Answers of exercises for

Chapter 2: From Graphics to Visualization

Note: Depending on the actual exercise type, the answers outlined below range from *exact* ones (*e.g.* in cases where the stated question required providing an exact formula) to *indicative* ones (*e.g.* in cases where the stated question involved providing a design suggestion). In all cases, instructors are encouraged to revisit the provided questions and corresponding answers to refine them, *e.g.* to produce more focused testing on specific subdomains of the taught material.

1 EXERCISE 1

This exercise aims to provoke the student in addressing use-cases for the visualization methods that will be introduced later on in Chapters 5 and 10, such as slices, isosurfaces, and volume rendering. We do not expect that these methods will be listed here as potential answers. Rather, the aim is to expose the students to the challenges of high-variate data visualization, and to comment their proposed solutions in terms of advantages and limitations. This way, the students will arguably get a better understanding of the added-value of the visualization methods presented further in Chapters 5 and 10.

Example visualization methods that students could list here (including limitations) can be the following:

1. *Slices:* Sample the range $[z_{min}, z_{max}]$ of the z parameter at a number of points z_i, *e.g.* equally spread over that range. For each such value z_i, draw a 3D plot $t_i = f(x, y, z_i)$, where z_i is a fixed value. Display the plots of t_i side-by-side in a matrix or table layout. Use the same viewpoint for all 3D plots. This solution requires a relatively large display space to show all plots at the same time.

2. *Stacked slices:* A variant of the above solution is to draw all 3D plots t_i in the same view or screen

space, using color to emphasize the value of z_i and transparency in order to 'see through' the different plots. This requires less screen space than the previous solution. However, blending together a large number of plots having different colors is going to cause color artifacts and occlusion.

3. *Animation:* Display a single 3D plot $t = f(x, y, z_0)$, where the value z_0 is fixed. Next, allow z_0 to change, either interactively by the user by means of *e.g.* a slider, or by animating z_0 over the range $[z_{min}, z_{max}]$. This solution requires less display space than the 'slices' solution. However, it does not show plots for different values of z at the same time.

2 EXERCISE 2

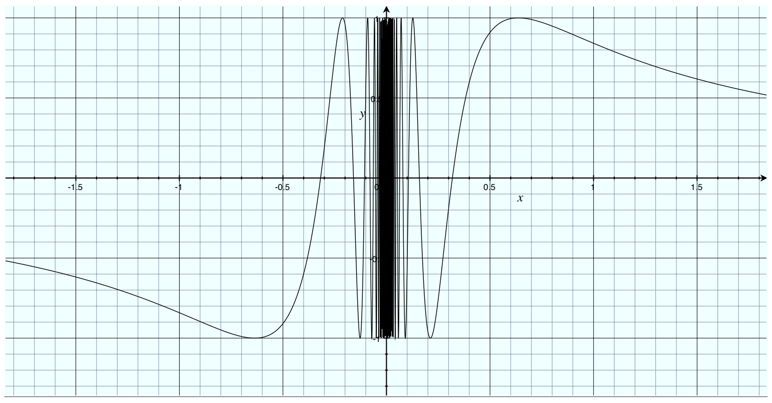
Solving this problem in general is very hard, as the potential ranges of the x and y variables are infinite (the entire real axis). However, several solutions can be proposed:

1. *Overview and detail:* Consider two ranges $R_x = [x_{min}, x_{max}]$ and $R_y = [y_{min}, y_{max}]$ for the x and y variables respectively. We call these the *overview* ranges. Within these ranges, we define two smaller ranges $r_x \subset R_x$ and $r_y \subset R_y$. We call these the *detail* ranges. Next, we display our function sampled over the detail ranges *i.e.* over $r_x \times r_y$. Along this, we display two 1D bars showing the overview ranges R_x and R_y and, highlighted within these, the detail ranges r_x and r_y. We use interaction to zoom in/out the detail ranges within their respective overview ranges, and update the 1D bars to show this effect. Separately, we can use interaction to scroll, or pan, the detail ranges within their respective overview ranges. Finally, we can use interaction to zoom and pan the overview ranges over the entire real axis. The overall design resembles the effect given by sliders and scrollbars for a text document.

2. *Areas of interest:* Assume that we are interested only in specific areas of our function-domain, *e.g.* only in areas where it takes non-zero values, or values larger than a certain threshold, or where it has a rapid variation. If we can detect such areas, either analytically or by analyzing the sampled values of the function, we can next preset the ranges R_x and R_y to cover these areas. As a refinement of this technique, assume that the areas of interest are not compact intervals. In that case, we can highlight the areas of interest along the 1D bars proposed in the previous solution, *e.g.* using color. Next, the user sees where interesting events occur over the overview ranges R, and can zoom and pan the detail ranges r to focus on such areas.

3 EXERCISE 3

The plot of the function $z = \sin(1/x)$ is shown below (for a small range of x centered around the origin). We clearly see the clutter around the origin, shown by the thick black band in the figure.



Plot of function $z = \sin(1/x)$.

Several techniques can be used to alleviate the clutter problem, as follows.

First, we can use the overview-and-detail techniques presented for Exercise 2 to allow users to selectively zoom in in the area around the origin. If this is done interactively, zooming in and out repeatedly can convey a better idea of how the function evolves as it approaches the value $x = 0$. An extension of this idea is to display several plots for nested and increasingly narrower ranges $[x_{min}, x_{max}]$ centered around the origin. Each plot will thus convey a different level of detail. Separately, we can use alpha blending when drawing the graph of the function. By using an opacity value $\alpha < 1$, the opacity of the function graph will convey the (increasing) frequency of the function, and partially alleviate the clutter problem. Letting the user interactively vary the α value, *e.g.* by means of a slider, allows visually exploring the variation of the frequency of the function as it approaches $x = 0$.

4 EXERCISE 4

We essentially have the problem of visualizing a function f whose results are tuples of values rather than individual values.

If the function takes values in $\mathbb{R}^2$, *i.e.* its values are pairs of real numbers, several simple solutions exist. First, we can 'split' the function into two real-values functions $z_1 = f_1(x, y)$ and $z_2 = f_2(x, y)$, and draw two superimposed height plots for z_1 and z_2. We can use two different colors for the two height plots to distinguish them in the superimposed view. Alternatively, we can draw a single height plot $z_1 = f_1(x, y)$, and color each plot point by the value z_2, using a color map that maps real values to colors. This way, the plot height conveys the value of z_1, while the plot color conveys the value of z_2. The advantage of the superimposed plot solution is that it makes it easy to compare values of z_1 and z_2 at the same point. However, it may create undesired occlusions. The advantage of the height-and-color solution is that suffers less from occlusion. However, it makes it hard(er) to compare values of z_1 and z_2, as one value is mapped to height and the other one to color.

Both above solutions have problems when increasing the number n of tuple elements beyond a few. For example, the superimposed plot solution will create too much occlusion when n is larger than 3..5.

Transparency can alleviate this problem only up to a limited extent. The second solution – encoding each tuple dimension z_i in a separate visual channel – is also limited to the number of independent visual channels that we can reliably encode at a single screen location. Examples of such channels are height, hue, transparency, texture, and luminance. Clearly, these channels are not fully independent. As such, the number of variables we can reliably visualize in the same time will be limited to a small value (3..5).

An alternative solution follows the solution of Exercise 1: For each tuple value z_i, we display a separate 3D plot $z_i = f_i(x, y)$ in a separate window. Next, we can use interaction to select a point in all plots simultaneously, and display the values z_i as *e.g.* annotations. While this solution scales to higher values of n (tuple size), it puts a higher burden on the user to visually correlate all the multiple views to reason about the data variation.

More advanced solutions for this problem are discussed in Chapter 11 (multivariate data visualization).

5 EXERCISE 5

The solution for this problem is closely related to Exercise 4. We can model our data either as two scalar functions $\textit{rainfall} = f_1(x, y)$ and $\textit{accuracy} = f_2(x, y)$, or as a single two-valued function $(\textit{rainfall}, \textit{accuracy}) = f(x, y)$. For the visualization of such a function, the best is to use a single view, so we can easily correlate the rainfall and accuracy values at a single location. One simple way to do this is to draw a height plot of the rainfall, and modulate its opacity at each vertex by the accuracy value. This way, inaccurate measurements will show up as more transparent plot regions; such regions are harder to see, which is intuitively in line with the fact that their measured values are harder to assess (since inaccurate). Using the alternative solutions presented for Exercise 4 (superimposed plots or separate plots) is less optimal, since these solutions do not allow us to easily correlate measurements and accuracies.

6 EXERCISE 6

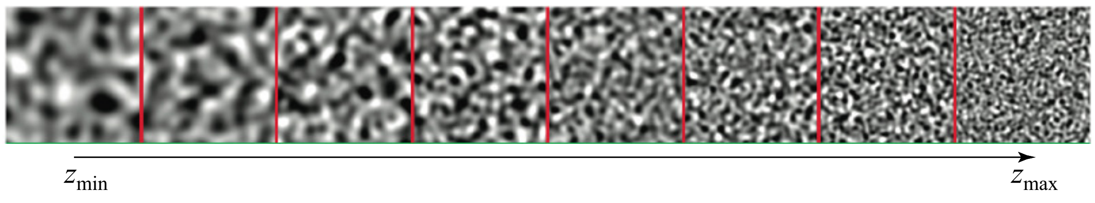
The illumination of a point on a 3D surface, as captured by the Phong model, depends mainly on three elements: the surface normal at that point, the light direction, and the view direction. Thus, the answer to this question depends largely on the freedom offered to the user to manipulate the view direction (since the surface normal is a given' characteristic, while the light direction is something our application can control).

If the user is allowed to freely manipulate the view direction and camera location (eye point), we cannot make many assumptions about these parameters. In this case, a good preset is to fix the light direction to be equal (or close to) the view direction. This way, we are sure that surfaces we 'look at' will appear bright in the visualization. The effect is often used in 3D graphics, and known under the name of *headlight* (the illumination is as if the viewer carried a light fixed to his/her head and oriented along the view direction). The main added-value of this solution is that we avoid having the entire plot dark in the view.

7 EXERCISE 7

Consider a function $z = f(x, y)$ where $z = (z_1, z_2) \in \mathbb{R}^2$ is a pair of two real values. For instance, consider that we measure temperature and rainfall over a 2D spatial domain. We would like next to display this function using the classical height plot metaphor. For this, we need to map two scalar values (z_1, z_2) at each point (x, y). One way to do this is to map z_1 to height and z_2 to color. However, there may be cases when color needs to be reserved for other purposes, or is not usable (like in case we have a grayscale output). In such cases, we could map z_2 to the texture pattern.

To do this, we need to imagine a set of textures whose appearance conveys us different scalar values. One way to create such textures is to consider that the texture is a random signal, whose *frequency* or graininess pattern encodes the scalar value. The figure below shows one such possible design: The eight different textures shown there range from coarse-scale, low-frequency luminance patterns, to small-scale, high frequency patterns. Such textures can be precomputed or generated on-the-fly. By texture-mapping a 3D plot with such patterns, we can thus map one scalar value to the texture pattern in the plot. Since the texture does not use color (hue), we can keep the color channel free to show additional aspects.

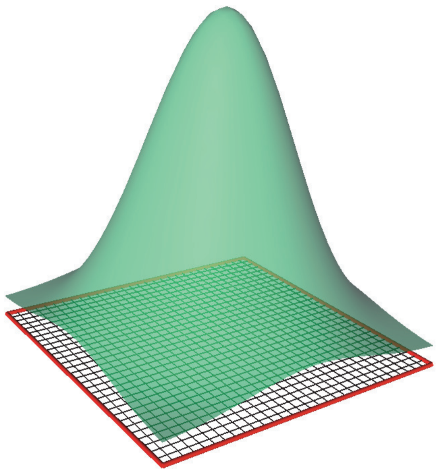


Texture patterns encoding a scalar value z.

Other texture designs are possible, as long as (a) they can be easily generated; (b) there exists a 'natural' and intuitive order of the different textures (used to map scalar values); and (c) different texture patterns can be easily stitched together. However, one important limitation of mapping scalar values to textures is that this mechanism has a limited *resolution*: We cannot easily distinguish more than a few such texture patterns; and the more patterns we want to use, the larger needs to be the area allocated to each pattern in screen space (thus, the fewer sample points in (x, y) space can we take).

8 EXERCISE 8

Transparency basically 'mixes' the color of a point $\mathbf{x}$ (*e.g.* on the surface of a 3D shape) with colors of other points $\mathbf{x}_i$ situated behind that point $\mathbf{x}$ along a view ray, and also with the background color. In our example figure (shown here below), the green color of the plot surface is mixed with the red and black colors of the grid shape and the white background color. If too many such colors are mixed at a single location, it can be hard to understand what type of object (or objects) are visible at that location.



Transparent height plot (see Chapter 2).

For instance, consider that we visualize a 3D plot of a function $z = f(x, y)$ which has many dense variations along the z direction. If we use transparency, we will effectively 'see through' a large number of 3D surfaces at each screen pixel. The resulting color will thus be highly dependent on the number of such surfaces that exist along a view ray *and* on the illumination of each such surface. As such, it can be (very) hard to understand what is the actual shape we are looking at. For instance, it is hard to tell which shapes are in front and which ones behind. Interactive viewpoint manipulation can help to understand depth information, but only up to a certain extent.

Additionally, imagine that we map some scalar value defined on a 3D shape to *color*, and also use transparency to visualize the color mapped surface. If multiple surface intersections exist with a view ray, then the resulting color (at the pixel corresponding to that view ray) will be a linear combination of the colors at the intersection points. However, such a resulting color may not represent any meaningful data value according to our color mapping scheme!

One partial solution to these problems involves reducing the amount of transparency used in the visualization. This way, we cannot 'see through' too many surface layers, and thus limit the depth-perception problems. One particular instance of this solution involves drawing the majority of the shapes present in a visualization as opaque, followed by drawing a limited number of shapes with transparency atop the opaque shapes. This is the solution used in the height plot in the above figure. Separately, one should avoid using color mapping (of data values) together with transparency in case that the color-mapped shape has many concavities (and thus, admits several intersections of a view ray with color-mapped surface fragments).

9 EXERCISE 9

For all applications where reasoning about the actual shape of the displayed objects, *e.g.* where we want to compare sizes along the x and y screen axes, we should avoid using different scaling factors for the x and y viewpoint axes. Most visualizations involving the depiction of 'natural' 3D shapes, such as anatomical structures or other real-world natural or synthetic shapes, fall in this category.

However, many visualizations depict *abstract* shapes, which do not have a 'natural' aspect ratio of their x and y sizes. Consider for instance displaying a 2D diagram, or graph drawing, consisting of nodes (boxes) connected by edges (lines or curves). In such a diagram, the relative position of elements and the connections shown between elements is conveying information. Additionally, the relative sizes of elements may convey information. Stretching or compressing the image along the x or y axes would not change relative positions, displayed connections, or relative sizes. As such, in these cases, using non-uniform scaling can be appropriate.

10 EXERCISE 10

The answers to the three sub-questions follow:

- For each combination of values of the x and y variable, we do have a unique value z. Thus, the answer is affirmative: We can represent the dataset as a function $z = f(x, y)$. Note that, depending on our actual data, it may not be so that all combinations of values x (time) and y (stock price) make sense. For instance, for a given time moment x, only a specific set of values y (stock prices) do exist. This means that our function f would not be defined over the Cartesian product of the ranges of x and y, but a subset of the (x, y) space. However, this does not change the functional nature of the dependence $z = f(x, y)$.

- Since our data can indeed be modeled as a function $z = f(x, y)$, then we can indeed depict the data by using a height plot. Compared to a 'classical' height plot of a function $f \to \mathbb{R}$, the main difference is that our function takes values in the ordinal domain described by our five-point valuation scale rather than on the continuous real axis. Thus, at each point (x, y), we have to display one of the possible five values that our function takes there. Since our valuation scale is ordinal, we can map these values to five height levels $z_1 < \ldots < z_5$ to generate our height plot. A point related to this concerns *interpolation*: For real-valued functions $z = f(x, y)$ which are also continuous, we typically construct heigh plots by using polygons connecting their sample points, and emphasize continuity by using smooth shading. For our stock data, it is not evident that the underlying function $z = f(x, y)$ is continuous. If this is not the case, we should draw the plot as a sequence of separate points $z_i = f(x_i, y_i)$ rather than a continuous surface.

- For each combination of values of x (time) and y (stock price), we have a unique value z representing the recommended stock. Thus, the data can indeed be represented by a function $z = f(x, y)$. The difference is that our function now takes values in a set of stock-names rather than in a set of valuations. Representing this function as a height plot *can* be done technically, but is arguably of little meaning and use. Indeed, to do this, we would need to map the set of our stock names (or stock IDs, for that purpose) to the real axis z. This could be done *e.g.* by sorting stock names alphabetically and assigning to each stock name a value z equal to the position of that stock name in the sorted list. However, the height plot thus constructed would be almost surely meaningless. Indeed, what we are interested to *see* in this dataset is the *identity* of the stock pick at a given time-price combination, rather than the alphabetical *order* of that stock name among all existing stocks. Such a height plot *may* be a useful metaphor when we have a very small set of stock names (*e.g.* three or four). In such cases, we could easily map the height z of the height plot back to the identity of the stock. However, if our stock-set had hundreds of different items, this mapping would be very hard or even impossible to do.

11 EXERCISE 11

There are multiple solutions for this problem. A few are sketched below (more options could be presented by students):

One solution is to vary the polygon vertex transparency values, so as to make important map areas more salient, *e.g.* less transparent, and unimportant map areas as less salient, *e.g.* more transparent. The drawback of this solution is that the structure of the map becomes less visible in low-importance areas, which can impair the overall understanding of the geography.

A second solution is to draw super-imposed annotations atop of the important areas, such as markers or labels. This solution does not decrease the visibility of unimportant areas, but rather attracts the attention of the user to the important areas. A drawback of this solution is that it requires a robust and simple way to determine which areas are ‘important’, and the careful placement of markers so as not to create too much occlusion. Separately, markers will inevitably occlude the underlying map information.

A third solution is to use a light shining from above the map surface, and change the material parameters, *e.g.* ambient, diffuse, or specular material properties of the surface, as a function of the signal of interest (importance). When interactively navigating in the scene, or when moving the light position, the shading of important areas will change more dramatically than for other areas, thereby attracting the userÕs attention to these areas. The advantage of this solution is that it does not affect the map transparency or geometry (as it was the case for the previous two solutions). The disadvantage is that encoding values in terms of shading has a limited dynamic range; also, shading depends not only on local material properties, but also on local surface properties (normals). As such, it is hard to reliably encode a signal having a wide dynamic range in shading.

End of answers of exercises for
Chapter 2: From Graphics to Visualization