Программный комплекс для графического представления процесса и результатов работы интервальных алгоритмов

Н. В. Панов, В.В. Колдаков

Новосибирский Центр Информационных Технологий "УниПро",
Россия, 630090 г. Новосибирск, просп. Лаврентьева, 6/1
rupo@ngs.ru kold@nbsp.nsk.su

Аннотация. Предметом нашей работы является наглядное представления процесса выполнения и результатов работы различных алгоритмов глобальной интервальной оптимизации функции двух переменных.

1 Постановка задачи

Данная работа посвящена разработке и реализации пакета программ, позволяющих ставить, решать задачу глобальной оптимизации функции двух переменных интервальными методами и наглядно демонстрировать процесс и результаты работы различных алгоритмов.

Реализованный программный комплекс основан на клиент-серверной архитектуре. Включает в себя мультиплатформенную клиентскую часть со встроенным собственным трехмерным ядром и серверную часть, объединяющую собственную программу-сервер и решатель, в свою очередь состоящий из интервального ядра, библиотеки алгоритмов и символьного интерпретатора.

Эта система призвана способствовать популяризации интервальных методов, в частности, методов глобальной оптимизации и создавалась с рекламно-демонстрационной целью, но может рассматриваться и как полноценный инструмент для решения двумерных задач глобальной оптимизации.

2 Специфика задачи

Математических пакетов с поддержкой интервальной математики все еще мало, общедоступные не очень удобны, имеется ряд существенных недостатков. При проектировании нашей системы, мы старались избежать основных недостатков существующих решений, мешающих программам такого рода завоевать массового пользователя:

1) “ресурсоемкость” (требуют от пользователя большой мощности процессора, много места на диске и оперативной памяти)
2) платформеннозависимость.

Использование клиент-серверной архитектуры позволяет разгрузить клиентскую машину, “унеся” все тяжелые вычисления на серверную сторону. Это не только практически снимает ограничение на мощность пользовательской машины, но и позволяет повысить скорость и точность результатов, за счет использования мощных серверных станций. Выбор Java в качестве языка реализации визуализационной части гарантирует мультиплатформенность. Математическая часть выполнена на языке C++, более подходящего для этих целей.

Одно из основных условий - система должна накладывать как можно меньше ограничений на потенциального пользователя и его рабочее место.

Вся работа осуществляется через сеть Internet при помощи любого Веб-браузера с поддержкой языка Java (Internet Explorer, Netscape Navigator, Mozilla, Opera), но без дополнительных модулей (например, для поддержки трехмерной графики).

Так как системе предстоит работать с трехмерной графикой, но, возможно, у пользователя не установлена система DirectX или OpenGL, а требовать от пользователя установить их мы не можем, встала необходимость создания собственного трехмерного ядра для клиентской части. Оно реализовано в виде внутреннего класса. В его задачи входит не только отрисовка трехмерных объектов средствами двумерной графики, но и немалая математическая часть:

1) вся рутина по вычислению проекций;
2) пересчет координат при повороте и прочие матричные преобразования;
3) расчет затененных областей;
4) удаления невидимых граней.

Далее взаимодействие с внутренним трехмерным ядром остальной программой осуществляется через предоставляемый набор стандартных интерфейсов, как если бы она пользовалась услугами, скажем DirectX.

В задачи сервера входит

1) непосредственно отслеживание и поддержание соединений,
2) аутентификация пользователей и разделение прав,
3) перекодировка и передача информации между клиентом и системой решения,
4) запуск решателя и постановка задачи для него.

Особенность серверной части в том, что она должна сочетать в себе модули, написанные на языках Java и C++. Серверная часть реализована в двух вариантах — в виде сервлета (Java Servlet) и в виде самостоятельного сервера. В силу большой специфичности круга задач, решаемых сервером и особенностей архитектуры, программа-сервер создавалась "с нуля", без использования готовых основ.

Математический решатель (Solver) состоит из интервального ядра, библиотеки алгоритмов, модуля обеспечения и символьного анализатора.

Интервальное ядро реализует представление типа данных «интервал» и набор базовых функций. Библиотека функций — расширяемый набор интервальных алгоритмов. Модуль обеспечения — отвечает за ввод-вывод, синхронизацию, целостность системы и взаимодействие с сервером (пользователем).

3 Краткий перечень требований, предъявляемый к создаваемому программному обеспечению

Разрабатываемое программное обеспечение должно позволять пользователю ставить задачу глобальной оптимизации. Поставленная задача должна быть решена за конечное время. Решение должно быть гарантированным, пропуск какого-либо решения недопустим. Создаваемое ПО должно обеспечивать наглядную демонстрацию работы интервальных алгоритмов решения задачи глобальной оптимизации. Результаты работы должны быть также представлены в наглядном виде. Система должна накладывать как можно меньше ограничений на потенциального пользователя и его рабочее место.

4 Описание алгоритмов трехмерной графики — приводится здесь в весьма сокращенном виде, порой, не затрагивая целые классы проблем, без решения которых трехмерное графическое ядро невозможно. Заинтересованного читателя мы отсылаем к работам [1, 3], подробно излагающим данный вопрос.

4.1 Матричные преобразования

В трехмерном пространстве любое движение, согласно теореме Шаля, может быть представлено в виде суперпозиции поворота и параллельного переноса. И именно поэтому нам достаточно знать, как сделать два преобразования — *перенос* и *поворот*.

4.1.1. Перенос точки. Точки будут рассматриваться как вектора с началом в начале координат и концом в собственно точке. Перенос — это сложение вектора-точки и вектора-переноса.

4.1.2. Поворот. Это занятие уже более интересное. Но тоже простое. Рассмотрим для примера поворот точки (x,y,z) относительно оси z. В этом случае z не меняется, а (x,y) меняются так же, как и при 2D повороте относительно начала координат. При этом координаты точки A' — результат поворота A(x,y) на угол можно узнать по формулам поворота в плоскости, для других осей поворота — аналогично.

$$x' = x*\cos(\alpha)-y*\sin(\alpha)$$
$$y' = x*\sin(\alpha)+y*\cos(\alpha)$$
$$z' = z$$

Можно заметить, что $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \quad \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

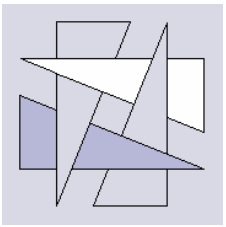
Но умножение матрицы на вектор требует больше операций, чем расчет x' и y' по формулам. Удобство матриц заключается как раз в свойстве A*(B*C) = (A*B)*C. Если мы делаем несколько поворотов подряд, например, пять (столько надо для поворота относительно произвольной оси), то, посчитав один раз матрицу преобразования, являющуюся комбинацией пяти поворотов, можно в дальнейшем без проблем делать сложное

преобразование из пяти поворотов с помощью всего одного умножения матрицы на вектор. Таким образом, можно задать любой поворот матрицей, и любая комбинация поворотов также будет задаваться матрицей, которую легко посчитать.

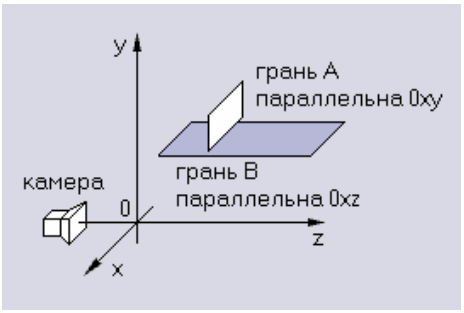
4.2 Удаление невидимых граней

4.2.1. Отброс нелицевых граней. Пусть у нас есть объект, внутри которого камера заведомо не окажется. Обычно такие объекты составляют большую часть или всю сцену. Тогда для каждой грани мы можем увидеть только одну ее сторону — лицевую, ее видно только из одного полупространства, из того, в которое "смотрит" нормаль к этой грани, направленная "из" объекта. Проверив, в какое полупространство попадает камера, можно сразу определить, является ли грань лицевой и надо ли ее рисовать.

Отдельный вопрос - как считать нормали к граням. Точнее, как выбрать одну из двух нормалей, которая будет смотреть из объекта. Обычно эта проблема решается еще на этапе построения 3D моделей - например, пакет 3D Studio записывает вершины граней в порядке A-B-C так, чтобы векторное произведение BA*CA и было нормалью. Еще один способ - выбрать внутреннюю точку для объекта (либо вручную, либо взять его центр тяжести, либо еще как-нибудь) и использовать ее: если для этой точки функция видимости положительна, то есть грань якобы видна, то надо поменять знак nx, ny и nz. Для выпуклых объектов этот метод полностью решает задачу об удалении невидимых частей.



4.2.2 Алгоритм художника. Пусть имеется некий набор граней (т.е. сцена), который требуется нарисовать. Отсортируем грани по удаленности от камеры и отрисуем все грани, начиная с самых удаленных. Довольно распространенная характеристика удаленности для грани ABC - это среднее значение z, mid_z = (A.z+B.z+C.z)/3. Вот и весь алгоритм. Просто, и обычно достаточно быстро. Существует, правда, несколько проблем. Во-первых, при некотором расположении граней этот алгоритм вообще не может дать правильного результата - в каком порядке грани не рисуй, получится неправильно. Стандартный пример справа. Во-вторых, при некотором расположении граней и использовании среднего значения z как характеристики удаленности алгоритм тоже дает неправильный результат. Пример слева. В этом случае горизонтальную грань надо рисовать второй, но по среднему значению z она лежит дальше и таким образом получается, что ее надо отрисовывать первой. Возможные пути решения этой проблемы - какие-то изменения характеристики удаленности, либо моделирование, не вызывающее таких ситуаций. И наконец, при использовании этого алгоритма отрисовываются вообще все грани сцены, и при большом количестве загораживающих друг друга граней мы будем тратить большую часть времени на рисование невидимых в конечном итоге частей. То есть совершенно впустую.



4.2.3. Z-буфер. Заведем буфер (собственно z-буфер) размером с экран, и забьем его каким-то большим числом. Для каждой рисуемой точки считаем значение z; если оно больше, чем значение в z-буфере (точка закрыта какой-то другой точкой), или меньше, чем -dist (точка находится за камерой), то переходим к следующей точке. Если меньше, то рисуем точку, а в z-буфер записываем текущее значение z. Вот и все. Это самый простой метод удаления невидимых частей, причем всегда дающий полностью правильные результаты.

5 Интервальные алгоритмы глобальной оптимизации функций нескольких переменных

Для решения задачи глобальной оптимизации применяются интервальные методы, основанные на адаптивном дроблении области определения в сочетании с оцениванием области значений по получающимся подобластям. Методы этого типа хорошо работают для функций сложного рельефа, надежно находя гарантированные оценки как для глобального оптимума, так и для доставляющих его значений аргументов.

5.1 Бисекция

Классический метод, на каждой итерации выбирает брус, доставляющий наихудшую оценку и делит его пополам.

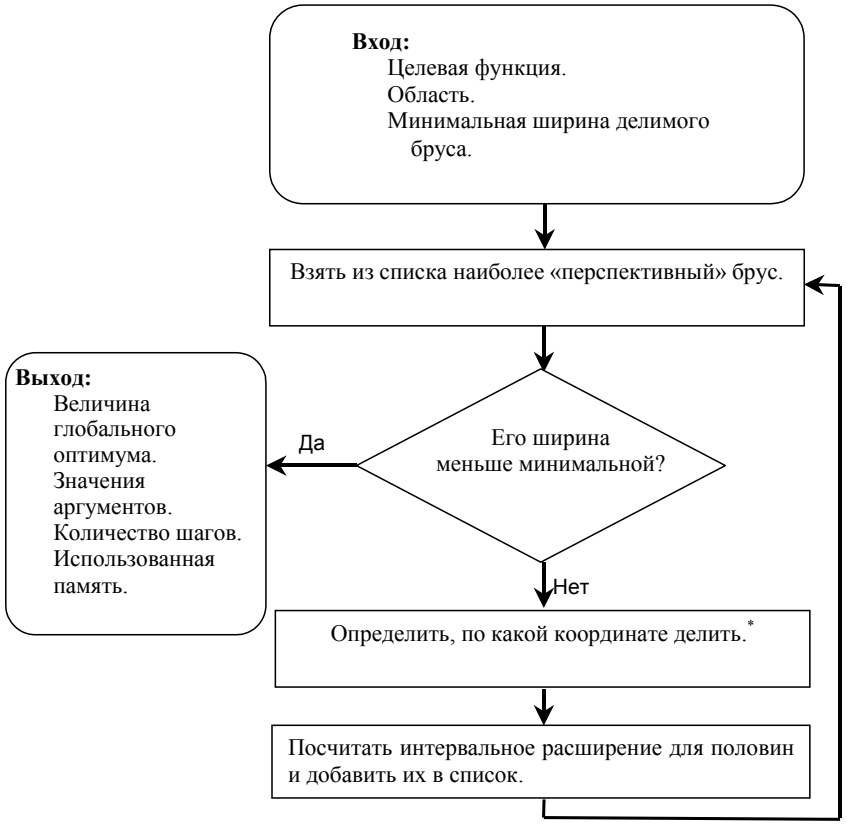
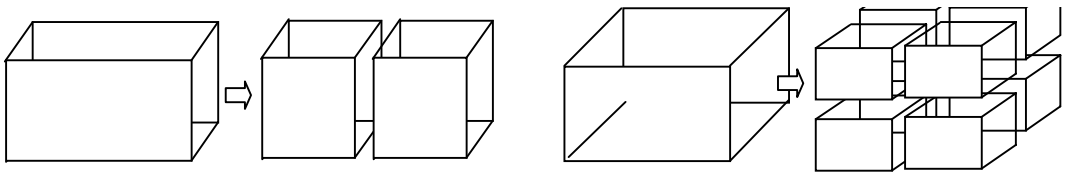


Рис 1. Блок-схема алгоритма глобальной оптимизации с использованием «Бисекции»
*) Деление может осуществляться несколькими способами.

– Естественное дробление (Дробление большой стороны).
За одну итерацию алгоритма выполняется одно деление. Для деления выбирается более широкая сторона бруса. Это связано с тем, что точность интервальной оценки зависит от ширины интервала, в котором меняются аргументы, и от зависимости целевой функции по каждому из аргументов. Не делая никаких предположений о конфигурации функции, алгоритм последовательно уточняет интервальное расширение, деля наибольшую, а значит, вероятно, дающую наименее точную оценку сторону. Например, брус такого вида (рис. 2а) будет поделен таким (рис. 2б) образом.



а) б) в) г)

Рис 2. а), б) — деление по большей стороне. в), г) — тотальное деление

– Тотальное дробление (Дробление всех сторон).
За одну итерацию алгоритма брус делится на восемь частей. Пример дробления по всем сторонам на рис. 3

– Дробление с памятью (Эвристическое дробление).

За одну итерацию алгоритма выполняется одно деление. Сторона для деления выбирается исходя из предыдущих итераций, используется вместе с алгоритмом естественного деления (деление большой стороны). Каждый раз, производя деление очередного бруса, этот алгоритм анализирует, насколько улучшилась интервальная оценка при делении именно по этой координате. В существующем варианте надежд не оправдал.

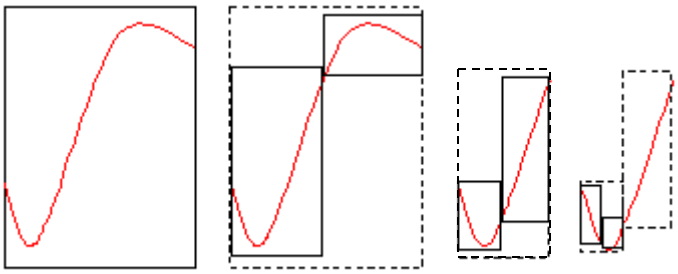


Рис 3. Поиск минимума методом «бисекции»

Вот пример, иллюстрирующий работу бисекции. Ищем минимум.

Бисекция — классический алгоритм, его достоинства — простота реализации и неприхотливость. В рамках данной работы был разработан ряд алгоритмов, также основанных на адаптивном дроблении.

5.2 Бисекция с перекрытием (Трисекция)

Алгоритм, основанный на бисекции. Деление происходит не пополам, а на две пересекающиеся части, по две трети их исходного бруса каждая.

Если при сравнении интервальных расширений получившихся брусов, рекордную оценку доставляет левый брус, делается предположение, что экстремум находится в первой трети, если правый брус рекорден, то наиболее вероятное положение экстремума для этого этапа — в последней трети. Таким образом, для каждой следующий итерации алгоритма будет получаться область в три раза меньше текущей. Но случай, когда оценки равновелики (Рис.3 *б*), требует отдельного рассмотрения. В этом случае необходимо проверить, не имеем ли мы дело с двумя равновеликими экстремумами в первой и третьей третях. Если это действительно так, алгоритм прекрасно распараллеливается, и каждый экстремум в дальнейшем обрабатывается в отдельном потоке. Если это не так, алгоритм предполагает, что экстремум находится во второй трети.

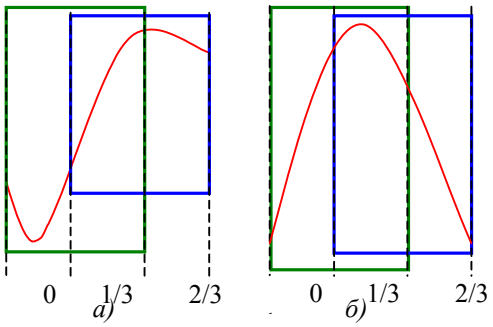


Рис 4. Пример разбиения исходной области перекрытием. (Метод «трисекции»)

Применение этого алгоритма для относительно хороших функций с единственным экстремумом на области дает выигрыш в полтора раза.

5.3 Алгоритмы с неравномерным дроблением

Эти алгоритмы отличаются от описанных выше тем, что деление очередного бруса происходит на неравные части, в зависимости от предыдущего дробления.

На рис.7 рассмотрено поведение подобных методов на основе алгоритма Неравномерной бисекции.

5.4 Вероятностный алгоритм

Другим популярным подходом к решению задач глобальной оптимизации является так называемый алгоритм *«симулированного отжига»* (известный еще как *алгоритм Метрополиса*). Это вероятностный метод, моделирующий одноименный физический процесс, и его имеет смысл применять в тех ситуациях, где доминирующими требованиями является желание получить в сложной задаче хоть какие-то результаты за заданное

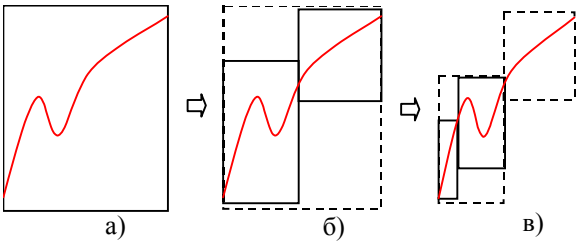


Рис 5. Пример работы алгоритма «Неравномерная бисекция»

ограниченное время. Разработанная С.П.Шарым и впервые реализованная в рамках данной работы интервальная версия алгоритма «симулированного отжига» для глобальной оптимизации функций сочетает в себе гарантированность интервальных методов с неприхотливостью вероятностного подхода.

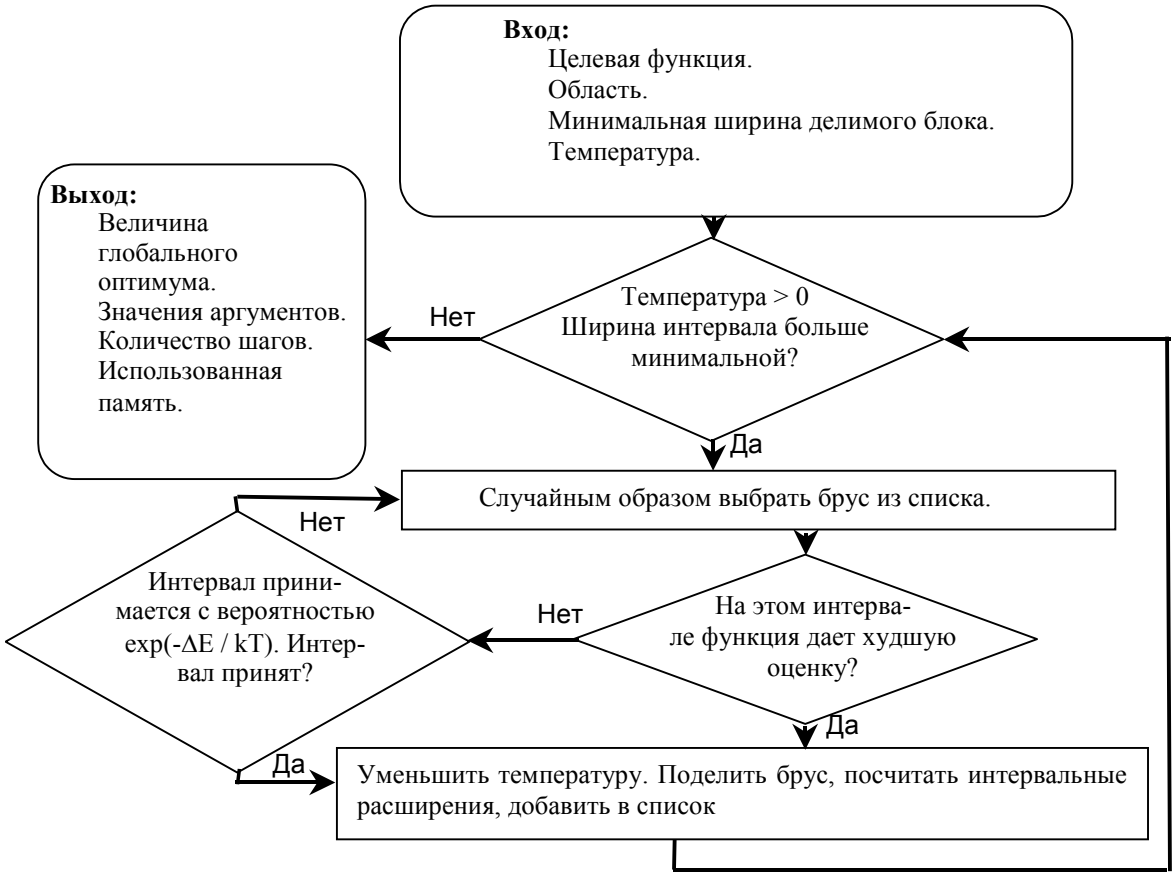


Рис 6. Блок-схема алгоритма интервального симулированного отжига для глобальной оптимизации.

6 Реализация

6.1 Решатель

Решатель реализован на языке C++. Платформеннозависим. Существующая реализация позволяет осуществлять переход между операционными системами Windows (поддерживается вся линейка, начиная с Windows 95) и клонами Linux без изменения кода. Поддержки графического интерфейса пользователя нет. Самодостаточен.

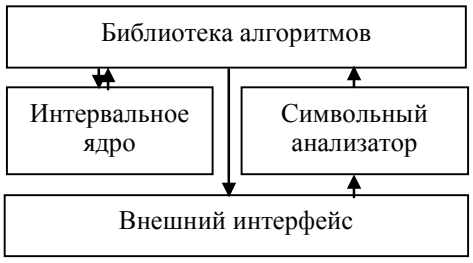


Рис 7. Структурная схема решателя.

Интервальное ядро реализует представление типа интервал и базовые функции. Может выступать в роли самостоятельного продукта. В библиотеке алгоритмов реализованны функции высокого уровня, алгоритмы глобальной оптимизации, методы построения графиков функций (поверхностей). Символьный анализатор позволяет ставить задачу, используя командную строку или файл инициализации. Позволяет определять константы, функции, переменные, понимает данные интервального типа, допускается ввод дробных чисел и чисел в экспоненциальном формате.

6.2 Сервер

Сервер реализован на языке Java в двух вариантах. В виде java программы и с использованием технологии Servlet.

Сервлет не является самостоятельной программой и должен исполняться в рамках какого-либо HTTP-сервера, поддерживающего работу сервлетов. Кроме того, он немного тяжелее в исполнении, хотя при повторных обращениях ситуацию несколько улучшает кэш сервера, пытаясь при этом подсунуть старые данные. При обращении к сервлету вызывается функция doGet. В ней, в зависимости от строки запроса либо ставится задача решателю, либо вызывается соответствующая native-функция для получения и обработки требуемых данных. Затем они посылаются клиенту, либо ему сообщается, что данные не готовы.

Такую же функциональность реализует TCP-IP сервер, для его работы не требуются какие-либо дополнительные программы. Поддерживает множество одновременных соединений.

Сервер и Решатель общаются через Посредника. Посредник кодирует и форматирует данные, решает проблемы разделения доступа и требований безопасности Java машины. Посредник реализован в виде динамически подключаемой библиотеки на языке C++.

6.3 Реализация клиентской части

Клиентская часть реализована на языке Java в виде аплета, предоставляющего графический интерфейс пользователя со встроенным трехмерным ядром, собственной системой меню, диалогов и командной строкой.

Список литературы

[1] Калмыков С.А., Шокин Ю.И., Юлдашев З.Х. *Методы интервального анализа*. — Новосибирск, Наука, 1986.
[2] Фоли Дж., Дэм А.. *Машинная графика.* — 2т., Москва, Мир, 1988.
[3] Шарый С.П. *Интервальные алгебраические задачи и их численное решение.* — Диссертация на соискание ученой степени доктора физико-математических наук, Новосибирск, ИВТ СО РАН, 2000 г. — 327 с. Текст в Internet доступен по адресу http://www.ict.nsc.ru/interval/Library/InteDiss
[4] Hansen E.R. *Global optimization using interval analysis*. — Marcel Dekker, 1992.
[5] Kearfott R.B. *Rigorous Global Search: Continuous Problems.* — Dordrecht, Kluwer, 1996.
[6] http://www.enlight.ru/faq3d/main.html — *demo.design 3D. Programming FAQ*.

Приложение

Структура комплекса