Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Волков Никита Игоревич

Система web-визуализации информационных графов для проекта AlgoWiki

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
вед.н.с., к.ф.-м.н.
Антонов Александр Сергеевич

Москва 2018

1. Аннотация

Основная цель данной работы — разработка и реализация программной системы web-визуализации графов алгоритмов. Данная система должна обладать следующими свойствами: во-первых, возможность автоматического преобразования описания графа алгоритма в формате некоторого известного внутреннего представления в описание его же в одном или нескольких стандартных форматах, используемых для хранения 3D-моделей; во-вторых, возможность визуализации полученных 3D-моделей в интерактивной среде как частей описания соответствующих графов алгоритмов в рамках проекта AlgoWiki.

Решение указанной задачи подразумевает многоэтапный процесс, который включает в себя:

1) Формирование понятия внутреннего представления графа алгоритма.

2) Создание шаблона и интерфейса web-страницы, на которой производится визуализация графа алгоритма.

3) Создание программного средства, преобразующего внутреннее представление графа алгоритма в формат 3D модели

4) Создание программного средства, производящего непосредственно web-визуализацию предложенной 3D-модели в рамках web-страницы и обеспечивающего его интерактивность.

5) Интеграция разработанных программных средств в открытую энциклопедию свойств алгоритмов AlgoWiki [1].

2. Введение

Под информационным графом алгоритма понимается [2] ациклический граф, вершины которого соответствуют операциям алгоритма, а дуги — связям по данным между этими операциями. При этом в классическом определении графа алгоритма вершины соответствуют базовым операциям. Визуально граф может быть представлен как в произвольном виде, так и в так называемой ярусно-параллельной форме [2], в которой на одном уровне (ярусе) располагаются все вершины, которые могут быть выполнены параллельно на одном этапе работы алгоритма. Под этапами работы алгоритма подразумевается следующее: N-ый этап может быть выполнен только после завершения предыдущих N–1 этапов. К первому этапу относятся те операции в алгоритме, которые не требуют никаких предварительных действий в рамках этого алгоритма. Структура и сложность графа полностью определяются самим алгоритмом. В общем случае искомый граф может иметь сколь угодно большое число вершин, а отношение числа дуг к числу вершин ограничено таковым у полносвязного графа с соответствующим числом вершин.

Автоматическая визуализация графа алгоритма в виде интерактивной 3D-модели как элемента описания алгоритма направлена на решение ряда задач: во-первых, на обеспечение наглядного представления внутренней структуры алгоритма, в том числе и для лиц, не являющихся экспертами, либо вообще не имеющими отношения к области, в которой применяется описываемый алгоритм; во-вторых, на облегчение процесса подробного анализа алгоритма для составления его описания, содержащего все характерные особенности алгоритма как на уровне собственно алгоритма, так и на уровне его конкретных реализаций; в-третьих, на освобождение составляющего стандартизованное описание алгоритма от необходимости вручную производить его непосредственную визуализацию.

Для достижения второй цели в полной мере интерактивная 3D-модель графа алгоритма должна содержать вспомогательную информацию и быть структурирована таким образом, чтобы обеспечить максимальную ясность как принадлежности отдельных дуг отдельным вершинам, так и общей логической структуры алгоритма, а также компенсировать возможную сложность внутренней структуры алгоритма собственной простотой восприятия. Для достижения третьей цели необходима четкая взаимосвязь со схемой работы реализации этого алгоритма и с исходным кодом реализации.

Собственно автоматизированность системы визуализации направлена на достижение четвертой цели.

Пятая и последняя цель работы задаёт основную область для практического применения полученных результатов, а именно их использование как часть описаний алгоритмов в рамках проекта AlgoWiki. Проект AlgoWiki, который развивается на протяжении уже нескольких лет, ставит своей главной целью дать исчерпывающее описание алгоритма, которое поможет оценить его потенциал применительно к конкретной параллельной вычислительной платформе. В такое описание включаются как свойства алгоритмов на машинно-независимом уровне собственно алгоритмов, методов или задач [15], так и на уровне их особенностей их реализации на самых различных платформах [16]. К машинно-независимым свойствам относятся, например, параллельная структура, параллельная сложность, ресурс параллелизма, которые как раз и могут быть хорошо отображены с помощью информационного графа алгоритма в произвольной или ярусно-параллельной форме. Следует особо отметить, что в процессе построения интерактивной визуализации первичным «сырьём» является всё же программная реализация нужного алгоритма на языке C или Fortran. Однако подразумевается, что эта реализация референсная, без оптимизаций и перестроения алгоритмов под конкретные вычислительные платформы, то есть реализация машинно-независимая. Чтобы подчеркнуть родственность с проектом AlgoWiki, было решено назвать систему AlgoView.

3. Постановка задачи

Предполагается, что имеется строго стандартизованное внутреннее (не в виде изображения) представление графа алгоритма. В качестве конкретного представления используется форма отображения графа алгоритма через .XML файл. Подробное описание структуры этого файла можно найти в [5]. Здесь же приведены его свойства, имеющие наибольшее значение в рамках этой работы:

1) В нём представляется совокупность всех возможных графов алгоритмов.
2) Конкретный граф алгоритма определяется значениями внешних параметров.

Первым этапом работы является создание программы, принимающей на вход указанный .XML файл и создающей по нему набор 3D-моделей файла в общеупотребительном формате. В процессе работы программой могут быть запрошены соответствующие внешние параметры. Полученные 3D-модели должны удовлетворять ряду свойств, а именно:

1) Корректность при любых графах алгоритма, который можно описать в искомом формате .XML файла при внешних параметрах (таких, как объём входных данных) порядка нескольких десятков. Большие значения не требуются, т. к. рост размера информационного графа алгоритма отрицательно сказывается на удобстве его визуального восприятия.
2) Генерируемые 3D-модели должны содействовать достижению целей 1)-4) из [4] и в максимально возможной для 3D-модели степени соответствовать стандарту визуализации «3D графа» оттуда же, за исключением требования наличия вспомогательной информации, которой в самих моделях быть не должно.

Второй этап подразумевает интерактивную web-визуализацию полученного набора 3D-моделей посредством технологии WEBGL [6]. Данный процесс подразумевает создание соответствующей web-страницы и использование её в качестве графического интерфейса для программы визуализации. При этом необходимо соблюсти следующие требования:

1) Интерактивная визуализация должна работать в режиме реального времени на ПК рядового пользователя и иметь интерфейс для взаимодействия с последним. Визуализация по возможности должна быть

хороша с эстетической точки зрения, объективные критерии этой характеристики будут описаны позднее.
2) В визуализацию должно включаться как можно больше дополнительной информации согласно требованиям стандарта визуализации из [7].
3) Система визуализации AlgoView должна быть совместима с программным движком MediaWiki. Web-страницы с примерами интерактивных моделей графов алгоритмов будут включаться в страницы описания алгоритмов в рамках проекта AlgoWiki, который базируется на указанном движке.

4. Обзор существующих решений

Визуальное представление графов в целом – задача распространённая, по этой теме имеется целый ряд исследований и рабочих решений. Понятие графа алгоритма не так широко известно, но неявным образом встречается в научных трудах по параллельным вычислениям и алгоритмике, например [8].

Основная задача, решаемая с помощью инструментов визуализации графов – автоматическое построение графов с кластеризацией вершин по некоторым признакам. При этом граф может иметь очень большие размеры (миллионы, десятки миллионов вершин), а регулярная структура у него отсутствует. Примерами решений по отображению таких графов могут служить [9] и [10].

Интерес в данном случае представляют размеры отдельных кластеров, признаки, по которым ведётся кластеризация, а также количество связей (дуг в графе) между вершинами из разных кластеров. Удобное для человеческого восприятия выделение последнего свойства в некоторых случаях обозначается как отдельная задача, и она «идеологически близка» к нашей, хотя может быть порождена совершенно иными научными дисциплинами.

Одним из примеров визуализации графа, где требуется решение описанной выше задачи, может служить отображение зависимостей внутри программных систем. В таких графах основной интерес как раз и представляет порядок связей между различными вершинами. Кроме того, в большинстве случаев используется отображение графа на плоскости, а не в пространстве. Для удобной демонстрации этих связей используются методы, являющиеся развитием описанного в [11]. Пример визуализации графа без и с применением подобного метода представлен ниже. Основными отличиями нашей задачи от упомянутых выше с точки зрения теории графов являются малые размеры самого графа и его регулярная структура, которую необходимо показывать с точностью до дуги. В этом смысле гораздо более близкой задачей является визуализация с целью анализа и исследования моделей молекул, в том числе и в режиме on-line. Примерами работы таких систем могут служить web-ресурсы, такие как [12] и [13]. Принципиальное отличие этого подхода от предлагаемого нами — только в предмете исследования. Моделирование молекул, в том числе компьютерное — зрелая область науки, намного превосходящая по возрасту технологию WEBGL. В частности, существует целый ряд специализированных форматов (PDB, MOL, …), предназначенных для хранения соответствующих 3D моделей. Поэтому успех 3D-моделирования молекул и возможности имеющихся в

этой области средств являются некоторым идеалом, к которому нужно стремиться. Ниже демонстрируются примеры работы подобных систем.

Родственной эта задача является и в смысле использования WEBGL непосредственно для отображения молекул в браузере. WEBGL – широко распространённая технология, представляющая собой программную библиотеку для языка JavaScript. Её главная цель – создание на JavaScript интерактивной 3D-графики, функционирующей в web-браузерах. Она применяется в широком спектре практических задач, таких, как создание погодных карт[14], 3D-проектирования и дизайна помещений, и даже наработок по созданию браузерных CAD-систем. С ключевыми концепциями WEBGL можно ознакомиться здесь [15].

5. Исследование и построение решения

Данная работа является частью создаваемой системы по преобразованию программного кода референсных реализаций алгоритмов в интерактивные 3D-модели соответствующих графов, доступные в режиме on-line. Эта система представляет собой цепочку преобразований следующего вида: *программный код → внутреннее представление → разрешенное внутреннее представление (формат SGM) → представление в формате .xml → некоторый формат хранения 3D-модели → интерактивная модель в браузере*. В рамках работы осуществляются последние 2 перехода, что и обуславливает её двухэтапность.

1. Теоретическая основа формирования визуализации информационного графа алгоритма

Система визуализации информационных графов алгоритмов работает, исходя из нескольких базовых положений о методике изображения информационных графов алгоритмов и требований к итоговому внешнему виду визуализации информационного графа алгоритма.

Во-первых, в рамках проекта AlgoWiki существует т.н. «стандарт визуализации графа алгоритма», в котором даны базовые рекомендации для самостоятельного создания двумерной визуализации графа алгоритма, а именно определены используемые геометрические примитивы, их характеристики и комбинации, используемые для отображения особенностей графа алгоритма. Основные рекомендации из искомого стандарта перенесены и в разработанную систему построения интерактивных визуализаций.

Принципы работы системы визуализации исходят из следующих утверждений: во-первых, любой граф можно отобразить в трёхмерном пространстве так, чтобы его вершины и рёбра не пересекались между собой; во-вторых, для каждого набора входных данных алгоритма (а в случае недетерминированного алгоритма – для каждого его прогона) получаемый граф алгоритма всегда представляет собой набор взаимосвязанных гнёзд циклов конечной и известной для каждого гнезда глубины; в-третьих, объединяя группы вершин, соответствующих операциям в алгоритме, в одну макрооперацию, всегда можно добиться представления графа алгоритма в

виде последовательных гнёзд циклов, причём каждое гнездо будет иметь глубину не больше 3, что поможет построить удобное 3D-представление.

Основным методом при создании трёхмерной интерактивной визуализации является следующий: вводится трёхмерная декартова система координат и сетка фиксированного размера на ней. Любая вершина информационного графа алгоритма располагается в уникальном узле сетки и задаётся координатами этого узла. Любая дуга информационного графа алгоритма задаётся парой узлов сетки – её началом и концом. Таким способом можно отобразить в трёхмерном пространстве любой информационный граф алгоритма, однако результирующая визуализация может не удовлетворять эстетическим и, главное, функциональным требованиям.

Ключевое выдвигаемое к визуализации требование эстетического характера – отсутствие наложения дуг информационного графа алгоритма друг на друга и, по возможности, избегание пересечений дуг друг с другом. Это требование отчасти и функциональное – визуализация графа с наложением его дуг друг на друга может дать неверное понимание структуры алгоритма. Так, на рисунке 1 даны три простых графа алгоритма. Верхний граф отражает серию последовательных операций, два остальных – массовую рассылку данных. Однако из-за наложения дуг графа друг на друга два верхних графа визуально неотличимы. Для решения этого вопроса используется следующий подход. Каждая дуга графа алгоритма в визуализации отображается не прямой линией, а кривой Безье второго порядка. На рисунке 1 с использованием этого подхода отображен нижний граф, по своей структуре идентичный второму верхнему графу. Как видим, первый верхний и нижний графы, имеющие разную структуру, теперь визуально различимы, чего и требовалось достичь.

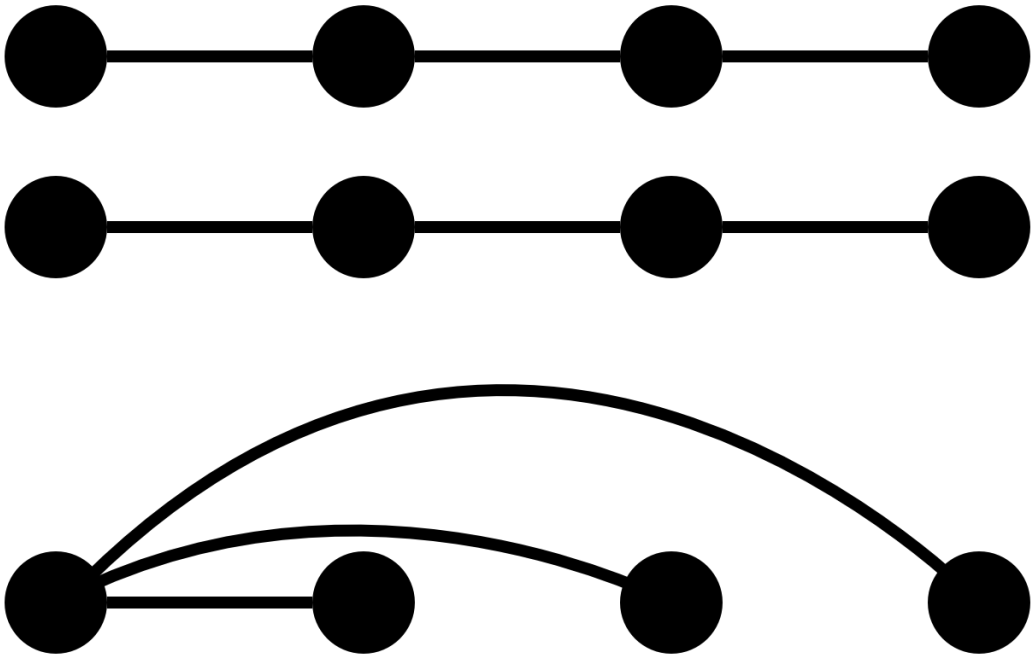

Рис.1 Первый и второй граф - разные. Второй и третий - одинаковые

Как известно, кривая Безье второго порядка задаётся по трём точкам, а из представления дуги информационного графа алгоритма известно только две. Для нахождения третьей точки производится следующая процедура:

1. Вычисляется длина отрезка L, соединяющего начало и конец будущей дуги информационного графа алгоритма.
2. Начало дуги помещается в начало координат, а конец дуги – в точку (0, 0, L).
3. Временные координаты третьей точки есть (0, Q, L/2), где Q линейно зависит от L с некоторым коэффициентом, который определяется пользователем при построении визуализации.
4. К полученным трём точкам применяются ортогональные преобразования в трёхмерном пространстве, располагающие начало и конец дуги в их исходном положении. Порядок преобразований и используемые при поворотах оси вращения едины для всех обрабатываемых дуг информационного графа алгоритма.

После чего по трём точкам строится результирующая дуга информационного графа алгоритма.

К функциональности визуализации, получаемой с помощью системы AlgoView, помимо собственно интерактивности, предъявляются два требования. Во-первых, интерактивная визуализация должна поддерживать отображение ярусно-параллельной формы информационного графа алгоритма. Во-вторых, из ранее высказанных тезисов, составляющих основу для принципов работы системы, следует: мы можем иметь дело не с самим информационным графом алгоритма, а с некоторым родственным ему графом, полученным путём объединения групп вершин в макрооперации. А значит, в визуализации таких графов эти вершины должны быть явно выделены как не являющиеся непосредственно операциями. Учёт обоих требований производится на этапе разбора исходных данных для построения визуализации. При этом ярусно-параллельная форма графа алгоритма строится автоматически по мере добавления в граф новых вершин и дуг; макровершины же графа необходимо объявлять непосредственно в исходных данных для графа алгоритма. Подробнее эти используемые при построении информационного графа механизмы описаны в разделах 2) и 3).

2. Средство экспорта XML файла в формат хранения 3D-модели

Поскольку исходный код программы, генерирующей XML представление графа алгоритма, недоступен, часть AlgoView, отвечающая за экспорт не может быть выполнена в виде модуля — компонента основного программного средства. Поэтому она представляет собой отдельное десктопное приложение, написанное на C++. Программный код собственно генератора является ОС-независимым, для его графической обёртки используется фреймворк Qt. Перед запуском генератора предлагается указать XML файл с описанием графа алгоритма и папку, в которой будет сгенерирован итоговый набор 3D-моделей в формате JSON. Внешние параметры XML описания запрашиваются у пользователя в процессе работы программы.

Для облегчения разбора XML файла используется библиотека RapidXML [16], содержащая следующие файлы-ресурсы:

- rapidxml.hpp
- rapidxml_iterators.hpp
- rapidxml_print.hpp

- rapidxml_utils.hpp

Для оценивания значений выражений, содержащихся в XML-файлах, используется библиотека exprtk.

- Exprtk.hpp

Исходный код собственно генератора содержится в следующих файлах:

- BaseClasses.h
- BaseClasses.cpp
- ParaParser.h
- ParaParser.cpp
- JsonBuilder.h
- JsonBuilder.cpp
- Export.cpp

Исходный код графической обёртки на Qt расположен в основном в следующих файлах:

- GenController.h
- GenController.cpp
- GraphAppWindow.h
- GraphAppWindow.cpp

В качестве входных данных используются, как уже было упомянуто, имя файла, содержащего описание графа алгоритма в формате XML, а также имя папки, в которую будет записан созданный набор 3D-моделей. Кроме того, можно задать ряд дополнительных параметров, отвечающих за более тонкую настройку параметров модели – размер и удалённость вершин друг от друга, толщина дуг в итоговой модели и тому подобное. При запуске генерации вызывается функция ProduceModels, которая работает в несколько этапов.

На первом этапе создаются и инициализируются объекты классов ParaParser (ответственного за разбор XML файла) и JsonBuilder (ответственного за непосредственное построение набора моделей). Значения параметров визуализации передаются объекту JsonBuilder.

На втором этапе работает метод Parse класса ParaParser. Вначале средствами RapidXML исходный XML файл разбирается (RapidXML позволяет сделать это за время o(strlen())) и представляется в виде DOM-дерева. RapidXML далее позволяет оперировать вершинами этого дерева, что и происходит по следующему алгоритму:

1. В корневой вершине DOM-дерева для любого описания информационного алгоритма в XML-формате имеются атрибуты, которые соответствуют количеству внешних параметров P и количеству групп Q в описании графа алгоритма.
2. В классе ParaParser выделяется память под P переменных - внешних параметров, затем рассматриваются первые P дочерних вершин. Значения соответствующих внешних параметров запрашиваются у пользователя.
3. Оставшиеся дочерние вершины разбираются рекурсивной процедурой. Заведомо известно, что массив оставшихся дочерних вершин состоит из некоторого числа вершин типа Loop, обозначающих нахождение их дочерних вершин внутри цикла, и некоторого числа вершин типа Group, обозначающих блок операций фиксированной структуры. Вершины типа Loop, в свою очередь, могут содержать некоторое число вершин типа Loop и некоторое число вершин типа Group. Суммарное количество вершин типа Group = Q. Изначально процедура запускается с параметром Depth = 0 (это — член класса Para_parser) и перебирает все оставшиеся дочерние вершины. Для каждой вершины типа Loop запускается та же процедура с параметром Depth, увеличенным на 1. Для каждой вершины типа Group запускается нерекурсивная ветвь обработки, о которой далее.
4. Процедура обработки группы принимает в качестве входных параметров вершину типа Group и текущее значение Depth. Извлекаются атрибуты вершины — group_id (номер обрабатываемой группы), num_oper (число различных statements – частей решения в группе) и or_parts (число подгрупп со своими условиями построения элементов графа алгоритма). Затем для каждой дочерней вершины (это вершина типа or_part) запускается процедура её обработки.
5. Процедура обработки вершины типа or_part происходит следующим образом: вначале определяется идентификатор or_part и число and_parts (условий на выбор конкретного подграфа графа алгоритма из множества описанных в этой or_part). По значению Depth определяется число внутренних параметров. Затем строится гнездо циклов (один цикл по каждому внутреннему параметру), в котором первоначальные

значения параметров равны 0. Для каждого набора значений внутренних параметров значения-строки, содержащиеся под тегами CDATA внутри вершин типа and_part, трактуются как выражения-неравенства и проверяются на истинность или ложность. Таким образом, формируется набор значений этих выражений, который преобразуется в номер-идентификатор. Затем для каждого statement в рамках or_part запускается процедура обработки statement с этим идентификатором в качестве параметра.

6. Каждый statement обрабатывается следующим образом: выбирается дочерняя вершина, номер которой соответствует значению идентификатора-параметра. В любом случае это будет вершина типа input. Для этой вершины рассматривается атрибут num_solutions. Если он равен нулю, то ничего не происходит. В противном случае для каждого из solutions запускается соответствующая процедура.
7. В вершине solution наиболее важными атрибутами являются num_altarea и dimsolut. Для первых num_altarea вершин проверяется истинность содержащихся в них (под тегами CDATA) выражений для текущего набора значений внутренних параметров (из пункта 5). Истинность всех этих выражений говорит о наличии зависимости для вершины. Для хранения вершин и ребёр в классе ParaParser имеются 2 члена – m_Edges b m_Vertices. Первый имеет тип std::vector<edge>, где edge – структура, хранящая координаты начальной и конечной вершины, уровень дуги в ярусно-параллельной форме и её принадлежность к множеству дуг ввода-вывода. Второй имеет тип std::vector<Vector3, std::pair<int, int>> и хранит координаты вершины, её уровень в ярусно-параллельной форме и тип (вершина с операцией, вершина ввода-вывода, макровершина). Для каждой новой вершины графа алгоритма данные по ней добавляются в m_Vertices. Последние q координат определяются текущим набором значений внутренних параметров (предполагается, что граф не более чем трехмерный). Также все значения координат смещаются на программно заданный offset в зависимости от номера обрабатываемой группы. В случае наличия зависимости в m_Edges также добавляется элемент. Первый элемент пары определяется как смещение по координатам от текущей вершины в вершине типа dependency (той же по порядку, что и вершина altarea, в которой все выражения истинны). Второй элемент — текущая вершина. Затем, если зависимости не было, то вершине присваивается 1-ый уровень ярусно-параллельной формы. В противном случае запускается функция Adjust_levels, рекурсивно обновляющая значения уровней всех уже созданных ребёр и вершин. Это нужно для

реализации интерактивного отображения ярусно-параллельной формы графа алгоритма.

Таким образом, после выполнения функции Parse в классе ParaParser содержатся 2 структуры типа vector, причём в первой из них хранятся координаты вершин графа алгоритма, а во второй — координаты пар вершин графа алгоритма, которые связаны ребрами (ребро является исходящим для первой вершины в паре). Кроме того, в процессе работы сохраняется общее количество уровней ярусно-параллельной формы в графе алгоритма, пусть оно равно L.

На третьем этапе запускается метод Build класса JsonBuilder. В качестве входных параметров он принимает указатели на структуры типа vector в классе ParaParser. Этот метод работает следующим образом:

1. Класс JsonBuilder имеет приватные методы, обеспечивающие построение примитивных 3D-моделей в формате JSON, а именно: сферы (соответствует вершине), цилиндра (соответствует ребру), конуса (соответствует указанию ориентации ребра). Третий метод запускается с параметрами, позволяющими изменить ориентацию конуса в пространстве. Второй метод, кроме того, может изогнуть цилиндр в определенном направлении.
2. Формируется 4*L вспомогательных файла: VertexPositions_q.txt (координаты вершин 3D-модели), VertexNormals_q.txt (нормали к вершинам 3D-модели, нужно для внедрения освещения), VertexTextureCoords_q.txt (текстурные координаты, нужно для наложения текстур на 3D-модель), Indices_q.txt (используется для связывания вершин в единую модель при отрисовке). Количество файлов каждого типа равно L.
3. Для каждой вершины в m_Vertices строится соответствующий примитив со значениями координат вершины в качестве параметра. Информация о нём распределяется должным образом по 4*L описанным выше файлам.
4. Для каждого ребра в m_Edges строится соответствующая пара примитивов (цилиндр и конус). Его параметрами являются значения координат пары вершин. Цилиндр изгибается в зависимости от нормы вектора, соответствующей расстоянию между парами вершин. Направление изгиба цилиндра определяется в зависимости от положения вершин. Если вершины различаются одной координатой, то цилиндр изгибается по наименьшей из независимых координат. Если

вершины различаются двумя координатами, то цилиндр изгибается по независимой и наибольшей из зависимых координат. В случае различия по всем трём координатам изгиб распределяется по ним в равной степени. Конус поворачивается в пространстве так, чтобы центр его основания, вершина и центр последнего «пояса» цилиндра находились на одной прямой.

5. После полной обработки m_Vertices и m_Edges каждая четверка файлов .txt преобразуется в 1 файл формата JSON. Это означает, что набор 3D-моделей для информационного графа алгоритма готов!

3. Механизм упрощённого разбора XML файлов, содержащих готовые наборы вершин и дуг

Создание XML-представления графа алгоритма путём разбора его референсной реализации на языке C или Fortran выполняется утилитой, не являющейся частью этой работы. Поэтому мы не можем гарантировать наличие корректного XML представления любого интересующего нас алгоритма. Поэтому, а также в целях создания тестовых примеров работы системы, в неё встроен упрощённый алгоритм построения наборов 3D-моделей, для которых заданы непосредственно наборы вершин и дуг. Этот алгоритм аналогичен вышеописанному, за исключением части рекурсивного разбора получаемого DOM-дерева. Вместо этого подразумевается, что в корневой вершине XML-файла <algorithm> заданы общее количество вершин и дуг, после чего последовательно обрабатываются сначала вершины, а затем дуги. При этом, помимо координат, можно сразу указать и их тип, и уровень в ярусно-параллельной форме. Если этого не сделать, то будет использован тип по умолчанию (вершина с операцией), а уровень будет вычисляться рекурсивным пробегом по уже построенным вершинам и дугам.

4. Средство визуализации 3D-модели в web-браузере

По логике работы экспортирующего модуля граф алгоритма представляется в виде единой 3D-модели. Сгенерированные модели помещаются на специальный созданный под эти цели файловый сервер. Кроме того, на этом сервере размещаются немногочисленные библиотеки, необходимые для работы системы web-визуализации. Сама система располагается на web-сервере, причём каждая web-страница предназначена строго для

визуализации конкретного графа алгоритма. Однако каждая web-страница может работать с несколькими 3D-моделями, так как для выделения в графе алгоритма, к примеру, n-го яруса цветом, в случае использования простейших форматов (таких, как JSON) требуется создание отдельной модели.

Система web-визуализации графа алгоритма включает в себя web-страницу с HTML-разметкой и собственно инструкции по визуализации на языке JavaScript. Она включает в себя требуемые для работы библиотеки, файлы-ресурсы (текстуры, 3D-модели со вспомогательной информацией и.т.п.), а также ряд собственных элементов. Используются следующие библиотечные файлы:

- webgl-utils.js
- glMatrix-0.9.5.min.js
- dat.gui.js
- dat.gui.min.js

Используются следующие файлы-ресурсы:

- legend.json
- axis.json
- Различные файлы с текстурами

Собственно элементами системы визуализации являются:

- Main_page.html
- Main.js
- UI_Setup.js
- Textures.js
- Shaders.js (содержит функции инициализации шейдеров, но не их описание)
- Matrices.js
- Axis.js
- Legend..js
- Objt.js
- Mouse.js

Визуализация графа алгоритма в браузере — задача более простая, чем построение 3D-модели графа алгоритма по его XML представлению. Она также выполняется в несколько этапов.

1. При открытии страницы в браузере запускается функция InitGUI, создающая простой графический интерфейс для работы с системой визуализации. Параллельно вызывается функция webGLStart, в которой: создаётся canvas, на который будет отображаться модель графа алгоритма; инициализируется ключевой объект типа GL; инициализируются шейдеры и текстуры; в память загружаются JSON-модели графа алгоритма «по умолчанию»; устанавливается реакция на действия пользователя с canvas (конкретно — на действия при помощи мыши и на клавиши управления); запускается функция tick(), обеспечивающая работу системы визуализации в реальном времени.
2. Функция tick включает в себя 2 действия — отрисовку текущего кадра и формирование параметров для следующего кадра. Каждое из них реализуется отдельной функцией.
3. Функция DrawScene обеспечивает отрисовку текущего кадра. Это действие выполняется в несколько этапов. Вначале проверяются (и при необходимости — например, изменении размера окна браузера) меняются настройки canvas. Затем устанавливается перспектива (PoV сцены); ответственная за это матрица может быть изменена через графический интерфейс, что позволяет отображать проекции ГА на некоторые заранее заданные плоскости (используются декартовы плоскости oXY, oXZ, oYZ). Затем перспектива дополнительно меняется в зависимости от действий пользователя с мышью. Наконец, отрисовываются объекты визуализации в следующем порядке: вначале отрисовываются оси (они являются отдельно загружаемой JSON-моделью), затем легенда для осей (непосредственно в java-скрипте), в последнюю очередь — искомые JSON-модели графа алгоритма.

Для лучшего структурирования системы визуализации методика отображения каждого объекта, появляющегося на canvas, описана в отдельном классе. Для хранения заготовленных JSON-моделей графа алгоритма используется отдельный файловый сервер. Программный движок MediaWiki не поддерживает технологию webgl напрямую. Именно поэтому вся система визуализации располагается на отдельном web-сервере. В качестве инструмента интеграции системы в проект AlgoWiki используется расширение Iframe программного движка MediaWiki https://www.mediawiki.org/wiki/Extension:IframePage. Это расширение позволяет создавать окна, содержащие в себе web-страницу с произвольным URL, непосредственно в тексте wiki-статьи. В качестве недостатка этого решения следует отметить большую нагрузку на сетевое соединение, особенно при наличии нескольких тегов {{#widget:Iframe …}} на одной

странице описания. Однако подобная технология снимает любые возможные ограничения на структуру собственно страницы, содержащей интерактивную визуализацию графа алгоритма, что и послужило причиной такого выбора.

6. Применение описанных методов

Практическая часть включает в себя внедрение разработанной системы и её использование в проекте AlgoWiki, а также построение ряда примеров отображения графов алгоритмов, описываемых в рамках этого проекта.

Прежде всего нас интересовали простые примеры с отображением характерных для информационных графов алгоритмов регулярных структур. К таковым относятся массовые рассылки данных, примеры координатного и скошенного параллелизма [2] в двумерных и трёхмерных областях, гнёзда циклов, а также другие подобные структуры.

На рисунках 1-3 представлены последовательная передача данных и один из самых частых видов параллелизма – скошенный. На рисунках 4-8 представлены трёхмерные структуры с такими же характерными особенностями, а также структуры, содержащие массовые рассылки данных.

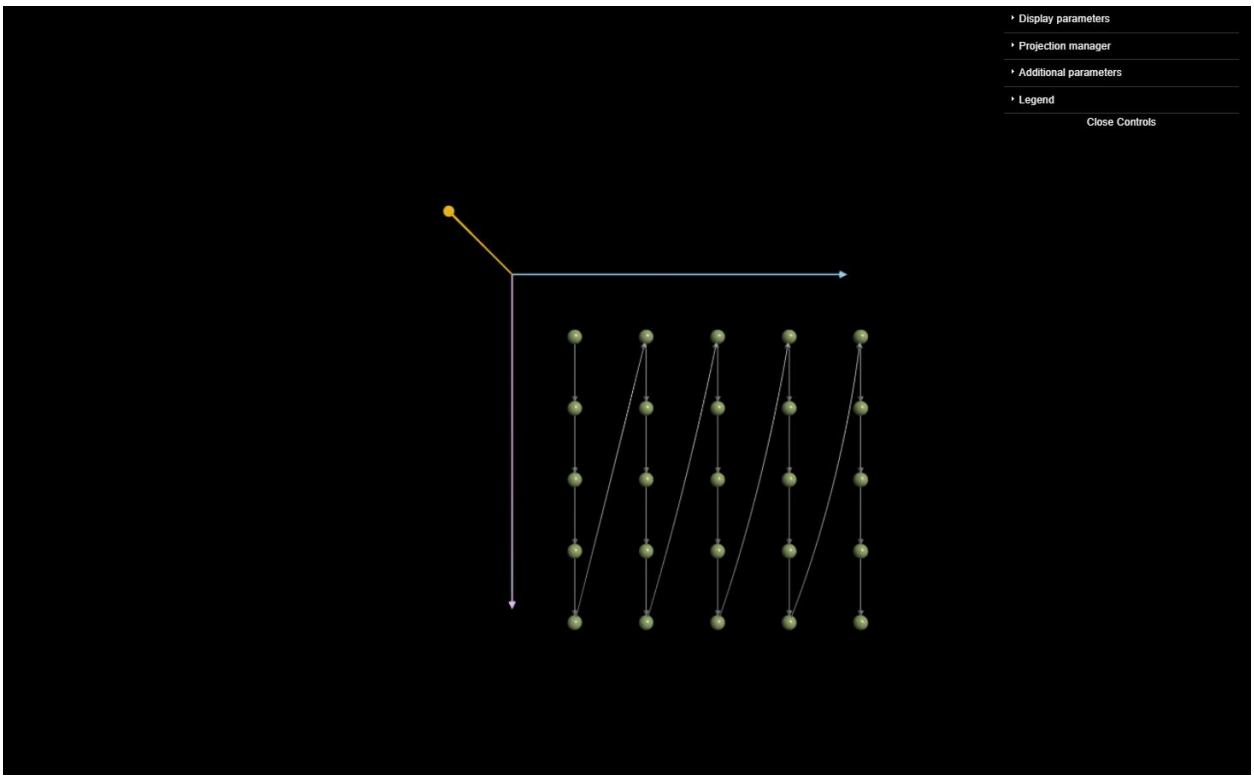

Рис.2 Чисто последовательная структура в 2D представлении

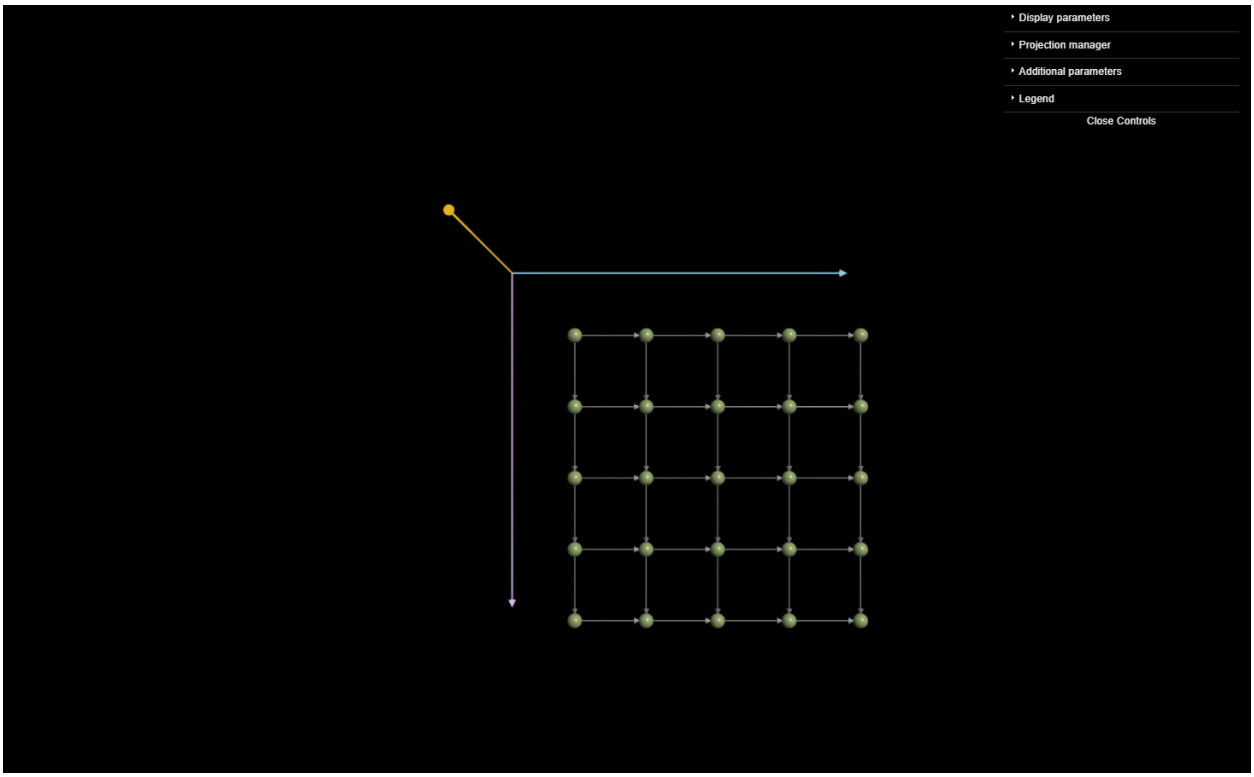

Рис.3 Двумерная структура со скошенным параллелизмом

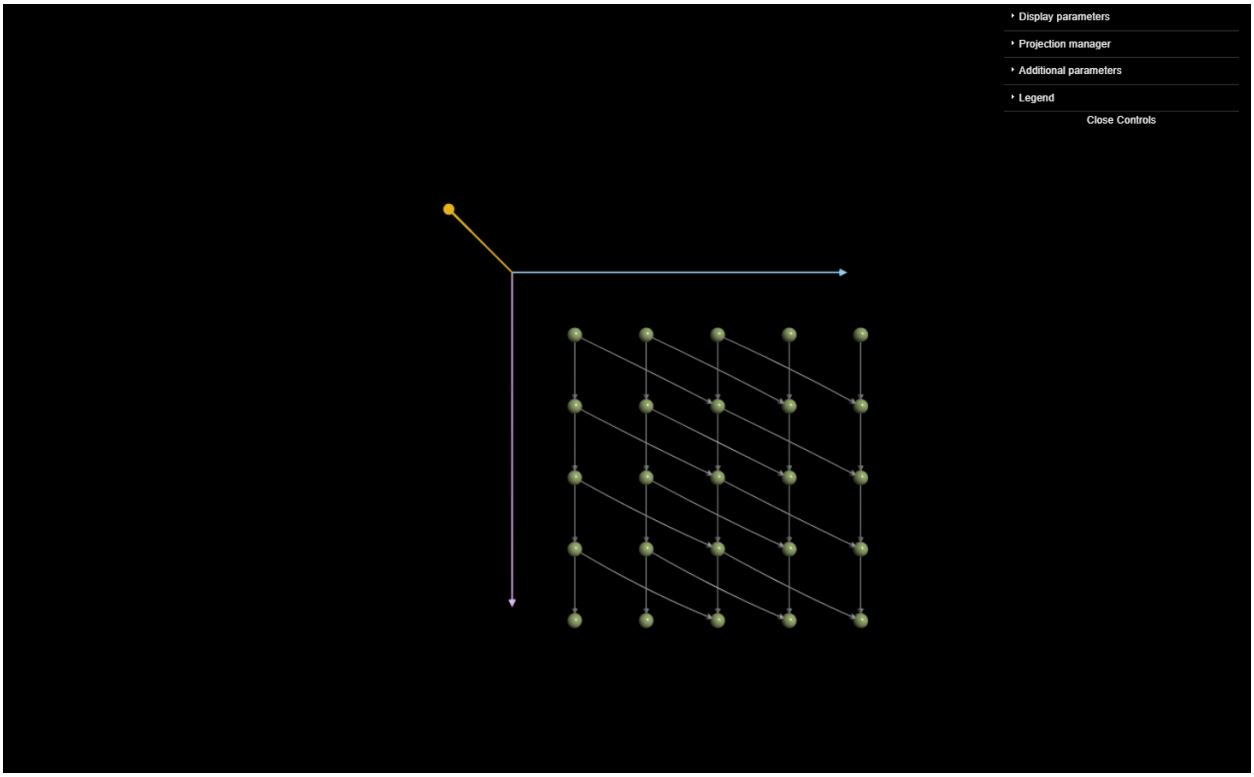

Рис.4 Скошенный параллелизм и скошенная передача данных

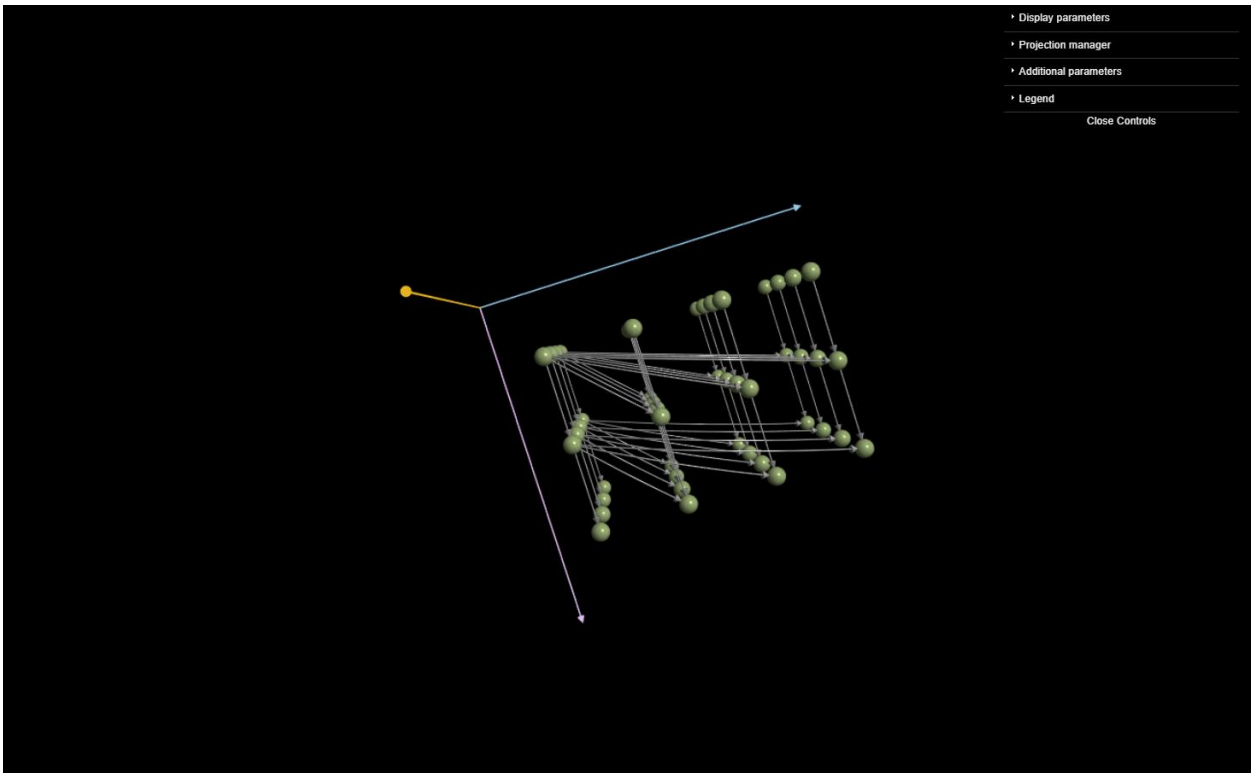

Рис.5 Параллельные массовые рассылки данных по одному измерению

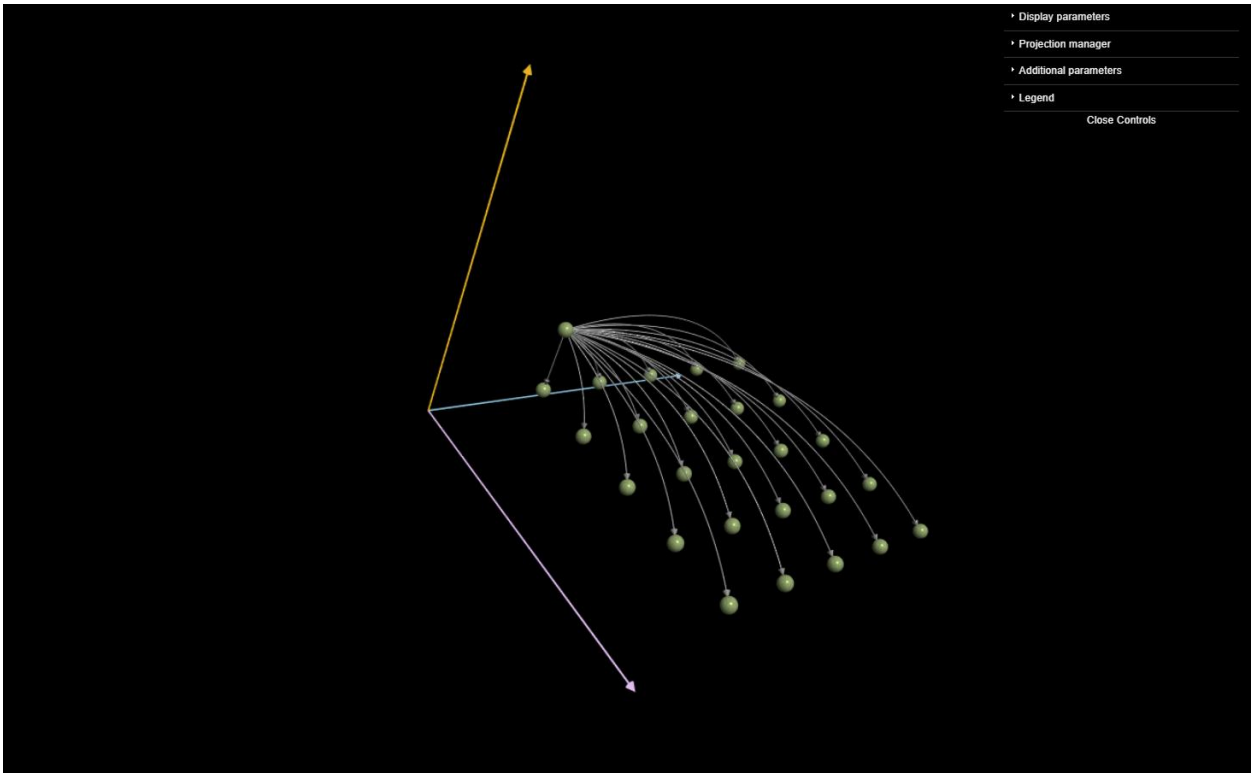

Рис.6 Массовая рассылка данных по двум измерениям

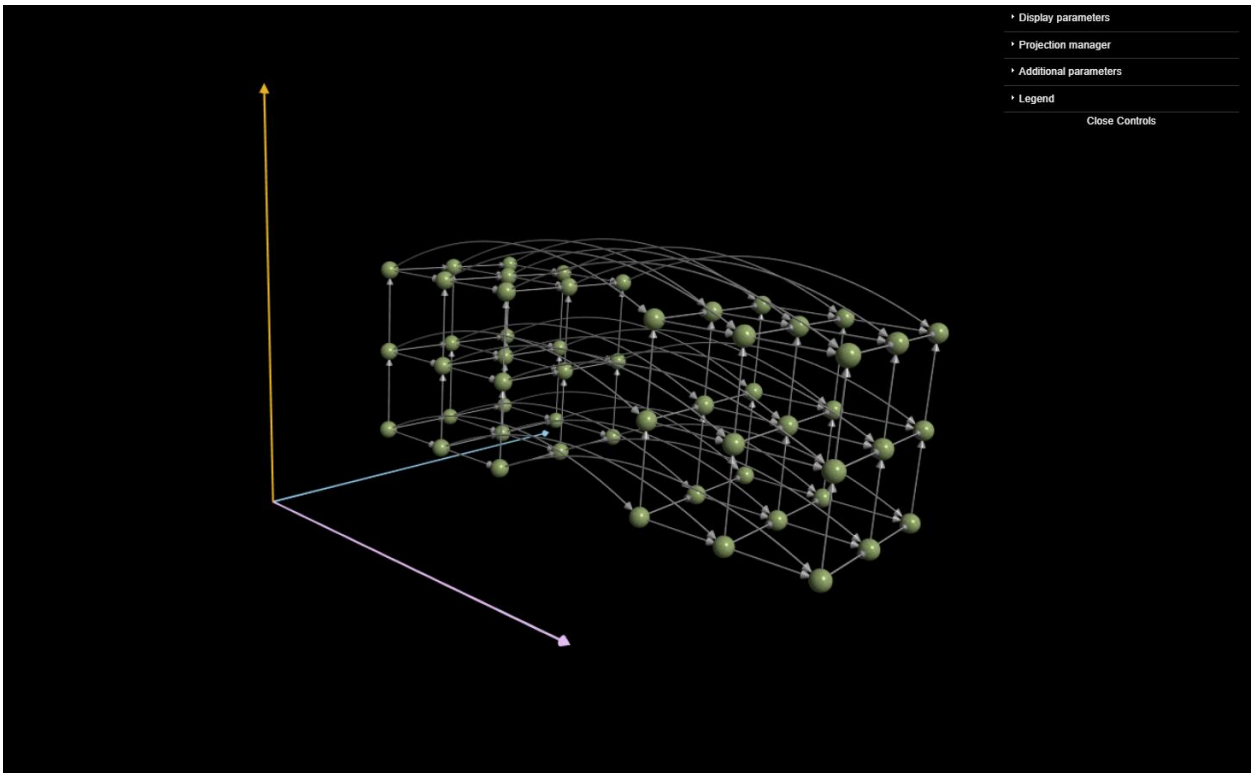

Рис.7 Две последовательно выполняемые трёхмерные структуры

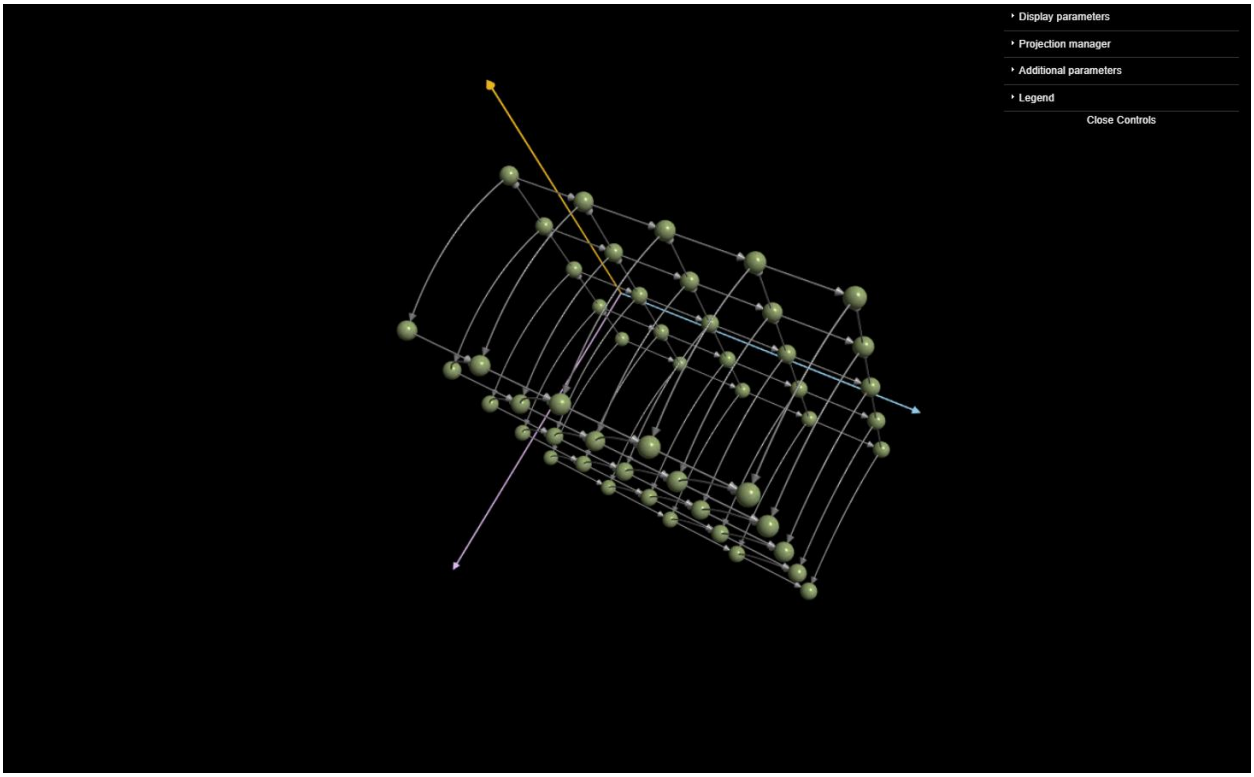

Рис.8 Объединение двух вариантов схем со скошенным параллелизмом

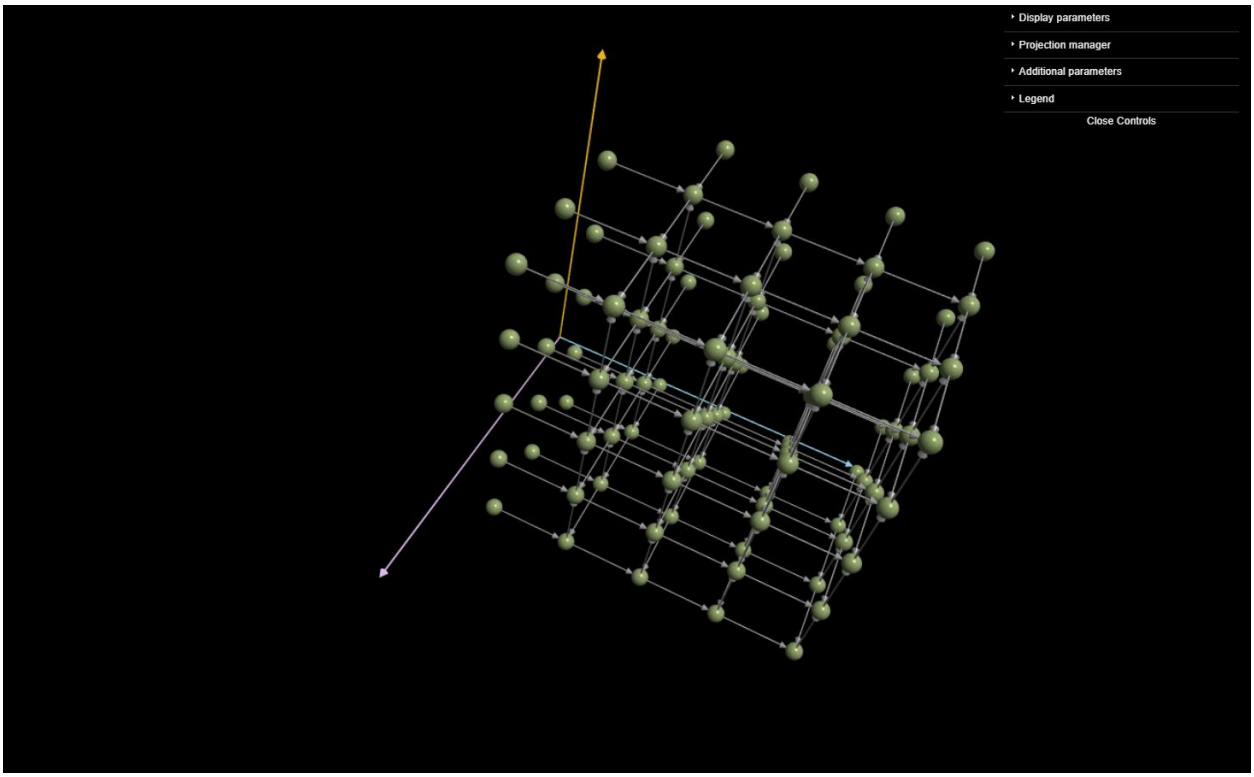

Рис.9 Трёхмерный вариант схемы со скошенным параллелизмом

Исторически сложилось так, что для некоторых алгоритмов визуальное представление их информационного графа приводилось в двух вариантах – соответственно определению и с добавлением специальных вершин, обозначающих входные и выходные данные. В системе AlgoView поддерживается и этот функционал. Одним из примеров такого подхода является визуализация LU-разложения матрицы по методу Холецкого. Далее приведен вариант интерактивной визуализации этого алгоритма для N = 4 в двух режимах – с отображением входных и выходных данных и без него.

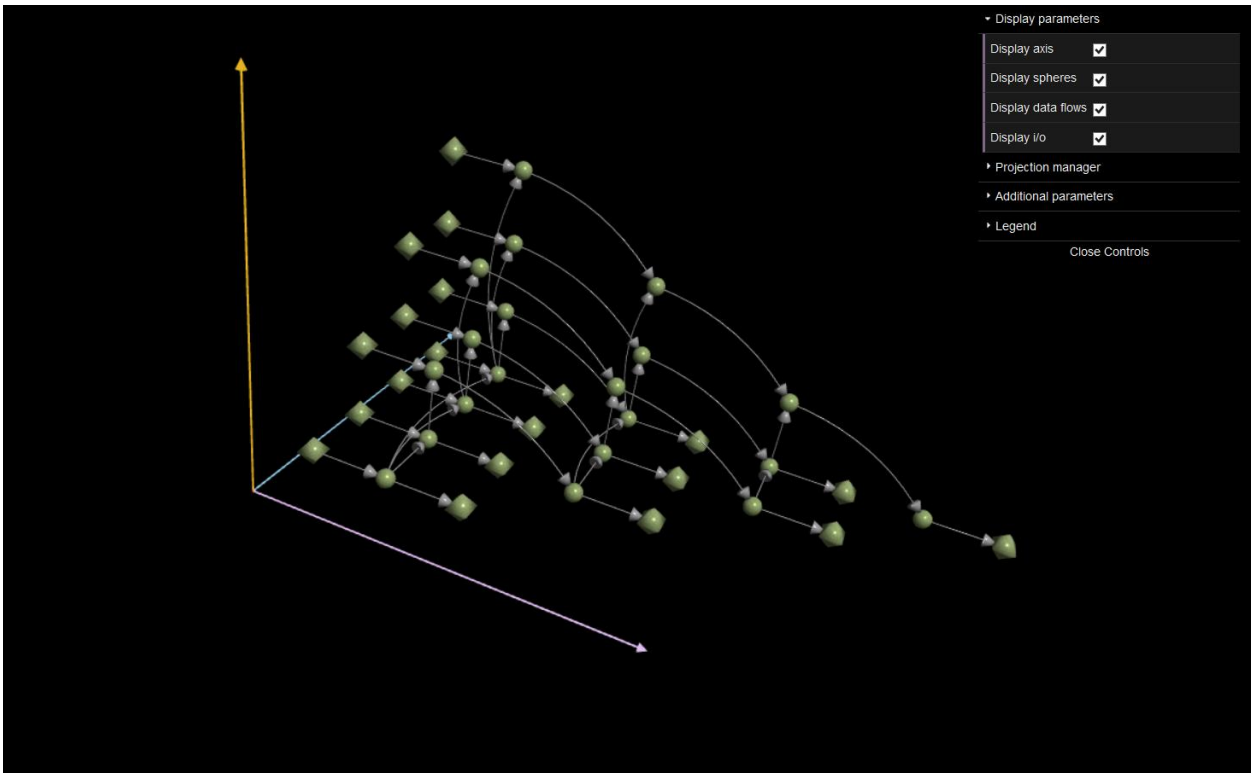

Рис.10 Граф алгоритма с отображением входных и выходных данных

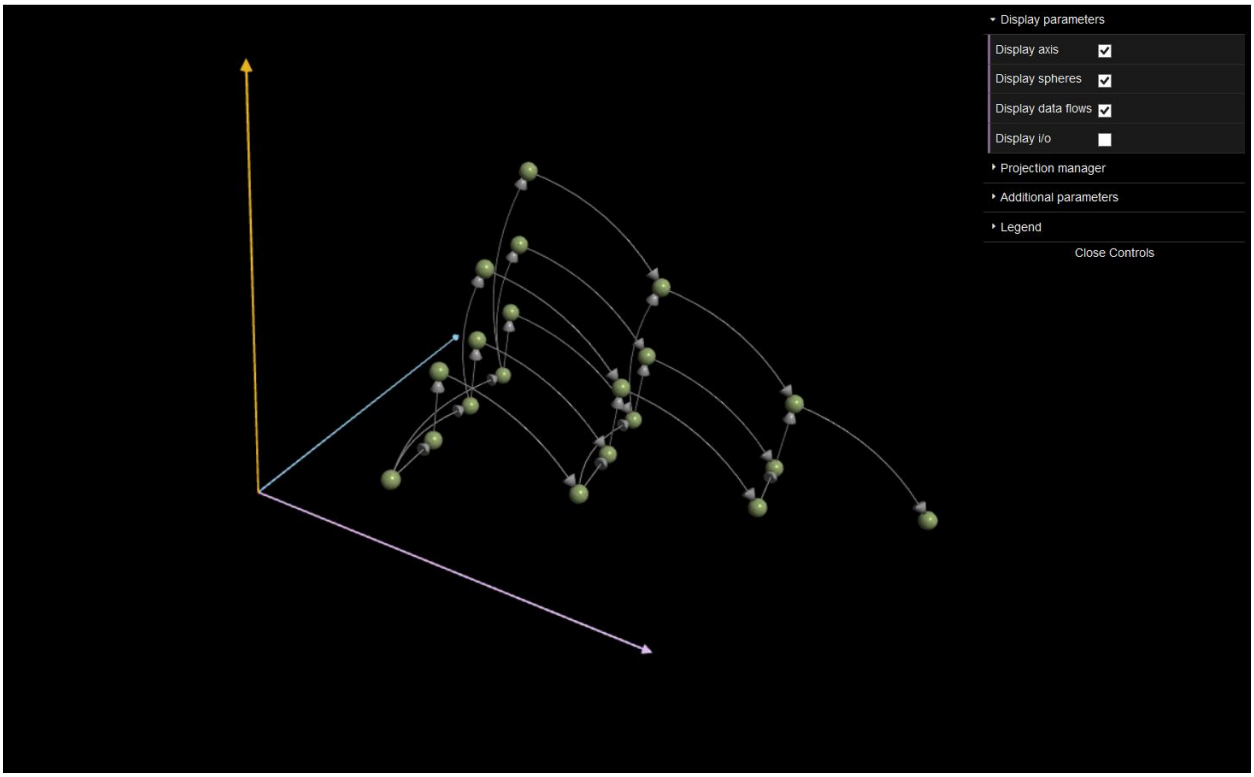

Рис.11 Граф алгоритма без отображения входных и выходных данных

Кроме того, с практической точки зрения было интересно и очень важно представить варианты визуализации реальных алгоритмов, описываемых в рамках проекта AlgoWiki. Поскольку ключевой особенностью AlgoWiki является описание алгоритмов с точки зрения их ресурса параллелизма и реализации на распределённых системах, важна возможность показывать информационный граф алгоритма в его ярусно-параллельной форме. В качестве примера приводится визуализация метода Гивенса.

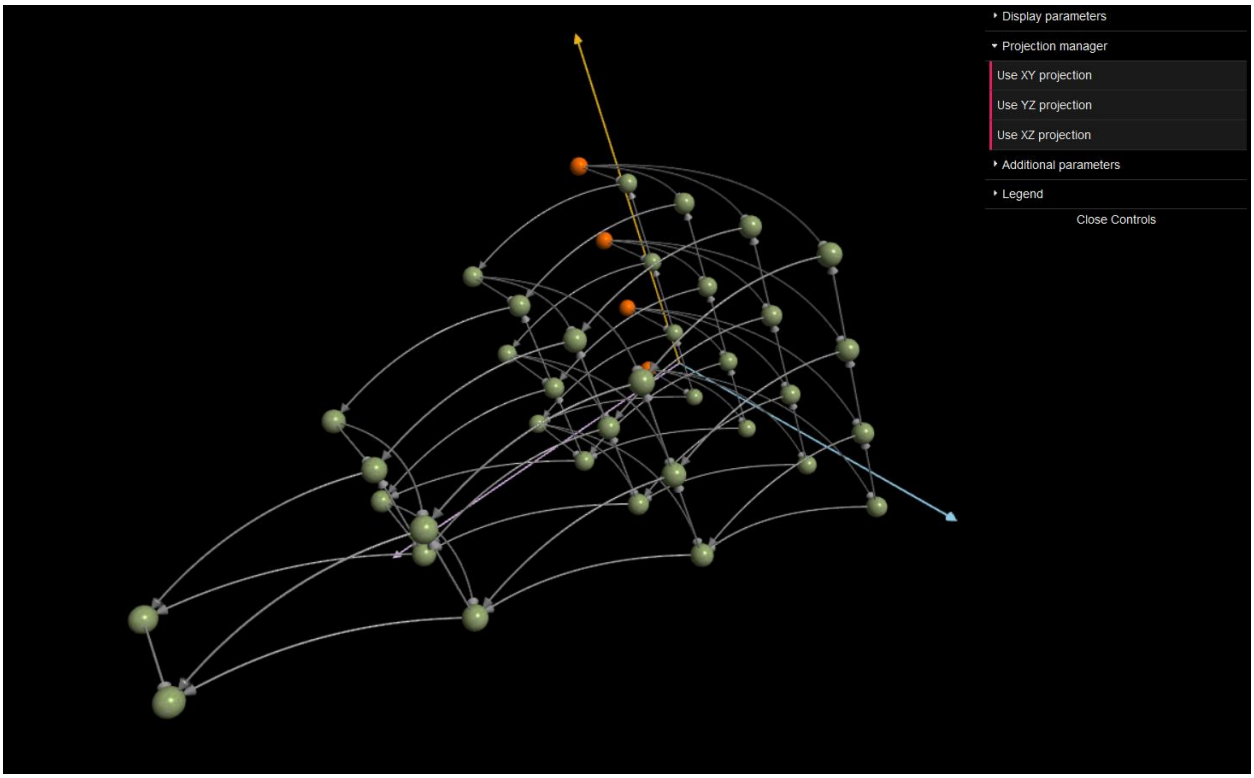

Рис.12 Метод Гивенса - первый шаг ЯПФ для N = 4

Наконец, многие алгоритмы представляются в виде совокупности нескольких более простых. Необходимость отображать такие сложные структуры и привела к понятию макровершин, скрывающих в себе информационные графы отдельных, более тривиальных, алгоритмов. Разумеется, такие вершины хочется видеть и в интерактивной визуализации. На рисунках 15-18 даны примеры визуализации некоторых алгоритмов с такими особенностями. Следует отметить, что используемый генератор XML-представлений не способен работать с макровершинами. Поэтому пришлось внедрить упрощённый режим генерации моделей непосредственно по списку вершин и ребёр, чтобы получить эти результаты.

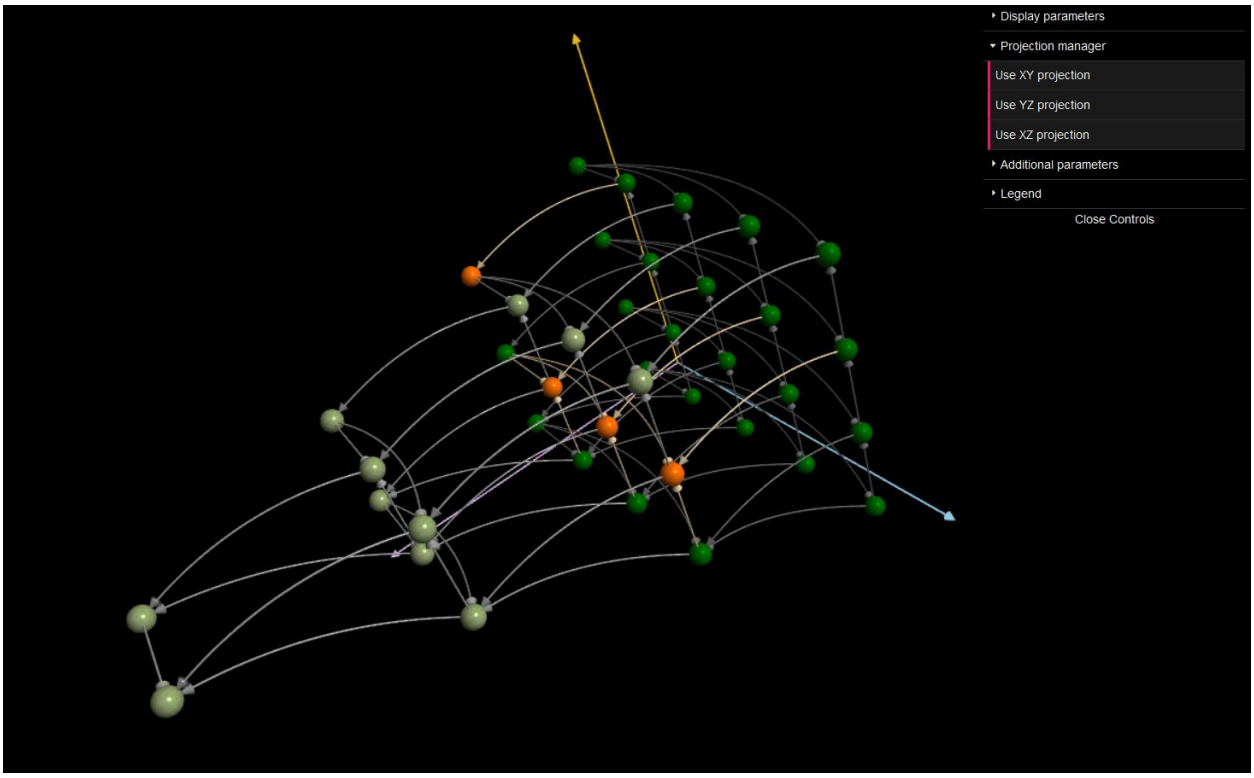

Рис.13 Метод Гивенса - 6 шаг ЯПФ для N = 4

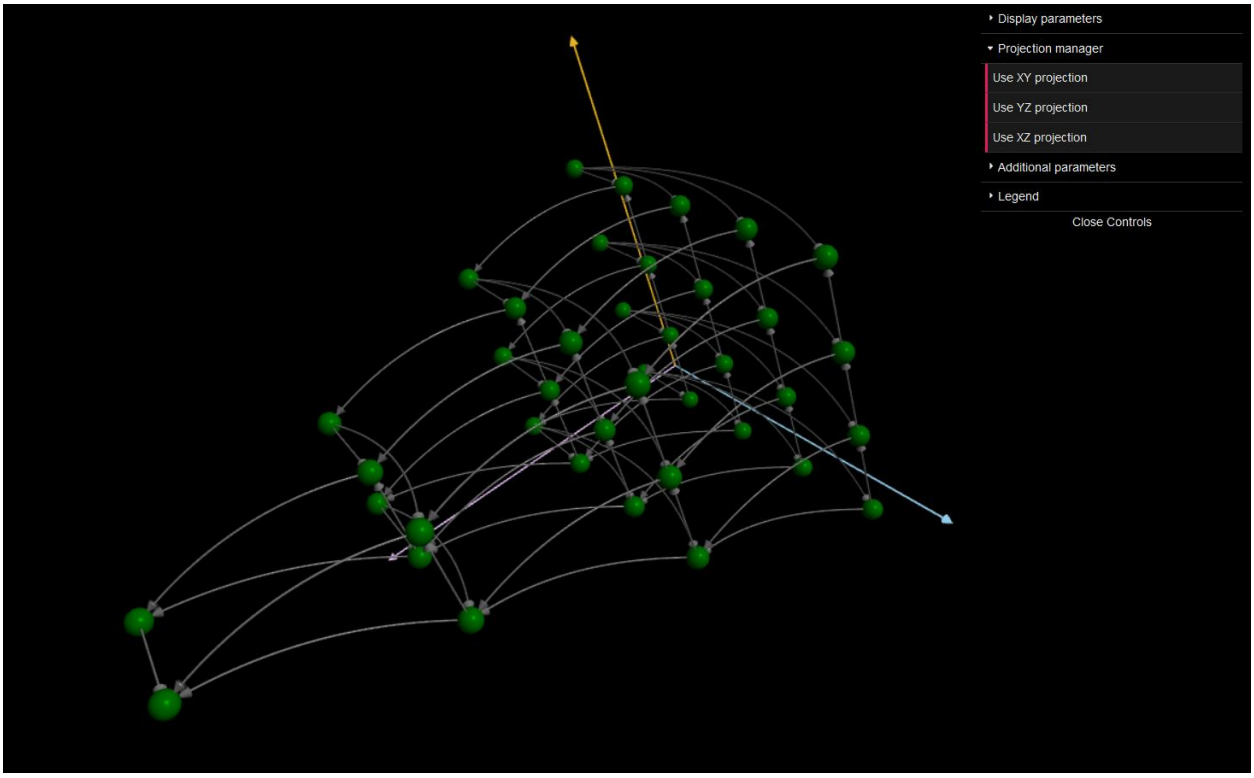

Рис.14 Метод Гивенса – 12-й шаг ЯПФ для N = 4

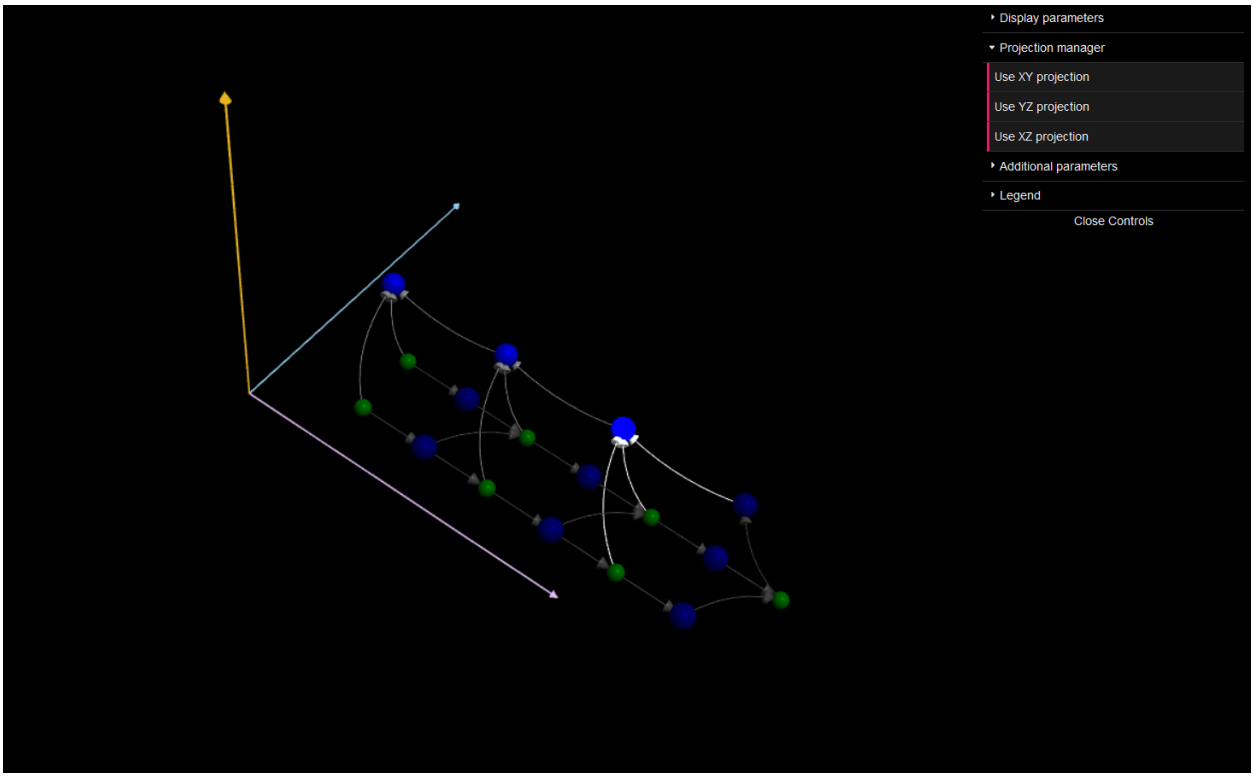

Рис.15 Информационный граф алгоритма прогонки для N = 4

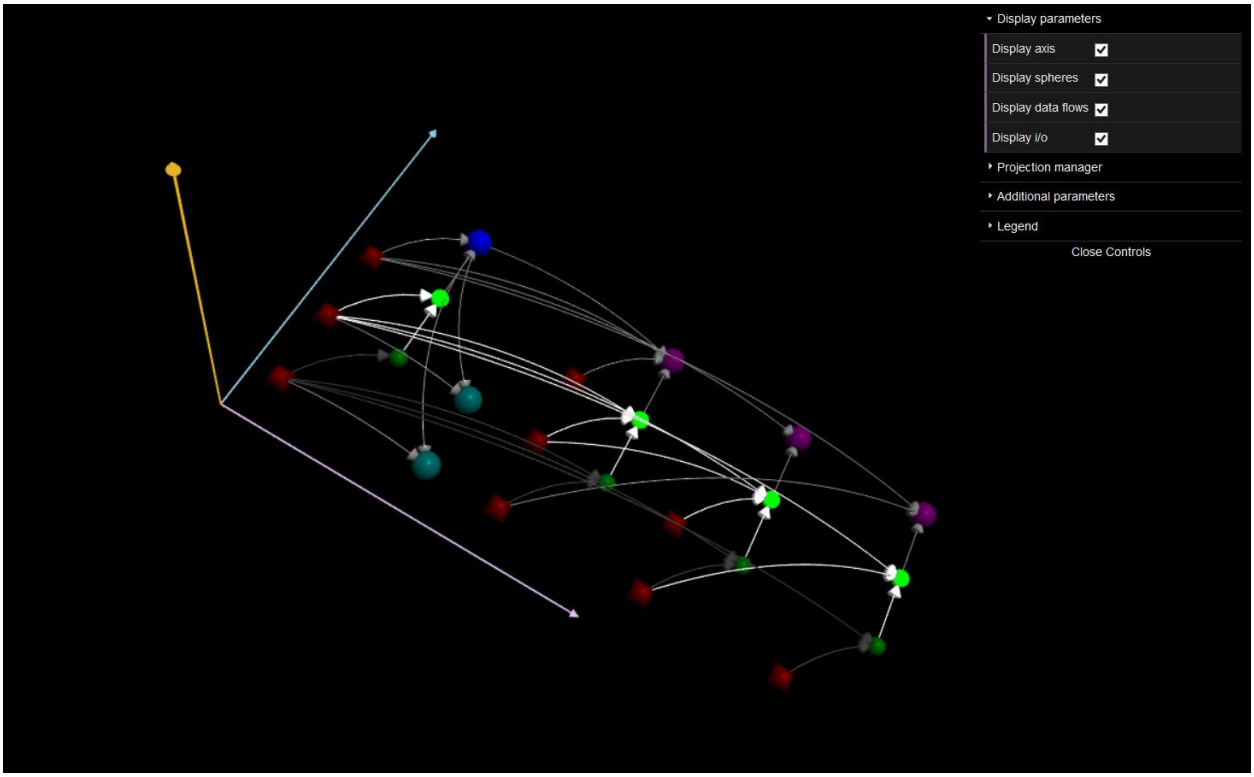

Рис.16 Граф первой половины шага алгоритма Хаусхолдера для N = 3

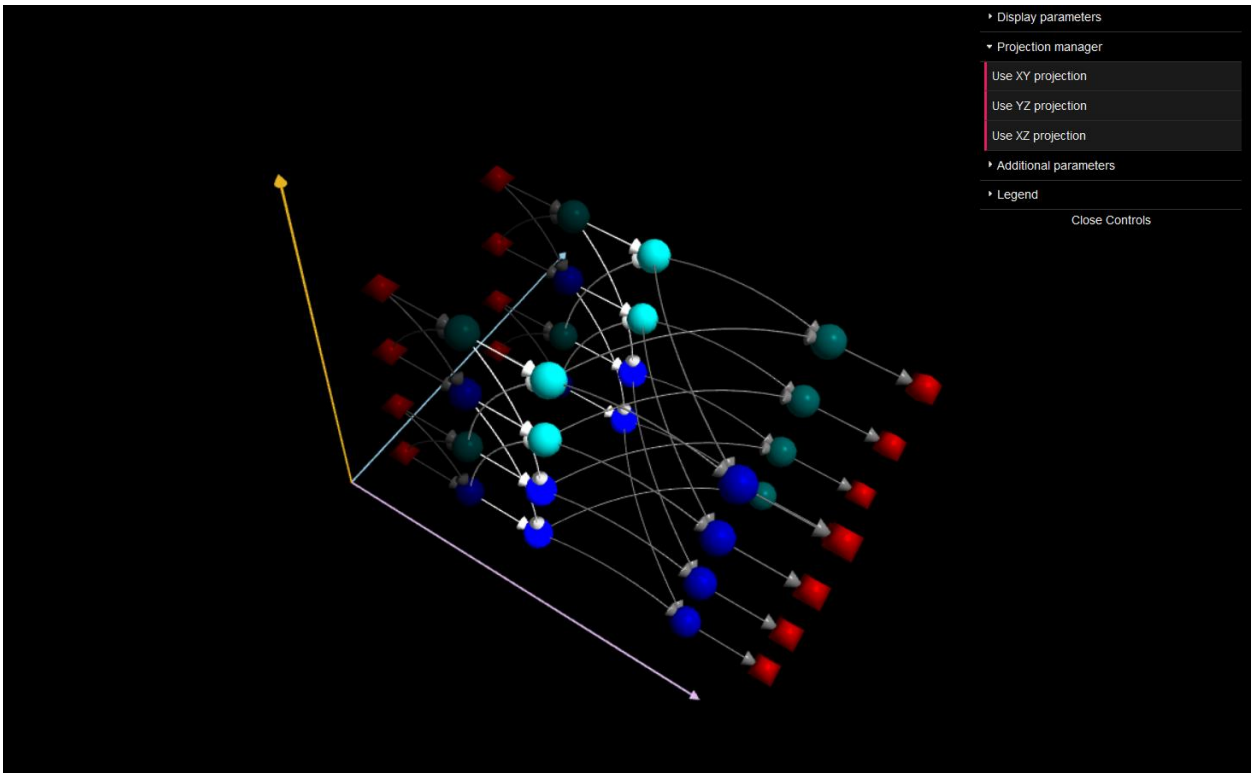

Рис.17 Простой алгоритм Кули-Тьюки для N = 8

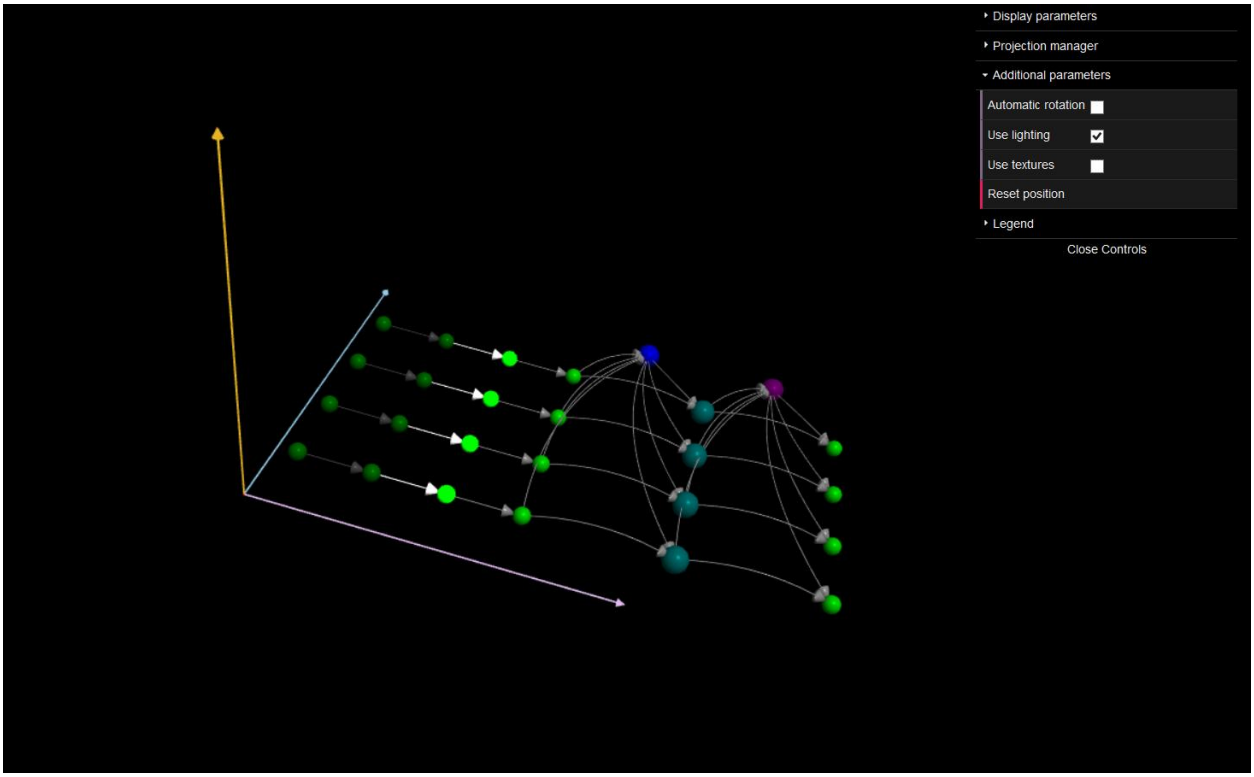

Рис.18 Одна итерация алгоритма Ланцоша (без переортогонализации)

Каждое из продемонстрированных выше изображений – не что иное, как часть снимка экрана монитора с открытым окном браузера. Однако простой доступности интерактивной визуализации информационного графа в режиме online нам мало – нужно внедрить эти визуализации в страницы описания алгоритмов проекта AlgoWiki, который, как уже говорилось ранее, работает на движке MediaWiki. Технически эта возможность реализована при помощи механизма расширений для MediaWiki, а конкретно – расширения IFrame. Оно позволяет встраивать в wiki-страницы «окна» других сайтов в сети по их URL-адресам. Все web-странички с интерактивными визуализациями информационных графов хранятся на том же сервере, что и традиционные используемые в AlgoWiki файлы изображений, такие как формулы, графики локальности данных и масштабируемости алгоритмов.

7. Результаты и заключение

По итогам проведённых исследований, разработки требуемого для достижений поставленных целей программного обеспечения и представления полученных материалов обществу удалось добиться следующих результатов:

1) Сформулированы принципы построения интерактивных визуализаций информационных графов алгоритмов и требования к ним.
2) Создано программное обеспечение для автоматического построения удовлетворяющих этим требованиям визуализаций, их отображения в режиме online.
3) Созданные визуализации включены в описания алгоритмов в рамках проекта AlgoWiki.
4) По соответствующей тематике опубликована статья, с которой можно ознакомиться по ссылке [5].

Кроме того, есть планы на будущее по данной работе. К ним относятся, прежде всего, механизм удалённой генерации нового набора 3D-моделей для заданного алгоритма по команде из браузера и обновление функционала для инструмента генерации XML файлов. Последний хоть и не относится к нашей работе напрямую, но его возможности на данный момент уже отстают от возможностей нашей системы визуализации, из-за чего и пришлось создать инструмент для упрощённого построения 3D-моделей по списку вершин и ребёр. Иначе бы нам просто не удалось продемонстрировать все возможности системы за отсутствием исходных данных.

8. Список литературы

1) Проект AlgoWiki https://algowiki-project.org/
2) Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. - 608 с.
3) V.Voevodin, A.Antonov, J.Dongarra. AlgoWiki: an Open Encyclopedia of Parallel Algorithmic Features // Supercomputing Frontiers and Innovations, Vol.2, No.1 (2015). Pp.4-18.
4) Alexander Antonov, Vadim Voevodin, Vladimir Voevodin, Alexey Teplov. A Study of the Dynamic Characteristics of Software Implementation as an Essential Part for a Universal Description of Algorithm Properties // 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Proceedings, 17th-19th February 2016. Pp.359-363.
5) Alexander S. Antonov, Nikita I. Volkov. An AlgoView Web-visualization System for the AlgoWiki Project // Communications in Computer and Information Science. Vol. 753. 2017. Pp. 3-13. DOI: 10.1007/978-3-319-67035-5_1
6) WebGL 2.0: https://www.khronos.org/registry/webgl/specs/latest/2.0/
7) Стандарт визуализации информационного графа алгоритма https://algowiki-project.org/ruСтандарт_визуализации_ГА
8) George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Pierre Lemarinier, Jack Dongarra. DAGuE: A generic distributed DAG engine for High Performance Computing. Parallel Computing 38 (2012) 37-51
9) Yifan Hu1 and Lei Shi. Visualizing large graphs. Yahoo Labs, 111 W 40th St, New York, NY 10018, USA; SKLCS, Institute of Software, Chinese Academy of Sciences, China
10) Stephen P Borgatti. NetDraw: Graph Visualization Software. University of Kentucky , January 2002
11) Benjamin Finkel Sc. B. Curvilinear Graph Drawing Using the Force-Directed Method B., Brown University, 2003
12) Interactive molecular view: https://chemagic.org/molecules/mini.html
13) Interactive molecular view: http://molview.org/
14) Weather maps: https://habrahabr.ru/company/yandex/blog/343518/
15) WebGL: https://hacks.mozilla.org/2013/04/the-concepts-of-webgl/
16) RapidXML: http://rapidxml.sourceforge.net/manual.html