Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

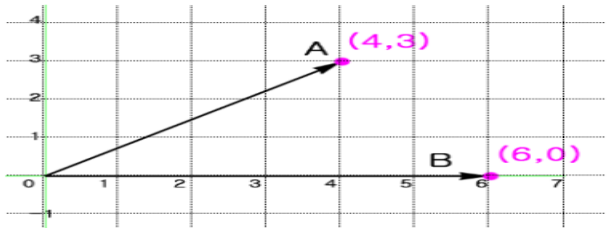
hslee00001@naver.com

목차

1. 벡터의 내적

– Scalar product(스칼라곱) or Dot product 기호로 Dot을 씀.

구하는 방법은 두가지가 있는데, 첫번째가 좌표값의 각 성분을 곱해서 더하는 것이다.

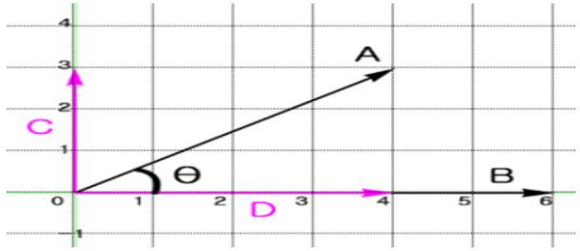


A·B (A와 B의 내적)

$= (A_x \times B_x) + (A_y \times B_y)$

$= (4 \times 6) + (3 \times 0) = 24$

두번째는 벡터의 크기를 곱하는 것이다.



A·B (A와 B의 내적)

= B 크기 × D의 크기

$= |B| \times |A| \times \cos\theta$

$= 6 \times 4 = 24$

여기서 A의 크기가 아니라 A를 분해한 벡터 D의 크기를 곱해주는 이유는 B벡터에 실제적으로 영향을 주는 벡터가 D이기 때문이다.

(벡터 C는 벡터 B의 방향으로 어떠한 영향도 주지 못하기 때문에, 내적에서 무시한다.)

프로그래밍할 때, 내적은 보통 단위벡터와 함께 사용된다.

➔ 두 단위벡터의 내적값으로 벡터의 사이각을 알 수 있다.

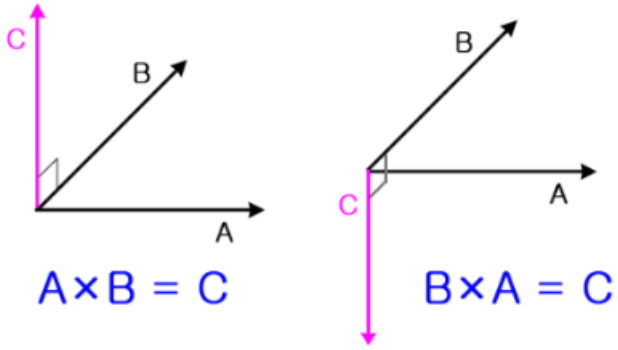
두단위벡터의 내적값은 $\cos\theta$ 가 된다. 그러면 이 값으로 많은 정보를 얻을 수 있다.

3D프로그래밍에서는 이 내적값을 사용하여 빛이나 컬링, 충돌 등 많은 부분을 계산한다. 내적 계산은 연산에 큰 비용이 들지 않으므로, 최적화에도 많은 도움을 준다.

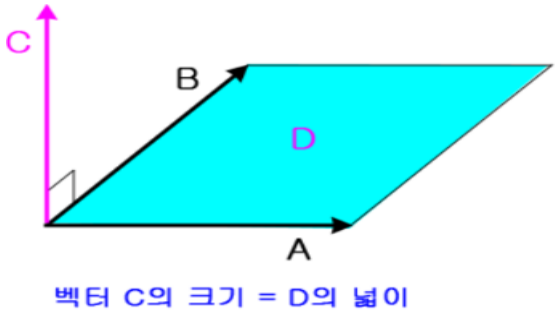
2. 벡터의 외적

- 외적은 Cross Product라고 한다.

내적은 2차원이나 3차원에서 사용될 수 있으나, 외적은 3차원에서만 의미를 가진다. 내적의 결과는 스칼라 반환이지만, 외적은 두벡터와 수직이 되는 벡터를 반환하기 때문이다.



: 내적은 교환법칙이 성립하지만, 외적은 그렇지 않다. 오른손으로 감싸쥐면 엄지의 방향임.



벡터 C의 크기 = D의 넓이

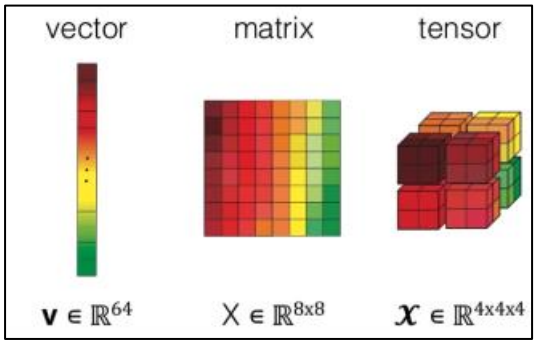
: 또한 외적의 결과값으로 나온 벡터 C의 크기는 A와 B를 연결한 평행사변형의 넓이이다.

A와 B(단위벡터일 때)의 외적 결과값인 벡터 C의 크기로 A와 B의 사이각을 판단 할 수 있다. 벡터 C의 크기가 1이라면 A와 B를 연결해서 만든 도형이 가로세로가 1인 정사각형임을 알 수 있고, A와 B가 수직임을 알 수 있다.

외적은 두 벡터와 수직이 되는 벡터를 구할때, 두벡터를 포함하는 평면을 바라보는 방향을 찾는다할 때 많이 쓰인다.

3. Tensor – multidimensional array

임의의 차원을 갖는 배열들을 뜻함.



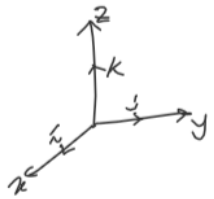
1차원 2차원(행렬) 3차원(다차원배열)

이후 텐서플로우를 사용하면 차원 수가 아주 높은 텐서를 조작할 수 있다.

4. 벡터의 정규화

기저벡터

단위벡터: 크기가 1인벡터

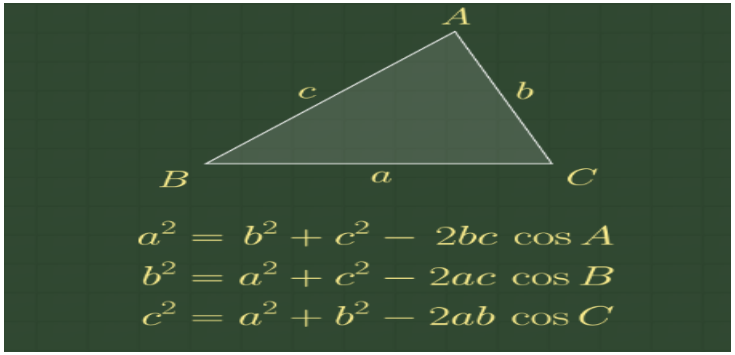


x,y,z축에 대한 단위벡터.

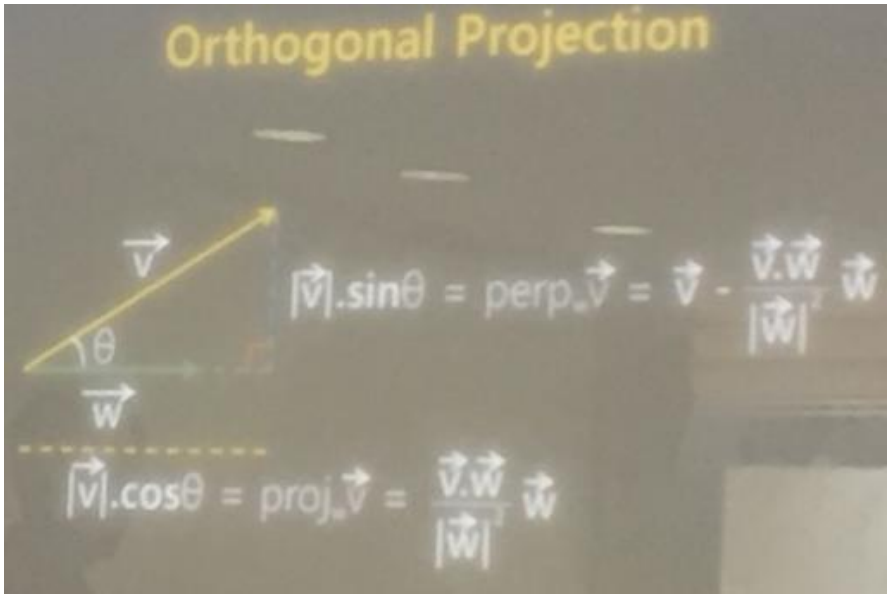
기저벡터는 벡터공간에서 각각의 축방향을 가리키는 단위 벡터들이다.

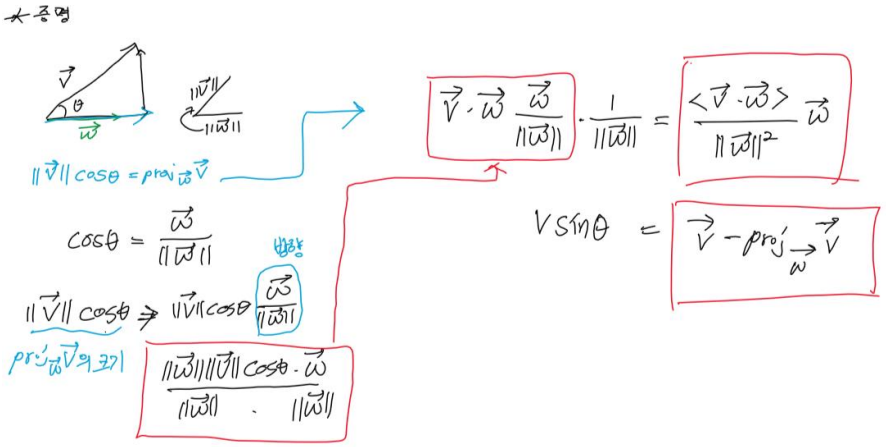
↳ 방향성분만

5. 코사인 제2법칙

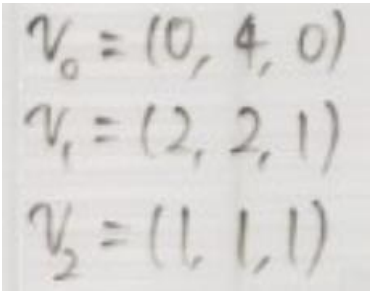


6. 그램슈미트 정규 직교화





예제)



➔ 정규직규화 하라.

Code

main

```c
#include "vector_3d.h"
#include <stdio.h>

int main(void)
{
        vec3 A = {3, 2, 1};
        vec3 B = {1, 1, 1};
        vec3 X = {1, 0, 0};
        vec3 Y = {0, 1, 0};
        vec3 v[3] = {{0, 4, 0}, {2, 2, 1}, {1, 1, 1}};
        vec3 w[3] = {};
        vec3 R = {0, 0, 0,
                        vec3_add, vec3_sub, vec3_scale,
                        vec3_dot, vec3_cross, print_vec3,
                        gramschmidt_normalization};

        printf("A add B =\n");
        R.add(A, B, &R);
        R.print(R);

        printf("A sub B =\n");
        R.sub(A, B, &R);
        R.print(R);

        printf("3 scale\n");
        R.scale(3, R, &R);
        R.print(R);

        printf("A dot B = %f\n", R.dot(A, B));
        printf("A cross B=\n");
        R.cross(X, Y, &R);
        R.print(R);

        printf("gramschmidt\n");
        R.gramschmidt(v, w, R);

        return 0;
}
```

header

```c
#ifndef __VECTOR_3D_H__
#define __VECTOR_3D_H__

#include <stdio.h>
#include <math.h>

typedef struct vector3d vec3;

struct vector3d
{
        float x;
        float y;
        float z;

        void (* add)(vec3, vec3, vec3 *);
        void (* sub)(vec3, vec3, vec3 *);
        void (* scale)(float, vec3, vec3 *);
        float (* dot)(vec3, vec3);
        void (* cross)(vec3, vec3, vec3 *);
        void (* print)(vec3);

        void (* gramschmidt)(vec3 *, vec3 *, vec3);
};

void vec3_add(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.x + b.x;
        r->y = a.y + b.y;
        r->z = a.z + b.z;
}

void vec3_sub(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.x - b.x;
        r->y = a.y - b.y;
        r->z = a.z - b.z;
}

void vec3_scale(float factor, vec3 a, vec3 *r)
{
        r->x = a.x * factor;
        r->y = a.y * factor;
        r->z = a.z * factor;
}

float vec3_dot(vec3 a, vec3 b)
{
        return a.x * b.x + a.y * b.y + a.z * b.z;
}
```

```c
void vec3_cross(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.y * b.z - a.z * b.y;
        r->y = a.z * b.x - a.x * b.z;
        r->z = a.x * b.y - a.y * b.x;
}

void print_vec3(vec3 r)
{
        printf("x = %f, y = %f, z = %f\n", r.x, r.y, r.z);
}

float magnitude(vec3 v)
{
        return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

void gramschmidt_normalization(vec3 *arr, vec3 *res, vec3 r)
{
        vec3 scale1 = {};
        float dot1, mag1;

        mag1 = magnitude(arr[0]);
        r.scale(1.0 / mag1, arr[0], &res[0]);
        r.print(res[0]);

        mag1 = magnitude(res[0]);
        dot1 = r.dot(arr[1], res[0]);
        r.scale(dot1 * (1.0 / mag1), res[0], &scale1);
        r.sub(arr[1], scale1, &res[1]);
        r.print(res[1]);
}

endif
```