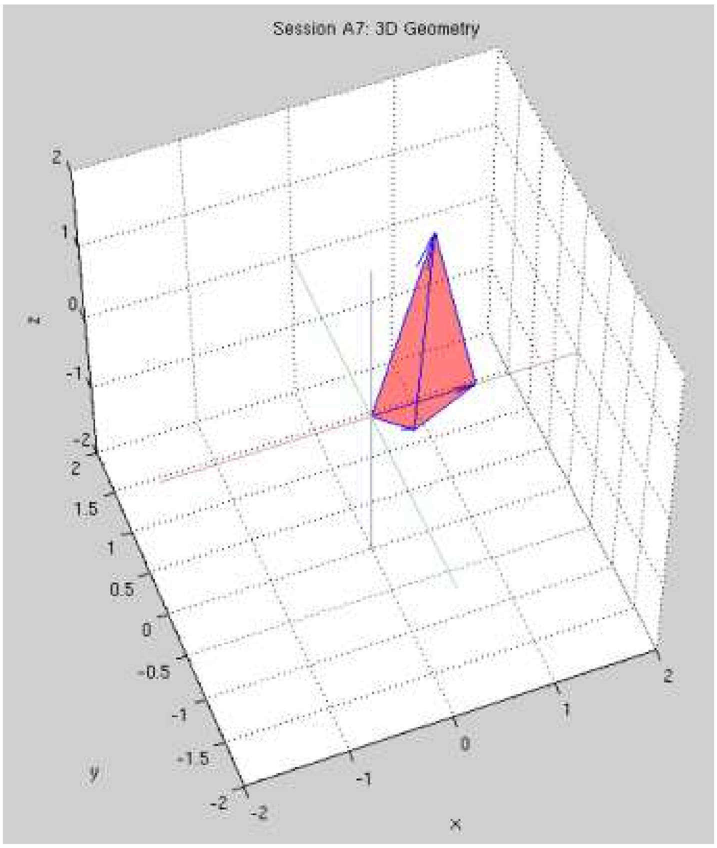
3D GEOMETRY

COMPUTER SESSION A7



Background

Geometry has a close relationship with linear algebra. Most linear algebra concepts can be visualized geometrically, and most geometric problems can be solved using linear algebra.

Computer graphics is the rendering of geometry on a computer. 3D geometry can be difficult to visualize in the mind or on paper, and computers are therefore an aid. For complex geometries, computers are a necessity.

PREPARATIONS

The session is based on some simple graphics functions which are provided. Start Matlab. Download the m-files in *Programs and templates* to your Matlab work directory.

PROBLEMS

Problem 1 - Testing the graphics functions.

(1) Initializing the graphics system

Execute the function `initgraphics;` to setup a drawing window and configure the necessary graphics parameters. The viewing volume is a cube with one corner in the point $(-2,-2,-2)$ and the other in $(2,2,2)$.

Press the mouse on the 3D coordinate system and drag around to learn how to manipulate the camera. You can change the camera mode by pressing buttons on the toolbar above. Try pressing for example "zoom camera". "orbit camera" is the default, and perhaps most useful.

(2) Drawing axes

Execute the function `drawaxes;` to draw some lines representing the x, y and z axes. Open the file `drawaxes.m` and observe that it calls the function `drawline()` three times with different arguments. Type `help drawline` to learn how to use the function.

The function `cleargraphics()` clears the viewing volume. Try drawing the axes, then clearing.

(3) Drawing lines (line segments)

Now try to draw some lines (line segments) yourself using the function `drawline()`. The function header looks like this:

```
function drawline(p1, p2, color)
```

It takes in two points, p_1 and p_2, representing the points between which the line is drawn, and a color. The color is represented by a 3-element vector holding the red, green and blue component respectively, going from 0 to 1. The function returns nothing. The color argument is optional.

Try to draw some lines in the viewing volume. Draw a line from $(-1,-1,-1)$ to $(1,2,0)$ for example. Move the camera around and try to verify that the line is where you think it should be. Execute `cleargraphics;` to clear all the lines.

(4) Drawing arrows

We want to visualize vectors in the end, and we want to draw vectors as arrows. The function `drawarrow(p1, p2)` draws an arrow from the point p_1 to the point p_2. Look at the function implemented in `drawarrow.m` if you want, it is implemented with a cross product and by drawing three lines.

(5) Drawing vectors

A vector v can be seen as an arrow from the point $(0, 0, 0)$ to the point v. The function `drawvector(v)` does exactly that. Try drawing some vectors.

(6) Drawing triangles

We also want to draw filled shapes, such as triangles, to be able to represent planes or look at solid objects.

The function `drawtriangle(c1, c2, c3, color)` takes three corners c_1, c_2 and c_3 and optionally a color as arguments. It then draws the corresponding triangle in the viewing volume. Note that the triangle is semi-transparent so geometry behind it shows through.

Try drawing some triangles using the function. For example, draw the triangle with the corners $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$.

Problem 2 - Visualizing linear algebra.

(1) Vector addition

Write a function `drawaddition(v1, v2)` which takes two vectors v_1 and v_2 and visualizes the addition process of $v1 + v2$.

Visualize the addition by drawing the vectors v_1, v_2 and $v_1 + v_2$ as well as the arrows going from v_1 to $v_1 + v_2$ and from v_2 to $v_1 + v_2$.

(2) Projection of a vector on a vector

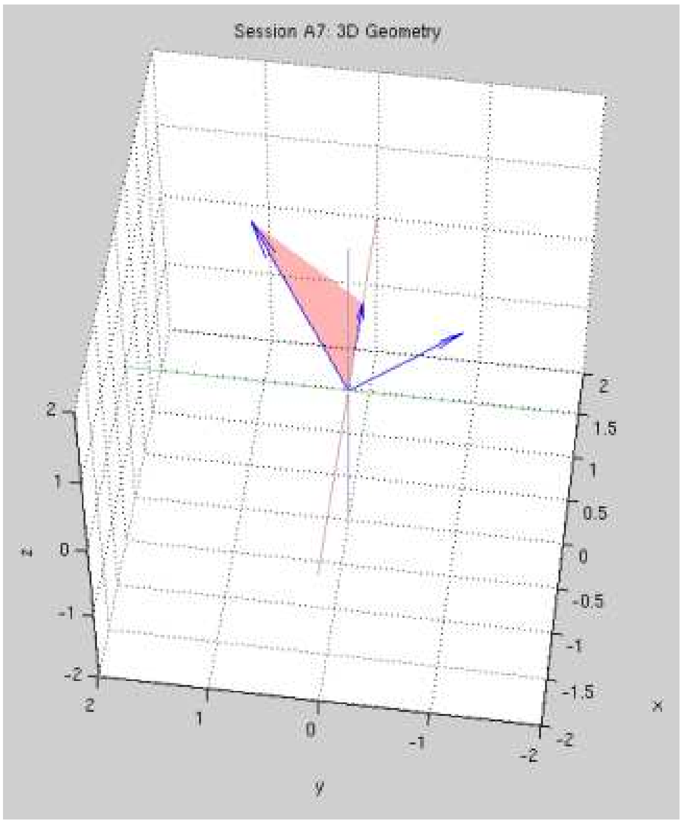
Write a function `projectionvv(v1, v2)` which computes the projection of the vector v_2 on the vector v_1 and returns it.

Write a function `drawprojectionvv(v1, v2)` which visualizes the projection of the vector v_2 on v_1 by using your function `projectionvv(v1, v2)`.

Test your functions by drawing the projection of $v_2 = (1, 1, 1)$ on $v_1 = (2, 0, 0)$. Then draw the projection of v_1 on v_2.

Verify by hand computation.

(3) Cross product (or vector product)



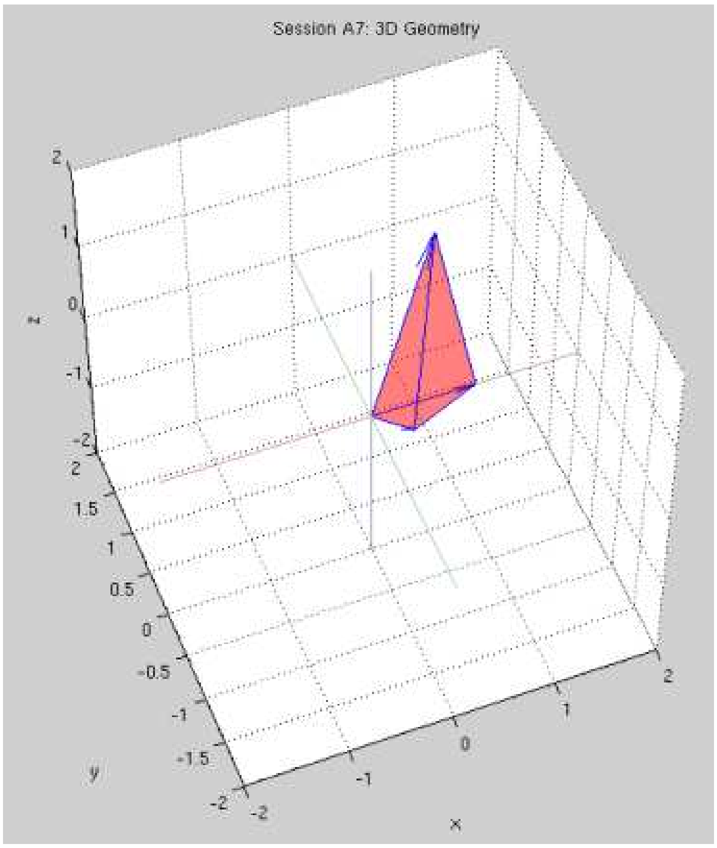
Write a function `drawcross(v1, v2)` which visualizes the cross product of the vectors v_1 and v_2.

Visualize the operation by drawing the vectors v_1, v_2 and $v_1 \times v_2$. Draw a triangle with the corners 0, v_1 and v_2 to show that $v_1 \times v_2$ is normal to the plane spanned by v_1 and v_2 (it is normal to both the vectors).

Test your function by visualizing $(1, 0, 0) \times (1, 1, 1)$.

Verify by hand computation.

(4) Volume of a tetrahedron



Write a function `drawvolume(v1, v2, v3)` which visualizes the tetrahedron defined by the vectors v_1, v_2, v_3. Perform this by drawing triangles covering the faces of the tetrahedron.

Try your function with the vectors $v_1 = (1, 0, 0)$, $v_2 = (1, 1, 1)$, $v_3 = v_1 \times v_3$.

Also write a function `tetvolume(v1, v2, v3)` which computes the volume of the tetrahedron. If V is the volume of the parallelepiped spanned by the three vectors v_1, v_2, v_3 (which the AMBS book describes how to compute), then the volume of the tetrahedron is $\frac{V}{6}$.

Verify your function by computing the volume of the tetrahedron defined by the vectors $v_1 = (0, 0, 1)$, $v_2 = (1, 0, 0)$, $v_3 = (0, 1, 0)$, whose volume should be $\frac{1}{6}$ (the corresponding parallelepiped is a cube with the side 1).

Try some other tetrahedrons and verify by hand computation.

Alternatively, write the above functions for a parallelepiped instead.

(5) Projection of a point on a line

The function `drawlinepointdirection(p, d)` draws a line through the viewing volume defined by the point p and direction d. Open the implementation `drawlinepointdirection.m` and try to understand what it does.

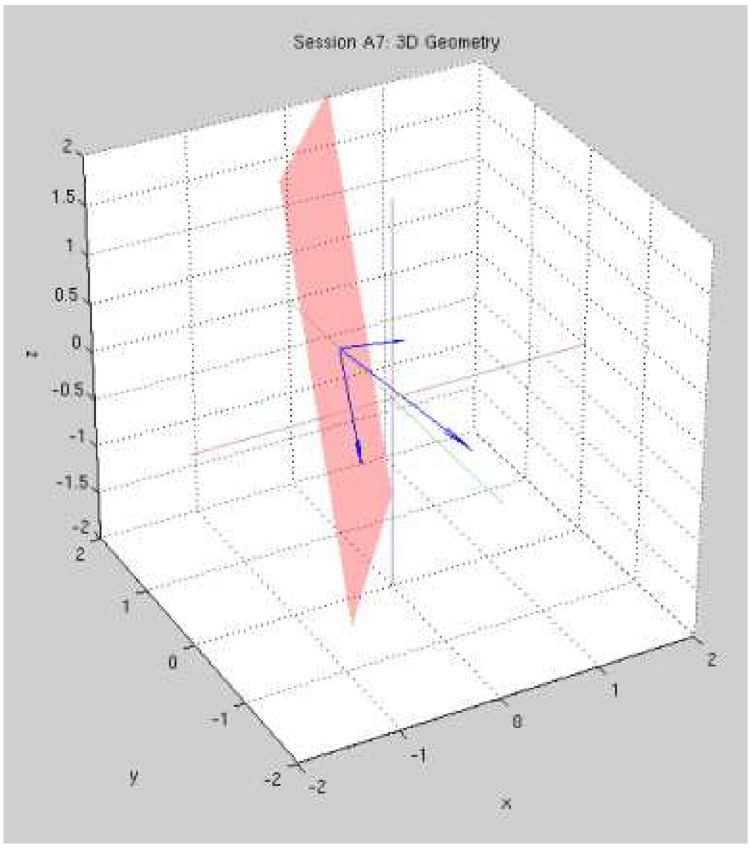
Write a function `projectionpl(p1, d, p2)` which computes the projection of the point p2 on the line defined by the point p1 and the direction d.

Write a function `drawprojectionpl(p1, d, p2)` which visualizes this projection by using the `drawlinepointdirection(p, d)` and the previous `drawarrow()` function.

Try your functions with the line represented by the point $p_1 = (0, 1, 0)$, the direction $d = 0.5, 0.3, 0.1$, and the point to project on the line $p_2 = (1, 1, 1)$.

Construct a more elementary example and verify with hand computation.

(6) Projection of a point on a plane



The function `drawplanepointnormal(p, n)` draws a plane through the viewing volume defined by the point p and the normal

n. Open the implementation `drawplanepointnormal.m` and try to understand what it does.

Write a function `projectionpplane(p1, n, p2)` which computes the projection of the point p2 on the plane defined by the point p1 and the normal n.

Write a function `drawprojectionpplane(p1, d, p2)` which visualizes this projection by using the `drawplanepointnormal(p, n)` and the previous `drawarrow()` function.

Try your functions with the plane represented by the point $p_1 = (0, 1, 0)$, the normal $d = 0.5, -0.3, 0.1$, and the point to project on the line $p_2 = (1, 0.3, -1)$.

Construct a more elementary example and verify with hand computation.

(7) Distance from a point to a line/plane

Write a function `distancepline(p1, d, p2)` which computes the distance between the point p2 and the line defined by the point p1 and the direction n. Hint: use the orthogonal projection.

Write a function `drawdistancepline(p1, d, p2)` which visualizes this projection using the previous functions.

Alternatively, write the corresponding functions for a point and a plane.

SOLUTIONS

Make sure that you really try to solve each problem before looking at the solutions. Have you really tried to solve the problem or should you try again before looking at the solution?

The solutions are available on the web page of this session under *Solutions to problems*.

ABOUT

This Computer Session is part of the Body and Soul educational program. More information can be found at

`http://www.phi.chalmers.se/bodysoul/`

This Computer Session is maintained by Johan Jansson (johanjan@math.chalmers.se).