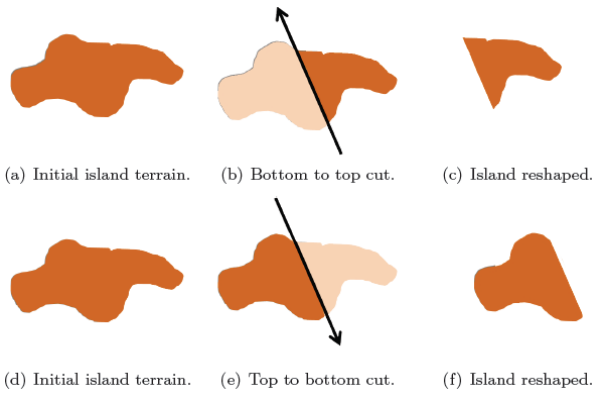
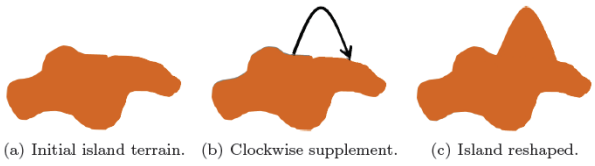
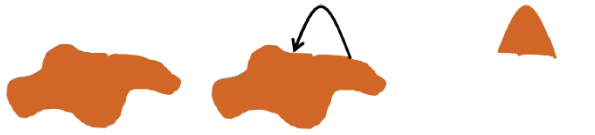



Figure 2: Cutting and reshaping the island.

 ISBN 978-80-86943-94-7



(a) Initial island terrain. (b) Clockwise supplement. (c) Island reshaped.



(d) Initial island terrain. (e) Anticlockwise supplement. (f) Island reshaped.

Figure 3: Supplementing and reshaping the island.

(a) Initial island terrain. (b) Combined operations. (c) Island reshaped.

Figure 4: Two consecutive operations.

The height of each onshore point $H_i$ is calculated as a weighted value between the old height $H_o$ and the new one $H_n$, the latter being proportional to $D_n$. We implement the calculation of the heights with Equation 1,where the weight $W$ is calculated by Equation 2.

$$H_i = H_o(1.0 - W) + H_n W \quad (1)$$

$$W = 0.5 + 0.5 tanh((D_o)^2 - D_n) \quad (2)$$

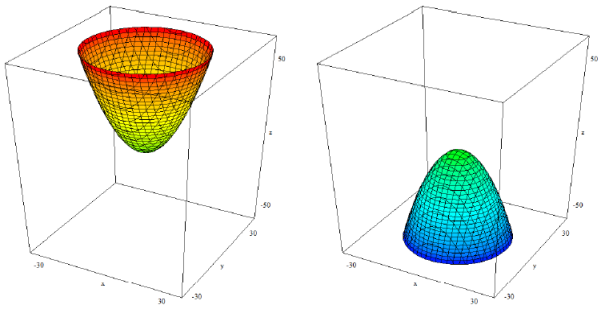
The hyperbolic tangent ($tanh$) function is chosen because it has the appropriate shape, which is close to $-1$ for points near the old coastline and close to 1 for points near the new coastline. Moreover, it never goes outside this range. $D_0$ is squared so that points close to neither coastline are treated as being closer to the old coastline rather than to the new one.

Whether a pixel is onshore or not is assessed by referring to a silhouette of the island which is recalculated after each change to the coastline by drawing the coastline on a blank array of pixels and using a flood-fill routine (starting from a point clearly outside the coastline) to distinguish sea from land.

## 3.3 Generating Hills

In previously presented methods for sketching islands [22], the orography was difficult to define. In our work we want to improve on the user definition of terrain. The idea is to allow the user to create multiple hills having the desired radii, height and location on the terrain.

In our algorithm we define hills as *elliptic paraboloids*. An elliptic paraboloid is shaped like an oval cup and can have a maximum or minimum point. In a suitable coordinate system, it can be represented by the equation:



(a) Elliptic paraboloid upwards. (b) Elliptic paraboloid downwards.

Figure 5: Example view of elliptic paraboloids.

$$\frac{z}{c} = \frac{x^2}{a^2} + \frac{y^2}{b^2} \quad (3)$$

considering that the *elliptic paraboloid* is centered on $0,0,0$ with radius $a,b$ (along the x and y axes), being $a,b \in \Re$ and $a > b$.

Equation 3 represents an elliptical paraboloid which opens upwards and can be seen in Figure 5a. This quadratic surface will be used in our algorithm to define the hills. It is important to note that we allow the user to define valleys by means of elliptical paraboloids which open downwards, as can be seen in Figure 5b.

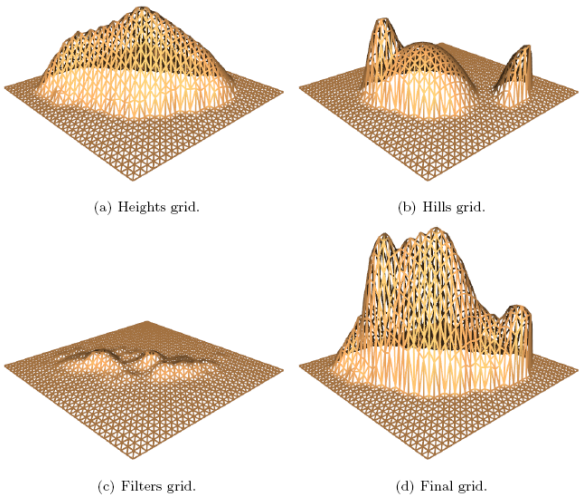
With this equation the user can introduce the central point, the radius and the height of each hill. Once we have this information, our algorithm will be able of calculate the height of each point affected by the hill. All those points are obtained with the central point and the radius that have been defined. The height of each single point will be modified by following Equation 3. We will add the new height to the previous one in order to obtain more realistic and integrated terrains. By so doing, we allow for the creation of valleys and volcanos.

## 3.4 Filtering the Terrain

In order to obtain a better appearance for the terrain being designed, we can introduce some fuzzy bumps to deform the regular surface. We have implemented a filter to introduce 'noise' into the previously defined rounded terrain. This filter can be applied as many times as the user desires and it will give us a number of perturbations that are proportional to the surface area of the island. These perturbations will also have an elliptical paraboloid shape, but they will be wider than taller and they will be produced upwards or downwards in a random manner.

## 3.5 Integrating the Processes

The final terrain that is visualized comprises three heightmaps that will be added one after the other.



(a) Heights grid. (b) Hills grid. (c) Filters grid. (d) Final grid.

Figure 6: Sample composition of the *Final Grid* of heights by adding the previously updated grids.

These heightmaps are stored as grids (2D matrix) of floats:

- *Heights Grid*, which contains the heights of the terrain obtained after modifying the coastline.
- *Hills Grid*, which stores the increments or decrements in height, due to the hill volumes.
- *Filters Grid*, which holds the variations introduced by the filters that have been inserted.

As a consequence, with those three data structures we obtain the *Final Grid*, which stores the sum of the three previous ones. We assume that the *Heights Grid* defines the basic features of the terrain. Then, the other data structures will add more details. It is important to comment that the division of the terrain information into those three data structures simplifies the updating process of any of them. This way, for example, adding a hill only involves modifying the *Hills* grid.

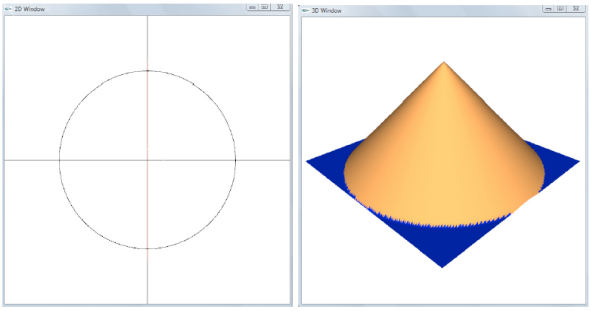
In Figure 6 we can see an example of the different grids that compose the *Final Grid*. The *Heights Grid* is obtained after defining the coastline of the island. The *Hills Grid* contains three different hills. Lastly, the *Filters Grid* stores the perturbations introduced by the user. Consequently, these three grids combined together give form to the final terrain.

This representation uses an internal triangle mesh for representing the 3D island. The 2D heightmap is properly linked with this 3D representation in order to allow fast and efficient updates. The implementation has been optimized to assure that each modeling operation entails to update the minimum amount of information, including both the heightmap and the vertices information: spatial coordinates, normals, colors, etc.

# 4 USER INTERFACE

This section describes our sketching application for terrain generation by using the ideas presented earlier. In paper [22], a simple interface for sketching heightmaps of islands was presented. This interface was close to the ideal of a modeless single-tool interface, with all of its major operations being controlled by a single device (pen or single-button mouse). A problem that appears in some of the most advanced sketching applications, like [16], is that they require a multi-modal push-button interface. Our intention is to maintain the original sketching objectives in order to keep our application as simple and natural to use as possible. Nevertheless, it has been necessary to develop a two-button mouse software application to integrate all the functionalities presented in the previous section.

Our proposed framework offers the user an interactive sketching application. This solution consists of two windows. The 2D window depicts the silhouette of the coastline of the island, as seen in Figure 7a. The 3D window represents the volumetric view of the whole island, as seen in Figure 7b. This 3D view presents a smooth surface which is automatically constructed with the information stored in the heightmap. When the implementation starts, the 2D window contains a circular coastline, as shown in Figure 7a. The 3D window, shown in Figure 7b, depicts a conical island, which is the initial terrain that the user will be able to modify.



(a) 3D window. (b) 2D window.

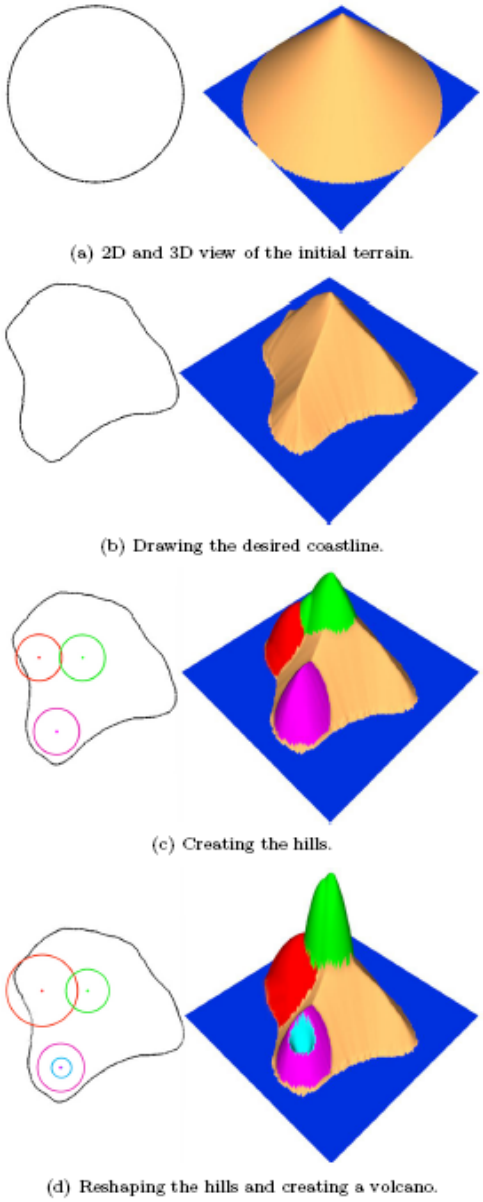
Figure 7: 2D and 3D Window on Startup.

In the following subsections we will provide a detailed description of the interaction with the aforementioned windows in our application. In Figure 8 we describe a step-by-step design of an island using our framework.

## 4.1 2D Modeling Operations

The 2D window allows the user to perform two basic sketching operations: defining the silhouette of the island and adding and modifying hills.

When interacting with the left mouse button in this window, the user is allowed to design the coastline of the island. It is possible to draw a free-form silhouette



(a) 2D and 3D view of the initial terrain.

(b) Drawing the desired coastline.

(c) Creating the hills.

(d) Reshaping the hills and creating a volcano.

Figure 8: Designing a sample island in four steps.

interactively and the system will simultaneously update the terrain. As we have mentioned in previous sections, it is possible to cut and extend the existing terrain by defining lines that start and end on the coastline. Figure 8 shows how we could draw any irregular shape to delimit the terrain of our island. Our system includes an additional feature, which has proved popular with users: when a change is made to the coastline, the old coastline gradually fades away, taking about two seconds to do so. Pressing the right mouse button during this period removes the new coastline and recovers the old coastline.

In our application, when the user clicks with the right button either inside or outside the coastline, the application understands that the user is defining the central point of a hill. Then a colored circular line will appear on the terrain surface surrounding the central point that has just been created. This line represents the area influenced by that particular hill. The user may add as many hills as desired and each one will be depicted in a different color. It is important to note that the hills which are located outside the coastline affect the terrain in those areas where the hills overlap the already existing coastline. After defining the hills, the user will be able to modify the radii and the location of the hills inside the island. Right-clicking on the center point of the hill and dragging, allows the user to change the position of the hill. The user can eliminate a hill by dragging it outside the island until its radius is completely outside the coastline. Alternatively, right-clicking and dragging on the circular line allows the user to modify the hill radius.

The application includes the possibility of *zooming* in on the sketched island by clicking the right button of the mouse. Using the zoom can help the user to get a better overview of the terrain. We have to click outside the island in order to zoom, but always away from the coastline. This is because if the user clicks too close to the coastline, the application will interpret that the user wants to create or modify a hill.

## 4.2 3D Modeling Operations

In the initial version of the application, the height of the island was defined by the *cross section* [22] selected in the Plan Window and the input given in the Elevation Window. This interface was complicated to use and it did not give the user full control.

In our application, the 3D window shows a volumetric view of the terrain. The user will be able to click with the right button on any of the previously defined hills and can decide on the height of each hill by dragging the mouse up and down. If we drag upwards, then the height will be positive and we will create a hill, and if we drag downwards the terrain will be a valley. Furthermore, by dragging the mouse left and right, the user will be able to decrease and increase the size of the radius of the selected mountain.

The 3D window also allows for the use of filters, which the user can decide to apply to the whole terrain in order to introduce some fuzzy bumps. Clicking with the left mouse button on any point on the island and dragging upwards will add filters to the terrain. The more we drag upwards, the more bumps are created. On the contrary, if we drag downwards then the application will understand that we want to decrease the number of perturbations.

In addition, the 3D window offers two more functions. Clicking with the left button away from any hill and dragging, acts as a rotating function. Also, clicking with the right button away from the terrain and dragging, acts as a zoom function.

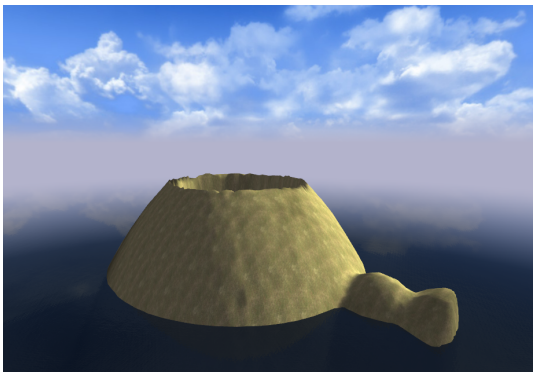


Figure 9: Examples of volcanic island in TGEA.

## 5 RESULTS

This section presents our results using a Pentium D 2.8 GHz processor with 2 GB. RAM and an nVidia GeForce 8800 GT graphics card. The framework was implemented in C++ with OpenGL.

In order to show the possibilities of our framework, we exported different heightmaps obtained from our terrain generation algorithm. Then we introduced those heightmaps as input to a game engine and we obtained several examples of islands. The game engine that we selected in our tests is the Torque Game Engine Advanced (TGEA) [4].

Figure 1 depicts an elongated island. The island is quite abrupt after having applied some filters in order to give the final terrain a more realistic appearance. In Figure 9 we have rendered a volcano with two tiny hills on one side. The volcano consists of a big hill but with a hole in the middle. This hole is performed by applying a slightly smaller negative-height hill, located in the middle of the first hill we created.

### 5.1 User Study

We consider important to perform a user study in order to evaluate the quality of our sketching application. We gave first-time users some basic indications in order to make them know how our application works. After that, we asked them to try to draw an island silhouette with the desired hills. They played with different coastline, hills and filters until they outlined the desired appearance of island and terrain.

Our informal user study considered the application from different perspectives. First, we ask our volunteers to grade from 1 to 10 the overall quality perception of the application, being 1 the worst and 10 the best. Secondly, we monitored their activity to record the time passed until they obtained a satisfactory terrain. Finally, we asked them to report any difficulty or mistake they could find in using our application.

Different studies have proven that 5 users are enough to assess the quality of software applications [14], although evaluating visualization results need a different amount of volunteers in order to obtain valuable results [9]. In our case we have conducted the test with 30 people. We have also grouped them in three groups depending on their expertise both in computer use in general and in computer design in particular.

In Table 1 we present the results obtained with university volunteers. Most of them were satisfied with their results after less than five minutes. It is important to mention that both computer scientist and designers found several problems that helped us to improve the application. Most of them found the application difficult to use at the beginning, although after some practice they started modeling their terrains. These initial usability problems helped us to modify some aspects of the application and also encouraged us to create a brief guide to explain how the application works. Moreover, most users found the application *funny* and played with it after acquiring a little expertise. Finally, we also encouraged some of them to include the terrain inside the Torque Game Engine [4], in order to give them the possibility to experience gaming over their modeled island and terrain.

## 6 CONCLUSIONS

This paper presents a method for terrain generation which is suitable for users who wish to have full control over the whole creation process. We have also presented a simple tool for creating solid models of imaginary islands. The tool is easy to use and requires only a minimal user interface, with all of its major operations being controlled by a two-button mouse. From this application, the user can add, remove and reshape existing hills interactively and the terrain will be updated accordingly. Moreover, the user is able to modify the silhouette of the island and add fuzzy bumps as desired.

With the images of islands that we have shown in the previous section, it can be seen how our approach is capable of offering very realistic terrains. The user can decide on the final appearance of the island, as it is possible to apply as any number of filters. Nevertheless, the user could choose not to apply filters in order to obtain a fairly rounded terrain which could be useful for a cartoon-like environment. The usability study, which was performed among persons with different computer skills, showed that the user interface we finally selected is comfortable and adequate in most cases.

Future lines of work on this application include the possibility of adding more tools at the expense of losing simplicity. In this sense, the authors are interested in allowing the user to include weather phenomena or vegetation and other decorative elements on their island.

## ACKNOWLEDGEMENTS

This work was supported by the Spanish Ministry of Science and Technology with grant TSI-2004-02940,

| Study Group | Number of volunteers | Overall Satisfaction | Avg. Required Time (min.) | Observed Usability Problems |
|---|---|---|---|---|
| Computer Scientists | 12 | 8 | 3 | 12 |
| Designers | 7 | 6 | 5 | 7 |
| Other Disciplines | 11 | 9 | 4 | 2 |

Table 1: Results obtained with 30 university volunteers.

project TIN2007-68066-C04-02 and through the Ramon y Cajal programme. Also by Bancaja with project P1 1B2007-56.

## REFERENCES

[1] M. Wacker B. Neidhold and O. Deussen. Interactive physically based fluid and erosion simulation. In *Eurographics Workshop on Natural Phenomena*, 2006.

[2] C. Dachsbacher. Interactive terrain rendering: Towards realism with procedural models and graphics hardware. Technical report, Fiedrich Alexander Universität, Germany. Thesis, 2006.

[3] S. Fefilatyev, V. Smarodzinava, L. O. Hall, and D. B. Goldgof. Horizon detection using machine learning techniques. *ICMLA*, 0:17–21, 2006.

[4] Garage Games. Torque game engine advanced. http://www.garagegames.com/, 2008.

[5] R. Gregg. Nem's 3d mega terrain generator. http://nemesis.thewavelength.net/, 2005.

[6] M. Kaplan and E. Cohen. Producing models from line drawings of curved surfaces. In *Eurographics Workshop on Sketch Based Interfaces and Modeling*, 2006.

[7] L.B. Kara and K. Shimada. Sketch-based design of 3d geometry. In *Eurographics Workshop on Sketch-Based Modelling*, pages 59–66, 2006.

[8] A. D. Kelley, M. C. Malin, and G. M. Nielson. Terrain simulation using a model of stream erosion. In *SIGGRAPH '88*, pages 263–268, 1988.

[9] R. Kosara, C. Healey, V. Interrante, D. Laidlaw, and C. Ware. User studies: Why, how, and when? *IEEE Comput. Graph. Appl.*, 23(4):20–25, 2003.

[10] Q. Li, G. Wang, F. Zhou, X. Tang, and K. Yang. Example-based realistic terrain generation. In *ICAT*, pages 811–818, 2006.

[11] M. Masry and H. Lipson. A sketch-based interface for iterative design and analysis of 3d objects. In *ACM SIGGRAPH 2007 courses*, page 31, 2007.

[12] Y. Mori and T. Igarashi. Plushie: an interactive design system for plush toys. In *SIGGRAPH 2007*, page 45, 2007.

[13] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3), 2007.

[14] J. Nielsen and T. K. Landauer. A mathematical model of the finding of usability problems. In *CHI '93: Proceedings of the INTERACT '93*, pages 206–213, 1993.

[15] T. J. Ong, R. Saunders, J. Keyser, and J. J. Leggett. Terrain generation using genetic algorithms. In *GECCO '05*, pages 1463–1470. ACM, 2005.

[16] S. Owada, F. Nielsen, K. Nakazawa, and T. Igarashi. A sketching interface for modeling the internal structures of 3d shapes. In *SIGGRAPH 2007 courses*, page 38. ACM, 2007.

[17] S. Schmitt. World machine. http://www.world-machine.com/about.html, 2006.

[18] D.K. Wright S. Lim U. Khan S.F. Qin, G. Sun and C. Mao. 2d sketch based recognition of 3d freeform shape by using the rbf neural network. In *Eurographics Workshop on Sketch Based Interfaces and Modeling*, 2005.

[19] Planetside Software. Terragen. http://www.planetside.co.uk/terragen/.

[20] K. Stachowski. Terraineer. http://terraineer.sourceforge.net/, 2006.

[21] A. Torpy. L3dt. http://www.bundysoft.com/L3DT/, 2008.

[22] P. A. C. Varley, M. Chover, and A. Puig-Centelles. Sketching islands for a game environment. In *5th Europ. Conf. on Visual Media Production*, 2008.

[23] P.A.C. Varley. Automatic creation of boundary-representation models from single line drawings. PhD Thesis, University of Wales, 2003.

[24] Kansas State University Wind Erosion Research Unit. Weru. wind erosion simulation models. http://www.weru.ksu.edu/weps.html, 2003.

[25] M. F. Worboys. *GIS: A Computing Perspective*. Taylor and Francis, 1995.

[26] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, 2007.

[27] J. Zimmermann, A. Nealen, and M. Alexa. Sketching contours. *Computers & Graphics*, In Press, Accepted Manuscript, 2008.