복셀맵을 기반으로 한 분자 간 상호작용 인터페이스의 계산

최 지훈[1] 김 병주[2] 김 구진[3]

[1] 경북대학교 전자전기컴퓨터학부 대학원

[2]경북대학교 전자공학과 대학원

[3] (교신저자) 경북대학교 컴퓨터학부

azureripple@yahoo.co.kr, kbj113@yahoo.co.kr, kujinkim@yahoo.com

Molecular Interaction Interface Computing Based on Voxel Map

Jihoon Choi[1] Byungjoo Kim[2] Ku-Jin Kim[3]

[1]Graduate School of EECS, Kyungpook National University

[2]Graduate School of Electronics, Kyungpook National University

[3] (Corresponding author) School of Computer Science & Engineering, Kyungpook National University

요 약

본 논문에서는 단백질 분자 간의 인터페이스를 계산하는 알고리즘을 제안한다. 분자가 반데르바스 (van der Waals) 반경을 갖는 구의 집합으로 표현될 때, 공간 상의 한 점 p로부터 분자까지의 거리는 p로부터 가장 가까운 구까지의 거리에 대응한다. 분자 인터페이스는 두 개의 분자에 대해 같은 거리에 있는 점들로 구성된다. 제안된 알고리즘은 공간을 복셀의 집합로 분할한 뒤, 각 복셀을 지나는 구의 위치 정보를 저장하여 복셀맵 (voxel map)을 구성하였다. 복셀맵을 이용하여 한 점으로부터 분자까지의 거리를 계산하며, GPU (graphic processor unit)를 이용하여 병렬처리를 수행함으로써 효율적으로 인터페이스를 근사한다.

Abstract

In this paper, we propose a method to compute the interface between protein molecules. When a molecule is represented as a set of spheres with van der Waals radii, the distance from a spatial point p to the molecule corresponds to the distance from p to the closest sphere. The molecular interface is composed of equi-distant points from two molecules. Our algorithm decomposes the space into a set of voxels, and then constructs a voxel map by storing the information of spheres intersecting each voxel. By using the voxel map, we compute the distance between a point and the molecule. We also use

투고일 : 2012.01.20 / 심사일 : 3차 2012.07.05 / 게재확정일 : 2012.07.10

GPU for the parallel processing, and efficiently approximate the interface of a pair of molecules.

키워드: 단백질 분자, 복셀맵, 분자 인터페이스, GPU

Keywords: protein molecule, voxel map, molecular interface, GPU

1. 서론

생명체의 생리적인 현상에 있어서 단백질 분자 간의 상호작용은 매우 중요한 역할을 수행한다. 세포의 분할이나 성장뿐만 아니라 신약 개발을 위한 분자 간의 도킹 (docking), 바인딩 (binding) 등에 있어서도 분자 간의 상호작용이 필요하다 [1-3]. 이에 따라 분자 간의 상호작용이 일어나는 인터페이스를 효율적으로 계산하고 가시화하기 위해 수많은 연구자들이 연구를 수행하여 왔다 [4-6]. 기존의 연구들은 많은 연산량으로 인해 실시간에 분자 간의 인터페이스를 계산하는 방법을 제시하지는 못 하였다.

최근에 들어서서 분자 간의 인터페이스를 빠르게 계산하기 위한 방법이 제시되었다 [7]. Seong 등[7]은 단백질 분자를 kd-tree로 구성하여 임의의 점과 분자 사이의 거리계산을 효율적으로 수행하였고, 이를 바탕으로 BP tree를 사용하여 분자 인터페이스를 포함하는 공간 영역을 복셀들로 추출하였다. Marching cube 알고리즘[8]을 이용하여 추출된 복셀들로부터 인터페이스를 곡면으로 근사하고, 가시화하였다. 이들은 kd-tree와 BP tree를 활용하여 계산량을 최소화함으로써 실시간에 분자 간의 인터페이스를 생성하였다.

기존 연구에서는 단백질 분자를 표현하는 자료구조로 격자 (grid) 형태를 이용하여 원자 위치를 sampling하거나 원자들을 계층적으로 그룹화 하여 트리(tree) 형태로 나타내는 방법을 주로 사용하였다 [9,10]. 격자 형태는 정확도 측면에서 오차가 많고, 트리 형태는 트리의 차수(degree) 및 깊이(depth)가 커서 포인터를 사용해야 한다는 한계점으로 인해 CUDA 기반의 병렬처리를 수행하기에 부적합하다. 본 논문에서는 포인터의 사용이 필요하지 않고, 병렬처리에 효과적이면서 오차가 적은 계산결과를 얻을 수 있는 복셀맵 자료구조를 이용하여 분자를 표현한다.

본 논문에서는 분자 간의 인터페이스를 발견하는 효율적인 알고리즘을 제안한다. 분자 간의 상호작용은 서로 다른 분자에 속해 있으면서 가장 가까운 영역에 있는 원자들에서 발생할 수 있다. 따라서 분자 인터페이스는 공간 상에서 두 개의 분자로부터 같은 거리에 있는 점들의 집합으로 정의할 수 있다. 이러한 점들을 발견하고 곡면으로 근사하여 분자 인터페이스를 추출한다.

단백질 분자에 속한 각 원자들은 반데르바스 (Van der Waals) 반경을 갖는 구로 표현할 수 있다. 따라서 단백질 분자는 반경이 서로 다른 구의 집합으로 표현된다. 이 때 각 구의 중심점은 원자의 중심점에 대응하고, 구의 반경은 원자의 반데르바스 반경에 대응한다. 분자 인터페이스를 효율적으로 계산하기 위해서는 공간 상의 점으로부터 분자까지의 거리 계산이 효율적으로 수행되어야 한다. 이를 위해 공간을 복셀로 분할하고, 각 복셀을 지나는 구의 정보를 저장하여 복셀맵을 구성한다. 또한, GPU 를 활용하는 CUDA 프로그램을 이용하여 공간 상의 각 점과 분자 간의 거리를 병렬로 계산함으로써 효율적으로 분자 인터페이스를 추출한다.

본 논문의 구성은 다음과 같다. 2 절에서는 복셀맵의 구성방법 및 거리 계산에서의 활용방법을 설명한다. 3 절에서는 복셀맵을 이용한 인터페이스 계산 방법을 제시한다. 4 절에서는 실험결과를 제시하고, 5 절에서 결론을 내린다.

2. 복셀맵의 구성 및 활용

원자의 중심점과 반데르바스 반경이 각각 $\mathbf{c}_i, r_i$라 할 때, 다음과 같이 구 $S(\mathbf{c}_i, r_i)$에 대응시킬 수 있다.

$$S(\mathbf{c}_i, r_i) = \{\mathbf{p} \in \mathrm{R}^3 \mid \|\mathbf{p} - \mathbf{c}_i\| = r_i\}$$

구 $S(\mathbf{c}_i, r_i)$와 내부의 모든 점을 포함하는 영역을 ball로 표현하며 $B(\mathbf{c}_i, r_i)$로 나타낸다.

$$B(\mathbf{c}_i, r_i) = \{\mathbf{p} \in \mathrm{R}^3 \mid \|\mathbf{p} - \mathbf{c}_i\| \le r_i\}.$$

한 개의 분자는 구의 집합 M으로 다음과 같이 정의된다.

$$M = \{S(\mathbf{c}_i, r_i) \mid 0 \le i < n\}.$$

두 개의 분자 M_1과 M_2에 속한 ball 들을 각각 $B_{1,i}$와 $B_{2,i}$로 나타낼 때, M_1과 M_2에 대한 복셀맵을 구성하는 알고리즘은 다음과 같다. M_1과 M_2를 포함하는 바운딩 박스(bounding box) B가 주어질 때, x, y, z 축 방향의 크기가 각각 S_x, S_y, S_z인 B를 $n_x \times n_y \times n_z$ 개의 복셀로 분할하여 복셀의 집합 V를 구성한다. 이때, 각 복셀 $V^{\alpha\beta\gamma}(0 \le \alpha < n_x, 0 \le \beta < n_y, 0 \le \gamma < n_z)$는 다음과 같은 성질을 갖는다.

$$\begin{aligned} V^{\alpha\beta\gamma} = \{ (x, y, z) \mid & \alpha\,(S_x/n_x) \le x \le (\alpha + 1)\,(S_x/n_x), \\ & \beta\,(S_y/n_y) \le y \le (\beta + 1)\,(S_y/n_y), \\ & \gamma\,(S_z/n_z) \le z \le (\gamma + 1)\,(S_z/n_z) \}. \end{aligned}$$

$V^{\alpha\beta\gamma}.box$ 가 복셀 $V^{\alpha\beta\gamma}$의 영역을 나타낼 때, $V^{\alpha\beta\gamma}.balls$ 는 다음과 같이 복셀에 포함된 원자의 정보를 가진다.

$$V^{\alpha\beta\gamma}.balls = \{B_{i,j} \mid B_{i,j} \cap V^{\alpha\beta\gamma}.box \ne \varnothing\}.$$

또한, $V^{\alpha\beta\gamma}.\mathrm{nn}(l)$는 $V^{\alpha\beta\gamma}$ 복셀을 포함하여 l 개의 복셀에 해당하는 거리 이내에 이웃하고 있는 복셀들을 나타낸다. l 이 0 보다 크거나 같은 값으로 주어질 때, $V^{\alpha\beta\gamma}.nn(l)$는 다음과 같다.

$$\begin{aligned} V^{\alpha\beta\gamma}.nn(l) = \{ V^{ijk} \mid & \alpha - l \le i \le \alpha + l, \\ & \beta - l \le j \le \beta + l, \\ & \gamma - l \le k \le \gamma + l \}. \end{aligned}$$

l 이 0 보다 작은 값으로 주어질 때, $V^{\alpha\beta\gamma}.nn(l)$ 는 공집합이다.

복셀맵을 구성하는 알고리즘은 CUDA 언어를 이용하여 각 원자 별로 thread를 할당하여 병렬로 처리된다. 알고리즘 1에서는 GPU를 이용하여 복셀맵을 구성하는 과정을 보인다.

Algorithm 1: Voxel map construction
function VoxelMap (M_1, M_2, V)
Begin
For each $B(\mathbf{c}, r) \in M_1 \cup M_2$ **do in parallel** /* GPU code */
  VS := the set of voxels that intersect with $B(\mathbf{c}, r)$;
  For each voxel $V \in VS$ **do**
    Add $B(\mathbf{c}, r)$ to $V.atoms$;
End.

공간 상의 한 점 $\mathbf{p}$로부터 분자 $M = \{S(\mathbf{c}_i, r_i) \mid 0 \le i < n\}$에 대한 Euclidean distance는 다음의 식에 의해 계산된다.

$$\mathrm{Dist}(\mathbf{p}, M) = \mathrm{Min}_{0 \le i < n}(\|\mathbf{p} - c_i\| - r_i).$$

복셀맵을 이용하면 공간 상의 한 점으로부터 분자까지의 거리를 효율적으로 계산할 수 있다. 복셀맵 구조에서 효율적으로 최소 거리를 계산하기 위해 점 $\mathbf{p}$가 속한 복셀 $V_\mathbf{p}$를 포함하여 $V_\mathbf{p}$ 와 이웃하고 있는 복셀들에 대해 차츰 범위를 넓혀가며 탐색한다. 복셀에 포함된 원자마다 $\mathbf{p}$까지의 거리를 계산하는데, 만약 어떤 원자에 대한 거리 d가 이전에 구한 min_dist보다 작으면 d를 새로운 min_dist로 결정한다. 수정된 min_dist 이내에 $\mathbf{p}$와 더 가까운 다른 원자가 있는 지 확인하는 추가 작업이 필요하므로, max_step 값을 이용하여 거리를 계산할 복셀의 범위를 제한한다. 알고리즘 2에서는 GPU를 이용하여 거리를 계산하는 과정을 보인다.

Algorithm 2: Distance computation
function Dist ($\mathbf{p}$, M)
Begin
$max\_step := \infty$;
$min\_dist := \infty$;
$update := false$;
For $step := 0$ **to** max_step
  $V_\mathbf{p}$:= the voxel containing $\mathbf{p}$;
  $V_l := V_\mathbf{p}.nn(step)$;

```
    V2 := Vp.nn(step-1);
  V := V1 – V2;
    For each v ∈ V
    For each B(c, r) ∈ v.atoms
      if ‖ p – q ‖ – r ≤ min_dist then
        min_dist := distance;
        update := true;
    if update = true then
      max_step := Ceil( min_dist / min(Sx/nx, Sy/ny, Sz/nz));
return min_dist;
End.
```

3. 분자 인터페이스의 계산

두 개의 분자 M_1과 M_2 와 같은 거리에 있는 점을 **q**라 할 때, **q**는 다음 성질을 만족한다.

$$\{\mathbf{q} \mid \mathrm{Dist}(\mathbf{q}, M_1) = \mathrm{Dist}(\mathbf{q}, M_2)\}.$$

공간 상의 임의의 점 **p**가 주어질 때, 다음의 함수 F(**p**)=0을 만족하는 모든 점의 집합이 분자 인터페이스라고 할 수 있다.

$$F(\mathbf{p}) = \mathrm{Dist}(\mathbf{p}, M_1) - \mathrm{Dist}(\mathbf{p}, M_2).$$

따라서, 분자 인터페이스를 추출하는 문제는 함수의 제로-셋(zero-set)을 추출하는 문제로 대응될 수 있다. 함수의 제로-셋을 발견하는 효율적인 방법으로 marching cube 알고리즘 [8]이 널리 사용되고 있다. Marching cube 알고리즘은 3차원 공간을 일정한 크기의 격자로 분할하고, 격자의 각 정점마다 지정된 함수에 의해 스칼라(scalar) 값을 부여한다. 각 격자에 존재하는 8개의 정점들이 가진 스칼라 값에 따라 격자에 포함될 수 있는 제로-셋 곡면의 형태는 256가지로 분류된다. 분류된 형태에 맞게 각 격자 별로 포함된 제로-셋 곡면을 추출한다.

Marching cube 알고리즘은 각 격자에서 수행되는 작업 (각 정점에 스칼라 값을 부여하는 작업과, 격자에 포함된 제로-셋 곡면의 형태를 분류하고 추출하는 작업)이 격자 간의 의존성이 없이 독립적으로 수행될 수 있다. 따라서, 각 격자 별로 병렬 처리를 수행하기에 용이하다.

본 논문에서 제안한 알고리즘은 GPU 기반의 marching cube 알고리즘 (http://developer.nvidia.com/cuda-toolkit-sdk)을 적용하여 분자 간의 인터페이스를 계산하였으며, 알고리즘 3에 알고리즘을 제시한다.

Algorithm 3: Interface construction

```
function Interface (M1, M2, V)
Begin
New triangleList;
For each v ∈ V do in parallel
  Initialize dist_value;
  Initialize vertex_list;
    cube_index := 0
  For each pi ∈ v.vertices ( 0 ≤ i < 8)
        dist_value[i] := Dist(pi, M1) – Dist(pi, M2);
        if dist_value[i] ≤ 0 then cube_index += 2^i;
  For each ei ∈ v.edges (0 ≤ i < 12)
        ep0 := e.lEnd_point;
        ep1 := e.rEnd_point;
        lerp_param := dist_value[ep0] / (dist_value[ep1]
                    – dist_value[ep0]);
        //linearly interpolate the position where an isosurface cuts
          an edge between two vertices
        vertex_list [i] := Lerp_point(ep0, ep1, lerp_param);

  triangle t := Lookup(cube_index, vertex_list);
  triangleList.add(t)
return triangleList
End.
```

4. 실험 결과

Intel core(TM) i5 CPU (2.8GHz)와 4.0GB의 DRAM 및 nVidia GTX590 GPU가 장착된 컴퓨터에서 실험을 수행하였다. 표 1, 2에서는 각각 marching cube 알고리즘에 사용되는 격자 개수가 16×16×16, 32×32×32일 때, 분자 쌍에 대한 인터페이스 계산 시간을 제시하며, 여기에서 각 분자는 PDB id (http://www.pdb.org)로 표시되었다. 분자 쌍에 대해 생성한 복셀맵은 7.5Å × 7.5Å × 7.5Å 크기의 복셀들로 구성된다. 표 1, 2에서 P.A, P.B는 각각 분자의 PDB id.를 의미한다. #A와 #B는 각각 P.A 및 P.B에 속한 원자의 개수를 나타낸다. D.C는 자료 구조 구성의 계산 시간, I.C는 인터페이스를 구성하기 위해 소요된 계산 시간을 의미한다. 기존 연구와의 비교를 위해 Seong 등 [7]이 제시한 방법을 구현하여 실험한 결과도 함께 제시한다. 공간을 2^{12}개의 복셀로 분할할 경우 본 논문에서 제안한 방법은 CPU 기반으로 우수한 성능을 보여주는 [7]의 방법보다 평균 4.75배 가량 개선된 성능을 보인다. 공간을 2^{15}개의 복셀로 분할할 경우에는 본 논문에서 제안한 방법이 [7]의 방법보다 평균 8.11배 가량 개선된 성능을 보인다. 그림 1에서는 다양한 분자 쌍에 대해 분자 인터페이스를 계산한 예를 보인다.

5. 결론

본 논문에서는 복셀맵을 이용하여 단백질 분자 쌍에 대한 인터페이스를 계산하는 알고리즘을 제시하였다. 제안된 방법은 GPU를 활용한 병렬처리를 수행하여 효율적으로 인터페이스를 계산하였다. 공간을 2^{12}개의 복셀로 분할할 경우 실험을 통해 평균적으로 50ms 내외로 인터페이스를 계산하는 빠른 속도를 보였다.

감사의 글

이 논문은 2012 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2012-0001755).

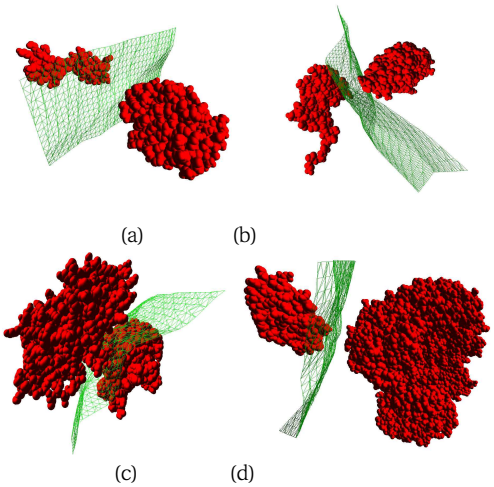

그림 1 분자 인터페이스의 예 (a) 1BTP와 1LU0, (b)1RGH와 1A19, (c)1QLP와 1UTQ, (d)1BU6와 1F3Z.

참고 문헌

[1] D. Levine, M. Facello, P. Hallstrom, G. Reeder, B. Walenz, F. Stevens, Stalk: an interactive system for virtual molecular docking, IEEE Computational Science and Engineering 4 (2) (1997) 55–65.

[2] H. Nagata, H. Mizushima, H. Tanaka, Concept and prototype of protein-ligand docking simulator with force feedback technology, Bioinformatics 18 (1) (2002) 140–146.

[3] S. K. Lai-Yuen, Y. S. Lee, Interactive computer-aided design for molecular docking and assembly, Computer-Aided Design and Applications 3 (6) (2006) 701–709.

[4] N. Ray, X. Cavin, J. C. Paul, B. Maigret, Intersurf: dynamic interface between proteins, Journal of Molecular Graphics and Modelling 23 (4) (2005) 347–354.

[5] W. Humphrey, A. Dalke, K. Schulten, VMD: visual molecular dynamics, Journal of Molecular Graphics and Modelling 14 (1996) 33–38.

[6] S. Gong, G. Yoon, I. Jang, D. Bolser, P. Dafas, M. Schroeder, H. Choi, Y. Cho, K. Han, S. Lee, H. Choi, M. Lappe, L. Holm, S. Kim, D. Oh, J. Bhak, PSIBASE: A database of protein structural interactome map (psimap), Bioinformatics 21 (10) (2005) 2541–2543.

[7] J. Seong, N. Baek, K. Kim, Real-time computation of molecular interaction interfaces, Computer-Aided Design 43 (12) (2011) 1598-1605.

[8] W. E. Lorensen, H. E. Cline, Marching Cubes: A high resolution 3D surface construction algorithm, Computer Graphics 21 (4) (1987) 163-169.

[9] M. Petrek, M. Otyepka, P. Banas, P. Kosinova, J. Koca, CAVER: a new tool to explore routes from protein clefts, pockets and cavities, BMC Bioinformatics 7 (2006) 316.

[10] C. Bajaj, R. A. Chowdhury, M. Rasheed, A dynamic data structure for flexible molecular maintenance and informatics, Proceedings of SIAM/ACM Joint Conference on Geometric and Physical Modeling (2009).

표 1. 분자 인터페이스의 계산 시간 (Marching cube 알고리즘에 사용되는 격자의 개수: 16×16×16)

protein molecules				execution time (msec)		Speedup of our algorithm compared to the algorithm in [7]
P.A	#A	P.B	#B	Algorithm in [7]	Our algorithm	
				CPU : Intel i5 760 2.8GHz RAM : 4.0GB	CPU : Intel i5 760 2.8GHz RAM : 4.0GB GPU : GeForce GTX590	

				D.C.[1]	I.C.[2]	Total	D.C.[1]	I.C.[2]	Total	
1QQU	1,642	1BA7	2,496	11.01	154.26	165.27	0.10	31.75	31.85	5.19
1RGH	2,883	1A19	1,438	11.88	183.32	195.20	0.09	43.79	43.88	4.45
1GJR	2,338	1CZP	1,571	10.21	108.59	118.80	0.11	112.09	112.20	1.06
1QLP	2,956	1UTQ	3,267	60.66	191.18	251.84	0.09	38.61	38.70	6.51
1BTP	1,629	1LU0	476	5.04	114.14	119.18	0.10	30.41	30.51	3.91
1HJ9	3,206	2UUX	434	10.31	147.02	157.33	0.09	13.93	14.02	11.22
1BU6	15,621	1F3Z	1,107	64.81	159.79	224.60	0.16	132.05	132.21	1.70
1O3Y	2,664	1OXY	1,051	21.80	178.53	200.33	0.11	49.60	49.71	4.03
2F0R	2,314	1YJ1	1,738	10.66	224.65	235.31	0.10	34.89	34.99	6.73
1ACC	5,282	1SHU	1,396	19.93	146.61	166.54	0.11	61.49	61.60	2.70
average	4,054	-	1,497	22.63	160.81	183.44	0.11	54.86	54.97	4.75

[1]D.C. : data structure construction time(ms), [2]I.C. : interface computation time(ms)

표 2. 분자 인터페이스의 계산 시간 (Marching cube 알고리즘에 사용되는 격자의 개수: 32 × 32 × 32)

protein molecules				execution time (msec)						Speedup of our algorithm compared to the algorithm in [7]
				Algorithm in [7]			Our algorithm			
P.A	#A	P.B	#B	CPU : Intel i5 760 2.8GHz RAM : 4.0GB			CPU : Intel i5 760 2.8GHz RAM : 4.0GB GPU : GeForce GTX590			
				D.C.[1]	I.C.[2]	Total	D.C.[1]	I.C.[2]	Total	
1QQU	1,642	1BA7	2,496	11.01	594.33	605.34	0.10	63.72	63.82	9.49
1RGH	2,883	1A19	1,438	11.88	648.23	660.11	0.09	96.59	96.68	6.83
1GJR	2,338	1CZP	1,571	10.21	413.01	423.22	0.11	366.78	366.89	1.15
1QLP	2,956	1UTQ	3,267	60.66	750.25	810.91	0.09	79.38	79.47	10.20
1BTP	1,629	1LU0	476	5.04	437.41	442.45	0.10	64.13	64.23	6.89
1HJ9	3,206	2UUX	434	10.31	624.26	634.57	0.09	30.03	30.12	21.07
1BU6	15,621	1F3Z	1,107	64.81	658.38	723.19	0.16	314.02	314.18	2.30
1O3Y	2,664	1OXY	1,051	21.80	633.25	655.05	0.11	111.59	111.70	5.86
2F0R	2,314	1YJ1	1,738	10.66	1021.96	1032.62	0.10	78.72	78.82	13.10
1ACC	5,282	1SHU	1,396	19.93	559.55	579.48	0.11	139.24	139.35	4.16
average	4,054	-	1,497	22.63	634.06	656.69	0.11	134.42	134.53	8.11

[1]D.C. : data structure construction time(ms), [2]I.C. : interface computation time(ms)

〈저자소개〉

최지훈

- 2011년 금오공과대학교 소프트웨어공학과 학사
- 2011년~현재 경북대학교 전자전기컴퓨터 석사과정
- 관심분야: 컴퓨터 그래픽스, 계산기하학, 계산생물학

김병주

- 2002년 경북대학교 전자전기공학부 학사
- 2004년 경북대학교 전자공학과 석사
- 2006년 경북대학교 전자공학과 박사수료
- 2006년~2010년 (주)휴원 과장
- 2010년~2011년 (주)LG전자 MC연구소 선임연구원
- 2011년~현재 경북대학교 전자공학과 박사과정
- 관심분야: 컴퓨터그래픽스, 컴퓨터비전, 기하 모델링, 병렬처리 등

김구진

- 1990년 이화여자대학교 전자계산학과 학사
- 1992년 한국과학기술원 전자계산학과 석사
- 1998년 포항공과대학교 컴퓨터공학과 박사
- 1998년~2000년 Dept. of Computer Sciences, Purdue University, PostDoc.
- 2000년~2002년 아주대학교 정보통신전문대학원 BK21 조교수
- 2002년~2003년 Dept. of Mathematics and Computer Science, University of Missouri-St. Louis, Visiting Assistant Professor
- 2004년~2007년 경북대학교 컴퓨터학부 조교수
- 2008년~현재 경북대학교 컴퓨터학부 부교수
- 관심분야: 컴퓨터 그래픽스, 기하모델링, 계산생물학