МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА

Факультет информатики и систем управления

Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №4

по курсу «Моделирование»

«Построение разрезов поверхностей»

Выполнил:

студент ИУ9-111

Выборнов А. И.

Руководитель:

Домрачева А. Б.

Москва 2015

1. Постановка задачи

Одной из базовых задач анализа триангуляционных поверхностей является построение разрезов — вертикальных (профилей) и горизонтальных (изолиний).

Изолиниями уровня h называют геометрическое место точек на поверхности, имеющих высоту h и имеющих в любой своей окрестности другие точки с меньшей высотой: $I_h = \{(x,y)|z(x,y) = h, \forall\epsilon > 0 \colon \exists(x',y') \colon |(x',y'),(x,y)| < \epsilon, z(x',y') < h\}$.

Изоконтурами между уровнями h_1 и h_2 называют замыкание геометрического места точек на поверхности, имеющих высоту $h \in [h_1, h_2)$, т.е. множество точек $I_h = \overline{\{(x,y)|h_1 <= z(x,y) < h_2\}}$.

В задаче построения изоконтуров требуется построить множество непересекающихся регионов, каждый из которых представляет область, высоты точек внутри которой лежат в определенном диапазоне. Обычно задаётся система диапазонов с помощью начального значения самого первого диапазона, конечного значения последнего диапазона и шага построения диапазонов.

По трёхмерной модели поверхности нужно построить множество изоконтуров, лежащих в диапазоне $[h_1, h_2)$ с шагом Δh.

2. Теоретическая часть

2.1. Построение изолиний

Для построения изолиний высотой h используется следующий алгоритм:

1. Помечаем каждый треугольник триангуляции, по которому проходят изолинии (т.е. выполняется условие $min(z_1, z_2, z_3) < h << max(z_1, z_2, z_3)$, где z_i — высоты трех его вершин), флагом $C_i := 1$, а все остальные треугольники – $C_i := 0$. Если обнаружен хотя бы один треугольник, у которого хотя бы одно ребро лежит в плоскости изолинии, то h уменьшается на некоторое малое Δh и алгоритм повторяется заново.

2. Для каждого треугольника с $C_i = 1$ выполняем отслеживание очередной изолинии в обе стороны от данного треугольника, пока один конец не выйдет на другой или на границу триангуляции. Каждый пройденный при отслеживании треугольник помечается $C_i := 0$. Конец алгоритма.

Трудоемкость такого алгоритма, очевидно, является линейной относительно размера триангуляции.

2.2. Построение изоконтуров

Для построения множества изоконтуров, лежащих в диапазоне $[h_1, h_M)$ с шагом Δh используется следующий алгоритм:

1. Пусть заданы уровни $h_1, ..., h_M$, Обнуляем множества ломаных, входящих в изоконтуры: $C_i = \varnothing, i = \overline{0, M}$.

2. Для каждого уровня h_i строим изолинии. Каждую замкнутую изолинию добавляем во множество C_i .

3. Определяем все кусочки границы триангуляции между точками выхода изолиний на границу. Формируем граф, в котором в качестве узлов выступают точки выхода на границу, а в качестве рёбер — кусочки границы между этими точками и рассчитанные изолинии. Каждая изолиния должна войти в граф дважды в виде одинаковых ориентированных рёбер, но направленных в разные стороны. Для рёбер —кусочков границы — устанавливаем такую ориентацию, чтобы внутренности триангуляции находились справа по ходу движения. В результате в каждом узле графа должны сходиться четыре ребра.

4. По полученному графу строим контуры. Начинаем движение с любой вершины графа и двигаемся вперед в соответствии с ориентацией рёбер до тех пор, пока не вернемся в начальную вершину. Повторный проход по одному и тому же ребру запрещен, для чего делаются специальные пометки на рёбрах. При попадании в узел графа из граничной цепочки далее надо двигаться по ребру, соответствующему изолинии, иначе — по граничному ребру. Обратим внимание, что каждая изолиния войдет в два контура, соответствующих разным диапазонам высот.

5. Для каждого полученного на предыдущем шаге контура определяем, какому диапазону высот он соответствует. Для этого нужно взять и проверить любое ребро триангуляции, входящее в составе граничной цепочки, использованной в каждом контуре. На основании этого помещаем цепочку в соответствующее множество C_i. Конец алгоритма.

Трудоемкость данного алгоритма линейно зависит от размера триангуляции и количества изолиний.

3. Реализация

В рамках лабораторной работы была написана программа на языке python, её разработка производилась в несколько этапов:

1. Получение и хранение триангулированной модели объекта.
2. Считывание модели в оперативную память.
3. Построение изолинии.
4. Построение изоконтуров.
5. Визуализация результатов.

3.1. Получение и хранение триангулированной модели объекта

Для построения трёхмерного объекта и его триангулированной модели был использован пакет *Blender* — свободный, профессиональный пакет для создания трёхмерной компьютерной графики, включающий в себя средства моделирования, анимации, рендеринга, постобработки и монтажа видео со звуком, а также для создания интерактивных игр. На рисунке 1 показан скриншот работы по созданию модели с помощью blender.

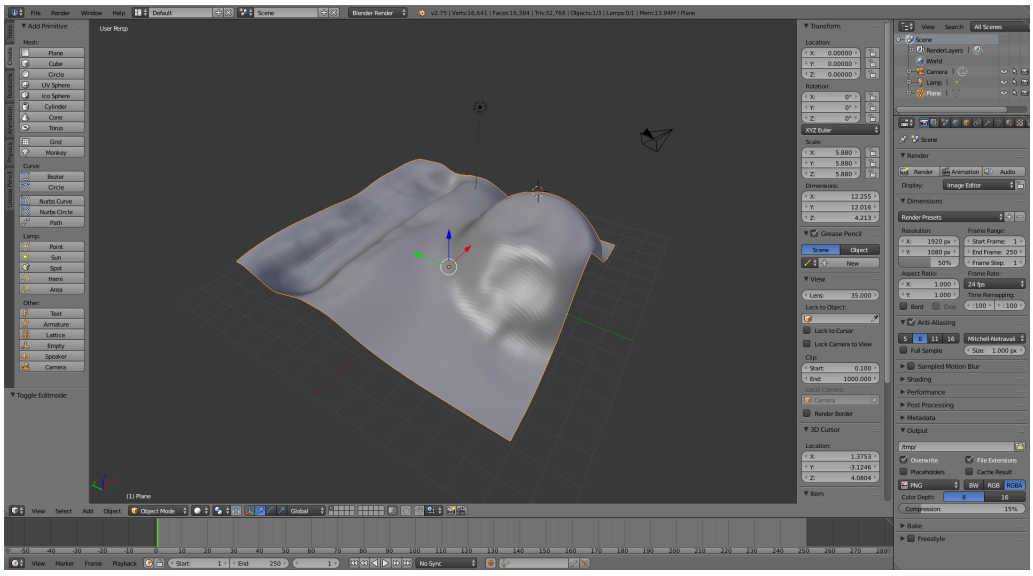

Рисунок 1 — Скриншот работы по созданию модели в blender.

OBJ — это простой формат данных, который содержит только геометрию, а именно, позицию каждой вершины, связь координат текстуры с вершиной, нормаль

для каждой вершины, а также параметры, которые создают полигоны. Данный формат был выбран из-за простоты, человекочитаемости человеком, а также поддержки этого формата всеми основными графическими редакторами.

3.2. Считывание модели в оперативную память

Программа получает на вход файл в OBJ формате. Затем она считывает из него информацию о координатах каждой вершины и о полигонах (в рамках этой работы все полигоны треугольные), которые связывают эти вершины.

В программе эта информация описывается с помощью трёх классов: Triangle (соответствует треугольному полигону), Edge (соответствует ребру) и Vertex (соответствует вершине).

- Triangle — класс, описывающий треугольный полигон. Этот класс содержит информацию о трёх рёбрах, его составляющих.

- Edge — класс, описывающий отрезок, представляющий собой грань полигона. Содержит информацию о вершинах его составляющих, а также о треугольниках, ребром которых он является. Для каждого ребра существует только один экземпляр класса Edge, это обеспечивается с помощью образца проектирования Фабрика (Factory), который возвращает новое ребро, если оно ещё не было создано, иначе возвращает уже существующий экземпляр класса.

- Vertex — класс, описывающий вершину. Хранит информацию о координатах этой вершины в трёхмерном пространстве и о рёбрах, в которые эта вершина входит.

3.3. Построение изолинии

Построение изолинии выполнялось согласно алгоритму описанному в теоритической части. В итоге по модели мы получаем список изолиний, с отметками являются ли они замкнутыми. Пример получающихся изолинию приведён на рисунке 2.

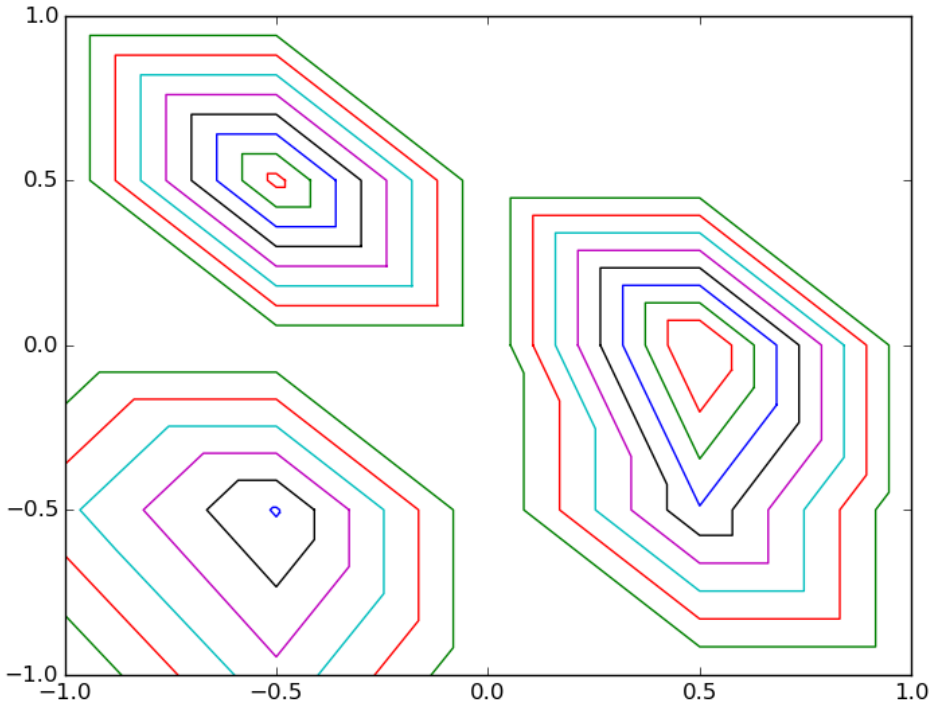

Рисунок 2 — Изолинии.

3.4. Построение изоконтурова

Для замкнутых изолиний получение изоконтура тривиально — помещаем изолинию в множество, соответствующее её высоте, а для незамкнутых построение происходит в два этапа:

1. Формирование графа, содержащего точки выхода на границу изолиний и вершины всех граничных рёбер.
2. Построение по графу контуров.

При реализации строился граф, немного отличный от описанного в теоретической части. По причине того, что описанный в теоретической части граф достаточно

сложно получить. Опишем используемый граф, с помощью алгоритма его построения:

1. Добавим в граф в качестве вершин точки выхода изолиний на границу. Каждую изолиню добавим в виде двух рёбер направленных в разные стороны, которые соединяют соответствующие изолинии вершины. Во время добавления вершин, запомним для каждого граничного ребра, какие изолинии их пересекают.

2. Для каждого граничного ребра модели выполним следующий алгоритм:

   а) Если ребро не пересекается изолиниями, то добавим в граф в качестве вершин, его узлы и соединим их ребром графа, направленным так, чтобы внутренность треугольника осталась справа (проверка выполняется с помощью векторного произведения).

   б) Иначе разобьём ребра на множество последовательных не пересекающихся отрезков, вершинами которых являются узлы ребра и точки пересечения изолиний с этим ребром. Каждый отрезок добавим в граф, аналогично ребру, не пересекаемому изолиниями.

Граф в программе представлен в виде трёх классов:

- Graph — класс, хранящий граф в формате отображения вершины графа в множество пар (ребро, вершина). Каждый элемент множества соответсвует исходящему из исходной вершины рёбру и вершине, куда это ребро ведёт.

- GraphEdge — класс, описывающий ребро графа. Содержит информацию о изолинии, если она в нём содержится.

- GraphNode — класс, описывающий вершину графа. Хранит в себе экземпляр класса Vertex. Для каждой вершины существует только один экземпляр класса GraphNode, это обеспечивается с помощью образца проектирования Фабрика.

Пример получаемоего графа приведён на рисунке 3. Вершины обозначены овалами с координатами, рёбра обозначены стрелками с подписями: border — ребра соответствующее участку границы, isoline — ребро соответствующее изолинии. Все вершины графа расположены в пространстве в соответствии с их координатами на двумерной плоскоти $x0y$.

Для получения по графу изоконтура, выполняем обход графа, который начинается с любой вершины. Обход заключается в проходе по рёбрам графа согласно их направлениям, до тех пор пока не вернёмся в исходную вершину. При этом если

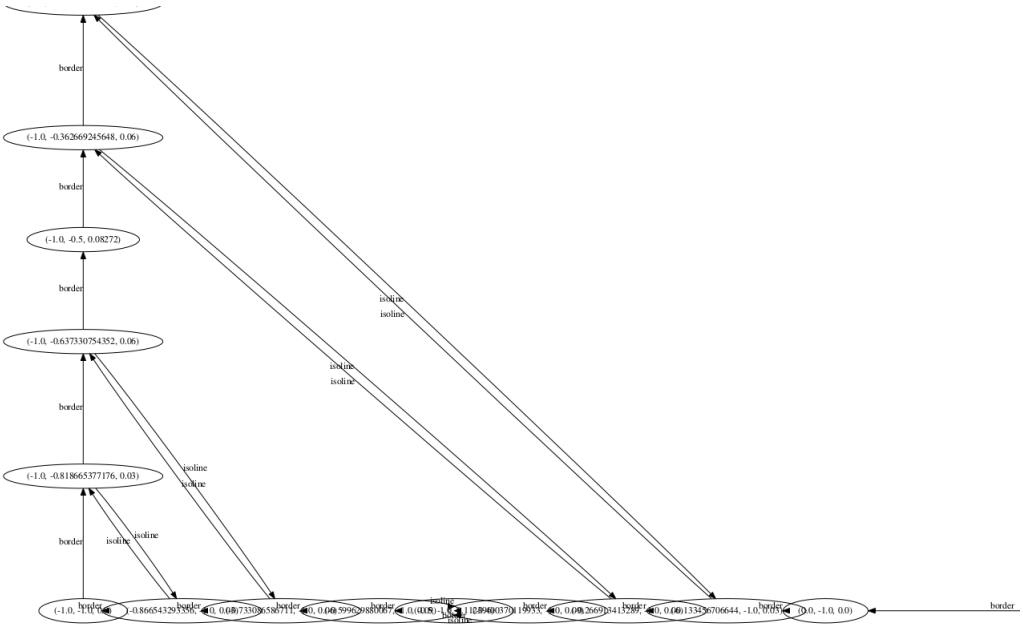

Рисунок 3 — Граф, используемый при построении изоконтуров.

мы прошли по изолинии, то в следующий раз мы обязательно должны пройти по ребру модели. А если мы прошли по ребру модели и перед нами стоит выбор идти далее по изолии или ребру - всегда выбираем изолинию. В результате получаем изоконтур, которые соотносим с высотой, соответствующей наиболее низкой изолинии, входящей в этот контур. Пример контура, который мы получаем после обхода можно увидеть на рисунке 4.

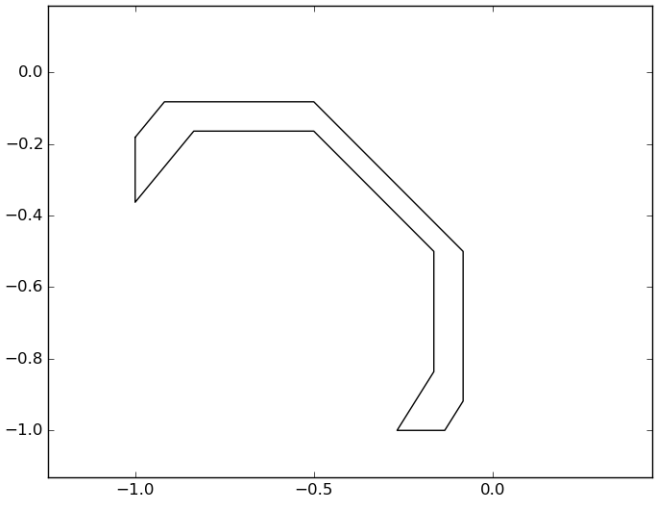

Рисунок 4 — Изоконтур, который получился из незамкнутой изолинии.

3.5. Визуализация результатов

Визуализация изолиний и изоконтуров выполняется с помощью matplotlib — библиотека на языке программирования Python для визуализации данных. Визуализация графа выполняется с помощью порождения файла на языке DOT, с последующей визуализацией с помощью graphviz — пакета утилит по автоматической визуализации графов, заданных в виде описания на языке DOT.

Результатом работы программы является изображение изоконтуров. Изображение получается с помощью последовательного рисования раскрашенных фигур, соответствующих изоконтурам. Изоконтуры рисуются последовательно (начиная с самого нижнего, заканчивая самым верхним), друг поверх друга. При этом с увеличением высоты пропорционально изменяется цвет заливки от светло-серого то тёмно-серого. Пример изображения, получаемого в результате работы программы, показан на рисунке 5.

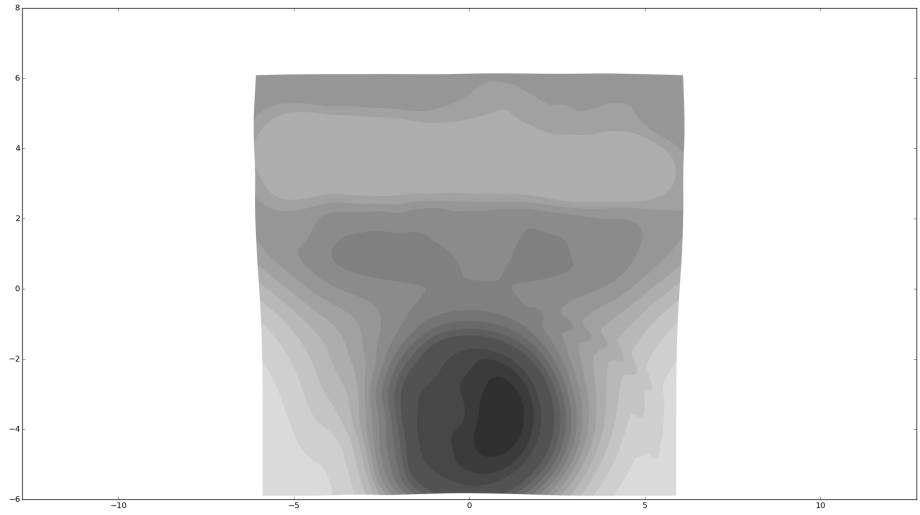

Рисунок 5 — Визуализация результата работы программы.

4. Тестирование

Для тестирования были использованы две модели: модель 1, состоящая из 32 полигонов, изображена на рисунке 6, и модель 2, состоящая из 32768 полигонов, изображена на рисунке 7.

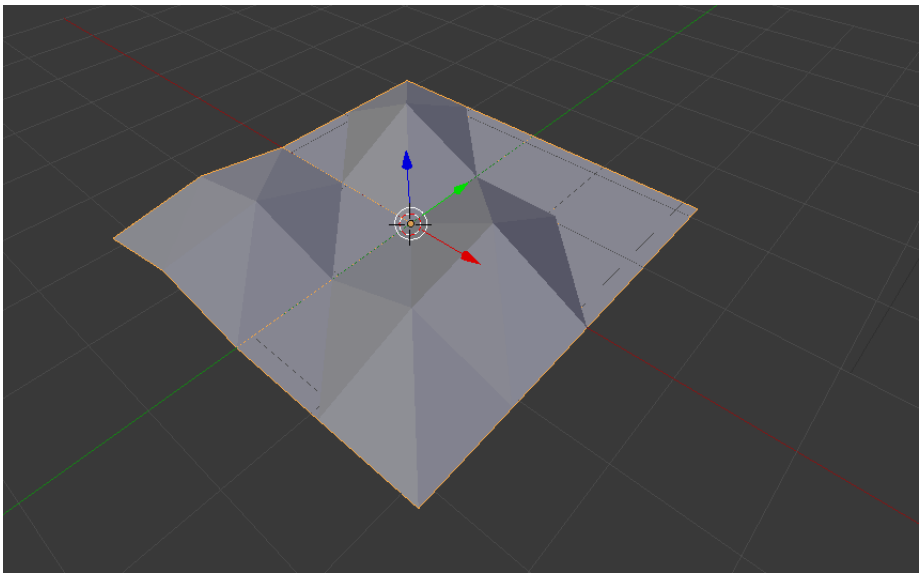

Рисунок 6 — Модель 1, состоящая из 32 полигонов.

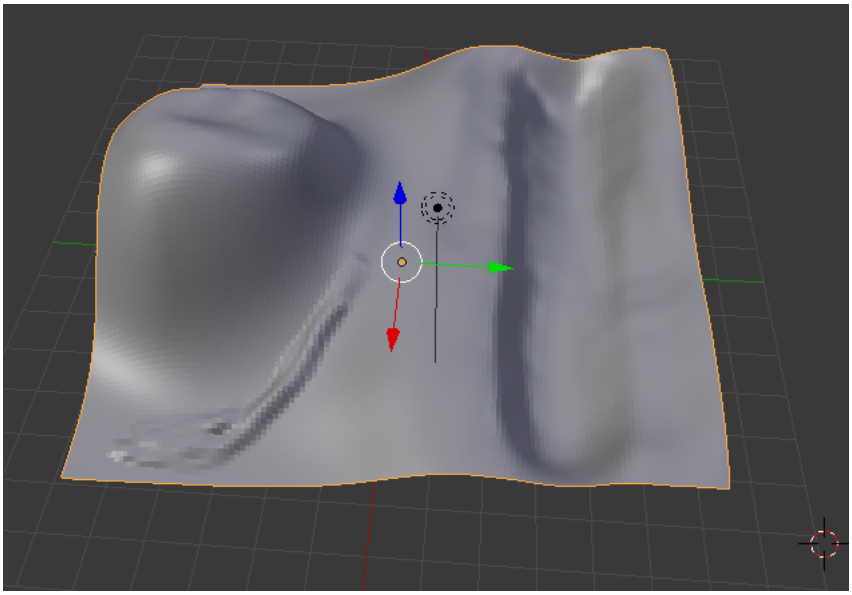

Рисунок 7 — Модель 2, состоящая из 32768 полигонов.

Результаты работы программы в зависимости от входных параметров приведены в таблице 1. Из результатов видно, что больший шаг соответствует менее

подробному разрезу и меньшему времени работы.

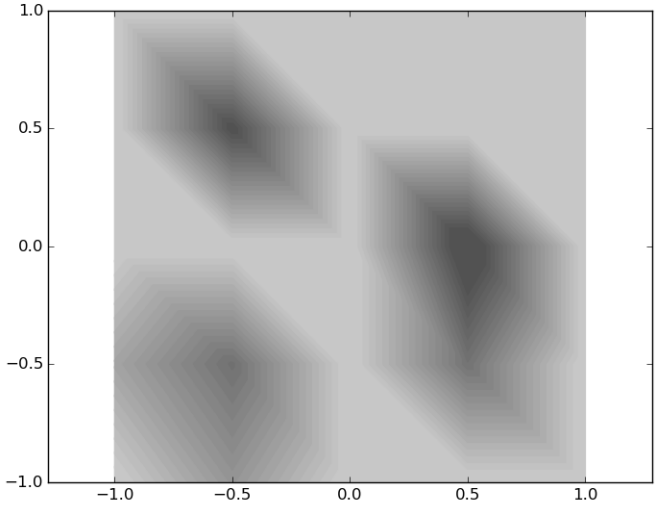

Рисунок 8 — Результат работы программы. Модель 1, нижняя граница 0, верхняя граница 0.25, шаг 0.01.

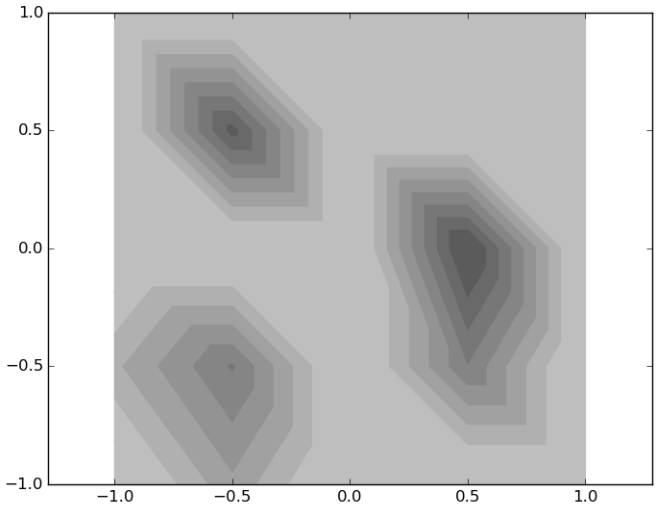

Рисунок 9 — Результат работы программы. Модель 1, нижняя граница 0, верхняя граница 0.25, шаг 0.03.

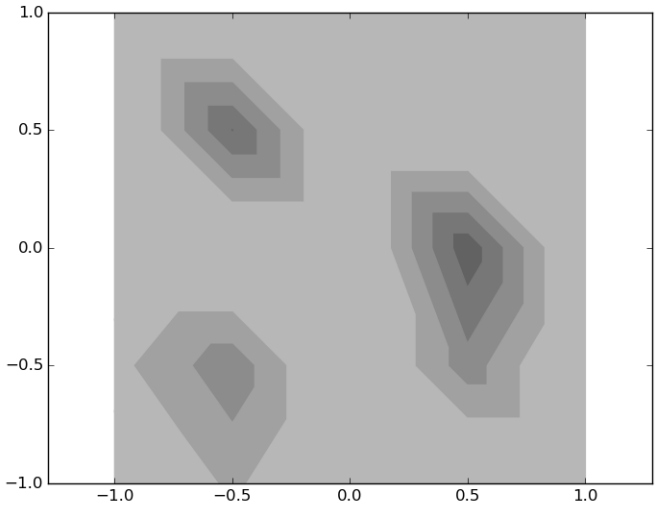

Рисунок 10 — Результат работы программы. Модель 1, нижняя граница 0, верхняя граница 0.25, шаг 0.05.

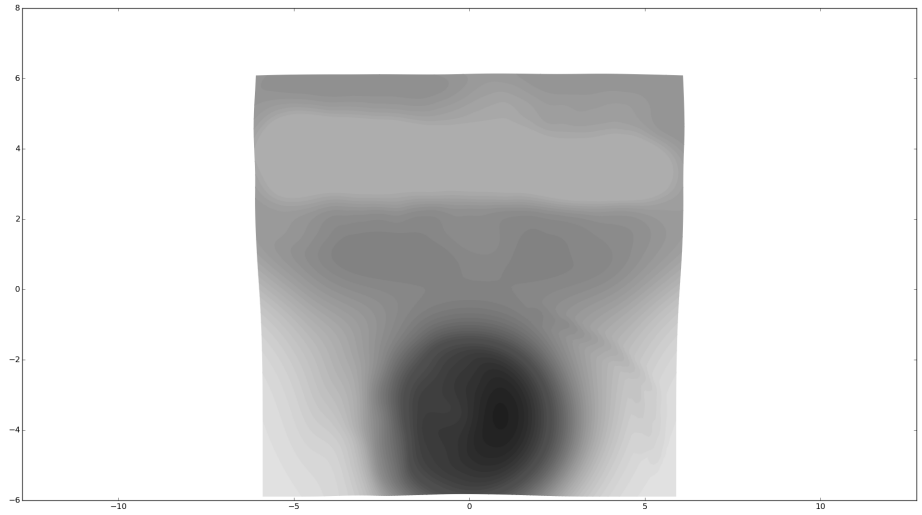

Рисунок 11 — Результат работы программы. Модель 2, нижняя граница 0, верхняя граница 4.25, шаг 0.1.

Таблица 1: Результаты работы программы в зависимости от входных параметров.

Название модели	**Нижняя граница**	**Верхняя граница**	**Шаг**	**Время работы (секунды)**	**Рисунок с результатом**
Модель 1	0	0.25	0.01	0.513	Рисунок 8
Модель 1	0	0.25	0.03	0.440	Рисунок 9
Модель 1	0	0.25	0.05	0.413	Рисунок 10
Модель 2	0	4.25	0.1	13.312	Рисунок 11
Модель 2	0	4.25	0.25	6.670	Рисунок 12
Модель 2	0	4.25	0.5	4.116	Рисунок 13

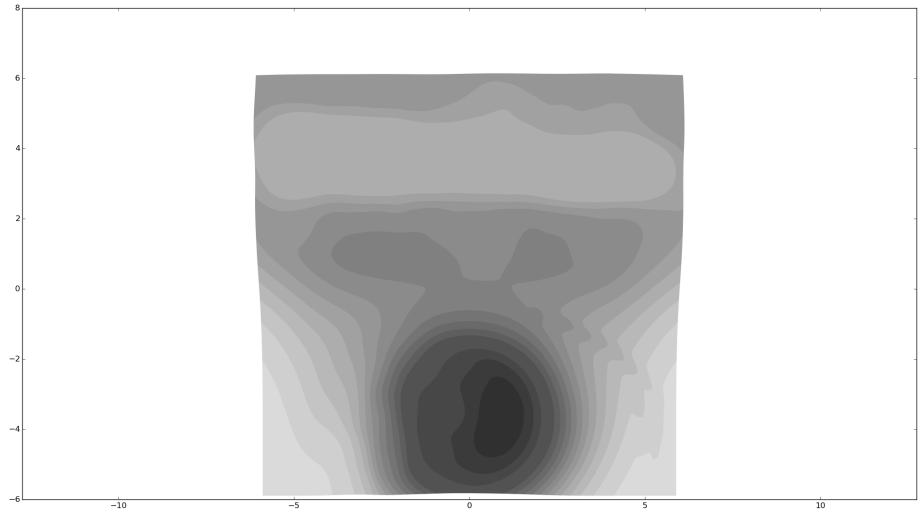

Рисунок 12 — Результат работы программы. Модель 2, нижняя граница 0, верхняя граница 4.25, шаг 0.25.

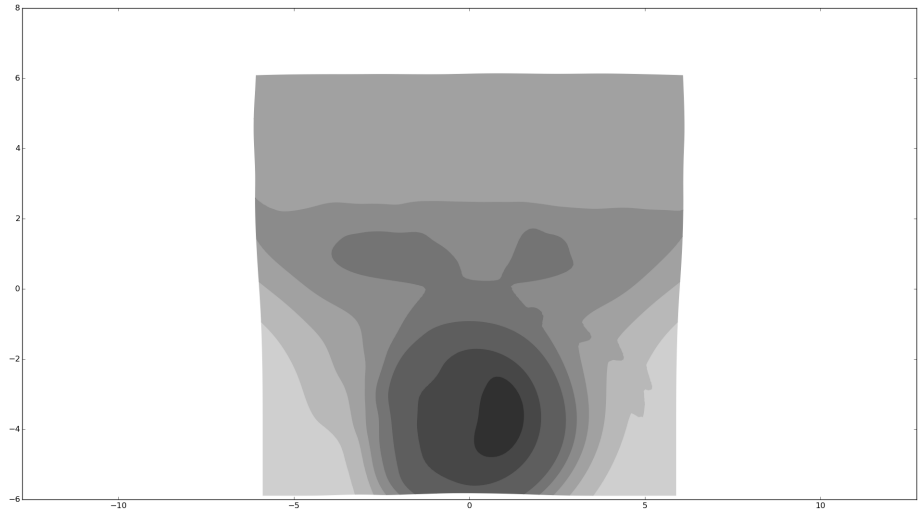

Рисунок 13 — Результат работы программы. Модель 2, нижняя граница 0, верхняя граница 4.25, шаг 0.5.

5. Выводы

В рамках лабораторной работы были реализован алгоритм построения изоконтуров. Написана программа, позволяющая по модели строить горизонтальный разрез и визуализировать результаты. Из анализа результатов было получено: чем более крупный шаг мы выбираем, тем получаем менее подробную модель, но за меньшее время.