제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

# 원자 블록 합병을 이용한 동영상 압축

이동희[1], 위영철[2]
아주대학교 정보통신전문대학원

e-mail : {[1] poison, [2] ycwee}@ajou.ac.kr

# Moving Image Compression Using Atomic Block

Donghee Lee, Youngcheol Wee
Graduate School of Information and Communication Ajou Univ.

요 약

이미지 압축에서 레인지 블록들을 분할하는 것은 매우 중요한 문제이다. 본 논문에서는 프랙탈 이미지 압축기법에서 사용하는 분할-합병 기법을 동영상의 압축에 적용하였다. Quad-tree 방식은 top-down 방식으로 유사한 도메인 블록을 찾지 못하였을 경우 그 레인지를 4 등분하여 각각의 도메인 블록을 찾는 방식이다. 분할-합병 기법은 bottom-up 방식으로 Quad-tree 방식과 달리 주어진 크기의 원자 블록(atomic block)들을 도메인과의 유사성에 따라 연결함으로써 레인지를 불규칙적인 형태로 합병할 수 있다. 이 방식을 이용하여 동영상에서의 화질과 압축률의 개선을 기대할 수 있다.

## 1. 서론

동영상은 여러 장의 정지된 이미지로 구성되어 있다고 볼 수 있다. 정지된 이미지 하나를 하나의 프레임이라고 할 때, 동영상을 효율적으로 압축하기 위해서는 프레임들간의 관계를 이용해야 한다. 프레임들간의 유사성을 많이 활용할수록 높은 압축률을 얻을 수가 있다. 이전 프레임을 이용하여 다음 프레임을 압축한다고 했을 때, 현재의 프레임은 레인지가 되고 이전의 프레임은 도메인이 된다. 도메인에서 레인지와 유사한 부분을 찾아 매치시킴으로써 많은 양의 정보를 줄일 수 있다.

프랙탈 이미지 압축에서는 동영상 압축과는 다르게 레인지와 도메인이 같은 정지 이미지 안에 존재한다. 하지만 동영상 압축에서와 마찬가지로 도메인과의 유사성을 이용하기 때문에 도메인과 레인지를 매치시키는 방식과 정보를 저장하는 방식에 대한 문제는 동영상 압축과 프랙탈 이미지 압축이 동시에 갖고 있는 문제라 할 수 있다. 본 논문에서는 프래탈 이미지 압축에 쓰이는 방식을 동영상 압축에 적용해 보고자 한다.

프랙탈 이미지 컴프레션에서 이미지 분할(image partitioning)은 중요한 문제이다. 이미지 분할 방법에 따라 유사한 블록을 찾는 방법이 결정되고 이것은 압축률과 직접적인 관련이 있기 때문이다. 이미지 분할 방식은 크게 두 가지로 나눌 수 있다. 하나는 구조적인 방법(hierachical partitioning)이며 다른 하나는 분할-합병 방식(split-and-merge)이다. 구조적인 방법에는 Quadtree 방식 [1], HV partitioning [2], Polygonal partitioning [3] 등이 있다. 그리고 분할-합병 방식에는 Delaunay triangulations [5][6], Quadrilateral [7], Heuristic search [8], Evolutionary [9] 등이 있다.

본 논문에서는 분할-합병 방식을 동영상 이미지에 적용하였다. 분할-합병 방식에서 레인지들은 초기에 원자 블록(atomic block)이라고 불리는 작은 정사각형으로 나누어 진다. 후에 이 이미지 블록들이 서로 합병되어 이어짐으로써 불규칙한 형태의 레인지로 변하게 된다. 하나의 원자 블록을 표현하는 방법에는 두 가지가 대표적이다. 하나는 레인지의 경계를 표시하는 체인코드(chain codes)방식이고 다른 하나는 두 비트로 다른 블록과 이어졌는지 여부를 표시하는 지역경계지도(region edge map)방식이다. 후자가 전자보다 효율이 높은 것으로 알려져 있으므로 본 논문에서는 후자를 선택하였다[10]. 지역경계지도방식에서 두 비트는 해당 원자 블록의 왼쪽에 모서리(edge)가 있는지 또는

위쪽에 모서리가 있는지를 표현하는 데 사용된다. 예를 들면, <그림 1>에서와 같이 원자 블록의 왼쪽과 위쪽에 모서리가 있다면 3 으로 표현되고 왼쪽에만 모서리가 있다면 2 가 되며 위쪽에만 모서리가 있다면 1 로 표현된다. 그리고 왼쪽과 위쪽에 모두 모서리가 없다면 0 으로 표현된다.

| | | | |
|---|---|---|---|
| 3 | 3 | 3 | 1 |
| 3 | 0 | 1 | 2 |
| 2 | 3 | 1 | 3 |
| 3 | 0 | 3 | 0 |

<그림 1>

## 2. 적용 알고리즘

인코딩이 될 현재의 이미지는 일단 4x4 pixel 크기의 원자 블록들로 나뉘게 된다. 이미지의 가로 크기를 X 라하고 세로 크기를 Y 라 하면 레인지의 총 전체 개수는 (X/4)x(Y/4)개가 된다. 이들 원자 블록들이 최초의 레인지가 된다. 그리고 나서 각각의 레인지들에 대해 가장 작은 에러값을 갖는 블록 즉, 가장 유사한 도메인 블록 N 개를 일정범위 내에서 풀-서치(full-search)로 찾은 다음에 그 블록들을 에러가 작은 순으로 정렬한다. 그런 다음에는 두 레인지가 합병되었을 경우에 에러를 가장 적게 증가시키는 두 레인지부터 합병하게 되는 것이다. 여기서 어떻게 에러를 가장 작게 증가시키는 두 레인지를 선택할 수 있는 지에 관한 문제가 발생한다. 이 문제를 해결하기 위해서 우선순위 큐(priority queue)를 이용한다. 큐는 힙(heap)으로 구현된다. 이 큐에는 이웃한 레인지를 합병했을 경우에 가장 작은 에러값을 내는 두 레인지들이 순서대로 정렬되어 있다. 따라서 큐의 맨 위에 있는 엘리먼트부터 합병을 해나가면 결국 가장 작은 에러값을 내는 두 레인지부터 차례로 합병한 것과 같은 결과를 얻게 된다.

각각의 레인지들은 N 개의 도메인을 가지고 있다. 두 레인지를 합병할 때에는 두 레인지가 가지고 있는 모든 도메인들의 위치에 대해서 합병되었을 경우에 대한 레인지의 에러값을 구해야 하므로 결국 하나의 레인지당 2xN 개의 도메인에 대해 에러값을 계산하게 된다. 그리고 나서 다시 가장 작은 에러를 증가시키는 도메인 N 개만 남겨두고 나머지는 버린다. 본 논문에서는 N 을 5 로 설정하였다.

합병이 되면서 레인지들의 형태가 변하게 된다. 그런데 우선순위 큐는 여전히 이전 형태의 레인지 정보로 구성되어 있으므로 큐를 갱신해야 할 필요가 생긴다. 한 번 합병한 뒤에 우선순위 큐를 갱신해도 상관은 없다. 하지만 합병된 두 레인지가 하나의 레인지가 되었을 때 그 레인지는 다른 레인지들에 비해 더 큰 에러값을 가질 확률이 크기 때문에 이 레인지가 합병될 차례는 늦어질 가능성이 크다. 따라서 합병이 일어날 때마다 큐를 갱신하지 않고 큐에 있는 순서대로 합병을 하면서 유효하지 않은 레인지가 나왔을 때 큐를 갱신하는 것이 효과적이다.

두 레인지 사이에 합병이 일어났을 때 한 프레임에 들어있는 전체 레인지 수가 하나 감소하게 된다. 그러므로 한 프레임에서의 전체 레인지의 최소 개수를 미리 설정함으로써 한 프레임을 인코딩하는 프로세스를 종료하는 시기를 정해 둘 수 있다. 프로세스를 종료하는 다른 하나의 방법은 레인지가 합병될 때마다 증가하는 에러를 구하여 일정량 이상을 초과하지 않을 때까지 프로세스를 진행하는 것이다.[11]

다음은 한 프레임을 처리하는 알고리즘이다.

**INPUT**: image f, 레인지당 남겨둘 도메인 개수 N, 프레임당 최종 레인지 개수 k, 원자 블록 크기 j.

- 원자 블록 사이즈 j 로 나누어진 레인지들에 대해 최소의 에러값을 갖는 도메인 N 개를 구한다.
- 합병했을 때 에러가 작게 증가하는 순서대로 정렬한다. ( 우선순위 큐)

While 현재 프레임의 레인지 개수 > k do

- 우선순위 큐의 최상위 엘리먼트 꺼낸다.

If 해당 레인지가 유효하다면 then

- 두 레인지 합병
- 새로운 레인지에 대한 N 개의 도메인 결정
- 현재 레인지 개수 ← 현재 레인지 개수 – 1

Else

- 우선순위 큐 갱신(두 레인지의 에러값을 다시 계산한 후 큐에 넣기)

End If

End While

**OUTPUT**: 원자 블록들이 합병된 이미지, 각각의 레인지에 대한 도메인 주소.

## 3. 결론

본 연구에서는 동영상의 압축에 분할-합병 기법을 적용해 보았다. 효율을 알아보기 위해 전통적인 방법의 하나인 Quad-tree 방식과 비교해 보았다.

Quad-tree 방식에서는 8x8 pixel 크기로 먼저 정해진 범위 내에서 도메인을 풀-서치(full-search)하고 나서 하나의 가장 유사한 도메인의 위치를 저장한다. 만약 유사한 블록을 찾지 못했을 경우에는 해당 레인지를 4x4 pixel 크기로 나누어 4 개의 레인지에 대해 각각 가장 유사한 도메인을 찾는 방식이다. 4x4 크기의 레인지들 역시 정해진 범위 내에서 풀-서치(full-search)를 적용했다.

8x8 블록의 풀-서치 범위는 자기자신의 위치에서 2 pixel 씩 상하좌우로 움직인 크기이며, 4x4 레인지의 풀-서치 범위는 자기 위치에서 5 pixel 씩 상하좌우로 움직인 크기로 설정하였다. 마찬가지로 본 연구에서 적용한 분할-합병 방식에서는 초기화중에 풀-서치를

| news | | |
|---|---|---|
| Ratio | Total Error | range |
| 12.80000 | 3803679 | 6336 |
| 18.81049 | 3804307 | 4000 |
| 23.54296 | 3823633 | 3000 |
| 31.45718 | 3826807 | 2000 |
| 47.38673 | 3950848 | 1000 |
| foreman | | |
| Ratio | Total Error | range |
| 12.80000 | 12119530 | 6336 |
| 18.81049 | 12119542 | 4000 |
| 23.54296 | 12147521 | 3000 |
| 31.45718 | 12337729 | 2000 |
| 47.38673 | 13071595 | 1000 |
| stefan | | |
| Ratio | Total Error | range |
| 12.80000 | 28097306 | 6336 |
| 18.81049 | 28122482 | 4000 |
| 23.54296 | 2824813 | 3000 |
| 31.45718 | 28850544 | 2000 |
| 47.38673 | 30543914 | 1000 |

[표 1: Our Method]

| news | | | |
|---|---|---|---|
| Ratio | Total Error | 8x8 suc | 8x8 fail |
| 14.49057 | 3803679 | 0 | 1584 |
| 20.73635 | 3804419 | 654 | 930 |
| 25.26254 | 3854127 | 914 | 670 |
| 41.52670 | 4069144 | 1431 | 153 |
| 48.20484 | 4140549 | 1501 | 83 |
| foreman | | | |
| Ratio | Total Error | 8x8 suc | 8x8 fail |
| 14.49057 | 12119530 | 0 | 1584 |
| 16.08338 | 12277363 | 228 | 1296 |
| 25.86358 | 13099045 | 1045 | 539 |
| 33.90565 | 13992446 | 1318 | 266 |
| 46.98238 | 16037493 | 1540 | 44 |
| stefan | | | |
| Ratio | Total Error | 8x8 suc | 8x8 fail |
| 14.49057 | 28097306 | 0 | 1584 |
| 20.40716 | 28526376 | 505 | 1079 |
| 26.77095 | 29857628 | 736 | 848 |
| 30.14483 | 30902916 | 795 | 789 |
| 48.39256 | 46187744 | 1508 | 76 |

[표 2: Quadtree]

하는 범위 역시 자기 위치에서 5 pixel 씩 상하좌우로 움직인 크기로 설정하였다. 따라서 Quad-tree 방식에서 모두다 4x4 풀-서치를 한 결과와 분할-합병 방식에서 합병이 일어나지 않고 풀-서치만 한 경우에는 한 프레임당 증가된 에러의 크기가 같게 된다. 이 때가 두 가지 방식 모두에서 가장 좋은 화질을 보여주는 상태이다.

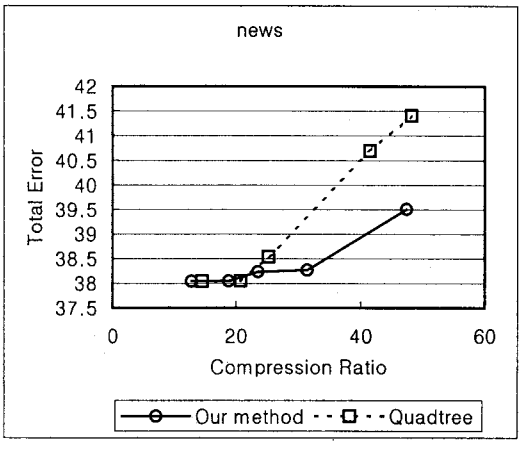
Quad-tree 방식에는 8x8 블록의 서치가 더 많이 성공할수록 압축률은 올라가고 화질은 떨어진다. 분할-합병 방식에서는 합병을 많이 할수록 압축률이 올라가고 화질은 떨어지게 된다.

Quad-tree 방식에서 인코딩해야 할 대상은 다음과 같다. 8x8 크기의 서치가 성공했는지 아닌지를 알기 위해 각각의 8x8 블록에 대해 한 비트가 필요하며 성공한 블록은 도메인 주소를 저장할 비트가 필요하고 실패한 블록은 나누어진 4 개의 블록에 대해 각각 4 개의 도메인 주소를 저장할 비트가 필요하다. 분할-합병 방식에서는 원자 블록들에 대해 합병된 레인지의 모서리를 표현하기 위해 2 비트가 들고 레인지의 개수만큼 도메인 주소를 저장할 비트가 필요하다. 본 실험에서 도메인 주소를 저장하기 위한 주소로는 13 비트를 사용하였다. 비교하는 두 가지 방식 모두에서 엔트로피 코딩(entropy coding)은 적용하지 않았다.

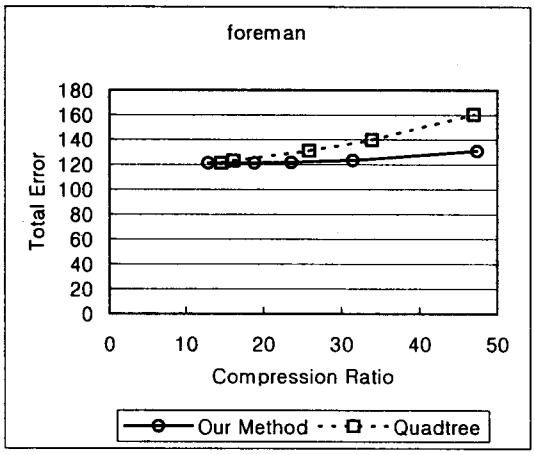
[표 1]과 [표 2]는 세 가지의 동영상인 352x288 pixel 크기의 이미지 50 프레임에서 첫 프레임을 제외한 49 프레임에 대해 위에서 설명한 두 가지 방식을 적용한 결과이다. 이것을 그래프로 표현한 것이 [그림 2], [그림 3], [그림 4] 이다. 그림에서 알 수 있듯이 압축률이 같을 때에 분할-합병 방식이 Quad-tree 방식보다 더 적게 에러를 증가시킨다는 것을 알 수 있다.

인코딩된 이미지와 오리지널 이미지와의 비교를 위해 두 이미지 사이의 에러값을 이용하였다. 에러값은 오리지널 이미지와 인코드된 이미지에 대한 각각의 pixel 값의 차이의 절대값들의 합으로 계산된다.

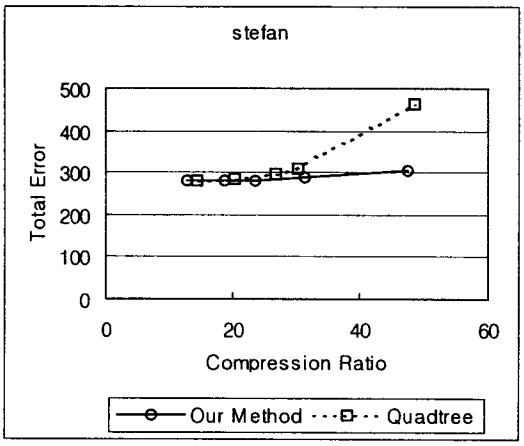
분할-합병 방식에서 한 프레임에서 합병을 하지 않은 상태에서의 레인지의 개수는 총 6336 개이다. 이때의 Total Error 는 4x4 풀-서치를 한 것과 마찬가지 이므로 Quadtree 방식에서 모두 8x8 fail 이 일어났을 때의 Total Error 와 같게 된다.




[그림 2]




[그림 3]




[그림 4]

News 가 가장 움직임이 적은 영상이고 Foreman, Stefan 순으로 움직임이 많은 영상이다. 움직임이 많을수록 Total Error 가 증가하는 것을 볼 수 있다. 그래프에는 Total Error 을 100000 으로 나눈 값으로 표시하였다.

그림의 형식은 YUV 형식이다. 이 중 UV 는 컬러에 해당된다. 그런데 명암에 해당하는 Y 에 비해 상대적으로 비중이 낮기 때문에 실험은 Y 에 대하여만 국한하였다.

분할-합병 방식이 증가시키는 에러가 적으므로 같은 압축률에서 더 화질이 좋다고 볼수 있다. 하지만, Quad-tree 방식에 비해 분할-합병 방식으로 인코딩 하는 데는 더 많은 시간이 소요된다. 분할-합병 방식은 초기화 과정에서 무조건 풀-서치를 필요로 하기 때문이다. 그뿐아니라 추가적으로 합병이 일어날 때마다 레인지와 도메인의 에러값을 비교해야 한다.

차후의 연구에서는 UV 에 대한 본 알고리즘의 적용과 초기화 과정에서의 시간을 줄일 수 있는 방안에 대한 연구가 필요할 것으로 본다.

## 참고문헌

[1] E. Jacobs, Y. Fisher, R.D. Boss. "Image compression: A study of the iterated transform method". Signal Processing 29, 1992, pp 251-263.

[2] Y. Fisher, S. Menlove, "Fractal encoding with HV partitions", Fractal Image Compression – Theory and application. New York, NY: Springer-Verleg, 1994.

[3] E. Reusens, "Partitioning complexity issue for Iterated Functions Systems based image coding". In Proc. Of VII European Signal Processing Conference, Vol. 1, Edinburg, U.K., September 1994, pp 171-174

[4] M. Tanimoto, H. Ohyama, S. Katsuyama, T. Kimoto, T. Fujii, "A new fractal image coding employing blocks of variable shapes," in Proc. Of IEEE Interational Conference on Image Processing(ICIP'96), Laussane, Sept. 1996

[5] F. Davoine, M. Antonini, J.-M. Chassery, M. Barlaud, "Fractal image compression based on Delaunay triangulation and vector quantization," in Proc. of IEEE Transactions on Image Processing Vol. 5, No. 2, February 1996, pp 338-346

[6] F. Davoine, E. Bertin, J.-M. Chassery, "From rigidity to adaptive tessellations for fractal image compression: comparative studies," in Proc. of IEEE 8th Workshop on Image and Multidimensional Signal Processing, Cannes, September 1993, pp. 56-57

[7] F. Davoine, J. Svensson, J.-M. Chassery, "A mixed triangular and quadrilateral partition for fractal image coding," in Proc. of IEEE International Conference on Image Processing (ICIP'95), Washington, D.C., 1995

[8] Thomas, L., Deravi, F., "Region-based fractal image compression using heuristic search", IEEE Transactions on Image Processing 4,6, 1995, pp. 832-838.

[9] Saupe, D., Ruhl, M., "Evolutionary fractal image compression", Proceedings IEEE ICIP, Lausanne, 1996, pp. 129-132

[10] Hartenstein H., Ruhl M., Saupe D., "Region Based Fractal Image Compression", IEEE Transactions on Image Processing, vol. 9, no. 7, pp. 1171-1184, July 2000.

[11] Ruhl M., Hartenstein H., Saupe D., "Adaptive partitionings fractal image compression", Proceedings IEEE International Conference on Image Processing (ICIP), Santa Barbara, October 1997.