РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
А В Т О М Е Т Р И Я
№ 6 1994

УДК 681.3.06

А. В. Романовский
(Новосибирск)

ЯЗЫК СМЕШАННЫХ ВЫЧИСЛЕНИЙ ДЛЯ ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ ТРЕХМЕРНЫХ СЦЕН И ИХ ДИНАМИКИ РЕАЛЬНОГО ВРЕМЕНИ

Предлагается язык для программирования визуальных моделей 3-мерных сцен и их динамики. Язык предоставляет средства для явного описания смешанных вычислений, которые вместе со специальными операторами, набором предопределенных типов данных, функций и процедур делают визуальное моделирование достаточно большого класса 3-мерных сцен более эффективным и непосредственным.

Зачем еще один язык? Основное преимущество предлагаемого здесь языка L– («Л-минус») перед другими языками, используемыми для тех же целей, заключается в предоставлении программисту больших возможностей для явного задания смешанных вычислений, что позволяет создавать более эффективное программное обеспечение для визуального моделирования 3-мерных сцен и их динамики.

В различных императивных языках есть средства для явного задания смешанных вычислений. Например, в языках Modula-2 и Oberon предполагается вычисление константных выражений во время компиляции, в C и C++ есть макросредства, в C++ введены "in line"-функции, в языке ADA существует механизм прагм. В дополнение к этому компиляторы выполняют ряд контролируемых пользователем оптимизаций программы.

Достигнут также прогресс в автоматическом частичном вычислении в языках типа Lisp [1], Prolog [2] и нетипизированных объектно-ориентированных языках [3]. Известен автоматический частичный вычислитель для подмножества языка C [4].

Выбирая из этих средств, программист часто ожидает получить более эффективную программу и фактически рассматривает исполнение программы в две стадии. Первая стадия представляет собой специализацию программы и получение остаточной программы. Вторая стадия является собственно исполнением остаточной программы, т. е. выполнением всех вычислений, оставшихся в программе после первой стадии.

Для ряда приложений чрезвычайно важным является выполнение возможно большего числа вычислений на первой стадии и перенос на вторую стадию тех действий, которые зависят только от внешних по отношению к задаче данных.

Однако, используя практические языки и компиляторы, программист не имеет достаточно возможностей для распределения вычислений между двумя стадиями, и даже при условии очень хорошей оптимизации, выполняемой компилятором, он вынужден изобретать довольно сложные и подчас искусственные алгоритмы для повышения эффективности программ. Что касается автоматических частичных вычислений, их результативность неоднократно была успешно продемонстрирована, но, насколько известно автору данной статьи, не для визуального моделирования в реальном времени.

Поскольку язык L– используется для написания программ, рассчитанных на специальную архитектуру аппаратного обеспечения, он в отличие от полностью автоматических частичных вычислителей дает программисту полный и детальный контроль над распределением времен исполнения программы и обеспечивает средства для программирования аппаратуры с более полным учетом ее особенностей. В этом языке разработчику предлагаются одинаковые возможности для программирования на обеих стадиях вычислений. Как на первой, так и на второй стадиях есть средства ввода/вывода и распределения динамической памяти, выражения с традиционными скалярными арифметическими операциями, 3-мерными векторно-матричными и другими операциями, необходимыми для визуального моделирования 3-мерных сцен и их динамики, операторы присваивания, условные операторы, операторы выбора варианта и т. п. так же, как и процедуры (функции), с параметрами и без параметров.

На L– можно запрограммировать, например, алгоритм нахождения простых чисел так, что все вычисления и вывод результатов будут выполнены на первой стадии либо простые числа могут быть вычислены на первой стадии, а их вывод отложен на вторую стадию. Можно также запрограммировать, чтобы этот алгоритм вместе с выводом простых чисел был выполнен на второй стадии. Конечно, для визуального моделирования в реальном времени нахождение простых чисел не значит столько, сколько значат, например, операции конструктивной геометрии над твердыми телами, расчет статической освещенности или предопределенной динамики. И все это в L– может быть исполнено на первой стадии, оставляя таким образом для второй стадии только отображение (вывод) результирующих граней.

В противоположность автоматическим частичным вычислениям программист, зная узкое место алгоритма и используя средства, предоставляемые L–, может планировать и распределять вычисления между стадиями наиболее эффективным образом. В определенном смысле это подразумевает технологию программирования, в которой из тысяч строк текста программы только небольшое число важно с точки зрения эффективности.

Излагаемая в общих чертах в данной статье схема специализации программ является более быстрой по сравнению с подходами, используемыми в настоящее время в исследованиях по частичным вычислениям, а именно, специализация L– программ производится не перед компиляцией и не в ее процессе, а после того, как программа откомпилирована и получен смешанный (объектный) код для двух соединенных в конвейер виртуальных компьютеров. Частичное исполнение этого кода на первом виртуальном компьютере дает в результате код для второго. Идеи, на которых основана эта схема, близки к идеям генерирующих расширений Ершова [5].

Другой оригинальный момент состоит в поддержке частичных вычислений не только синтаксически (аннотирование), но и семантикой языка, которая до некоторой степени предохраняет от написания незавершающихся смешанных программ. Ввиду этого проблема завершения, вместо «завершит ли частичный вычислитель для произвольного языка программирования свою работу», была сформулирована таким образом: «какую семантику должен иметь язык для предотвращения незавершающихся частичных вычислений».

И наконец, аннотирование переменных и параметров процедур так же, как и самих процедур, позволяет более легко оценивать эффективность специализированной программы и улучшать ее по сравнению с полностью автоматическими частичными вычислителями. Выбор ручного аннотирования также существенно поддерживается технологией программирования визуальных моделей 3-мерных сцен и их динамики. Для этого класса приложений некоторый набор шаблонов неаннотированных L– модулей (размером до 100 000 строк) может генерироваться автоматически. Затем примерно 50 критических строк в этих модулях аннотируются вручную.

Классическими статьями по смешанным вычислениям являются [5, 6]. Текущее состояние исследований по частичным вычислениям, определение

терминологии частичных вычислений и дополнительная библиография даны в [7].

Обзор средств языка L–. Большинство тем, относящихся к визуальному [illegible] текстурой и полупрозрачностью многоугольников Гуро в одном кадре на растре 512 × 512 пикселов с 24 битами цвета в каждом пикселе.

Модель 3-мерной сцены и ее динамики представляется на L– набором программных модулей. Результатом их исполнения являются последовательности кадров (фильм), соответствующих динамике моделируемой сцены в реальном времени.

Первая стадия (c1) исполнения L– модуля осуществляется не в реальном времени на любом процессоре общего назначения, после чего модуль становится частично исполненным. Затем исполняются вычисления второй стадии (c2) на аппаратном обеспечении системы синтезирующей визуализации, и тем самым модуль вычисляется полностью. Исполнение c2-вычислений может быть повторено несколько раз с различными внешними (входными) данными с получением в результате различных визуальных представлений моделируемой сцены.

Типы данных. В L– нет возможности определять новые типы данных, поскольку предопределенные типы данных считаются достаточными для моделирования большого класса 3-мерных сцен и их динамики. Набор типов данных языка L– представляет собой обобщение и развитие типов из [11, 12] и подробно описан в [13]. Предопределенные типы, существенные для последующего изложения, таковы:

— Integer и Real. Множества значений и операторы, применимые к объектам данных этих типов, такие же, как, например, для типов int и float в языке Си. Как и в Си, целый 0 соответствует ложному значению, а все остальные целые значения — истинному значению в логических выражениях;

— Lamp. Объекты данных этого типа представляют невидимые источники освещения в моделируемой сцене. Так, например, константа типа Lamp с именем sun используется вместо солнца;

— Reper. Объекты данных типа Reper задают правосторонние ортонормальные системы координат в 3-мерном пространстве. Эти системы координат могут образовывать древовидные иерархические структуры. Так, например, указатель на объект данных типа Reper с именем world соответствует абсолютной (мировой) системе координат, которая содержит все остальные системы координат моделируемой сцены;

— Viewer. Объекты данных типа Viewer используются для определения части сцены, отображаемой на экране дисплея;

— Faces. Объекты данных типа Faces — наборы плоских выпуклых односторонних полигонов (граней), которые аппроксимируют поверхности моделируемой сцены.

Для конструирования объектов-примитивов типа Faces имеются функции Box, Circle, Convex, Polygon, Revolved, Strip и Tube. К операндам типа Faces применимы мультипликативные операции конструктивной геометрии: "*" (пересечение), "/" (отсечение) — и аддитивные операции: "+" (объединение), "–" (разность). Для «заглаживания» граней с помощью одинаковых нормалей в смежных вершинах к операндам типа Faces применима одноместная операция "smo" и т. д.

Процедуры Show, ShowRanged применимы к объектам типа Faces и в результате отображают грани в окне текущего наблюдателя, если они видимы, т. е. находятся в пирамиде видимости текущего наблюдателя и повернуты в его сторону. Процедура Show отображает грани в той последовательности, в кото-

рой они находятся в наборе граней параметра процедуры. Процедура ShowRanged производит пространственное упорядочение граней для произвольного положения наблюдателя и отображает их, вызывая процедуру Show, в правильном порядке для алгоритма художника.

Определение данных. Так же как и в других типизированных языках, при определении данных на L– необходимо задать тип определяемых данных, их имена, принадлежность их к константам или переменным и, возможно, их начальные значения. В дополнение к этому данные в L–, которые планируется использовать в c1- или c2-вычислениях, должны быть явно помечены (аннотированы). Например, целые переменные v1 и v2, которые будут использоваться на стадиях 1 и 2 соответственно, должны быть определены следующим образом:

```
Integer v1
Integer @ v2
```

Лексема "@" во втором определении показывает, что переменная v2 будет использоваться в c2-вычислениях. Заданная один раз стадия использования данных не может быть изменена в дальнейшем, т. е. стадии использования так же, как и типы, являются неизменяемыми свойствами данных в L–. Данные, представляемые литералами: числа, строки и т. д., применяются в c1-вычислениях.

Процедуры. В L– также необходимо явно аннотировать процедуры, вызываемые во время c1- или c2-вычислений. Например, заголовки процедур с именами proc1, proc2, вызываемые соответственно на стадиях 1 и 2, выглядят так:

```
PROCEDURE proc1 ();
PROCEDURE @ proc2 ();
```

Определения формальных параметров процедуры такие же, как и определения данных. Заголовок процедуры с именем q1 и двумя формальными параметрами: неизменяемым параметром fp1 и переменным параметром fp2 — записывается следующим образом:

```
PROCEDURE q1 (Integer fp1#; Real @ fp2:);
```

Процедуры, вызываемые во время c1-вычислений, могут иметь как c1-, так и c2-параметры.

Процедуры, вызываемые во время c2-вычислений, должны иметь только c2-параметры:

```
PROCEDURE @ q2 (Integer @ fp1#; Real @ fp2:);
```

При вызове процедур стадии использования формального переменного параметра и фактического параметра должны совпадать. Для неизменяемого формального c2-параметра в качестве фактического параметра может использоваться c1- или c2-выражение.

Выражения. Выражения в L– синтаксически очень похожи на выражения в других языках:

```
1994
(i + j) * (i – j)
a[i + j] * a[i – j]
r * abs(sin(alpha))
```

Порядок вычисления выражения определяется приоритетом операций и с помощью скобок. Сначала вычисляется часть выражения, заключенная в скобки. В части выражения, не содержащей скобок, операции с большим

приоритетом исполняются прежде операций с меньшим приоритетом, а операции с одинаковым приоритетом выполняются слева направо.

Некоторые предопределенные операции, функции и процедуры могут вызываться как во время c1-, так и во время c2-вычислений. Бремя исполнения вызова зависит от стадии использования операндов (фактических параметров). Если все операнды предопределенной операции вычисляются на первой стадии, то эта операция исполняется на этой же стадии и ее результат может использоваться на первой стадии. Если один или оба операнда предопределенной операции относятся ко второй стадии, то такая операция исполняется во время c2-вычислений и ее результат может использоваться только на второй стадии.

Например, разрешается написать присваивание, в котором значение c1-выражения присваивается c2-переменной. Точнее, выражение будет вычислено во время c1-вычислений, полученный в c1 результат преобразуется (приведен) в c2-константу, и затем во время c2-вычислений, действительно, осуществится присваивание. Однако в языке запрещено записывать присваивание, в котором значение c2-выражения присваивается c1-переменной.

Операторы. В L– есть операторы общего назначения — присваивание, условный оператор, оператор выбора варианта, оператор цикла и т. п., а также несколько специальных операторов для программирования визуальных моделей 3-мерных сцен. Операторы, содержащие другие операторы, называются составными.

Любой оператор общего назначения синтаксически одинаков для обеих стадий исполнения и выполняется либо на стадии 1, либо на стадии 2. Рассмотрим для примера условный оператор:

```
IF усл_выраж DO ... END
```

В зависимости от состава условное выражение может быть вычислено во время стадии 1 или 2, которая будет также и стадией исполнения оператора. На этой стадии, если значение выражения истинно, то будут выполнены операторы между DO и END, иначе они не исполняются.

В языке L– есть два оператора общего назначения, которые, несмотря на их универсальность, не встречаются в других языках программирования. Это оператор исполнения по порядку и оператор сортировки и исполнения по порядку. Оба оператора используются в L–, кроме всего прочего, для программирования сцен, изображение которых строится по алгоритму художника.

Первый оператор выражает более естественным образом идею исполнения нескольких кластеров операторов либо в прямом, либо в обратном порядке. Например, оператор исполнения по порядку:

```
AT усл_выраж DO statements1 | statements2 | statements3 END
эквивалентен оператору
IF усл_выраж DO
  statements1; statements2; statements3;
ELSE
  statements3; statements2; statements1;
END
```

Оператор AT позволяет избегать ненужного повторения вложенных кластеров операторов. Число вложенных кластеров не ограничено.

Оператор сортировки и исполнения по порядку выражает идею упорядочивания нескольких кластеров операторов перед их исполнением. Например, оператор

```
SORT вещ_выраж1, вещ_выраж2, вещ_выраж3, вещ_выраж4 DO
  statements1 | statements2 | statements3 | statements4
END
```

означает, что после вычисления выражений типа Real вещ_выраж1, ..., вещ_выражN их текущие результаты сортируются (в неубывающем порядке). Затем кластеры операторов между DO END, т. е. statements1, ..., statementsN, исполняются по порядку, задаваемому результатом сортировки, который может не совпадать с явным порядком перечисления этих кластеров между DO и END. Число вложенных кластеров здесь также не ограничено.

Наиболее важными специальными операторами для программирования 3-мерных визуальных моделей являются:

— оператор начала кадра;

— оператор установки текущей системы координат;

— оператор назначения текущего списка источников освещения.

Эти операторы исполняются на стадии 2 и подразумевают существование модуля, который является частью любой программной модели 3-мерной сцены. Такой модуль называется DDDSCOPE («3-мерный контекст») и содержит несколько скрытых структур данных, доступ к которым осуществляется посредством упомянутых операторов. Это следующие структуры данных:

— Текущий наблюдатель сцены. Текущий наблюдатель устанавливается оператором начала кадра, т. е. если vv обозначает объект данных типа Viewer, то

VIEW vv

есть оператор начала кадра, исполнение которого начинает новый кадр и делает vv текущим наблюдателем. Когда все c2-вычисления в той последовательности, к которой принадлежит этот оператор, исполняются, текущий наблюдатель становится неопределенным. Изображение сцены для текущего наблюдателя является результатом всех вычислений в последовательности за данным оператором.

— Стек систем координат сцены. Система координат на вершине стека называется текущей системой координат. Она помещается на стек посредством оператора установки текущей системы координат, т. е. если ss обозначает объект данных типа Reper, то

SPACE ss

есть оператор установки текущей системы координат и его исполнение делает ss текущей, переопределяя таким образом систему координат, бывшую текущей перед этим. Когда все c2-вычисления в той последовательности, к которой принадлежит этот оператор, исполняются, переопределенная система координат снова становится текущей.

— Стек списков источников освещения. Список на вершине стека называется текущим списком. Список источников освещения помещается на вершину стека посредством оператора назначения текущего списка источников освещения, т. е. если L0 обозначает объект данных типа Lamp, то

NEW_LAMP L0

есть оператор назначения текущего списка источников освещения и его исполнение делает список из одного источника L0 текущим, переопределяя список источников освещения, бывший текущим перед этим. Когда все c2-вычисления в той последовательности, к которой принадлежит этот оператор, исполняются, переопределенный список источников освещения снова становится текущим.

Правила видимости имен для смешанных вычислений. Для того чтобы результат исполнения программы был независимым от распределения вычислений между стадиями, к программам на языке L– применяются два следующих правила видимости:

— только c1-константы из объемлющего контекста могут использоваться в c2-процедурах и только те c1-процедуры могут вызываться внутри c2-проце-

дур, в которых не используются прямо или косвенно (в других вызываемых c1-процедурах) переменные из объемлющего контекста;

— внутри составного c2-оператора ни одна определенная вне его переменная не может быть изменена прямо или косвенно (как побочный эффект вызова какой-либо c1-процедуры).

Для того чтобы избежать проблемы завершения c1-вычислений, к любому находящемуся внутри c1-процедуры составному c2-оператору применяется следующее правило видимости:

— внутри такого оператора не может быть вызвана какая-либо процедура, если это приведет к вызову процедуры, содержащей данный оператор.

Примеры программирования на L–. Изображения сцены (кадры), соответствующие программной модели 3-мерной сцены, строятся в результате исполнения L– модулей на стадии 2. Последовательности рекурсивных вызовов процедур и вложенность операторов соответствуют статической структуре сцены. Изменения значений переменных модуля на каждом кадре влияют на эти последовательности вызовов и исполнение операторов, представляя таким образом динамику сцены в виде динамики программы.

Например, для получения одного кадра простой сцены следует запрограммировать такие действия:

— устанавливается текущий наблюдатель;

— в зависимости от положения объекта или поверхности в моделируемой сцене устанавливается новая текущая система координат;

— если объект (поверхность) освещается, кроме солнца, другими источниками освещения, назначается новый текущий список источников освещения;

— при использовании предопределенных процедур Show (ShowRanged) грани, представляющие объект (поверхность), предъявляются текущему наблюдателю. Эти грани считаются расположенными в текущей системе координат. Грани переводятся из текущей системы координат в систему координат текущего наблюдателя. Невидимые грани отбрасываются, а видимые — отображаются в окне текущего наблюдателя в соответствии с их визуальными характеристиками (цвет, полупрозрачность, текстура и т. п.), условиями освещенности (содержимым текущего списка источников освещения) и свойствами среды, в которой находится сцена (туман, сумерки и т. д.).

Следующая процедура на L– описывает простую визуальную модель (один кадр) 3-мерной сцены:

```
PROCEDURE@ simple_scene (Reper@ ss)
DO
  VIEW vv;
  SPACE ss;
  NEW_LAMP LL;
  ShowRanged (box + tube);
END simple_scene
```

Здесь vv обозначает объект данных типа Viewer, LL — объект типа Lamp, box и tube — c1-объекты типа Faces, box + tube обозначает в смысле конструктивной геометрии операцию объединения твердых тел, которая вместе с вызовом ShowRanged () в этом примере выполняется на стадии 1, тогда как сама процедура simple_scene будет в действительности вызвана и исполнена на стадии 2.

Вызов этой процедуры в цикле с различными фактическими параметрами для ss даст в результате последовательность кадров:

```
FOR Integer i # 0 TO 59 DO
  simple_scene (path[i]);
END
```

Здесь path есть c1-массив, элементы которого имеют тип Reper и вычисляются на стадии 1 до этого оператора. Такой массив представляет анимационную траекторию для 60 кадров. Оператор цикла исполняется на стадии 1.

Пусть hill1, hill2, hill3 обозначают объекты данных, представляющие далекие холмы на местности, а to_hill1, to_hill2, to_hill3 являются данными типа Real, которые представляют расстояния от места наблюдения до «центров» соответствующих холмов. Тогда, чтобы использовать алгоритм художника для построения изображения граней холмов, необходимо упорядочить эти холмы в соответствии с их расстояниями до точки наблюдения. Так как холмы расположены вдалеке, достаточно их упорядочивания на стадии 1. Для этой цели подойдет оператор SORT:

```
SORT to_hill1, to_hill2, to_hill3 DO
```

Таким образом, в этих примерах выполнение операции объединения твердых тел, расчет анимационной траектории, осуществление итераций и сортировки на первой стадии уменьшает время исполнения на второй стадии.

Замечания по поводу реализации компилятора. Для упрощения последующего изложения предполагается, что структура L– компилятора близка к Pascal– («Паскаль-минус») компилятору, описанному в [14]. Так как L– предполагает две стадии исполнения программы, то компилятор генерирует код для двух виртуальных компьютеров (ВК1 и ВК2). Каждый компьютер представлен своим набором команд, которые имеют суффиксы 1 или 2 (например, Multiply1, Divide2), памятью для хранения кода (код1 и код2), стеком данных (стек1 и стек2) и тремя индексными регистрами: программным регистром (ПР1 и ПР2), стековым регистром (СР1 и СР2), базовым регистром (БР1 и БР2). Каждый программный регистр содержит адрес текущей (исполняемой) инструкции для соответствующего виртуального компьютера. В стековом регистре находится адрес верхушки соответствующего стека, а каждый базовый регистр используется для доступа к данным на стеке и содержит стартовый адрес записи активации исполняемой процедуры.

Память кода1 содержит инструкции для обоих виртуальных компьютеров. Сначала исполняются инструкции из кода1. Когда ВК1 встречает инструкцию1, он исполняет эту инструкцию в соответствии с ее семантикой и операндами. Когда ВК1 встречает инструкцию2, он помещает эту инструкцию вместе с операндами в код2. После исполнения кода1 код2 содержит все инструкции для ВК2, готовые для исполнения.

В стеках хранятся глобальные данные модуля, временные результаты исполнения операторов, локальные данные блоков, а также записи активации процедур. Стек1 содержит все данные для c1-вычислений и адреса в стеке2, стек2 — только адреса для c2-вычислений.

Адреса глобальных данных и смещения локальных данных блоков и параметров процедур в стеке1 вычисляются во время синтаксического разбора, адреса глобальных данных в стеке2 также вычисляются во время разбора, а смещения временных значений и локальных данных блоков в стеке2 вычисляются во время c2-вычислений.

Рассмотрим детали генерации смешанного кода на нескольких примерах.

Пример 1. Преобразование c1-данных в c2-данные.

Результат любого c1-выражения по мере необходимости преобразуется в c2-константу посредством инструкции Lift1To2, генерируемой компилятором. Эта инструкция исполняется во время c1-вычислений и помещает в код2 инструкцию Constant2 вместе с операндом, который берется с верхушки стека1. Инструкция Constant2, исполняемая во время c2-вычислений, помещает свой операнд на стек2.

Пример 2. Вычисление адресов для определяемых данных.

Обрабатывая определения данных, компилятор распределяет память для c1-данных на стеке1 и во время разбора вычисляет смещения данных на стеке. То же самое возможно сделать во время разбора для глобальных c2-данных или параметров c2-процедур. Не так для локальных c2-данных в блоках и для c2-параметров c1-процедур ввиду возможной рекурсии последних. Поэтому стек1 содержит не только c1-данные, но и смещения c2-данных на стеке2. Адреса на стеке1, по которым находятся значения этих смещений, вычисляются во время разбора. Таким образом, для доступа к этим данным необходимо двойное разыменование адреса. Первое разыменование выполняется во время c1-вычислений, а второе — во время c2-вычислений.

Пример 3. Передача параметров.

Если c1-процедура имеет c2-параметр, то он всегда передается в эту процедуру как смещение ячейки в стеке1, в которой находится адрес фактического параметра на стеке2. Если, например, c2-параметр является неизменяемым параметром, тогда соответствующая ячейка в стеке1 содержит адрес в стеке2, т. е. адрес некоторого временного результата c2-вычислений. Перед инструкцией завершения процедуры в конце данной c1-процедуры компилятор генерирует также ClearStack2 инструкцию для удаления временных значений для c2-параметров со стека2.

Перспективы. Для описания более универсальных визуальных моделей 3-мерных сцен и их динамики, используемых в авиационных и космических тренажерах, изготовлении анимационных фильмов и в других областях, рассматриваемый язык должен быть развит в направлении современных языков программирования общего назначения, а именно, планируется ввести в язык средства для определения новых типов, поддержку объектно-ориентированного программирования, обработку исключительных ситуаций. Объединение этих возможностей с возможностью явно задавать смешанные вычисления даст в результате мощный инструмент для эффективного программирования в тех областях, где автоматическая оптимизация и автоматические частичные вычисления не позволяют достичь нужного быстродействия.

Я выражаю благодарность моим коллегам И. В. Белаго, Ю. Ю. Некрасову, Ю. В. Тарасову, в дискуссиях с которыми концепции, представленные в данной статье, были уточнены и обобщены. Этим людям принадлежит большинство идей, относящихся к набору встроенных типов языка и связанных с ними алгоритмами визуализации 3-мерных сцен, но не изложенных в данной статье. И наконец, создав среду исполнения (для стадии2) L– программ, мои коллеги сделали возможным для меня написать L– компилятор и эмулятор виртуального компьютера, на котором L– программы исполняются на стадии 1.

Я благодарю также М. А. Бульонкова, оказавшего мне значительную помощь в овладении литературой по смешанным (частичным) вычислениям, и Ларса Оле Андерсена, задававшего полезные вопросы.

СПИСОК ЛИТЕРАТУРЫ

1. Khan K. M. A partial evaluator of Lisp programs written in Prolog // First International Logic Programming Conference /Ed. M. Van Caneghem.—Marseille, France, 1982.—P. 19.
2. Sahlin D. The mixtus approach to automatic partial evaluation of full Prolog // Logic Programming: Proc. of the 1990 North American Conference /Ed. S. Debray, M. Hermenegildo.—Austin, Texas: MIT Press, 1990.—P. 377.
3. Marquard M., Steensgaard B. Partial Evaluation of an Object-Oriented Imperative Language: Master's Thesis.—DIKU, University of Copenhagen, Denmark, 1992.
4. Andersen L. O. Partial evaluation of C and automatic compiler generation (extended abstract) // Compiler Construction /Ed. U. Kastens, P. Pfahler.—Berlin a.o.: Springer, 1992.—P. 251.—(Lecture Notes in Computer Science. Vol. 641).
5. Ershov A. P. On the essence of compilation // Formal Description of Programming Concepts /Ed. E. J. Neuhold.—Amsterdam: North Holland, 1978.—P. 391.
6. Ershov A. P. Mixed computation: potential applications and problems for study // Theoretical Computer Science.—1982.—18.—P. 41.

7. **Jones N. D., Gomard C. K., Sestoft P. et al.** Partial Evaluation and Automatic Program Generation: Draft, Typeset January 6, 1993.
8. **Foley J. D., Van Dam A.** Fundamentals of Interactive Computer Graphics: Reading, PA.—Addison-Wesley, 1984.
9. **Rogers David F.** Procedural Elements for Computer Graphics.—N. Y.: McGraw Hill, 1985.
10. **Magnenat-Thalmann N., Thalmann D.** Image Synthesis: Theory and Practice.—Berlin a.o.: Springer, 1987.
11. **Романовский А. В.** VML — язык описания геометрических моделей трехмерных сцен для синтезирующих систем визуализации // Автометрия.—1993.—№ 5.
12. **Белаго И. В., Романовский А. В., Тарасов Ю. В.** SDL — язык описания визуальных моделей трехмерных сцен // Там же.
13. **Белаго И. В., Некрасов Ю. Ю., Романовский А. В., Тарасов Ю. В.** Типы данных и операции для описания визуальных моделей трехмерных сцен и их динамики реального времени // Автометрия.—1994.—№ 6.
14. **Hansen B. P.** Brinch Hansen on Pascal Compilers.—N. Y.: Prentice Hall, 1985.

Поступила в редакцию 26 мая 1994 г.
