Matrix’in İlk Üç Template Parametresi

1. Matrix’in İlk Üç Template Parametresi

Matrix sınıfı 6 template parametresine sahiptir fakat şimdilik ilk 3 parametreyi öğrenmek yeterli olacaktır.

Matrix sınıfının 3 zaruri template parametesi şunlardır:

```
Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>
```

- Scalar sayısal tiptir yani katsayıların tipidir. Yani float matrisi isteniyorsa burada float seçilir.
- RowsAtCompileTime ve ColsAtCompileTime matrisin satır ve sütun sayılarını belirler.

Eigen içerisinde olağan durumlar için kolaylık sağlayan typedef tanımlamaları bulunur. Örneğin, Matrix4f, 4x4 float matrisidir:

```
typedef Matrix<float, 4, 4> Matrix4f;
```

2. Vektörler

Vektörler 1 satır veya 1 sütuna sahip özel matrislerdir. Genellikle 1 sütun olarak oluşturulurlar ki bu vektörler sütun vektörü olarak adlandırılır, 1 satırlı vektörler ise satır vektörü olarak adlandırılır.

Örneğin, Eigen tarafından tanımlanmış Vector3f (sütun) vektörü:

```
typedef Matrix<float, 3, 1> Vector3f;
```

Satır vektörleri için de kolaylık sağlayacak typedef tanımlamaları bulunmaktadır:

```
typedef Matrix<int, 1, 2> RowVector2i;
```

2.1. Dynamic Özel Değeri

Eigen boyutları derlenme zamanında bilinen matrislerle sınırlı değildir. RowsAtCompileTime ve ColsAtCompileTime template parametreleri derlenme zamanında boyutunu belirsiz yapan Dynamic özel değerini alabilirler. Referans gösterilmiş boyutlar dinamik boyut, derlenme süresinde bilinen boyut sabit boyuttur. Örneğin, MatrixXd typedef'i dinamik boyuta sahip double matris anlamına gelir:

```
typedef Matrix<double, Dynamic, Dynamic> MatrixXd;
```

Benzer şekilde VectorXi şöyle tanımlanır:

```
typedef Matrix<int, Dynamic, 1> VectorXi;
```

2.2. Constructor

Varsayılan yapıcı fonksiyon her zaman mevcuttur, asla herhangi bir dinamik hafıza tutma veya matris katsayısı tanımlama işlemi yapmaz.

```
Matrix3f a;
MatrixXf b;
```

- a, elemanları atanmamış, float değerde 3x3 bir matristir.
- b, elemanları henüz atanmamış ve boyutu henüz 0x0 olan dinamik boyutlu bir matristir.

Boyut alan yapıcı fonksiyonlar da mevcuttur. Matrisler için her zaman satırlar önce girilir. Vektörler için ise sadece vektör boyutu girilir. Matris elemanları için yer ayrılır fakat kendi kendine elemanlara atama yapılmaz:

```
MatrixXf a(10, 15);
VectorXf b(30);
```

- a, yeri ayrılmış fakat henüz atanmamış 10x15 dinamik boyutlu matristir.
- b, yeri ayrılmış fakat henüz atanmamış 30 boyutunda dinamik boyutlu vektördür.

2.3. Eleman Erişimi

Eigen'de ilkel eleman erişimi parantez operatörünün overload edilmesiyle yapılır. Matrisler için her zaman satır indeksi ilk olarak gönderilir. Vektörde ise sadece sadece bir indeks gönderilir. Numaralama 0 ile başlar.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 MatrixXd m(2, 2);
 m(0, 0) = 3;
 m(1, 0) = 2.5;
 m(0, 1) = -1;
 m(1, 1) = m(1, 0) + m(0, 1);
 cout << "m:\n" << m << endl;
 VectorXd v(2);
 v(0) = 4;
 v(1) = v(0) - 1;
 cout << "v:\n" << v << endl;

    return 0;
}
```

2.4. Virgül ile Atama

Matris ve vektör elemanları virgül ile kolayca atanabilir.

```
Matrix3f m;
m << 1, 2, 3,
     4, 5, 6,
     7, 8, 9;
cout << m;
```

Çıktı:

```
1 2 3
```

```
4 5 6
7 8 9
```

2.5. Yeniden Boyutlandırma

Bir matrisin mevcut boyutu rows(), cols() ve size() fonksiyonları ile okunabilir. Bu fonksiyonlar sırasıyla satır sayısı, sütun sayısı ve eleman sayısını döndürür. Dinamik boyutlu bir matris resize() fonksiyonu ile yeniden boyutlandırılabilir.

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 MatrixXd m(2, 5);
 m.resize(4, 3);
 cout << "m matrisi boyutu: "
  << m.rows() << "x" << m.cols() << endl;
 cout << "Eleman sayisi: " << m.size() << endl;
 VectorXd v(2);
 v.resize(5);
 cout << "v vektoru boyutu : " << v.size() << endl;
 cout << "Matris olarak, v'nin boyutu: " << v.rows() << "x" << v.cols()
 << endl;

    return 0;
}
```

Matrisin boyutunun değişmediği durumda resize() fonksiyonu bir işlem yapmaz, diğer durumlarda ise yıkıcı bir fonksiyonudur: elemanların değerleri değişebilir. Eğer değişkenlerin değişmesi istenmiyorsa conservativeResize() kullanılabilir.

Bu fonksiyonlar sabit boyutlu matrisler için de geçerlidir. Tabii ki sabit boyutlu bir matris yeniden boyutlandırılamaz. Sabit boyutu farklı bir değerle değiştirmek bir ekleme hatasını tetikleyecektir; fakat aşağıdaki kod hata vermez:

```
#include "stdafx.h"
```

```
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 Matrix4d m;
 m.resize(4, 4); // işlem yapılmaz
 // m.resize(5, 5); // hata verir
 cout << "m matrisinin boyutu: "
  << m.rows() << "x" << m.cols() << std::endl;

    return 0;
}
```

2.6. Atama ve Yeniden Boyutlandırma

Atama, operator= ile bir matrisi bir diğerine kopyalama işlemidir. Eigen otomatik olarak eşitliğin solundaki matrisi eşitliğin sağındaki matrisin boyutuna boyutlandırır. Örneğin:

```
#include "stdafx.h"
#include <Eigen>
#include <iostream>

using namespace Eigen;
using namespace std;

int main()
{
 MatrixXf a(2, 2);
 cout << "a matrisinin boyutu: " << a.rows() << "x" << a.cols() << endl;
 MatrixXf b(3, 3);
 a = b;
 cout << "a matrisinin yeni boyutu: " << a.rows() << "x" << a.cols() <<
 endl;

    return 0;
}
```

Tabii ki, eşitliğin sol tarafı sabit boyutlu ise buna izin verilmeyecektir.

2.7. Sabit vs. Dinamik Boyut

Ne zaman sabit boyut (Matrix4f), ne zaman dinamik boyut (MatrixXf) tercih edilmelidir? Küçük boyutlar için olabildiğince sabit boyut kullanılmalıdır ve büyük boyutlar için dinamik boyut kullanılabilir. Küçük boyutlar için, özellikle (yaklaşık olarak) 16'dan küçük boyutlar için sabit boyut kullanmak performans açısından büyük yarar sağlar, Eigen'in dinamik hafıza tutmasını engellemeyi sağlar. Sabit boyutlu Eigen matrisi sadece bir dizidir:

```
Matrix4f m;
```

şuna denk gelir:

```
float m[16];
```

yani çalışma zamanı maliyeti yoktur. Buna karşın, dinamik boyutlu matrisin dizisi heap üzerinde yer alır.

```
MatrixXf m(satir*sutun);
```

şuna denk gelir:

```
float *m = new float[satir*sutun];
```

ve buna ek olarak MatrixXf nesnesi satır ve sütun sayılarını üye değişken olarak saklar.

Sabit boyut kullanmanın kısıtı tabii ki derleme zamanındaki boyutun değiştirilememesidir. Ayrıca büyük boyutlar için, (yaklaşık olarak) 32'den büyük boyutlar için sabit boyutun sağladığı performans yararı önemsizdir.

2.8. Diğer Template Parametreleri

Konunun başında Matrix sınıfının 6 template parametresi olduğu belirtilmişti fakat henüz sadece ilk üçü incelendi. Diğer 3 parametre seçmelidir:

```
Matrix<typename Scalar,
```

```
        int RowsAtCompileTime,
        int ColsAtCompileTime,
        int Options = 0,
        int MaxRowsAtCompileTime = RowsAtCompileTime,
        int MaxsColsAtCompileTime = ColsAtCompileTime>
```

- MaxRowsAtCompileTime ve MaxColsAtCompileTime belirlenmek istendiği zaman kullanışlı olabilir, derlenme zamanında matrisin kesin boyutu bilinmese bile, sabit bir üst sınır derlenme zamanında bilinir. Bunu yapmanın en önemli sebebi dinameik hafıza ayırmayı önlemektir. Örneğin aşağıdaki matris dinamik yer alma yapmadan 12 float'lık bir dizi oluşturur:

```
Matrix<float, Dynamic, Dynamic, 0, 3, 4>
```