Использование Трехмерных Метафор Визуализации

В.Л. Авербух, А.Ю. Байдалин, Д.Р. Исмагилов, А.Ю. Казанцев, С.П. Тимошпольский
averbukh @ imm.uran.ru, bajur @ imm.uran.ru
ИММ УрО РАН, Екатеринбург
Работа выполнена при поддержке гранта РФФИ № 04-07-90120

В данной работе рассматривается использование трехмерных метафор для разработки систем визуализации программного обеспечения.
Ключевые слова: метафора визуализации, визуализация программного обеспечения

1. Введение

Априори представляется, что методы трехмерной графики должны резко усилить выразительность компьютерной визуализации , в том числе и методов визуализации программного обеспечения. Действительно существует целый ряд систем на базе трехмерных визуальных языков программирования, а также других систем визуализации программного обеспечения с использованием трехмерных метафор. В работах [1-2] дан подробный анализ систем визуализации программного обеспечения параллельных вычислений, включая языки визуального программирования, системы отладки правильности и настройки производительности. Значистельное внимание в обзорах уделено использованию методик виртуальной реальности.

Вместе с тем далеко не всегда использование трехмерных методик и методик виртуальной реальности приводит к положительному результату. Иногда и трехмерность и виртуальная реальность явно избыточны или же не дают нового качества. Так, несмотря на применение в системе Avatar [3] довольно экзотических методик, основанных на виртуальной реальности, сам репертуар видов отображения, используемых при визуализации ограничен теми же графиками, представляющими метрики производительности. Мало того, по нашему мнению пользователь может испытывать затруднения как при интерпретации десятков и сотен графиков, так и при навигации в рамках виртуальной реальности с использованием несовершенных методов взаимодействия. В качестве некоторых неформальных требований к качеству трехмерных видов отображения можно отметить необходимость сохранения структуры информации, хотя бы в одном из видов совокупности видов отображения. Вид отображения, представляющий некоторый набор связанных сущностей (подмножество), должен сохранять структуру взаимосвязанности этих сущностей.Представление сущности или структуры не должно увеличивать мерность прообраза. Т.е. не рекомендуется представлять очевидно двумерный объект чем-то существенно трехмерным. В тоже время значительное сокращение мерности может иметь успех (и не повредит качеству) только в том случае, когда исходная информация допускает такую агрегацию. Однако для объектов с «неопределенной» мерностью увеличение мерности образа может быть использовано для разработки более удачного (с точки зрения восприятия пользователя) способа представления. Примером этого может служить опыт представления графа вызовов (CallGraph) описанный ниже.

2. Задача представление графа вызовов

В [4] предложена стратегия поиска узких мест в смысле эффективности параллельных программ, основанную на использовании графа вызова функций (callgraph). Граф вызовов функций программы представляет собой ориентированный граф, где вершинами являются функции, дуги отражают вызов функций, а направление дуг указывает на направление вызова. В рамках системы анализа производительности больших программ VTune, разработанной в фирме Intel, также большое внимание уделяется визуализации графа вызовов функций. В системе VTune используется двумерное представление графа вызовов. При анализе большой по объему и сложной по структуре программы с большой глубиной вложенности вызовов функций и большим количеством пользовательских функций могут (и возникают) значительные сложности в двумерном отображении протяженной структуры на экране монитора и в анализе пользователем конечного изображения. Проблему нехватки места на экране дисплея можно решить, добавив к разрабатываемой модели представления еще одно измерение. Получаемая трехмерная картинка при отображении на том же экране монитора может представить гораздо больше сущностей. Трехмерное изображение лучше анализируется и воспринимается пользователем.

Были предприняты попытки трехмерного представления графа вызовов на базе нескольких предложенных нами метафор.

2.1 Метафора комнаты

В процессе исследования возникла гипотеза о представлении с помощью метафоры комнаты графа вызова функций в программе для его возможного использования в визуальной отладке. При рассмотрении имеющихся систем отладки была отмечена двумерность используемых в них видов отображения, что отразилось на затрудненной интерпретации изображения, особенно при работе с большими программами. Можно предположить, что переход в трехмерное пространство позволит нам добиться лучшей интерпретации графа вызовов функций программы. Попутно, расширяя рамки первой гипотезы, можно высказать следующую – отображение основных сущностей отладочного процесса на графе вызовов с использованием трехмерной метафоры комнаты повышает понимание процесса отладки и улучшает интерпретацию результатов. Под таковыми сущностями понимаются, например, точки останова программы, стек вызовов, узкие места производительности и т.п. Для проверки гипотез была написана система CG Visualization. Система предоставляет несколько идейно связанных между собой видов отображения.

Рассмотрим систему с концептуальной точки зрения. Применение данной метафоры здесь следующее – в качестве комнаты рассматривается функция исследуемой программы. В данной системе визуализируются программы, написанные

на языке С, функция понимается соответственно стандарту этого языка. Однако стоит заметить, что данный факт не является принципиальным в контексте проводимого исследования. Аналогичные системы можно реализовать для различных языков программирования, основанных на фон-Неймановской парадигме программирования (необходимо только написать разборщик исходных текстов для нового языка). Сама исследуемая программа представляется в виде взаимосвязанной системы комнат, расположенных в пространстве особым образом.

2.2 Разбиение функций на группы и их размещение в пространств

Введем некоторые определения. ***Родителем*** данной функции называется функция, из тела которой непосредственно вызывается хотя бы один раз данная функция. ***Потомком*** данной функции называется функция, которую вызывает данная функция непосредственно из своего тела хотя бы один раз. Под ***системными функциями*** будем понимать те используемые в исследуемой программе функции, которые не описаны программистом в ее тексте. Заметим, что данное определение является не до конца точным. так как в больших программах функции могут быть вызваны из различных библиотек, написанных как программистом, так и сторонними разработчиками.

Все функции программы разделяются на три обособленные группы, непересекающиеся между собой (отметим, что здесь под программистом понимается человек, написавший код исследуемой программы):

- Функции, написанные программистом, которые имеют в качестве потомком функции написанные программистом;
- Функции, написанные программистом, которые не имеют таковых в качестве потомков; (либо другими словами – все потомки таких функций исчерпываются системными функциями);
- Системные функции.

Кроме этих, выделим еще две группы функции, которые могут пересекаться с вышеописанными тремя группами– рекурсивные функции и функции, не имеющие родителей. Согласно этому разделению происходит пространственное расположение и раскраска функций в основном виде отображения системы – обзоре всего графа вызовов. Расположение комнат-функций в трехмерном пространстве должно соответствовать рассуждениям об экономии экранного пространства для отображения больших программ образом, поддающимся достаточно легкой интерпретации. Была выбрана следующая трехуровневая модель размещения комнат – в соответствии с разделением множества функций программы на три группы. На верхнем уровне располагаются функции, имеющие в качестве потомков только системные функции (вторая группа). На среднем – функции первой группы и на нижнем уровне размещены системные функции (третья группа). Приведем обоснование выбора такой последовательности размещения по уровням. В результате проведенных наблюдений и рассуждений высказывается предположение, что функции первой группы чаще других имеют большее число вызовов, соответственно их размещение должно быть таким, чтобы по возможности уменьшить пересечения дугами графа границ уровней. Кроме этого расположение комнат внутри верхнего и нижнего уровней решено было сделать наподобие амфитеатра (на верхнем уровне «амфитеатр» перевернут), что также позволяет нам разместить большее число элементов в ограниченном пространстве. Такое размещение комнат на верхнем и нижнем уровнях было выбрано еще и по причине того, что «амфитеатр» оставляет пространство над «ареной» свободным, а как раз в этом пространстве наблюдается наибольшая плотность дуг графа вызовов. Радиус «амфитеатра» вычисляется на стадии загрузки данных об исследуемой программе исходя из количества функций на соответствующих уровнях таким образом, чтобы в «амфитеатре» было не более пяти уровней.

Кроме того, за функцию «отвечает» и пиктограмма на стене комнаты функции-родителя в графе вызовов. Связи между комнатами – дуги графа вызовов естественно переносятся на изображение и отображаются на экране линиями следующими от пиктограммы функции на стенке родителя к комнате-кубу данной функции. Цвет линий градиентно переходит от белого к красному и обозначает направление вызова - вызов идет от функции, откуда «выходит» белая часть, и к функции, к которой подходит красный конец дуги.

Как уже было сказано, в соответствии с разделением на уровни внешний вид комнаты различаются по цвету: функции первой группы – фиолетовый, функции второй группы – синий, функции третьей группы – серый, рекурсивные функции – ярко зеленый, функции, не имеющие родителя – красный. Цвета выбраны в соответствии с основными принципами использования цветов.

2.3 Основные возможности системы

Система предоставляет все необходимые возможности навигации в этом виде отображения – приближение, поворот, перемещение. Пользователь имеет возможность рассмотреть каждую интересующую деталь графа вызовов, проследив, если потребуется какие-то отдельные вызовы, рассмотрев конкретные совокупности комнат. Хотелось бы отметить, что навигация производится путем непосредственного взаимодействия с самими изображением, без введения дополнительных элементов управления навигацией. Этот факт также важен, так как экран монитора не загромождается лишними кнопками и переключателями, освобождая максимум пространства для самого изображения. Да и само по себе прямое взаимодействие качественно улучшает удобство работы с системой, снижая сложность навигации.

В связи с идеей путешествия по комнатам можно рассмотреть понятие потока управления программы. Можно получить хорошую эффективность в смысле простоты интерпретации за счет использования анимации анимации перемещения по комнатам. То есть пользователь при реализации данного подхода получает возможность следовать за потоком управления из комнаты в комнату, из функции в функцию, находясь как бы внутри программы. Использование такого представления потока управления в отладке, и особенно в отладке параллельных программ, может привести к интересным результатам.

Система CG Visualization разработана при помощи системы программирования Microsoft Visual Studio версии 6.0 Service Pack 5 с установленным Platform SDK от ноября 2001 года (версия 5.1.3590.2). Графическая часть системы реализована с помощью набора библиотек OpenGL. Система реализована как диалоговое приложение с использованием библиотеки MFC - управление диалогом на уровне API скрыто внутри MFC библиотеки.

Как показало исследование, метафора комнаты – естественная метафора, возникшая перенесением

моделируемого объекта из реального мира – оправдывает свое применение в данном вопросе визуализации, решает проблемы нехватки экранного пространства, масштабируемости, повышения реалистичности и естественности изображения.

Закономерным можно назвать предположение о том, что использование метафоры комнаты вкупе с системами виртуальной реальности наиболее подходит для реализации множества различных систем визуализации.

3. Метафора молекулы

Кроме того, была предложена следующая метафора трехмерного представления графа вызовов – *метафора молекулы*. В данной метафоре функции (вершины CallGraph'а) будем отображать в виде сфер, а вызовы функций (связи между вершинами) будем отображать в виде стрелок (линия с конусом на конце). Полученную трехмерную сцену будем отображать на экране в перспективной проекции

Для улучшения восприятия полученного изображения будем использовать освещение и позволим пользователю свободно перемещаться в трехмерном пространстве отображаемой сцены.

Однако, при отображении графа вызовов с большим числом функции (больше 10), возникает следующая проблема: каким образом разместить большое количество связанных объектов в трехмерном пространстве?

Необходимо предусмотреть такой механизм размещения объектов в 3D, чтобы выполнялись следующие условия:

- структура программы должна хорошо просматриваться даже при большом количестве объектов на экране,
- программы с примерно одинаковой структурой должны выглядеть примерно одинаково,
- пользователь должен прилагать для этого минимальное количество усилий, т.е. процедура расстановки должна быть автоматической.

Итак, необходимо автоматически разместить вершины в 3-х мерном пространстве (3D), чтобы обеспечить приемлемую картину (чтобы можно было увидеть (и понять) структуру CallGraph'а).

Поэтому, воспользуемся примером из окружающей среды: любая система в природе стремится к минимуму потенциальной энергии (некой характеристики, которая определяет глобальную устойчивость системы).

Чтобы определить потенциальную энергию для нашей системы введем взаимодействие: в вершинах графа разместим электростатические заряды, а связи между вершинами заменим упругим взаимодействием.

Тогда под потенциальной энергией всей системы будем понимать сумму потенциальных энергий ее составляющих (потенциальная энергия взаимодействия всех пар вершин).

Итак, имея описанный алгоритм автоматической расстановки вершин CallGraph'а, можно воспринимаемым образом отображать на экране с достаточно большим числом элементов (до нескольких сотен и даже тысяч). При этом, получаемые изображения удовлетворяют всем свойствам, описанным выше, включая независимость конечного положения вершин от начального (программы с одинаковой структурой будут выглядеть одинаково).

При анализе получившейся 3-х мерной модели можно отметить интересную особенность, подтверждающую перспективность использования *метафоры молекулы* для отображения CallGraph'а: функции, часто используемые вместе, располагаются недалеко друг от друга.

Следует отметить, что *метафора молекулы* может быть использована и для представления структуры других сложных систем, таких как структура баз данных, а также, возможно, для научной визуализации в области химии и биологии.

При использовании *метафоры молекулы* для отображения структуры CallGraph'а также удается естественным образом (не внося специальных изменений в программу отрисовки) отобразить рекурсивные вызовы функции: функция, вызывающая саму себя, будет отображаться в виде сферы с прикрепленным к ней одиноким конусом (без связи с другой вершиной) (Рис. 3). Вследствие того, что при 3-х мерном отображении сфера имеет большую часть своей поверхности открытой, одинокий конус достаточно легко обнаружить на общей картине CallGraph'а.

Использование алгоритма автоматической расстановки уже сейчас позволяет добиться за приемлемое время воспринимаемой картины при отображении сложных структур с большим числом элементов (~1000). В сочетании с применением цвета и текстур с анимацией для отображения различных параметров функций (таких как время, затраченное процессором для выполнения данной функции и др.), возможно будет использовать *метафору молекулы* для анализа эффективности приложения.

Программа, осуществляющая автоматическую расстановку вершин и отображение структуры CallGraph'а, используя метафору молекулы, написана на Microsoft Visual C++ с использованием функций WinAPI и интерфейсов DirectX (Direct3D9, DirectInput8).

Визуализация отладки/настройки производительности параллельных программ зачастую сводится к изучению огромных массивов данных. Для реализации навигации в пространстве отображаемых данных необходимо использование метафор перемещения. В этом качестве были изучены метафора *полета над* информационным пространством.

4. Полет над инфопространством

Для отображения произвольного (конечного) 2х мерного массива информации предлагается метафора сферы. Но, в противоположность известной метафоре «рыбий глаз», информация отображается не на внешней, а на внутренней стороне сферы Таким образом, если при отображении «рыбьим глазом» выбранный элемент оказывается крупнее (ближе к пользователю), при отображении внутри сферы мы получим в начальном положении все элементы информации на равном расстоянии от камеры, что позволяет сначала выбрать интересующий нас элемент «стоя на месте», и только поворачивая камеру, а у же затем приблизиться к нему или, «нырнув», исследовать его изнури. Для отображения и исследования некоторого конечного 3х-мерного массива информации рассмотрим метафору «полет над ландшафтом». В качестве ландшафта здесь будет выступать сам информационный массив, элементы которого расположены специальным способом.

Нами были реализованы пилотные версии систем визуализации с использованием метафор полета и размещения информации на внутренней поверхности сферы. Оказалось, что эти метафоры полезны не только для нужд

информационной визуализации и визуализации программного обеспечения, но и научной визуализации.

Перспективой данных исследований можно считать разработку новых систем визуализации на базе созданных нами, или же разработку абсолютно новых систем на основе той же идеи: комбинирования нескольких, возможно разнородных, метафор визуализации в одной системе [5].

5. Литература

1. Авербух В.Л., Байдалин А.Ю. Проектирование средств визуализации отладки производительности для системы параллельного программирования DVM // Алгоритмы и программные средства параллельных вычислений. УрО РАН, 2001 Вып 5. 37 стр.
2. Авербух В.Л., Байдалин А.Ю. Проектирование визуальных средств разработки программ для системы параллельного программирования DVM // Алгоритмы и програм. ср-ва параллельн.вычислений: Сб.науч.тр. Вып. 6 / ИММ УрО РАН. Екатеринбург, 2002. Стр. 3-33.
3. Reed D., Scullin W., Tavera L., Shields K., Elford Ch. Virtual Reality and Parallel Systems Performance Analysis // IEEE Computer, V.28, N 11, (November 1995) pp. 57-67.
4. Cain H., Miller B.P., Wylie B.J.N. A Callgraph-Based Search Strategy for Automated Performance Diagnosis // Lecture Notes In Computer Science Proceedings from the 6th International Euro-Par Conference on Parallel Processing (Munich, Germany), August 2000. London Springer-Verlag Pages: 108 – 122.
5. Авербух В.Л., Байдалин А.Ю., Исмагилов Д.Р., Казанцев Д.Р. Поддубная С.В. Трехмерное представление графа вызовов функций // Супервычисления и математическое моделирование: Тезисы международного семинара. Саров, 2003 ВНИИЭФ-РФЯЦ Стр. 12-13

6. Иллюстрации

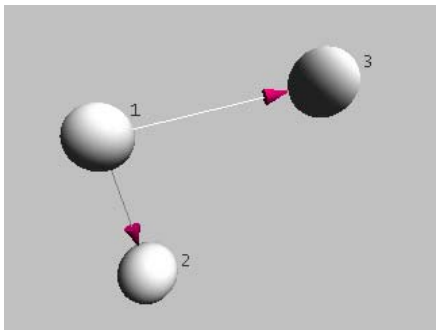

Рисунок 1. пример графа вызова

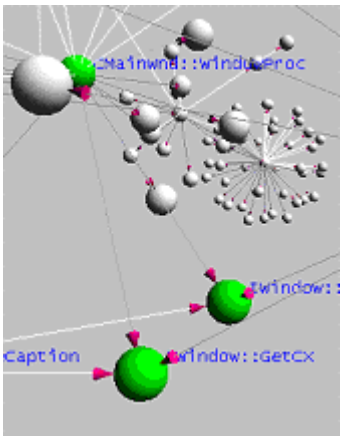

Рисунок 2. группировка узлов

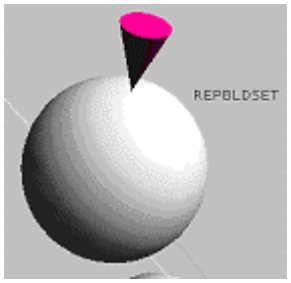

Рисунок 3. Обозначение рекурсии

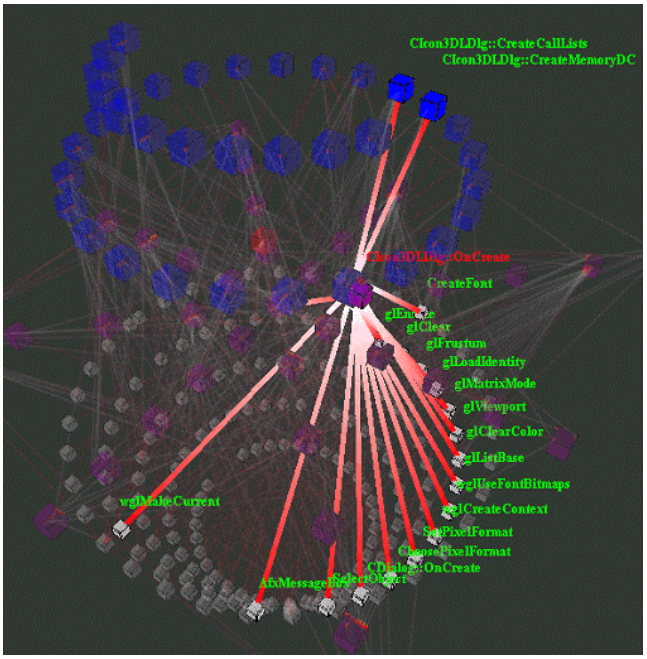

Рисунок 4. Общий вид CG Visualization, системы 3 мерного представления графа вызовов