

# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-05-16 (55 회차)

강사 - Innova Lee(이상훈)  
[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)  
학생 - 정유경  
[ucong@naver.com](mailto:ucong@naver.com)

## 1. vector\_3D.c (Gramschmidt 정규직교화 C 로 구현하기 )

```
#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 RES = {0, 0, 0};
    vec3 R = {0, 0, 0,
               vec3_add, vec3_sub, vec3_scale,
               vec3_dot, vec3_cross, print_vec3,
               gramschmidt_normalization};

    R.add(A, B, &R);
    printf("1. A + B = R\n");
    R.print(R);
    R.sub(A, B, &R);

    printf("\n2. A - B = R\n");
    R.print(R);

    R.scale(3, R, &R);
    printf("\n3. 3 x (A - B) \n");
    R.print(R);

    printf("\n4. A dot B = %3.1f\n", R.dot(A, B));

    R.cross(X, Y, &R);
    printf("\n5. A cross B\n");
    R.print(R); //왜 세번출력되는거지?

    printf("\n6. GramSchmidt of v1, v2, v3\n");
    R.gramschmidt(v, w, R);
    R.print(w[0]);
    R.print(w[1]);
    R.print(w[2]);

    /*
    */
    return 0;
}
```

## 2. vector\_3D.h

```
#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;
}

    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;
}

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("(%4.1f, %4.1f, %4.1f)\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)
{
    /*
    v[3] -> arr[3]
    w[3] -> res[3]
    struct vec3 R -> r
    */

    vec3 scale1 = {};
    vec3 scale2 = {};
    float dot1, dot2, mag1, mag2;
```

```

/* 벡터 w0 구하기 */
    mag1 = magnitude(arr[0]); // |v0|
    r.scale(1.0 / mag1, arr[0], &res[0]); // v0/|v0| -> w0
    r.print(res[0]); // print w0
void vec3_scale(float factor, vec3 a, vec3 *r)
{
    r->x = a.x * factor;

/* 벡터 w1 구하기 */
    mag1 = magnitude(res[0]); // |w0|
    dot1 = r.dot(arr[1], res[0]); // <v1,w0>
    r.scale(dot1 * (1.0 / pow(mag1,2)), res[0], &scale1);
    // v1,w0 의 내적을 |w0|의 제곱으로 나눈 값 * w0 = scale1

    r.sub(arr[1], scale1, &res[1]); // v1-scale1 = w1
    r.print(res[1]); // print w1

/* 벡터 w2 구하기 */

//    mag1 = magnitude(res[0]); // |w0|
//    mag2 = magnitude(res[1]); // |w1|
//    dot1 = r.dot(arr[2], res[0]); // <v2,w0>
//    dot2 = r.dot(arr[2], res[1]); // <v2,w1>

//    r.scale(dot1 * (1.0 / pow(mag1,2)), res[0], &scale1);
//    // v2, w0 의 내적을 |w0|의 제곱으로 나눈 값 * w0 = scale 1
//    r.sub(arr[2], scale1, &res[2]);

//    r.scale(dot2 * (1.0 / pow(mag2,2)), res[1], &scale2);
//    // v2, w1 의 내적을 |w1|의 제곱으로 나눈 값 * w1 = scale 2
//    r.sub(res[2], scale2, &res[2]);

    r.print(res[2]); // print w2
}
#endif

```

# Gramschmidt 정규직교화 이론 정리

\* 그람-슈미트 정규화 과정을 통해 주를 바꿀 수 있다.

ex)  $\vec{w}_1 = (0, 4, 0)$   
 $\vec{w}_2 = (2, 2, 1)$   
 $\vec{w}_3 = (1, 1, 1)$

새로운 주를 잡는다.  $\vec{u}_1 = \frac{\vec{w}_1}{\|\vec{w}_1\|}$

$\vec{w}_2$ 에는  $\vec{u}_1$ 와 수직인 성분, 평행한 성분이 있는데 평행한 성분을 빼掉하면 수직인 성분만 남는다.

$\vec{w}_2 - \langle \vec{w}_2, \vec{u}_1 \rangle \vec{u}_1 = \vec{v}_2$  라고 하면

$\vec{u}_2 = \frac{\vec{v}_2}{\|\vec{v}_2\|}$  (크기 1로 정규화)

$\vec{w}_3 - \langle \vec{w}_3, \vec{u}_1 \rangle \vec{u}_1 - \langle \vec{w}_3, \vec{u}_2 \rangle \vec{u}_2$

마지막으로  $\vec{w}_3$ 에서  $\vec{u}_1$  방향의 성분과  $\vec{u}_2$  방향의 성분을 빼掉하면,  $\vec{w}_3$ 가 갖고있는 성분중에  $\vec{u}_1, \vec{u}_2$ 에 수직인 성분만 남는다.

$\vec{u}_3 = \frac{\vec{v}_3}{\|\vec{v}_3\|}$

ex)

ex)  $\vec{v}_1 = (2, 2, 1)$   
 $\vec{v}_2 = (-2, 1, 2)$   
 $\vec{v}_3 = (1, 0, 0)$

$\vec{u}_1 = \frac{\vec{v}_1}{\|\vec{v}_1\|} = \frac{1}{3}(2, 2, 1) = (\frac{2}{3}, \frac{2}{3}, \frac{1}{3})$

$\vec{v}_2' = \vec{v}_2 - \text{proj}_{\vec{u}_1}(\vec{v}_2)$   
 $= \vec{v}_2 - \langle \vec{v}_2, \vec{u}_1 \rangle \vec{u}_1$   
 $= (-2, 1, 2) - (-\frac{4}{3}, \frac{2}{3}, \frac{2}{3}) = (-2, 1, 2)$   
 $\therefore \vec{u}_2 = \frac{\vec{v}_2'}{\|\vec{v}_2'\|} = \frac{1}{3}(-2, 1, 2) = (-\frac{2}{3}, \frac{1}{3}, \frac{2}{3})$

$\vec{v}_3' = \vec{v}_3 - \text{proj}_{\vec{u}_1}(\vec{v}_3) - \text{proj}_{\vec{u}_2}(\vec{v}_3)$   
 $= \vec{v}_3 - \langle \vec{v}_3, \vec{u}_1 \rangle \vec{u}_1 - \langle \vec{v}_3, \vec{u}_2 \rangle \vec{u}_2$   
 $= (1, 0, 0) - 12(\frac{2}{3}, \frac{2}{3}, \frac{1}{3}) - (-12)(-\frac{2}{3}, \frac{1}{3}, \frac{2}{3})$   
 $= (1, 0, 0) - (8, 8, 4) + (-8, 4, 8)$   
 $= (-5, -4, 4)$   
 $\therefore \vec{u}_3 = \frac{\vec{v}_3'}{\|\vec{v}_3'\|} = \frac{1}{6}(-5, -4, 4) = (-\frac{5}{6}, -\frac{2}{3}, \frac{2}{3})$