Vol. 41 No. SIG 7(TOM 3) 情報処理学会論文誌：数理モデル化と応用 Nov. 2000

# プログラムスライシングのVRMLへの導入とその改良

丸 山 博 史[†] 荒木 啓二郎[††]

プログラムスライシング技術は，プログラムの各部分の依存関係を解析し，それをグラフによってモデル化し，ある着目点に関連する部分的なプログラムを求めるという各種問題に有効なものである．また，三次元物体を表示するための言語VRML（Virtual Reality Modeling Language）は，非常に有望であるが，まだツール等による支援は十分ではないため，アニメーション等の動的な部分の開発の作業が困難となっている．この作業に関し，着目する部分に関連する対象群を表すVRML部分プログラムを取り出すことで，困難さを軽減できる．本研究では，VRMLという新規応用分野に対しスライシング技術における従来のグラフによるモデル化を適用することで，VRMLにおけるその部分プログラム抽出の問題を解決した．また，より効果的な適用方法として，コストが低くなるように従来のスライシング手法を改良した．以上を支援ツールとして実装し，有効性を実際に確認した．

# Program Slicing Introduction into VRML and Its Refinement

Hiroshi Maruyama[†] and Keijiro Araki[††]

A program slicing technique is very useful for many kinds of problem. It analyzes dependency in a program, and constructs a graph model, then extracts a relevant partial program for a designated focus from the graph. VRML (Virtual Reality Modeling Language) is a programming language with promising effectiveness in describing 3-dimensional objects. But currently, developments of dynamic features such as animation can not be supported well, thus the developments become difficult. This difficulty can be reduced by extracting a description which is relevant for a focused part. In order to solve this extraction problem, we applied the typical modeling with graphs into the new area of VRML, then indicated a VRML slicing method. Moreover, we proposed more effective slicing methods which can be calculated at lower cost. We implemented a VRML slicing support tool and evaluated its effectiveness.

## 1. はじめに

まず準備として，プログラムスライシングの概要と言語VRMLにおける開発の困難さ，およびそれを軽減するために解くべき問題を説明する．その後，VRMLにスライシングを適用することによりその問題を解決する本研究の背景と目的について述べる．

### 1.1 プログラムスライシングについて

プログラムスライシング技術[1)]は，プログラム内の命令間の関係把握を容易にするものであり，スライシング基準と呼ばれるものにより与えられたある着目点に対し，関連する部分をスライスと呼ばれる部分プログラムとして抽出する．そこでは，プログラムの各部分が節点となり，解析した依存関係が枝となる有向グラフでプログラムはモデル化される．着目点に相当する節点から到達可能な節点集合に対応する部分プログラムがスライスとなる．

スライス対象であるプログラムの文脈を解析してスライスを求める静的スライスと，プログラムを実際に実行した際に得られた情報を用いて求める動的スライスとがある．実用的には，いかに本質的に関連性のある部分のみを，いかに低コストでスライスとして求められるか，すなわちスライスの大きさ（スライスサイズ）と導出コストとが重要になる．

ただし，静的スライスでは，ある実行では実際に使用されない依存関係も考慮して求めるため，得られるスライスが大きくなりがちであるという欠点がある．それに対し，動的スライスでは，サイズを小さくできるものの，プログラムの実行状況を詳細に取得する必要があるため，コストがかかるという欠点がある．

そこで，動的スライスに近い比較的小さなスライスを，静的スライスにおけるコストに近い比較的低コストで求めるような，両スライスの中間的なハイブリッ

† NECソフトウェア九州
NEC Software Kyushu, Ltd.
†† 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

ドスライス[2]と呼ばれるものが現れた．そこでは，静的な解析により構築するグラフに対し，ある実行に関し実際に使用されなかったことが判定できた依存関係の枝を除去した後，スライスを求める．ある特定の実行に関し，動的スライスではなく，このハイブリッドスライスを抽出することは，低コストや実装の容易さの点で有効である．

### 1.2 VRMLについて

三次元物体を含んだ空間を構築できる言語VRML（Virtual Reality Modeling Language）[3),4)]のプログラムは，VRML対応のビューアにて実行し表示され，ユーザはあたかもその三次元空間内に自らが存在し，行動しているかのように，表示状態を様々に変化させることができる．VRMLの第1版では静的な物体しか表現できなかったが，近年の第2版ではユーザとの対話性を持たせ，動的に変化する物体も表現できるようになった．本研究では，この第2版を扱う．

VRMLでは，ノードと呼ばれる物体の構成単位を階層的にまとめ，シーンと呼ばれる空間を構築する．その階層はグラフをなし，シーングラフと呼ばれている．各ノードには，状態を保持するための領域であるフィールドが存在する．実行時にイベントと呼ばれるものをノード間で送受信し，フィールドに値を設定することで，シーンは動的に変化する．

VRMLプログラムの開発において，静的な三次元物体作成は，CADのようなツールによって効果的に支援されるようになってきた．しかし，ノード間の相互作用，イベント送受信タイミング等，VRMLが第2版で拡張された重要な部分は，新しい部分であり，しかも三次元空間に時間という次元も加わり概念的に複雑になったため扱いが難しく，まだツール等による支援は十分ではない．そのため開発が人手による高度に知的な作業となっており，特にテスト，デバッグといった作業やプログラムの理解を困難にしている．

### 1.3 VRMLへのスライシングの導入

作業対象とする領域を小さく分割し，対象領域の概念を簡潔にすることで，前述の作業を支援することを本研究の目的とする．すなわち，VRMLプログラムから，着目している部分に関連するVRML部分プログラムを取り出すことで，その作業の困難さを軽減する．

この部分プログラム抽出の問題の解決には，スライシング技術が適している．そこで以降では，まず2章で，VRMLで表される三次元対象群という新規応用分野に対し，従来のグラフによるモデル化とスライシングとを適用したことについて述べる．ここでは，前述の利点を考慮し，静的およびハイブリッドスライスを扱う．次に3章では，より効果的な適用のために，低コストで求められるように改良したスライシング手法を提示する．4章では，以上を実装した支援ツールを作成し，実験により有効性を確認したことを述べ，最後に5章で，まとめと今後の課題を述べる．

## 2. VRMLスライシングの方法

ここでは，各種要件を考慮し，2種類のVRMLスライスを提示する．次に，VRMLプログラムを各種依存関係のグラフによってモデル化する方法と，それを利用しVRMLスライシングを行う方法とを述べる．

### 2.1 VRMLの特徴分析

VRMLは，オブジェクト，メソッド，クラス，インスタンスといった，オブジェクト指向の概念を取り入れた言語ととらえることができる[4),5)]．ここではその点を念頭におき，VRMLの特徴を分析する．

**ノード** VRMLにおいて，シーンを構成する単位である1つのノードは，オブジェクト指向における1つのオブジェクトに対応する．そこには，変数と，それに関する処理とがカプセル化されている．

**イベント** オブジェクト指向において，オブジェクトがメッセージをやりとりするように，VRMLでは，ノードがイベントをやりとりする．ROUTE文というものが，その送受信経路を指定し，ノード間を関係付けている．

**プロトタイプ** オブジェクト指向におけるクラス定義に対応し，いくつかのノードとROUTEとをPROTO句によりまとめ，プロトタイプと呼ばれるものをVRMLでは定義できる．定義したプロトタイプ名を記述することで，実体（インスタンス）をいくつも生成できる．各インスタンスは，実行時にそれぞれ独立した状態を持つ．

**ノードの共用** VRMLでは，あるノードをUSE句により別の場所で共用（多重参照ともいわれる）することができる．元のノードの状態が変化した場合，それをUSE句により共用しているノードへも動的に反映される．

**並行動作** VRMLでは，各ノードは並行して実行されるため，各ノードは動作する順序が実行のたびに変わる場合がある等，非決定的な性質がある．

### 2.2 VRMLスライスと依存関係との提示

VRMLにおけるスライスも一般的なものと同様に，直観的にはVRMLプログラム中，着目している部分にある影響を与えるプログラム文を含み，実行可能である元のプログラムの一部分（もしくは全部）と考えられる．

**表 1** VRML スライシングのための各依存関係
Table 1 Each dependence for VRML slicing.

| 依存関係名 | 概要 |
|---|---|
| AD (Appearance Dependence) | ノードの親子関係における表示上の影響 |
| BD (Bindable Dependence) | 視点や背景等の影響 |
| CD (Choice Dependence) | 状況に応じて子ノードを選択する場合の関係 |
| GD (Grammatical Dependence) | ヘッダ行と全シーンとの関係 |
| HD (Hit Dependence) | ビューアにおいて対象を指示した場合の影響 |
| ID (Inline Dependence) | 外部ファイルにある他シーンと自シーンとの関係 |
| LD (Light Dependence) | 光源等による表示上の影響 |
| MD (Membership Dependence) | プロトタイプのインタフェースと内部ノードとの関係 |
| PD (Prototype Dependence) | プロトタイプの定義とそのインスタンスとの関係 |
| PDin, PDout | プロトタイプの定義とそのインスタンス間のパラメータ送受の関係 |
| QD (Quotation Dependence) | ノード間で他ノードの属性を読み出す関係 |
| RD (Route Dependence) | イベント送受信ノードと ROUTE 文との関係 |
| SD (Summary Dependence) | PDin から PDout へと依存関係をたどった推移的な関係 |
| WD (Write Dependence) | ノード間で他ノードの属性に書き込む関係 |

一般のスライシングでは，着目している部分とは，ある文のある変数であるので，VRMLにおいても，あるノードのある属性とする．さらに，それが複数ある場合や，あるノードの全属性である場合もある．また，ある影響とは，一般のスライシングでは，変数値への影響のみであった．VRML では，たとえばある空間の絶対座標が変化した際，その空間に属する各物体もそれに従って移動するが，それらの物体は座標という属性の値を持っていないため，物体のいかなる変数値へも影響を与えずに，空間が移動する影響を表示上受ける．以上のように VRML においてスライシングを考える際は，ビューアでの見え方にも配慮する必要があるため，実行時における属性値への影響に加え，属性の値が変わらなくてもビューアで見た際の見え方に関して影響する場合も含める必要があるという複雑さがある．

以上をふまえ，VRML の静的スライシングにおける基準は，任意のノード内の任意の属性から構成され，それは複数で構成されてもよいとする．VRML のハイブリッドスライシングにおける基準には，さらに着目する動作の情報も含める．これは，ある実行の状況を一意に表す情報であるとする．したがって，ユーザから対話的にビューアを通じて動的に与えられたイベントの情報や非決定性等の動作の情報を含んでおり，ある特定の実行状況を表すのに十分な情報を含んだ履歴情報と考えてもよい．

以上で，2 種類の VRML スライスを直観的に示したが，ここで単に影響と表現しているものは，**表 1** に示す VRML における依存関係である．

VRML にスライシングを導入する際，並行動作を行うオブジェクト指向言語に関するプログラムスライシングの研究[6]を参考にした．前述のように，VRML もそのような言語と同様の特徴を持っているため，その依存関係を一部流用した．さらに，VRML の特徴や，VRML スライシングの性質から，独自に必要なものをそれに追加し，表 1 に示す新しい依存関係の体系[7]を構築した．

### 2.3 VRML における依存関係グラフ

以上で定義した VRML における各種依存関係を，実際の VRMLプログラムに適用すると，VRMLプログラムを構成するノードや ROUTE 文等が連結されたグラフができる．これは，ノードの階層関係を示すシーングラフにおいて，各枝に親から子へと向きを付けた有向グラフを含んでいる．これをここでは VRML 依存関係グラフ（VDG）と呼ぶ．

VRMLプログラムを実際に実行せず，その文脈を解析し前述の依存関係を抽出し求められるグラフを静的 VDG とする．それに対し，VRMLプログラムを実際に実行した際，依存関係のうち，それに基づき影響が実際に伝わらなかった事実が確認された依存関係を，静的 VDG から削除して求まるものを動的 VDG とする．

動的 VDG を求めるにあたり，ある実行において全依存関係について各々が有効であったかどうかを判定するために，VRML 実行系の完全な動作の情報を得るのは非常にコストがかかる．そこで，前述のハイブリッドスライスにおける情報取得方法を参考にする．文献 2) では，その地点が実行されたか否かを調べるための情報取得用のコードを，適切と思われる位置に手作業で埋め込み，実行時にその埋め込んだコードから情報を取得している．VRML においても，コードの付与による実行情報取得を採用する．スライシング基準とした振舞いを，コードを付与した VRMLプログラムの実行で再現できた場合のハイブリッドスライ

スは，元のプログラムにおける同一基準のそれと一致する．ただし，コードの付与により元のプログラムの動作の状況が変化し，スライシング基準とした振舞いを，コードを付与したVRMLプログラムでは再現不可能，もしくは再現が困難になる可能性がある．その場合は，ハイブリッドスライスではなく，静的スライスを利用する必要がある．

VRMLの第2版で実現された，内部動作を他言語のプログラムで記述できるノードに関し，本アプローチでは，このノード内のVRMLによる記述部分のみを参照して依存関係を抽出する．これらの他言語のプログラムスライシングの研究成果を利用すれば，その他言語プログラム内部も解析する，より詳細なスライシングが可能となると考える．しかし，今回は内部動作を記述している他言語プログラム内の依存関係まで扱っていないため，VRMLスライスとしては，他言語プログラムで記述された各部分のそれぞれに関し，全体が抽出されるか否かである．したがって，埋め込まれた他言語プログラムの記述の一部分のみが，VRMLスライス中に抽出されることはない．

### 2.4 VRMLにおけるスライスの抽出方法

VRMLにおける静的スライスは，プログラムを静的に解析し静的VDGを構築した後，指示された基準に基づきグラフをたどるという操作により求めることができる．そこでの作業は，順に，解析段階（静的VDG構築），基準指示段階，操作段階からなる．

ハイブリッドスライスは，静的VDG構築後，着目する実行状況に関する情報を取得し，それを用いて静的VDGから動的VDGを構築した後は，その動的VDGにおいて静的スライシングと同じ操作により求めることができる．そこでの作業は，順に，解析段階（静的VDG構築），基準指示段階（実行情報取得を含む），解析段階（動的VDG構築），操作段階からなる．

スライス抽出の際は，文献6)のオブジェクト指向言語でのスライシングと同様に，2段階にわたって依存関係を逆向きにたどればよい．まずVDGにおいてPD，PDoutが存在しないものと仮定し（すなわちPD，PDoutを無視し），スライシング基準である各節点から依存関係を逆向きにたどり，抽出される節点に印をつける．また，基準の場所にも印をつける．その後，VDGにおいてPDinを無視し，すでに印がついた節点から再び逆向きにたどり，抽出される節点に印をつける．最終的に，印がついている節点に対応するプログラム片を集めた部分的なプログラムがスライスとなる．以上のように2段階にたどることで，プロトタイプを経由し各インスタンスが相互に干渉することを防ぎ，より本質的なスライスが求められる．

構築したグラフVDGを，節点集合 $V$，枝集合 $E$ を用いて $G(V,E)$ と表し，基準により与えられた着目点の集合を $Vc(\subseteq V)$ とすると，スライス $S$ は以下のアルゴリズムで求められる．

(1) $S \leftarrow Vc, S' \leftarrow Vc$

(2) $S' \leftarrow \{v' \mid \exists e(v',v) \in (PD$および$PDout$を除く$)\ E\ \ \wedge v \in S' \wedge v' \notin S\}$

(3) $S \leftarrow S \cup S'$

(4) もし $S' \neq \phi$ならば (2) へ戻る

(5) $S' \leftarrow S$

(6) $S' \leftarrow \{v' \mid \exists e(v',v) \in (PDin$を除く$)\ E\ \wedge\ v \in S'\ \wedge\ v' \notin S\}$

(7) $S \leftarrow S \cup S'$

(8) もし $S' \neq \phi$ならば (6) へ戻る

### 2.5 VRMLスライシングの例

ここで扱うVRMLプログラム例を**図1**に示す．説明に便利なように70行の短いプログラムではあるが，イベントやプロトタイプ，USEによるノードの共用といったVRMLにおける特徴的な機能を用いており，一般性も持ち合わせているため，取り扱うに値する例題と考える．また図1では，便宜上，行頭に「*，-，=，>」といった記号をつけている．この記号部分は後の説明で使用するためのものであり，プログラムの一部ではないことを断っておく．

この例は，ボタンを定義する1つのプロトタイプの別々のインスタンスである2つのボタンで，立方体をそれぞれの方向に回転させるものである．立方体の回転軸上方からの見え方を画面上部にUSEを用いて表示する．実行すると，図6の左上のような画面が表示される．左のボタンをクリックすることで立方体を回転させている画面は，**図2**のようになる．

このVRMLプログラム例に対する静的VDGを**図3**に示す．ここでは，各依存関係を矢印付きの線で示しているが，見やすさのために，親ノードから子ノードへのADは太線のシーングラフの階層関係（表示上，途中で2段に分割した）で示されるので省略しており，RDは破線で示している．

**スライス例1** まず，左のボタン（ノード名はBT1）を基準とした静的スライスは，静的VDGにおいてBT1（図3の右側中段のノード「DEF BT1 PBT」）を起点に矢印付の線を逆向きにたどると求まる．この場合，スライスは，ボタンのプロトタイプ，左ボタンのインスタンスからなり，スライスの規模は元のプログラムの約46%になった（図1で行頭に「*」記号がついている部分）．このスライスを実行すると，図6

```
*-=> #VRML V2.0 utf8
*-=  PROTO PBT [
*-=    eventOut SFTime timeON
*-=    ] {
*-=    DEF CYL Transform {
*-=      children [
*-=        Shape {
*-=          geometry Cylinder{}
*-=        }
*-=        DEF BTN TouchSensor {
*-=          touchTime IS timeON
*-=        }
*-=      ]
*-=    }
*-=    DEF BTS TimeSensor {}
*-=    DEF BPI PositionInterpolator {
*-=      key [ 0, 0.2, 0.4 ]
*-=      keyValue [ 0 0 0 , 0 -0.5 0 , 0 0 0 ]
*-=    }
*-=    ROUTE BTN.touchTime        TO BTS.set_startTime
*-=    ROUTE BTS.fraction_changed TO BPI.set_fraction
*-=    ROUTE BPI.value_changed TO CYL.set_translation
*-=  }
*-=> Viewpoint {
*-=>   position 4 3 22
*-=> }
 -=> DEF OBJ Transform {
 -=>   translation 4 0 0
 -=>   rotation 0 1 0 0
 -=>   children [
 -=>     Shape {
 -=>       geometry Box{}
 -=>     }
 -=>   ]
 -=> }
 -=  DEF TM1 TimeSensor {}
 -=  DEF OI1 OrientationInterpolator {
 -=    key [ 0, 1 ]
 -=    keyValue [ 0 1 0 0, 0 1 0 -1.57079 ]
 -=  }
 -   DEF TM2 TimeSensor {}
 -   DEF OI2 OrientationInterpolator {
 -     key [ 0, 1 ]
 -     keyValue [ 0 1 0 0, 0 1 0 1.57079 ]
 -   }
*-=  Transform {
*-=    translation 2 -2 4
*-=    children [
*-=      DEF BT1 PBT{}
*-=    ]
*-=  }
 -   Transform {
 -     translation 6 -2 4
 -     children [
 -       DEF BT2 PBT{}
 -     ]
 -   }
     Transform {
       translation 0 5 -25
       rotation 1 0 0 1.570795
       children [
         USE OBJ
       ]
     }
 -=  ROUTE BT1.timeON           TO TM1.set_startTime
 -=  ROUTE TM1.fraction_changed TO OI1.set_fraction
 -=  ROUTE OI1.value_changed    TO OBJ.set_rotation
 -   ROUTE BT2.timeON           TO TM2.set_startTime
 -   ROUTE TM2.fraction_changed TO OI2.set_fraction
 -   ROUTE OI2.value_changed    TO OBJ.set_rotation
```

**図1**　VRMLプログラム例
Fig. 1　VRML sample program.

の右上のような画面となり，左側の1つのボタン要素しか表示されないが，左のボタンをクリックするとボタン部分が上下に動く．

**スライス例2**　中央の立方体（図3の左側最下のノード「Box」）を基準とした静的スライスは，主にボタンのプロトタイプ，両ボタンのインスタンス，さらに立方体関連の ROUTE 文からなり，規模は元のプログラムの90%になった（行頭に「-」記号がついている部分）．実行では，立方体上部からの見え方を表示する部分を除くすべての物体が表示され，各ボタンのクリックにより，立方体を両方向に回転できる．

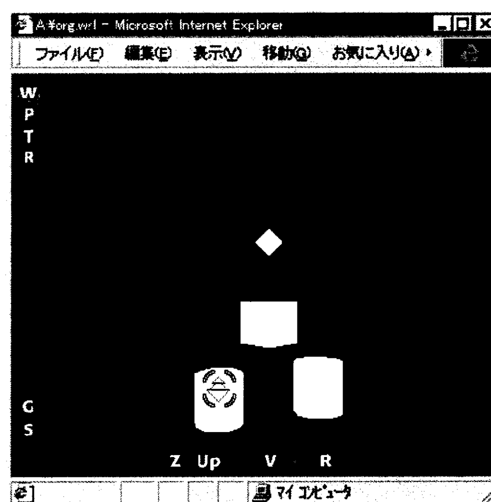


**図2**　プログラム例の実行画面
Fig. 2　An execution screen of the sample program.

**スライス例3**　中央の立方体という同じ基準に対し，左ボタンのみをクリックした場合におけるハイブリッドスライスを求める．プログラムの動作履歴は右ボタンに関するイベントが送信されていない旨を示しているので，図3の静的 VDG から，右下の3つの ROUTE 文が中継している RD を除去できる．そのようにして得た動的 VDG から，主にボタンのプロトタイプ，左ボタンのインスタンス，左ボタンと立方体とを関連付ける ROUTE 文からなるスライスを，元のプログラムの70%の規模で導出できた（行頭に「=」記号がついている部分）．実行時，左ボタンと中央の立方体の2つが表示され，左ボタンのクリックにより，立方体を片方向に回転できる．

**スライス例4**　さらに同じ基準に対し，左右どちらのボタンもクリックしない場合におけるハイブリッドスライスを求める．その場合，すべての RD が使用されていない旨を知ることができるため，全 RD を除去できる．スライスの規模はわずか約19%となった（行頭に「>」記号がついている部分）．中央の立方体のみが表示されるため回転できないが，これは実行時に回転させる影響がどの要素からも立方体に伝わらなかった事実によるものである．

## 3. スライシングの改良

以上で，VRML に対し従来のスライシングを適用

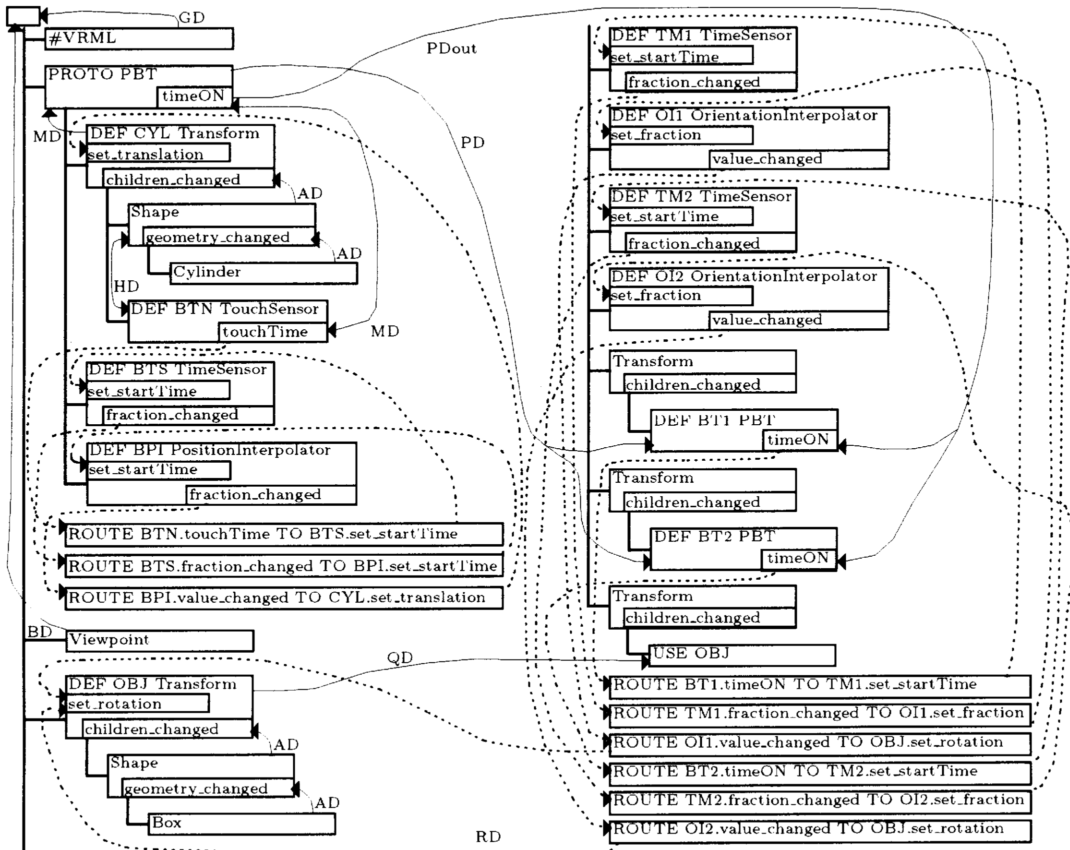


**図 3** プログラム例に対する静的 VDG
Fig. 3 Static VDG for the sample program.

したが，さらに効果的な適用のために，従来のハイブリッドスライシングをさらに改良したことを述べる．それに加え，静的スライシング，ハイブリッドスライシングの両方において適用可能なコスト削減策も提示する．これは，VRML に限らず，一般的なスライシングにおいて広く適用できるものと考えている．

### 3.1 ハイブリッドスライシングの改良

ハイブリッドスライシングに関し，前述のように VRML においても，文献 2) のような，コードの付与による実行情報取得を採用することにしたが，実行されるスライス対象プログラムの適切な位置にその地点が実行されたかどうかを調べるための情報取得用のコードを埋め込む必要がある．ここでは，その文献での手法と異なり，そのコード埋め込み地点の決定や，そこへ埋め込む作業は，人手によらず自動的に行えるようにしたい．そこで，その適切な位置として ROUTE 文の地点を選ぶものとする．

ROUTE 文を選ぶ第 1 の理由は，プログラムの実行時にその文が使用されたかどうかを，イベントの発生の有無により容易に判定できるからである．そのため本研究では，スライス対象プログラムにおける各 ROUTE 文が中継するイベントに関し，本来の受信ノードへの経路とは別の経路で並行して受信できるイベントトラップ用のノードを，計算機によりプログラムの文脈を解析して自動的に加える．このノードは，該当イベント発生時，本来の受信ノードがイベントを受け取る動作に影響しないように，この受信ノードと並行して実行され，各 ROUTE 文が初めてイベントを中継した場合にのみ，その旨を情報として記録する．また，ここで挿入するノードは元のプログラムに何の影響も与えず，コード部分がスライスとして抽出されないようなものとした．

また，VRML プログラムは，一般のプログラムと同様に，モジュールともとらえられるような，ある程

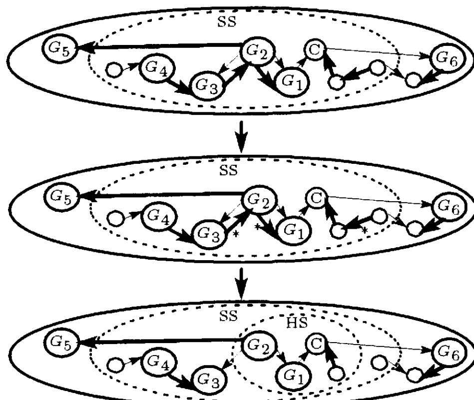


図4 イベントトラップによるハイブリッドスライシングの概念
Fig. 4 The notion of hybrid slicing with event trap.

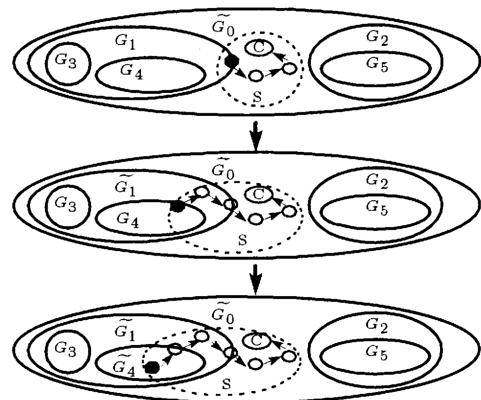


図5 遅延解析によるスライシングの概念
Fig. 5 The notion of slicing with delayed analysis.

度まとまった独立性の高い部分的なプログラム群から構成され，そこでは，そのモジュール間での相互作用が，そのモジュール間で送受信されるイベントのみからなる場合が多い．すなわちROUTE文がモジュール間の橋渡し的な存在となっており，モデル上でいえば，RDが，VDGにおける比較的密な部分グラフ間を連結するための数少ない枝のうちの1つ，もしくは唯一の枝（橋の性質に近いといえる）となる場合が多いのである．そのような場合，このRDが無効であることが判定できたためVDGから開放除去することは，影響を与えていないモジュールをスライスに含めないように作用するので重要であるということが第2の理由である．

その概念を**図4**のVDGによって説明する．ここでは，図3と異なり，各種依存関係を示す矢印のうち，特にここで着目している依存関係であるRDは太い矢印で示している．また，グラフ全体に含まれる部分的なモジュールに対応する部分グラフ $G_i$ を太い実線の円にて，その他でグラフを構成している節点を細い実線の円で示している．まず，上段の静的VDGにおいて，基準として着目している節点Cから依存関係を逆向きにたどることで，破線の円で示す静的スライスSSが求められる．ここで，中段の図に示すように，ある着目する実行において，「*」印をつけているRDによるイベントが発生しなかった旨の情報を得た場合，これらのRDを削除することで，下段の図に示す動的VDGを得る．その動的VDGで着目点Cから依存関係を逆向きにたどることで，破線で示すハイブリッドスライスHSが求められる．この概念図では，$G_3$ から $G_2$ へのモジュール間のRDを削除できたことは，Cに対し $G_3$ や $G_4$ といった広い範囲からの影響がなかったことを示し，スライスからそれらを除くことができたため，効果が大であったと考えられる．

前述のスライス例3および4では，プログラム内の合計9個のROUTE文のそれぞれの動作を監視するノードを計算機が自動的に挿入し新しいプログラムを作成することになる．それを実行することで各ROUTE文がその実行において動作したかどうかの情報を取得できる．

以上が，文献2)における人的コストを減らせるように，自動化という観点で改良したハイブリッドスライシング手法である．

### 3.2 スライシングにおける解析の遅延

ここではスライシングにおける計算コストを減らせるような改良を考える．これまで静的VDGを構築した後に行っていた基準指示段階を先に行い，そのスライス抽出のために解析が不要なプログラムの部分は解析しないように改良する．いくつかの部分に分割したプログラムの各部分に関し，操作段階における依存関係をたどる際，実際に解析が必要になるまで解析するタイミングを遅らせるようにする．

その概念を**図5**で説明する．各 $G_i$ はプログラム中のあるモジュールで，それが含む入れ子のモジュールを除いた部分に対応するが，ここではまだ依存関係の解析を行っていない部分とし，これを解析することで $G_i(V_i, \phi)$ 内部の依存関係を抽出しグラフを構築した場合に $\widetilde{G_i}(V_i, E_i)$ として示すものとする．まず，上段では着目点Cの存在するモジュールにおいてそれが含む全モジュールを除いた部分である $G_0$ に関し，依存関係を解析し $\widetilde{G_0}(V_0, E_0)$ に示されるVDGを構築した後，Cから依存関係を逆向きにたどることで，未解析領域 $G_1$ に属する節点にたどり着く．その段階

ではスライスの中間結果は破線の領域で示す S となる．次に中段の図のように，$G_1$ 内の依存関係を解析し，そこで得た $\widetilde{G_1}(V_1, E_1)$ を VDG に加え，その新たな VDG においてさらにたどり，未解析領域 $G_4$ にたどり着いたとする．すると下段の図のように，さらにその $G_4$ 内を解析し VDG に加えるというように操作を繰り返し，それが終了した場合，最終的なスライスとして下段の図の S が求められる．この概念図は，プログラム中で $G_2$，$G_3$，$G_5$ に対応する部分の依存関係を解析せずに着目点 C に関するスライスを求められたため，解析に関するコストを減らすことができたことを示す．

この方法によるスライス抽出のアルゴリズムを考える．ある段階における未解析のグラフ $G_i$ をまとめた集合を $Gset$ とすると，初期の $Gset$ は，プログラム中において基準の着目点集合 $Vc$ を含んでいないすべての $G_i$ であり，$E$ は前述の $E_0$ である．その後，前述のスライス抽出アルゴリズムにおいて，

$$E \leftarrow E \cup \{e_i \mid \exists G_i(V_i, \phi) \in Gset \wedge S' \cap V_i \neq \phi\}$$

の処理を (2) および (6) の処理の後に加えることでこの遅延解析によるスライシングのアルゴリズムを得る．ただしこの処理では，各 $e_i\ (\in E_i)$ は，それが必要な場合のみ $\widetilde{G_i}(V_i, E_i)$ を構築し，$Gset \leftarrow Gset - G_i$ とした後，得られるものとする．

ここでは，解析を遅らせる対象である単位として，ここではプロトタイプ定義の部分を採用する．それは，その定義部分は，それに対応するインスタンスの記述によりそれを具現化できるが，あるプロトタイプに対応するインスタンスの記述部がスライス対象として操作段階で初めて抽出された際に，そのプロトタイプ内の依存関係解析を行うようにする．

例として，前述のスライス例 4 で，この遅延解析を行う場合を考える．スライス抽出のために依存関係をたどる際，ボタンを定義するプロトタイプの全インスタンス（図 3 の右側中段のノード「DEF BT1 PBT」および「DEF BT2 PBT」）はいずれもたどられないために，このプロトタイプ内部は未解析のままスライスは求められることになる．すなわちこの例では，解析する必要のないボタンを定義するプロトタイプ部分の領域を実際に解析しない分，解析のコストを減らすことができる．

しかし，同じプログラムに対し，スライシング基準を変え，本手法で何度もスライシングを行う際，そのたびに VDG の同じ部分を解析しなおす実装方式の場合，一般に各スライシングのコストの合計がかえって大きくなってしまう．したがって，スライシングの用途や実装方式に注意し，遅延解析を用いるか否かを決定する必要がある．

## 4. ツールの試作およびスライシングの評価

ここでは，以上のスライシング作業の効率化のために試作したツールを説明する．さらに，ハイブリッドスライスにおいて静的スライスに比べ大きさをいかに縮小できたか，改良したスライシングにおいて計算コストをいかに削減できたかといったことを確認するために，そのツールを実行し評価する．

### 4.1 ツールの試作

静的スライシング，ハイブリッドスライシング，さらにそれぞれに対して遅延解析も行えるスライシングのツールを作成した．これを使用し，実際に前述のスライス例 1 を行った場合の画面例を**図 6** に示す．

このツールでは，左上のビューアで，スライス対象の VRML プログラムを実行し，スライシング基準としての動作実行と実行情報取得とを行い，右上のビューアで，スライス結果としての VRML プログラムを実行する．下部の画面において，左側は VRML プログラムテキスト表示域であり，中央が VRML のシーングラフ表示部，右側に各スライシング開始の各ボタン，

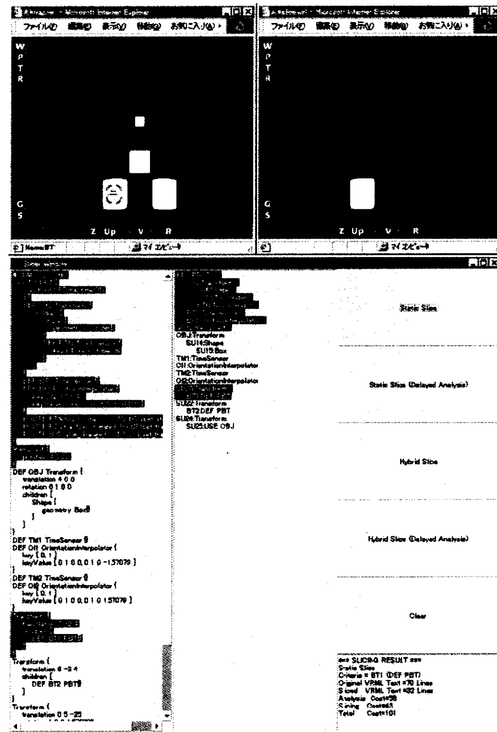


図 6　作成したスライシングツールの実行画面
Fig. 6　An execution screen of our slicing tool.

表2 例題のVRMLプログラムに対する各種スライシング結果
Table 2 Each slicing result for the sample VRML program.

| 基準における着目点 | 静的スライス | HS(両ボタン操作時) | HS(片方ボタン操作時) | HS(ボタン操作なし) |
|---|---|---|---|---|
| 視点(ViewPoint) | 4 ( 64, 38) | 4 ( 73, 47) | 4 ( 79, 53) | 4 ( 91, 71) |
| 左ボタン | 32 (101, 98) | 32 (110, 107) | 32 (116, 113) | 24 (115, 112) |
| 中央の立方体 | 63 (142, 142) | 63 (151, 151) | 49 (137, 134) | 13 ( 99, 79) |

(HSはハイブリッドスライスを意味する)

およびスライシング結果としてコスト等を報告する部分を備える.このシーングラフで基準を選択し,目的の種類のスライシングのボタンを押すことで,スライシングを行い,スライスとして抽出された部分を,プログラムテキストおよびシーングラフ表示部で反転表示し,コストを右下に表示し,さらにスライスは右上のビューアで実行できる.

左上のビューアは,スライス対象のVRMLプログラムに対し,実行情報取得のためのトラップノードを自動的に付与したプログラムの実行画面であるが,さらに表示されている対象に対応するノードの名称を,その対象にマウスを位置付けることで表示する機能も自動的に付与されている.これによりツール使用時に,スライシング基準における着目点としてのノードを実行画面上で視覚的に確認,決定することができる.

また,ここで求めるコストは,解析段階(静的および動的VDG構築)も含めたスライシングにおける全コストからスライス対象プログラムの実行時間を除いたものである.これはスライシングのたびに静的VDG全体を構築しなおしたコストの値であるため,それまでのスライシング作業には依存しない値となる.コストの単位は,1つの依存関係を抽出もしくは削除した場合,その1つをたどる場合,スライスとして節点を1つ抽出した場合,以上のすべてをそれぞれ1としており,ツールにてそれらの処理を行う部分にカウンタを埋め込むことでコストを計測することにした.

本ツールにより,スライシングにおける基準の指示や,結果としてのスライスを把握することが容易になった.また,VRMLプログラムの実行情報を取得する機能の実現部は,それ単独で一般的なトレーサと同様,デバッギング等に活用可能であることも分かった.

### 4.2 スライシングのコスト削減の評価

本ツールを使って,同じスライシング基準に対し,それぞれのスライシング法によるスライスの大きさ,およびコストを比較する.ただし,ハイブリッドスライシングの際は,対象プログラムを実行するコストは含めない.すなわちスライシングに必要な実行に関する情報はすでに得られていることを前提にした.

基準における着目点として,視点(Viewpointノード)の場合,左のボタンの場合,中央の立方体の場合という3つの場合を考える.

さらにハイブリッドスライシングにおいては,基準における実行として,左右両方のボタンを押した場合,左右どちらか一方のボタンを押した場合,どちらのボタンも押さなかった場合という3つ場合も考える.

**表2**に例題プログラムに対する各種スライシング結果を示す.各欄の内容は,「抽出されたスライスの行数(従来の一括解析によるコスト,遅延解析によるコスト)」である.

これまでの議論のとおり,多くの場合,ハイブリッドスライスは静的スライスよりも小さくなった.したがって,ある実行に関して小さいスライスを求めたい場合は,その実行を行い実行の情報を取得するコストは増えるけれども,ハイブリッドスライスを使用したほうが良い結果が期待できることが確認された.

また,本論文での議論を実験的に裏付ける結果として,遅延解析のスライシングにおいて大幅にコストを削減できる場合があることも確認できた.抽出されたスライスの構成によると,ViewPointノードを基準と選んだ場合の全スライス,およびボタンを押さなかった場合の中央の立方体に関するハイブリッドスライスには,ボタンに関するプロトタイプ定義が含まれておらず,その場合にはコストを削減できていることも分かった.この例題では,ボタンのプロトタイプ定義内の依存関係を解析しないことによるコスト削減は,表2から計算すると,およそ20%から40%程度であった.一般に,プログラムが大規模になってくると,プロトタイプ定義も増え,それらのうちスライスに含まれないものも増えてくると考えられるため,大規模なプログラムにおいては,遅延解析のスライシングによる大幅なコスト削減が期待できると考える.また,大規模なプログラムにおいてこそ,コストの議論は大切になってくるので,この遅延解析のスライシングは有用であるといえる.

### 4.3 VRMLスライスの規模と効用

VRMLの実行には,三次元の画像処理といった複雑な処理に多くのメモリやディスク容量,CPUパワーといった計算機資源を必要とするため,大規模なプロ

表 3　VRMLプログラムの例題に対するスライシング結果
Table 3　Slicing result for various sample VRML programs.

| 規模（L） | SS/L | HS/L | HS/SS |
|---|---|---|---|
| 70 | 90% | 70% | 78% |
| 151 | 55% | 33% | 60% |
| 205 | 60% | 49% | 82% |
| 平均 | 68% | 51% | 73% |

グラムを実行すること自体，非常にコストがかかる．さらにこれまでに述べたように，プログラムが大規模になると概念が複雑になってくるため，プログラマによる重要な部分の把握が困難になり，プログラミングにかかる人的コストも非常に大きくなってくる．

スライシングにより着目するプログラムの規模を小さくできる場合，動作テスト作業，動きの速さやモデルの位置等の各種パラメータの調整等の作業を行う場合を考える．その場合，スライスという小さくなったプログラムを扱うことで実行時にビューアのパフォーマンスが向上し，プログラム自身の可読性も向上し，スライス以外の部分についての知識が少なくても作業が可能となり，より小さい簡潔な環境でプログラミング作業が可能になる点で，作業効率が向上できる．

また，ある部分のテスト実行に必要な，ドライバと呼ばれる別の部分も，それらを新規に作成しなくてもスライスにより自動的に抽出できるという点は，テスト作業に有効である．

さらに，スライシングによって，バグの存在する範囲を狭くできることは，解くべき問題を小さくしたこととなるため，デバッグ作業に有効である．小さく単純になったVRMLの部分的なプログラムについて，文脈を見たり実行したりすることで，さらに，プログラマはそのプログラムに対する認識を深めることができ，糸口とする部分を新たに見つけたり，バグに関する考察を深めたりできる．

スライスサイズに関する実験として，前述の70行からなるプログラム例も含め，同程度の複雑さで規模（行数をLとする）の異なる別の例で，代表的な対象に関する静的スライス（行数はSS），および同じ対象に関するハイブリッドスライス（行数はHS）を求めた結果を**表 3**に示す．各ハイブリッドスライス抽出時は，それぞれの例で代表的な操作のうちのおよそ半分だけを行った実行を基準とした．この結果，ハイブリッドスライスは静的スライスの7割程度になっており，それらスライスの規模は元のプログラムの半分程度になることも期待でき，有効であるといえる．

実用規模のプログラムとして，実際に業務で開発中のプログラムに関してスライシングを適用してみた．それは，別の担当グループで作成された計算機の構成部品をアニメーションにより表示するものであり，全体で3000ライン程度であった．それに対し，CD-ROMドライブ部品に関する静的スライスを500ライン程度の部分として取り出せた．それにより新たに担当したCD-ROMドライブ部品のアニメーションのテストやタイミング調整の作業を小規模のモジュール構成で行うことができた．その際スライス部分以外の部品は，別の担当者による動作タイミング調整等の改版作業下にあったが，その改版作業とは独立して自らの作業ができたことは有用であった．さらにそこでは，他の部品についての詳しい知識がなくても作業でき，しかも開発マシンに要求されるスペックが低くても済んだため，自らの手元にあるノートPCの簡便な環境において従来より少ない工数で効率良く作業が行えた．

以上のようにVRMLにおいても，従来の言語の場合と同様，元のプログラムの規模に比べて比較的小さい規模の部分プログラムをスライスとして抽出でき，プログラミング作業に役立つため，本アプローチが有効であることを確認できた．

## 5. おわりに

我々は，VRMLプログラム開発における課題を支援するために，VRMLで表される三次元対象群から，着目している部分に関連する対象群を表すVRML部分プログラムを取り出すということを考えた．そのような部分プログラム抽出の問題を，プログラムスライシング技術におけるグラフによるモデル化をVRMLという新規応用分野に対し適用することで解決した．それに加え，より効果的な適用として，コストが低くなるように改良したスライシング手法も提示し適用した．以上を実装するために支援ツールを作成し，実験の結果，有効性を確認できたことも述べた．

今後は，より大規模で複雑な例への適用や，繰り返しスライシングを行う場合等，各種用途での有効性も細かく分析する必要がある．そのためにツールの各種実装方式の検討や，VRMLプログラム内の他言語記述部分のスライシング，幅広い利用に対応するためのユーザインタフェースの強化を行いたい．

また，今回のスライシング方法を発展させることで，三次元対象群から一部の対象群を取り出すという一般性を利用し，VRML以外の複雑大規模な三次元表示のシステム開発等，各種分野へも応用できるものと考えられる．さらに，VRMLの持つオブジェクト指向性から，VRMLに限らず，オブジェクト指向プログラ

ミング言語におけるスライシングという一般的な観点で，他の各種言語への適用も検討していく予定である．

## 参 考 文 献


1) 下村隆夫：プログラムスライシング技術と応用，共立出版 (1995).
2) Gupta, R. and Soffa, M.L.: Hybrid Slicing: An Approach for Refining Static Slices Using Dynamic Information, *Proc. 3rd International Symposium on the Foundation of Software Engineering*, pp.29–40 (1995).
3) The Virtual Reality Modeling Language Specification, *ISO/IEC DIS 14772-1* (1997).
4) Beeson, C.: An Object-Oriented Approach To VRML Development, *Proc. VRML97*, pp.17–24 (1997).
5) Park, S. and Han, T.: Object-Oriented VRML For Multi-user Environments, *Proc. VRML97*, pp.25–32 (1997).
6) Zhao, J., Cheng, J. and Ushijima, K.: Static Slicing of Concurrent Object-Oriented Programs, *Proc. 20th IEEE Annual International Computer Software and Applications Conference*, pp.312–320 (1996).
7) 丸山博史，荒木啓二郎：VRMLにおけるプログラムスライシングとその利用，情報処理学会論文誌：プログラミング，Vol.40, No.SIG 10 (PRO 5), pp.51–63 (1999).



(平成11年6月16日受付)
(平成11年8月 6 日再受付)
(平成11年9月18日採録)

**丸山 博史**（正会員）

1968年生．1992年九州大学大学院工学研究科情報工学専攻修士課程修了．同年NECソフトウェア九州入社．以来，汎用機からPCまで広く開発業務に従事．1998年より2年間，九州大学大学院システム情報科学研究科情報工学専攻社会人博士後期課程．博士（工学）．現在，サーバ管理システムやセキュリティシステムの開発，およびプログラムスライシング等の研究に従事．

**荒木啓二郎**（正会員）

1976年九州大学工学部卒業．1978年同大学院修士課程修了．九州大学助手，同助教授，奈良先端科学技術大学院大学教授を経て，現在，九州大学大学院システム情報科学研究院教授．（財）九州システム情報技術研究所研究室長兼務．工学博士．形式仕様記述，ソフトウェア開発方法論，プログラミング言語，並列/分散処理，インターネット，マルチメディア通信等の研究に従事．日本ソフトウェア科学会，ACM，IEEE CS等会員．ソフトウェア技術者協会常任幹事，KARRN協会事務局長，博多祇園山笠西流元赤手拭い等．