

# How to Easily Create Stunning 3D Plots in R Using 'rgl'

Authored by  
**stats writer**

December 4, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to Easily Create Stunning 3D Plots in R Using 'rgl'*.  
PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=104998>

Creating 3D plots in R is a powerful method for visualizing complex relationships between three variables, providing insight into surfaces and spatial distributions that flat, two-dimensional graphs cannot capture. While several packages exist for advanced interactive visualization, such as the widely respected rgl package, the simplest path often begins with R's built-in graphics capabilities. The initial approach using the rgl package involves meticulously defining the spatial coordinates (x, y, z) for points, lines, or surfaces. Once these fundamental points are established, users gain granular control to enhance the visual representation by adding customized elements like sophisticated color schemes, realistic lighting effects, and various other graphical attributes.

Furthermore, packages designed for 3D visualization offer dynamic functionalities essential for comprehensive data exploration. These functions allow users to manipulate the view of the data interactively, such as rotating the plot to examine different angles or overlaying multiple geometric objects within the same three-dimensional space. While the rgl package offers robust interactivity, the core functions available in base R provide an efficient starting point for generating static surface visualizations. These static plots, generated through base graphics, are crucial for introductory visualization and often serve as the foundation for more advanced rendering techniques.

## The Foundation: Utilizing the Base R **persp()** function

The most straightforward and accessible method to create a static 3D plot directly within the R environment, without relying on external dependencies, is by employing the **persp()** function. This function stands for perspective plot, and it is specifically designed to render surfaces defined by a functional relationship between X, Y, and Z coordinates. Achieving a meaningful visualization requires the input data to be structured precisely: two vectors (X and Y) defining the grid, and a Matrix (Z) containing the height values corresponding to every (X, Y) intersection point.

The general syntax for invoking this powerful function is remarkably simple, though the data preparation preceding it requires careful attention. Understanding this basic structure is the first step toward generating complex surface visualizations. The data supplied to Z must be a Matrix where the dimensions align perfectly with the lengths of the X and Y vectors, effectively mapping the height or function output across the defined two-dimensional plane.

### **persp(x, y, z)**

While the core function call requires only these three arguments, the true power of **persp()** lies in the dozens of optional parameters available for detailed customization, ranging from viewing angles and axis labels to color shading and lighting effects. The following comprehensive examples will demonstrate how to utilize this function in practical scenarios, moving from a bare-

bones plot to a highly customized, publication-quality graphic. We will explore how simple mathematical functions can be translated into stunning visual surfaces, illustrating the mathematical concept of a function of two variables.

## Prerequisites for Data Preparation: The XYZ Structure

Before calling the `persp()` function, the data must be organized into the required XYZ format. The X and Y arguments must be vectors defining the coordinates along the respective axes. The Z argument, however, must be a numerical Matrix containing the height of the surface at the grid points defined by the outer product of X and Y. This Matrix structure is non-negotiable for `persp()`, as it establishes the continuous surface being visualized.

Typically, when plotting a mathematical function, the Z Matrix is calculated using the `outer()` function in R. The `outer()` function takes the two vectors (X and Y) and the defined function, applying the function across every possible combination of x and y values. This results in the necessary grid of Z values, which forms the topographical data for the `persp()` function. Understanding this data transformation is key to successfully generating surface plots from raw mathematical expressions or empirical data.

The resulting plot offers a static snapshot of the surface, which is ideal for presentations or published documents where interactivity is not strictly required. While base R 3D graphics may lack the dynamic rotation capabilities of packages like rgl package, they are highly reliable, require no package installation, and offer extensive graphical control through parameters, as we will explore in the subsequent examples.

## Example 1: Constructing the Basic 3D Surface Plot

This first example illustrates the minimum necessary steps to generate a 3D plot using the base `persp()` function. We will define simple ranges for X and Y, create a mathematical function, and then use the `outer()` command to generate the corresponding Z Matrix. The chosen function,  $z = \sqrt{x^2 + y^2}$ , represents a cone-shaped surface, which serves as an excellent foundational shape for visualization.

The initial steps involve defining the range vectors. By setting both X and Y from -10 to 10, we establish a square grid covering 21 units along each dimension. Next, we define `z_values` as a function that accepts two arguments, X and Y, and returns the calculated height (Z). The use of the `sqrt(x ^ 2 + y ^ 2)` expression ensures that the height is proportional to the distance from the origin (0,0), creating the characteristic cone shape.

Finally, the `outer()` function efficiently calculates the Z Matrix by pairing every element in X with every element in Y and applying the `z_values` function to them. This populated Z Matrix is then

passed to the `persp()` function, which renders the surface plot using default settings for color, perspective, and axis labels. This basic plot provides a fundamental visualization but lacks the aesthetic refinements necessary for professional analysis.

**#define x and y**

```
x <- -10:10
```

```
y <- -10:10
```

**#define function to create z-values**

```
z_values <- function(x, y) {
```

```
  sqrt(x ^ 2 + y ^ 2)
```

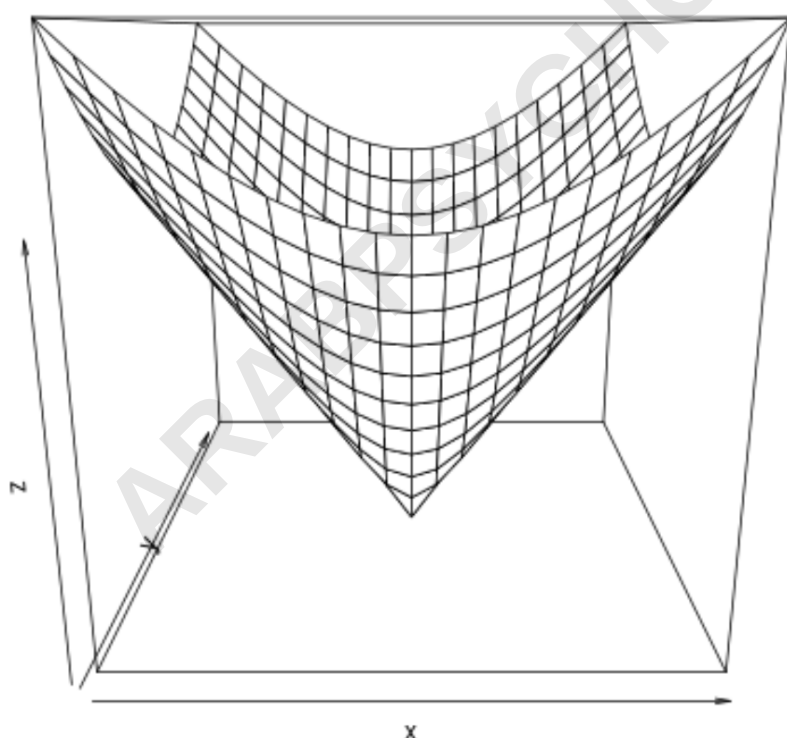
```
}
```

**#create z-values**

```
z = outer(x, y, z_values)
```

**#create 3D plot**

```
persp(x, y, z)
```



## Example 2: Enhancing Visual Appeal through Customization

While the default `persp()` function plot provides the necessary structure, it often requires significant

customization to meet professional standards for clarity and aesthetics. This example demonstrates how to control fundamental visual elements, including explicit axis labels, a descriptive title, surface color, and crucial shading effects, which significantly enhance the perception of depth and three-dimensionality.

Key arguments introduced here are `xlab`, `ylab`, and `zlab` for labeling the axes, and `main` for assigning a primary title to the plot. These parameters ensure that the graph is fully self-explanatory, clearly defining the variables being visualized. Furthermore, two vital aesthetic parameters are utilized: `col` and `shade`. The `col` argument specifies the color used to fill the surface facets. We have chosen 'pink' here, but any valid R color name or hexadecimal code can be used.

The `shade` argument is particularly important for 3D representation. It takes a value between 0 and 1, controlling the intensity of shading used to simulate directional lighting. By setting `shade=.4`, we introduce a moderate level of simulated shadow and highlight, which makes the surface contours more distinct and gives the observer a better sense of the plot's true 3D plots shape. Without proper shading, the plot can often appear flat or confusing.

**#define x and y**

**x <- -10:10**

**y <- -10:10**

**#define function to create z-values**

**z\_values <- function(x, y) {**

**sqrt(x ^ 2 + y ^ 2)**

**}**

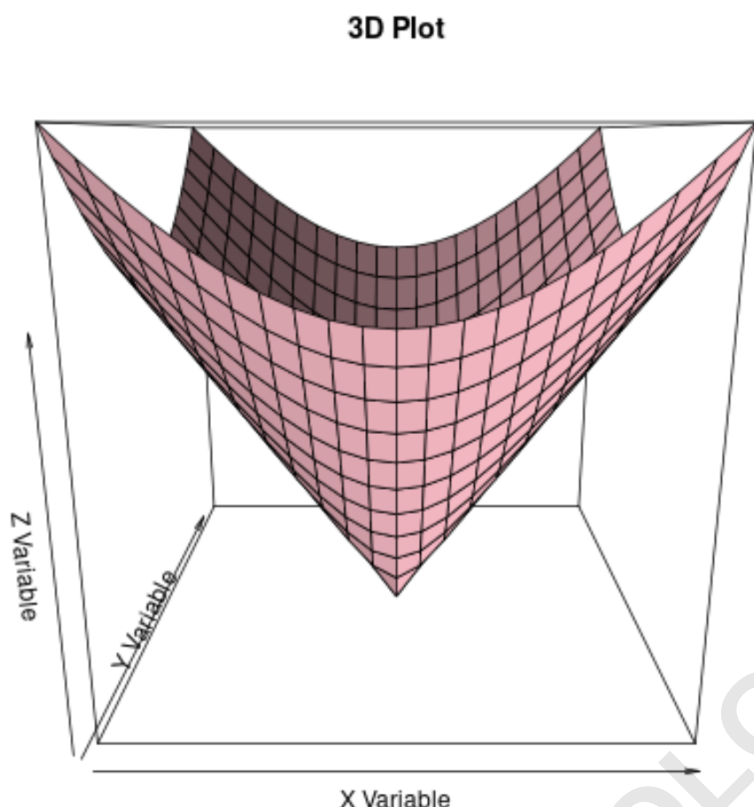
**#create z-values**

**z = outer(x, y, z\_values)**

**#create 3D plot**

**persp(x, y, z, xlab='X Variable', ylab='Y Variable', zlab='Z Variable',**

**main='3D Plot', col='pink', shade=.4)**



### Example 3: Manipulating Perspective through Rotation

One of the major challenges in static 3D plots is selecting the optimal viewpoint. If the viewing angle obscures critical features of the surface, the graph loses its explanatory power. The **`persp()`** function addresses this by providing two essential arguments for viewpoint control: `theta` and `phi`. These parameters allow the user to precisely specify the horizontal and vertical rotation of the plot, respectively, making it possible to highlight specific data features or present the most informative perspective.

The `theta` argument controls the azimuth angle, which is the angle relative to the horizontal plane (analogous to rotation around the Z-axis). It is specified in degrees. A value of 0 degrees corresponds to viewing the plot from the positive Y-axis, while 90 degrees views it from the positive X-axis. By setting `theta = 30`, we rotate the plot 30 degrees clockwise from the default viewpoint, offering a better view of the sides.

The `phi` argument controls the co-latitude angle, determining the vertical viewing angle (elevation). It is also specified in degrees, where 0 degrees means viewing the plot from directly above, and 90 degrees means viewing it horizontally. Setting `phi = 15` raises the viewpoint slightly above the horizontal plane, providing a more natural and less distorted perspective of the surface. Experimenting with these two rotation parameters is often necessary to find the angle that best

communicates the surface structure of the data.

**#define x and y**

```
x <- -10:10
```

```
y <- -10:10
```

**#define function to create z-values**

```
z_values <- function(x, y) {
```

```
  sqrt(x ^ 2 + y ^ 2)
```

```
}
```

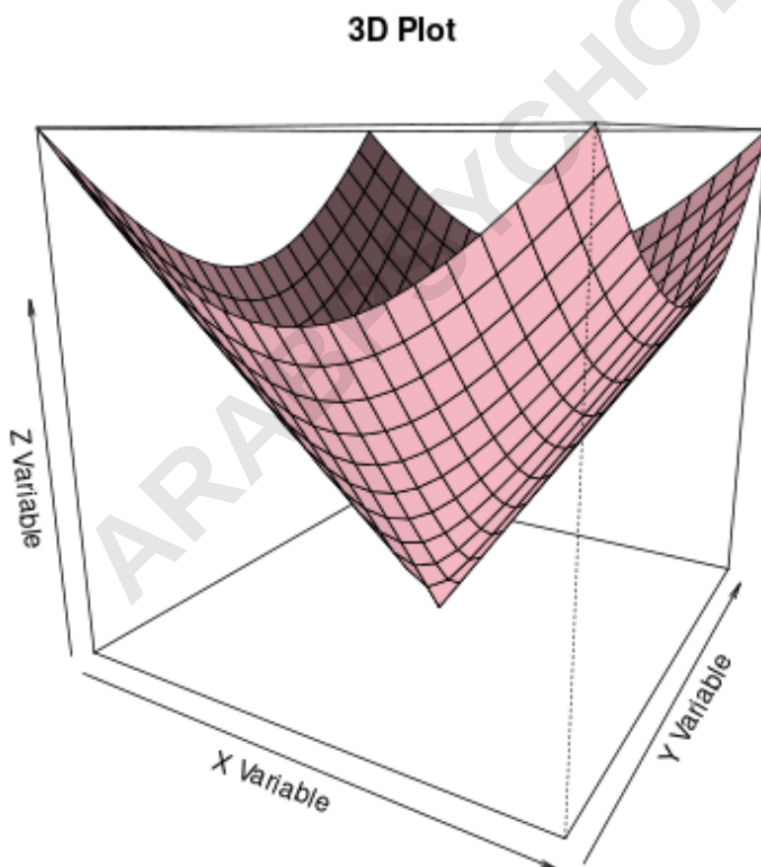
**#create z-values**

```
z = outer(x, y, z_values)
```

**#create 3D plot**

```
persp(x, y, z, xlab='X Variable', ylab='Y Variable', zlab='Z Variable',
```

```
main='3D Plot', col='pink', shade=.4, theta = 30, phi = 15)
```



## Example 4: Improving Clarity with Axis Detail and Tick Marks

A common limitation in default 3D visualizations is the lack of detailed axis markings, making it difficult for the viewer to accurately determine the scale and specific values along the axes. The `persp()` function addresses this crucial need for precision through the `ticktype` argument. By default, `ticktype` is set to 'simple', which provides minimal tick marks on the axes. However, by changing this setting, we can significantly improve the readability of the graph.

The most useful setting for enhancing axis clarity is `ticktype='detailed'`. When this value is supplied, R automatically calculates and draws detailed tick marks along all three axes (X, Y, and Z), ensuring that value labels are clearly placed on the perspective planes. This is especially important for the Z-axis, as the apparent height of the surface needs to be easily quantifiable. The 'detailed' setting also often improves the visualization of the bounding box surrounding the plot, grounding the surface within a defined coordinate system.

Incorporating `ticktype='detailed'` into the previous customized plot enhances its analytic utility, transforming it from a mere visual representation into a precise tool for data interpretation. This parameter should be considered mandatory for any scientific or technical visualization where precise numerical scaling is required for accurate assessment of the surface topology. This combination of rotation, shading, and detailed axes results in a robust, informative, and visually appealing static 3D plot.

**#define x and y**

```
x <- -10:10
```

```
y <- -10:10
```

**#define function to create z-values**

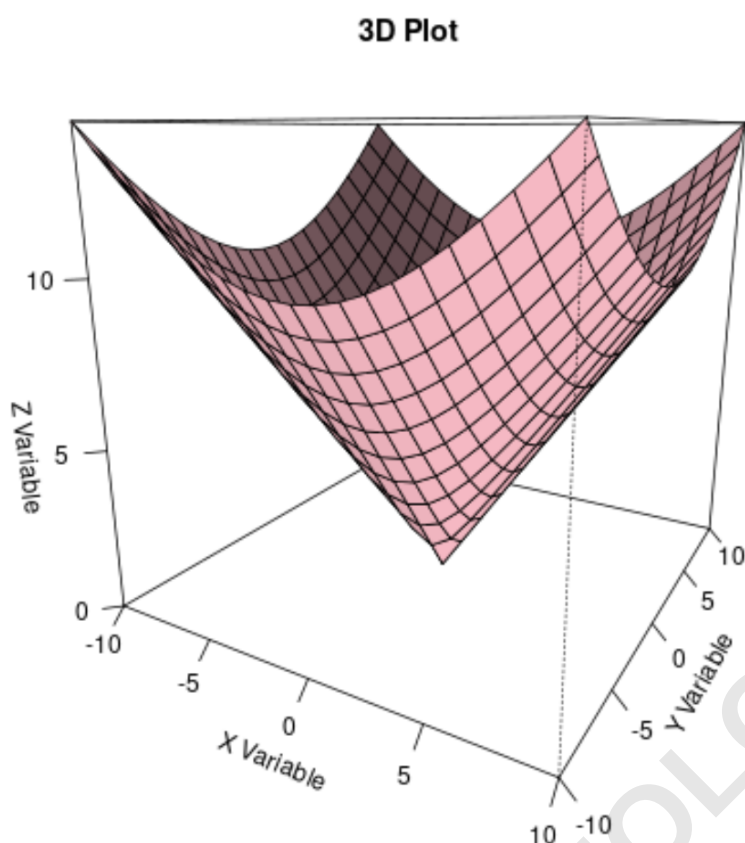
```
z_values <- function(x, y) {  
  sqrt(x ^ 2 + y ^ 2)  
}
```

**#create z-values**

```
z = outer(x, y, z_values)
```

**#create 3D plot**

```
persp(x, y, z, xlab='X Variable', ylab='Y Variable', zlab='Z Variable',  
main='3D Plot', col='pink', shade=.4, theta = 30, phi = 15, ticktype='detailed')
```



### Advanced Parameters and Further Customization of `persp()` function

Beyond the fundamental parameters demonstrated above, the **`persp()`** function offers numerous controls for high-level graphical refinement. Understanding these options allows for the creation of unique and informative visualizations tailored exactly to the data's characteristics. For instance, the `border` argument controls the color of the lines separating the facets of the surface. By setting `border = NA`, these lines are removed, often resulting in a smoother, more aesthetically pleasing look, particularly when the surface color is clearly differentiated by shading.

Another powerful parameter is `ltheta` and `lphi`, which control the direction of the simulated light source used for shading. While `shade` dictates the intensity of the effect, `ltheta` (azimuth of the light source) and `lphi` (colatitude of the light source) allow users to specify exactly where the virtual light is coming from. Manipulating the light source direction can sometimes reveal subtle topographical features that were hidden under the default lighting conditions, drastically impacting how local peaks and valleys are perceived.

Finally, parameters such as `zlim` can be utilized to set explicit limits for the Z-axis, ensuring consistency across multiple plots or focusing the visualization on a specific range of output values. These advanced customization features underscore the flexibility of R's base graphics system,

proving that high-quality, static 3D plots can be generated without resorting to complex external libraries.

## Moving Beyond Static Visualization: The Role of rgl package

While the base R persp() function is exceptional for generating static surface plots, data analysts often require interactive visualizations that allow users to explore the data dynamically. This is where specialized packages like rgl and plot3D become indispensable. The rgl package, specifically mentioned in the introduction, leverages OpenGL technology to create high-performance, interactive 3D graphics that can be rotated, zoomed, and manipulated in real-time within a separate interactive window.

The primary advantage of rgl is its capacity for rendering complex scenes, including points, lines, triangles, and even text, all within a fully navigable space. This interactivity is crucial when the optimal viewing angle is unknown or when presenting the plot to an audience that needs to examine multiple perspectives. Furthermore, rgl allows for the creation of animations, enabling the surface to rotate automatically, which is an extremely effective way to communicate the three-dimensional structure of the underlying data.

Ultimately, the choice between base R's **persp()** and packages like rgl depends on the project's requirements: use **persp()** for fast, reliable static visualization of surfaces, and employ rgl package when true interactivity, real-time navigation, and advanced geometric rendering are paramount to the analytical goal.

## Summary of R 3D Plotting Workflow

Creating robust 3D visualizations in R involves a logical three-step workflow, regardless of whether the static persp() function or an interactive package is used. Following this structured approach ensures data integrity and optimal visual communication.

**Data Preparation:** Define the X and Y coordinate vectors. Crucially, transform the data into the Z Matrix format, typically using the outer() function for mathematical functions or specialized data interpolation techniques for empirical data. The Matrix dimensions must align perfectly with the X and Y grid.

**Core Plot Generation:** Execute the primary plotting function (e.g., persp(x, y, z)). This step produces the fundamental structure of the surface before any aesthetic modifications are applied.

**Aesthetic Refinement and Customization:** Adjust parameters like theta and phi for perspective, col and shade for visual appeal and depth perception, and xlab/ylab/zlab along with ticktype for clarity and precision. This iterative process is essential for generating presentation-ready 3D plots.

By mastering the arguments of the **persp()** function, users can create complex, informative, and visually compelling surface visualizations directly within the base R graphics system, providing deep insights into multivariate data relationships.

The following tutorials explain how to create other common charts in R:

ARABPSYCHOLOGY.COM