CSC 7700: Scientific Computing

Module A: Basic Skills

Lecture 3: Vector Algebra and Basic Visualization Programming

Dr Frank Löffler

September 23 2010

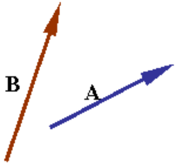
Vector Algebra

Vectors

Scalar: Quantity with magnitude
Vector: Mathematical Object with magnitude and direction

Characterizing properties:

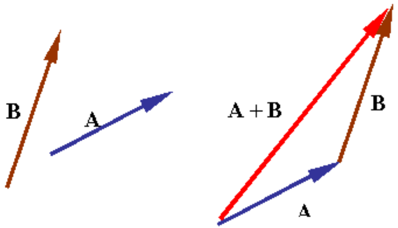
- Length
- Direction



Typical representation: arrow
Possible notations: $\vec{A}$, $\mathbf{A}$, $\underline{A}$
Magnitude is its length: $|A|$ or A

Vector addition

Addition: laying vectors head to tail in sequence



Rules for vector addition and multiplication with scalar:

$$\begin{aligned} \mathbf{A}+\mathbf{B} &= \mathbf{B}+\mathbf{A} \\ a\mathbf{A} &= \mathbf{A}a \\ a\left(\mathbf{A}+\mathbf{B}\right) &= a\mathbf{A}+a\mathbf{B} \end{aligned}$$

Unit vectors

Unit vector: Vector of unit length

Notation: $\hat{e}$, $\hat{\mathbf{e}}$, $\mathbf{e}$

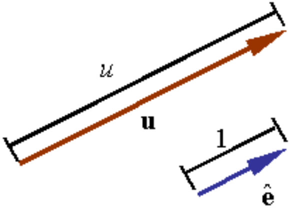
Property by definition: $|\hat{\mathbf{e}}| \equiv 1$

Almost all vectors can be made into unit vector:

$$\hat{\mathbf{e}} = \frac{\mathbf{u}}{|\mathbf{u}|}$$

Any vector can be fully represented by providing length and unit vector along its direction:

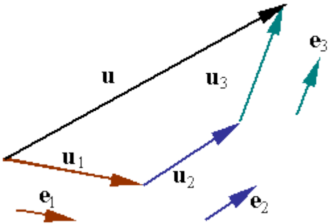
$$\mathbf{u} = u\hat{\mathbf{e}}$$



Base vectors

- Set of vectors
- Base to represent all other vectors
- Possible to construct all vectors from addition of vectors along base directions (and multiplication with scalars)

$$
\begin{aligned}
\mathbf{u} &= \mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 \\
&= u_1\mathbf{e}_1 + u_2\mathbf{e}_2 + u_3\mathbf{e}_3
\end{aligned}
$$



Vector components

$$\mathbf{u} = u_1\mathbf{e}_1 + u_2\mathbf{e}_2 + u_3\mathbf{e}_3$$

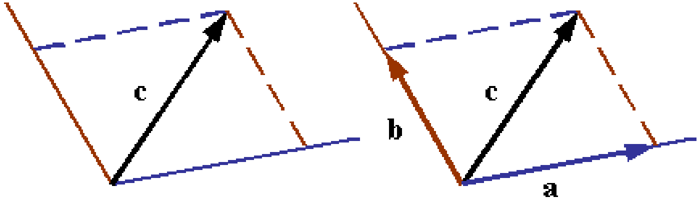
(u_1, u_2, u_3): components of $\mathbf{u}$ in base $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$

- If base vectors are unit vectors: components represent lengths of the three vectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$
- If base vectors are mutually orthogonal: base is known as orthonormal, Euclidean or Cartesian base

Vector components

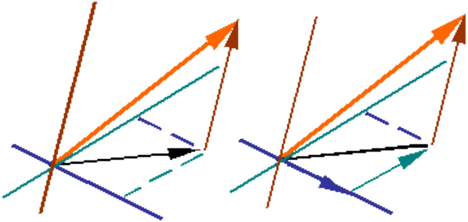
- Vectors in 2D can be resolved along any two (different) directions in a plane containing it.

Parallelogram rule to construct vectors **a** and **b** that add up to **c**:



Vector components

- Vectors in 3D can be resolved along any three non-coplanar lines
- First find vector in plane of two base directions
- Resolve this vector along two directions in plane



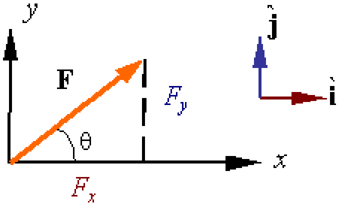
Vector components

Addition of vectors, represented by base vectors and components:

$$
\begin{aligned}
\mathbf{A} &= A_1\mathbf{e}_1 + A_2\mathbf{e}_2 + A_3\mathbf{e}_3 \\
\mathbf{B} &= B_1\mathbf{e}_1 + B_2\mathbf{e}_2 + B_3\mathbf{e}_3 \\
\mathbf{A} + \mathbf{B} &= (A_1 + B_1)\mathbf{e}_1 + (A_2 + B_2)\mathbf{e}_2 + (A_3 + B_3)\mathbf{e}_3
\end{aligned}
$$

Rectangular coordinates in 2D

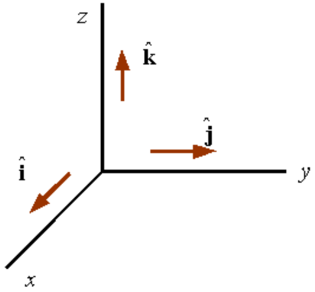
Base vectors of rectangular $x - y$ coordinate system given by unit vectors $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ along the x and y directions respectively:



$$
\begin{aligned}
\mathbf{F} &= F_x\hat{\mathbf{i}} + F_y\hat{\mathbf{j}} \\
F &= \sqrt{F_x^2 + F_y^2} \\
F_x &= F\cos(\theta) \\
F_y &= F\sin(\theta) \\
\tan(\theta) &= \frac{F_y}{F_x}
\end{aligned}
$$

Rectangular coordinates in 3D

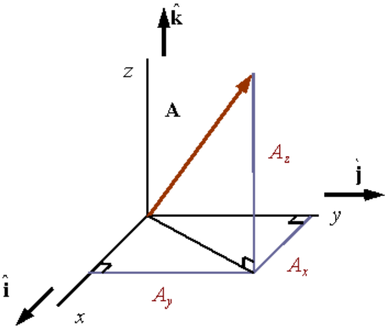
Base vectors of rectangular coordinate system given by set of three mutually orthogonal unit vectors $\hat{\mathbf{i}}$ $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ along x, y and z coordinate directions, respectively:



Shown: right-handed system

Rectangular coordinates in 3D

Vector components are projections of vector along x, y and z directions:

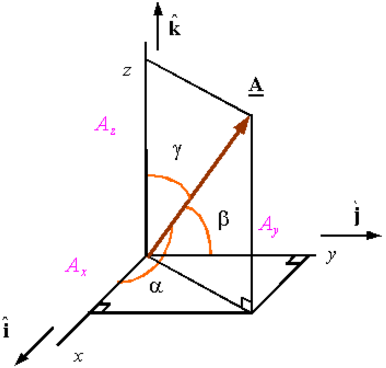


$$
\begin{aligned}
\mathbf{A} &= A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}} \\
A &= \sqrt{A_x^2 + A_y^2 + A_z^2}
\end{aligned}
$$

Rectangular coordinates in 3D

Direction cosines:

$$
\begin{aligned}
l &= \cos(\alpha) \\
m &= \cos(\beta) \\
n &= \cos(\gamma)
\end{aligned}
$$



$$l = \cos(\alpha) = \frac{A_x}{A}, \; m = \cos(\beta) = \frac{A_y}{A}, \; n = \cos(\gamma) = \frac{A_z}{A}$$

Rectangular coordinates in 3D

$$l = \cos(\alpha) = \frac{A_x}{A},\ m = \cos(\beta) = \frac{A_y}{A},\ n = \cos(\gamma) = \frac{A_z}{A}$$

Direction cosines not independent:

$$l^2 + m^2 + n^2 = \mathbf{1}$$

Proof:

$$l^2 + m^2 + n^2 = \cos^2(\alpha) + \cos^2(\beta) + \cos^2(\gamma) = \frac{A_x^2}{A^2} + \frac{A_y^2}{A^2} + \frac{A_z^2}{A^2} = \mathbf{1}$$

Rectangular coordinates in 3D

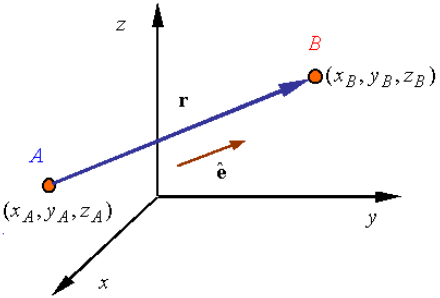
Construction of unit vector along vector **A**:

$$\begin{aligned}\hat{\mathbf{e}} &= \frac{\mathbf{A}}{A} = \frac{A_x}{A}\hat{\mathbf{i}} + \frac{A_y}{A}\hat{\mathbf{j}} + \frac{A_z}{A}\hat{\mathbf{k}} \\ &= \cos(\alpha)\hat{\mathbf{i}} + \cos(\beta)\hat{\mathbf{j}} + \cos(\gamma)\hat{\mathbf{k}} \\ &= l\hat{\mathbf{i}} + m\hat{\mathbf{j}} + k\hat{\mathbf{k}}\end{aligned}$$

Therefore:

$$\mathbf{A} = A\hat{\mathbf{e}} = A\cos(\alpha)\hat{\mathbf{i}} + A\cos(\beta)\hat{\mathbf{j}} + A\cos(\gamma)\hat{\mathbf{k}}$$

A vector connecting two points



Vector connecting point A to point B is given by

$$\begin{aligned}
\mathbf{r} &= (x_B - x_A)\hat{\mathbf{i}} + (y_B - y_A)\hat{\mathbf{j}} + (z_B - z_A)\hat{\mathbf{k}} \\
&= x_B\hat{\mathbf{i}} - x_A\hat{\mathbf{i}} + y_B\hat{\mathbf{j}} - y_A\hat{\mathbf{j}} + z_B\hat{\mathbf{k}} - z_A\hat{\mathbf{k}} \\
&= (x_B\hat{\mathbf{i}} + y_B\hat{\mathbf{j}} + z_B\hat{\mathbf{k}}) - (x_A\hat{\mathbf{i}} + y_A\hat{\mathbf{j}} + z_A\hat{\mathbf{k}}) \\
&= \mathbf{B} - \mathbf{A}
\end{aligned}$$

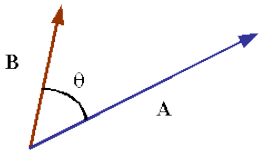
Vector products

There isn't a single product of vectors:

Product	Notation	Result
dot, scalar, inner	$\mathbf{A} \cdot \mathbf{B}$	scalar
cross, vector	$\mathbf{A} \times \mathbf{B}$	vector
tensor, outer	$\mathbf{A} \otimes \mathbf{B}$	matrix

Dot product

$$
\begin{aligned}
\mathbf{A}\cdot\mathbf{B} &= AB\cos(\theta) \\
\theta = 90^\circ \rightarrow \mathbf{A}\cdot\mathbf{B} &= 0 \\
\theta = 0^\circ \rightarrow \mathbf{A}\cdot\mathbf{A} &= \mathbf{A}^2 = A^2
\end{aligned}
$$



Some properties:

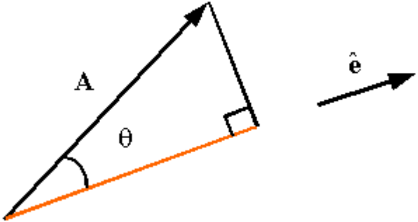
$$
\begin{aligned}
\mathbf{A}\cdot\mathbf{B} &= \mathbf{B}\cdot\mathbf{A} \\
\alpha(\mathbf{B}\cdot\mathbf{C}) &= (\alpha\mathbf{B})\cdot\mathbf{C} = \mathbf{B}\cdot(\alpha\mathbf{C}) \\
\mathbf{A}\cdot(\mathbf{B}+\mathbf{C}) &= \mathbf{A}\cdot\mathbf{B}+\mathbf{A}\cdot\mathbf{C}
\end{aligned}
$$

Dot product

Special cases:
Scalar product of two unit vectors: angle between vectors

$$\hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_j = \cos(\theta)$$

Scalar product of unit vector with arbitrary vector:



Scalar projection: $\mathbf{A} \cdot \hat{\mathbf{e}}$
Vector projection: $(\mathbf{A} \cdot \hat{\mathbf{e}})\hat{\mathbf{e}}$

Dot product

Rectangular coordinates:

$$\begin{aligned}
\mathbf{A} &= A_x\hat{\mathbf{i}} + A_y\hat{\mathbf{j}} + A_z\hat{\mathbf{k}} \\
\mathbf{B} &= B_x\hat{\mathbf{i}} + B_y\hat{\mathbf{j}} + B_z\hat{\mathbf{k}} \\
&\downarrow \\
\mathbf{A}\cdot\mathbf{B} &= A_xB_x + A_yB_y + A_zB_z
\end{aligned}$$

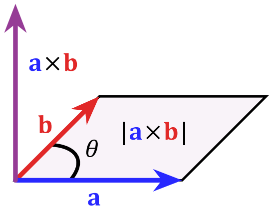
because

$$\begin{aligned}
\hat{\mathbf{i}}\cdot\hat{\mathbf{j}} &= 0 \\
\hat{\mathbf{i}}\cdot\hat{\mathbf{k}} &= 0 \\
\hat{\mathbf{j}}\cdot\hat{\mathbf{k}} &= 0
\end{aligned}$$

Cross product

Result of vector product of two vectors **a** and **b**: vector

- perpendicular to both **a** and **b**
- magnitude: area of paralellogram generated from **a** and **b**



$$\mathbf{a} \times \mathbf{b} = ab\sin(\theta)\mathbf{n}$$

Properties:

$$\begin{aligned}
\mathbf{a} \times \mathbf{a} &= 0 \\
\mathbf{a} \times \mathbf{b} &= -\mathbf{b} \times \mathbf{a} \\
\alpha(\mathbf{b} \times \mathbf{c}) &= (\alpha\mathbf{b}) \times \mathbf{c} = \mathbf{b} \times (\alpha\mathbf{c}) \\
(\mathbf{a} + \mathbf{b}) \times \mathbf{c} &= \mathbf{a} \times \mathbf{c} + \mathbf{b} \times \mathbf{c}
\end{aligned}$$

Cross product

Rectangular coordinates:

$$
\begin{aligned}
\mathbf{a} &= a_x \hat{\mathbf{i}} + a_y \hat{\mathbf{j}} + a_z \hat{\mathbf{k}} \\
\mathbf{b} &= b_x \hat{\mathbf{i}} + b_y \hat{\mathbf{j}} + b_z \hat{\mathbf{k}} \\
&\downarrow \\
\mathbf{a} \times \mathbf{b} &= (a_y b_z - a_z b_y)\hat{\mathbf{i}} - (a_x b_z - a_z b_x)\hat{\mathbf{j}} + (a_x b_y - a_y b_x)\hat{\mathbf{k}}
\end{aligned}
$$

Relations between base vectors:

$$
\begin{aligned}
\hat{\mathbf{i}} \times \hat{\mathbf{j}} &= \hat{\mathbf{k}} \\
\hat{\mathbf{k}} \times \hat{\mathbf{i}} &= \hat{\mathbf{j}} \\
\hat{\mathbf{j}} \times \hat{\mathbf{k}} &= \hat{\mathbf{i}}
\end{aligned}
$$

Basic Visualization Programming

3D Visualization options

Two main APIs, providing nearly th same level of functionality:

- Direct3D
  - Proprietary, Windows OS / Xbox only
  - Updates often coupled with OS updates
  - Targeted towards game development
- OpenGL
  - Open standard, available on most moderns OSs
  - General purpose 3D API

Large projects often wrap low-level 3D-API anyway.

OpenGL

- API for 2D and 3D computer graphics
- over 250 different function calls
- Originally developed by SGI
- Widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation, video games
- Now managed by the non-profit technology consortium Khronos Group

GLUT

The OpenGL Utility Toolkit (GLUT)

- Library of utilities for OpenGL programs
- Primarily performs system-level I/O with the host operating system, e.g. window definition, window control, and monitoring of keyboard and mouse input
- Routines for drawing a number of geometric primitives

Stated two aims:

- Allow the creation of rather portable code between operating systems
- Make learning OpenGL easier

Upstream development of original GLUT in stagnation, but alternatives exist, e.g.:

OpenGL / GLUT simple example

```
#include <GL/glut.h>

int main(int argc, char **argv) {
  // Initialize GLUT library
  glutInit(&argc, argv);
  // Set display modes
  glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
  // Define initial window position and size
  glutInitWindowPosition(100,100);
  glutInitWindowSize(320,320);
  // Set some title
  glutCreateWindow("CSC 7700 example");

  // Tell Glut which function does the initial and animation rendering
  glutDisplayFunc(renderScene);
  glutIdleFunc(renderScene);
  // Enable depth testing
  glEnable(GL_DEPTH_TEST);
  // Start Glut main loop. This will call our callbacks
  glutMainLoop();
  return 0;
}
```

OpenGL / GLUT simple example

```
// Angle of model rotation
float angle = 0.0;

// This function is called every time the scene is rendered
void renderScene(void) {
  // Clear old buffers
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // Save the previous settings, in this case save
  // the camera settings.
  glPushMatrix();

  // Perform a rotation around the y axis (0,1,0)
  // by the amount of degrees defined in the variable angle
  glRotatef(angle, 0.0, 1.0, 0.0);

  // Draw triangle
  glColor3f(0.9, 0.9, 0.9);
  glBegin(GL_TRIANGLES);
    glVertex3f(-0.5,-0.5, 0.0);
    glVertex3f( 0.5, 0.0, 0.0);
    glVertex3f( 0.0, 0.5, 0.0);
  glEnd();

  // Forget about the current transformation matrix
  glPopMatrix();

  // Swapping the buffers causes the rendering above to be
  // shown
  glutSwapBuffers();

  // Finally increase the angle for the next frame
  angle+=0.1;
}
```

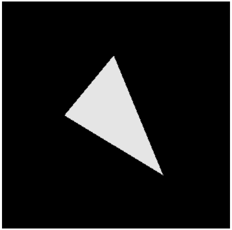
OpenGL / GLUT simple example

```
#include <GL/glut.h>

float angle = 0.0;

void renderScene(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glRotatef(angle, 0.0, 1.0, 0.0);
  glColor3f(0.9, 0.9, 0.9);
  glBegin(GL_TRIANGLES);
    glVertex3f(-0.5,-0.5, 0.0);
    glVertex3f( 0.5, 0.0, 0.0);
    glVertex3f( 0.0, 0.5, 0.0);
  glEnd();
  glPopMatrix();
  glutSwapBuffers();
  angle+=0.1;
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(320,320);
  glutCreateWindow("CSC 7700 example");
  glutDisplayFunc(renderScene);
  glutIdleFunc(renderScene);
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
  return 0;
}
```



OpenGL / GLUT simple example

However, this is essentially using OpenGL 2

Current Version: 4.1

- Only small differences between version 3 and 4
- Bigger changes between 2 and 3:
  - Move away from procedual interface, more object-oriented
  - OpenGL isn't handling transformation/rotation matrices anymore, using shaders instead
  - In some sense: more low-level
  - However: better performance on modern hardware

Why is example only using OpenGL 2?

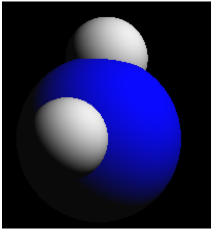
- Large code-base still using this version
- Shallower learning curve (IMHO)

Optional Coursework

Extend simple OpenGL/GLUT example

(https://svn.cct.lsu.edu/repos/sci-comp/public/Module-A/A3_template.cpp)

- Replace triangle by water molecule:
  - Three colored spheres
  - Correct (explicitly specified) angle between H-molecules
  - Distances and radii not important, as long as it 'looks ok'
- Send source code as attachement to instructors mailing list



Due: Sep 30th