Lecture 14: Graphics and Visualisation: 3D Plot

Line plots in 3D

* The `plot3` function is just like the `plot` function except that it accepts 3D data.
* The `plot3` function allows you to display line plots in 3D.

```
>> plot3(x, y, z); % plot3 also accepts a string specifying colour and line types.
```
* This form of plot is appropriate when you have two variables that are functions of a third variable:

```
x = f_1(z)  
y = f_2(z)
```
* Often the z variable represents time. This allows you to display how the x and y coordinates of a point vary over time.

3D line plot example

* For example, the following commands:

```
>> t = 0 : 0.1 : 10;         
>> x = exp(-0.2*t) .* cos(2*t); 
>> y = exp(-0.2*t) .* sin(2*t); 
>> plot3(x, y, t), xlabel('x'), ylabel('y'), zlabel('time');
```

produces the plot:

Control 3D plots

* If desired, axis ranges of 3D plots can be set with the `axis` command:

```
>> axis([xmin, xmax, ymin, ymax, zmin, zmax]);
```
* An important part of understanding a 3D plot is determining the orientation that you are viewing the data from.
* Use the following commands to help control a 3D plot:

```
>> box on; % Draws a 3D enclosing box. 
>> grid on; % Draws a grid on the axis surfaces.
>> rotate3d on; % Allows interactive viewing by dragging the mouse in the image.
```
Surface plotting in 3D

- The `plot3` function is suitable for displaying two variables that are a function of one independent variable:
  \[ x = f_1(t) \]
  \[ y = f_2(t) \]

- When you have a single variable that is a function of two independent variables, say:
  \[ z = f(x, y) \]
  then a surface display is more appropriate.

Matlab's `mesh`, `surf`, and `contour` functions

- The `meshgrid` function is extremely useful for generating the X and Y 2D arrays for a 3D plot.

```matlab
>> Xvals = -2 : 1 : 2;        % Generate axis vectors.
>> Yvals = -2 : 1 : 2;
>> [X, Y] = meshgrid(Xvals, Yvals)    % Make a mesh grid.

% X is a matrix with every element
X =
  -2 -1  0  1  2
  -2 -1  0  1  2
  -2 -1  0  1  2
  -2 -1  0  1  2
  -2 -1  0  1  2

% Y is a matrix with every element
Y =
  -2 -2 -2 -2 -2
  -1 -1 -1 -1 -1
   0   0   0   0   0
   1   1   1   1   1
   2   2   2   2   2
```

Meshgrid Example Continues

- Now calculate the function values on this mesh.

```matlab
>> Z = X .* exp(-(X.^2 + Y.^2));   % A 2D Gaussian .* X.

% Display the plot as a mesh.
>> mesh(X, Y, Z);
% Display the plot as a surface.
>> surf(X, Y, Z);
% Display the plot as a contour.
```
Mesh, Surface and Contour Plot

- By decreasing the increment in the axis vector generation commands, we can produce plots like:

Surface plotting with a single argument

- You can also invoke these surface displaying functions with a single argument.
- This single argument is treated as the 2D array of Z values for each point.
- The X and Y arrays default to a range of [1 .. the number of rows/columns of Z].
- For example:
  \[
  \text{>> surf}(Z)
  \]

Parametric surface representation

- This specification of surfaces by three separate arrays - one for each of the x, y and z coordinates of each data point is actually a parametric surface representation.
- A parametric surface is parameterized by two independent variables, \(i\) and \(j\), which vary continuously over a rectangle.
- You can then specify functions \(x(i,j)\), \(y(i,j)\), and \(z(i,j)\) which determine the way in which x, y and z coordinates vary with \(i\) and \(j\) over the surface.
- For example, the surface of the earth is parameterized in terms of longitude and latitude.
- You can draw a map of the earth as a flat rectangle, but at each longitude and latitude we can determine the x, y and z values that correspond to the actual location of that point in space.

2D arrays == flat?

- In discrete terms, Matlab represents parametric surfaces in terms of 2D arrays of X, Y, and Z values that correspond to each \((i,j)\) grid point on the surface.
- Hence the arguments to the mesh, surf, and contour functions are 2D arrays of points.
- In the plots shown so far, the X and Y 2D arrays are "flat", but there is no need for them to be like that...
A very wide variety of shapes can be represented, e.g. a triangular prism can be represented by:

\[
X = 
\begin{bmatrix}
1.0 & -0.5 & -0.5 & 1.0 \\
1.0 & -0.5 & -0.5 & 1.0 \\
0.0 & 0.866 & -0.866 & 0.0 \\
0.0 & 0.866 & -0.866 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
1.0 & 1.0 & 1.0 & 1.0
\end{bmatrix}
\]

Matlab provides some other handy functions for generating the points of parametric surfaces, e.g. `cylinder` and `sphere`:

\[
>> \{X, Y, Z\} = \text{cylinder}(\text{R}, \text{N});
\]

Generates an N sided cylinder. R is an array of radius values to be used along the length of the cylinder.

\[
>> \{X, Y, Z\} = \text{cylinder}([1 1], 3);
\]

-produces a 3 sided prism.

\[
>> \{X, Y, Z\} = \text{cylinder}([1 0], 4);
\]

-produces a 4 sided pyramid.

\[
>> \{X, Y, Z\} = \text{sphere}(\text{N});
\]

Matlab’s default way of displaying a surface is to shade each facet with a colour that is a function of the facet’s nominal z value (its height).

Matlab also superimposes the black mesh lines.

Often a nicer way to display a surface is with interpolated shading.

This is done with the `shading interp` command.

For example:

\[
>> \text{surf}(X, Y, Z)
\]

\[
>> \text{shading interp}
\]

With this shading option, the colour of each facet is varied linearly across the facet so that the shading at the edges of adjacent facets match (called Gouraud shading). No mesh is drawn.
Nice looking surfaces

- For a really nice looking surface use:
  
  ```
  >> surf1(X, Y, Z)
  >> shading interp
  ```

- The `surf1` function behaves just like `surf` but renders a surface according to the current lighting model (the default lighting model is fine for almost every purpose you might have).

- The shading across the surface is no longer a function of its "height".

- Instead the shading is a function of the relative angle between the incident light and the surface normal at each point.

Lambertian shading

- *Lambertian shading* is a common shading model.

- With Lambertian shading, the brightness of each point in the image is proportional to the cosine of the angle between the surface normal and the incident light at each point.

- When the light is shining directly on a surface one obtains full brightness \( \cos(0) = 1 \) which reduces to zero where the incident light is tangential to a surface \( \cos(\pi/2) = 0 \).
The process and the power of shading

- A communication between the **renderer** and the **shader**.

Why shading is useful?

- The human visual system is very good at interpreting a surface that has Lambertian shading.
- We can readily deduce the shape of an object from its shading pattern (this is an active research called “shape from shading”).
- We can also readily determine the viewing angle without, say, the aid of an enclosing box around the plot. This too helps in interpretation.
- Matt paint and sand dunes are approximations of Lambertian surfaces.
- The Moon is an interesting example of a non-Lambertian surface - no one knows why!