УДК 681.03

Параллельное программирование задач визуализации научных данных.

Дорошенко А. Е., Рухлис К.А.

Институт программных систем НАНУ, Киев, 03187, проспект Академика Глушкова, 40, тел. 266-15-38, dor@isofts.kiev.ua

Застосування візуалізації даних в науці набуло широкого вжитку. Але проблема своєчасного постачання даних для візуалізації досі не є вирішеною, що заважає організації інтерактивної роботи та прискоренню процеса отримання результатів. В роботі запропоновано використання паралельних обчислень для одного класу задач візуалізації, а саме візуалізації заданих у явному вигляді поверхонь. Розглянуто існуючі методи подання поверхонь (регулярні полігональні сітки, іррегулярні полігональні сітки, лінії рівня), а також деякі методи організації паралельної роботи програм. Висвітлюються результати з реалізації паралельних програм методами мультипоточності та розподіленності обчислень, а також методологія збільшення продуктивності обчислень завдяки адаптивності програм.

The use of the data visualization in science is widely spread now. However, problem, connecned with the well-timed reception of the data given for visualization, is not solved hitherto; that prevents organization of the interactive work and speedups of the process of the results reception. The paper offers using the parallel calculations for a class of the data visualization — visualization of the surfaces, given in evident type . Existing methods, representing the surfaces (the regular polygonal meshes, irregular polygonal meshes, level-lines), as well as some methods of the realization of the parallel working programs are reviewed. Also, the results of the parallel programs realization by multithreading approach and distributed calculations are described, including adapting methodology of perfomance increase.

Введение.

Электронные вычислительные машины позволяют облегчить и ускорить выполнение рутинных задач, которые до этого приходилось выполнять фактически вручную. В той или иной степени такие задачи были связаны с вводом, обработкой, выводом данных. И если ввод информации в настоящее время осуществляется автоматически или автоматизированным способом, то обработка, представление и вывод до сих пор требуют вмешательства человека (в качестве эксперта или пользователя). Особенно актуальна проблема визуализации данных, поскольку машинные данные — это не более, чем набор чисел, а интерпретация этих данных должна определяться самим пользователем. С математической точки зрения данные можно представить тремя способами: аналитическим (в виде формулы, системы уравнений и т.д.), табличным, графическим. Из всех трёх наибольшей наглядностью обладает графический, причём часто его применяют после получения решений в аналитическим или табличном виде. Подробнее о способах визуализации будет расказано в пункте 2., но стоит заметить, что в наше время создано достаточно большое количество программных продуктов для решения этой задачи, как специализированных ,в частности IBM DataViz, ScionImage, Vizio Pro, IP Lab, Erdas Imagine, Er Mapper, так и математические пакеты общего назначения типа MathCad и MatLab [11,12].

В настоящее время в научных вычислениях широко применяются параллельные вычисления. Этому способствуют не только хорошая распараллеливаемость математических методов [8], но также и использование кластерных систем, которые стоят намного дешевле, чем классические многопроцессорные системы, а по производительности уступают только незначительно. Для паралелльных систем созданы специальные стандартизированные программные среды для работы параллельных программ с интерфейсами стандартных языков программирования высокого уровня (ЯВУ) типа C/C++, Fortran, Pascal. Кроме того, создание параллельной программы для стандартных архитектур (например, SPMD [8]) отличается от создания последовательной по крайней мере необходимостью двух этапов: выбором способа организации параллелизма [1] и организацией протокола обмена информацией между параллельными вычислителями. Преимуществ же намного больше: большая скорость вычислений (в идеале в *n*-раз, где *n* — количество вычислителей), намного большая надёжность параллельной программы чем последовательной (в случае выхода из строя одного из вычислителей параллельная программа потеряет всего лишь часть полученных результатов, которые сможет получить заново на другом вычислителе), масштабируемость параллельной программы позволяет фактически без перекомпиляции использовать её на различных конфигурациях их 2-х, 3-х, и т.д. вычислительных узлов, а также динамически подключать к вычислениям новые узлы.

Как уже было замечено, достаточно эффективным оказывается использование параллельных вычислений для получения данных для визуализации, особенно в задачах реального времени, таких как визуализация плазмы, медицинских данных, виртуальной реальности, что, в принципе, ускоряет и саму визуализацию. Такие данные могут быть двух видов [9]:

1) данные, полученные экспериментальным путём;

2) данные, которые надо получать из аналитического описания ситуаций, систем [11].

Поскольку источники данных могут иметь абсолютно разную природу, это накладывает дополнительные условия как на саму визуализацию так и на собственно распараллеливание процесса получения данных. В данной работе мы ограничились одной из наиболее востребованных прикладных задач визуализации — визуализацией 3х мерных поверхностей заданных в явной форме(z=F(x,y)). Эта задача достаточно демонстративна и хорошо распараллеливается.

Следует отметить, что параллельная работа может быть организована с помощью:

1) мультипоточности;
2) организации собственного протокола обмена данными между процессами через механизм сокетов;
3) использования технологий типа MIDAS;
4) применения механизмов RPC;
5) использования сред параллельной работы процессов типа PVM, MPI;
6) агентых технологий.

1. Способы организации мультизадачности.

Мультипоточность программы позволяет организовать более эффективный обмен данными между потоками(нитями) благодаря тому, что все они разделяют общую память процесса-родителя. Вместе с тем, параллелизм выполнения потоков всецело зависит от конкретной реализации ОС и оборудования. Например, ОС Windows XP с поддержкой HyperThreading (HT) будет выполнять псевдопараллельно потоки на процессоре без поддержки HT и с максимально возможной параллельностью на процессорах, поддерживающих эту технологию(полная параллельность не получится потому, что HT — это не полноценные два процессора в одном, а один процессор с дублированными внутри себя основными модулями), а вот Windows 9x что на процессоре с поддержкой HT, что на процессорах без HT будет выполнять потоки псевдопараллельно. Кроме того, реализация потоков в разных ОС отличается как на уровне API, так и на уровне их планировки самим ядром ОС.

Организации собственного протокола обмена данными между процессами через механизм сокетов имеет следующие преимущества:

1) Возможность одновременного использования разных ОС для решения поставленных задач(сокеты — кроссплатформенный стандарт), что позволяет произвести "заточку" каждого отдельного процесса под свою ОС;

2) В некоторых случаях большая надёжность(при потери связи с некоторым узлом вся система не должна будет производить все расчёты по новому, а только той части данных, которая вычислялась на отключившемся узле). Следует заметить, что в общем случае это справедливо только когда данные независимы друг от друга;

3) Защищённость процессов-вычислителей друг от друга. В случае, если процессы запущены на одном и том же компьютере этот фактор становится зависим от ОС(В реальности такой случай возможен только при отладке протокола взаимодействия, ибо для вычислений окажется эффективней использовать многопоточность);

Но есть у данного способа организации параллельной работы и недостатки:

1) Накладные расходы на передачу данных между различными узлами сети(этим страдает в принципе любой способ организации параллельной работы на сети);

2) Необходимость сперва разработать протокол обмена информацией, отладить его, а только потом заниматься организацией параллельных вычислений;

3) В случае, если часть узлов являются многопроцессорными системами, данный метод не сможет использовать преимущества этих узлов;

Использования технологий типа MIDAS позволяет воспользоваться всеми её преимуществами, "встроенной" поддержкой со стороны семейства ОС Win32, а также поддержкой со стороны наиболее популярных компиляторов и интерпретаторов под эту платформу. Поддержка со стороны компиляторов на самом деле достаточно весомый аргумент в пользу MIDAS, так-как вместе с компиляторами поставляются наборы библиотек, облегчающих организацию информационного обмена. К недостаткам следует отнести два фактора: MIDAS фактически является решением под одну платформу —Win32; MIDAS разрабатывалась не для использования для научных вычислений, что довольно таки значительно влияет на скорость работы приложений, использующих её.

Применения механизмов RPC. Вся загвоздка с этим методом организации параллельных вычислений состоит в том, что существует несколько стандартов и API RPC, как кроссплатформенных, так и расчитанных на определённую платформу. Применение того или иного RPC в каждом конкретном случае зависит, в первую очередь, от того рода задач, для которого организовываются параллельные вычисления. Поскольку RPC — довольно обширная тема, выходящая за предел статьи, мы не будем рассматривать их дальше. Стоит только отметить, что многие реализации RPC реализованы на уровне ядра, что в свою очередь обеспечивает параллельные вычисления(с использованием этих API RPC) достаточно большой скоростью информационного обмена и выполнения, в частности. Как недостаток выступает тот факт, что многие API RPC не являются кроссплатформенными — что ограничивает область их применения.

Использования сред параллельной работы процессов типа PVM, MPI. Данный способ объединяет в себе преимущества и недостатки мультипоточности, механизма сокетов, RPC. Как и сокеты, такого рода среды позволяют объединить узлы с различными платформами в одну вычислительную систему, но при этом они могут использовать многопроцессорность некоторых узлов(аппаратная мультипоточность), а также непосредственные возможности ОС и, в частности, ядра ОС. Такие среды обычно являются кроссплатформенными, с развитыми примитивами обмена информацией, управления, планирования, что

позволяет программисту(ам) больше времени уделять собственно параллельным вычислениям. Недостатки у данного подхода не столь очевидны, но всё же есть:

1) необходимость в развёртывании на узлах, где будут производится вычисления сперва самих сред параллельной работы(с полным ручным конфигурированием каждого отдельного узла), а уж потом самих процессов-вычислителей;

2) невозможность функционирования процессов-вычислителей без установленной среды(причём, часто, невозможность функционирования в той же среде, но другой версии, или реализации);

Агентные технологии. Позволяют использовать в параллельных вычислениях такие качества как: интелектуальность, мобильность, способность автоматической адаптации под изменения в окружающей среде, мультиплатформенность. Последнее качество достигается тем, что большинство агентных платформ написаны на Java(или других специализированных интерпретируемых языках), либо на C/C++ под компиляторы GNU C/C++(что позволяет фактически без особых изменений переносить агентную-систему на несколько десятков программно-аппаратных платформ, поддерживаемых компилятором и его библиотеками). Недостатки у этого подхода фактически такие же как и у сред параллельной работы: необходимость в наличии на узлах развёрнутых агентных-сред, невозможность выполнения вычислителей без этих сред. Исключение составляют некоторые платформы(ориентированные на C/C++ и реализованные на C/C++), которые позволяю пристыковать себя к файлу каждого отдельного вычислителя — тогда отпадает потребность в предварительном развёртывании агентной-платформы, увеличивается надёжность всей параллельной системы, но увеличивается ресурсоемкость каждого отдельного вычислителя. Стоит добавить, что в случае реализации агентной-платформы на интерпретируемых языках, возникает потребность в наличии как самой агентной-платформы на каждом узле, так и достаточно(для каждой конкретной задачи) эффективного интерпретатора этого языка.

2. Задача визуализации заданных в явном виде поверхностей.

Как уже отмечалось, смысл в данные вкладывает конечный пользователь, а сама природа данных зависит от конкретных задач. Отсюда и множество способов, которыми могут эти самые данные быть представленны.

Поскольку, в общем случае F(x,y) — не тривиальная функция, а отрезки x є $[x_1,x_2]$ и y є $[y_1,y_2]$ могут быть достаточно большими(с точки зрения возможностей компьютера: объёма памяти, производительности CPU), расчёт данных на однопроцессорном компьютере становится проблематичным(с точки зрения времени выполнения расчётов, ибо сама визуализация выполняется графическим акселератором). Вот здесь то и приходится использовать параллельные вычисления.

Независимо от способа организации параллельных вычислений возникает необходимость в следующих действиях:

1) Определение способа разделения пространства XxY(X=$[x_1,x_2]$ и Y=$[y_1,y_2]$), с тем, чтобы с максимальной эффективностью загрузить каждый вычислительный узел.

При этом, обязательно:

$$\bigcup_{j=1..n} (XxY)_j = XxY \text{ ,где}$$

(XxY)j — подпространство (XxY), выделенное для вычислений j-му вычислителю.

2) Определение критерия эффективности загрузки вычислительных узлов;

Следует заметить, что для разных способов организации параллельных вычислений эти два действия могут отличаться для одной и той же поверхности.

Итак, для организации параллельной работы вычислителей с использованием любой среды параллельной работы необходимо:

- Выполнить действия 1) и 2);
- Выбрать или разработать протокол коммуникаций между вычислителями;
- Реализовать вычислители в соответствии с требованиями среды параллельной работы;
- Выбрать и реализовать способ визуализации.

Последний пункт не относится напрямую к параллельной работе процессов, но именно от него зависит выбор и организация структур данных, что в свою очередь, влияет на протоколы взаимодействия, скорость расчётов, ресурсоёмкость каждого отдельного вычислителя. Существует достаточно много способов реализации, но наиболее популярны следующие [6,7, 14]:

1) Регулярные полигональные сети

2) Нерегулярные полигональные сети

3) Линии уровня

4) Специально включённые в API визуализации изоповерхности [7], использующие кубические сплайны.

Регулярные полигональные сети(в литературе они называются структурированными множествами данных)[7 с.494-496] имеет такие преимущества как простоту организации данных(фактически это n-1 мерный массив, где n — размерность пространства), "мгновенный" доступ к данным в любой точке (x,y) є R, где R — область определения z=F(x,y), сохранение результатов выборки без "потерь". Как результат — потребность в памяти составляет $(x_2-x_1)*(y_2-y_1)*S$ байт, где x1,x2 — область значения первого аргумента; y1,y2 — область значения второго аргумента функции z, а S— размер в байтах используемого типа данных для значений(например:byte-1, word-2, dword-4 и т.д.). Кроме того, в данном случае существует избыточность данных: если для x є $[x_i,x_j]$ и y є $[y_k,y_l]$ z(x,y)=const , то в регулярных сетях всё равно будут храниться $(x_j-x_i)*(y_l-y_k)$ элементов, вместо того, чтобы сохранить только значения x_i, x_j, y_k, y_l и z(x,y) на этом отрезке.

Данного недостатка лишён метод нерегулярных полигональных сетей[12]. Как уже отмечалось, он позволяет хранить данные отдельных участков, необязательно последовательных(такая особенность особо удобна, если нас интересует, например, поведение поверхности в зонах локальных минимов/ максимумов, а не на всей области определения). Кроме того, в случае наличия нерегулярных дискретных данных этот метод становится единственно приемлимым. Но у него есть и весьма серьёзные недостатки:

1) В случае, если нам потребуются данные из тех областей определения, которые не интересовали нас во время предыдущих расчётов, нам прийдётся расчитывать их по новому;
2) Очень часто, для уменьшения количества данных, хранимых в нерегулярных сетях применяют механизм flatten: если на некоторой подобласти определения R' z=F(x,y)=const для (x,y) Є R' за исключением подобласти R" Є R' Є R, где z=F(x,y)=const для (x,y) Є R" и R"<<R', то про такой "выброс" "забывают", записывая в память только то, что на всей без исключения R' z=F(x,y)=const. Это ведёт к тому, что теряется информация о поведении z в R", что иногда довольно существенно.

Изображение поверхности в виде полигональной сети наглядно, но не даёт возможности количественно оценить значения функции в отдельных точках или в окрестностях точек. Именно для количественных оценок и используются линии уровня — кривые, соответствующие конкретным значениям функции. Для построения линии уровня необходимо найти решения уравнения f(x,y)=c для *x,y* — из области определения, а *c*- искусственно выбираемый параметр [7]. Несмотря на то, что изображениям, построенным с использованием этого метода явно не хватает наглядности, он до сих пор используется в топографии, а также при построении развёрток температурных режимов.

Визуализация с использованием изоповерхностей, основанных на кубических сплайнах [7] требует дополнительных математических преобразований (вычислений коэффициентов поверхности), что автоматически увеличивает время построения поверхности и расхода памяти. Но, вместе с тем, такая визуализация наиболее презентабельна, что и породило сферу её применения — шоу, презентации, конференции.

3. Параллельное программирование визуализации.

Как было сказано в разделе 1, организация параллельной работы может быть произведена с использованием мультипоточности, сокетов, использования технологий типа MIDAS, применения механизмов RPC, использования сред параллельной работы процессов типа PVM, MPI, агент-технологий. Решение задачи визуализации поверхности, заданной в явной форме решено было проводить независимо на платформах Win32 и Linux.

Отказ от использования технологии MIDAS вызван теми недостатками, что были перечислены в разделе 1 и прежде всего низкой скоростью работы.

Решено было отказаться от применения механизмов RPC в связи с тем, что существует достаточно большое количество различных API RPC, как кроссплатформенных, так и нет, но в подавляющем случае несовместимых друг с другом. Это накладывает довольно существенные ограничения на структуру вычислительной сети, делая невозможной организацию параллельных вычислений на вычислителях с разной платформой в случае отсутствия кроссплатформенности API RPC; и низкопроизводительной в противоположном случае.

В качестве способа визуализации использованы регулярные полигональные сети. Как упоминалось в разделе 2 это даёт нам такие преимущества как наглядность, простоту организации данных, скорость доступа к данным в любой точке поверхности. Решение проблемы с необходимым объёмом памяти возложено на ОС(большинство современных ОС реализуют механизм виртуальной памяти(теоретически неограниченного объёма), кроме того, для уменьшения объёма необходимых данных можно увеличивать шаг выборки).

3.1. Реализация с использованием потоков.

Реализацию было решено проводить с использованием среды разработки Delphi 5. Тому есть несколько причин: скорость реализации графического интерфейса программы; использование ЯВУ Object Pascal с возможностью применения встроенного ассемблера; скорость работы грамотно разработанных откомпилированных программ фактически не уступает аналогам на C++, а иногда и выше. В качестве API

визуализации мы использовали OpenGL — она кроссплатформенная, поддерживается всеми современными аппаратными ускорителями графики, реализует все современные технологии.

Поскольку, нами использовались регулярные полигональные сети, а также мультипоточность, решено было применить модель параллелизма с общей памятью. Это автоматически потребовало от нас использовать такой способ разбивки пространства, при котором каждый отдельный вычислитель работал со своим участком массива, непересекающимся с любым другим(Нужно отметить, так как любой из наших параллельных вычислителей проводит расчёты одной и той же функции z=F(x,y) , то "затирание" вычислителем *i* некоторых данных, полученных вычислителем *j* фактически их не изменит. Однако пренебрежение требованием "непересечения" несёт с собой проблему дополнительных, ненужных вычислений.). Поскольку планирование потоков является прерогативой ОС , нами был выбран способ разделения пространства на равные участки(так как непонятно, какой из потоков будет выполнятся на каком процессоре в *i* квант времени, что исключает возможность измерить производительность конкретного процессора).

Программа организована следующим образом:

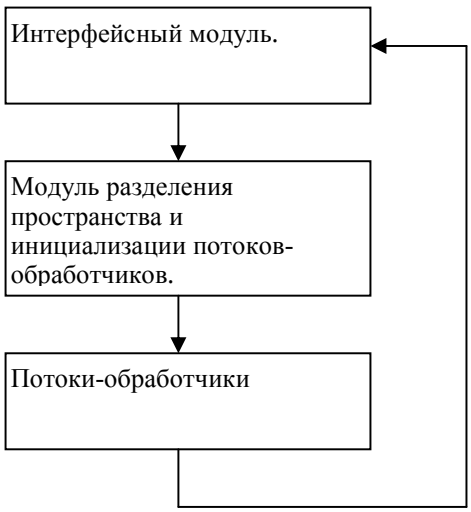

Рис. 1. Схема программы

, где —> обозначает поток данных от одного модуля другому.

В задачу интерфейсного модуля входит ввод параметров поверхности, области определения(для расчёта на ней этой поверхности), передача этих параметров "модулю разделения пространства и инициализации потоков-обработчиков", а также вывод расчитанных значений на экран в виде графика поверхности.

Модуль разделения пространства и инициализации потоков-обработчиков занимается разбивкой области определения на равные участки по количеству потоков-обработчиков, а также инициализацией потоков-обработчиков данными о соответствующих им подобластях определения и уравнении поверхности z=F(x,y), и запуск потоков на выполнение.

Каждый из потоков-обработчиков производит расчёты поверхности на своём участке области определения и записывает полученные значения в свою область глобального массива значений z. После окончания работы всех вычислителей интерфейсный модуль проводит визуализацию поверхности согласно данным глобального массива значений z.

Рабочее окно программы можно увидеть на Рис2.

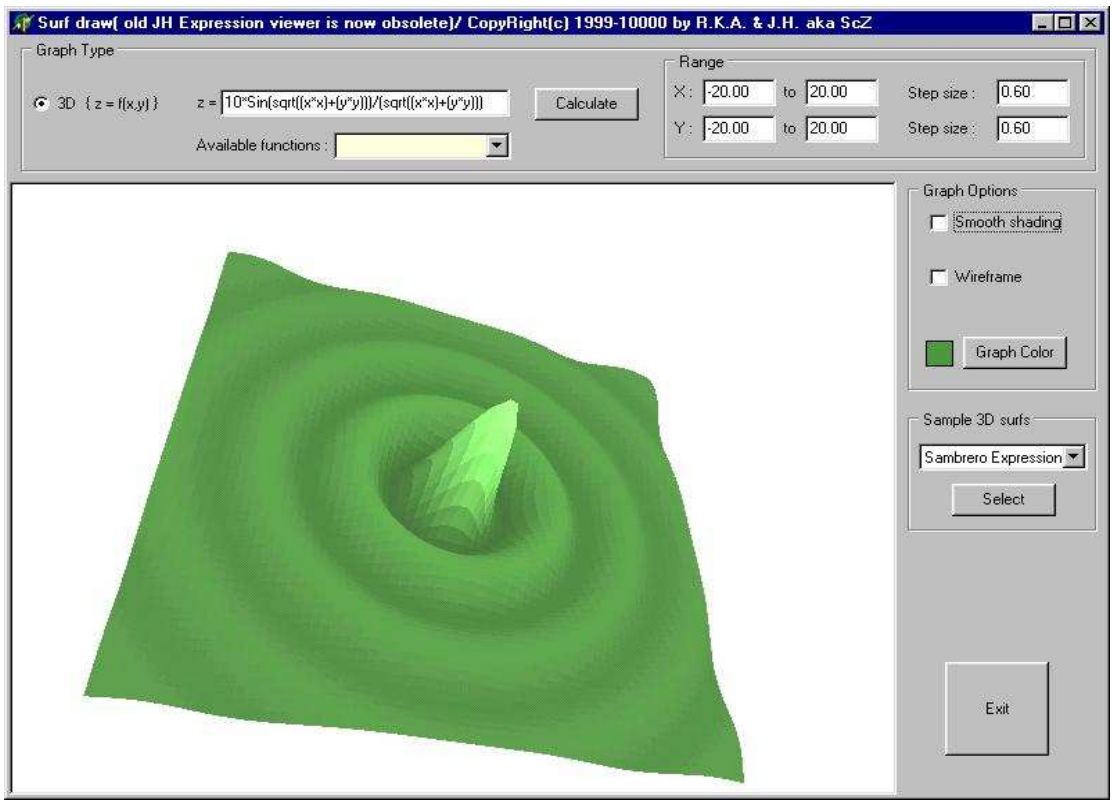

Рис. 2. Рабочее окно программы визуализации с использованием мультипоточности вычислителей

Результаты тестирования(скорость относительно версии программы без параллелизма) представлены в таблице 1. Для тестирования взяты процессоры с классической архитектурой и процессор Intel с поддержкой технологии HyperThreading(HT). Соответственно, в первом случае единицей отсчёта были результаты, показанные на процессоре AMD K6-II+ последовательной версией программы, а во втором результаты, показанные на процессоре Pentium 4 2.8 GHz с отключённой поддержкой HT последовательной версией программы. В качестве тестовой поверхности использовалась поверхность сомбреро(см. Рис. 2).

Таблица 1. Скорость относительно последовательной программы.

	Процессор			
Тип реализации	K6-II+ 500	PII450	Pentium 4 2.8 GHz HT support off	Pentium 4 2.8 GHz HT support on
Последовательный	1	1	1	1.12
Мультипоточный	1.15	1.1	1.12	1.14

3.2. Распределенная реализация с использованием сокетов.

Так же как и предыдущий метод реализации, мы использовали ЯВУ Object Pascal и среду Delphi 5. Однако, в этом случае мы уделили больше внимания кроссплатформенности исходного кода, в связи с чем, вынуждены были отказаться от построения интерфейса визуализатора с использованием компонентов VCL(CLX). Проблема с использованием "родных" для Delphi/Kylix компонент в *nix(nux) системах отнюдь не в том, что ОС написанная на C плохо работает с программами, написанными на других языках(во-первых, Kylix компилирует программы в стандартный для Linux формат двоичного исполняемого файла ELF; а во вторых ОС семейства Windows также написаны на C, и с ними у Delphi "проблем не возникает"). Всё дело, в двух вещах:

- для работы программы, использующей GUI в *nix(nux) системах необходимо в 99% случаях наличие соответствующего оконного менеджера(или его библиотек), в то время как для OpenGL абсолютно всё равно, что за оконный менеджер используется. Отсюда следует потеря мобильности и необходимость в развёртывании дополнительных программных комплексов для запуска основного приложения;
- Delphi/Kylix могут создавать программы под ОС Windows/Linux, а стандарт сокетов реализован фактически на всех ОС. Отсюда уже следует потеря кроссплатформенности.

Если первая проблема фактически не связана с Delphi/Kylix, и является общей для любой среды разработки под *nix(nux) платформы, то вторая проблема — напрямую относится к продукции Borland. Решение её можно разбить на два этапа:

- Использование в программе стандартных(а также стандартизированных) функций библиотеки выполнения(runtime library) и синтаксических конструкций, а также прямой работы с 'X'-ами, без привязки к оконному менеджеру;
- Применение компилятора языка Pascal максимально совместимого по синтаксису и функциям с Object Pascal, но умеющего компилировать под разные програмно-аппаратные платформы;

В качестве такого компилятора подходит FreePascal. Фактически, он полностью поддерживает диалект Object Pascal, а по платформам для которых он может компилировать приложения намного "разнообразнее": среди процессоров — процессоры Intel и Motorola, а среди ОС —Dos32, WIN32, различные *nix(nux), QNX, AmigaOS. Именно с учётом требования совместимости программного кода FreePascal и Delphi/Kylix, а также требования кроссплатформенности и проводилась реализация с использованием сокетов.

Мы применили клиент-серверную модель взаимодействия: в качестве клиента выступает интерфейсный модуль, а в качестве серверов — вычислители(Рис 3). Метод разделения пространства тот же, что и в 3.1.

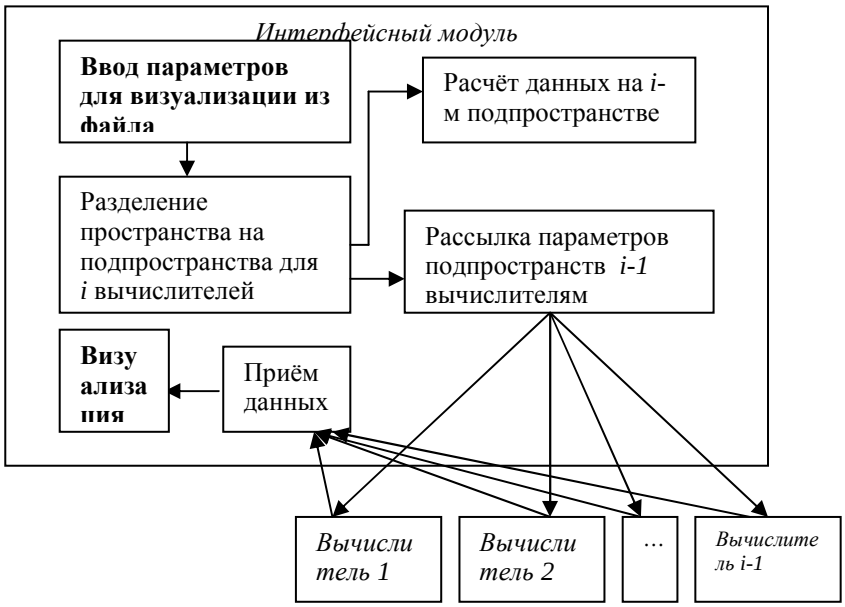

Рис. 3.

, где —> обозначает поток данных от одного модуля другому.

В задачу интерфейсного модуля входит ввод параметров поверхности, области определения(для расчёта на ней этой поверхности) из файла, разделение пространства, обмен данными с серверами-обработчиками, а также вывод расчитанных значений на экран в виде графика поверхности.

Каждый из потоков-обработчиков производит расчёты поверхности на своём участке области определения, и возвращает, по-окончании расчётов, данные одним информационным пакетом интерфейсному модулю, который записывает полученные значения в свою область глобального массива значений z. После окончания работы всех вычислителей интерфейсный модуль визуализирует поверхность согласно данным глобального массива значений z. Следует заметить, что в целях исключения простоя компьютера с интерфейсным модулем(ИМ) во время расчётов данных поверхности, мы интегрировали в ИМ один из вычислителей.

Тестовые испытания проводились для двух случаев: для локальной сети с практически одинаковыми(по конфигурации) компьютерами-узлами; и для локальной сети с различными компьютерами-узлами.Рабочее окно программы, с обозначением подпространств отдельных вычислителей(n=4) представленно на Рис 4.

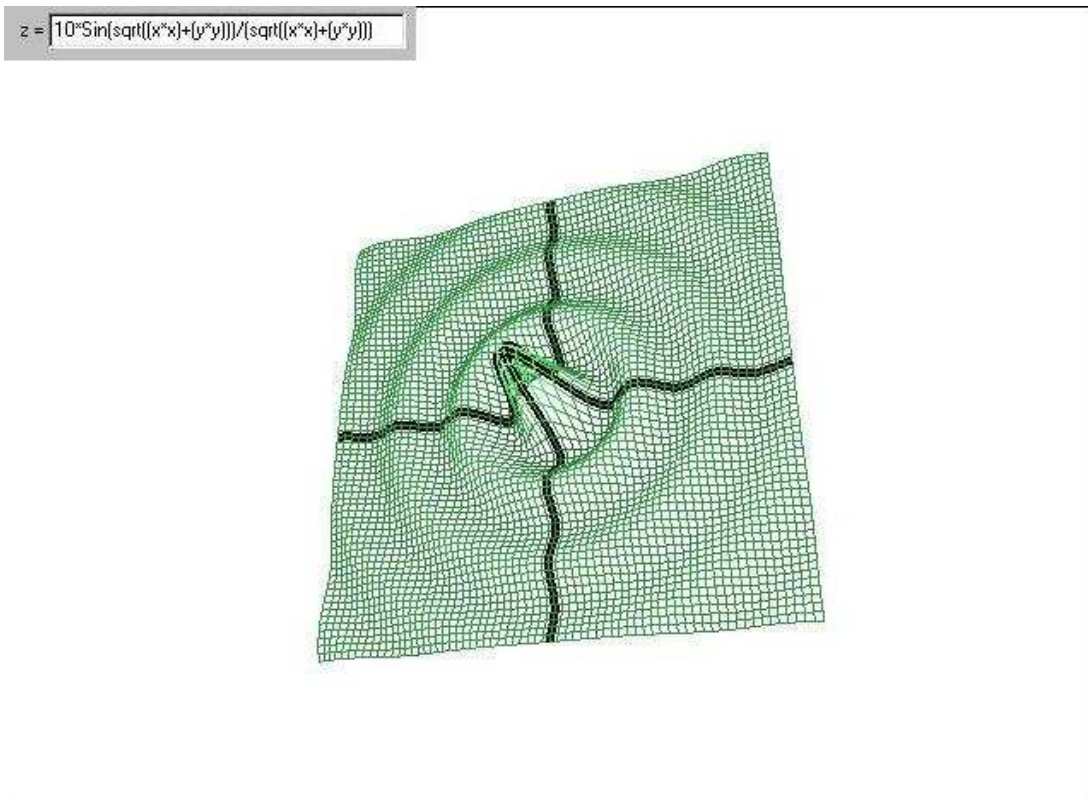

Рис 4. Рабочее окно распределённой программы.

В Таблице 2 приведены результаты(скорость относительно версии программы без параллелизма), полученные на: фактически однородной сети с 12-тью узлами на базе компьютеров с процессорами уровня Celeron 1700, 128-256Mb RAM, 40-80Gb HDD; разнородной сети с 12-тью "типичными" конфигурациями компьютеров от PII-450 до PIV-2.8HT; а также последовательная версия программы, выполненная на узле Celeron 1700, 128Mb RAM, 60Gb HDD.

3.3. Распределенная реализация с использованием агентных технологий.

Применение агентных технологий для реалиации распределённости вычислений обусловлено такими их характеристиками, как мобильность, адаптивность, наличие развитых средств планировки и информационного обмена. Эти качества позволяют организовать более эффективную работу по сравнению с традиционным программированием, а также упростить процесс развёртывания распределённой программы(благодаря свойству мобильности). В качестве конкретной агентной среды использовалась платформа Аглетов фирмы IBM[15]. Эта платформа обладает следующими положительными качествами:

- мобильность агентов;
- развитые средства коммуникации между агентами;
- кроссплатформенность аглетов (благодаря тому, что они построенны на базе Java);
- включение в код агента только тех компонент, которые действительно используются в процессе выполнения программы, а не всех предоставляемых агентной платформой.

Такой подход позволяет в нашем случае более-менее "безболезненно" перейти от обычных распределённых систем (например, на базе сокетов) к агентным распределённым системам — вместо инициализации/финализации механизма сокетов инициализируются/финализируются платформенно-зависимые параметры аглета, взамен вызова send/recv для отправки сообщений в сокетах используются разнообразные средства из com.ibm.aglet.* и com.ibm.agletx.*. Дополнительно приходится реализовывать процесс миграции аглетов с родительского (командного) узла на дочерние, т.е. мобильность.

Алгоритм работы представлен на Рис 5.

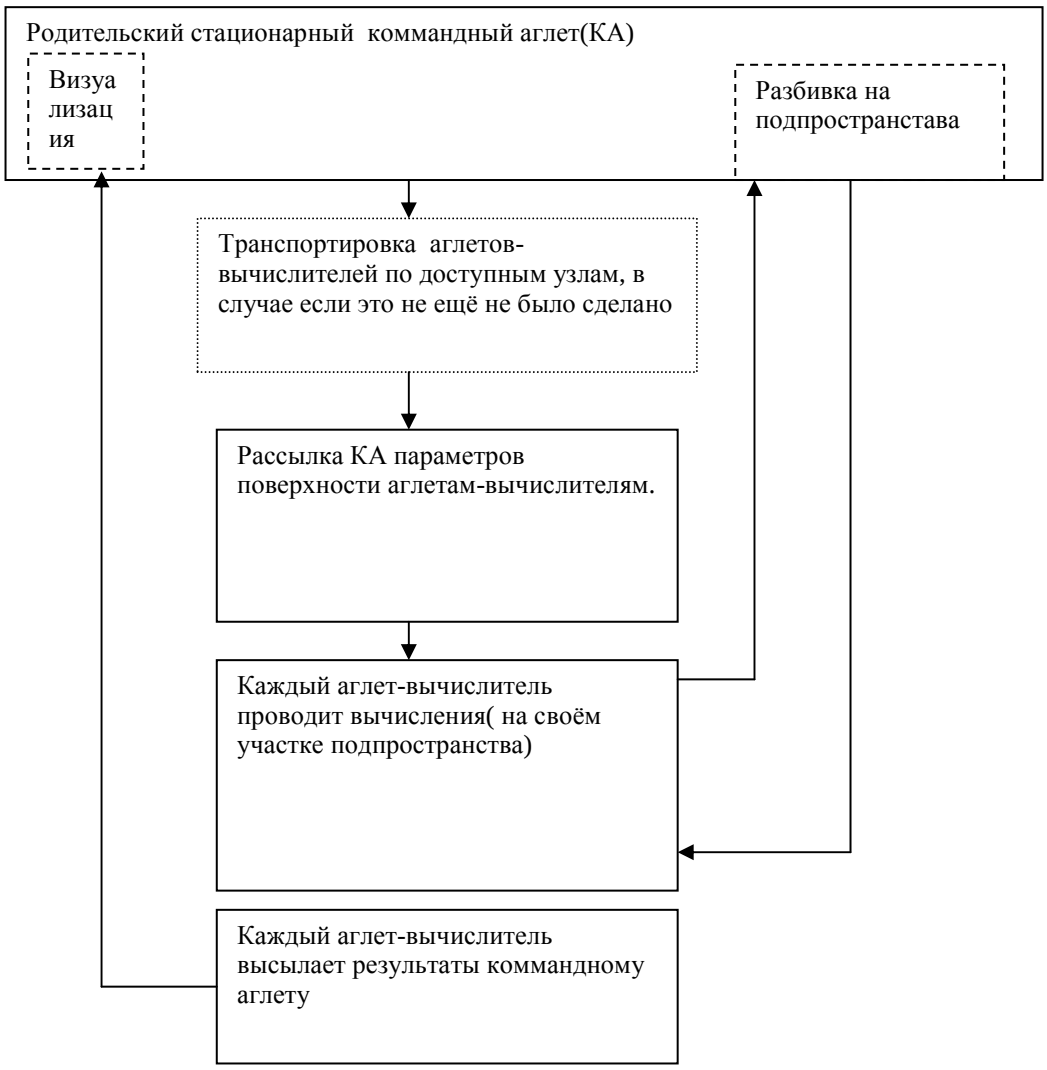

Рис 5.

По завершению расчётов аглеты-вычислители возвращают массивы значений коммандному Аглету, а он проводит визуализацию цельной поверхности.

Результаты производительности (3 вычислителя) можно увидеть в Таблице 2. Как можно видеть производительность программы не меняется
при переходе на другую вычислительную среду, т.е программа не адаптируется к среде, что лишает её способности более эффективно использовать ресурсы платформ.

Таблица 2. Скорость относительно последовательной программы.

Вычислительная среда	Распределённая на однородной сети	Распределённая на гетерогенной сети
Распределённая с использованием сокетов(12 узлов)	7.5	8.2
Распределённая с использованием агентных технологий(3 узла)	2.7	2.7

4. Адаптивность.

Как уже было сказано, любая программа без свойства адаптивности к окружающей среде обречена на недостаточно эффективную работу, что автоматически снижает и уровень отдачи от заложенных в неё усилий и

средств. В большой степени это относится к распределённым приложениям. По-этому при разработке программы необходимо уже на этапе проектирования использовать средства обеспечивающие адаптивность.
С точки зрения поставленной задачи (визуализация поверхности с использованием параллельных вычислений) мы имеем два "бутылочных горлышка"— производительность каждого конкретного вычислительного узла и пропускная способность каналов передачи данных. Для достижения максимальной производительности распределённая программа должна учитывать оба этих параметра при адаптации. Сложностей ещё добавляет и динамический, изменяющийся во времени, характер этих параметров. Так, например, на узле может выполнятся в некоторый момент времени ещё несколько не связанных с нашей программой программ, а канал может использоваться для информационного обмена другими программами (Рис. 6).

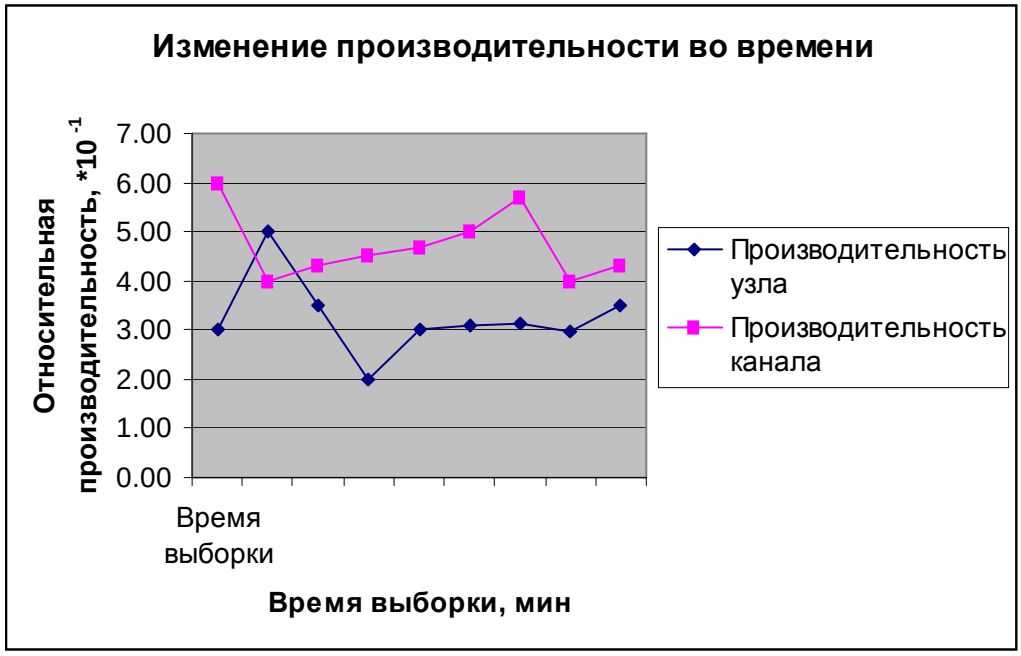

Рис 6. Изменение производительности узла и канала связи во времени.

Пусть относительная производительность узла (отношение производительности данного узла к максимальной производительности в данном кластере; способ определения производительности узла зависит от разработчика) $V_c(t)$, а относительная производительность(отношение пропускной способности данного канала к максимальной пропускной способности в кластере; способ определения производительности узла зависит от разработчика) канала передачи данных $V_n(t)$. Введём новый параметр — интегральная производительность $V_i(t)$, $V_i(t)=K_1*V_c(t)+K_2*V_n(t)$, где K_1-вес относительной производительности узла, а K_2-относительной производительности канала в интегральной производительности. Следует заметить, что $V_c(t)$ и $V_n(t)$ в общем случае не могут быть заранее заданы аналитической функцией, а представляют собой набор дискретных данных полученных в результате замера в конкретные временные моменты. Для построения аналитического представления этих функций необходимо использовать интерполяцию. Во время выполнения работы нами была выбран интерполяционный полином Ньютона для получения аналитических представлений этих функций. Причина тому — сравнительно быстрый расчёт полинома, достаточно большая точность по сравнению с такими полиномами как канонический полином, полином Лагранжа, сплайны; хотя для конкретной задачи платформы может быть выбран и другой.
В соответствии с алгоритмом построения полинома Ньютона имеем:
$V_c(t)=f_0+(t-x_0)(f_{01}+(t-x_1)(f_{012}+(t-x_2)(f_{0123}+\ldots.))$[2,3], где
$f_0=V_c[0]$; $f_{01}=(f_0-V_c[1])/(x_0-x_1)$;
$f_{02}=(f_0-V_c[2])/(x_0-x2)$; $f_{012}=(f_{01}-f_{02})/(x_1-x_2)$ и т.д.;

$V_n(t)=g_0+(t-x_0)(g_{01}+(t-x_1)(g_{012}+(t-x_2)(g_{0123}+\ldots.))$[2,3], где
$g_0=V_n[0]$; $g_{01}=(g_0-V_n[1])/(x_0-x_1)$;
$g_{02}=(g_0-V_n[2])/(x_0-x2)$; $g_{012}=(g_{01}-g_{02})/(x_1-x_2)$ и т.д.,

а x_0,x_1,x_2 и т.д.— момент выборки,
а $V_c[0],V_c[1],V_c[2],V_n[0],V_n[1],V_n[2]$ и т.д.— дискретные значения в моменты выборки x_0,x_1,x_2 и т.д.

Степень полинома подбирается эксперементально, либо исходя из требуемой точности. Теперь мы имеем функцию $V_i(t)=K_1*V_c(t)+K_2*V_n(t)$. Для достижения максимальной производительности необходимо, чтобы при заданных K_1 и K_2

$V_i(t)$ была максимальной для узла в момент t. На практике тут возникают некоторые проблеммы. Например, нашей программе может поступить комманда на расчёт функции z=F(x,y)=sin(x)+y, xє [-100,100], yє [200,300] и в данном случае пропускная способность каналов будет иметь большее значение чем производительность узла. В случае, если у нас K_2 указан меньше чем необходимый для решения задачи, то канал будет использован неэффективно. Если же у нас задана достаточно сложная, с математической точки зрения, поверхность, а K_1 указан меньше чем необходимый, то узел будет простаивать (что и происходило см. в 3.), закончив вычисления раньше чем остальные. Для решения этой проблеммы можно пойти двумя путями:

перед разбивкой на подобласти анализировать формулу на вычислительную сложность, а также размер области определения, и в соответствии с полученными результатами менять приоритеты коэфициентов (динамическая адаптация);
использовать некоторые усреднённые по приоритетности значения K_1 и K_2 (статическая адаптация).
Динамическая адаптация позволяет оперативно, ориентируясь на конкретную задачу по состоянию на конкретный момент времени менять требуемое соотношение производительности узла и канала передачи, но требует большего объёма подготовительных вычислений, в то время как статическая не гарантирует максимальную производительность на конкретном узле в конкретный момент времени, но позволяет фактически непосредственно после поступления задачи начать её исполнение.
Блок схема распределённого приложения представлена на Рис. 7. В случае статической адаптации решение о коэфициентах К1 и К2 происходит единожды, а в случае динамической — для каждой поверхности.

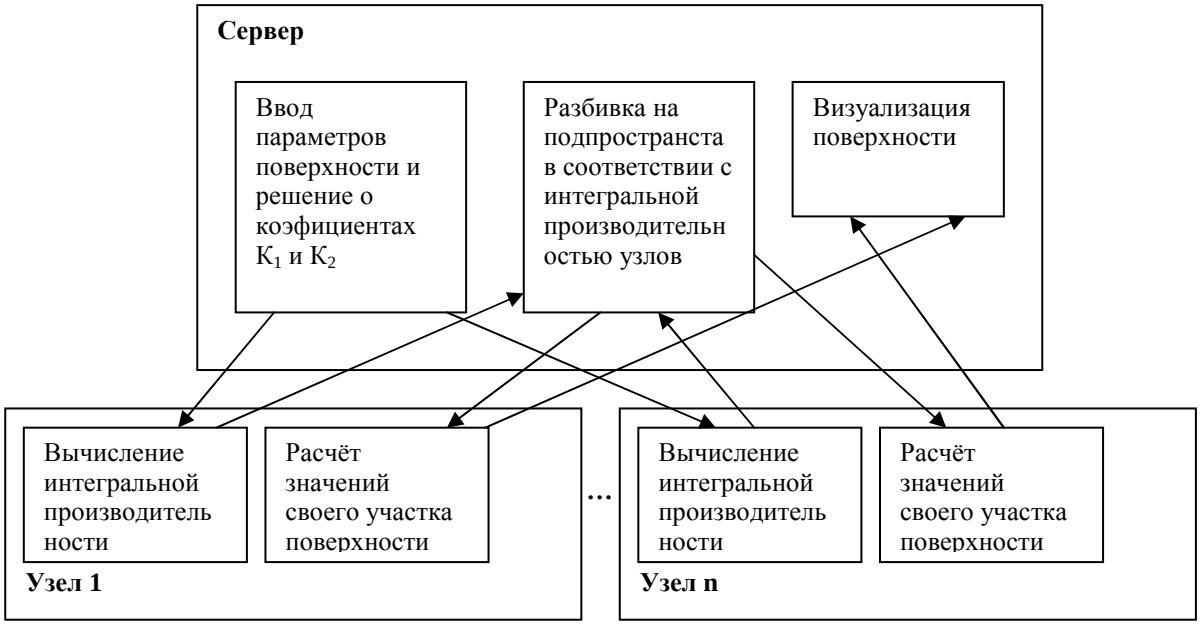

Рис 7. Адаптивное распределённое приложение.

Выводы.

Применение параллельных вычислений для расчёта узлов полигональной сетки для визуализации заданных в явном виде поверхностей позволяет значительно, особенно в случае распределённости системы, повысить скорость вычислений и стабильность (в случае распределённой системы). При этом, единственными накладными расходами при распараллеливании являются: реализация способа деления пространства на подпространства и создание протокола обмена информацией между узлами. Сравнительная простота и дешевизна такого решения является следствием, с одной стороны, архитектуры SPMD и хорошей распараллеливаемости математической задачи, а с другой – примененной методике распараллеливания вычислений, объединяющей в себе использование ефективных вычислительных методов, стандартных средств визуализации широкодоступных средств параллельного программирования.

Вместе с тем, были обнаружены некоторые специфические особенности построения такого рода параллельных программных комплексов, а именно потребность в интелектуализации разбиения пространства на подространства; и интелектуализации стадии обмена данными. Данные проблемы могут быть решены с использованием разработанной методики адаптивности.

Библиографический список

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления.– СПб.: БХВ-Петербург, 608с., 2002.

2. Г.Корн, Т. Корн. Справочник по математике(для научных работников и инженеров). М.:"НАУКА", 832с., 1977.

3. Мудров А.Е. Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. Томск: МП "РАСКО", 272с., 1991.
4. Немнюгин С., Стесик О. Параллельное программирование для многопроцессорных вычислительных систем. СПб.:БХВ-Петербург, 400с., 2002.
5. Румянцев П.В. Работа с файлами в Win32. М.: Горячая линия — Телеком, 200с., 2000 г.
6. Хилл Ф. OpenGL программирование компьютерной графики для профессионалов. СПб.: ИД "Питер", 1088с., 2002
7. Эдвард Эйнджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд. М.:ИД "Вильямс", 592 с., 2001 г.
8. Эндрюс Г.Р. Основы многопоточного, параллельного и распределённого программирования. М.: ИД "Вильямс", 512с., 2003
9. "Открытые системы" №11-12, 1999г. Изд-во "Открытые системы".
10. http://www.altlinux.ru/module=articles&action=show&artid=13&part=234
11. http://www.amlab.ru/Vizpaper_max.htm
12. http://www.amlab.ru/Vizspectrum_moi.htm
13. http://www.dev.dtf.ru/articles/read.php?id=126&page=1..4
14. http://www.ixbt.com/3dterrains-generation.html
15. http://www.trl.ibm.co.jp/aglets/