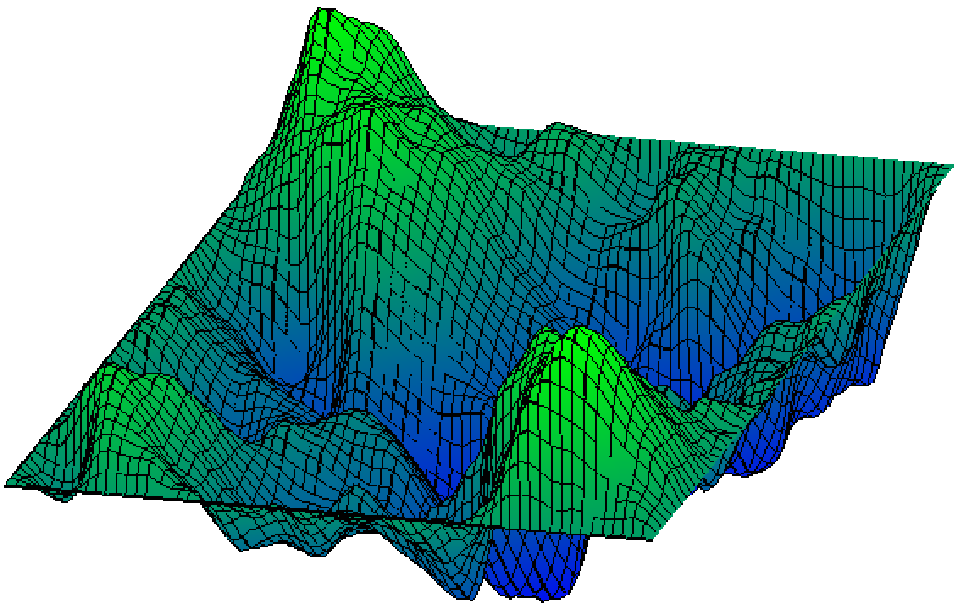
Computer Landscape Generation and Smoothing

Masters Comprehensive Exam

Grant Macklem

Spring 2003



Abstract

Due to the prevalence of 3D-accelerated graphics cards in modern computers, the ability to create heightened realism in computer graphics has become more important. Computer landscapes are important in many different fields of computer graphics such as gaming and simulation. Height maps are mentioned as a method of using real world data to construct a landscape. Various fractal methods of creating landscapes are also investigated including fractional Brownian motion and recursive subdivision methods such as triangular-edge, diamond-square, and square-square subdivision. Often the generated landscape is too jagged for a particular application, so various smoothing methods are investigated. The landscape height matrix can also be viewed as a set of pixel intensities for an image. As such, image filtering techniques such as Fourier and wavelet analysis can be used to process the image. Other methods include erosion simulation, polynomial interpolation, Bézier curves, B-splines, and Bézier patches. Results of four of these methods are analyzed for their effective smoothness.

1 Introduction

Computer landscapes are important in many fields including flight simulation, animation, and especially computer gaming. Sometimes a landscape is needed for robotic simulation of a foreign terrain where only a small amount of data is present [2, 27]. The techniques described in this paper (with the exception of using a height map) concentrate on modeling the macroscopic features of landscapes. That is, they attempt to create realistic-looking landscapes, but nothing which is intended to accurately represent any real-world landscapes.

Most landscapes are generated on a rectangular grid [8, 11, 19, 20, 27, 22] although they can be generated on any arbitrary lattice [5, 9, 10, 22]. For photorealistic landscapes, the grid can be made as small as one pixel. However, we note immediately that due to practical requirements of rendering the landscape, this level of detail would probably be used for a single computer-rendered photograph rather than an application which requires motion over the landscape.

Much of the underlying mathematics of fractals was regarded as a curiosity. However, with the advent of computer graphics, fractal techniques have been studied. The ability to generate computerized images using fractal geometry helped popularize this area of mathematics. Fractal geometry plays an increasing role in computer graphics to generate realistic scenes [22].

Fractal geometry, a branch of mathematics dealing with irregular shapes of the real world, has its origins around 1975. However, the concepts and shapes have been around much longer, especially in art. Tessellations became popular partly due to paintings by M. C. Escher. The essence of fractals involves their inherit self-similarity, a fact which is obvious in many of Escher's drawings. Take, for example, a coastline seen from space. One can examine a series of photographs of the same coastline taken at differing levels of magnification. The images are not identical, but it is not clear if they are simply a magnification of one area or if they came from an unrelated area of coastline. The main feature of fractals in nature is the property that objects differ in detail but still look similar [22]. Fractals are important in understanding many natural phenomena because, as Benoit Mandelbrot noted, "clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line" [18].

Clifford Pickover argues in "Generating Extraterrestrial Terrain" [23] that mountains are not statistically self similar, nor are ridges or craters. He proposes an alternate method of generation by altering the properties of the random number generation so they are non-periodic but similar to a Gaussian random number generator. Another method of interest to geomorphologists involves manipulating a fractal and an image representation of it to produce a landscape [24].

Some generated landscapes are often too jagged for a particular application. Many methods are considered in attempting to smooth the resulting landscape including polynomial interpolation, Bézier curves [17, 25, 26, 30], Bézier patches [12, 25, 26, 30], image processing techniques [29], erosion simulation [13, 14, 21], and B-splines [6, 17]. In this paper, the terms landscape and terrain will refer to the same object.

This paper is organized as follows. Section 2 provides a description of the various methods of landscape generation. Height maps and fractional Brownian motion are mentioned, but the main emphasis is on recursive subdivision methods. Section 3 describes a few smoothing techniques for generated landscapes. Only cubic order methods are considered for each technique. Section 4 provides results of a few smoothing techniques. Both the Fourier method and the Bézier patch work the best, although the Bézier patch is better overall since there are small oscillatory artifacts created with the Fourier method. Finally, Section 5 concludes.

2 Landscape Generation: Height Maps, Brownian Motion, and all the rest

The first problem encountered when dealing with computerized landscapes is generating a computer representation of a landscape. The simplest method is to read in a matrix of heights. This matrix is referred to as a height map and can represent real world data corresponding to a landscape or any set of data corresponding to measured heights. If a height map is not available for a particular application, the landscape can be generated. Since real world landscapes seem to have a sense of self similarity, it makes sense to consider fractal-generation methods as techniques for generating landscapes. Fractional Brownian motion uses a series of function evaluations to generate the landscape. Other methods use recursive subdivision to define a discrete grid and generate points. When finished, all methods have produced a grid in memory with corresponding height values.

2.1 Height Maps

The simplest method of generating a landscape is to use a predefined mapping of location to height. One source of such a map is a set of physical measurements from an actual geographic region. This map can be used to create authentic landscapes [27]. Such data is available from, for example, the U.S. Geological Survey for various parts of the country. However, a height map could be any arbitrary definition of heights. A grayscale image is another example of a height map.

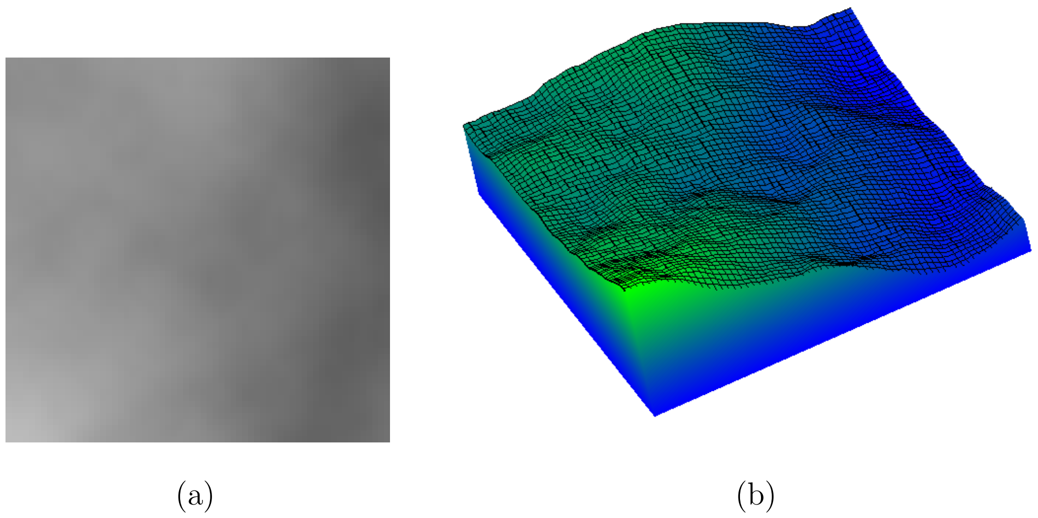


Figure 1: (a) A grayscale image height map. (b) The corresponding smooth landscape generated from the image height map.

When dealing with grayscale images, the intensity of each pixel value is taken to be the corresponding height value for each grid point. The mapping could be as fine as each pixel in the image representing one pixel of the generated landscape. Figure 1a illustrates an image of randomly generated clouds using the Adobe® Photoshop® Render Clouds function [1]. Rendering clouds can be accomplished using fractals as well [22]. Figure 1b shows the resulting landscape using the grayscale height map.

2.2 Fractional Brownian Motion

Often times it doesn't matter what the landscape looks like in a simulation as long as there is one. Other times many random landscapes are necessary but storing a large number of them causes other problems, e.g. storage, lack of variety, etc. It would be better to have a way to generate random landscapes when needed. Fractional Brownian motion (fBm) provides one way to generate a random landscape.

Brownian motion is also known as a random walk. Compared to white noise, however, Brownian motion is fairly correlated. That is, the next point of the function is not completely random, but rather related to previous points (see equation (1)).

Fractional Brownian motion [2, 3, 11, 22, 27] is a simple extension of Brownian motion. Fractional Brownian motion is described using a function of one variable, $V_H(t)$. For two-dimensional landscapes, $V_H(t)$ is replaced by $V_H(x, y)$ and Δt is replaced with a function of x and y. This function has the property that

$$\Delta V \propto \Delta t^H \tag{1}$$

The value H is a scaling parameter which characterizes the roughness of the function. Typical Brownian motion has $H = \frac{1}{2}$. Values closer to 0 result in rougher functions while values closer to 1 result in smoother functions. A value of $H \approx 0.8$ is typically used for landscape generation. Fractional Brownian motion is considered to be a fractal because the features of fBm repeat statistically under magnification. In general, a landscape generation algorithm will look most realistic if it closely approximates fBm.

Fractional Brownian motion can be approximated using a technique called midpoint displacement [11, 22]. A line is drawn and the midpoint is displaced a random amount proportional to the length of the line. This method can then continue over the newly created line segments.

Fourier transform filtering is another technique that can be used to simulate fBm [22]. A random function is created in the Fourier domain which has certain frequency properties similar to that of fBm (e.g. the spectral density). The inverse Fourier transform can be applied to this function which produces realistic landscapes. However, since the Fourier transform is not compactly supported, it is not possible to vary the "roughness" of the surface in different parts of the landscape. This would allow for smooth transitions and eliminate the need to smooth the landscape after generation.

Another approximation technique involves shear displacement or fault lines [14, 22]. This is the two-dimensional equivalent of the midpoint displacement technique. In this method, a line is chosen across the terrain grid. The current height is raised or lowered by different amounts on each side of the fault line. This process is repeated an arbitrary number of times (with a different fault line) until the resulting terrain is acceptable. Since the terrain on each side of the fault line is adjusted relative to its current value, this method takes into account the influence of neighboring point values to each new point value. The first iteration will usually have a large height adjustment and this amount will decrease with each iteration. This creates one prevailing fault line with additional fault lines being less influential.

Fractional Brownian motion (or the methods to approximate it) in general produces rough results. As such, the resulting landscape will need to be smoothed somewhat to be acceptable for an application.

2.3 Recursive Subdivision Methods

The methods of approximating fractional Brownian motion affect the entire landscape at once. Changing a value in the Fourier transform affects the whole landscape as well as adjusting the heights on each side of a fault line. Recursive subdivision methods, on the other hand, recursively define a landscape by subdividing it into smaller pieces. Once a grid point has been set with this method, it does not change.

Recursive subdivision methods are considered to be a fractal due to their self-similarity in construction. Each subdivision presents exactly the same problem as the larger terrain. This fact makes the use of recursion a natural choice. In domains where the terrain creation time is important, these techniques run in linear time rather than $O(N \log N)$ like the Fourier filtering method (see [19]). Each method begins with edge grid points defining a square. This region is subdivided into four new squares and new grid points are calculated until a suitable resolution is achieved.

In each method, a random offset is added to the new vertices created. This amount is proportional to the edge length of the current iteration. The randomness allows for greater

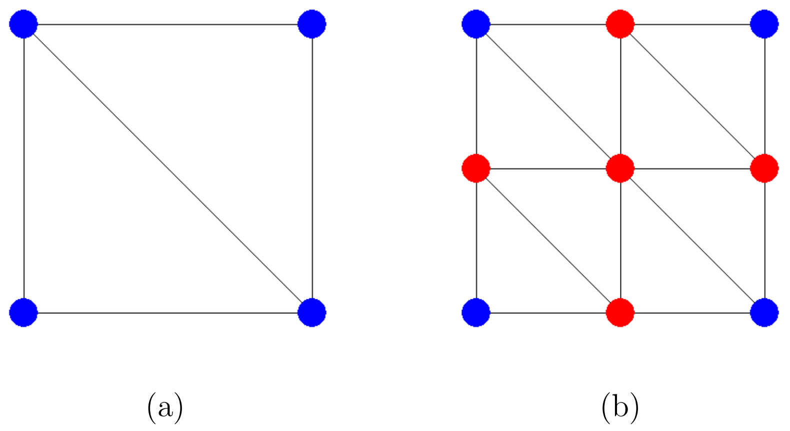


Figure 2: **Triangular-Edge Subdivision**. (a) The original data points for the grid are shown in blue. (b) A recursive subdivision generates new points (in red).

variation in the terrain. The proportionality prevents an unnaturally large spike in the terrain in a fine subdivision, which increases realism.

These methods all share a common disadvantage: their use of random numbers. Unless the use is controlled, zooming in on the landscape (and generating a finer grid) may produce a different landscape altogether. That is, since additional levels of recursion are involved, points that are now in the center of the smallest subdivision might have been previously generated somewhere else in the landscape.

To avoid artifacts resulting from creating the terrain on a rectangular grid, Mandelbrot discusses a method based on a hexagonal grid [10, 22]. This method is described to dramatically reduce creases (artifacts which make the underlying grid structure very apparent).

2.3.1 Triangular-Edge Subdivision

The first subdivision method is also the simplest. It is referred to as triangular-edge subdivision [8, 11, 19]. This method takes each grid square and logically divides it into two triangles. The midpoint of each triangle's edge is set to a random amount.

Figure 2a shows the four grid points required to begin this subdivision. Figure 2b shows the new midpoint values for each edge. Creating a grid with these new points creates four smaller squares which can be subdivided again.

However, visual artifacts are created with this method. Since the heights are added without relation to any of the surrounding points, creases appear when viewing the landscape from directly above and from other angles.

2.3.2 Diamond-Square Subdivision

A second method is the diamond-square subdivision [11, 19, 27]. Instead of generating the midpoint of each line, this method averages the four corners of the surrounding square (or diamond) to generate the midpoint. A random offset is then added to this value. Figure 3 illustrates this method.

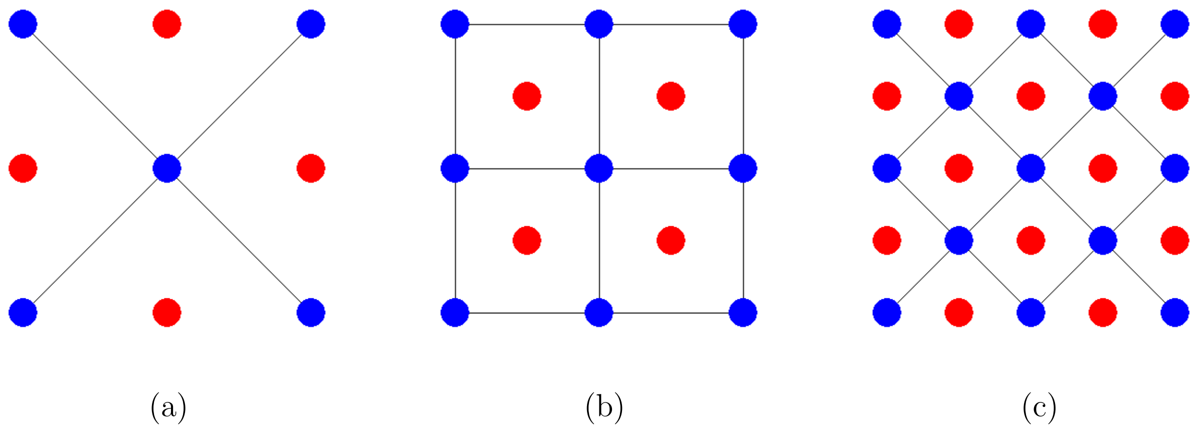


Figure 3: **Diamond-Square Subdivision**. Current data points at each step are shown in blue while newly generated data points are shown in red. (a) The first diamond iteration. Only half of each diamond is shown. (b) The square iteration. (c) The next diamond iteration.

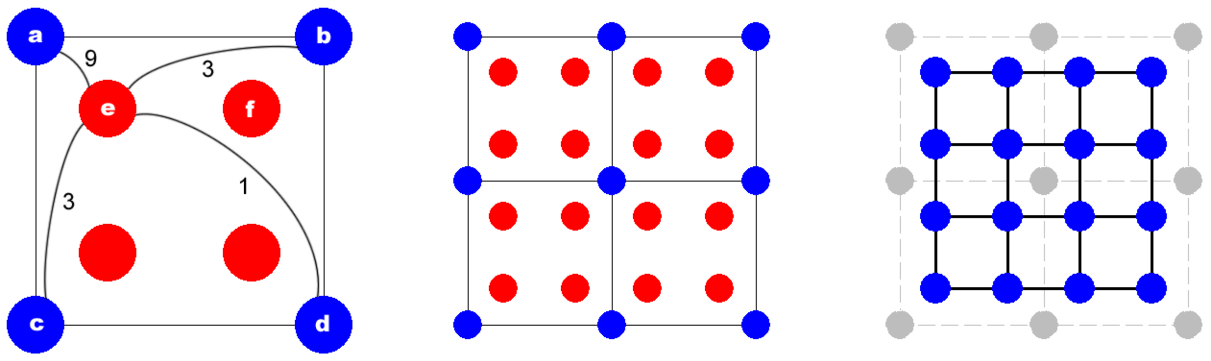
Figure 3a shows the five grid points required to begin this subdivision. Alternatively, the initial center point can be generated in a previous iteration. In each diamond iteration, there is a problem with generating points on the edge of the grid. These points can be linearly interpolated from the two neighboring points. Creating a lattice from these new points produces the rectangular grid shown in Figure 3b. The centers of the squares are generated, which produces a new lattice (See Figure 3c). This lattice is slightly offset from the original grid. However, rotating this lattice by 45° reveals the rectangular structure. Thus, squares in this rotated lattice produce diamonds on the original grid.

The midpoints of these diamonds are then computed which produces a rectangular grid again. Because of the alternation between diamonds and squares, this method is a two-step process. This process of diamond-square iteration is repeated until a suitable landscape is achieved. Since the surrounding points are factored into the computation of the midpoint, artifacts such as creases do not appear. However, Miller notes that other artifacts still appear due to the two-step iteration [19].

2.3.3 Square-Square Subdivision

In order to eliminate the visual artifacts produced by the previous two methods, Miller proposed a third method called square-square subdivision [19, 20]. It is based on work by Catmull and Clark [5] as well as Doo and Sabin [9, 28]. This method incorporates an approximation method during generation. Taken in the limit that points get arbitrarily close together, this method will produce a biquadratic surface. In order to eliminate the creases and other artifacts, Miller relaxes the condition that the terrain actually interpolate the original grid points. This produces a smoother curve since it ensures continuity of the derivative of the interpolating curve.

Figure 4b shows the nine grid points required to begin this subdivision. Each iteration subdivides the current square into a smaller square. This smaller square is half the scale



(a) (b) (c)

Figure 4: **Square-Square Subdivision**. Current data points at each step are shown in blue while newly generated data points are shown in red. (a) Generating new square vertices as a weighted average of the surrounding vertices. (b) The first square iteration. (c) The second square iteration. The newly generated vertices become the new grid. The old grid is simply discarded.

of the original square and is centered inside the original square. Equivalently, this places each new vertex exactly halfway between the corresponding old vertex and the center of the square. The vertices of the new square are a weighted average of the vertices of the original square (see Figure 4a). A random offset is also added to this amount. For example, the point e is calculated to be

$$e = \frac{9a + 3b + 3c + 1d}{16} + r \tag{2}$$

where r is a random amount. Equivalently, f is calculated to be

$$f = \frac{3a + 9b + 1c + 3d}{16} + r \tag{3}$$

At this point, the original grid is replaced by the newly generated points (see Figure 4c). Since at each iteration the existing points are only used to influence the points in the next iteration, the terrain only really interpolates the grid points at the last iteration.

Conceptually, the grid shrinks with each iteration. In the limit, the area of the subdivided grid will be one quarter that of the original grid. However, one can simply rescale the edge lengths to be whatever size is required.

This technique (by construction) produces landscapes which are smoother than the previously discussed techniques. However, if the resulting landscape is too smooth, the amount of random fluctuation can be adjusted (see [19]).

2.4 Modified Triangular-Edge and Diamond-Square Subdivision

With several generation methods presented, the next step is to reduce the roughness of the simulated landscape. However, first we will select a method to analyze in more detail. The

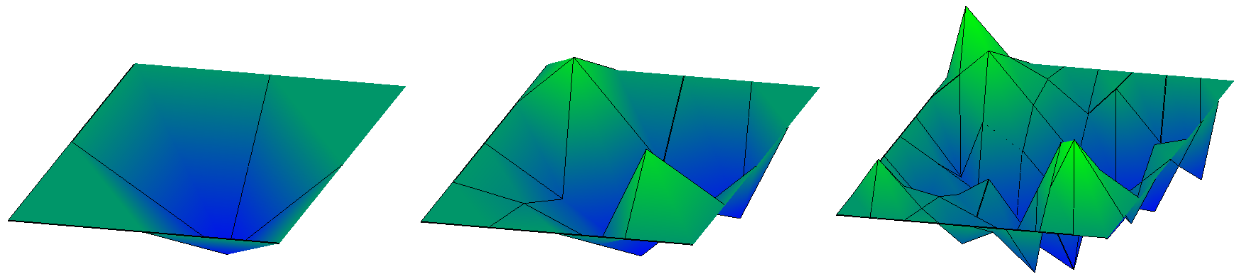


Figure 5: Example of recursive subdivision. This shows the first three iterations of the modified technique.

generation technique used in this paper is a mixture of the triangular-edge and diamond-square subdivision methods. The center of each square is calculated as the average of the surrounding points and a random offset is added. However, the edge midpoints are simply linearly interpolated along the sides. For more variation, a random offset could be added to these midpoints as well. This produces a single-step method.

This method of landscape generation is fairly basic, but it works well and runs in linear time with respect to the number of points. For example, Figure 5a shows a terrain with corner points equal to zero. The center point is shown as well as the interpolated edge values. These edge values could be generated by some other method as well.

This method continues recursively (see Figures 5b and 5c) until the edge length is less than some threshold. This threshold is related to the number of polygons needed to render the landscape. (For the final generated landscape, see Figure 15.)

3 Smoothing Techniques

The previous section describes various methods of generating a computer landscape. However, some methods create smooth landscapes and others do not. Depending on the intended use of the landscape, this may create a problem. For example, if one desires a very "rocky mountain," then the techniques might work perfectly. If, instead, one desires rolling hills that simulate real terrain, these methods don't work very well. Several solutions present themselves. One is to use a different method which produces smoother terrain. The second is to alter the random offset added to each new point to reduce the sharp edges. A more context independent technique is to smooth the resulting terrain.

This section investigates various smoothing techniques to reduce the sharp peaks and valleys that are a consequence of generating landscapes with the technique of Section 2.4. Within each smoothing technique, only cubic methods are considered as most of the literature concentrates on these. Higher order methods might produce a better result since there are more degrees of freedom to enforce constraints between the curves. However, a higher order method means more computational work which is not desirable in computer graphics.

In order to visualize and analyze the results of smoothing, the image must first be rendered. Throughout this paper we use OpenGL to provide the 3-D rendering capabilities [30]. Another technique would be to use voxel graphics [15, 16]. A voxel is like a pixel, except it is a volume element instead of a pixel element. Thus, it includes some information about its distance from the viewer. Voxel graphics are based on ray casting and were popularized as a terrain rendering technique in NovaLogic's *Comanche: Maximum Overkill* in the early 1990's. OpenGL is more standard than other commercial products.

The first section describes using standard image processing techniques to filter the landscape, followed by a brief overview of erosion models which attempt to mimic natural phenomena. Next come more rudimentary techniques such as polynomial interpolation, Bézier curves, and B-Splines. Finally, the Bézier curves are generalized to two-dimensional Bézier patches.

3.1 Image Filtering

As shown in section 2.1, landscapes can be generated from images. Equivalently, an image can be created from a landscape height matrix. After such a transformation, image filtering techniques such as Fourier and wavelet analysis can be used [29]. Various de-noising or blurring techniques can be applied to the image and the result again viewed as a landscape.

Sharp edges are represented with high frequencies in the Fourier domain or with large difference coefficients in the wavelet domain. Such high frequencies can be eliminated with the use of a low-pass filter. This is straightforward to implement in the Fourier domain. The Fast Fourier Transform [7] allows $O(n \log_2 n)$ transformations between the space and frequency domains and back again.

A formal introduction to Fourier and wavelet analysis is beyond the scope of this paper (see [29] for more details). Only a brief outline of the methods is provided here. One method of low-pass filtering is to use well-known filters for convolution with the image. The process is to first convolve each row of the image with the filter, and then convolve the columns of the result with the filter. This process will reduce the influence of the high frequencies. Given a filter h, this method computes for each row or column vector v,

$$v = v * h \tag{4}$$

since each row or column is replaced by its filtered value.

Equivalently, a filter can be constructed in the frequency domain which will reduce or eliminate the significance of high frequencies. One such filter is the Gaussian,

$$h(x) = e^{-\lambda x^2}, \quad x \in [-\omega_0, \omega_0] \tag{5}$$

where λ is a scaling parameter that controls the decay of the Gaussian and $[-\omega_0, \omega_0]$ is the region of interest. This filter is simply multiplied by the Fourier coefficients in the frequency domain. Note that even though this filter is defined in the spatial domain, it has the structure of a desired low-pass filter which justifies the multiplication in the frequency domain. For two-dimensional images, this filter generalizes to

$$h(x, y) = e^{-\lambda(x^2+y^2)}, \quad x, y \in [-\omega_0, \omega_0]. \tag{6}$$

When smoothing with wavelets, one technique is to compute the wavelet transform and then reconstruct the original image using a percentage of the transformed coefficients. Ideally, this would involve keeping the majority of the averaging coefficients and eliminating some of the difference coefficients. This is usually the case when keeping the largest magnitude coefficients.

Another approach involves selectively zeroing out portions of the wavelet transform which correspond to undesirable features. Finally, just as there were many choices for low-pass filters, there are also many choices for the particular wavelet to use.

Clearly, viewing a landscape as an image allows one to smooth the image using image processing techniques. However, this level of analysis is not always necessary to represent the macroscopic features of a landscape.

3.2 Modeling Erosion

Instead of treating the landscape as an image, perhaps a better method is to use techniques employed by nature. Landscapes in the real world are affected by erosion which takes sediment from mountains and deposits it in the plains. "Erosion techniques" can be applied to the terrain as it's being generated or after it is finished.

Robert Krten [14] describes a method of erosion that is applied during fault-line landscape generation. A low-pass filter can be used to simulate erosion in this context. After each fault and height adjustment, this filter is applied over the terrain. Then a new fault is generated and the filter is applied again. This smoothes the new terrain as well as the already-smoothed terrain from the previous iteration. Krten claims this is equivalent to geological wind erosion. He also describes a filtering technique where the filter is applied forwards and then backwards over each row and column. This simulates rain erosion where particles move in all directions.

Kelley et al. [13] describe a method of erosion by modeling a stream drainage network. In this case, the landscape is generated separately from the drainage network. However, the landscape should be a rough idea of the general slope and location of the desired terrain. The drainage network is created on the landscape based on geological parameters about streams, junction angles, profiles, etc.

Likely the most general model is that of Musgrave et al. [21]. A hydraulic erosion model is described which models the physical effects of erosion using a drop of water. This drop is placed on vertices of the terrain and allowed to move to lower vertices. In this process, the drop picks up sediment and carries it along. Factors such as the size of the drop and amount of sediment already present affect the erosive properties of the drop. Sediment is also displaced along the way. A second model described is one of "thermal weathering." This model is designed to encompass all other erosion models (specifically those which cause material to break off and accumulate at the bottom of a slope). These methods have both produced very realistic results.

While these technique may produce realistic results, they have the disadvantage of requiring an additional model to be applied to the landscape. A simpler method might be to look at the points of the landscape and attempt to reduce their extreme values.

3.3 Polynomial Interpolation

In attempting to smooth a landscape, one approach is to smooth the sharp peaks by interpolating a curve and using additional surrounding points. Lagrange interpolating polynomials [4] are considered because they are simple to implement. Since this method of polynomial interpolation was not mentioned in the literature, its viability could not be ascertained.

The Lagrange polynomial basis functions of degree n are given by

$$L_{n,k}(x) = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)} \tag{7}$$

for $k = 0, 1, \ldots, n$. As noted in the introduction to Section 3, only cubic methods are of interest. Thus, only the case where $n = 3$ is considered.

Landscape generation produces a discrete function of two variables. As such, the corresponding x-axis and y-axis of the landscape can be assigned to either the horizontal or vertical direction. The grid points can be dispersed in a convenient manner. For this smoothing method, we'll define the grid points to lie at integer values of the function. Neighboring points will be separated by a distance of 2. This allows the interpolated values to lie on integral coordinates as well. Now consider the paired data $\{(0, f(x_0))$, $(2, f(x_1))$, $(4, f(x_2))$, and $(6, f(x_3))\}$. The four associated cubic Lagrange polynomial basis functions corresponding to these points are graphed in Figure 6.

These basis functions describe how much emphasis each original control point exerts towards the final function. It is obvious that each function places the greatest emphasis on the point from which it is created. Each function is exactly 1 at this point and the other functions are 0. The Lagrange interpolating polynomial can then be constructed by multiplying the appropriate basis function by the corresponding function values. This polynomial is given by

$$P(x) = \sum_{k=0}^{3} f(x_k) L_{3,k}(x). \tag{8}$$

As an illustrative example of this method, consider the paired data $\{(0, 0)$, $(2, -3)$, $(4, 7)$, and $(6, -10)\}$. The Lagrange polynomial computed over these data points is

$$\begin{aligned} P(x) &= 0 - 3\frac{x(x-4)(x-6)}{16} + 7\frac{x(x-2)(x-6)}{-16} - 10\frac{x(x-2)(x-4)}{48} \\ &= -0.833x^3 + 6.625x^2 - 11.417x. \end{aligned} \tag{9}$$

Figure 7 shows a graph of the Lagrange interpolating polynomial (equation (9)) for this problem as well as the curve connecting the original data points (in black). The original points were at $x = 0, 2, 4, 6$. By evaluating the function at $x = 1, 3, 5$, a new curve is constructed that interpolates the original points (in blue). Using Lagrange interpolating polynomials reduces the severity of the extremes by adding additional points around them.

One concern with this method is that this construction of the Lagrange polynomial is only one dimensional and the landscape has two dimensions. One way to solve this is to first interpolate in one direction and then interpolate in the other direction (see Section 3.1 on convolution).

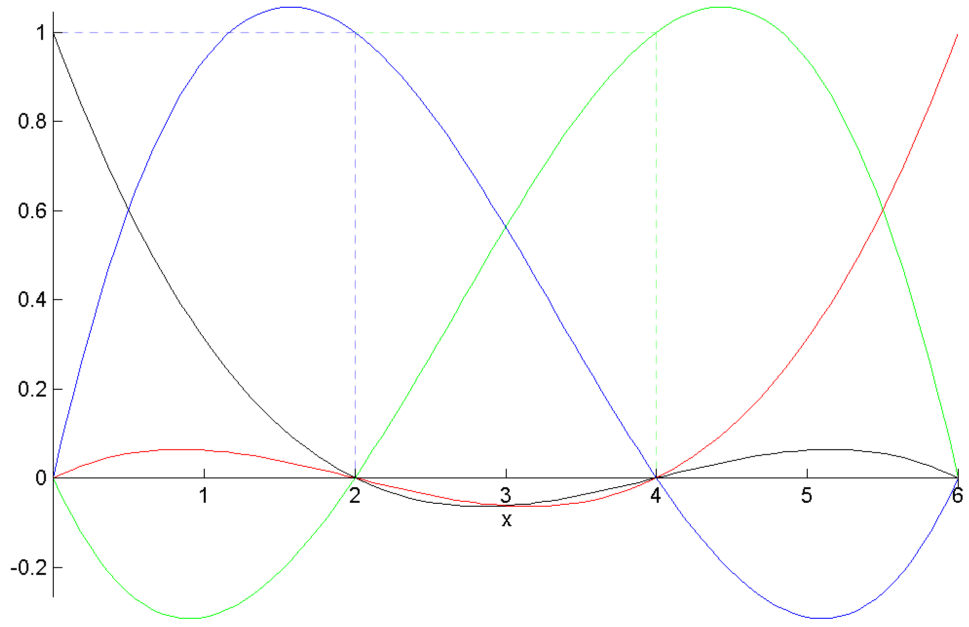


Figure 6: The Lagrange Polynomial Basis Functions. $L_{3,0}$ is shown in black, $L_{3,1}$ in blue, $L_{3,2}$ in green, and $L_{3,3}$ in red.

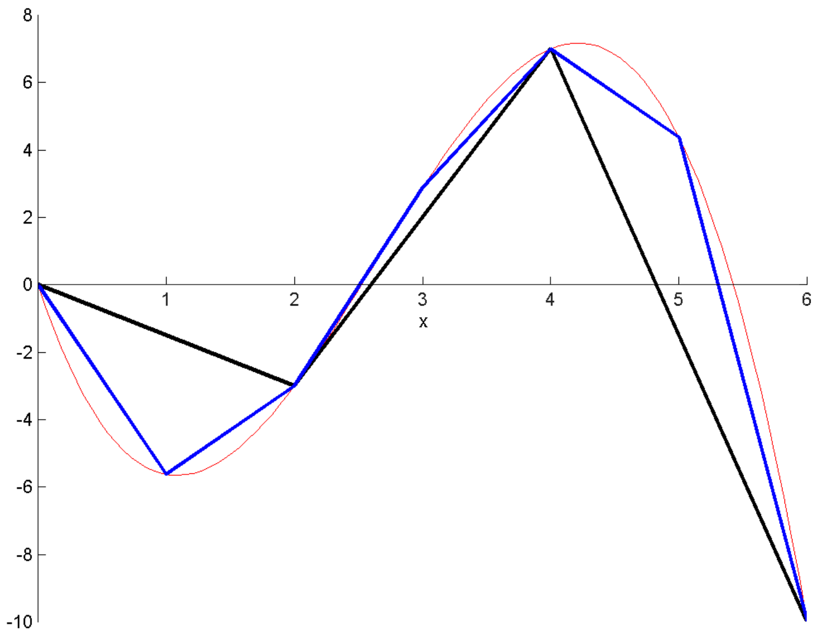


Figure 7: Smoothing with Lagrange Polynomials. The curve connecting the original data points is shown in black. The interpolating polynomial is shown in red. The curve connecting new data points is shown in blue.

Another problem is the need for essentially twice as many data points. Since interpolation, by definition, must pass through the original points, we need points in between every data point in order to avoid getting the same curve after smoothing. Often times these extra points are undesirable and other times they are infeasible since additional memory is required. In these cases, modifying the existing points will provide a better solution.

3.4 Bézier Curves

Since the interpolating polynomial necessarily passes through each original control point, a better method might be to consider approximating polynomials. One such polynomial is a Bézier curve [17, 25, 26, 30].

The Bézier curves are constructed so that the curve interpolates the endpoints. Also, each basis function of the curve should have control near the point where it's approximating. This is similar to the Lagrange basis functions. The basis functions of Bézier curves are the Bernstein basis functions and are defined by

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} \qquad 0 \le i \le n. \tag{10}$$

where n is the order of the polynomial and the binomial coefficient is given by

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}. \tag{11}$$

As with the Lagrange polynomials, only cubic polynomials are considered here. Evaluating equation (10) with $n = 3$ produces the four basis functions graphed in Figure 8. Note that these basis functions are defined over the interval $[0, 1]$ while the Lagrange basis functions were defined over the interval $[0, 6]$. Each basis function puts the most emphasis on the point from which it is created. However, the basis functions are not all 1 at their control point nor are the other basis functions 0. Thus, the resulting curve will not interpolate all the points. The other control points will have a sort of "pull" on the curve proportional to their distance from that point. The middle basis functions have the most pull to their control point about 1/3 and 2/3 through the curve, respectively. Two interesting notes about the Bernstein polynomials are that they are all nonnegative for u in the range $[0, 1]$ and the sum of the basis functions add to 1 for any value of u.

The cubic Bézier curve can be constructed by multiplying the appropriate basis function by the corresponding control point. The control points are simply the function values. The curve is defined by

$$B(x) = \sum_{i=0}^{3} p_i B_i^3(x_i) \tag{12}$$

where p_i is the value at the ith control point and $x_i = \frac{i}{3}$.

As an example of this method, we return to the sample problem described for Lagrange polynomials. Since the spacing between the points was arbitrary, it can be adapted to fit this problem. In general, the data will need to be adjusted to lie in the interval $[0, 1]$. The

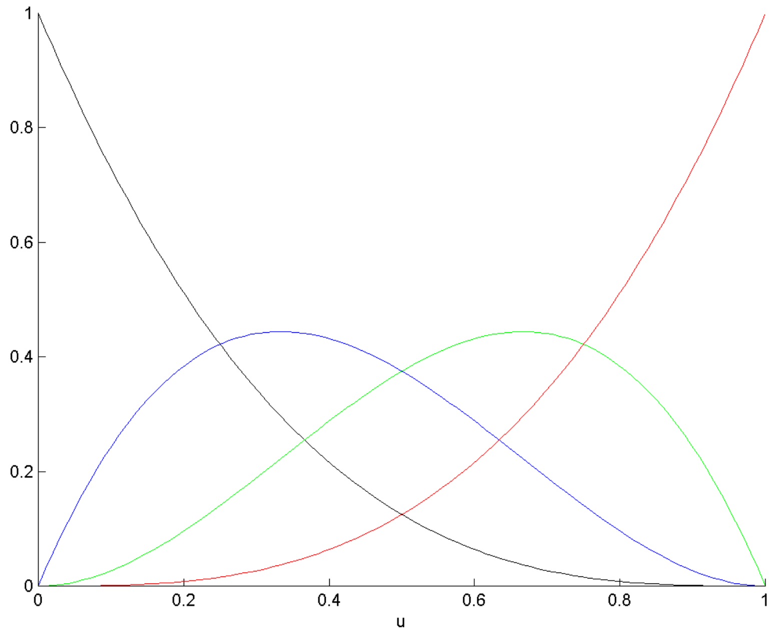


Figure 8: The Bernstein Polynomial Basis Functions. B_0^3 is shown in black, B_1^3 in blue, B_2^3 in green, and B_3^3 in red.

adjusted data are $\{(0,0),\ (1/3,-3),\ (2/3,7),\text{ and }(1,-10)\}$. The resulting Bézier curve is

$$\begin{aligned} B(x) &= 0 - 3 \cdot 3x(1-x)^2 + 7 \cdot 3x^2(1-x) - 10x^3 \\ &= -40x^3 + 39x^2 - 9x. \end{aligned} \tag{13}$$

Figure 9 shows the graph of the Bézier curve (equation (13)) for this problem as well as the curve corresponding to the original data points (in black). In order to use this curve as a smoothing technique, the Bézier curve is resampled at the intermediate data points, $x_1 = \frac{1}{3}$ and $x_2 = \frac{2}{3}$. The blue curve shows the resulting curve after resampling. We can see that the sharp peaks in the middle have been eliminated. Although this nearly flattens the terrain, in general this technique will not always produce such a bland result. The peak at the right endpoint is not affected, however. If this technique is repeated over sets of four points, every fourth point will remain unchanged.

There are two solutions to this problem. One method is to only shift by three points after each smoothing. In this way, the third point of the previous curve would be the first end point of the current curve. Since the curve interpolates the endpoints, this old control point will not be modified. Another method is to average the last two control points of a given curve and to use this point as the last point of the current curve and the first point of the next curve. This is the method implemented in this paper.

Figure 10 illustrates this method. For the given points P_0 through P_7, the two new averaged points are

$$Q_0 = \frac{P_2 + P_3}{2} \tag{14}$$

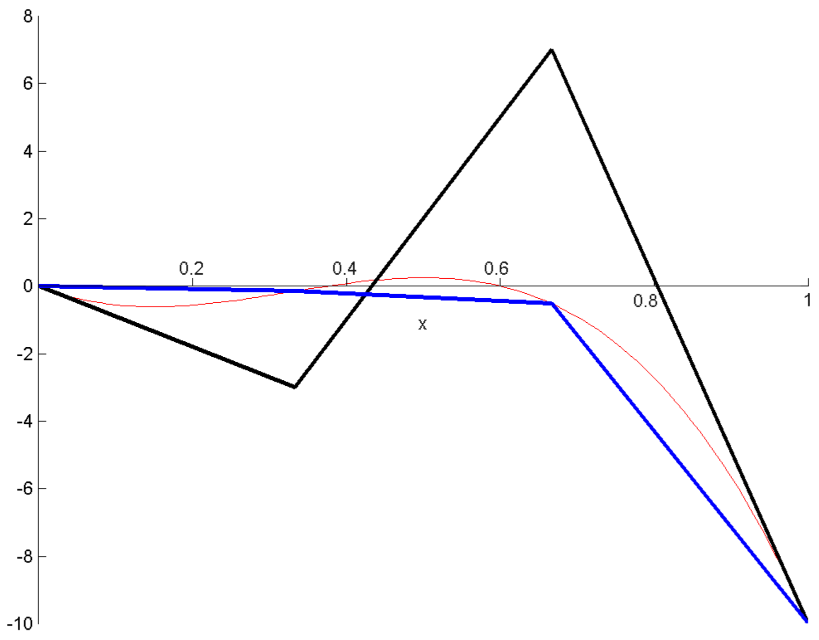


Figure 9: Smoothing with Bézier Curves. The curve connecting the original data points is shown in black. The approximating Bézier curve is shown in red. The curve connecting the new points is shown in blue.

$$Q_1 = \frac{P_4 + P_5}{2} \tag{15}$$

Thus, the control points for the curves are $\{P_0, P_1, P_2, Q_0\}$, $\{Q_0, P_3, P_4, Q_1\}$, and $\{Q_1, P_5, P_6, P_7\}$. These averaged points are not stored in the final result. They are purely for computation. Since these averaged points are in between previous control points, the sampling is slightly offset from before. However, the goal is not accurate representation of the terrain, but rather to smooth the terrain.

Since the Bézier curve is one-dimensional, like the Lagrange polynomial, the same technique of applying the polynomial first in one direction and then in the other is used. The implication of this technique is that the original data points are not used when smoothing in the second direction. This is not really a problem since the Bézier curve does not interpolate the points; the result is simply an approximation of an approximation. A benefit of using Bézier curves instead of the Lagrange polynomial is that there are as many data points in the end as in the beginning. This eliminates the need for extra storage space. However, this method introduced a new problem dealing with interpolation of the endpoints. The next method removes this restriction as well.

3.5 B-Splines

To reduce the effect of endpoint interpolation, a higher-order Bézier curve can be generated over more points. However, it becomes more and more difficult to ensure its smoothness. Cubic B-splines can be used to generate a curve over many control points like a higher-order

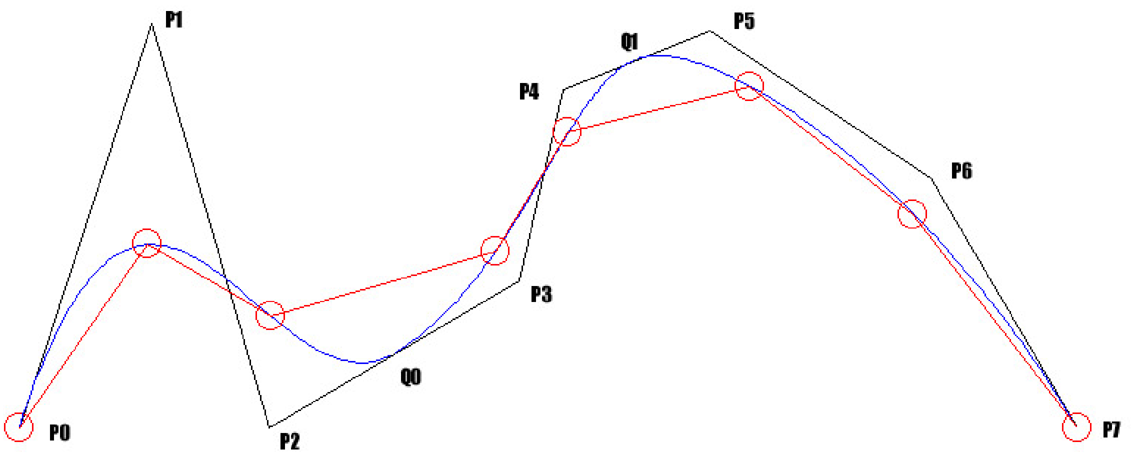


Figure 10: Extending Bézier curves to more than four points by averaging endpoints.

Bézier curve or polynomial. The B-spline [6, 17] has local control since it is influenced by only four control points and will not suffer from oscillations characteristic of higher-order polynomials [4]. Most importantly, unlike a natural cubic spline or other methods seen thus far, a B-spline does not interpolate its endpoints. Also, the curve and its first and second derivatives are all continuous.

B-splines are defined in terms of blending functions. These functions are defined recursively by the equations

$$B_{i,0}(x) = \begin{cases} 1, & x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

$$B_{i,p}(x) = \frac{x - x_i}{x_{i+p} - x_i} B_{i,p-1}(x) + \frac{x_{i+p+1} - x}{x_{i+p+1} - x_{i+1}} B_{i+1,p-1}(x) \tag{17}$$

Equivalently, blending functions of degree p can be obtained by computing the convolution of a box with itself p times. Consider the sequence $x_i = [0, 1, 2, 3, 4]$. The cubic blending function is defined by

$$B_{0,3}(x) = \begin{cases} \frac{1}{6}x^3, & 0 \le x < 1 \\ -\frac{1}{2}x^3 + 2x^2 - 2x + \frac{2}{3}, & 1 \le x < 2 \\ \frac{1}{2}x^3 - 4x^2 + 10x - \frac{22}{3}, & 2 \le x < 3 \\ \frac{1}{6}(4 - x)^3, & 3 \le x < 4 \end{cases} \tag{18}$$

Blending functions defined over a uniformly spaced grid such as x_i will produce a uniform B-spline. In this case, all the cubic blending functions can be obtained by translating equation (18). Thus, $B_{k,p}(x)$ can be obtained from $B_{0,p}(x)$ by

$$B_{k,p}(x) = B_{0,p}(x - k) \tag{19}$$

Figure 11 shows a graph of a few of these basis functions.

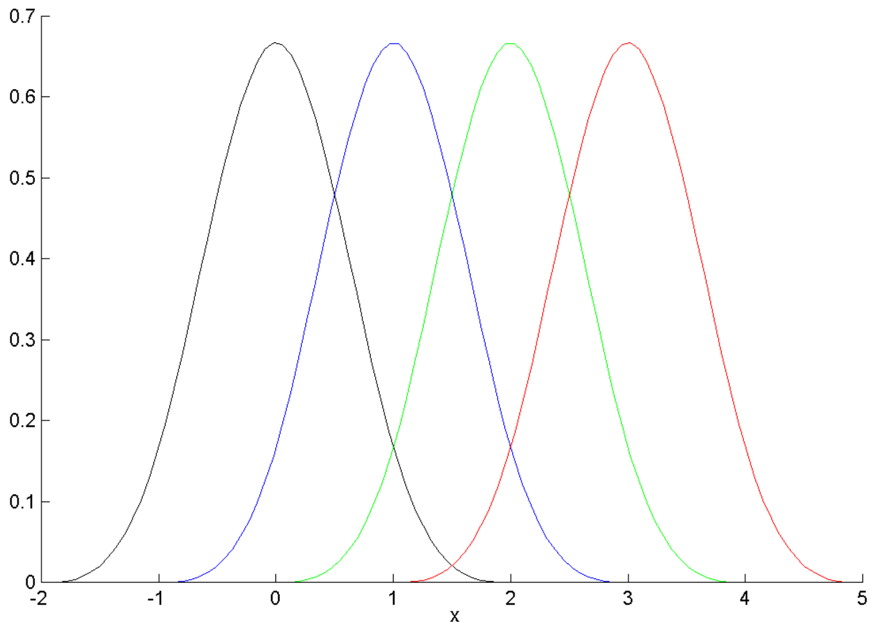


Figure 11: Some Cubic B-Spline Basis Functions. $B_{-2,3}$ is shown in black, $B_{-1,3}$ in blue, $B_{0,3}$ in green, and $B_{1,3}$ in red.

In general, given an array of control points $P_i,\ i = 1, \ldots, n$, the B-spline is defined by

$$f(x) = \sum_{k=0}^{n} P_k B_{k,3}(x) \tag{20}$$

However, since the basis function multiplied by each control point doesn't have its maximum value until two units later, this produces a B-spline which appears to be shifted to the right. In this paper, the basis function which peaks where the control point is located is used for that point. In effect, this shifts the curve to the left by two units. Mathematically, the B-spline is redefined to be

$$f(x) = \sum_{k=0}^{n} P_k B_{k-2,3}(x) \tag{21}$$

As an example, consider the problem described previously for Lagrange polynomials and Bézier curves. Equation (18) is defined over an interval with uniform spacing of 1. Thus, the sample data points are separated by a distance of 1. Consider the paired data $\{(0,0)$, $(1,-3)$, $(2,7)$, and $(3,-10)\}$. The B-spline computed over these data points is

$$\begin{aligned} f(x) &= \sum_{k=0}^{n} P_k B_{k-2,3}(x) \\ &= 0B_{-2,3}(x) - 3B_{-1,3}(x) + 7B_{0,3}(x) - 10B_{1,3}(x) \end{aligned} \tag{22}$$

Figure 12 shows a graph of the B-spline (equation (22) for this problem as well as the curve connecting the original data points (in black). Resampling the B-spline at the original data points produces new points. The resulting smoothed data points are shown in blue.

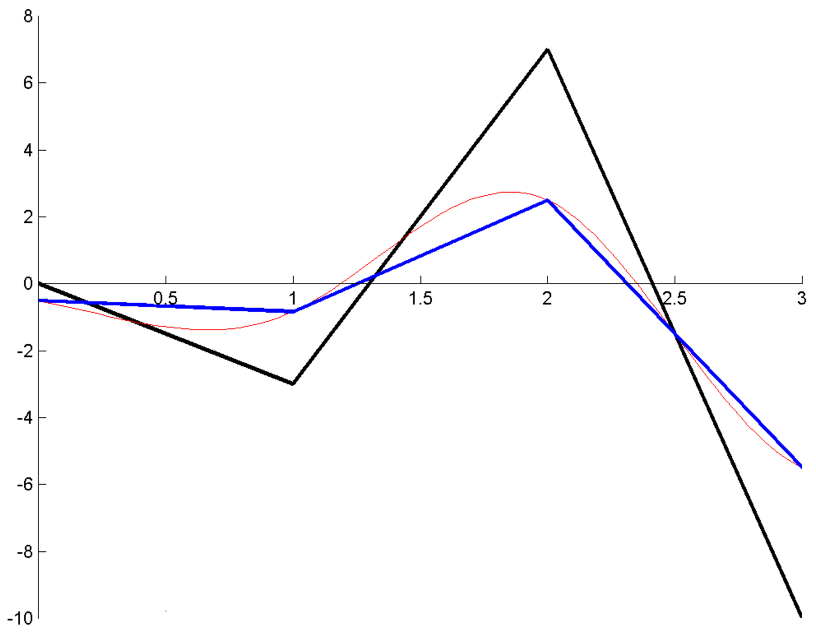


Figure 12: Smoothing with Cubic B-Splines. The curve connecting the original data points is shown in black. The B-spline is shown in red. The curve connecting new data points is shown in blue.

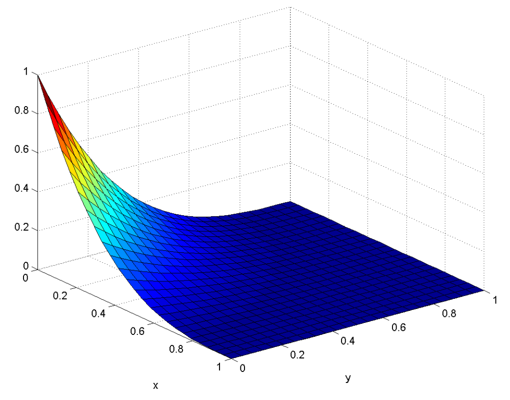
The peak and valley have been reduced in height and the endpoints are not interpolated. As a result, this process can be extended to an arbitrary length list of control points without any special consideration for the endpoints. Each new point can be calculated by evaluating the B-spline controlled by the surrounding four points.

The previous three methods have concentrated on using the data points of the landscape for smoothing in one dimension. Perhaps better results can be obtained by taking into account the influence of surrounding points in two dimensions.

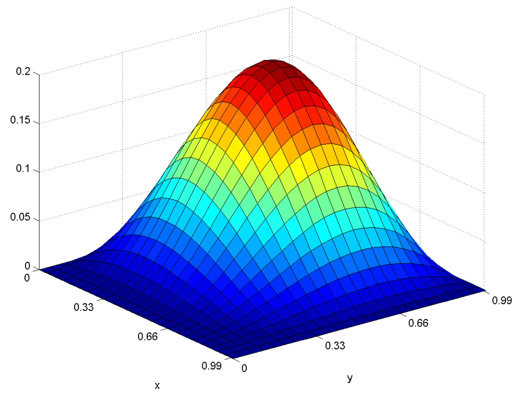
3.6 Bézier Patches

When working with computer graphics, Bézier patches are most commonly mentioned for creating surfaces [12, 25, 26, 30]. The Bézier patch is an extension of the one-dimensional Bézier curve. Since the Bézier curve was a function of one variable and had a one-dimensional array of control points, it's natural that the Bézier patch is a function of two variables and has a two-dimensional array of control points. Bézier patches can be made from arbitrary degree Bézier curves, but this section will only concentrate on "bicubic patches." Bicubic simply means that the function is cubic along each dimension.

The Bézier patch allows us to evaluate our surface at some point (x, y). The construction of the patch is to take into account the influence of a grid of 16 control points. Multiplying the basis functions from the one-dimensional case produces the following equation for the



(a)



(b)

Figure 13: Bézier patch basis functions. (a) The basis function corresponding to $i = 0,\ j = 0$. (b) The basis function corresponding to $i = 1,\ j = 2$.

patch

$$B(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} B_i^3(x) B_j^3(y). \tag{23}$$

The basis functions of the Bézier patch cannot all be plotted on a single graph since they are three-dimensional. However, Figure 13a shows a graph of the $i = 0,\ j = 0$ basis surface and Figure 13b shows the $i = 1,\ j = 2$ basis surface. The Bézier patch will interpolate the four corner points of its control matrix, but the remaining points will only be approximated. As we can see from Figure 13b, this surface has local control near its control point $(1/3, 2/3)$. In addition, the height has a maximum value around 0.2, compared to a value of 1 for the interpolating patch of Figure 13a.

Applying these patches to smoothing presents the same problem encountered with the Bézier curves. This patch can be naïvely applied over every set of 16 control points, which will flatten any extremities in the middle of the patch. However, this is still not perfect since the four corner control points are not changed. One solution is to average the end points with other surrounding points. This is more complicated than with Bézier curves since there are surrounding points in two dimensions. The method used in this paper averages the corner control point with the four neighboring grid points. If the patch is constructed over points along the edge or corner of the overall grid, some of these neighboring points don't exist. In this case, the corner control point is assigned a heavier weight in the average. As with the Bézier curves, smoothing with Bézier patches yields the same number of data points after smoothing as in the beginning.

As an example, consider the grid defined by the points

$$P = \begin{bmatrix} 0 & 5 & 6 & 0 \\ 4 & 2 & 7 & 4 \\ 6 & 7 & 5 & 2 \\ 0 & -2 & 4 & 0 \end{bmatrix} \tag{24}$$

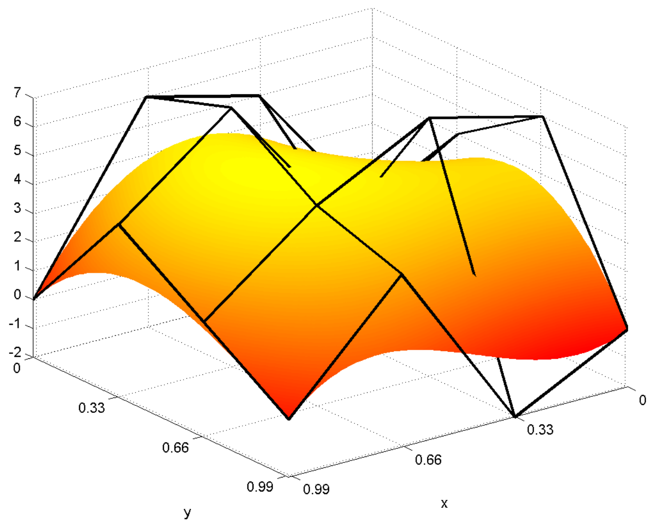


Figure 14: A Bézier patch computed from the rectangular grid of control points.

Figure 14 illustrates this grid as a wireframe mesh as well as the corresponding Bézier patch. It is obvious that resampling at the original grid points will produce a much smoother grid. Note that the last row of equation (24) is illustrated on the right-front side of the figure. The points may appear reversed, but this is due to the orientation of the x-axis.

4 Results

The previous section described six techniques for smoothing a simulated landscape. This section discusses the results of four of the techniques. The goal of this section was to find a method to smooth the image as much as possible. That is, methods which removed more peaks and valleys are noted to perform better. This is subjective to the eye of the beholder, but in all cases the resulting terrain is compared with the original terrain. Depending on the needs of a particular application, certain techniques may be preferred over others.

Image filtering techniques are considered first. Results obtained with Fourier analysis are better than those with wavelet analysis. Polynomial interpolation is considered next, but this did not produce worthwhile results. Next, Bézier curves are considered and they perform acceptably. The best results come from Bézier patches, considered last.

4.1 Image Filtering

To demonstrate image filtering techniques, an image was created from the landscape where pixel intensities were represented by vertex heights. The original landscape is shown in Figure 15. Figure 16a shows the results of convolving the image with a low-pass filter in the spatial domain. The filter used was defined by

$$H_0(z) = -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + \frac{3}{4} + \frac{1}{4}z - \frac{1}{8}z^2 \tag{25}$$

As Figure 16a shows, there is no visible difference between the filtered landscape and the original. The filtering process could be repeated several times, but a better solution would be to possibly use different or longer filters.

Low-pass filtering of the image can also be performed in the frequency domain, as illustrated in Figure 16b. This landscape was smoothed with the Gaussian function defined by

$$h(x, y) = e^{-\frac{1}{4}(x^2+y^2)} \tag{26}$$

over the interval $x, y \in [-4, 4]$. Equation (26) reduces the influence of the high frequencies much more than that of the previous filter, but a few oscillations are noticeable upon close examination of the edges. The resulting landscape is very smooth, but still preserves many of the original features of the terrain.

Moving to the wavelet domain, low-pass filtering is approximated by removing some of the difference coefficients. Figure 17a shows the result of smoothing with the Haar wavelet. A fifth-level wavelet transformation was computed. The largest 30% of the coefficients were kept since this produced a terrain that was noticeably different than the original. However, the resulting terrain looks unnatural in that it has a general blocky structure. Indeed, the peaks and valleys have been reduced, but the terrain is not smooth to the eye. It still has sharp elevation changes between neighboring grid points. This is caused by the poor performance of the Haar wavelet.

A better wavelet is the Daubechies D4 wavelet. Figure 17b shows the result of smoothing with this wavelet. Again, a fifth-level wavelet transformation was computed, but only the largest 10% of the coefficients were used for reconstruction. A reduction of this magnitude in the number of coefficients was necessary to produce a noticeably different terrain. This produces a much better result than with the Haar wavelet, although it is still not as smooth as other methods. Better results might be achieved by using a different level wavelet transform or different wavelets altogether.

Wavelets also may not be the method of choice for this domain since they are compactly supported. The desired smooth terrain has more of an undulating structure typical of waveforms. As such, Fourier methods with natural oscillatory properties may be better suited than wavelets.

4.2 Polynomial Interpolation

Rather than concentrating on frequencies of the image as a whole, polynomial interpolation attempts to reduce extreme values in more of a localized fashion. However, as Figure 18 shows, smoothing with only Lagrange polynomials is not the best way to reduce sharp peaks and valleys in a landscape. All of the peaks and valleys have been preserved. Generally, this is not what a landscape should look like. The peaks have been smoothed a bit. Further, they now have a little curvature in each direction due to sampling the interpolating polynomial in between the original points. However, this is far from the most desirable.

The landscape does look better in general due to the increased number of polygons approximating the surface. However, since only a cubic polynomial was used to interpolate the data, there is not much room for the polynomial to fluctuate in the span of four data points. Another method would be to consider higher order interpolating polynomials. Since they still must pass through every point, this method will probably not produce much better results. In addition, attempting to interpolate many points with a high degree polynomial tends to introduce a wildly oscillating polynomial [4]. Clearly, this is also not desired. We conclude that if it is important to use additional data points to provide a better approximation, another technique should be applied first to reduce the peaks and valleys and then apply the Lagrange polynomials.

4.3 Bézier Curves

The results when interpolating the grid points were not that great. However, the Bézier curve approximates the grid points. The resulting landscape is shown in Figure 19. This is a great improvement over the Lagrange polynomials since most of the sharp peaks have been eliminated.

The Bézier curve does interpolate the endpoints, so the control points at the ends of each curve were averaged so that every point was smoothed. As with the Lagrange polynomials, higher order Bernstein polynomials could be considered to construct other Bézier curves. This might improve the resulting landscape more than with Lagrange polynomials since more control points would influence each curve. However, this will also add to the mathematical complexity and run time for the code.

Other polynomials exist which do not interpolate the endpoints. Although not implemented, B-splines are one such example. These are expected to produce better results over intervals of arbitrary length.

4.4 Bézier Patches

Since the terrain is two-dimensional, better results might be obtained by taking into account surrounding points in both directions. Figure 20 shows the results of smoothing with Bézier patches. Comparing this with Figure 19, it is obvious that there is a lot of similarity. However, the patch has fewer sharp peaks. Overall, it seems to produce a better smoothing of the terrain than the one-dimensional Bézier curve. This is very likely due to the larger number of control points influencing each point. Also, all original points were used for each patch (with the exception being the averaged points used for blending patches together). With the one-dimensional cases, the curve had to be smoothed in one direction and then in the other. Intuitively, the Bézier patch seems to have a better preservation of the original data.

As mentioned in the comments for the Lagrange polynomial interpolation, the resulting landscape did look better due to a larger number of points representing it. As such, a very nice landscape should be produced by first smoothing with a Bézier patch and then using an interpolating polynomial to refine the remaining rough edges.

Figure 21 shows the result after applying the Lagrange interpolating polynomial to Figure 20. This landscape looks very smooth. However, creating both the Bézier patch and the

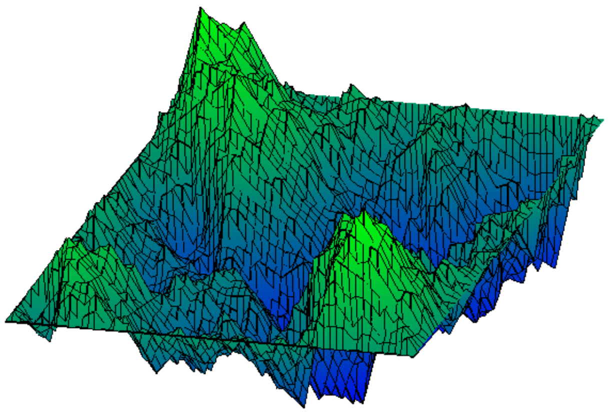


Figure 15: The original landscape

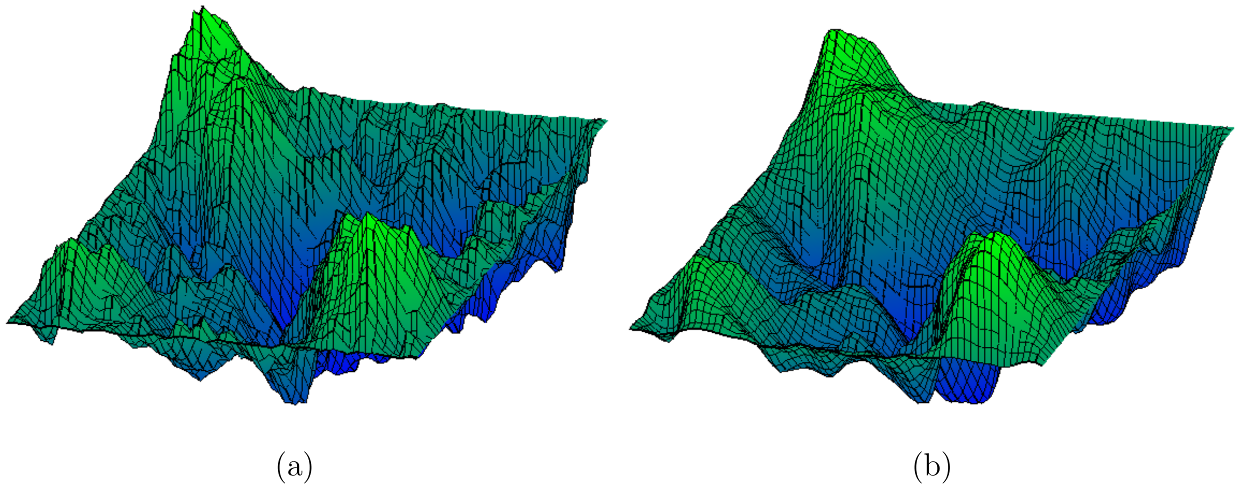


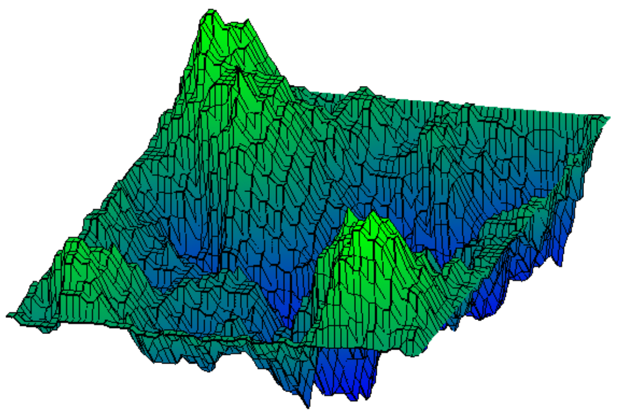
Figure 16: Smoothing with Fourier analysis. (a) The resulting landscape after convolving a low-pass filter with the image in the space domain. (b) The resulting landscape after filtering with a Gaussian low-pass filter in the frequency domain.

Lagrange polynomial is computationally very expensive. Perhaps the best way to accomplish this result would be to only compute the Bézier patch and to sample it more often.

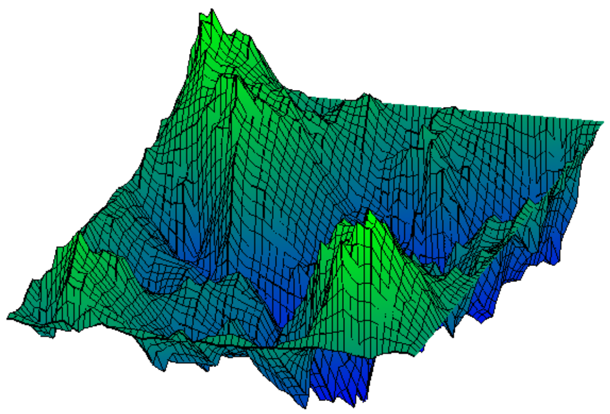
5 Conclusion and Future Work

This project examined the generation of random landscapes. The first generation method considered was the use of height maps. This method can be used to construct landscapes from real world data or images. Fractal generation methods were then considered including fractional Brownian motion and recursive subdivision techniques. These methods provide a rather simple method for generating landscapes, but they are often not representative of realistic landscapes.

In order to achieve a more reasonable appearing landscape, various smoothing techniques were considered. These included image processing techniques, modeling natural erosion,



(a)



(b)

Figure 17: Smoothing with wavelet analysis. (a) The resulting landscape after reconstructing from the largest 30% of the Haar wavelet coefficients. (b) The resulting landscape after reconstructing from the largest 10% of the Daubechies D4 wavelet coefficients.

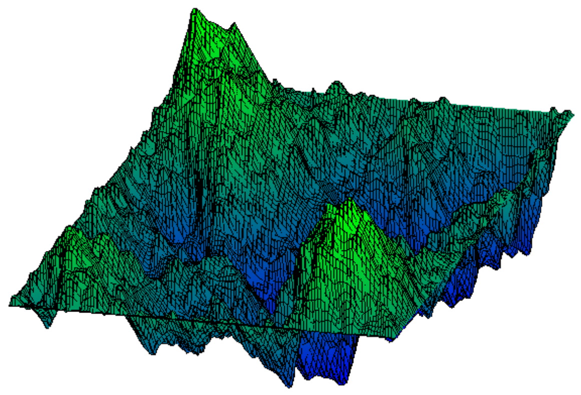


Figure 18: The resulting landscape after smoothing with Lagrange polynomials.

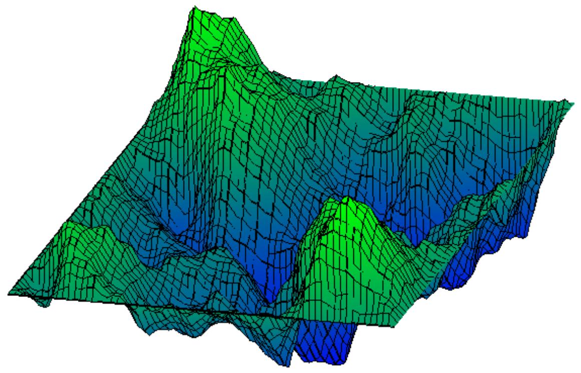


Figure 19: The resulting landscape after smoothing with Bézier curves.

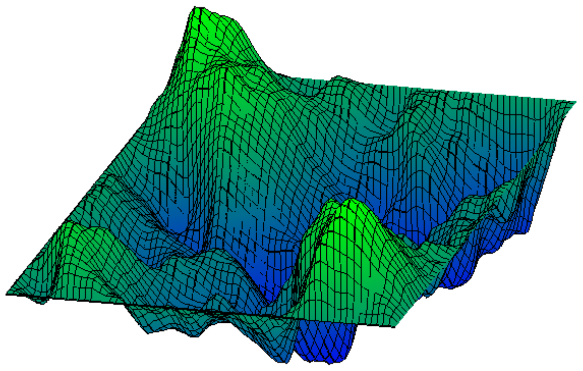


Figure 20: The resulting landscape after smoothing with Bézier patches.

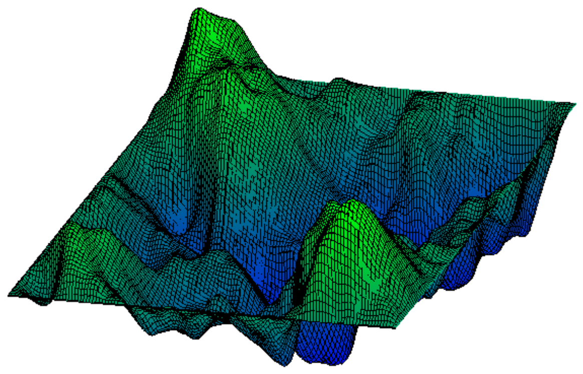


Figure 21: The resulting landscape after smoothing with Bézier patches and then the Lagrange interpolating polynomial.

interpolating polynomials, and approximating polynomials and surfaces. Applying these methods to the landscape data produces a smoother landscape.

Image processing techniques such as those based on Fourier and wavelet analysis were considered. Using the Gaussian function in the frequency domain produced a landscape without any sharp peaks or valleys. Since the Fourier basis functions are not compactly supported, care must be taken when choosing a low-pass filter. This will avoid unintended consequences on another region of the landscape. The wavelet techniques did not produce the best results. While these techniques work well in image processing where a small region of influence is desired, Fourier analysis is better suited to the wave-like properties desired in a terrain of rolling hills.

The Lagrange interpolating polynomial did not reduce the peaks and valleys by a noticeable amount. Since it interpolates the data points, resampling at additional points is necessary to avoid regenerating the same landscape. An increased number of points requires more computation to generate the landscape which may not be desirable. Also, additional points require increased memory requirements, which may also not be desirable. Finally, the resulting landscape was not much of a visual improvement over the original landscape.

Bézier curves were considered next in an attempt to avoid interpolation. These curves did an acceptable job of smoothing the terrain. However, they are one-dimensional and must be applied twice over the landscape. The endpoints of each Bézier curve present a problem since the curves are formed from the Bernstein basis polynomials. These basis functions are defined to interpolate the endpoints. Without special consideration for the endpoints of each curve, every fourth point would remain unchanged. One solution to this difficulty is to average neighboring endpoints.

Smoothing with the Bézier patch also eliminated the sharp peaks and valleys. Since the patch is two-dimensional, a neighborhood of surrounding points affects each point on the patch. In addition, the original data points are used in both dimensions, so the final result more closely resembles the original landscape. The patch interpolates its corner points so these points are averaged with other surrounding points. Because polynomial interpolation did seem to marginally reduce peaks and valleys, this method was applied after smoothing with Bézier patches. This produced nice results, but at a higher computational cost. If this result is desired, sampling the patch more often will produce similar results.

A direction for future research would be to further analyze Fourier and wavelet image processing techniques when applied specifically to images. This paper merely introduced the idea of using such methods and as such presented very elementary techniques. More advanced techniques could be considered as well as experimenting with different low-pass filters. Another area would be to consider the effect of B-splines and B-spline patches. Extending one-dimensional B-splines to a two-dimensional patch is done in the same manner as extending Bézier curves to patches. Finally, objects known as NURBS (non-uniform rational B-splines) should be considered since they are often used in modeling two-dimensional curved surfaces. It is unclear whether the nonuniformity would help since the landscape is defined over a uniform grid. However, this could open an entirely different branch of landscapes generated over nonuniform grids.

Acknowledgements

Thanks to Professor Michael Main for introducing fractal landscape generation, Professor James Curry for allowing the initial opportunity to investigate these landscapes and smoothing techniques to a greater depth, and Professor François Meyer for help with the image processing aspect of this paper.

Word count: 7635. This excludes the abstract, section headers, captions, mathematics, parenthetical comments of a reference nature (e.g. "See Figure x"), acknowledgements, and references.

References

[1] Adobe Systems Inc. *Adobe Photoshop 7.0 User Guide for Windows and Macintosh.* Adobe Systems Inc., 2002.

[2] Kenichi Arakawa and Eric Krotkov. "Fractal Modeling of Natural Terrain: Analysis and Surface Reconstruction with Range Data". *Graphical Models and Image Processing*, 58(5):413–436, Sep 1996.

[3] L. T. Bruton and N. R. Bartley. "Simulation of Fractal Multidimensional Images using Multidimensional Recursive Filters". *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41(3):181–188, Mar 1994.

[4] Richard L. Burden and J. Douglas Faires. *Numerical Analysis.* Brooks/Cole, Boston, seventh edition, 2001.

[5] E. Catmull and J. Clark. "Recursively generated B-spline surfaces on arbitrary topological meshes". *Computer Aided Design*, 10(6):350–355, 1978.

[6] Stephen Chenney. "B-Spline Blending Functions", Spring 2001. http://www.cs.wisc.edu/ schenney/courses/cs559-s2001/lectures/lecture-22-online.ppt.

[7] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". *Mathematics of Computation*, 19(90):297–301, Apr 1965.

[8] A. K. Dewdney. "Computer Recreations: Of fractal mountains, graftal plants and other computer graphics at Pixar". *Scientific American*, 255(6):14–20, Dec 1986.

[9] D. Doo and M. Sabin. "Behavior of recursive division surfaces near extraordinary points". *Computer Aided Design*, 10(6):356–360, 1978.

[10] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice.* Addison-Wesley Publishing Company, Reading, Massachussetts, second edition, 1996.

[11] Alain Fournier, Don Fussell, and Loren Carpenter. "Computer Rendering of Stochastic Models". *Communications of the ACM*, 25(6):371–384, Jun 1982.

[12] Kevin Hawkins and Dave Astle. *OpenGL Game Programming*. Prima Publishing, Roseville, California, first edition, 2001.

[13] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. "Terrain Simulation Using a Model of Stream Erosion". *Computer Graphics*, 22(4):263–268, Aug 1988.

[14] Robert Krten. "Generating Realistic Terrain". *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 19(7):26–28, Jul 1994.

[15] Kyle G. Freeman (to Novalogic, Inc.). U.S. Patent 5,550,959, Aug 1996.

[16] Kyle G. Freeman (to Novalogic, Inc.). U.S. Patent 6,020,893, Feb 2000.

[17] Jeffrey M. Lane and Richard F. Riesenfeld. "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1), Jan 1980.

[18] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, New York, 1983.

[19] Gavin S. P. Miller. "The Definition and Rendering of Terrain Maps". *Computer Graphics*, 20(4):39–48, Aug 1986.

[20] John R. Moss. Fractal Terrain Generation System Specification, May 1997. http://www.cosc.brocku.ca/Project/info/moss/system/system32.html.

[21] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. "The Synthesis and Rendering of Eroded Fractal Terrains". *Computer Graphics*, 23(3):41–50, 1989.

[22] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images*. Springer-Verlag, New York, 1988.

[23] Clifford A. Pickover. "Generating Extraterrestrial Terrain". *IEEE Computer Graphics and Applications*, 15(2):18–21, 1995.

[24] B. S. Daya Sagar and K. S. R. Murphy. "Generation of a Fractal Landscape using Nonlinear Mathematical Morphological Transformations". *Fractals*, 8(3):267–272, 2000.

[25] Brian Sharp. "Implementing Curved Surface Geometry". *Game Developer*, pages 42–53, Jun 1999.

[26] Brian Sharp. "Optimizing Curved Surface Geometry". *Game Developer*, pages 40–48, Jul 1999.

[27] Joost van Lawick van Pabst and Hans Jense. "Dynamic Terrain Generation Based on Mulitfractal Techniques". In *Proceedings of the International Workshop on High Performance Computing for Computer Graphics and Visualisation*, pages 186–199, 1996.

[28] Luiz Velho. "Using Semi-Regular 4-8 Meshes for Subdivision Surfaces". *Journal of Graphics Tools*, 5(3):35–47, 2000.

[29] James S. Walker. "Fourier Analysis and Wavelet Analysis ". *Notices of the AMS*, 44(6):658–670, Jun/Jul 1997.

[30] Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, Boston, third edition, 1999.