[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.12.10 목요일)

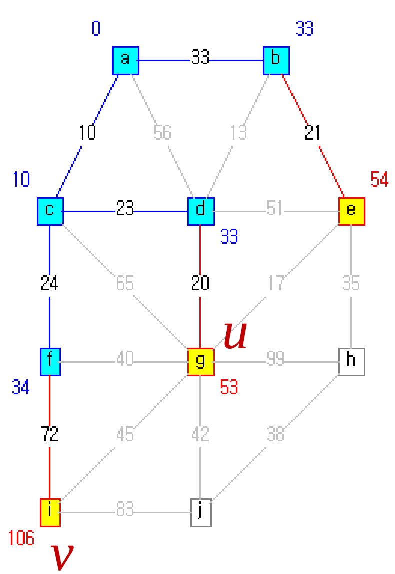
서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

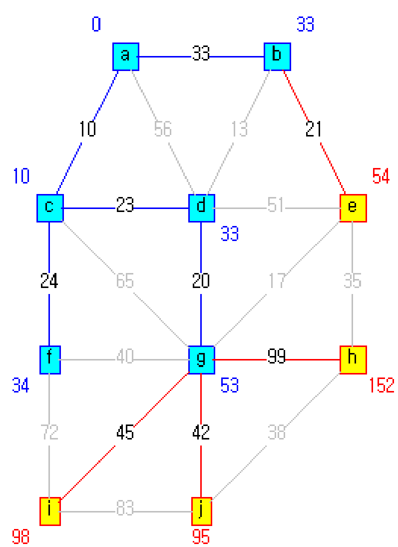
[주제 6]

Graph Algorithms

Dijkstra's Single-Source Shortest Path Algorithm

- **Dijkstra's algorithm**
  - Maintains a set **S** of vertices whose final shortest-path weight from the source **s** have already been determined.
  - 1. Select repeatedly the vertex **u** in **V-S** with the minimum shortest-path estimate, 2. adds **u** to **S**, and 3. relaxes all edges leaving **u**.



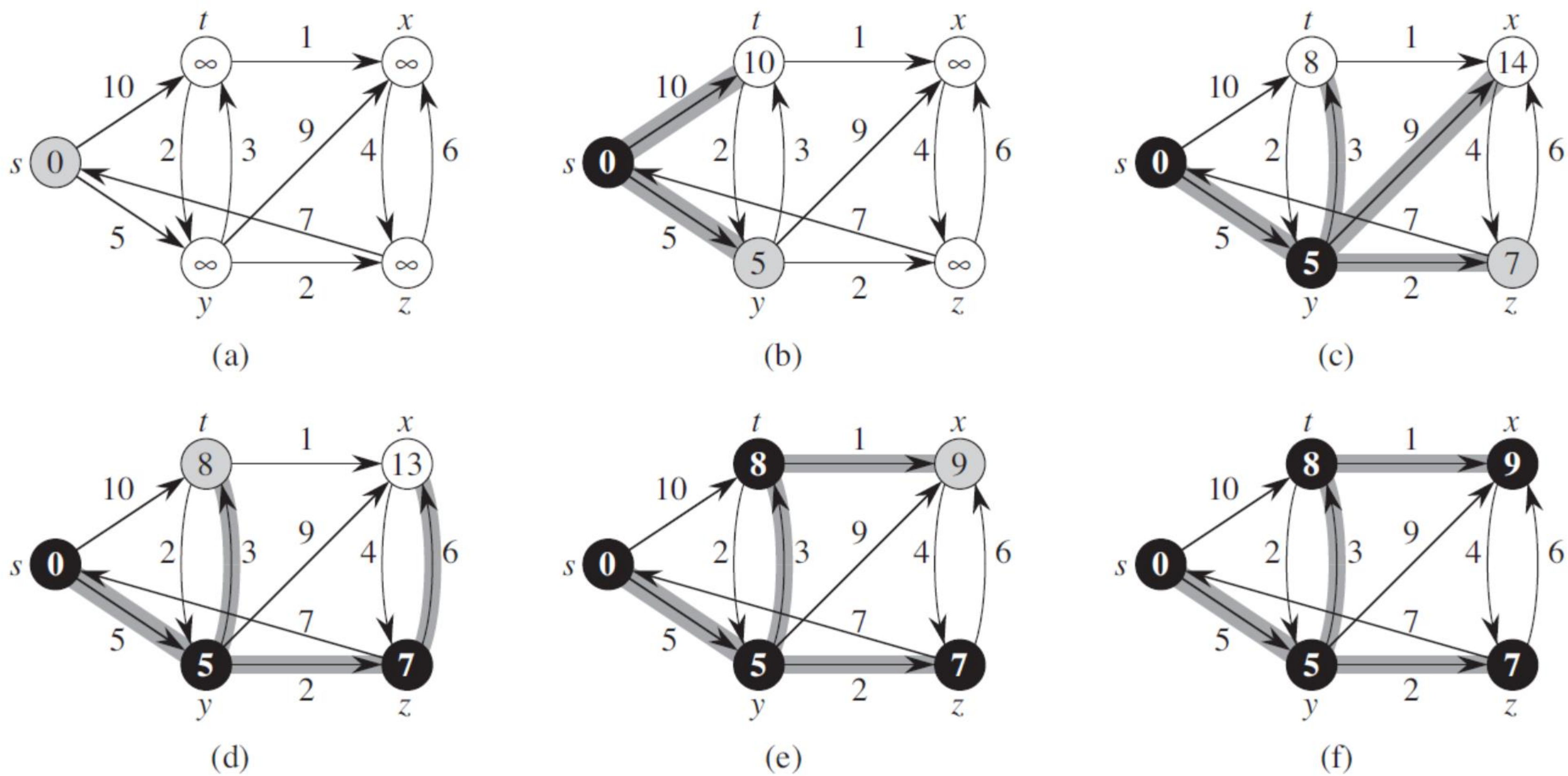


DIJKSTRA(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

➢ When the algorithm adds a vertex **u** to the set **S**, **u.d** is the final shortest-path weight from **s** to **u**.

- **계산 과정 예**



v.d

DIJKSTRA(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S = \emptyset$
3 $Q = G.V$
4 **while** $Q \neq \emptyset$
5 $\quad u = $ EXTRACT-MIN(Q)
6 $\quad S = S \cup \{u\}$
7 $\quad$ **for** each vertex $v \in G.Adj[u]$
8 $\quad\quad$ RELAX(u, v, w)

- **Correctness of Dijkstra's algorithm**

Theorem Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function $w : E \to R$ and source s, terminates with $v.d = \delta(s, v)$ for all vertices $v \in V$.

- **Loop invariant**

At the start of each iteration of the **while** loop of lines 4–8, $v.d = \delta(s, v)$ for each vertex $v \in S$.

- **A key in the proof**

To show that for each vertex $u \in V$, we have $u.d = \delta(s, u)$ at the time when u is added to set S.

Suppose for contradiction that u be the first vertex for which $u.d \neq \delta(s, u)$ when it is added to set S. $\cdots$

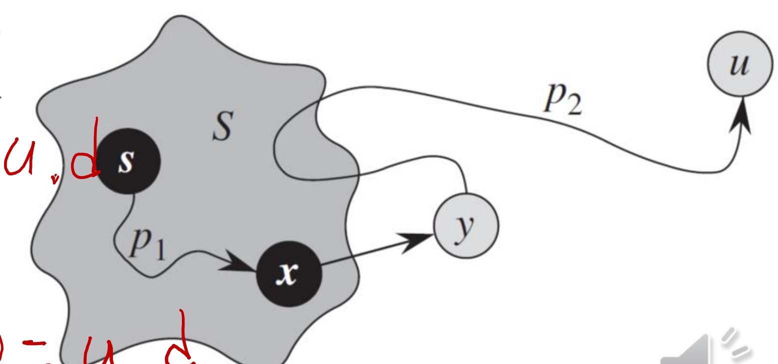
s ---> x ->y가 shortest path이므로,
x가 S에 add 되면서 x->y에 relaxation할 때,
y.d에 δ(s, y)가 저장. 따라서 u가 S에 add가 될 때에도 y.d = δ(s, y).

$u.d \le \delta(s, u) \le u.d$

- $y.d = \delta(s, y)$
- $y.d = \delta(s, y) \le \delta(s, u) \le u.d$
- $u.d \le y.d$

$\Downarrow$

$\delta(s, u) = u.d$



An O(n^2) Implementation : Adjacency Matrix 사용

S: the set of vertices, including v_0, whose shortest paths have been found

distance[w]: the length of the shortest path starting from v_0, going through vertices only in S, and ending in w (w not in S)

n = | V |

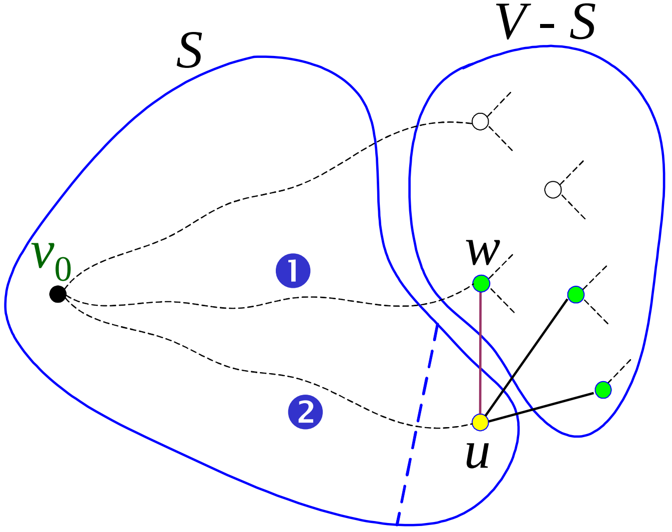
- Observations

  When the shortest paths are generated in nondecreasing order of length,

  – If the next shortest path is to vertex u, then the path from v_0 to u goes through only those vertices that are in S.

  – Vertex u is chosen so that it has the minimum distance *distance*[u] among all the vertices not in S.

  – Adding u to S may change the distance of shortest paths starting at v_0 going through vertices only in the new S, and ending at a vertex w that is not currently in the new S.




```
                       ❶
distance[w] ← min{distance[w],
                  distance[u] + length(<u, w>)}
                       ❷
```

$V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ with $|V| = n$

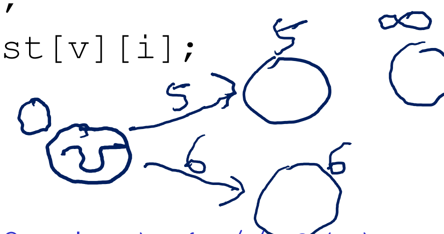
[Horowitz]

```
int choose(int distance[], int n,
                          int found[]) {
  int i, min = INT_MAX,
          minpos = -1;

  for (i = 0; i < n; i++) // O(n)
    if (distance[i] < min
                  && !found[i]) {
      min = distance[i];
      minpos = i;
    }
  return minpos;
}
```

```
void shortestpath(int v,
    int cost[][MAX_VERTICES],
    int distance[], int n, int found[]) {
  int i, u, w, tmp;

  for (i = 0; i < n; i++) {
    found[i] = FALSE;
    distance[i] = cost[v][i];
  }
  found[v] = TRUE;
  distance[v] = 0;
  for (i = 0; i < n-2; i++) { // O(n)
    // find the next u to be added to S
    u = choose(distance, n, found);
    found[u] = TRUE; // add u to S
    for (w = 0; w < n; w++)
      if (!found[w]) // for w not in S
        if ((tmp = distance[u] +
              cost[u][w]) < distance[w])
          distance[w] = tmp;
  }
}
```



`distance[i]` : the length of the SP from vertex `v` to `i`
`found[i]`: **FALSE** if the SP from vertex `i` has not been found,
TRUE otherwise.
`cost[i][j]`: cost adjacency matrix

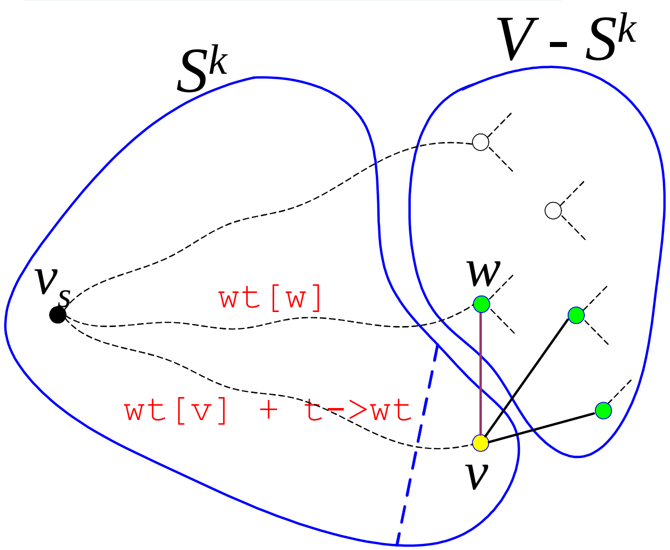
An O(e log n) Implementation: Adjacency List 사용

- 매 순간 `wt[w]`에는 항상 S^k 의 꼭지점들만 사용할 경우에 대한 v_s에서 w까지의 shortest path의 길이가 저장되어 있음.

n = | V |, e = | E |

```c
typedef struct node *link;
struct node { int v; double wt,
              link next; };
struct graph { int V; int E;
               link *adj; };
typedef struct graph *Graph;
```

`wt[]`는 앞의 프로그램에서의 `distance[]`에 해당



v.π v.d

```c
#define GRAPHpfs GRAPHspt
#define P (wt[v] + t->wt)
void GRAPHpfs(Graph G, int s, int st[],
                                   double wt[]) {
    int v, w;  link t;
    PQinit(); priority = wt;
    for (v = 0; v < G->V; v++) {           O(n)
      st[v] = -1; wt[v] = maxWT; PQinsert(v);
    }                              log n
     wt[s] = 0.0; PQdec(s);
     while (!PQempty())
       if (wt[v = PQdelmin()] != maxWT)     log n
         for (t = G->adj[v]; t != NULL; t = t->next)
           if (P < wt[w = t->v]) {
             wt[w] = P; PQdec(w); st[w] = v;
           }                                 log n
}
```

O(e log n)

O(n log n)

* 언제 끝날까?

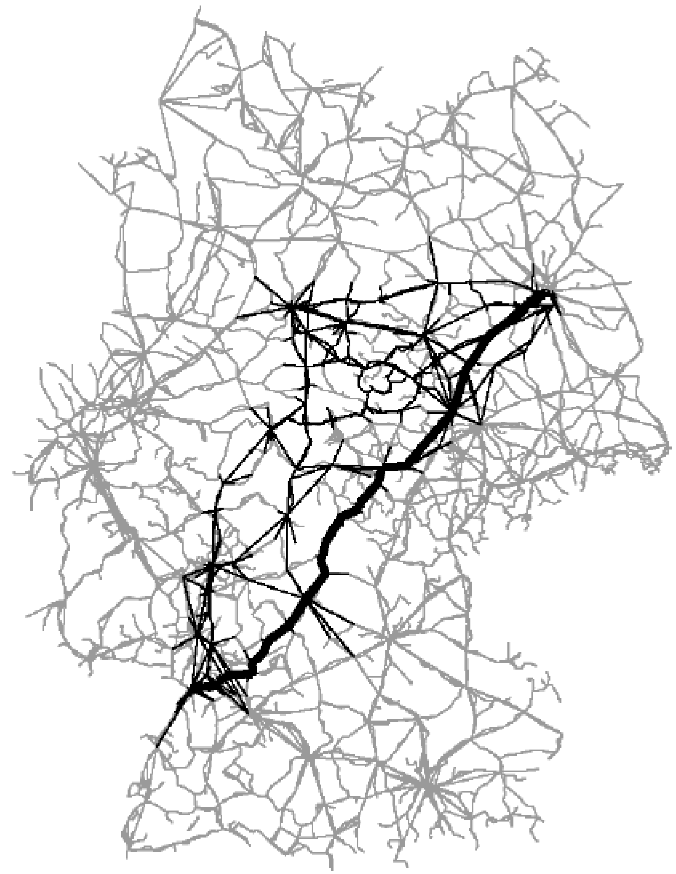
* 어떤 경우인가?

* 다 끝난 후 각 꼭지점에 대한 sp를 어떻게 출력할까?

v v wt

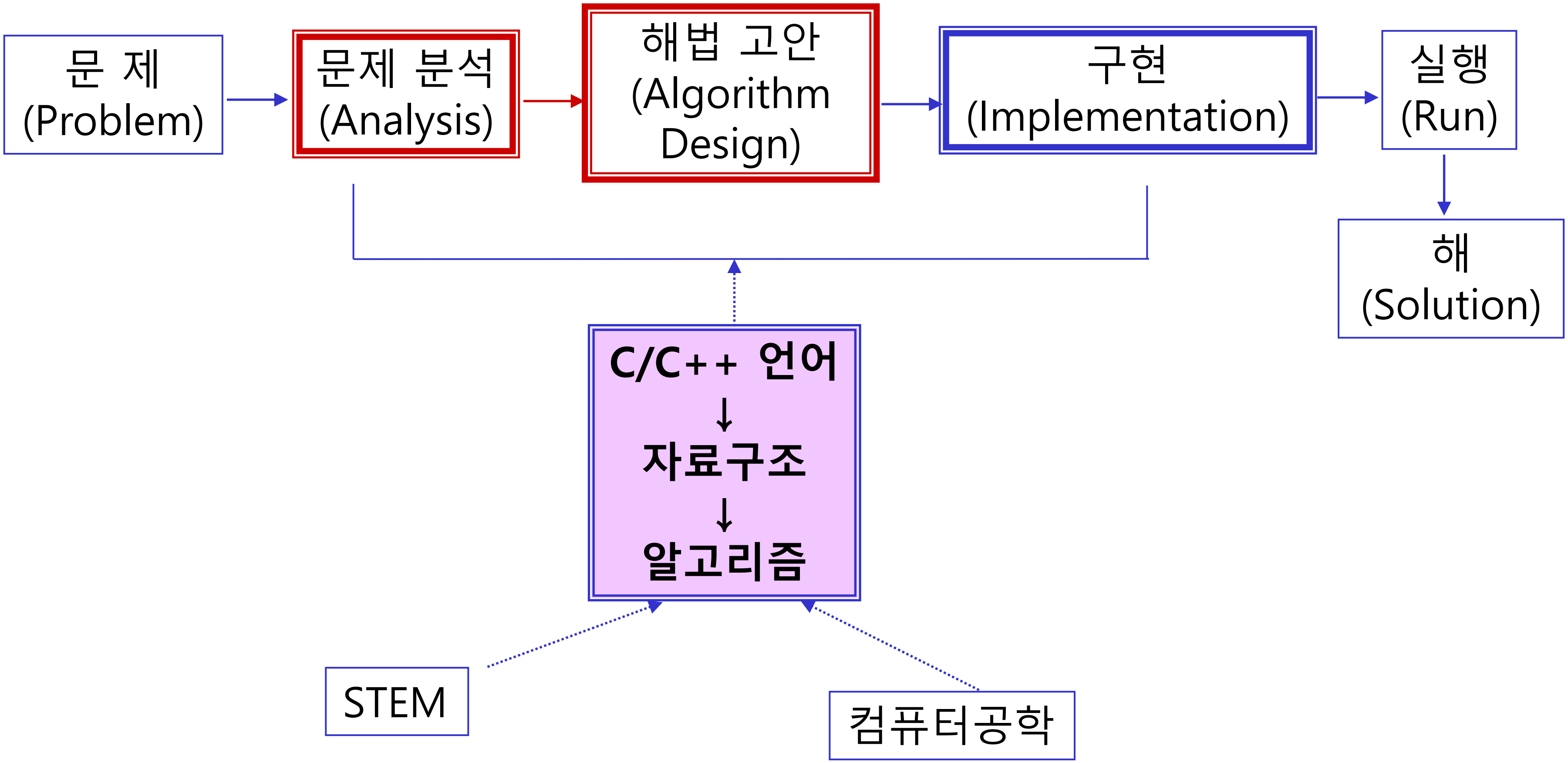
MO¨HRING et at., *Partitioning Graphs to Speedup Dijkstra's Algorithm*, ACM Journal of Experimental Algorithmics, 2006.

Standard Dijkstra's Algorithm



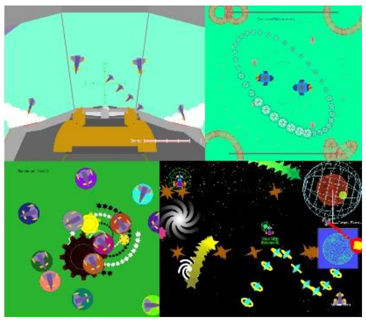
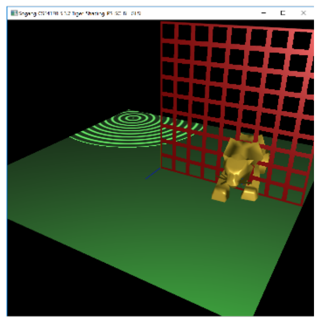
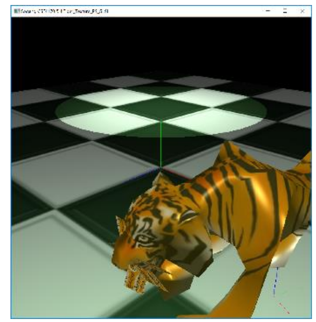
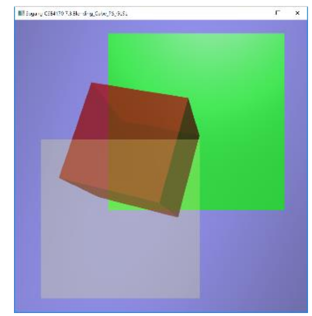
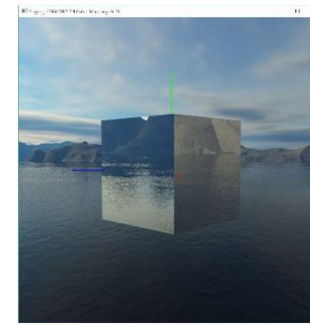
Acceleration Algorithm

강의를 마치며


문 제
(Problem)
문제 분석
(Analysis)
해법 고안
(Algorithm Design)
구현
(Implementation)
실행
(Run)
해
(Solution)
C/C++ 언어
↓
자료구조
↓
알고리즘
STEM
컴퓨터공학

CSE4170 기초 컴퓨터 그래픽스

3D Real-Time Rendering Using OpenGL

Unity Shader Programming

Virtual/Augmented/Mixed Reality