Объектно-ориентированная инфраструктура систем компьютерной графики

Н.Б. Дерябин, Е.Ю. Денисов

Институт прикладной математики им. М.В.Келдыша РАН, Москва, Россия

dek@keldysh.ru, eed@spp.keldysh.ru

Аннотация

Рассматривается внутренняя программная инфраструктура сложных систем компьютерной графики, разработанных в Институте прикладной математики им. М.В.Келдыша РАН.

В основу инфраструктуры положен ряд оригинальных решений. Особое внимание уделяется возможностям расширения графических систем с минимизацией трудозатрат на разработку.

Ключевые слова: системы компьютерной графики, расширяемость, подключаемые компоненты, объектно-ориентированная инфраструктура.

1. ВВЕДЕНИЕ

По роду деятельности наш коллектив связан с построением сложных систем компьютерной графики для задач проектирования, дизайна, научных исследований [1].

Область приложения таких графических систем постоянно растет и усложняется. На начальных этапах нашей работы графические системы применялись для таких, сегодня классических, областей, как синтез реалистических изображений и компьютерное моделирование освещенности (рис. 1а). В настоящее время стали актуальными такие приложения как моделирование с использованием сложных оптических элементов (например, жидкокристаллических дисплеев или светодиодов) и использование волновой оптики. Для примера на рис. 1б приведены результаты моделирования оптически сложных красок в сочетании с использованием изображения с большим динамическим диапазоном яркостей как источника освещения. На рис. 2 показана визуализация композиции микрочастиц, моделирующих объемный рассеивающий слой; эта модель используется для расчета оптических характеристик слоя с использованием методов волновой оптики.

(а)

(б)

Рис. 1: Некоторые применения сложных систем компьютерной графики.

Все это накладывает серьезные требования к инфраструктуре (внутренней программной организации) систем компьютерной графики. Первейшим требованием к инфраструктуре является расширяемость и минимизация трудозатрат на расширение системы для новых приложений. «Голых» средств языка программирования оказывается недостаточно, и требуется наработка специального внутреннего инструментария. Соответствующие программистские решения были заложены в основу архитектуры разрабатываемых нами систем компьютерной графики. Им и посвящена настоящая статья.

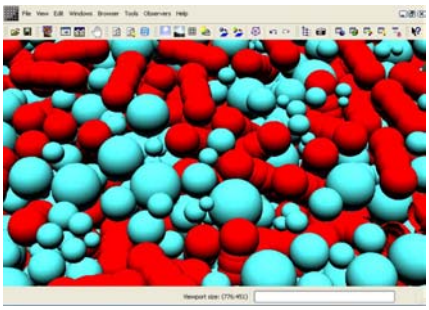

Рис. 2: Моделирование оптических свойств объемного рассеивающего слоя с использованием методов волновой оптики.

При разработке инфраструктуры были проанализированы имеющиеся решения, в частности 3D Studio MAX SDK [2] и Open Inventor [3]. Очень полезными оказались некоторые идеи языка C# (особенно концепция отражения) [4] и архитектуры .NET [5].

Языком разработки является C++.

2. БАЗОВЫЙ УРОВЕНЬ

Базовый уровень средств разработки включает в себя следующие общеупотребительные компоненты:

- базовые математические библиотеки для работы с векторами, матрицами, простейшими геометрическими объектами
- управление строками, массивами, файлами и т.п.
- управление объектами
- управление процессами и потоками
- управление подключаемыми компонентами

Помимо всего прочего, базовый уровень инкапсулирует в себе зависимости от конкретной программно-аппаратной платформы.

Важнейшей функциональностью базового уровня является управление объектами. Сложность систем компьютерной графики предъявляет большие требования к управлению объектами, в частности:

- расширяемость (иерархия классов и типов)
- динамическая типизация объектов
- управление памятью (сборка мусора)
- сериализация иерархии объектов
- механизм уведомления о событиях

Родные объекты в данном инструментарии называются **паевыми объектами** (plugs). Классы паевых объектов наследуются (прямо или опосредованно) от базового класса Plug.

Паевые классы образуют иерархию типов по отношению наследования. Каждый такой тип может быть представлен явно специальным объектом-типом. **Объект-тип** расширяет функциональность паевых объектов методами, общими для экземпляров данного типа, но, в отличие от статических методов C++, допускает динамическую адресацию типов. Сами по себе типы также являются паевыми объектами (в частности, могут иметь свои типы).

Для управления паевыми объектами используются **паевые ссылки**. Паевая ссылка являются «умным» указателем на объект. Объект «живет» до тех пор, пока на него указывает хотя бы одна паевая ссылка. Можно использовать и массивы (списки) паевых ссылок.

Паевые ссылки используются для построения сложных древовидных иерархий объектов, таких как сцена. Типичная сцена может включать тысячи паевых объектов разных типов, связанных паевыми ссылками. Независимо от сложности, такая сцена может быть записана в файл на диске (или в линейный массив памяти), а затем считана обратно при помощи единственного акта сериализации. При этом каждый паевой класс должен позаботиться только о сериализации своих членов-данных.

В отличие от обычных «умных» указателей, паевые ссылки поддерживают механизм обращения, т.е. позволяют при определенных условиях находить ссылающийся объект по объекту, на который он ссылается (рис. 3):

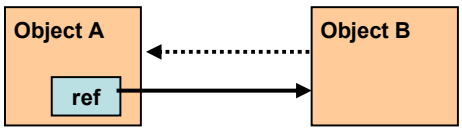

Рис 3: Обращение паевых ссылок.

Для этого объект, содержащий ссылку, должен объявить себя владельцем этой ссылки при помощи специального метода

```
Ref.SetOwner(this);
```

Теперь владельца ссылки на данный объект можно найти, зная тип этого владельца:

```
obj_a = obj_b->GetRefOwner(TClassA);
```

Можно также получить список всех владельцев указанного типа.

Этот механизм очень важен для сложных графических систем. Например, он позволяет находить, какие конретно объекты сцены используют данный материал (атрибуты поверхности или объема).

Механизм обращения ссылок интенсивно используется в самой инфраструктуре. Так, на его базе построен механизм уведомления о событиях, также чрезвычайно важном для сложных графических систем имеющих дело с большими и динамически меняющимися структурами данных (сценами).

Чтобы получать уведомления о событиях от объекта B, объект A просто должен владеть паевой ссылкой на объект B. Теперь при возбуждении событий объектом A будет вызываться метод обработки событий в объекте B.

3. УРОВЕНЬ МОДЕЛИРОВАНИЯ

В графических приложениях приходится поддерживать очень сложную модель данных - сцену, состоящую из многих различных объектов, связанных определенными иерархическими связями. Сцена может динамически меняться путем добавления, удаления или замены некоторых объектов в иерархии (например, смена вида поверхности), или изменения существующих объектов в иерархии путем применения к ним некоторых действий (например, включить или выключить источник света). В процессе эволюции добавляются новые и новые типы объектов сцены.

Несмотря на то, что число типов объектов, составляющих сцену, потенциально бесконечно (разные типы геометрии, внешнего вида, источников света и т.п.), все они должны иметь некоторую общую функциональность. Так, все классы, моделирующие сцену, должны обеспечивать решение следующих общих задач:

- обеспечение однородного иерархического представления сцены, так чтобы вся сцена могла быть визуально отображена в виде дерева с поименованными вершинами (объектами сцены) и поименованными ребрами (связями между объектами);

- обеспечение однородного способа получения списка возможных действий над объектами сцены и выполнения этих действий. Например, конечный пользователь может щелкнуть мышкой на каком-нибудь объекте в окне с визуальным изображением сцены, чтобы получить контекстное меню со списком действий, применимых именно к этому объекту;

- поддержка операций копирования / вставки и перетаскивания объектов при помощи мыши;

- возможность отмены выполненных действий и повторное выполнение отмененных действий (undo / redo);

- возможность работы со сценой с помощью языка сценариев (script language).

Для достижения этих целей было введено понятие **целевого объекта** – объекта, составляющего часть целевого представления сцены. Классы целевых объектов наследуются (прямо или опосредованно) от специального паевого класса Entity.

Для целевого объекта определяются свойства и действия. **Целевое свойство** представляет собой поименованную связь между объектами, представляющими сцену; оно определяется на базе паевой ссылки (списка ссылок) – члена целевого класса. **Целевое действие** – это поименованная процедура (функция) с заданным набором параметров,

применимая к данному целевому объекту; она определяется на базе конкретного метода целевого класса.

Целевые действия, как правило, реализуются как операции над целевыми свойствами (например, заменить значение данного свойства новым объектом). При соблюдении этого условия, автоматически поддерживается возможность отката / повторного выполнения целевых действий.

Принципиально важной является возможность работы со свойствами и действиями исключительно через интерфейс базового класса Entity. Это обеспечивает применимость унифицированных компонент пользовательского интерфейса (например, иерархическое представление сцены) и процессора языка сценариев как к существующим, так и будущим (еще не разработанным) целевым объектам и классам. Таким образом, введение нового целевого класса автоматически расширяет возможности приложения на уровне конечного пользователя.

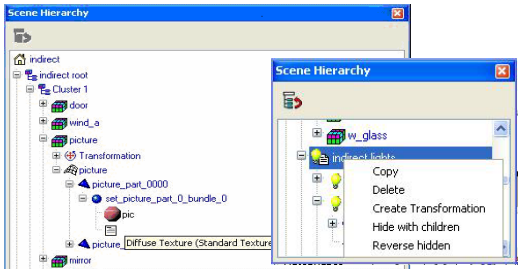

Рис. 4: Иерархия сцены.

4. ПРЕДСТАВЛЕНИЕ СЦЕНЫ

Расширяемость графических приложений потребовала введения различных представлений сцены внутри приложения. Мы используем три вида представления сцены:

- **Целевое представление**. Это представление, ориентированное на конечного пользователя. Оно использует уровень моделирования, описанный в предыдущем разделе. (Вообще говоря, возможно использование различных целевых представлений для одних и тех же сцен.)

- **Каноническое представление**. Это унифицированное представление сцены, содержащее всю информацию о сцене в некотором каноническом виде. Это представление оптимизировано для быстрого доступа и является достаточным для простейшей визуализации сцены (например, с использованием OpenGL), достаточной для процесса моделирования.

- **Специальное представление** – представление, оптимизированное для конкретного использования, например, для моделирования освещенности методом Монте-Карло. В зависимости от конкретных вычислений, возможно использование различных специальных представлений. Так, различные специальные представления удобны для работы с разными моделями света (RGB, спектральной). Специальное представление необходимо и для ускорения вычислений с использованием средств SIMD современных процессоров.

Связь между различными представлениями сцены показана на рис.5.

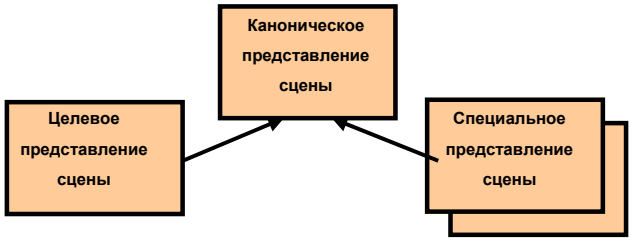

Рис. 5: Различные представления сцены.

Все представления имеют вид древовидной иерархии паевых или целевых объектов.

Исходным представлением сцены является ее целевое представление. Каноническое представление сцены строится по исходному целевому представлению. Специальные представления строятся по каноническому представлению. Объекты целевого и специального представлений могут ссылаться на объекты канонического представления, но не наоборот и не между собой (это показано стрелочками на рис. 5).

Модификации сцены выполняются при помощи целевых действий в объектах целевого представления. При этом целевой объект соответствующим образом модифицирует каноническое представление. Специальные представления, о которых каноническое представление формально ничего не знает, отслеживают модификации канонического представления и перестраиваются должным образом, используя аппарат уведомлений о событиях.

5. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

К пользовательскому **интерфейсу** компьютерных систем мы предъявляем следующие требования:

- ядро графической системы, обеспечивающее функциональную часть графического приложения, не должно зависеть от пользовательского интерфейса;

- ядро графической системы должно иметь возможность работы с различными пользовательскими интерфейсами, в том числе через Интернет;

- повторное (многократное) использование готовых компонент пользовательского интерфейса.

Эти требования достигаются последовательным применением специально разработанной концепции управляющего элемента (**IControl**).

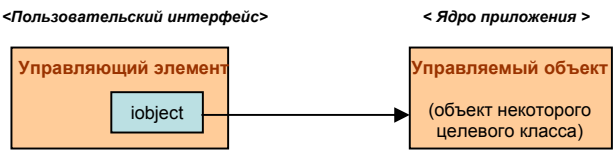

Рис. 6: Концепция управляющего элемента.

Концепция управляющего элемента основана на механизме обращения паевых ссылок. Класс управляющего элемента наследуется от подходящего класса системы

пользовательского интерфейса (такой как MFC, Qt) и хранит в качестве члена паевую ссылку на управляемый объект (обычно некоторый целевой объект ядра графической системы). Управляющий элемент визуально представляет состояние управляемого объекта для конечного пользователя (например, в виде диалоговой панели). Если пользователь вносит какие-либо изменения, управляющий элемент изменяет контролируемый объект (путем исполнения подходящего целевого действия) по имеющейся у него ссылке. Если состояние управляемого объекта изменяется по какой-то внешней причине, управляющий элемент узнает об этом через механизм уведомления о событиях и модифицирует визуальное представление для конечного пользователя соответствующим образом.

Таким образом, управляемый объект формально не знает о присутствии управляющего элемента. Тем самым обеспечивается независимость ядра графической системы от пользовательского интерфейса и возможность использования различных пользовательских интерфейсов с одним и тем же ядром.

Концепция управляющего элемента применяется иерархически ко всем элементам пользовательского интерфейса. На нижнем уровне мы имеем элементарные управляющие элементы для отображения / редактирования простых величин - соответствующие управляемые элементы называются **полями**. На самом верхнем уровне мы имеем управляющий элемент, соответствующий голому «каркасу» графической системы (окно верхнего уровня, с почти пустым меню и пустым набором инструментальных линеек). Управляемым элементом служит объект, описывающий специфику конкретной графической системы.

6. ПОДКЛЮЧАЕМЫЕ КОМПОНЕНТЫ

Большинство компонент графической системы оформляется в виде подключаемых компонент (динамических библиотек).

Все подключаемые компоненты имеют единый весьма общий интерфейс: функции запуска и завершения (InitClasses() / TermClasses()), которые гарантированно вызываются при подключении / отключении компоненты. Конкретная специфика подключаемой компоненты определяется тем, что она делает при подключении. Компоненты ядра графической системы, как правило, вводят новые паевые классы (наследуемые от некоторых уже определенных базовых классов) и создают новые объекты-типы для них (которые автоматически регистрируются в системе). Компоненты пользовательского интерфейса, как правило, регистрируют исполняющие объекты в специальных древовидных структурах, соответствующих меню и инструментальным линейкам графической системы. Тем самым они обеспечивают появление новых функциональных элементов в пользовательском интерфейсе графической системы.

Концепция подключаемых элементов используется как во внутренней архитектуре графической системы («LEGO»-принцип построения графической системы из набора компонент), так и для поддержки внешних пользователей-программистов (возможность расширения графической системы подключаемыми компонентами собственного производства).

7. ЗАКЛЮЧЕНИЕ

Рассмотренная инфраструктура используется в реальных системах компьютерной графики [1, 6], включая системы, работающие через Интернет [7].

Работа поддержана грантами РФФИ № 06-07-89162 и 07-01-00450, а также компанией Integra Inc. [8].

8. СПИСОК ЛИТЕРАТУРЫ

[1] Волобой А.Г., Галактионов В.А. *Машинная графика в задачах автоматизированного проектирования. "Информационные технологии в проектировании и производстве", № 1, 2006, с. 64-73.*

[2] 3ds Max SDK Documentation. *http://www.autodesk.ru/adsk/servlet/item?siteID=123112&id=7481368*

[3] Josie Wernecke. *The Inventor Toolmaker. Addison-Wesley, 1994.*

[4] The C# Language
http://msdn2.microsoft.com/en-us/vcsharp/aa336809.aspx

[5] Microsoft .NET Framework
http://www.microsoft.com/net/default.aspx

[6] Ignatenko A., Barladian B., Dmitriev K., Ershov S., Galaktionov V., Valiev I., Voloboy A. *A Real-Time 3D Rendering System with BRDF Materials and Natural Lighting. The 14-th International Conference on Computer Graphics and Vision GraphiCon-2004, Moscow, 2004, pp. 159-162.*

[7] Барладян Б.Х., Волобой А.Г., Вьюкова Н.И., Галактионов В.А., Дерябин Н.Б. *Моделирование освещенности и синтез фотореалистичных изображений с использованием Интернет технологий // "Программирование", № 5, 2005, с.66-80.*

[8] *http://www.integra.jp*

Object-oriented infrastructure for computer graphics systems

Abstract

An internal program infrastructure of complex computer graphics systems developed in the Keldysh Institute for Applied Mathematics RAS is considered.

The infrastructure is based on the number of original ideas. Special attention is paid to the system extensibility and reduction of development efforts.

Keywords: Computer graphics systems, extensibility, plug-ins, object-oriented infrastructure.

About the authors

Nickolay B. Deryabin, researcher of the Keldysh Institute for Applied Mathematics RAS. E-mail: dek@keldysh.ru

Eugeny Y. Denisov, junior researcher of the Keldysh Institute for Applied Mathematics RAS. E-mail: eed@spp.keldysh.ru