VRMLにおけるプログラムスライシングとその利用

丸山博史[†,††] 荒木啓二郎[††]

VRML(Virtual Reality Modeling Language)は，3次元空間や3次元物体をリアルタイムに表示するための言語であり，サイバースペース構築，マルチメディアデータの表現等で非常に有望な言語である．物体の形状等の静的な部分の開発に関しては，CADにおける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が，アニメーションのように動作時に動的に変化するような重要な部分に関しては，まだツール等による支援は充分ではない．そのため動的な部分の開発に関して，デバッグやテスト，動作のタイミングの調整といった作業が困難になる場合がしばしばある．そこで本研究は，従来の一般的なプログラミング言語において，デバッグ，テスト，プログラム理解といったことに効果があるプログラムスライシング技術を，VRMLに対して導入しようというものである．まず，VRMLプログラムの構造や実行の特徴を分析し，VRMLにおいてスライスのあるべき姿を考え，各種スライスを定義した．次に，並行動作するオブジェクト指向プログラミング言語に対する最近のスライシングの研究結果を用い，VRMLプログラムの特徴に注意しながらVRMLにおける各種依存関係を考えた．最後に，それらの依存関係を用いた各種スライス導出法を提案した．以上を各種の例に適用し，支援ツールを試作した．それにより，特にデバッグやテスト，動作のタイミングの調整といった作業での有効性を実際に確認できた．本研究は，VRMLにおける開発作業の効率化の他にも，3次元データの解析や理解等，様々な応用が考えられる．

Program Slicing on VRML and Its Use

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

VRML(Virtual Reality Modeling Language) is a programming language with promising effectiveness in describing cyber space and multi-media data because of its real-time expressive power for 3-dimensional objects. Concerning 3-dimensional design with VRML, kinds of CAD tools can support static features in system developments effectively. But currently, developments of dynamic features such as animation can not be supported well, thus the tasks of debugging, testing, and behavior parameter adjustment are sometimes difficult. Program slicing is an effective technique for many areas, e.g. debugging, testing and program understanding. We have applied such program slicing to VRML. In this paper, we analyzed VRML program structures and features, then we studied what the VRML slices should be, and presented their definitions. Based on the recent slicing techniques of concurrent object-oriented programs, we considered some features peculiar to VRML behaviors, and defined various kinds of dependencies in VRML. Finally we proposed some VRML slicing approaches based on such dependencies. We applied the above approach to some examples, and implemented several prototypes of support tools. Then we evaluated the effectiveness of our approach especially in debugging, testing and parameter adjustments phases. Not only for efficient VRML programming, we think our research results can be applicable to many areas like 3-dimensional data analysis and understanding.

1. はじめに

ここでは準備として，まずVRMLとプログラムスライシングとを概説し，本研究の背景と目的について述べる．

† 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科
Graduate School of Information Science and Electrical Engineering, Kyushu University

1.1 VRMLについて

VRML (Virtual Reality Modeling Language) [1]は，各種3次元物体を含んだ空間をインターネット上で構築できる言語である．VRML対応のブラウザにて，ユーザは，あたかもその3次元空間内に自らが存在し，行動しているかのように，状態をいろいろ変化させることが出来る．VRML言語は，第1版では静的な物体しか表現できなかったが，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．しかも，Java言語等の他言語でのよ

り詳細な記述を組み込むことも可能となった．そこで本稿では以降，この第2版であるVRML 2.0を扱う．

VRMLでは，空間を構成する単位の部品をノードと呼び，そのノードを階層的にまとめて構成される各種物体を含む空間をシーンと呼ぶ．また，そのノードの階層はシーングラフと呼ばれる．ノードは状態を持ち，ノード間でイベントと呼ばれるものを実行時にやりとりし，その状態を変えることで，シーンは動的に変化する．

VRMLにおける開発の大きな流れは，3次元物体の作成に始まり，その物体間のやり取りをイベントとして実現し，それらで3次元空間であるシーンを構築する．その後，動作の調整やテストを行うわけであるが，そこでは実際に各種イベントを発生させ，その時のシーンの遷移が仕様通りかどうかを検査し，必要ならば各種修正を行う．

3次元物体を作成することに関しては，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．

それに対し，作成した3次元物体のいくつかを組み合わせ，それらが動的に変化するようなプログラムを作成する場合，実行時に表示されるシーンの変化が重要となる．そのため，プログラミング後にそれらを実際に動作させ，そこで表示される結果を確認し，修正により目的とするものへ近づけていくというプロトタイピング的な開発方法が有効である．従って，そのサイクルを効率良く進めるためにはプログラムの理解やテスト，デバッグといった作業が重要である．

しかしVRMLが第2版で拡張されたこのような動的な部分に関しては，新しい部分であり，さらに3次元空間に時間の流れという次元が加わり概念的に複雑になったことで扱いが難しく，まだツール等による支援は充分ではない．そのような動的な部分のプログラミングではイベントの送受信の設定が中心となるが，そこでは，イベントをやり取りするノード間の関連性やイベント値，イベント送受信タイミング等を把握することが重要である．しかし，数多くのノードが存在し，各種のイベントがやり取りされるというように，シーンが複雑になればなるほど，それが把握しにくくなるため，先ほど述べた各種作業が困難になっている．

1.2 プログラムスライシング技術について

プログラムスライシング技術[2]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，プログラム内の各種依存関係に基づき，プログラム中のある着目している(スライシング基準と呼ばれる)部分に関連する部分を解析により抽出する．スライシング基準に基づき抽出された部分プログラムは，スライスと呼ばれ，これによりその着目点に関する開発や部分的なテスト等を効果的に行える．また，仕様と違う動作を行った際，その部分にそのような誤った動作をさせるように影響を与えているプログラム中の範囲を限定することができるので，デバッグ作業が容易になる．

スライスには，プログラムを実行させずに文脈を解析してスライスを導出する静的スライスと，ある状況でプログラムを実行し，その実行の情報を用いて導出する動的スライスとがある．

しかし，ある特定の実行に関するスライスを求めたい場合に静的スライスを用いることは，その実行では実際に使用されない依存関係も考慮しているため，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．反対に動的スライスでは，スライスの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．以上の問題を軽減するために，本質的なもののみからなるスライス，もしくはそれに近いサイズのスライスを，より低コストで求めるような，静的スライスと動的スライスとの中間的なハイブリッドスライス[3]というものが考えられるようになってきた．ここでは，プログラム内の依存関係を静的に解析した後，その依存関係のうちのいくつかに関し，ある特定の実行において依存関係に基づき影響が発生したか否かを判定する．その後，影響がなかったため無効であったと判定できた依存関係を削除し，最終的に残った依存関係を用いてスライスを求める．従って，動的スライシングに比べ，スライス対象プログラムを実行し動作情報を取得する際のオーバーヘッドが小さく，静的スライスよりも小さくなり得るスライスを求めることができ，さらにその実装が比較的容易であるという利点がある．

1.3 VRMLへのスライシングの導入

プログラムスライシングはデバッグ，テスト，プログラム理解といったことに効果があるため，それを用いるとVRMLでは動的に変化する複雑なシーンの開発が効率化できると考える．そこで本研究ではプログラムスライシング技術をVRMLに対して導入した．

VRMLプログラムの実行では，3次元に関する計算や表示に大量の計算機資源を使用するため，前述のスライシングにおけるオーバーヘッドを小さくできる点は重要となる．また，まずは早い段階で本アプローチの有効性を実際に確かめるために，実現の容易性も大切である．従って本論文では，任意の実行に対するスライシングのためには静的スライスを，ある特定の実

行に対してはハイブリッドスライスを扱うことにした．

以降では，VRMLプログラムの特徴を分析し，それに注意しVRMLにおけるスライスを定義する．次に，並行動作するオブジェクト指向プログラミング言語に対する研究成果をもとに，VRMLにおける各種依存関係を考え，スライスの導出法を提案する．最後に，支援ツールを試作し有効性を確認する．

2. VRMLスライシングの方法

ここでは，VRMLプログラムの特徴を分析し，VRMLにおいてスライスはどのようであるべきかを考え，二種類のスライスを定義する．次に，並行動作するオブジェクト指向プログラミング言語に対する最近のスライシングの研究結果を用い，VRMLのプログラムの実行の特徴に注意しながらVRMLにおける各種依存関係を考え，スライスの導出法を提案する．

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もできる[4)5)]．そこで，ここではオブジェクト指向の概念を念頭において，VRMLの言語の特徴を分析してみる．

ノード シーンを構成する単位である1つのノードは，1つのオブジェクトと考えられる．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．ノード内部のexposedField，fieldという属性を持つ2種類のフィールドは，それぞれpublic，private変数に対応するものである．

イベント オブジェクトがメッセージをやりとりするように，ノードはイベントをやりとりする．ノードはそのために，イベントの入口であるeventIn，イベントの出口であるeventOutを，それぞれ複数持つことができる．イベントを受信すると，その受信ノードの内部処理が動作を開始し，受信したイベントの値を参照したり，処理結果をイベントとして送出することができる．ROUTE文というものにより，あるノードのeventOutと別のノードのeventInとを関係付けることができる．すなわちROUTE文は，イベントの送受信経路を指定するものと捉えられる．

並行動作 VRMLでは各ノードは並行して実行できる．そのため，どのノードが先に動作するか等は実行のたびに変わる場合がある．また，複数のイベントが，ある一つのノードに同時に渡された場合，受信ノードにおいて，どのイベントが先に内部で処理されるかという点も非決定的である．

プロトタイピング オブジェクト指向におけるクラス定義に対応するものとして，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作ることができる．プロトタイプ内部のノードの属性の一部をパラメタ化し，外部に示すこともできる．シーンに実体を生成するには，定義したプロトタイプ名と，カスタマイズのためのパラメタの値とを記述する．同じプロトタイプの別々のインスタンスは．実行時にそれぞれ独立した状態を持つことができる．

ノード再利用 USE句により，あるノードを別の場所で再利用することができる．同一のノードの再利用を複数行っても，プロトタイプの場合と異なり，それらは全て同じ状態になる．ここでは，元のノードの状態が動的に変わった場合，それを参照しているUSE句により出現したノードへも動的に反映される．

他言語による記述 ユーザがそのノードの内部処理をJavaやJavaScript等の他言語を用いて記述し，新しいノードをつくるためにScriptノードというものがある．このノードでは，他のノードへの参照をフィールド値として持つことで，内部処理において他ノードの属性を自由に読み書きすることもできる．

2.2 VRMLスライスの要件

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．

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が，VRMLでは，ビューアでの見え方を重視したいので，実行時に属性値を決定することに関する影響に加え，着目している属性の値が変わらなくとも実行時に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ことにする．

スライスとして抽出されるプログラム片に関して，あるノードのある属性値が基準に影響を及ぼしたとしても，スライスを動作可能にするためには，ノード記述部においてその属性が記述してある部分だけではなく，そのノード記述部分全体を抽出する必要がある場合もある．そのように，スライスとして抽出されるプログラム片の最小単位という概念が存在する．ここでは，この抽出単位を以下のものとする．

- ROUTE文．

- 1つのノードタイプからなるノード全体．ただし，それが，子ノードを含むことができる場合で，かつその子ノードが抽出対象でない場合は，それを保有するための属性記述部分を除く．
- プロトタイプ定義のインタフェース部分．ただしインタフェース内の属性記述部分は，影響する属性のみを含む．

実行可能でないスライスという概念も存在するが，今回は実行してビューアで視覚化できることによりプログラマがスライスを把握しやすくなるという利点を重視するために，典型的な実行可能なスライスを扱っている．

しかしそれでもなお，例えば空間を表現するためのノードがスライスとして抽出された場合，そのスライスを実行してもその空間のみしか表示されず，その空間が存在し動作しているといったことが，表示上では何も確認できないといった問題がある．視覚化を重視するための回避策として，その空間に含まれる全ての可視のノードもスライスの中に抽出することも考えられる．しかしそれでは，空間内に多くの可視ノードが含まれる場合，スライスが大きくなり過ぎるので，今回はそれは行わない．代わりに，抽出される空間内において注目したい可視ノードもスライシング基準で指定することで，その可視ノードも含めたスライスを抽出でき，その空間の動作も表示できるため，これを用いる．ただしそれだけでは，基準の選定に多くの手間や知識を要するため，今回は可視ノードの直接の親がスライスとして抽出される場合にその可視ノードも抽出されるようにする．それにより，基準の選定の容易さと抽出されるスライスの可視性の高さとを両立し，しかもスライスの大きさの増加を比較的押さえることが出来るものと考える．

2.3 VRMLにおけるスライスの定義

以上をふまえ，VRMLの静的スライシングにおけるスライシング基準は，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良いとする．

VRMLのハイブリッドスライシングにおける基準には，着目する動作の情報も含める．ここで，この動作の情報とは，VRMLにおけるある実行の状況を一意に表す情報であるとする．従ってこれは，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などの動作の実行の情報を含んでおり，ある実行状況を表すのに十分な情報を含んだ動作履歴と考えても良い．

以上の基準で指定された属性群の値へ影響する抽出単位，もしくはそれらの属性を含むノードへと見え方に関して影響する抽出単位を含み，実行可能な部分プログラムがVRMLにおける静的スライスである．ハイブリッドスライスは，基準で指定された動作状況下でのみ影響する部分を含むものとする．

以上が直観的なVRMLにおけるスライスの定義であるが，ここで単に影響と表現しているものは，以降で詳しく述べる各依存関係である．

2.4 並行動作を行うオブジェクト指向言語におけるスライシング

VRMLにスライシングを導入する際，VRMLと似た性質を持つ言語におけるスライシングが既に考えられていれば，当然それに準拠して導入を行いたい．参考となる研究として，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]がある．その研究では，オブジェクト指向や並行動作といった概念を含む言語におけるスライシングを，各種依存関係で取り扱っている．前述のように，VRMLもそのような言語と同様な特徴を持っていると考える．そこで以降では，その研究にて提示された依存関係について，その説明(および例)を示した後，それがVRMLではどのような関係になるのかを検討した結果を示す．

Control Dependence 実行経路制御処理が与える影響(例：if文中の条件式→そのifブロック内の処理)．VRMLでは，イベントにより制御を操作することができる．イベントを受信した際，そのノードの処理が起動されるが，同時に値も伝播するので次のData Dependenceとあわせて扱う必要がある．

Data Dependence 変数の値が与える影響(例：ある変数への代入式→その変数を含む式を別の変数へ代入する式)．VRMLでは，イベントによる値の受け渡しやノード間での属性の読み書きでは，値が伝わるためData Dependenceが存在すると考える．

Selection Dependence 非決定的な実行順序(例：ある処理→その処理終了後，実行が非決定的に選択される複数の処理)．VRMLにおける非決定な動作に関しては，前述のようにイベントや属性へのデータ受け渡しを伴なっている．従って，先のControl DependenceやData Dependenceを用いて表現でき，この依存関係を特別に用意する必要はないと考える．

Synchronization Dependence 実行を待ち合わせる場合(例：いくつかの処理→それらの処理の全ての終了を待って起動される処理)．VRMLでは，あるノードから発生したイベントが，複数のノードを経由し同時に一つのノードに到着する場合がある．ある1つのノードが複数の属性値の更新を待つような

内部処理を持つ場合もある．しかし，これらの依存関係も，値の送受という点で Control Dependence や Data Dependence により表現できると考える．

Communication Dependence 別プロセスにおいて，それぞれの変数間で与える影響 (例：あるプロセスが，あるオブジェクトの共有領域に書き込む→別のプロセスがそれを読み出す)．VRML では，複数のノードがある一つのノードの同一の属性を読み書きする場合となるが，これも値の送受を伴なうため Control Dependence や Data Dependence を用いて表現できる．

Class Membership クラスと，それが保有するメソッドとの関係 (例：あるクラス→それに属するメソッド群)．VRML では，プロトタイプの宣言部分と，その定義部分に含まれる各ノードとの関係となり，この Class Membership を用いる必要がある．

Call クラス内のメソッドを呼び出した場合 (例：処理の呼び出し元での call 文→呼び出し先メソッドの入口)．

Parameter 処理単位間での引数によるデータ受け渡し (例：処理の要求元の実パラメタ→要求された側の仮パラメタ)．

Summary 呼び出し元での変数間で，処理を呼び出したことにより生じる依存関係 (例：実パラメタとして呼び出し先に渡すある変数→呼び出し後に呼び出し元で影響を受けている変数)．

最後の 3 つの依存関係は，あるまとまった処理の呼び出しに関するものであるが，VRML では，あるノードの処理を呼び出すには，イベントを送るなどしてそのノードの属性値に値を書き込む必要がある．従って Call の関係は，値の送受という点で先の Control Dependence や Data Dependence を用いて表現できる．プロトタイプ内のノードの処理と，そのインスタンスとの関係は，Parameter や Summary と同様の関係を用いて対応する必要があると考える．

2.5 VRML における各種依存関係

上述の依存関係にさらに VRML スライシングで独自に必要と思われるものを追加し，新しい依存関係の体系を構築した．

VRML の多様な性質を扱うには，数多くの依存関係が必要となった．そこで，以下ではそれらの依存関係を数種の部類に分け，VRML においてスライシングに必要な依存関係としてまとめて提示する．各依存関係についての補足説明は付録にて示すことにする．

2.5.1 データ伝播関係

これは前述の Data Dependence に対応する．ただし，ここではイベント送受信関係等の，データを受け取ったノードの内部処理を起動する関係は除く．

Quotation Dependence (QD) 他のノードを参照し，その属性を読み出す関係．

2.5.2 制御&データ伝播関係

これは前述の Control Dependence と Data Dependence との両方を含む場合に対応する．

Write Dependence (WD) 他のノードを参照し，その属性に書き込む関係．書き込みと読み出しとの両方，もしくはそのどちらか特定できない場合は先程の QD と共に存在する．

Route Dependence (RD) ROUTE 文により指定されるノード間でのイベントの経路を示すものであり，送受信ノードと ROUTE 文との関係．

2.5.3 実行制御関連

これは前述の Control Dependence に対応するが，ここでは値が伝わる関係は除くものとする．

Choice Dependence(CD) ノードの階層において，状況に応じて親ノードがどの子ノードと親子になるのかを選択する場合の関係．

Hit Dependence (HD) ポインティングデバイスによる対象の指示，もしくはビューアの操作で視点が対象に衝突するといった場合を表現するための関係．

Inline Dependence (ID) Inline ノードと呼ばれる，外部ファイルの部分的なシーンを取り込むノードが存在する場合の，その部分的なシーンとの関係．

2.5.4 プロトタイプ関連

既に述べた Parameter や Summary，Class Membership というプロトタイプに重要な関係も含め，プロトタイプに関連するものとしては，以下のような依存関係があると考える．

Prototype Dependence (PD) プロトタイプの定義とそのインスタンスとの関係．さらに，パラメタ受け渡しのために，インスタンスからプロトタイプへの PDin と，その逆の PDout とを，PD とは別に導入する．

Summary Dependence (SD) インスタンス内の属性間にて，PDin から PDout へとプロトタイプ内部の各種依存関係をたどった推移的な関係．

Membership Dependence (MD) これは，プロトタイプのインタフェースと，その定義内のノードのうちパラメタ化で外部に公開されている属性との関係である．また，プロトタイプ定義内の先頭のノードがインスタンスとなるノードであるので，そのノードとインタフェースとの関係も含む．

2.5.5 シーンの見え方関連

VRMLスライスの要件や定義にて述べたように，一般のスライシングと異なり，VRMLにおけるスライシングでは，直接ノードの属性値を変化させる関係ではないものの，シーンの表示上の影響という関係も考慮する．このような関係として，ここでは物体に色や質感を与えたり，ノードが属している親ノードからの支配的な影響，光の影響，ビューアへの影響等を扱う．

Appearance Dependence (AD) ノードの階層における，親から子への関係．また，前述のスライス可視性向上も考慮し，可視のノードからその親ノードへの関係も含む．

Light Dependence (LD) 光源のノードは，他のノードの属性値を変化させるわけではなく，ビューアで光源から影響を受けたノードを表示した場合の見え方に影響する．そこで各光源のノードから，そのプログラムが構築するシーングラフのトップレベルの全構成要素へとLDがあると定める．ここで全トップレベルの親となる仮のノードを用意すれば，このような依存関係は，その仮ノードとの1つの依存関係のみでまとめて示すことができる．

Bindable Dependence (BD) バインダブルノードと呼ばれるものはシーン内に複数存在することができるが，ある一つが有効になった(バインドされた)場合は，他のものは無効化される性質を持つ．これには視点や背景等があり，ビューアの操作を通じてユーザがそのバインドの状態を変更することができ，これらもシーンの見え方に影響する．そこでそのノードから，全トップレベルへBDがあると定める．

2.5.6 文法関係

ある部分プログラムを実行可能なものとするために必要な別の部分は，これまでの依存関係でその多くが関係付けられるようになっている．ここでは，それでもなお関係付けることができなかったものを扱う．

Grammatical Dependence (GD) VRMLプログラムであることを示す"#VRML V2.0 utf8"のようなヘッダ行は，そのテキストがVRMLプログラムであることを示す記述である．スライスとして抽出される全ての部分プログラムにおいて，その先頭にこの行が存在するようにしたい．そのためにシーンの全トップレベルとこの記述との関係GDを導入する．

2.6 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．これをここではVRML依存関係グラフ(VDG)と呼ぶことにする．

VRMLプログラムを実際に実行せず，前述の依存関係をもとに求められるものを静的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，静的なVDGから削除して求まるものを動的VDGとする．従って動的VDGは静的VDGの部分グラフとなり，動作の非決定性やビューアでの操作等で実行状況が異なれば，同じプログラムに対する動的VDGは複数存在する可能性がある．

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際，それに必要な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3)では，情報取得用のコードを手作業で埋め込み，その埋め込んだ部分が実行されたかどうかといった情報を取得している．それに対し，VRMLにおいて着目したいのは，中継すべきイベントが発生しなかった場合，それに対応するROUTE文はそのイベントの中継を当然行わないという特徴である．その場合，RDがその役目を果たさなかったということなので，そのようなRDを削除できるのである．実行するVRMLプログラムにそのような事実を取得するためのコードを，実行前に埋め込んでおくことでその情報を取得できる．また，VRMLにおけるこのようなコードの付与は，プログラムの文脈を利用することで自動的に行える．従ってこの手法は，文献3)における手作業でコードを埋め込む方法よりも，自動化という点で優れていると考える．

2.7 VRMLにおけるスライスの抽出方法

VRMLにおけるスライスは，VDGを構築した後，そのVDGにおける操作により求めることができる．静的スライスは静的VDGから，ハイブリッドスライスは動的VDGから，それぞれ求めることができる．

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たどれば良い．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点から依存関係を逆向きにたどり，抽出される点に印をつける．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，既に印がついた点から逆向きにたどり，抽出される点に印をつける．最終的に，印がついている点に対応するプログラムの抽出単位を集めた部

分的なプログラムがスライスとなる．

3. VRMLスライシングの例

ここでは，プロトタイプ等のVRMLにおける各種の重要な要素を持ったVRMLプログラムを例に，実際にVDGを構築し，スライシングを行う．また，実用規模のVRMLプログラムに対して適用した時の考察も述べる．

3.1 プロトタイプ等を含むプログラム例

ここではVRMLプログラムの例として，中央にある立方体を前面の2つのボタンで回転させるものを考える．2つのボタンは，ボタンを定義するプロトタイプの別々のインスタンスとして存在する．立方体の回転軸が視点から見て垂直で回転がわかりにくいので，その立方体を回転軸の上方から見た景色を画面上方にUSEを用いて表示する．各ボタンはクリックされると一瞬下がった後に上へ戻る．左のボタンがマウスでクリックされると中央の立方体を上方から見て時計回りに90度回転させ，右のボタンではその逆となる．

このVRMLプログラム例を図1に示す．説明に便利なように短いプログラムではあるが，プロトタイプ，USEによる引用といった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VRMLプログラム例に対する静的VDGを図2に示す．ここでは，各依存関係を矢印付きの線で示しており，RDは特別に矢印付きの破線で示している．また，親から子へのADは，太い線のシーングラフにて示されるので省略している．さらに，全トップレベルをまとめる仮ノードを表記上使用している．これを実行した際のビューアで見た画面は，図3の上部左のようになる．

3.2 各種スライスの実行とスライス規模の比較

スライスの例として，左のボタン，中央の立方体といった各基準についてのスライスを求める．

まず，左のボタン(ノード名はBT1)についての静的スライスは，BT1についての全属性を基準とし，静的VDGをたどると求まる．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成り，スライスの規模は元のプログラムの約46%になった(図1にて行頭に「*」記号がついている部分)．このスライスを実行すると，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，

```
*-> #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 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.

ボタン部分が上下に動く．

中央の立方体についての静的スライスは，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文から成り，スライスの規模は元のプログラムの90%になった(図1にて行頭に「-」記号がついている部分)．このスライスを実行すると，立方体上部からの景色を示す部分を除く全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．

次に，左のボタンをクリックするという実行における中央の立方体についてのハイブリッドスライスを求める．その実行の情報として，右のボタンからのROUTE文が使用されなかった旨の事実から，そのボタンから中央の立方体の存在する空間(ノード名はOBJ)に至る一連のRDを全てVDGから削除するこ

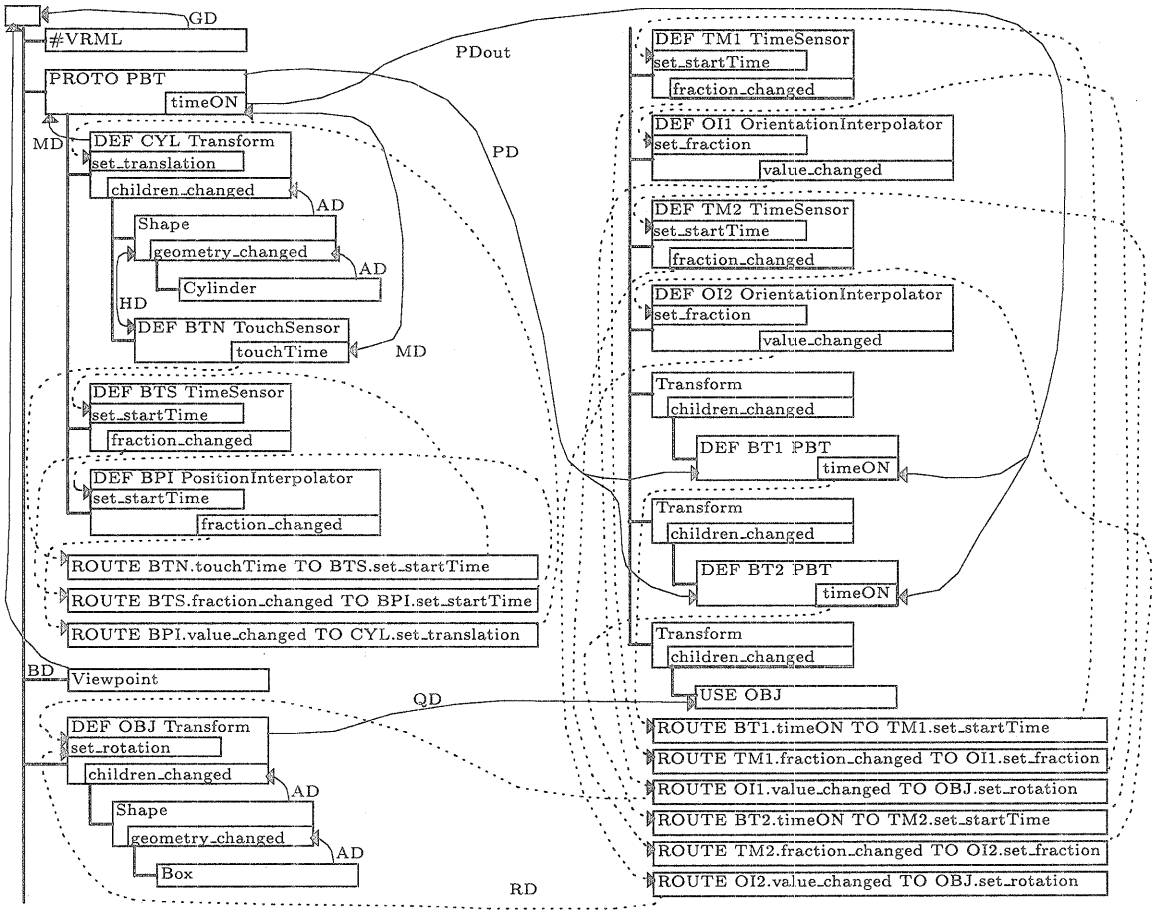

図 2 プログラム例に対する静的 VDG
Fig. 2 Static VDG for the sample program.

とで，その実行に関する動的 VDG を得る．すると，ハイブリッドスライスの場合の規模は元のプログラムの 70%になり (図 1 にて行頭に「>」記号がついている部分)，同じ中央の立方体についての静的スライスに比べ，その約 22%をさらに削除することができた．このスライスを実行すると，左のボタンと中央の立方体との 2 つだけが表示され，その左のボタンのクリックにより，立方体を片方向にのみ回転できる．

以上が，小規模の VRML プログラムに対して適用した例である．その 70 行からなる例も含め，同程度の複雑さで規模 (行数を L とする) の異なる別の 2 つの例でも，それぞれにおいて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 1 に示す．ここでは，各ハイブリッドスライスは，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ものである．この結果から，一般にハイブリッドスライスは静的スライスの 7 割程度になり，それら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，有効であると言える．

表 1 VRML プログラムの例題に対するスライシング結果
Table 1 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%

さらに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，別の担当グループにて作成された計算機の構成部品をアニメーションにて表示するもので

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出来た．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出来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無くとも作業でき，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ことが分かった．

4. VRMLスライシングの利用とその効用

4.1 テストや調整作業の効率化

VRMLの実行には，3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．

しかし前述の場合のように，スライスを利用することでサイズが小さくなった分，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タの調整等の作業に関し，以下の点で効率を向上できることがわかった．

- 実行時にビューアのパフォーマンスが向上
- プログラム自身の可読性も向上
- スライス以外の部分についての知識が少なくとも作業が可能
- エディタ等に関しても，より小さい簡潔な環境でプログラミング作業が可能

VRMLプログラミングに適すると考えられるプロトタイピングに基づく開発にて，試作完了，テスト実行，修正案決定，修正実施，再びテスト実行といったサイクルを繰り返す場合を考える．すると，スライスを利用することは，そのサイクル自身を円滑に繰り返すことができるため効率向上に貢献するとも言える．

また，ある部分をテストのために実行したい場合，テスト用のドライバやスタブと呼ばれるような，その部分を実行するための別の部分も必要になる．それらを個別に用意する方法もあるであろうが，スライシングにより，それらを元のプログラムから自動的に抽出できる点も有用である．

4.2 デバッグ作業の効率化

VRMLプログラム開発において，文法エラーなどを修正し，開発中のプログラムが動作するようになった段階で，デバッギングに取りかかる場合を考える．プログラマはVRMLを動作させ，その振る舞いと自分の頭の中にある動作の仕様とを突き合わせ，まずは表面上で怪しいと思われる糸口を見つけ出すだろう．

例えばシーン内において，あるノードn1をクリックした後，別のノードn2だけが予想される動作と異なる動作をした場合，プログラマはこのノードn2を糸口として分析を始めるであろう．そこで，その糸口のノードを基準としたスライスを求めるとこで，誤った動作をさせた原因がスライス内に捕らえられる．

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のである．

実際に，デバッギングのために使用してみると，バグを狭い範囲に追い込めることに加え，VDGを直接用いて依存関係をエラーからバグの位置までさかのぼることができる点でも有効であることが分かった．

5. ツールの試作

ここでは，スライス作業を計算機で行えるように試作した各種ツールと，プログラミング支援のためにそれらを統合したスライシング統合環境とを述べる．

5.1 トレーサーとその利用

VRMLプログラムの開発において，シーンを行き交うイベントを調べることは，テストやデバッグ等で役に立つ．しかも，ある実行において実際にシーン内で発生したイベントの情報は，ハイブリッドスライスを求める際に有効に利用できる．

そのために試作したトレーサは，与えられたVRMLプログラムを解析し，そのプログラム中の各ROUTE文で中継するイベントを取得し出力するようなコードを付与し，その改造結果のVRMLプログラムを出力する．ここで自動的に付与されるコードは，VRML及びJavaScript言語からなる．以上の意味で，ここ

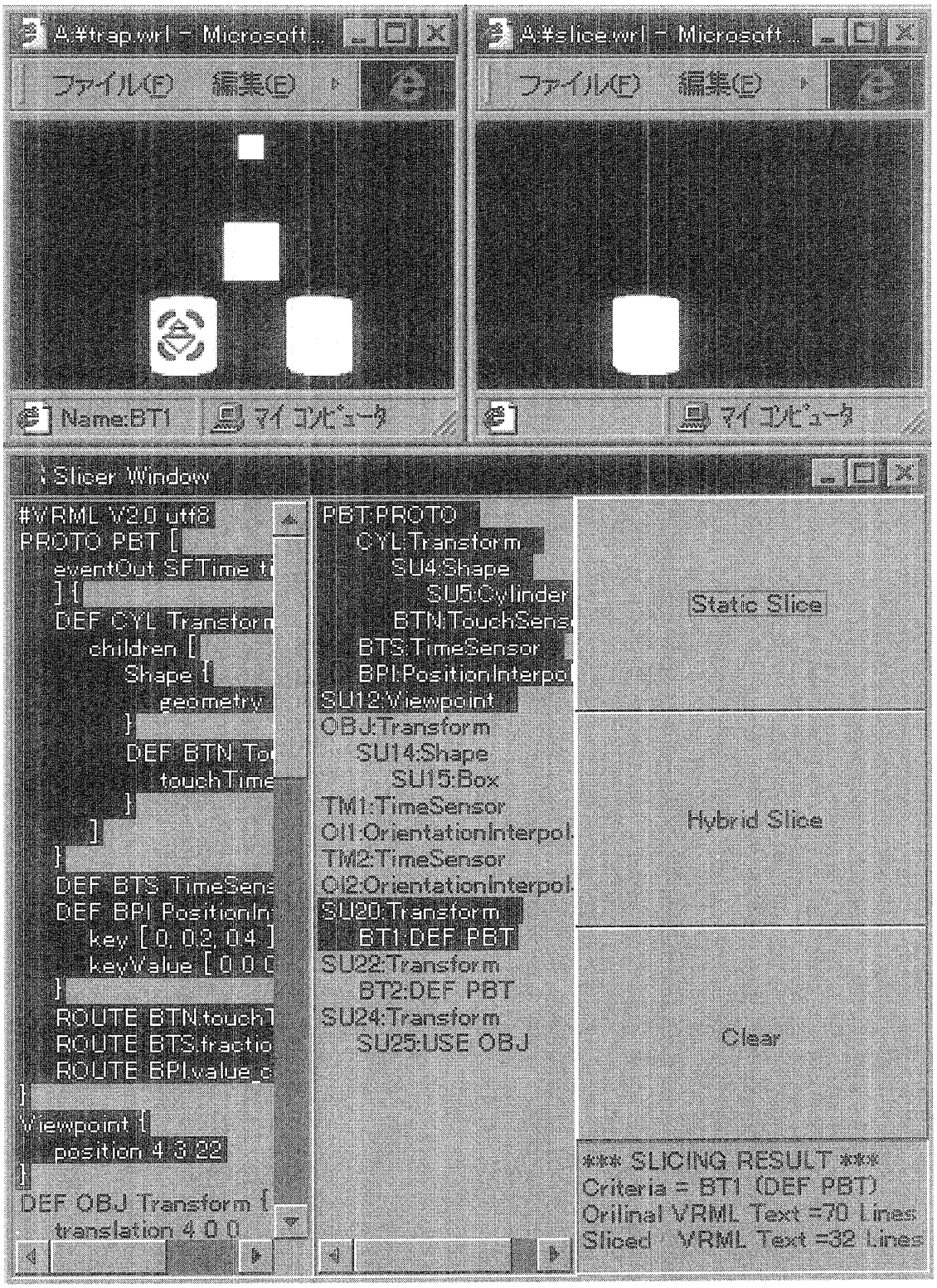

図 3 スライシング統合環境
Fig. 3 Integrated slicing environment.

で試作したのはトレーサというよりはトレース機能付与ツールと見なした方が良いのかもしれない.

改造された VRML プログラムを実行すると, VRML プログラムにおける標準出力に, 目的としたイベントの情報を, その発生のたびに実行時に取得し出力する. それは, ビューアの機能により, リアルタイムで画面で見ることもできるし, それをファイルに保存することもできる.

5.2 スライサとその利用

ここで試作したスライサは, VRML プログラム, スライシング基準 (ハイブリッドスライシングの場合はトレーサの出力も必要) を指定すると, 解析して VDG を構築した後, 基準の場所から VDG を探索し, スライス結果を VRML プログラムとして出力する.

このスライサは, 各ノードに関し, プログラマが DEF により指定した名前をつけていない場合は, 解析により自動的に各ノードに名前を付け, その結果を出力する機能を持つ. 本ツールでこの機能を実行した後, さらにスライス実行時には, スライシング基準における着目点を, そのような名前を用いて指定する.

5.3 スライシング統合環境

試作したスライサをエンジン部分として, さらに基準の指定やスライスを表示するための GUI を付与し, スライシング統合環境を構築した.

解析により自動的に付与された各ノードの名前を分かり易く表示するために, それらの名前を伴なうシーングラフ表示機能, および VRML ビューアにてマウスで示した物体の名前を表示する機能を付与している.

基準は, シーングラフ表示域における選択操作で指定でき, その後, 静的もしくはハイブリッドスライス実施のボタンを押すことでスライス抽出を開始できる. スライス結果は, プログラム全体を表示している領域にて, 該当部分を反転して示す. 同時に, シーングラフ表示域においても, 該当部分が反転する. さらに, 抽出されたスライスは VRML のビューアにて実行, 表示できる.

ここでは, 前述の VRML プログラム例において, 左のボタンを基準として指定して静的スライスを実施したスライシング統合環境の画面を図 3 に示す.

上部左側のビューアは, トレース出力機能とノード名表示機能とを持つように改造された VRML プログラムの実行域であり, ここでは, 左ボタンにマウスカーソルが移動されているため, ブラウザのステータスバーに, そのノード名 BT1 が表示されている. ハイブリッドスライシングの際, このビューアにおける実行に関するスライシングを行うこととなる.

上部右側は, 結果として抽出したスライスの実行域であり, 左ボタンのみが表示されているが, ここでこのボタンをマウスでクリックするとボタンが上下に動くため, ボタンの動作記述部分もスライスに含まれていることが分かる.

下部の大きなウィンドウの左側には, スライス対象プログラムのテキスト表示域があり, ここでは図 1 にて行頭に「*」記号がある行と同じ部分がスライスとして反転表示されている. 中央が, シーングラフ表示域であり, 同様に該当部分が反転している. 右側には, スライス開始のためのボタン群, およびスライス結果のプログラム行数等を報告する表示域もある.

このスライシング統合環境により, 着目点の指示や着目する動作の実行といった, スライシング基準の指定を容易に行うことができ, ハイブリッドスライシングの際の動作情報取得やその利用も自動的に行われ, プログラム中におけるスライス該当部分が視覚的に認識しやすくなった. そのためスライシング作業を効率的に行うことができた.

また, VRML におけるプログラミングにおいて, プロトタイピングのように段階的にプログラムを製作し, パラメタ値の決定やテストにより洗練していくプログラミング形態においては, その洗練作業をこのスライシング統合環境で支援できる. そのような意味では, これは VRML におけるプログラミング支援環境と捉えることもできる.

6. おわりに

我々は, VRML プログラム開発の効率を向上させるために, VRML にスライシングを導入することを提案した. スライシングを行うために VRML を分析し, 各種の依存関係を定義した. これに基づいて VRML におけるスライスの定義と導出法を与え, スライス生成ツールを作成し, 実際の業務での評価に基づいて, 本研究の有効性を述べた.

デバッギングに関しては, 誤った動作に影響した範囲をスライスにより限定できたので, アルゴリズミックデバッギング等により, 今後はさらにバグまで到達する手段も考えたい.

また, スライスとして抽出された部分は, それだけで独立して動く部品ともいえる. リバースエンジニアリングに基づき, 既存の VRML プログラムからそれらを部品としてライブラリ化し, 開発時にそこから必要な部品を流用する部品再利用に基づくプログラミングの方法を提示することも今後の課題である.

参考文献

1) The Virtual Reality Modeling Language Specification, *ISO/IEC DIS 14772-1* (1997).
2) 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).
3) Gupta, R. and Soffa, M. L.: Hybrid Slicing: An Approach for Refining Static Slices Using Dynamic Information, *Proceedings of the 3rd International Symposium on the Foundation of Software Engineering*, pp. 29–40 (1995).
4) Beeson, C.: An Object-Oriented Approach To VRML Development, *Proceedings of VRML97*, pp. 17–24 (1997).
5) Park, S. and Han, T.: Object-Oriented VRML For Multi-user Environments, *Proceedings of VRML97*, pp. 25–32 (1997).
6) Zhao, J., Cheng, J. and Ushijima, K.: Static Slicing of Concurrent Object-Oriented Programs, *Proceedings of the 20th IEEE Anuual International Computer Software and Applications Conference*, pp. 312–320 (1996).

付録 VRMLにおける依存関係の詳細

ここでは，本論文で提案している依存関係に関し，VRMLに関する若干の知識を必要とすると思われるが，それらの依存関係の方向等について，本文中の説明を補足するために記述する．

データ伝播関係

Quotation Dependence (QD) ノード内の属性で他ノードへのUSEによるポインタを持ち，そのノードの属性を読み出す場合，その参照されるノードから，それがUSE記述で複製されるノードへQDがある．

制御&データ伝播関係

Write Dependence (WD) ノードにて，他のノードを参照し，その属性に書き込むことのできる場合，そのUSE記述された複製ノード部から，参照されている複製の元となるノードへWDがある．

Route Dependence (RD) イベント送信元のノードの指定された属性からそれを中継するROUTE文へ，さらにそのROUTE文からイベント送信先のノードの指定された属性へRDがある．

実行制御関連

Choice Dependence(CD) SwitchノードもしくはLODノードに関し，その子ノードからそれらへCDがある．

Hit Dependence (HD) TouchSensorノードの兄弟ノードからそのTouchSensorとの間に双方向にHDがあり，Anchorノードの全ての子ノードからそのAnchorへHDがあり，Collisionノードの全ての子ノードからそのCollisionへHDがある．

Inline Dependence (ID) Inlineノードで読み込まれる部分的なシーンにおけるトップレベルの全要素から，そのInlineノードへIDがある．

プロトタイプ関連

Prototype Dependence (PD) PROTO句にて定義されている宣言部分からインスタンスのノードへとPDがある．さらに属性間の関係としてプロトタイプからインスタンスへのPDoutと，インスタンスからプロトタイプへのPDinがある．これらの関係は，EXTERNPROTO句のインタフェースを経由する場合もある．

Summary Dependence (SD) PDinからPDoutへと，プロトタイプ内部の各種依存関係をたどり，その推移にならい，そのインスタンスにおける属性間にSDがある．

Membership Dependence (MD) これは，プロトタイプ定義のインタフェースの各属性と，そのプロトタイプ定義内のノードでIS句で指定されている属性との間に双方向に存在する依存関係である．また，プロトタイプ定義内の先頭ノードからPROTOのインタフェースへもMDがある．

文法関係

Grammatical Dependence (GD) この依存関係は，VRMLプログラムであることを示すヘッダ行から，トップレベルの全要素への依存関係である．

シーンの見えかた関連

Appearance Dependence (AD) ノードの親子関係にて，親から子へADがある．また，アピアランスノードタイプ群，形状ノードタイプ群，形状特性ノードタイプ群のノード，もしくはShapeノードから，それらの親ノードへもADがある．

Light Dependence (LD) DirectionalLight以外の各光源ノードもしくはSoundノードから，全トップレベルへLDがある．DirectionalLightの場合は，そこからその全ての兄弟ノードへLDがある．

Bindable Dependence (BD) バインダブル

ノードタイプ群の全ノードもしくはWorldInfoノードから，全トップレベルへBDがある．

(平成11年 5 月 28 日受付)
(平成11年10月 13 日採録)

丸山 博史（学生会員）

昭和43年生まれ．平成4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し，現在サーバ管理システムの開発を担当．平成10年より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．VRMLやJava言語を用いた開発，プログラムスライシングなどの研究に従事．

荒木啓二郎（正会員）

福岡市生まれ．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科教授．(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などの研究に従事．情報処理学会，日本ソフトウェア科学会，ACM，IEEE CS等の会員．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．