GK-DEVS의 실시간 시각 시뮬레이션을 위한 외곽공간의 관리

황문호*

Bounding Space Management for Real-time Visual Simulation of GK-DEVS

Moon Ho Hwang

Abstract

This paper presents bounding space(BS) management for real-time visual simulation when using GK-DEVS models. Since GK-DEVS, extended from DEVS formalism, has information of 3D geometry and 3D hierarchical structure, we employe three types of bounding spaces: BS of its own shape, BS of its children GK-DEVS, and total BS. In addition to next-event scheduling functionality of previous GK-Simulator, its abstract simulation algorithms is extended to manage the three types of BSs so that BSs can be utilized in the rendering process of a renderer, so called GK-Renderer. We have implemented the method and evaluated it with an automated manufacturing system. In the case study, the proposed BSs management method showed about 2 times improvement in terms of rendering process speed.

Key Words: Real-time Simulation, Rendering, Bounding Space, GK-DEVS, GK-Simulator, GK-Renderer

* (주) 큐빅테크, 큐빅기술연구소, 시뮬레이션 및 제어 부

1. 서론

DEDS(Discrete Event Dynamic System) 이론은 모델링 대상의 사건 발생 시점에 상태가 변하는 특성을 기술하여 인간제작시스템(man-made system)을 표현하는데 적합하다[1][2]. DEDS를 기술하는 여러 방법론 중에서 DEVS(Discrete Event System Specification)는 입력과 출력을 주고받는 시스템적 특성과 부품이 모여 시스템을 구성하는 계층적인 특성을 지니고 있어 여러 구성부품들로 구성된 복잡한 시스템을 대상으로 한 모델링 및 시뮬레이션 분야에서 널리 사용되어왔다[1][2][9].

최근 들어서 DEVS는 공간의 문제에도 관심을 갖게 되었는데 크게 두 가지 접근 방법으로 요약된다. 하나는 공간을 격자(lattice)로 구성하고 격자간의 인접관계를 연결하여 모델링 하는 방법[8]이며 다른 하나는 형상 및 기구의 정보를 계층적으로 정의하는 접근방법이다[5]. 전자의 격자를 이용한 방법은 DEVS 이론과 Cellular Automaton이론을 접목한 것으로 볼 수 있으며 후자의 방법은 DEVS 이론과 Robotics 이론이 접목된 것으로 볼 수 있다.

DEDS 분야와는 별도로 계산 기하학에 기반을 둔 컴퓨터 그래픽스의 분야에서는 3차원 공간상에서 벌어지는 현상을 구조적으로 가시화(Structured Visualization)하려는 노력을 기울여왔다. 구조적 가시화란 가시화를 효율적으로 하기 위하여 공간을 구조화하는 것으로써 여기서도 공간을 격자 형태로 나누는 접근방법과 구성객체를 감싸는 볼륨을 계층적으로 구성하는 Scene Graph [3] 기법으로 크게 나뉜다. 공간을 구조화하게 되면 얻을 수 있는 이득은 보이는 부분과 그렇지 않은 부분을 빨리 판별할 수 있다는 것인데 카메라의 3차원 투영 시에 발생하는 가시 영역 판별(Viewing Frustum Culling)법은 가장 일반적으로 사용되는 방법 중 하나이다.

그러나 이산사건 시뮬레이션 분야와 컴퓨터 그래픽스 분야의 학제적(interdisciplinary) 연구는 진행되지 않고 있어서 이 분야에 연구는 찾아 볼 수 없다.

이러한 의미를 가지고 본 연구에서는 실시간(Real-time) 시각 시뮬레이션 환경을 구축하는데 필요한 GK-DEVS의 시뮬레이션 방법을 소개하고자 한다. 더 정확히 말하면, 시각 시뮬레이션에 가장 많은 계산이 요구되는 가시화 과정을 가시영역 판별법을 이용하여 소요시간을 단축시키고자 한다. 이를 위해서 본 연구에서는 기존 [5]에서 제안된 GK-Simulator의 추상화 시뮬레이션 알고리듬에 '외곽 공간(Bounding Space)'의 계층적 관리를 추가한다. 뿐만 아니라 이 외곽공간을 이용하여 가시영역 판별을 수행하는 GK-Renderer를 소개한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 GK-DEVS 형식론과 기존의 GK-Simulator 알고리즘 개념을 간략히 살펴보고 가시화에서의 영역 판별법의 개념을 소개한다. 제 3장에서는 외곽공간을 정의하고 예를 이용하여 설명한다. 제 4장에서는 계층적 외곽공간을 관리하는 GK-Simulator와 이것을 가시화 하는 GK-Renderer, 그리고 이들을 총괄하는 Root-Coordinator를 소개한다. 제 5장에서는 제안된 방법을 이용한 가시화 시뮬레이션의 수행성 평가 결과를 소개하고, 제 6장에서는 결론 및 추후 연구 방향을 제시한다.

2. 관련 기존 연구

본 절에서는 GK-DEVS 형식론과 이것의 기존 추상화 시뮬레이션 방법론에 대한 간단한 리뷰를 다룬다. 그리고 3차원 가시화 과점에서의 가시영역에 대한 소개를 통해서 가시화 시뮬레이션의 속도 향상을 모색한다.

2.1 형식론 및 추상화 시뮬레이션

2.2.1 GK-DEVS

본 논문에서 사용하는 모델링 형식론은 GK-DEVS이다. GK-DEVS는 [4]에서 소개되었고, [5]에서는 연속상태 변화에 초점이 맞춰 수정되

었다. GK-DEVS의 수학적 정의는 다음과 같다.

$GK\text{-}DEVS = \langle X, S, Y, \delta_{int}, \delta_{ext}, f, \lambda, ta, M, Z, SELECT \rangle$
where

- X: 입력 이벤트 집합;
- Y: 출력 이벤트 집합;
- S: 상태변수 집합, $S = \langle S^{disc}, S^{cont} \rangle$, S^{disc}: 이산 상태 집합, S^{cont}: 연속 상태 집합, 단 $S^{cont} = \langle GK, S^{cont-GK} \rangle$, $GK = \langle G, T \rangle$,> G: 형상집합[1]; $T = \begin{bmatrix} \boldsymbol{R} & | & \boldsymbol{P} \\ 0\ 0\ 0 & | & 1 \end{bmatrix}$; 국부 좌표계(local coordinates)(4x4 동차 변환 행렬), $\boldsymbol{R}$(3x3 matrix): 회전변환 행렬, $\boldsymbol{P}(\in \boldsymbol{R}^3)$: 위치 백터;): $S^{cont-GK}$GK를 제외한 연속 상태 집합;
- $\delta_{int}: S \to S$: 내부 상태천이 함수;
- $\delta_{ext}: Q \times X \to S$: 외부 상태천이 함수, 단, $Q = \{(s, e) \mid s \in S, 0 \le e \le ta(s)\}$, 총체적 상태;
- $f: Q \to S^{cont}$ 변화율 함수: 단 다음의 상태 세그먼트 $\Phi_q: \langle t_1, t_2 \rangle \to Q$를 만족시킨다

$$\Phi_q(t) = (s^{disc}, s^{cont} + \int_{t_1}^{t} f(\Phi_q(t'))\,dt', e + t)$$

여기서 $(s^{disc}, s^{cont}, e) \in Q$ 에 대하여
(1) $\Phi_q(t_1) = (s^{disc}, s^{cont}, e)$,
(2) $d\Phi_q(t)/dt = f(\Phi_q(t))$, $t \in \langle t_1, t_2 \rangle$; 이다.

- $\lambda: S \to Y$, 외부 출력함수; ● $ta: S \to R_0^{\infty}$, 시간 전진함수; ● M: 하위의 GK-DEVS의 집합;
- $Z \subseteq Y^H \times X^H$ 계층적 연결집합;

$Y^H = \bigcup_{m \in M} m.Y^H \cup Y$: 계층적 출력 이벤트 집합, $X^H = \bigcup_{m \in M} m.X^H \cup X$: 계층적 입력 이벤트 집합

- $SELECT$: $2^{M \cup \{self\}} - \{\} \to M \bigcup \{self\}$, 타이 해결함수;

GK-DEVS의 특성에 대한 자세한 설명은 [5]을 참고 바란다.

2.2.2 GK-Simulator

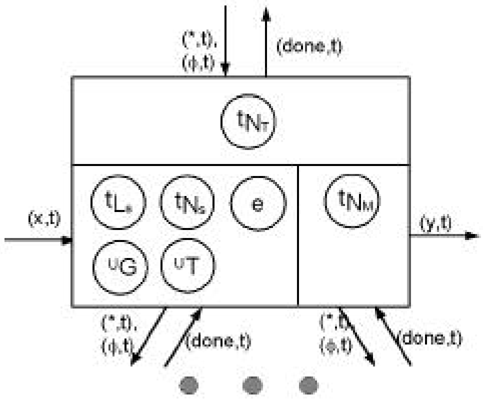
데이터와 콘트롤의 분리 차원에서 GK-Simulator는 GK-DEVS 모델을 데이터로 하는 콘트롤 역할을 담당한다. GK-Simulator에 대한 기존의 연구는 이산사건 시뮬레이션 및 연속상태(연속상태의 입출력은 제외된) 시뮬레이션을 다루고 있다. 다음 사건의 스케쥴을 계층적으로 관리 할 뿐만 아니라 전체 좌표계와 전체 좌표계에서 정의된 형상에 대한 관리도 GK-Simulator에서 담당하고 있다[5].

<그림 1>은 [5]에서 소개된 GK-Simulator의 도식화 한 것이다. 앞서 언급한 것처럼 시간에 관련된 기억변수들이 있는데, 전체 다음 사건 시각(tN_T), 자신의 이전 사건 시각(tL_s), 자신의 다음 사건 시각(tN_s), 자식모델집합 M을 구성하는 자식 중에 가장 빠른 다음 사건 시각(tN_M), 상태 유지 시간(e)이 그것들이다. 특히 전체 다음 사건 시각은 자신의 다음 사건 시각과 자식의 다음 사건 시각 중 빠른 시각, 즉 $tN_T = \min(tN_s, tN_M)$의 관계가 있다[2].

그리고 $^U T$는 전체 좌표계를, $^U G$ 전체 좌표계에서 본 형상을 기억하는 변수이다. <그림 1>에서 3부분으로 나뉘어진 의미는 왼쪽 하단은 GK-DEVS의 상태집합 S와 관련된 부분이고 오른쪽 하단은 자식모델 집합 M과 관련된 부분이고 상단은 이들을 통합한 부분이다.

1) 본 논문에서는 형상집합 G를 "삼각형 망들의 집합"으로 그 의미를 좁혀서 구현하였다.

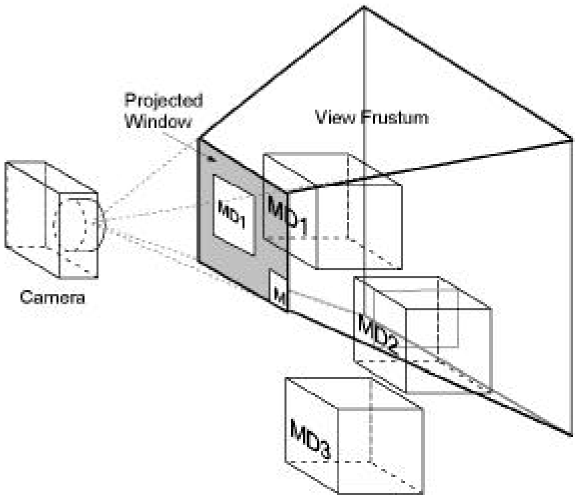
2) [5]에서 tN_{self}, tN_{child}, tN으로 표기했던 것을 본 논문에서는 tN_s, tN_M, tN_T으로 각각 표기하였다.



<그림 1> 기존의 GK-Simulator

2.2.3 가시 영역 판별(Viewing Frustum Culling)

3차원 컴퓨터 그래픽스에서는 관측하는 영역을 2차원 창(Window)에 투영된 영상으로 계산하여 그려야 한다[3]. <그림 2>는 이러한 카메라의 투영을 설명하고 있는데 가시영역은 짤린 피라미드(Frustum) 형상으로 표현되며 피라미드의 정점에 카메라가 위치하고 있다. 짤린 피라미드의 상단에 투영 창(Projected window)이 존재하는데, 가시 영역에 포함되어 있는 객체(MD1)나 걸쳐있는 객체(MD2)는 투영 창에 그려지나 가시영역 밖에 있는 객체(MD3)는 투영 창에 나타나지 않는다.



<그림 2> 가시영역과 투영 창

<그림 2>에서 볼 수 있듯이 가시영역 밖에 있는 객체를 판별할 수 있다면 모든 객체를 가시화 시키는 방법보다 더 효율적인 연산을 수행할 수 있다[3]. 따라서 본 연구에서는 가시영역의 판별을 위해 GK-DEVS의 외곽 공간(Bounding Space) 개념을 도입하고자 한다. 즉, 기존의 GK-Simulator는 다음 사건 시간의 관리를 하고 있었다면, 본 논문에 제안된 GK-Simulator는 가시영역 존재의 판단에 필요한 '외곽공간 관리'기능도 추가적으로 갖는다. 이 포함 영역이 가시영역 밖에 존재한다면 그 내부의 형상(geometry)은 가시화의 대상에서 제외시킨다는 전략이다.

3. GK-DEVS의 외곽공간

본 절에서는 GK-DEVS에 적용된 계층적 외곽공간(hierarchical bounding space)의 개념을 소개한다.

3.1 외곽영역의 정의

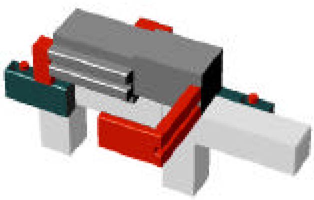
- BS_G : 형상 외곽공간; 어떤 GK-DEVS의 BS_G 는 $\forall g \in G$를 포함하는 공간이다.
- BS_M : 자식 외곽공간; 어떤 GK-DEVS의 BS_M 는 $\forall m \in M$를 포함하는 공간이다.
- BS_T : 전체 외곽공간; 어떤 GK-DEVS의 BS_T $\forall g \in G$와 $\forall m \in M$를 포함하는 공간이다.

이러한 외곽공간을 실용적으로 활용하기 위해서 본 논문에서는 '구(Sphere) 형상의 외곽공간을 사용한다.

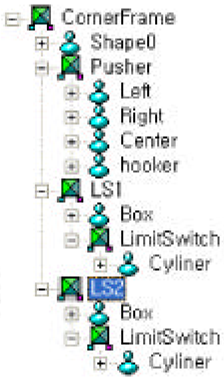
3.2 외곽영역의 예

외곽영역의 개념을 소개하기 위하여 <그림 3>의 방향전환 시스템을 소개하고자 한다. 이 예에서는 방향 전환 시스템은 뼈대를 구성하는 ConnerFrame이 있고, 그 내부에 두 개의 리밋스위치와 왕복운동을 하는 Pusher가 있다. <그림

3.b>에서 은 GK-DEVS를, 은 다면체 집합을 나타낸다. <그림 4>는 방향전환 시스템의 외곽 공간을 부품에서 조립 시스템까지 보여주고 있다.



b. 조립 형상

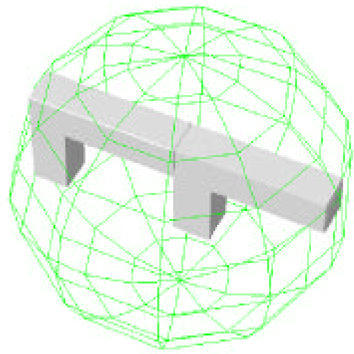


c. 조립 계층도

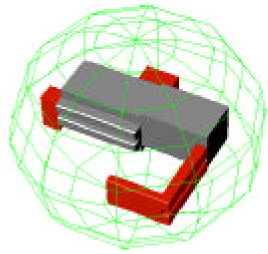
<그림 3> 방향전환 시스템의 예

먼저 <그림 4.a>는 조립되기 전의 형상만을 가지고 있는 ConnerFrame을 보여주고 있다. 아직 자식모델을 조립하기 이전의 상태여서 자식모델집합이 없다($M = \varnothing$). 따라서 자식모델을 포함하는 공간의 부피도 없다($BS_M = S_\varnothing$). 뿐만 아니라 형상 외곽공간과 전체 외곽공간이 동일한 $BS_G = BS_T$ 관계도 성립한다. 이와 유사하게, <그림 4.b>의 Pusher의 경우도 마찬가지이다. 형상 (G)을 구성하는 4개의 Polygon 집합만이 있을 뿐 자식 GK-DEVS 집합의 비어 있다. .

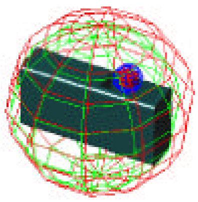
그러나 <그림 4.c>는 두 개의 GK-DEVS를 조립한 경우를 보여준다. 먼저 실린더 형상을 갖고 자식은 없는 GK-DEVS인 LimitSwitch가 있고, 이것을 자식으로 갖으면서 Box형상을 갖는 GK-DEVS LS1이 있다. <그림 4.c>는 조립된 LS1을 보여주는데 아주 작은 구는 LS1의 BS_M을 보여주고 있으며 이것은 LimitSwitch의 BS_T (=LimitSwith의 BS_G)과 같은 것이다. 또 중간크기의 구는 LS1의 BS_G를 보여주고 있으며 그 두 구를 감싸는 가장 외곽의 구는 LS1의 BS_T를 보여준다.



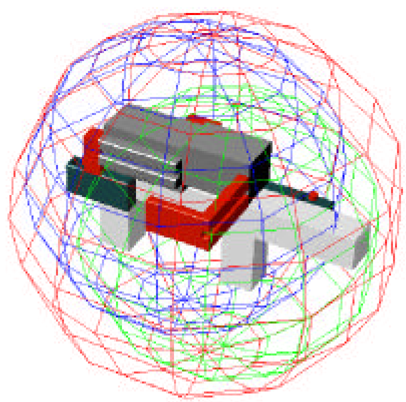
a. ConnerFrame 형상 및 외곽공간



b. Pusher 형상 및 외곽공간



c. 조립 리미트스위치 박스 형상 및 외곽공간



d. 조립된 시스템의 외곽공간

<그림 4> 방향전환 시스템의 외곽공간

<그림 4.c>와 마찬가지 <그림 4.d>은 ConnerFrame에 Pusher, LS1, LS2를 조립한 예에 대하여 BS_G, BS_M, BS_T를 보여주고 있다. BS_G는 <그림 4.a>의 것과 동일하며 BS_M은 Pusher, LS1, LS2를 감싸는 구이고 BS_T는 가장 외곽의 구이다.

4. 외곽공간의 관리 및 활용

본 장에서는 앞장에서 설명한 외곽공간을 관리하는 GK-Simulator와 이를 활용하여 가시화 작업을 수행하는 GK-Renderer, 그리고 이들을 총괄하는 Root-Coordinator를 소개한다.

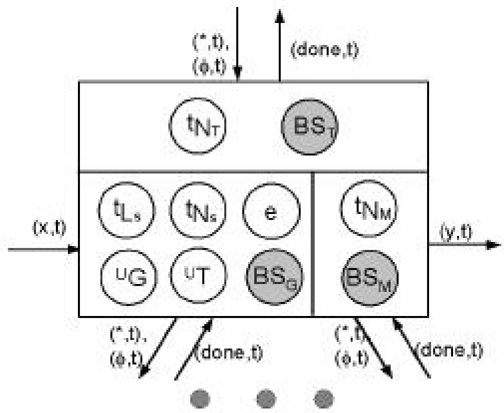
본 논문에서는 외곽공간의 계층구조를 모델의 구조와 일치시켜 시뮬레이터에서 관리하도록 하였다. 이와는 달리 이진트리 구조를 이용하는 연구도 있는데, [6]는 외곽공간으로 구를 이용하였고, [7]은 방향성을 갖는 박스(oriented bounding box:OBB)를 이용한 연구이니 관심 있는 독자는 참조 바란다.

그리고 본 논문에서 사용한 삼각형 집합으로 구성된 형상정보를 입력받아 이를 포함하는 외곽공간 구를 생성하는 방법은 부록의 Algorithm A1를 참고 바란다. 또 구의 외곽공간 집합을 입력받아 이들을 포함하는 외곽공간을 구하는 방법은 부록의 Algorithm A2와 A3에서 소개하고 있으니 참고 바란다.

4.1 GK-Simulator

4.1.1 GK-Simulator의 구조

형상을 포함하는 외곽공간 BSG와 자식들을 포함하는 외곽공간 BSM, 그리고 이들 두 공간 BSG와 BSM을 포함하는 외곽공간 BST들의 도입이 <그림 5>와 같이 추가되었다(<그림 5>에서는 회색으로 표시). 이들 외곽공간은 자신의 상태(정확히 말하면 형상의 상태)가 변했을 때, 또는 자식의 상태가 변했을 때 갱신되어진다.



<그림 5> 외곽영역을 관리하는 GK-Simulator의 개념

시뮬레이터간의 이동하는 메시지의 종류는 [5]의 것을 유지하였다. 즉, 다음 사건시각의 도달하였음을 알리는 (*,t), 외부로부터의 입력을 알리는 (x,t), 외부로의 출력을 알리는 (y,t), 스케쥴의 갱신을 요구하는 (done, t), 연속상태변화를 알리는 (∅,t)가 그것들이다.

4.1.2 GK-Simulator의 추상 알고리즘

기존 GK-Simulator의 추상 시뮬레이션 알고리듬이 다음 이산사건발생 시각관리를 주 역할로 하였다면, 본 연구에서는 외곽공간의 계층적 관리 기능이 추가된다. Algorithm에서 새로 추가된 외곽공간과 관련된 부분은 회색 영역으로 표시하였다. 그런데 이들 BSG, BSM, BST의 외곽영역을 구하는 방법은 외곽영역을 표현하는 방법에 따라서 다양할 수 있는데 본 논문에서는 구(Sphere)를 사용하는 알고리듬을 부록에서 소개하고 있으니 참고 바란다.

Algorithm1은 계획된 이산사건 발생 시각이 도래할 때 전달되는 메시지의 처리 과정을 보여주고 있다. 크게 어떤 GK-DEVS 자신의 상태변화로 처리되는 부분(2~10째줄)과 하부의 자식 모델의 변화로 처리되는 부분(12~16째줄)으로 나뉜다. 자신이 변화하였을 때에는 자신 형상의 외곽공간 BSG을 갱신하며(10째줄) 자식의 변화가 있었을 때에는 자식 외곽공간 BSM을 갱신한다. 마

지막으로 전체 외곽공간 BST를 갱신한다. 나머지의 알고리듬 부분은 기존 GK-Simulator와 동일하므로 [5]을 참조 바란다.

```
Procedure GK-Simulator::when_receive_(*,t)
1  if t == tN_T then
2    if tN_s < t_M or (tN_s == tN_M
          and this is selected by SELECT ) then
3      y := λ(s);
4      send (y,t) to influencee simulators;
5      s := δ_int(s);
6      tL_s := t;
7      tN_s := tL_s + ta(s);
8      ^uT := pareant's ^uT * T;
9      ^uG := G * ^uT;
10     BS_G := GetBS(^uG);
11   else
12     find the imminent child simulators;
13     select one,i*, and send the (*,t)to it;
14     resort children by their tN_T ;
15     tN_M := minimum of children's tN_T;
16     BS_M := GetBS(M);
17   end if
18   tN_T := MIN(tN_s,tN_M);
19   BS_T := GetBS(BS_G, BS_M);
20 else
21   ERROR;
22 end if
```

Algorithm 1. GK-Simulator Procedure for (*,t)

Algorithm 2는 외부로부터 전달된 이벤트의 처리 과정을 보여주고 있다. 여기서도 추가된 부분은 자신의 형상 변화에 따른 형상 외곽공간 BS_G의 갱신과 전체 외곽공간 BS_T의 갱신부분이다(9, 10째줄).

```
Procedure GK-Simulator::when_receive_(x,t)
1  if tL_s ≤ t ≤ tN_T then
2    e := t-tL;
3    s := δ_ext(s,e,x);
4    tL_s := t;
5    tN_s := tL_s + ta(s);
6    tN_T := MIN(tN_s,tN_M);
7    ^uT := pareant's ^uT * T;
8    ^uG := G * ^uT;
9    BS_G := GetBS(^uG);
10   BS_T := GetBS(BS_G, BS_M);
11   send (done, tN) to parent simulator;
12 else
13   ERROR;
14 end if
```

Algorithm 2. GK-Simulator Procedure for (x,t)

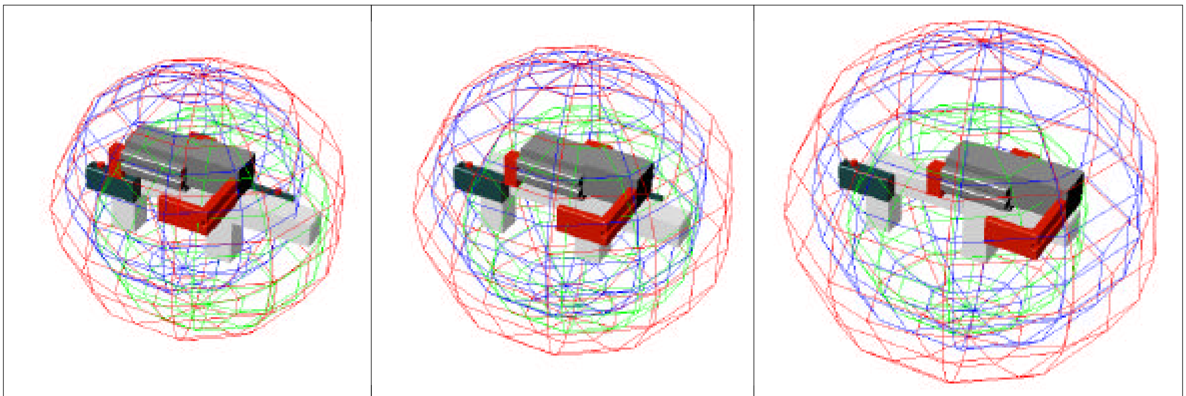
Algorithm 3은 자식으로부터 부모로 전달되는 갱신을 요구하는 메시지가 왔을 때 처리하는 과정을 보이고 있다. 자식의 변화가 있었으므로 자식을 감싸는 외곽공간 BS_M의 갱신과 전체 외곽공간 BS_T의 갱신이 추가되었다(5째, 6째줄).

```
Procedure GK-Simulator::when_receive_(done,t)
1  if tL_s ≤t then
2    resort children by their tN_T ;
3    tN_M := minimum of children's tN_T;
4    tN_T := MIN(tN_s,tN_M);
5    BS_M := GetBS(M);
6    BS_T := GetBS(BS_G, BS_M);
7    send (done, tN) to parent simulator;
8  else
9    ERROR;
10 end if
```

Algorithm 3. GK-Simulator Procedure for (done,t,)

Algorithm 4는 연속적인 상태변화를 알리는 메시지가 왔을 때 처리하는 과정을 보이고 있다. 여기서도 자신의 변화에 따른 형상 외곽공간의 BS_G의 갱신, 자식의 변화에 따른 자식 외곽공간 BS_M의 갱신 및 전체 외곽공간 BS_T의 갱신이 추가되었다(6, 8, 9째줄).

```
Procedure GK-Simulator::when_receive_(∅,t)
1  if tL_s ≤ t ≤ tN then
2    e := t - tL_s;
```



a. Pusher 위치=5.8에서의 외곽공간	b. Pusher 위치=22.5에서의 외곽공간	c. Pusher 위치=45.0에서의 외곽공간

<그림 6> 동적 외곽공간의 관리 예

$s^{cont} := s^{cont}(tL_s) + \int_{tL_s}^{t} f(\Phi(t')/dt';$

^{u}T := pareant's $^{u}T * T$;

$^{u}G := G * {}^{u}T$;

BS_G := GetBS(^{u}G);

$\forall$m in child simulators,send ($\varnothing$,t) to m;

BS_M := GetBS(M);

BS_T := GetBS(BS_G, BS_M);

else

ERROR;

end if

Algorithm 4. GK-Simulator Procedure for ($\varnothing$,t)

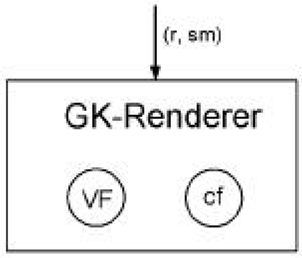
4.1.3 외관공간 관리 예

앞서 <그림 4>에서 소개한 예에서 Pusher는 직선 왕복운동을 하는 요소이다. 따라서 Pusher가 왕복운동을 하는 동안에 이와 관련한 외곽공간은 계속 갱신된다. <그림 6>은 Pusher의 이동에 따른 외곽공간의 변화를 보여주고 있다. 이 예에서는 Pusher의 이동에 따라서 ConnerFrame의 BS_M가 갱신되며 이에 따라서 BS_T도 갱신되고 있다.

4.2 GK-Renderer

4.2.1 GK-Renderer의 구조

GK-Renderer는 본 논문에서 새로이 소개돼는 부분인데 <그림 7>은 GK-Renderer의 내부 기억변수와 외부 메시지를 개념적으로 보이고 있다. 내부에는 <그림 2>에서 설명한 카메라의 가시 공간(Viewing Frustum: VF)과 가시영역이 대상을 어떻게 포함하고 있는지에 대한 상태 (Containment Flag:cf $\in$ {CONTAIN, OVERLAP, DISJOINT })를 기억한다. 메시지의 종류는 렌더링 메시지 (r, sm) 하나뿐이며 여기서 sm은 그릴 대상 GK-Simulator를 말한다.



<그림 7 > GK-Renderer의 개념

4.2.2 GK-Renderer의 알고리듬

가시영역이 외곽영역을 포함되는 지에 대한 검

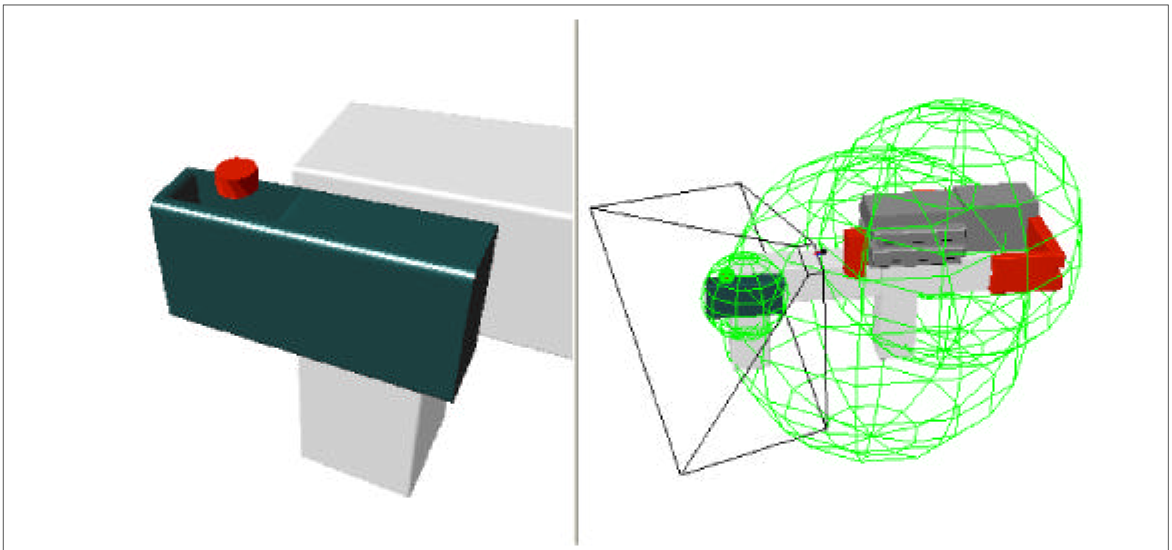
사방법은 [3]를 참고바라며, Algorithm 5는 GK-Simulator를 대상으로 한 렌더링 과정을 보여주고 있다. 여기서 주목 할 것은 본 렌더링 과정은 제귀적(recursive) 과정이라는 것과, 대상 GK-Simulator의 전체 외곽볼륨 BS_T가 CONTAIN이라면 그 이하 BS_G와 BS_M도 가시영역에 포함되므로, 더 이상의 포함 테스트는 할 필요가 없이 Rendering을 한다는 것이다(1~4번째 줄). 만약 BS_T가 가시영역에 걸쳐있는 OVERLAP의 경우에는(5~20번째 줄), 하위의 외곽영역 BS_G와 BS_M을 계속 내려가면서 포함관계 테스트를 수행하며, 그 결과가 가시영역 밖에 있는 DISJOINT의 경우에는 Rendering을 하지 않는다.

```
Procedure GK-Renderer::when_receive_(r,sm)
 1 if cf == CONTAIN then
 2    ∀ g in sm.G, render g;
 3    ∀ sm in children simulators,
 4       when_receive_(r,sm);
 5 else if cf == OVERLAP then
 6    cf := contain(VF, BS_G);
 7    if cf != DISJOINT then
 8       ∀ g in sm.G, render g;
 9    endif
10    cf := contain(VF, BS_M);
11    if cf == CONTAIN then
12       ∀ sm in children simulators,
13          when_receive_(r,sm);
14    else if cf == OVERLAP then
15       ∀ sm in children simulators,
16       begin
18          cf := contain(VF, sm.BS_T);
19          if cf != DISJOINT then
20             when_receive_(r,sm);
21          end if
22       end
23    end if
24 end if
```

Algorithm 5. GK-Renderer Procedure for (r,sm)

4.2.2 외곽공간을 이용한 Rendering 예

<그림 8>은 외곽공간을 이용하여 가시영역 판별법을 적용한 예를 보여준다. <그림 8.a>는



a. LS1 부위 관찰 예 b. 가시영역 및 BS_G

<그림 8> 외곽공간을 이용한 가시영역 판별 예

<그림 6.c>의 상태에 있는 방향 전환 설비의 LS1 부위를 관찰하는 화면이다. <그림 8.a>를 관찰하는 카메라의 가시 영역을 <그림 8.b>는 보여 주고 있다. <그림 8.b>의 구들은 방향전환 설비의 모든 BS_G도 보여주고 있는데 LS1.BS_G와 ConnerFrame.BS_G 가시영역 내에 포함됨을 알 수 있다. 따라서 이 예에서는

contain(VF, ConnerFrame.BS_T)
= OVERLAP,
contain(VF, ConnerFrame.BS_G)
=OVERLAP,
contain(VF, ConnerFrame.Pusher.BS_T)
= DISJOINT,
contain(VF, ConnerFrame.LS1.BS_T)
=CONTAIN,
contain(VF, ConnerFrame.LS2.BS_T)
= DISJOINT
의 경우이다.

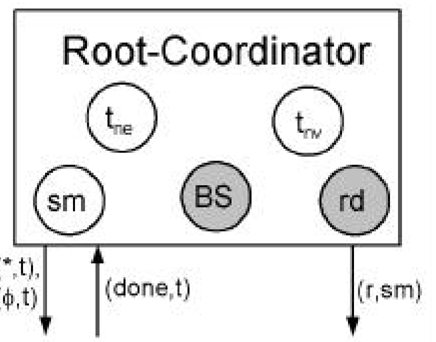
4.3 Root-Coordinator

4.3.1 Root-Coordinator의 구조

[5]에서 소개된 Root-Coordinator의 이산사건 및 연속상태 시뮬레이션의 기능에 외곽공간 BS의 정보와 렌더러 rd의 정보가 <그림 9>과 같이 추가되었다. 기존 알고리듬과 동일하게 이산사건 시간의 도래 시에 (*,t)를, 연속상태의 변화시에 (∅,t)를 자식 GK-Simulator인 sm으로 전달하고 이것으로부터 이산사건의 갱신요구 메시지인 (done,t)를 받는다. 여기에 더해진 것은 가시화가 필요한 시점에 GK-Renderer인 rd를 이용하여 가시화 작업을 수행한다는 것이다.

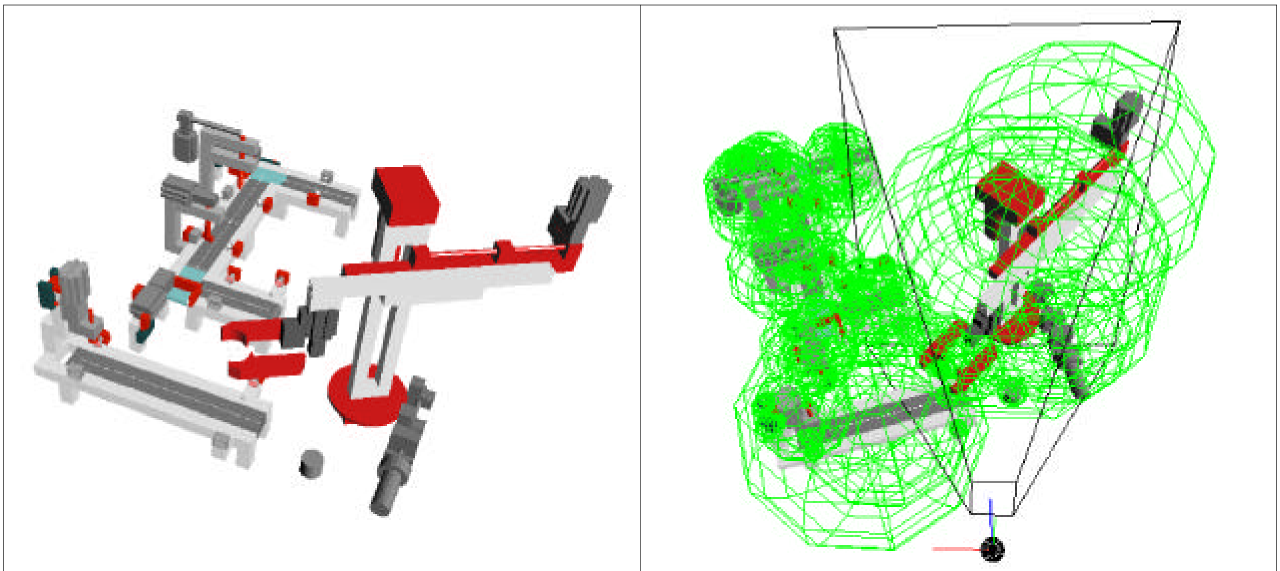
4.3.2 RootCoordinator의 추상 알고리즘

기존에 소개된 RootCoordinator의 이산사건 및 연속상태 시뮬레이션의 기능을 위한 시간 관리의 기능 이외에 전체 외곽공간의 정보 BS가 관리되며 (*,t) 전달 이후에(Algorithm 6.a의 13째줄) 또는 자식의 변화를 알리는 (done, t)의 전달 시(Algorithm 6.b의 3번째 줄)에 BS의 갱신이 이뤄진다. Root-Coordinator는 가시화의 시점마다 렌더러의 가시영역과 전체 외곽공간의 포함테스트를 하여 전체 외과공간이 가시영역의 밖에 존재하는 DISJOINT의 경우가 아닌 경우에 GK-Renderer rd에게 rendering을 요구하는 메지시(r, sm)을 전달하는데 여기서 sm은 Root-Coordinator에 붙어 있는 자식 GK-Simulator를 말한다.



<그림 9> Root-Coordinator의 개념

```
Procedure Root-Coordinator::run(TimeType tf,
                    TimeType tvi, TimeType tci)
 1 TimeType tne := its child simulator's tN;
 2 TimeType tnv := tvi;
 3 TimeType t := tci;
 4 while (MIN(tne, tnv) < tf) begin
 5    while( t < MIN(tne, tnv) ) begin
 6      send (∅, t) to sm;
 7      t = t + tci;
 8    end_of_while
 9    send (∅,MIN(tne, tnv)) to sm;
10    if tne ≤ tnv then
11      send (*, tne) to sm;
12      tne := sm.tN;
13      BS := sm.BS_T;
14    else
15      rd.cf= contain(rd.VF, BS);
16      if rd.cf != DISJOINT then
17        send (r, sm) to rd
18      end if
```



a. 자동화 제조시스템의 3차원 배치도

b. BS_G 와 로봇관측시의 가시영역의 표시

<그림 10> 외곽공간의 가시화 판변별을 이용한 자동화 제조시스템의 가시화 시뮬레이션

```
        tnv := tnv + tvi;
    end if
end_of_while
```

(a) Main Loop Procedure

```
Procedure Root-Coordinator::when_receive_(done,t)
if t < INFINITE then
    tne := t;
    BS := sm.BS_T;
else
    PASSIVE
```

(b) Procedure for a done message

Algorithm 6. The Root-Coordinator Procedures

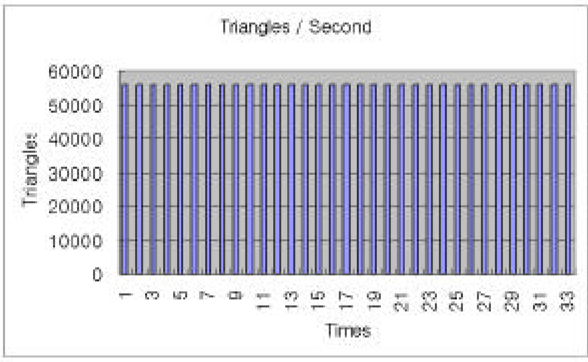
5. 구현 및 실험

본 연구에서는 구현을 위해 C++ 언어를, 컴파일러로 Microsoft사의 VC++을 사용하였다. 그리고 하드웨어로는 Toshiba Satellite 3000 노트북 컴퓨터를 사용하였다. 이 노트북 컴퓨터는 1.0 GHz CPU, 256 MByte RAM, 16 MByte의 NVIDIA GForce2 Graphic Card를 사용하고 있다.

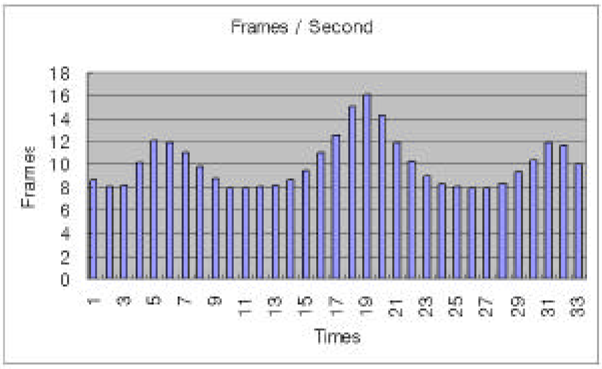
테스트 하고자 하는 대상은 <그림 10.a>의 자동화 제조시스템이다. 이 제조시스템은 5대의 컨베이어, 2대의 방향전환 설비(제 3장에서 소개하였던), 3개의 가공설비, 그리고 작업장간의 물류이송용 로봇이 1대 사용된다. <그림 10.b>는 외곽공간들 중 BS_G를 추가적으로 표시하였고 로봇쪽을 볼 때의 가시영역도 표시하였다. 본 장에서 다루고자 하는 실험내용은 <그림 10.b>와 같이 둘러보기(Navigation) 작업을 할 때에 수행속도를 조사하는 것이다.

실험에서 총 104개의 GK-DEVS Model이 사용되었다. 이들 104개의 모델 형상에 사용된 3차원 삼각형의 개수는 55842개에 이른다. 먼전 계층적 외곽공간관리(HBSM)를 이용한 가시영역 판별법(VFC)을 사용하지 않는 경우엔 한 화면 당 55842개의 삼각형 모두가 그래픽 파이프라이에서 <그림 11.a>과 같이 입력되고 있다. 이때의 렌더링 속도는 <그림 11.b>와 같이 초당 최소 7.9 프레임, 최대 16.0 프레임, 평균 10.1 프레임을 보였다.[3]

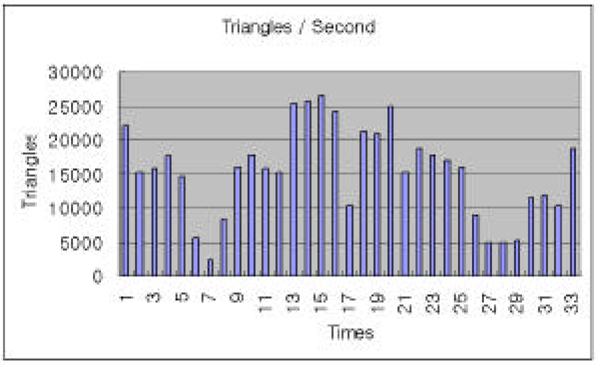
3) 여기서, 동일한 개수의 삼각형들이 랜더링 파이프라인에서 처리됨에도 불구하고 그리는 속도에 편차가 있는 이유는 View Frustum Culling과 무



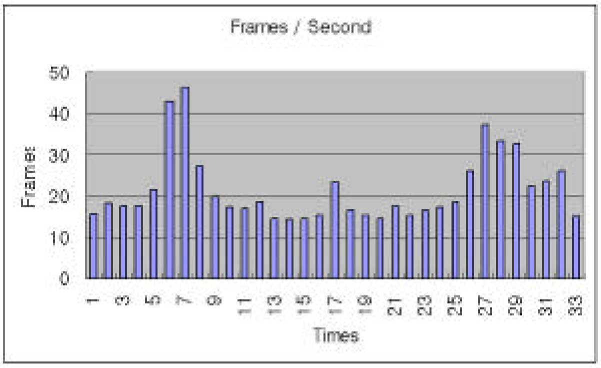
a. Trangles/Fame without VFC using HBSM



b. Fames/Second without VFC using HBSM



c. Trangles/Frame withVFC using HBSM



d. Fames/Second with VFC using HBSM

<그림 11> 자동화 제조시스템의 가시화 작업 수행성

계층적 외곽공간관리를 이용한 가시영역 판별법을 사용할 경우에는 시점에 따라 렌더링 파이프라인에 입력되는 삼각형의 개수가 달라진다. <그림 11.c>와 같이 가시영역 밖에 존재하는 삼각형들은 제외한 최종적으로 한 화면 당 렌더링 파이프라인에 입력된 삼각형의 개수는, 적게는 2414개, 많게는 26479개, 평균적으로 15365개에 달하고 있음을 알 수 있다. 이는 <그림 11.d>에서 볼 수 있듯이 초당 최소 15 프레임, 최대 46 프레임, 평균 21 프레임의 화면갱신의 수행속도를 보였으며 결과적으로 수행속도의 평균값을 가시영역 판별법을 사용하지 않는 경우와 비교할 때 약 2배의 향상을 보였다.

6. 결론 및 추후 연구

본 연구에서는 GK-DEVS의 실시간 시각 시뮬레이션을 위하여 GK-DEVS의 독특한 정보를 처리하는 외곽공간의 방법에 대해서 다루었다. 외곽공간은 크게 자신의 형상을 포함하는 것, 자식들을 형상을 포함하는 것, 이 둘을 포함하는 것으로 구성하였다. 이 외곽공간들은 추상화 시뮬레이션 객체인 GK-Simulator에 의해서 시뮬레이션 중에 관리된다. 가시화를 담당하는 객체 GK-Renderer도 본 연구에서 소개되었는데 GK

관하게 2차원 투영 이후에 Pixel에 점을 찍는 Rasterizing의 개수가 Navigation 시점에 따라 달라지기 때문이다[3].

-Renderer는 GK-Simulator의 계층구조를 따라 내려가면서 가시영역 판별을 이용하였다. 즉, 가시영역에 포함되는 않는 형상은 렌더링 과정에 포함시키지 않고 생략하여 가시화 속도를 향상 시키는 전략을 이용하였다. 이 방법을 사용하여 자동화 생산시스템의 가시화 시뮬레이션에서 Navigation시의 렌더링 수행속도를 실험하였는데, 평균적으로 약 2배의 속도 향상을 볼 수 있었다.

이 외곽공간의 관리와 관련하여서는 몇가지 추후 연구가 필요하다. 먼저 외곽공간의 관리가 추상화 시뮬레이션 알고리듬에 포함됨으로 발생하는 ‘비용(Cost)'에 관한 연구를 들 수 있다. 또 다양한 외곽공간의 적용이 가능 한데, 구 형태뿐만 아니라, 축평형외곽박스(Axis Aligned Bounding Box), 방향을 갖는 외곽박스(Oriented Bounding Box)등을 들 수 있다. 또 외곽공간을 갱신하는 알고리듬에 관한 여러 가지 대안들의 연구도 앞으로 진행될 수 있는 연구항목으로 보인다. 그리고 렌더링 작업뿐만 아니라 공간상의 충돌검색(Collision Detection)에 기반한 이산사건 시뮬레이션의 응용도 기대된다.

부록: 구를 이용한 외곽공간의 계산

Algorithm A1은 형상정보인 다면체 집합을 입력받아 이것을 감싸는 구를 구하는 알고리듬이다. 이는 BSG 구하는데 사용된다.

```
Sphere GetBS(Set<Polygon> G)
 Sphere BS((0,0,0), 0);
 ∀ g in polygons set G
   ∀ v in vertexes set of g
      BS.cp = BS.cp + v;
 BS.cp = BS.cp / no_of_vertexes_of(G);
 ∀ g in polygon set G,
   ∀ v in vertexes set of g,
      cp.r = MAX( cp.r, |v-BS.cp|);
 return BS;
```

Algorithm A1. BS of Set<Pologon> G

Algorithm A2는 GKDEVS 모델 집합을 입력받아 이것을 감싸는 구를 구하는 알고리듬이다. 이는 BSM를 구하는데 사용된다.

```
Sphere GetBS(Set<GKDEVS> M)
 Sphere BS((0,0,0), 0);
 Real radius_sum = 0;
 ∀ s, s is simulator of m in M,
 begin
   BS.cp = BS.cp + s.BS_T.cp;
   radius_sum = radius_sum + s.BS_T.r;
 end
 if ( radius_sum > 0 )
   BS.cp = BS.cp / radius_sum;
 ∀ s, s is simulator of m in M,
   BS.r = MAX(BS.r, |BS.cp-s.BS_T.cp|+ s.BS_T.r);
 return BS;
```

Algorithm A2. BS of Set<GKDEVS> M

Algorithm A3은 두 개의 구를 입력받아 이를 감싸는 구를 구하는 알고리듬이다. 이 알고리듬은 BS_G와 BS_M를 감싸는 외곽공간 BS_T을 구하는데 사용된다.

```
Sphere GetBS(Sphere A, Sphere B)
 Sphere BS((0,0,0), 0);
 Real radius_sum = 0;
 ∀ bs in {A, B},
 begin
   BS.cp = BS.cp + bs.cp;
   radius_sum = radius_sum + bs.r;
 end
 if ( radius_sum > 0 )
   BS.cp = BS.cp / radius_sum;
 ∀ bs in {A, B},
   BS.r = MAX(BS.r, |BS.cp-bs.cp|+ bs.r);
 return BS;
```

Algorithm A3. BS of Sphere A and B

감사의 글

본 논문에서 사용된 3차원 뷰를 구현한 (주) 큐빅테크의 이상욱 연구원과, 3차원 예제의 형상을 모델링 한 (주) 큐빅테크의 원재윤 연구원과

백동석 연구원에게 감사의 말씀을 드립니다.

참고문헌

[1] Yu-Chi Ho, "Scanning the Issue," *Proceedings of the IEEE*, V77, pp. 3-6, 1989
[2] Yu-Chi Ho, "Forward to the Special Issue," *Discrete Event Dynamic Systems: Theory and Applications*, V3, pp. 111, 1993
[3] Tomas Moller and Eric Haines, *Real-Time Rendering*, A K Peters, Ltd, 1999
[4] 황문호, 천상욱, 최병규, "GK-DEVS: 3차원 인간제작 시스템의 시뮬레이션을 위한 형상 기구학 DEVS," *한국시뮬레이션학회 논문집*, VOL. 9, No.1, 3월, pp39-54, 2000년
[5] M.H. Hwang and B.K. Choi, "GK-DEVS: Geometric and Kinematic DEVS Formalism for Simulation Modeling of 3 Dimensional Multi-Component Systems," *Transactions of SCS*, V18, N3, p159-173, 2001
[6] S.M. Omohundro, "Five Balltree Construction Algorithms," *Technical Report #89-063*, International Computer Science Institute, November 20, 1989. http://www.icsi.berkeley.edu/techreports/
[7] S. Gottschalk and M.C. Lin and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection," *SIGGRAPH 96*, p171-180, 1996
[8] G. A. Wainer, N. Giambiasi, "Application of the Cell-DEVS paradigm for cell spaces modelling and simulation," *Simulation*, V76, N1, pp22-39, 2001
[9] B.P. Zeigler, H. Praehofer, T.G. Kim, *Theory of Modeling and Simulation*, 2nd Edition, Academic Press, 2000

● 저자소개 ●

황문호

1990 홍익대학교 공과대학 산업공학과 학사
1992 한국과학기술원(KAIST) 산업공학과 석사
1999 한국과학기술원(KAIST) 산업공학과 박사
1998 2001 (주) 큐빅테크, 시뮬레이션 팀 팀장
2002 현재 (주) 큐빅테크, 시뮬레이션 및 제어 부장
관심분야: 이산사건 시스템 모델링 및 시뮬레이션, 실시간 렌더링, 제조시스템, 가상현실