ным программам доступ к базе данных через специальный интерфейс ОС. Ядро СРВ — полностью инвариантное к особенностям задачи.

Заключение. Обратившись к рисунку, отметим особенность выбранного подхода к организации программного обеспечения ССВ. Такой подход можно назвать языковым в силу аналогии с традиционной организацией систем программирования на базе одного или нескольких языков. SDL играет роль входного языка, а ВЯЗ — объектного. DBG выполняет функции построителя задач (редактора связей), БДРВ является аналогом образа задачи, а СРВ — исполнительной системой.

В заключение авторы выражают благодарность руководителю проекта ССВ А. М. Ковалеву за полезные обсуждения.

ЛИТЕРАТУРА

1. **Ковалев А. М., Талыкин Э. А.** Машинный синтез визуальной обстановки.— Автометрия, 1984, № 4.
2. **Тренажерные** системы/Под ред. В. Е. Шушунова.— М.: Машиностроение, 1981.
3. **Буровцев В. А., Власов С. В., Вяткин С. И. и др.** Геометрический процессор синтезирующей системы визуализации.— Автометрия, 1986, № 4.
4. **Богданов В. В., Ковалев А. М., Нефедов И. Б. и др.** Канал видеопреобразования синтезирующей системы визуализации.— Там же.
5. **Ковалев А. М., Курочкин В. В., Тарнопольский Ю. В.** Трехпортовая память телевизионного кадра.— Автометрия, 1984, № 4.
6. **Талыкин Э. А.** Система проектирования печатных плат на ЭВМ семейства «Электроника».— Там же.
7. **Талыкин Э. А.** PED — графический редактор в системе проектирования печатных плат.— Автометрия, 1984, № 5.
8. **Иоффе А. В., Талыкин Э. А.** BUSIC — диалоговая система с языковым процессором для наладки и тестирования оборудования.— Там же.
9. **Иоффе А. В.** Тестирование, диагностика и наладка цифровых устройств с использованием иерархической схемы программных моделей.— Автометрия, 1986, № 4.
10. **Талыкин Э. А.** Внутренний язык для описания визуальных моделей.— Автометрия, 1985, № 4.
11. **Гусев А. В., Талыкин Э. А.** SDL — язык описания трехмерных сцен в системах динамической машинной графики.— Автометрия, 1986, № 4.
12. **Гусев А. В., Ивашин С. Л., Талыкин Э. А.** Математические модели сцен в синтезирующих системах визуализации реального времени.— Автометрия, 1985, № 4.

Поступила в редакцию 3 марта 1986 г.

УДК 681.3.06

А. В. ГУСЕВ, Э. А. ТАЛЫКИН

(Новосибирск)

SDL — ЯЗЫК ОПИСАНИЯ ТРЕХМЕРНЫХ СЦЕН В СИСТЕМАХ ДИНАМИЧЕСКОЙ МАШИННОЙ ГРАФИКИ

Введение. Графические языки и графические расширения распространенных языков программирования применяются в машинной графике уже достаточно давно. Например, Калсруд [1] еще в 1968 г. сформулировал основные черты графического языка, которые и сейчас не выглядят устаревшими. Определенным итогом работ данного направления являются графические стандарты [2, 3], регламентирующие некоторую концептуальную основу языковой обстановки для систем интерактивной машинной графики. В большинстве реализаций программы с графическими возможностями работают под управлением универсальной операционной системы и взаимодействуют с графическими устройствами через стандартный драйверный механизм, т. е. ничем не выделяются среди других пользовательских программ. Чтобы обеспечить динамику доста-

точно сложных изображений, при такой организации требуется наличие сверхпроизводительной вычислительной техники со сверхбыстрыми каналами [4].

При разработке систем динамической машинной графики, описанных в [5—9], применяется несколько иной подход. Модель сцены представляется совокупностью объектов: Некоторые из них могут иметь определенную степень подвижности. В сцене должен быть как минимум один наблюдатель, представляемый пирамидой видимости, заданной в системе координат одного из объектов. К одному объекту может быть отнесено несколько наблюдателей (например, модель нескольких окон летательного аппарата). В системе имеется несколько каналов видеопреобразования [9] (аппаратура для генерации отдельного изображения), на каждом из которых можно получить вид сцены для соответствующего наблюдателя.

В качестве средства описания визуальных моделей представляется язык SDL (Scene Description Language). В результате трансляции текст описания модели переводится во внутренний язык [6] и поступает в библиотеку моделей. На завершающем этапе подготовки описания конкретной сцены строится база данных реального времени [7].

В процессе генерации изображений работают специализированная операционная система и база данных визуальной сцены — ее внутренний ресурс. Весь комплекс процедур генерации изображений построен на операционном уровне. Прикладные программы обеспечивают реализацию движения объектов либо предоставляют специфичную для задачи и базы данных программную поддержку.

Как видим, суть предлагаемого подхода состоит в разделении процессов описания объектов сцены, манипуляциях над ними, построении базы данных, генерации изображений и в управлении динамикой [10]. В традиционном подходе графический язык обеспечивает возможность реализации всех этих элементов в одном контексте. Разделение функций направлено на повышение эффективности этапа генерации изображений за счет освобождения его от ряда трудоемких операций, отнесенных за пределы реального времени.

Язык SDL позволяет описывать объекты трехмерной сцены, конструировать более сложные объекты из простых, используя в качестве примитивов готовые модели из библиотеки. В языке отсутствуют исполнительные элементы, т. е. результат трансляции — только структура данных без какого-либо программного кода.

Схема трансляции в языке состоит в получении древовидной структуры описания визуальной модели [6, 7] путем частичной свертки синтаксического дерева, представленного SDL-текстом.

Лексика языка. SDL-текст строится из идентификаторов, служебных слов, литералов, знаков операций и разделителей.

Идентификатор представляется последовательностью букв или цифр, начинающейся с буквы. Служебные слова являются зарезервированными идентификаторами и могут использоваться в тексте только в строго определенном синтаксисом языка смысле.

В данной статье служебные слова будут выделяться полужирным шрифтом. Приведем список служебных слов:

at	**do**	**for**	**model**	**string**
begin	**else**	**global**	**od**	**then**
blink	**end**	**if**	**of**	**to**
boolean	**external**	**insert**	**put**	**tup**
call	**face**	**integer**	**real**	**vector**
case	**fi**	**macro**	**sort**	**using**
colour	**fire**	**matrix**	**sphere**	
division	**fo**	**mobile**	**step**	

Литералами изображаются целые и вещественные значения, а также литерные строки, например

5 129 3.59E-10 'строка'

Знаки операций и синтаксические разделители показываются одной или двумя литерами. Некоторые операции обозначаются служебными словами. Ниже приведен список обозначений операций и синтаксических разделителей:

(	)	[	]
(*	*)	(*	*)
:	;	.	,
,	/	\	!
⊗	△	#	%
→	←	:=	//
+	−	*	**
=	><	>	<
<=	>=	"	not
or	**and**		

Любая последовательность символов, заключенная в парные ограничители (* и *), является комментарием. Допускается неограниченная вложенность комментариев. К комментариям относится также остаток строки, начиная с символа %. Строка, начинающаяся парой символов %%,— прагмат. Прагматы содержат информацию для транслятора и управляют режимами его работы. Единственный прагмат, отражающийся на результате трансляции:

%% **include** *спецификация файла*

— позволяет включить в данном контексте целиком текст другого файла.

Основные объекты и операции. В языке имеется шесть типов данных: логические (**boolean**), целые (**integer**), вещественные (**real**), векторные (**vector**), матричные или операторные (**matrix**), грани (**face**).

Переменные логического типа принимают два возможных значения, обозначаемые **true** и **false**. Числовые типы (целый и вещественный) также имеют традиционную нотацию.

Векторные значения используются для обозначения векторов или точек в трехмерном пространстве. В языке существует конструктор векторных значений

$$\backslash ex : ey : ez \backslash,$$

где ex, ey, ez — выражения, вырабатывающие числовые значения компонент вектора. Конструктор для векторных значений играет роль литерала, но не является элементом лексического уровня (компоненты могут быть выражениями самого общего вида; отдельные поля конструктора могут разделяться пробелами и комментариями).

Операторные значения представляют собой матрицы аффинных либо линейных преобразований в трехмерном пространстве. Для операторных значений также имеется конструктор вида

$$\begin{matrix} \|e_{11}, & e_{12}, & e_{13}\| \\ \|e_{21}, & e_{22}, & e_{23}\| \\ \|e_{31}, & e_{32}, & e_{33}\|, \\ \|e_{41}, & e_{42}, & e_{43}\| \end{matrix}$$

где e_{ij} — выражения, вырабатывающие числовые значения.

Все перечисленные типы данных имеют внутриязыковое назначение, т. е. используются для организации вычислительного процесса внутри языкового контекста. Шестой тип данных — грань; кроме того, он может являться элементарным объектом результата трансляции (свертки) программы.

Конструктор для грани представляется в виде последовательности вершин, заключенной в квадратные скобки. Ей могут предшествовать признак детали (?) и (или) признак табличного задания грани (⊗). Вслед

за списком вершин в конструкторе грани могут быть указаны ее цвет и его интенсивность.

Признак детали разрешает не отображать грань в случае возникновения информационных или вычислительных перегрузок при генерации изображения. Табличное задание позволяет представлять вершины граней ссылками на таблицу, где хранится полная информация о вершинах. Таблицы вершин создаются для объекта или его части. Наряду с экономией памяти, табличное представление позволяет экономить вычислительные ресурсы на преобразования.

Указание цвета грани выделяется символом (#), а интенсивности — (△). Цвет может быть задан: номером (0..255) из некоторой внешней таблицы цветов; непосредственным определением трех компонент — красной, зеленой и синей; внешним именем, записываемым идентификатором, заключенным в апострофы. Компоненты цвета задаются выражением, вырабатывающим числовое значение, и разделяются символом (:). Конкретное значение цвета, заданного номером или внешним именем, определяется при окончательном построении базы данных.

Каждая из вершин грани задается векторным значением положения точки, за которым может следовать векторное значение нормали поверхности в точке, выделенное символом (!), а также индивидуальное значение интенсивности в точке, выделенное символом (△). Приведем примеры конструкторов граней:

$$\otimes\, ?\,[V1, \ldots, Vn] \# 1:0:0 \triangle 0.8$$

$$[\backslash 1:0:0\backslash, \backslash 0:0:1\backslash, \backslash 0:1:0\backslash] \# 5$$

$$?\otimes [V1!N1 \triangle i1,\ V2!N2 \triangle i2,\ \ldots,\ Vk!Nk \triangle ik] \# \text{colour} \triangle i$$

Выражение в языке имеет традиционную скобочную структуру. Первичными поставщиками значений в выражении являются литералы, конструкторы значений, переменные, вызовы стандартных функций. Наивысший приоритет имеют одноместные операции: (+) — одноместный плюс, (−) — одноместный минус и **not** — логическое отрицание.

Наивысшим приоритетом среди двухместных обладают мультипликативные операции: умножение (*), деление (/) и векторное умножение (**). Произведение числовых аргументов дает числовой результат, причем если хотя бы один из сомножителей вещественный, то результат вещественный, а иначе — целый. Произведение числа на вектор дает вектор (растяжение вектора), произведение (*) двух векторов — число (скалярное произведение), произведение вектора на матрицу — вектор (преобразование исходного вектора), а двух матриц — матрицу (композицию преобразований). Операция деления определена только на числовых аргументах. Векторное произведение (**) имеет общепринятый смысл.

Следующими по старшинству идут аддитивные операции: сложение (+) и вычитание (−). Аддитивные операции применимы к числовым и векторным аргументам.

Операции сравнения (<), (>), (=), (><), (>=), (<=) определены на числовых аргументах и вырабатывают логический результат.

Логические операции (**and** и **or**) обладают самым низким приоритетом.

Встроенные функции в SDL включают такие традиционные, как sin, cos, abs, sqrt, arctan, exp, ln, round, trunc. Кроме того, существует ряд специализированных функций, например: $\mathrm{norm}(v)$ — норма вектора v, $\mathrm{ort}(v)$ — нормирование вектора v; $\mathrm{rot}\,x(\varphi)$, $\mathrm{rot}\,y(\varphi)$, $\mathrm{rot}\,z(\varphi)$ — матрицы поворотов на угол φ вокруг соответствующей оси координат; $\mathrm{scale}(a, b, c)$ — матрица растяжения вдоль осей x, y и z с соответствующими коэффициентами a, b и c; $\mathrm{move}(v)$ — матрица переноса на вектор v.

Всякая переменная в языке описывается до первого использования с помощью соответствующего описателя типа (одного из шести перечисленных выше). Для изменения значений переменных имеется оператор присваивания, записываемый традиционным для алголоподобных языков

образом. Значение может присваиваться переменной непосредственно при ее описании. Приведем несколько примеров:

```
integer k, n := 100, l;
real x, y, z, h := 5.5, s, fi := pi/(n * 180);
boolean b1, b2;
matrix m, t := rotz(pi/4) * move(\0 : 0 : h\);
vector v, u;
. . .
x := h * cos(fi); y := h * sin(fi);
s := norm(u ** v); u := u * t;
```

Над гранями определены две операции конкатенации: (→) — конкатенация слева и (←) — справа, которые совмещаются с операцией присваивания. Например:

$$F \leftarrow E1 \leftarrow E2;\ F \rightarrow E3;$$

Здесь F — переменная типа «грань», на которой производится операция и фиксируется результат; $E1$, $E2$ и $E3$ — конструкторы граней либо идентификаторы переменных типа «грань». Действие операций конкатенации поясняет следующий фрагмент:

```
face F := [v1, v2];
F ← [v3] ← [v4, v5];   %F = [v1, v2, v3, v4, v5]
F → [va, vb] ← [v6];   %F = [va, vb, v1, v2, v3, v4, v5, v6]
```

Управляющие конструкции служат для организации вспомогательных вычислений в процессе свертки текста либо для управления генерацией элементов выходной последовательности.

Блок представляет собой последовательность предложений языка, заключенную в парные ограничители **begin** и **end**. В блоке локализуются описания языковых объектов.

Условное предложение имеет традиционный синтаксис и обеспечивает возможность выбора одной из двух альтернатив в зависимости от логического значения, вырабатываемого *выражением*. Условное предложение записывается следующим образом:

if *выражение* **then** *последовательность предложений*
else *последовательность предложений*
fi

Вторая альтернатива может опускаться:

if *выражение* **then** *последовательность предложений* **fi**

Цикл позволяет организовать итеративную последовательность повторяющихся вычислений; он записывается следующим образом:

for *идент* := *выражение*
[**step** *выражение*] **to** *выражение*
do *последовательность предложений* **od**

Семантика конструкции традиционная, раздел **step** может быть опущен, и тогда шаг цикла принимается равным единице.

Операторы генерации описывают некоторые элементы визуальной модели. В процессе свертки SDL-текста операторы генерации перерабатываются во вполне определенные элементы внутреннего языка [6]. Все другие конструкции SDL могут оказывать косвенное влияние на результат трансляции: например, оператор присваивания изменяет значение переменной, которая далее может быть использована в качестве поставщика значения для соответствующего поля выходной структуры данных.

Элементарный геометрический объект описания визуальной модели — плоский многоугольник (грань), которому соответствует в языке тип **face**. Оператор размещения грани позволяет ввести новую грань в структуру описываемой модели. Оператор задается либо идентификатором переменной типа «грань», либо непосредственно конструктором для грани, например:

$$F;\ [v1, v2, v3];$$

Основной способ конструирования объектов из деталей — описание деталей в собственной системе координат с последующей «сборкой», т. е. размещением деталей в определенных положениях в системе координат объекта. Для реализации операций сборки служит оператор погружения, который записывается следующим образом:

using *выражение*

put *последовательность предложений* **tup**

Здесь *выражение* вырабатывает значение матрицы, задающей преобразование из системы координат детали в систему координат объекта. Деталь описывается *последовательностью предложений.*

Для обеспечения баланса между сложностью изображения и объемом вычислений, необходимых для его генерации, в структуре описаний визуальных моделей имеется механизм охватывающих сфер [7]. В языке есть соответствующая конструкция:

sphere (*центр, радиус*)

of *последовательность предложений* **fo**

Здесь *центр* — выражение, вырабатывающее векторное значение, а *радиус* — числовое. *Последовательность предложений* описывает объект, геометрический образ которого должен целиком лежать внутри сферы. Спецификация сферы может быть опущена, и тогда охватывающая сфера будет определена на одном из этапов построения базы данных реального времени. Во время генерации изображения анализ положения сферы может значительно сократить объем вычислений. Например, сфера может не попасть в изображение, и тогда не потребуется тратить ресурсы на обработку расположенных внутри нее объектов.

При построении визуальных моделей объектов реальной обстановки бывает целесообразным использовать различную степень детальности описания в зависимости от расстояния между объектом и наблюдателем. Для поддержки этого механизма в языке существует специальная конструкция:

case (*центр, радиус*)

of *последовательность предложений*

: *метка детализации* : *последовательность предложений*

. . .

: *метка детализации* : *последовательность предложений*

fo

Здесь выбор детализации производится путем оценки размера на изображении габаритного шара, заданного *центром* и *радиусом.* В системе принято восемь различных уровней детализации (1—8). *Последовательность предложений*, не помеченная *меткой детализации*, отображается на всех уровнях детализации, а остальные только на уровнях, задаваемых меткой. *Метка детализации* задается либо числом, определяющим конкретный уровень детализации, либо двумя числами, разделенными символом (≫), и тогда помеченная последовательность предложений относится к определяемому диапазону уровней детализации.

Как было отмечено выше, основное средство для конструирования объектов из деталей в языке обеспечивается оператором погружения. В результате сборки базы данных погружения могут сворачиваться, т. е. описания деталей могут преобразовываться в систему координат объекта. Это возможно, если матрица погружения не зависит от времени, и необходимо для сокращения объема вычислений в процессе генерации изображений. Для описания подвижных деталей или объектов служит оператор «подвижного» погружения:

mobile *идентификатор*

of *последовательность предложений* **fo**

Оператор запрещает свертку матрицы погружения объекта, описываемого *последовательностью предложений*, а *идентификатор* обеспечивает доступ к матрице для прикладных программ. Матрица погружения здесь — текущая матрица преобразования.

Задача удаления невидимых поверхностей в системах отображения

приоритетного типа решается с использованием механизма разделяющих плоскостей [7]. В языке есть конструкция (раздел), позволяющая отделять объекты в пространстве плоскостями. Текстуально последовательность предложений разделяется по следующей синтаксической схеме:

открывающий разделитель *последовательность предложений*
раздел *последовательность предложений*
...
последовательность предложений
закрывающий разделитель

В качестве замыкающих разделителей здесь могут использоваться пары **begin...end, of...fo, put...tup.** Каждый *раздел* отделяет расположенную перед ним последовательность предложений от всей последующей части конструкции до закрывающего разделителя. Раздел записывается в одной из следующих форм:

division (*грань*)
division (*нормаль, точка*)
division

В первом случае разделяющая плоскость проходит через *грань*, во втором — ортогонально вектору *нормали* через *точку*, а в третьем случае спецификация плоскости не указывается, т. е. плоскость будет определена позже в процессе построения базы данных.

Процедурные элементы в языке служат для определения типовых компонент конструирования трехмерных объектов. Они могут реализовываться открытой подстановкой (макросы) и закрытым вызовом (модели).

Макрос представляет собой параметризованный блок или строку символов языка, полученную в результате лексической свертки и не связанную никаким синтаксисом. На основе макросов возможны практически любые текстовые подстановки. В случае когда макрос описывает блок, он чаще всего имеет определенный геометрический смысл и представляется формально как параметризованное описание объекта, которое при каждом вызове раскрывается целиком. Поскольку не определен синтаксис тела макроса, то не определен и контекст его вызова, который может быть любой синтаксической позицией. Синтаксис макроопределений выглядит следующим образом:

macro *макроопределение, ..., макроопределение*;

Макроопределение имеет следующую структуру:

имя (*формальные параметры*) = *макротело*

Имя макроса представляется идентификатором, *формальные параметры* — списком идентификаторов, разделенных запятыми, а *макротело* — синтаксическим блоком либо произвольной строкой символов, не содержащей запятых и точек с запятыми. Для определения в качестве строки макротела произвольного набора символов она может заключаться в специальные макроскобки (* и *). Вызов макроса в зоне действия описания задается именем, за которым может следовать список фактических параметров, разделенных запятыми. Фактические параметры раскрываются в контексте их вызова.

Если некоторый геометрический объект описан в виде макроса, то он будет раскрыт столько раз, сколько есть вызовов, что может привести к неоправданным повторениям в базе данных. В языке имеется конструкция, позволяющая описать один раз некоторую модель, которая может размещаться неоднократно в различных местах сцены. Модель, используемая для построения сцены, не обязательно должна описываться в пределах того же SDL-текста; она может быть внешней по отношению к тексту, что позволяет создавать библиотеки моделей. Внешние модели, используемые в данном тексте, должны быть специфицированы следующим описанием:

external model *имя, ..., имя*;

Описание моделей выглядит следующим образом:

model *имя* = *блок*,

. . .

имя = *блок*;

Модели, к которым разрешается доступ из других текстов, определяются с помощью описателя **global model**. Вызов объекта, описываемого моделью, производится оператором вызова

call *имя*;

При желании модель может быть раскрыта целиком, что производится оператором открытого вызова

insert *имя*;

Следует отметить, что макровызов реализуется на этапе трансляции, открытый вызов — на этапе сборки базы данных, а вызов модели — в процессе генерации изображений (в реальном времени).

Заключение. Представленный язык, основные черты которого были определены в 1978—1979 гг., прошел несколько этапов модификации. Существующую версию не считаем окончательной и предполагаем продолжать работу по ее совершенствованию для применения в различных системах отображения трехмерных объектов и САПР. Следует отметить вклад Б. С. Новикова и Е. Г. Юрашанского, принимавших участие в первой реализации языка.

ЛИТЕРАТУРА

1. **Kalsrud H. E.** A general purpose graphic language.— CACM, 1968, v. 11, N 4.
2. **Status** Report of the Graphics Standards Planning Committee.— Computer Graphics, 1979, v. 13, N 3.
3. **Enderle G., Kansu K., Pfalf G.** Computer draphics programming. GKS — the draphics standard.— N Y.: Springer — Verlag, 1984.
4. **Demos G., Brown M. D., Weinderg R. A.** Digital scene simulation: The synergy of computer technology and human creativity.— Proc. IEEE, 1984, v. 72, N 1.
5. **Ковалев А. М., Талныкин Э. А.** Машинный синтез визуальной обстановки.— Автометрия, 1984, № 4.
6. **Талныкин Э. А.** Внутренний язык для описания визуальных моделей.— Автометрия, 1985, № 4.
7. **Гусев А. В., Ивашин С. Л., Талныкин Э. А.** Математические модели сцен в синтезирующих системах визуализации реального времени.— Там же.
8. **Буровцев В. А. и др.** Геометрический процессор синтезирующей системы визуализации.— Автометрия, 1986, № 4.
9. **Богданов В. В. и др.** Канал видеопреобразования синтезирующей системы визуализации.— Там же.
10. **Гусев А. В., Ивашин С. Л., Иоффе А. В., Талныкин Э. А.** Программные компоненты синтезирующих систем визуализации.— Там же.

Поступила в редакцию 3 марта 1986 г.