МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Ярославский государственный университет им. П. Г. Демидова»

Кафедра математического анализа

Сдано на кафедру
«16» июня 2020 г.
Заведующий кафедрой
д. ф.-м. н., доцент

_____ Невский М. В.

Выпускная квалификационная работа

Расчёт параметров и моделирование изгибаемых многогранных поверхностей, составленных из параллелограммов

направление подготовки
01.03.02 Прикладная математика и информатика

Научный руководитель
старший преподаватель

_____ Алексеев В. В.
«16» июня 2020 г.

Студент группы ПМИ-41БО

_____ Хорошева Е. В.
«16» июня 2020 г.

Ярославль 2020 г.

Реферат

Объем 40 с., 4 гл., 26 рис., 0 табл., 8 источников, 1 прил.

Ключевые слова: **вектор, двугранный угол, координаты, многогранные изгибаемые поверхности, параллелограммы, параллельный перенос, Python, STL.**

В работе рассмотрен частный случай многогранных изгибаемых поверхностей, а именно поверхностей, составленных из равных параллелограммов. Исследуется ситуация, когда при сгибе соседних параллелограммов по общему ребру смежная пара параллелограммов сгибается в противоположную сторону.

Задача состоит в определении параметров, описывающих изгибаемую многогранную поверхность, и её моделирование.

Поставленную задачу можно решить несколькими подходами. В работе рассмотрены два метода. Первый метод основывается на применении матриц поворота и отражения. Второй способ заключается в анализе углов между гранями поверхности.

Найдено соотношение, параметризующее все точки изгибаемой поверхности. По полученным результатам написана программа на языке Python. Реализованное приложение позволяет вычислить координаты вершин участка поверхности и визуализировать его. При задании различных начальных параметров можно увидеть особенности динамики изменения изгибаемой многогранной поверхности. Проведена адаптация и подготовка модели с целью получения возможности её 3D-печати.

Выбранная тема интересна для изучения и находит своё применение в инженерии и промышленности.

Содержание

Введение

Многогранной поверхностью называется поверхность, состоящая из набора плоских многоугольников (например, треугольников или параллелограммов). Элементами многогранной поверхности являются её вершины, рёбра — стороны граней и грани, которыми являются многоугольники.

Согласно известной классификации [1], многогранные поверхности могут быть изгибаемыми. Их особенностью является то, что пространственную форму можно изменить непрерывной деформацией. В процессе такой деформации каждая грань не изменяет своих размеров (движется как твёрдое тело), а деформация осуществляется только за счёт непрерывного изменения двугранных углов.

Отдельный интерес представляют изгибаемые поверхности, составленные из равных выпуклых 4-угольников, допускающие плоскую реализацию (в виде замощения плоскости). Они нашли применение в промышленности в качестве солнечных панелей для космических аппаратов [2]. Также они используются и в архитектурном дизайне: изогнутые фасады или крыши из четырехугольных стеклянных панелей встречаются в современных городах. Известны условия, при которых такие поверхности обладают свойством изгибаемости: комплекс, состоящий из граней и её 8 соседей, допускает непрерывное изменение [3].

В большинстве известных работ, связанных с исследованием свойств изгибаемых поверхностей, рассматриваемые случаи сводятся к рассмотрению общей модели. Интерес состоит в том, чтобы произвести численное моделирование изгибаемой поверхности.

Выбранная тема не только представляет собой увлекательное математическое исследование, но и будет актуальна благодаря наличию приложений в архитектуре и технической деятельности.

В работе рассматривается частный случай изгибаемых многогранных поверхностей: поверхностей, составленных из равных параллелограммов.

Цель выпускной квалификационной работы состоит в:

1. Определении параметров, описывающих изгибаемую многранную поверхность, составленную из равных параллелограммов.
2. Поиске алгоритма, позволяющего вычислить координаты произвольного участка поверхности по данному набору параметров.
3. Программной реализации трёхмерной модели данной поверхности.
4. Адаптации данной модели с целью обеспечения возможности её 3D-печати.

1. Постановка задачи

Рассмотрим замощение плоскости равными параллелограммами (Рис. 1):

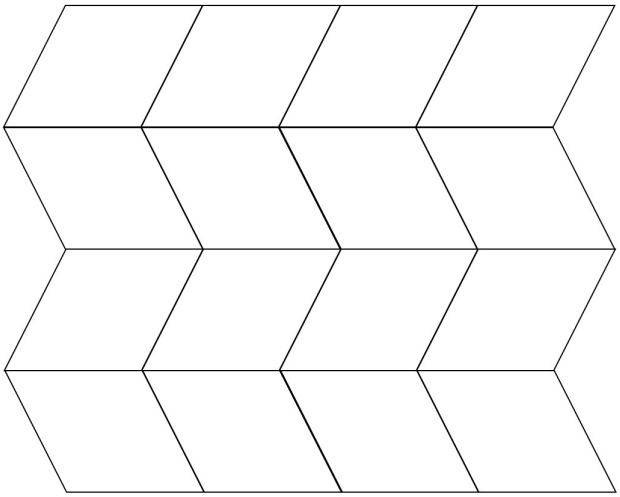

Рис. 1 — Пример составленной из равных параллелограммов поверхности

Рассмотрим два соседних параллелограмма. Если согнуть поверхность по их общему ребру, то следующая пара параллелограммов может согнуться в ту же сторону (этот случай тривиален), а может в противоположную (Рис. 2, Рис. 3). В данной работе нас будет интересовать именно второй случай.

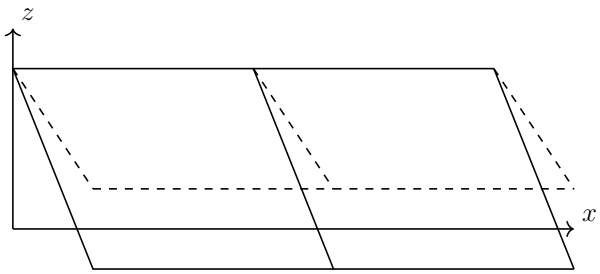

Рис. 2 — Следующая пара параллелограммов сгибается в туже строну

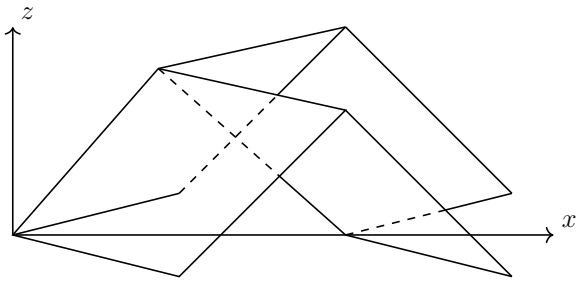

Рис. 3 — Следующая пара параллелограммов сгибается в противоположную сторону

Задача состоит в моделировании такой поверхности, параметризованной длинами сторон параллелограмма, углом α параллелограмма и двугранным углом сгиба ψ.

Рассмотрим участок поверхности, состоящий из четырех соседних параллелограммов (Рис. 4). Будем симметрично относительно плоскости Oxz сгибать

закрашенный параллелограмм к верхнему параллелограмму (Рис. 5). Опишем движение отмеченных вершин в зависимости от угла сгиба ψ. Сгибать можно только по линиям, грани жёсткие (несгибаемые).

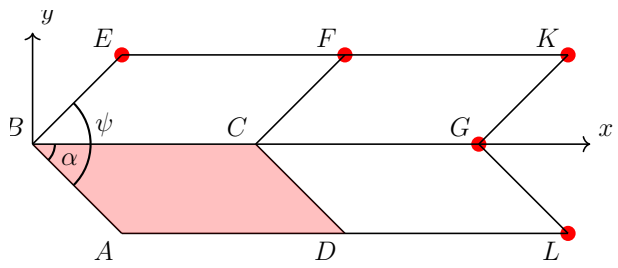

Рис. 4 — Участок поверхности, состоящий из четырех соседних параллелограммов

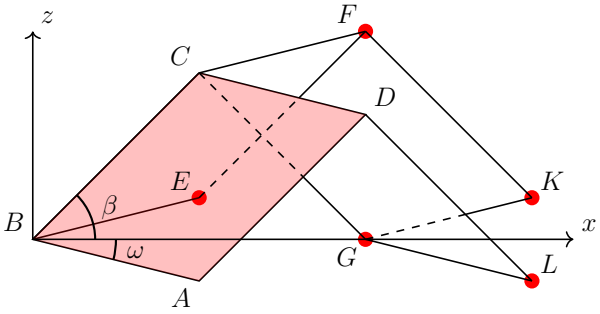

Рис. 5 — Вид рассматриваемого участка поверхности после сгиба

2. Методы решения

В данной главе будут рассмотрены два метода решения задачи: линейно - алгебраический и геометрический. В первом элементы можно найти, используя матрицы поворота и отражения. Второй способ будет основан на анализе углов между гранями поверхности.

2.1. Линейно - алгебраический метод

Пусть нам даны координаты вершин параллелограмма $ABCD$. Пусть B лежит в начале координат, ψ — двугранный угол сгиба между параллелограммами (Рис. 4).

Для того, чтобы найти точку E, необходимо осуществить поворот точки A на угол ψ вокруг $\overrightarrow{BC}$ [5].

Любое вращение в трёхмерном пространстве может быть представлено последовательностью поворотов вокруг трёх ортогональных осей (например, вокруг осей декартовых координат). Этой последовательности соответствует матрица, равная произведению соответствующих матриц поворота. Матрицей поворота назовём ортогональную матрицу, используемую для выполнения собственного ортогонального преобразования в евклидовом пространстве.

Пусть $\overrightarrow{BC} = (C_x,C_y,C_z)$. Пусть ψ — угол поворота, а ось вращения задана единичным вектором $\overrightarrow{V} = (V_x,V_y,V_z)$, где

$$\overrightarrow{V}(V_x, V_y, V_z) = \frac{(C_x, C_y, C_z)}{\sqrt{C_x^2 + C_y^2 + C_z^2}}. \tag{1}$$

Тогда матрица поворота в декартовых координатах имеет вид

$$M(\vec{v}, \psi) = \begin{pmatrix} \cos\psi + (1-\cos\psi)V_x^2 & (1-\cos\psi)V_xV_y - (\sin\psi)V_z & (1-\cos\psi)V_xV_z + (\sin\psi)V_y \\ (1-\cos\psi)V_yV_x + (\sin\psi)V_z & \cos\psi + (1-\cos\psi)V_y^2 & (1-\cos\psi)V_yV_z - (\sin\psi)V_x \\ (1-\cos\psi)V_zV_x - (\sin\psi)V_y & (1-\cos\psi)V_zV_y + (\sin\psi)V_x & \cos\psi + (1-\cos\psi)V_z^2 \end{pmatrix} \tag{2}$$

Координаты точки E определяются произведением $M \cdot \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$, (3)

где $A_x,\ A_y,\ A_z$ — координаты точки A.

Сдвинем точки F, C и D на вектор $\overrightarrow{BC}$. Получим новые точки K^*, G^*, L^* (Рис. 6).

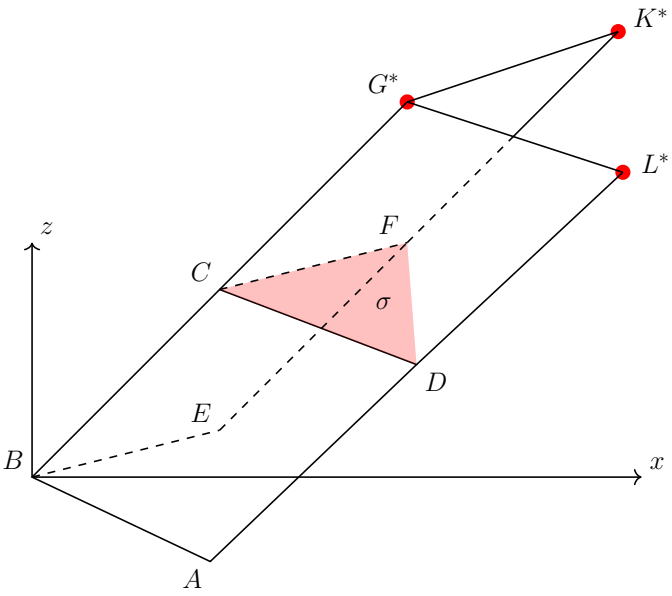

Рис. 6 — Расположение точек K^*, G^*, L^*

Пусть $F(F_x, F_y, F_z), C(C_x, C_y, C_z), D(D_x, D_y, D_z)$. Уравнение плоскости [4], проходящей через точки F, C, D, в координатной форме будет иметь вид (Рис. 6):

$$\begin{vmatrix} x - F_x & y - F_y & z - F_z \\ C_x - F_x & C_y - F_y & C_z - F_z \\ D_x - F_x & D_y - F_y & D_z - F_z \end{vmatrix} = 0. \tag{4}$$

$$\begin{vmatrix} C_y - F_y & C_z - F_z \\ D_y - F_y & D_z - F_z \end{vmatrix} \cdot x - \begin{vmatrix} C_x - F_x & C_z - F_z \\ D_x - F_x & D_z - F_z \end{vmatrix} \cdot y + \begin{vmatrix} C_y - F_y & C_z - F_z \\ D_y - F_y & D_z - F_z \end{vmatrix} \cdot z = 0 \tag{5}$$

Получаем уравнение

$$\Delta x \cdot x - \Delta y \cdot y + \Delta z \cdot z - (F_x \cdot \Delta x - F_y \cdot \Delta y + F_z \cdot \Delta z) = 0 \tag{6}$$

Обозначим заданную плоскость за σ. Пусть $\overrightarrow{N} = (\Delta x, -\Delta y, \Delta z)$ – вектор нормали к этой плоскости. Тогда единичным вектором нормали будет

$$\overrightarrow{n}(n_x, n_y, n_z) = \frac{(\Delta x, -\Delta y, \Delta z)}{\sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}}. \tag{7}$$

Координаты K, G, L получим отражением точек K^*, G^*, L^* относительно плоскости σ. Рассмотрим алгоритм решения для точки K. Точки G и L находятся аналогично.

Найдем уравнение прямой, проходящей через точку K^* перпендикулярно

плоскости σ. В качестве её направляющего вектора возьмём вектор нормали плоскости.

$$\frac{x - K_x^*}{\Delta x} = \frac{y - K_y^*}{-\Delta y} = \frac{z - K_z^*}{\Delta z} \tag{8}$$

Найдем точку пересечения прямой и плоскости.

$$\frac{x - K_x^*}{\Delta x} = \frac{y - K_y^*}{-\Delta y} = \frac{z - K_z^*}{\Delta z} = t \quad \Rightarrow \quad \begin{cases} x = K_x^* + t \cdot \Delta x \\ y = K_y^* - t \cdot \Delta y \\ z = K_z^* + t \cdot \Delta z \end{cases} \tag{9}$$

Найдём t, подставив координаты x, y, z в уравнение плоскости, получим

$$\begin{aligned} \Delta x \cdot (K_x^* + t \cdot \Delta x) - \Delta y \cdot (K_y^* - t \cdot \Delta y) + \Delta z \cdot (K_z^* + t \cdot \Delta z) - \\ - (F_x \cdot \Delta x - F_y \cdot \Delta y + F_z \cdot \Delta z) = 0. \end{aligned} \tag{10}$$

Откуда можно записать точку $K'(x, y, z)$ (Рис. 7). Точка K' является серединой отрезка KK^*, где точка $K(K_x, K_y, K_z)$ является точкой, симметричной относительно проскости σ, точке $K^*(K_x^*, K_y^*, K_z^*)$, поэтому

$$\begin{aligned} K'_x = \frac{K_x + K_x^*}{2} \quad &\Rightarrow \quad K_x = 2K'_x - K_x^*, \\ K'_y = \frac{K_y + K_y^*}{2} \quad &\Rightarrow \quad K_y = 2K'_y - K_y^*, \\ K'_z = \frac{K_z + K_z^*}{2} \quad &\Rightarrow \quad K_z = 2K'_z - K_z^*. \end{aligned} \tag{11}$$

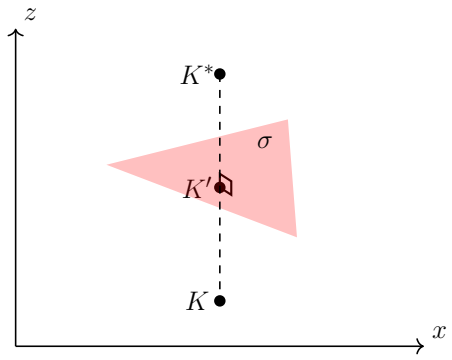

Рис. 7 — Отражение точки K^* относительно плоскости σ

Таким образом, мы нашли все координаты точек участка поверхности. Остальные точки получаются параллельным переносом участка поверхности на вектор

$$(m \cdot |BG|, n \cdot |AE|, 0), \quad m, n \in \mathbb{Z}. \tag{12}$$

2.2. Геометрический метод

В данном разделе будет рассмотрен метод параметризации вершин поверхности на основе геометрических соотношений.

Пусть заданы $a = |BC|$, $b = |BA|$, α — острый угол параллелограмма, ψ — угол сгиба вдоль отрезка BC. Проведём координатные оси, как показано на рисунке 4.

Поместим точку B в начало координат, ось Ox проведём в направлении вектора $\overrightarrow{BC}$.

Обозначим угол между отрезком BA и осью Ox как ω (Рис. 8).

Выразим координаты вершин A и E. Стороны BA и BE лежат в плоскости Oxy. Из этого следует, что координата z искомых точек равна нулю.

Рассмотрим ΔABN, изображённый на рисунке 8. Этот треугольник является прямоугольным, поэтому для угла ω будут верны следующие соотношения:

$$\cos\omega = \frac{|BN|}{|BA|},\ \sin\omega = \frac{|AN|}{|BA|}. \tag{13}$$

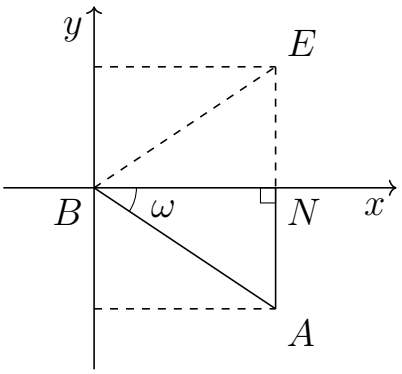

Рис. 8 — Вид на плоскость Oxy

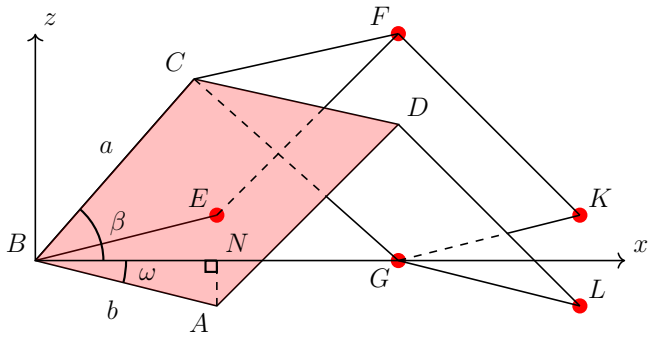

Рис. 9 — Расположение точки N

Выразим $\sin\omega$ через заданные элементы. В ΔAON (Рис. 10) точка N принадлежит оси Ox (Рис. 8, Рис. 9), а грани параллелограммов движутся симметрично относительно плоскости Oxz, поэтому $\angle AON = \frac{\psi}{2}$, $|AN| = \sin\frac{\psi}{2} \cdot |AO|$. Рассмотрим ΔABO (Рис. 11), получим $|BA| = \frac{|AO|}{\sin\alpha}$. Тогда

$$\sin\omega = \sin\alpha \cdot \sin\frac{\psi}{2}. \tag{14}$$

На рисунке 8 значение координаты x точки A будет равно длине отрезка BN, координата y равна $-|AN|$. Для точки E: $x = |BN|$, $y = |EN| = |AN|$ (в силу симметрии относительно Oxz). Запишем координаты искомых точек:

$$A = (b\cos\omega, -b\sin\omega, 0),\ E = (b\cos\omega, b\sin\omega, 0). \tag{15}$$

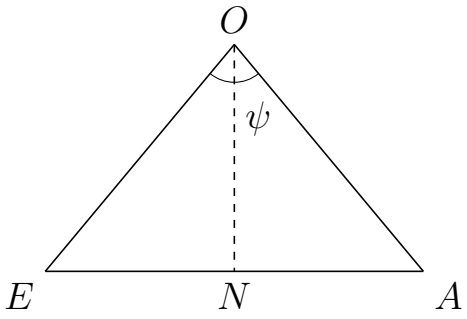

Рис. 10 — Расположение элементов ΔAON

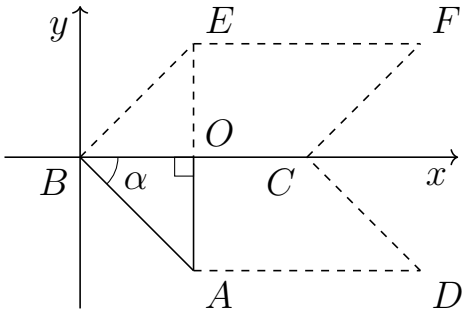

Рис. 11 — Точка O на плоскости

Для получения координат точки C сначала рассмотрим отрезок BC (Рис. 12). Он лежит в плоскости Oxz. Поэтому координата y этой точки принимает нулевое значение.

Пусть β — угол между отрезком BC и осью Ox (Рис. 12), угол ω — угол между отрезком BA и осью Ox на плоскости Oxy (Рис. 8), угол α — угол между отрезками BA и BC (Рис. 11). Рассмотрим трёхгранный угол, ограниченный плоскостями Oxy, Oxz и гранью параллелограмма $ABCD$. Запишем теорему косинусов для трёхгранного угла. Угол между плоскостями Oxy и Oxz равен $\frac{\pi}{2}$.

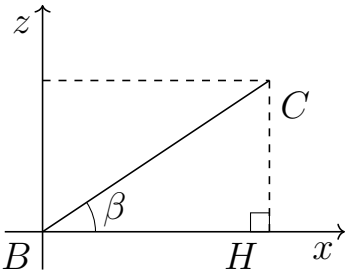

Рис. 12 — Расположение угла β

Получаем,

$$\cos\alpha = \cos\beta \cdot \cos\omega + \sin\beta \cdot \sin\omega \cdot \cos\frac{\pi}{2}. \tag{16}$$

Из этого следует, что $\cos\beta = \dfrac{\cos\alpha}{\cos\omega}$. В прямоугольном треугольнике BCH для $\sin\beta$ и $\cos\beta$ справедливы следующие равенства (Рис. 12):

$$\cos\beta = \frac{|BH|}{|BC|}, \quad \sin\beta = \frac{|CH|}{|BC|}. \tag{17}$$

Все необходимые значения найдены, и мы можем записать координаты точки.

$$C = (a\cos\beta, 0, a\sin\beta) \tag{18}$$

Перейдём к рассмотрению точки G. Она принадлежит плоскости Oxz. Поэтому $y = 0$. Также эта точка лежит на оси Ox. Поэтому $z = 0$. Пусть $C(C_x, C_y, C_z)$, тогда

$$G(2\,C_x, 0, 0). \tag{19}$$

Координаты остальных точек выражаются из следующих векторных соотношений:

$$\begin{aligned} \overrightarrow{BF} &= \overrightarrow{BC} + \overrightarrow{BE}, \quad \overrightarrow{BD} = \overrightarrow{BC} + \overrightarrow{BA}, \\ \overrightarrow{CK} &= \overrightarrow{CG} + \overrightarrow{CF}, \quad \overrightarrow{CL} = \overrightarrow{CG} + \overrightarrow{CD}. \end{aligned} \tag{20}$$

Пусть $C(C_x, C_y, C_z)$, $A(A_x, A_y, A_z)$, $E(E_x, E_y, E_z)$, $F(F_x, F_y, F_z)$, $D(D_x, D_y, D_z)$. Тогда координаты оставшихся точек могут быть найдены по следующим формулам:

$$\begin{aligned} F &= (C_x + E_x, C_y + E_y, C_z + E_z), \\ D &= (C_x + A_x, C_y + A_y, C_z + A_z), \\ K &= (F_x + C_x, F_y - C_y, F_z - C_z), \\ L &= (D_x + C_x, D_y - C_y, D_z - C_z). \end{aligned} \tag{21}$$

Координаты вершин всех четырёх параллелограммов найдены.

Координаты вершин произвольного участка поверхности получаются параллельным переносом на вектор

$$(m \cdot |BG|, n \cdot |AE|, 0), \quad m, n \in \mathbb{Z}. \tag{22}$$

3. Программная реализация

Для демонстрации смоделированных поверхностей по различным наборам параметров, написана программа на языке Python. Для построения и работы с графиками использована библиотека matplotlib [6].

Интерфейс приложения показан на рисунке 13. Программа позволяет производить построение участка поверхности произвольной величины с указанием параметров.

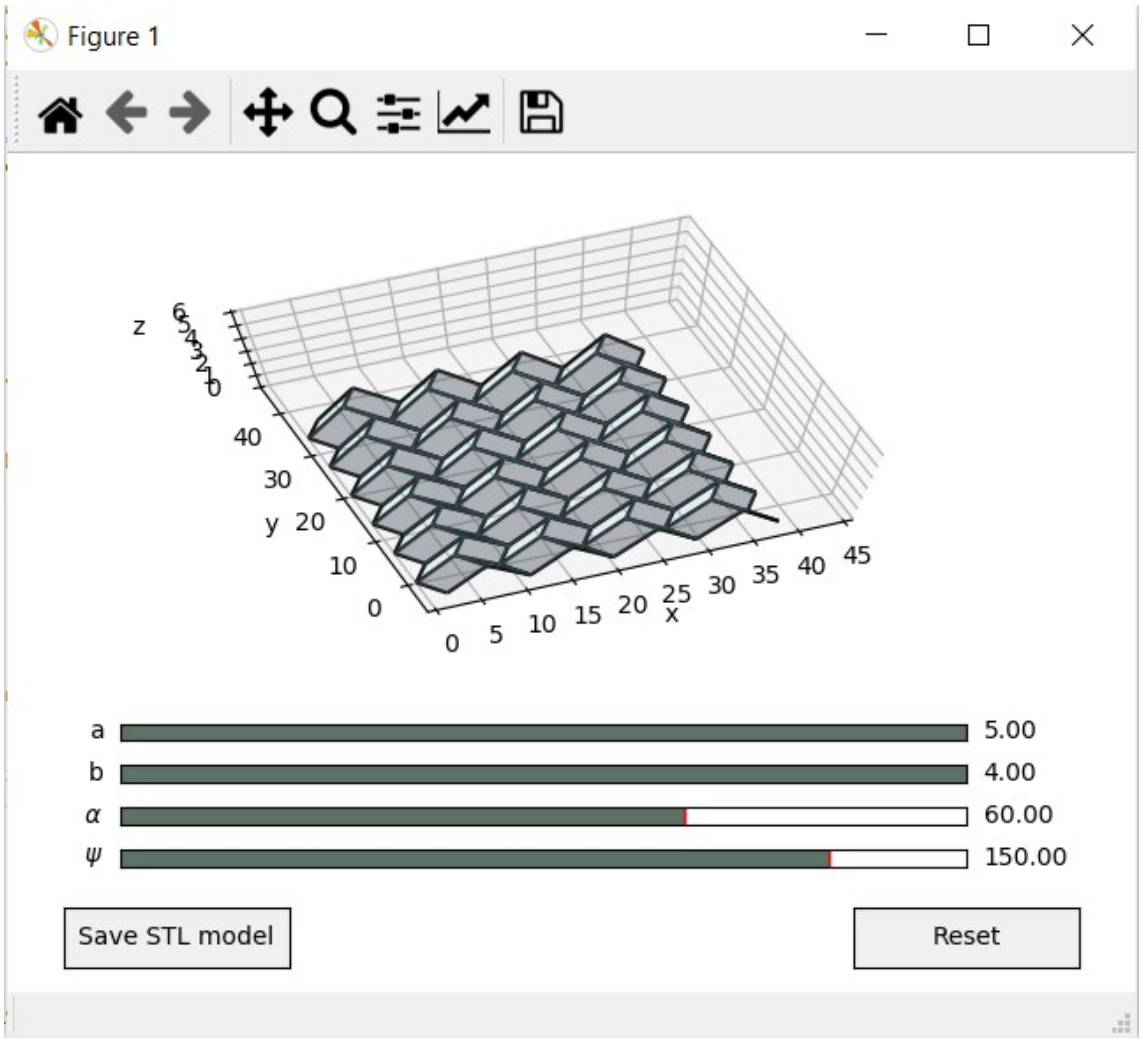

Рис. 13 — Интерфейс приложения для демонстрации смоделированных поверхностей по различным наборам параметров

Основную часть панели под названием Figure 1 занимает непосредственно сам график поверхности в трёхмерной системе координат. На осях x, y, z указаны значения. Под графиком добавлены слайдеры (Slider), которые позволяют изменять значения параметров «на лету». a — длина отрезка BC, b — длина отрезка BA, α — градусная мера угла параллелограмма α, ψ — градусная мера угла ψ,

угла сгиба вдоль отрезка BC (Рис. 4). Ниже расположены две кнопки (Button). По нажатию на Save STL model программа позволяет сохранить участок поверхности в формате STL. Подробнее см. Глава 4. Код программы см. в Приложении А.

Вычисления и построение реализованы в функциях CalkParam и Draw. В CalkParam вычисляются длины отрезков AE, BG и координаты точек A, E, C, F, D, G, K, L. В функции draw формируются списки всех вершин поверхности, задаются оси и их размеры, вызывается процедура отрисовки графика.

Далее продемонстрированы примеры поверхностей, смоделированных по различным наборам параметров.

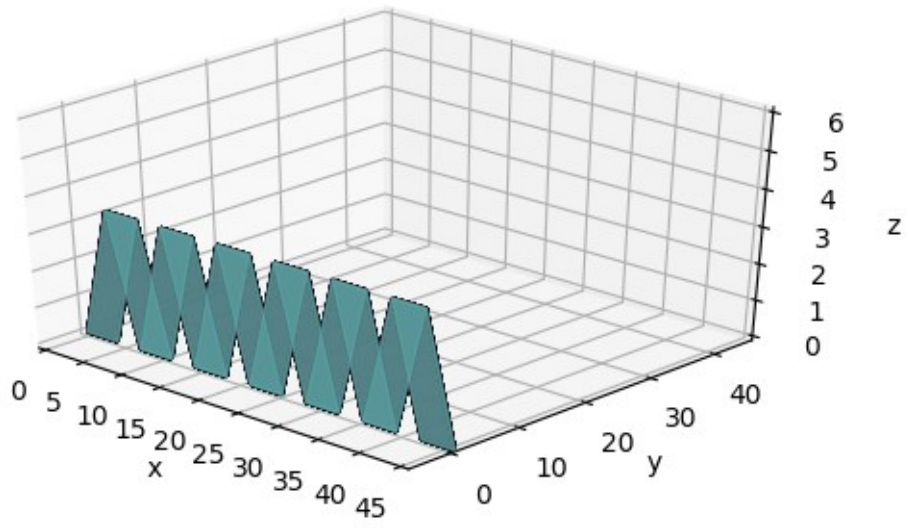

Рис. 14 — Пример многогранной поверхности, построенной с помощью второго метода при $|\overrightarrow{BC}| = 5, |\overrightarrow{BA}| = 4, \alpha = 45°, \psi = 0°$.

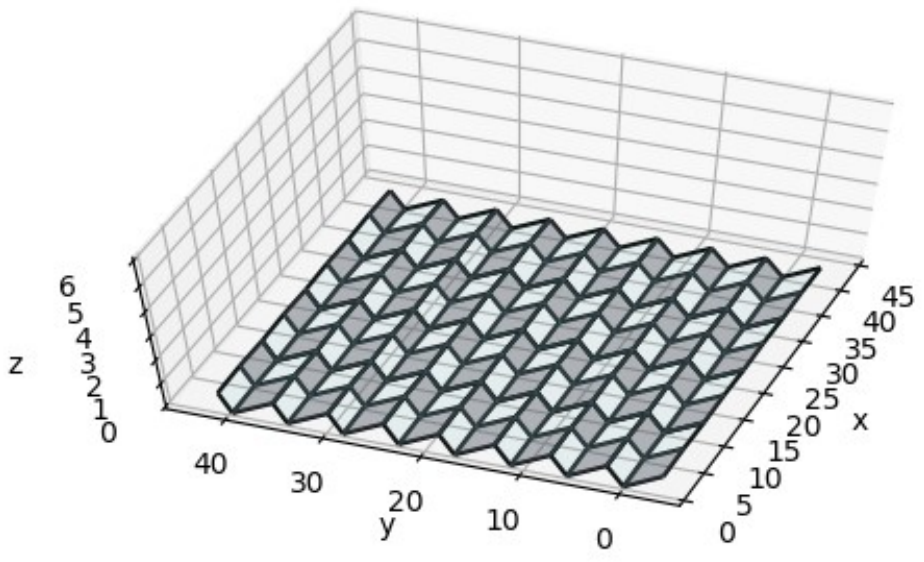

Рис. 15 — Пример многогранной поверхности, построенной с помощью второго метода при $|\overrightarrow{BC}| = 5, |\overrightarrow{BA}| = 4, \alpha = 45°, \psi = 179°$.

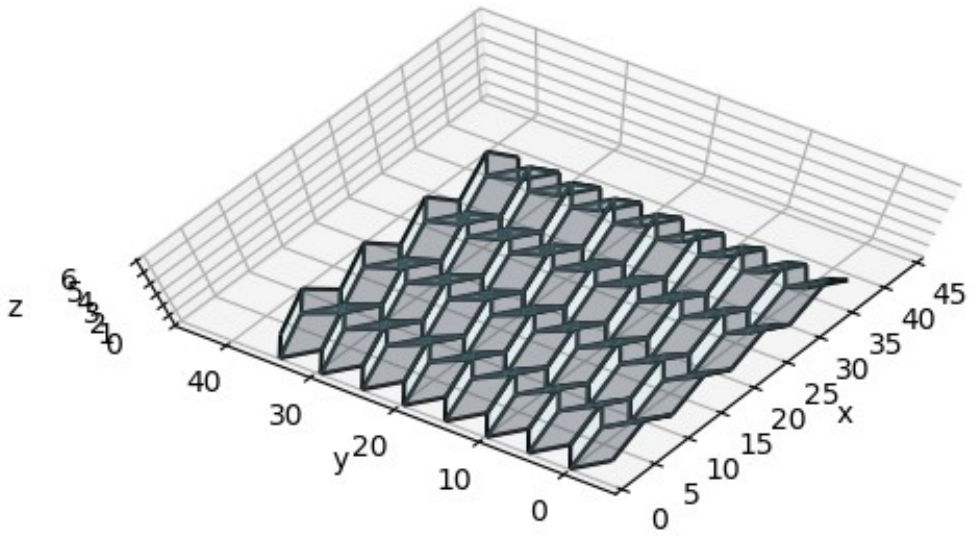

Рис. 16 — Пример многогранной поверхности, построенной с помощью второго метода при $|\overrightarrow{BC}| = 5, |\overrightarrow{BA}| = 4, \alpha = 45°, \psi = 120°$.

4. Создание трёхмерной модели

3D-моделирование - процесс (совокупность инструментов и приемов), при помощи которого можно создавать объемные объекты. Он обеспечивает разработку наглядной копии поверхности, что представляет возможность наиболее полно передавать информацию об изменениях объекта, анализировать характеристики. Для создания трехмерной модели требуются специальные программные (приложения 3D-визуализации) и аппаратные средства (компьютер, 3D-мониторы, 3D-принтеры).

Полигональная трёхмерная модель состоит из множества точек, которые соединяются между собой гранями и образуют полигоны. Вершиной является точка, имеющая в трехмерной системе свои уникальные координаты (x, y, z). Преимуществом трёхмерных моделей является их наглядность и информативность.

В настоящее время создание и использование 3D-моделей находит применение в различных областях деятельности: образовании, науке, строительстве, картографии и т.д.

В данной работе требуется создать трёхмерную модель поверхности пригодной для 3D-печати. Глава 4 посвящена описанию того, каким образом можно создать подобную модель.

4.1. Необходимые вычисления

Построенная в Главе 3 поверхность является двумерной и не пригодной для 3D-печати. Целью данного раздела является демонстрация техники преобразования трёхмерной модели из двумерной поверхности.

Аппроксимируем реализованную изгибаемую поверхность набором многогранников (каждой грани будет соответствовать некий многогранник).

В Главе 2 показан алгоритм вычисления координат вершин двумерной изгибаемой поверхности.

Заменим грани поверхности призмами высоты e (e — параметр модели). При этом образуется множество самопересечений поверхности на стыке граней. Чтобы избежать самопересечений, раздвинем соседние грани на величину l (l — параметр модели) (Рис. 17).

Рассмотрим произвольную грань $ABCD$ поверхности. Далее на рисунках 18 и 19 она выделена. Расположим её в декартовой системе координат, как показано на рисунке 5. Обозначим грань $ABCD$, как $\mu_{0,0}$. Индексируем все грани поверхности целыми числами так, что грань $\mu_{m,n}$ получается параллельным переносом грани $\mu_{0,0}$ на вектор $(m \cdot |BG|, n \cdot |AE|, 0), \quad m,n \in \mathbb{Z}$ (Рис. 18).

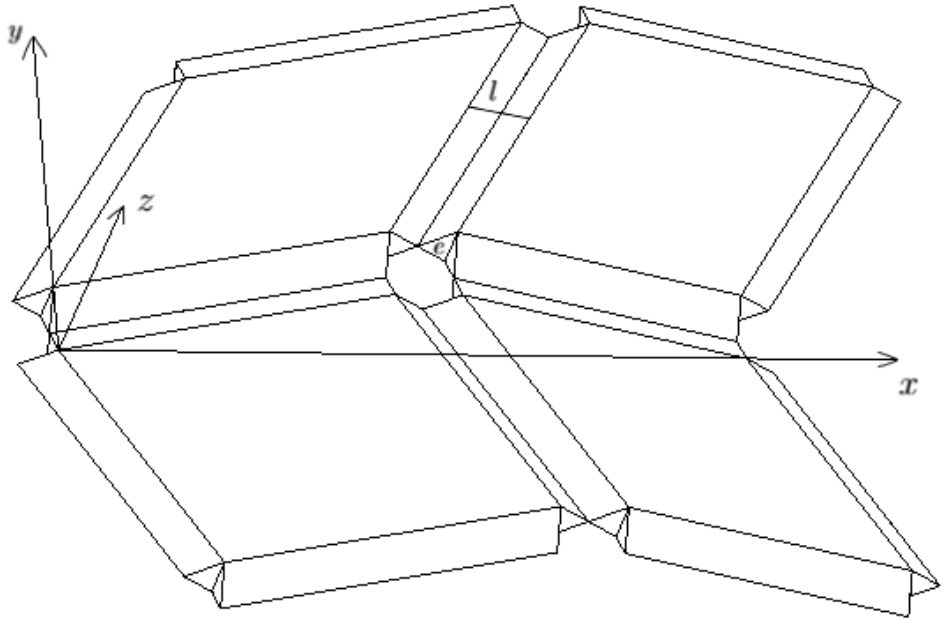

Рис. 17 — Часть создаваемой трёхмерной модели

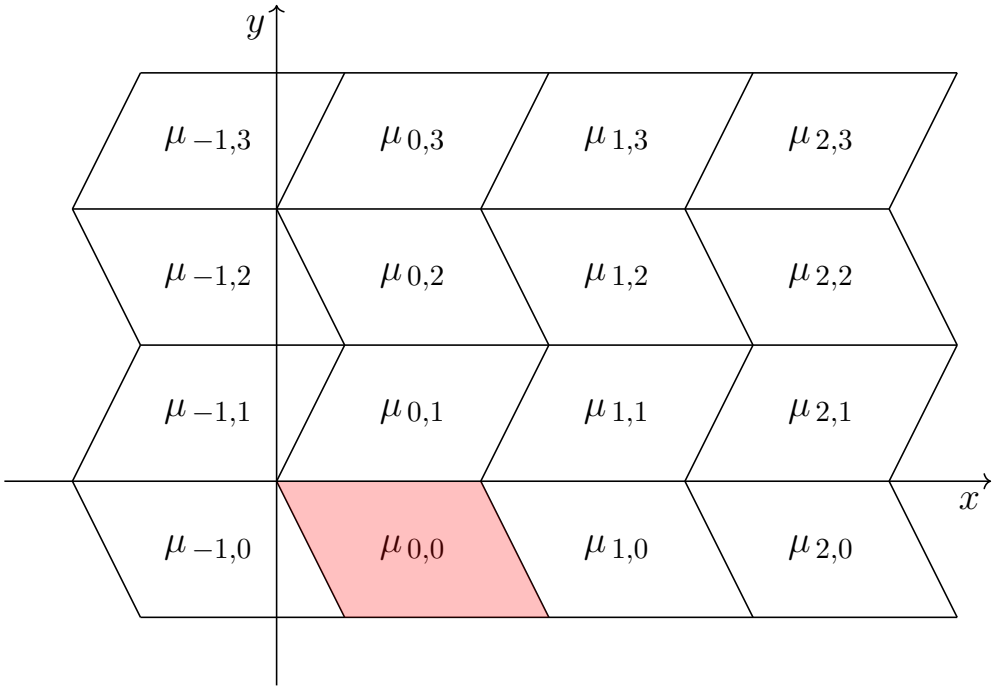

Рис. 18 — Индексация граней двумерной поверхности

Далее каждую грань $\mu_{m,n}$ сдвигаем на вектор $(l_m, l_n, 0), \quad m,n \in \mathbb{Z}$. Результат преобразований покажем на рисунке 19.

На следующем этапе создадим смещённую копию полученного ранее слоя. Это позволит получить объёмный аналог исследуемой поверхности. Построим на гранях полученные поверхности, в отрицательном направлении оси Oz, как

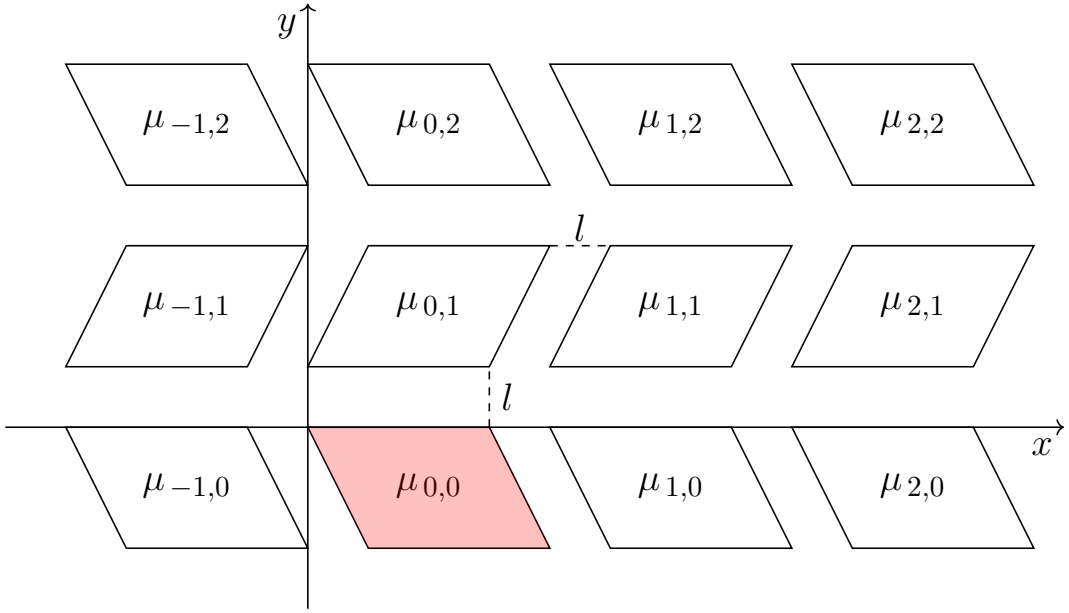

Рис. 19 — Вид двумерной поверхности после раздвижения граней

на основаниях, призмы высотой e. В результате получим набор прямых призм с ребром e; промежуток между соседними равен l.

Для обеспечения связности и изгибаемости модели, на каждой из боковых граней полученных призм, как на боковой грани, построим прямую треугольную призму, с высотой основания равного $\frac{l}{2}$. Тогда треугольные призмы, построенные на соседних гранях поверхности пересекутся по отрезку, что обеспечит изгибаемость модели (Рис. 17). Запишем формулы для расчёта координат точек соединения (Рис. 20).

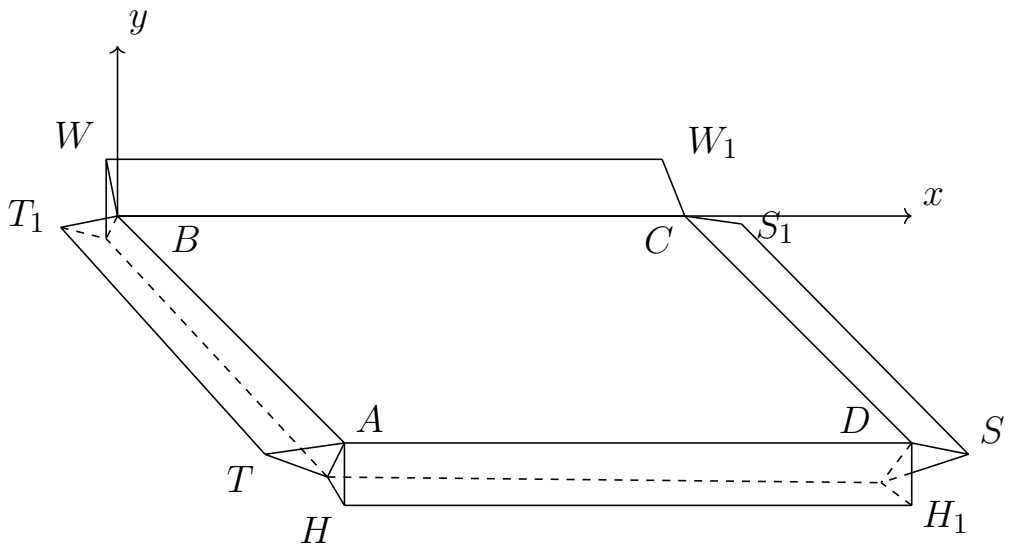

Рис. 20 — Вид трёхмерного элемента модели

Пусть $B(x_1, y_1, z_1), C(x_2, y_2, x_2), D(x_3, y_3, z_3), A(x_4, y_4, z_4)$, тогда

$$\begin{aligned} W &= (x_1,\, y_1 + \frac{l}{2},\, z_1 - \frac{e}{2}), \quad W_1 = (x_2,\, y_2 + \frac{l}{2},\, z_2 - \frac{e}{2}), \\ S &= (x_3 + \frac{l}{2},\, y_3,\, z_3 - \frac{e}{2}), \quad S_1 = (x_2 + \frac{l}{2},\, y_2,\, z_3 - \frac{e}{2}), \\ H &= (x_3,\, y_3 - \frac{l}{2},\, z_3 - \frac{e}{2}), \quad H_1 = (x_4,\, y_4 - \frac{l}{2},\, z_3 - \frac{e}{2}), \\ T &= (x_4 - \frac{l}{2},\, y_4,\, z_4 - \frac{e}{2}), \quad T_1 = (x_1 - \frac{l}{2},\, y_1,\, z_1 - \frac{e}{2}). \end{aligned} \tag{23}$$

Указанные соотношения задают трёхмерную модель изгибаемой поверхности.

4.2. Реализация

Для хранения информации о 3D-моделях существуют различные форматы файлов. Самые популярные из них: STL, OBJ, FBX, COLLADA.

3D-файл хранит информацию о геометрии, внешнем виде модели, а так же другую дополнительную информацию. Геометрия модели описывает ее форму. Внешний вид включает в себя цвета, текстуры, материал и т.д.

В работе используется STL-формат, как наиболее подходящий по функциональности и набору хранимых данных. STL — формат файла, который часто используют для хранения трёхмерных моделей объектов. Хранимые данные включают в себя список треугольников и их нормалей, в совокупности задающих поверхность объекта. STL-файл может быть текстовым (ASCII) или бинарным [8].

Пример файла типа ASCII STL выглядит следующим образом (Рис. 21).

solid name
facet normal $n_i\ n_j\ n_k$
outer loop
vertex $v1_x\ v1_y\ v1_z$
vertex $v2_x\ v2_y\ v2_z$
vertex $v3_x\ v3_y\ v3_z$
endloop
endfacet
endsolid name

Рис. 21 — Пример структуры файла STL в кодировке ASCII

Файлы этого типа начинаются со строки $solid\ name$ и заканчиваются строкой $endsolid\ name$. Название объекта, обозначенное словом $name$ — необяза-

тельно, но если $name$ опущено, то всё равно должен быть пробел после $solid$. Дальше файл заполняется списком произвольного числа треугольников, каждое n и v — число с плавающей точкой. В строке $facet\ normal\ n_i\ n_j\ n_k$ указываются координаты вектора нормали единичной длины; важна согласованность ориентаций нормалей. Далее записываются координаты каждой вершины треугольника. Они должны быть перечислены в порядке против часовой стрелки, относительно вектора нормали.

Поскольку файл ASCII STL может быть очень большим, существует двоичная версия этого формата. Такой файл начинается с заголовка из 80 символов (который обычно игнорируется, но не должен начинаться с $solid$, так как с этой последовательности начинается файл ASCII STL). После заголовка идет 4-байтовое беззнаковое целое число, указывающее количество треугольных граней в данном файле. После этого идут данные, характеризующие треугольники. Свойства для нормалей остаются такие же, как и в формате ASCII STL.

Каждый треугольник описывается двенадцатью 32-битными числами с плавающей точкой: 3 числа для нормали и по 3 числа на каждую из трёх вершин для координат x, y, z. Далее идут 2 байта беззнакового «short», который называется attribute byte count.

Пример файла STL двоичного формата приведён на рисунке 22.

UINT8[80] – Header
UINT32 – Number of triangles

foreach triangle
REAL32[3] – Normal vector
REAL32[3] – Vertex 1
REAL32[3] – Vertex 2
REAL32[3] – Vertex 3
UINT16 – Attribute byte count
end

Рис. 22 — Пример структуры двоичного файла STL

В работе будет использован формат ASCII STL.

Сформированная ранее модель состоит из четырёхугольников и треугольников. Разобьём на треугольники имеющуюся поверхность. Рассмотрим часть поверхности, состоящую из конструкции, изображённой на рисунке 20. После разбиения четырёхугольных граней на треугольники поверхность примет вид, показанный на рисунке 23.

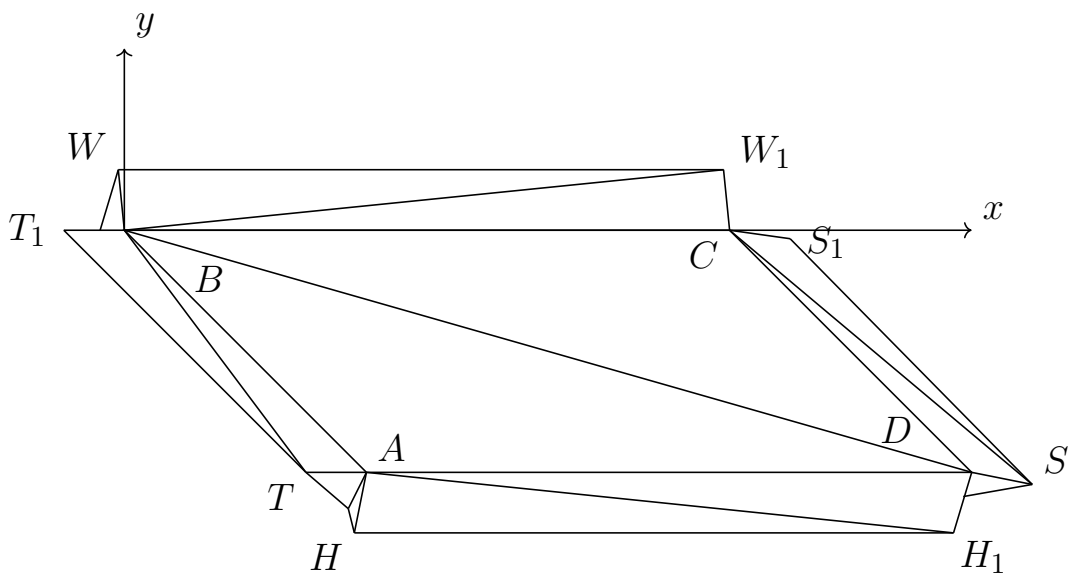

Рис. 23 — Элемент трёхмерной модели с замощением его поверхности треугольниками

После указанного разбиения рассматриваемая часть поверхности представляет собой набор треугольников и подходит для сохранения в формате STL. Аналогичную операцию проделаем с каждым элементом модели.

Приведённый алгоритм создания 3D-модели реализован в виде программы на языке Python. Преобразование данных в формат STL производится помощью библиотеки Numpy-STL [7]. С кодом программы можно ознакомиться в Приложении А.

Интерфейс приложения показан на рисунках 13 и 24. По нажатию кнопки Save STL Model вызывается функция ModelSTL, выполняющая преобразования данных модели в формат STL и сохранение полученного файла.

Основными функциями программы являются функции ModelSTL и CreateParallelogram. ModelSTL принимает на вход набор вершин параллелограммов, осуществляет преобразования координат. Затем для полученного набора вершин вызывается функция CreateParallelogram, реализующая преобразование набора вершин в набор призм. Далее осуществляется разбиение призм на треугольники, расчёт нормалей и сохранение модели в формате STL.

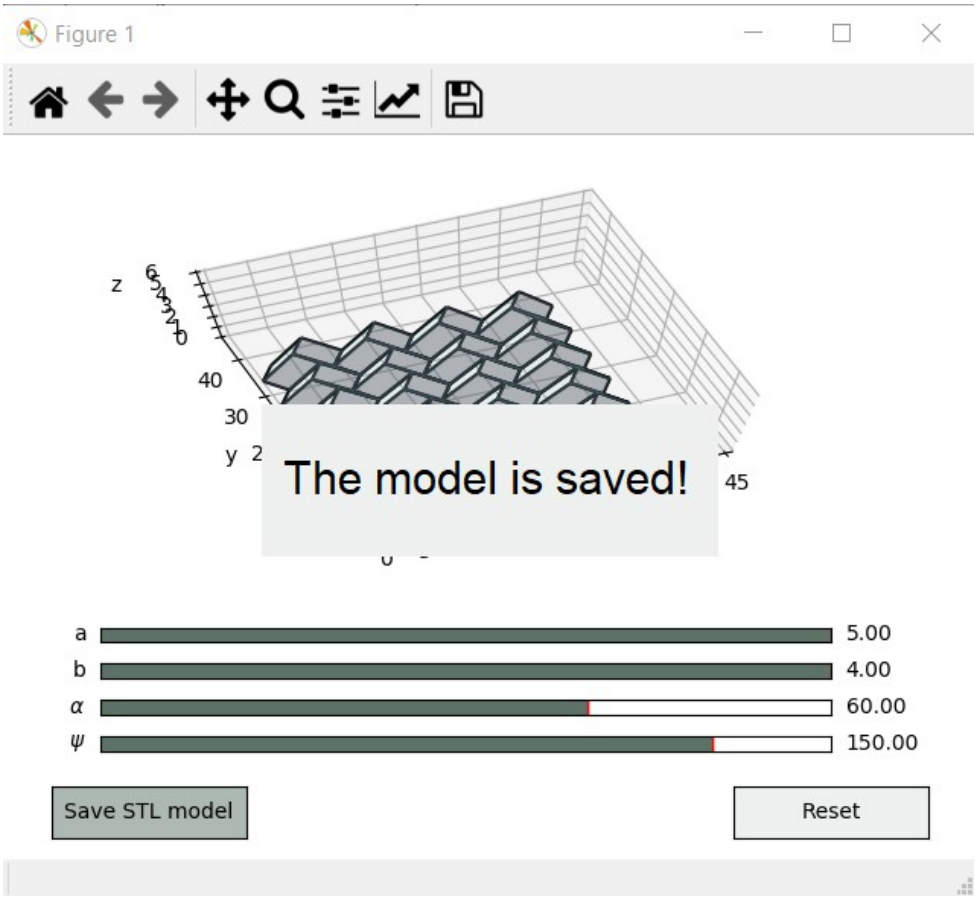

Рис. 24 — Иллюстрация всплывающего окна при успешном сохранении STL-модели на компьютер

Модель, полученная в результате работы программы, продемонстрирована на рисунках 25 и 26.

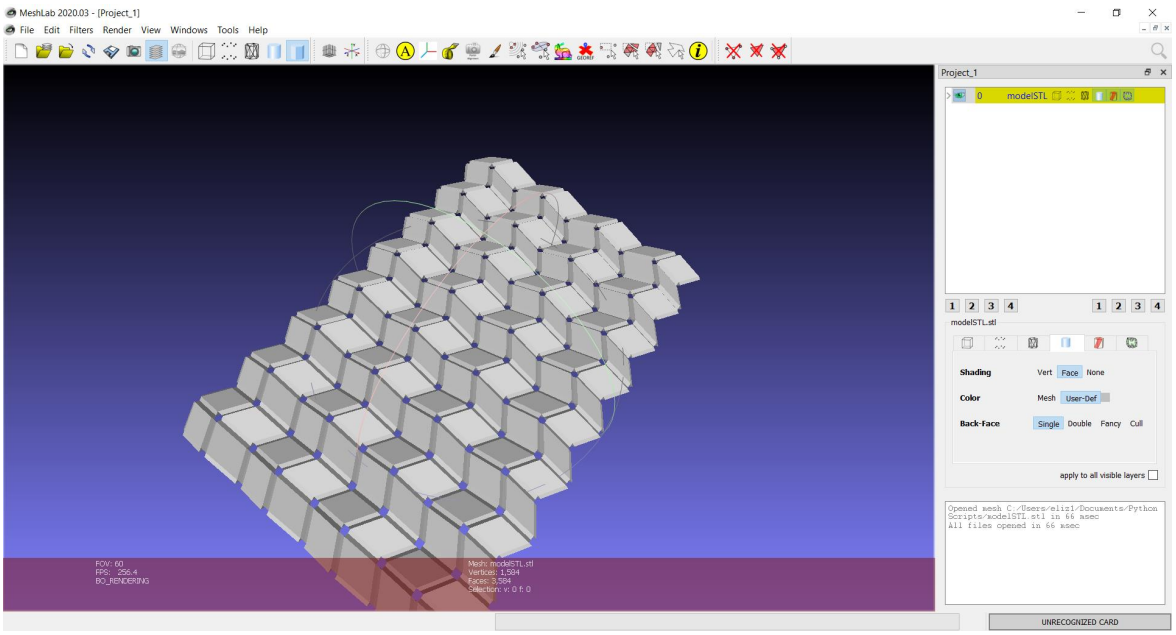

Рис. 25 — Общий вид созданной модели в интерфейсе программы MeshLab

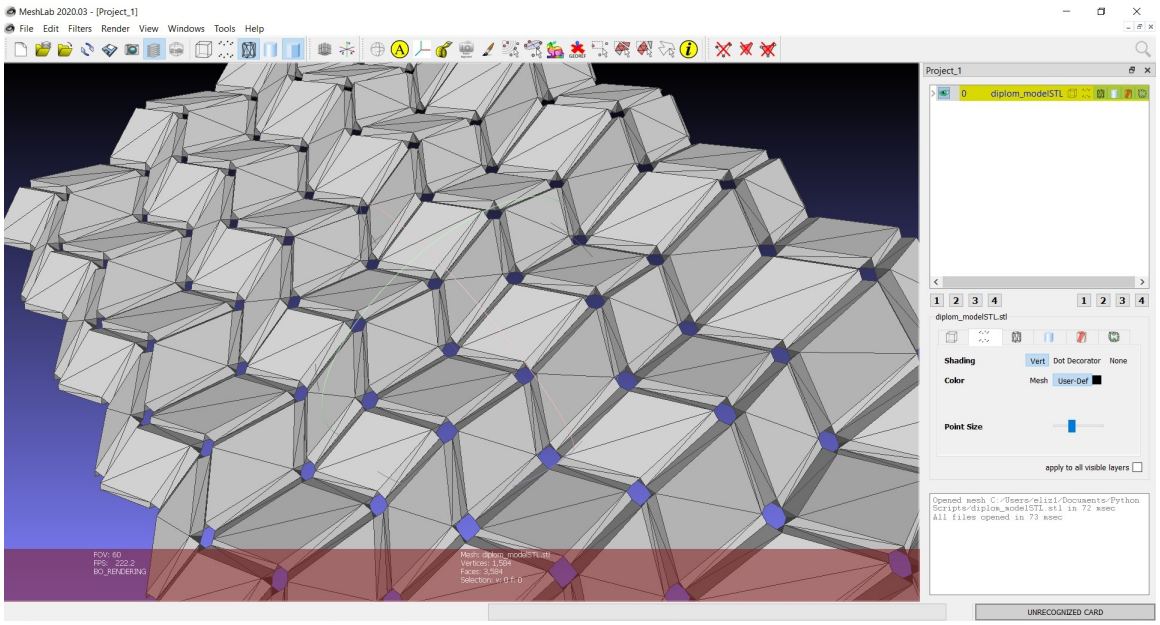

Рис. 26 — Приближенный вид модели в интерфейсе программы MeshLab

Заключение

В результате выполнения выпускной квалификационной работы были решены следующие задачи:

1. Описано движение вершин поверхности, параметризованной длинами сторон параллелограмма, углом α параллелограмма и двугранным углом сгиба ψ.
2. Смоделирован полученный результат.
3. Разработана трёхмерная модель по полученному результату.

Созданная модель поверхности позволяет наглядно увидеть особенности и свойства рассмотренного частного случая изгибаемых многогранных поверхностей. Таким образом, цель выпускной квалификационной работы считаю достигнутой.

Список литературы

1. Izmestiev, Ivan. (2016). Classification of Flexible Kokotsakis Polyhedra with Quadrangular Base. International Mathematics. Research Notices. 2017. 10.1093/imrn/rnw055.

2. Nishiyama, Yutaka. (2012). Miura folding: Applying origami to space exploration. International Journal of Pure and Applied Mathematics. 79. 269-279.

3. Karpenkov, Oleg. (2009). On the flexibility of Kokotsakis meshes. Geometriae Dedicata. 147. 10.1007/s10711-009-9436-4.

4. Яблокова С.И. Лекции по курсу "Аналитическая геометрия". Часть 1: Учебное пособие / Яросл. гос. ун-т. Ярославль, 2002. 108 с.

5. Яблокова С.И. Лекции по курсу "Аналитическая геометрия". Часть 2: Учебное пособие / Яросл. гос. ун-т. Ярославль, 2003. 112 с.

6. Hunter, J. D. Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9. 3. 90–95. Matplotlib is a 2D graphics package used for Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems. IEEE COMPUTER SOC. 10.1109/MCSE.2007.55. 2007.

7. Библиотека Numpy-STL 2.11.2. URL: `https://pypi.org/project/numpy-stl/`.

8. The STL Format. URL: `http://www.fabbers.com/tech/STL_Format`.

Приложение А

Исходный код программы на Python

```
%matplotlib qt


# pip install numpy-stl


import numpy as np
import matplotlib.pyplot as plt

from matplotlib.collections import LineCollection
from matplotlib import colors as mcolors
import mpl_toolkits.mplot3d as plot3d
import matplotlib.animation as animation

from matplotlib.widgets import Slider, Button

import stl
from stl import mesh

from tkinter import Label, messagebox, Tk


a = 5
b = 4
Alpha = 45
Psi = 120

# Количество параллелограммов по y
n = 8
# Количество параллелограммов по x
m = 4

global slider_a
global slider_b
global slider_Alpha
```

```
36 global slider_Psi
37
38 # Точки поверхности
39 polygons1 = list()
40 polygons2 = list()
41
42 # Счетчик количества треугольников в stl-модели
43 global k
44
45 # Высота прямой призмы
46 e = 1
47 # Ширина раздвижения параллелограммов
48 l = e
49
50 # Хранилище для набора треугольников stl-модели
51 data = np.zeros((n * m * 4 * 28), dtype = mesh.Mesh.dtype)
52
53
54 def CalkParam(a, b, Alpha, Psi):
55
56     SinOmega = np.sin(np.radians(Alpha)) * np.sin(
57         np.radians(Psi / 2))
58     CosOmega = np.sqrt(1 - np.power(SinOmega, 2))
59
60     A = [b * CosOmega, -b * SinOmega, 0]
61     # print("Координаты точки A: ", A)
62     E = [b * CosOmega, b * SinOmega, 0]
63     # print("Координаты точки E: ", E)
64
65     CosBeta = np.cos(np.radians(Alpha)) / CosOmega
66     SinBeta = np.sqrt(1 - np.power(CosBeta, 2))
67
68     C = [a * CosBeta, 0, a * SinBeta]
69     # print("Координаты точки C: ", C)
70     F = [C[0] + E[0], C[1] + E[1], C[2] + E[2]]
71     # print("Координаты точки F: ", F)
72     D = [C[0] + A[0], C[1] + A[1], C[2] + A[2]]
73     # print("Координаты точки D: ", D)
74     G = [2 * C[0], 0, 0]
75     # print("Координаты точки G: ", G)
```

```
76      K = [F[0] + C[0], F[1] - C[1], F[2] - C[2]]
77      # print("Координаты точки K: ", K)
78      L = [D[0] + C[0], D[1] - C[1], D[2] - C[2]]
79      # print("Координаты точки L: ", L)
80
81      lengthAE = np.sqrt(np.power(E[0] - A[0], 2)
82                         + np.power(E[1] - A[1], 2)
83                         + np.power(E[2] - A[2], 2))
84      # print("Сдвиг по y:", AEdl)
85      lengthBG = np.sqrt(np.power(G[0], 2)
86                         + np.power(G[1], 2)
87                         + np.power(G[2], 2))
88      # print("Сдвиг по x:", BGdl)
89
90      return A, E, C, F, D, G, K, L, lengthAE, lengthBG
91
92
93  def Draw(A, E, C, F, D, G, K, L, lengthAE, lengthBG):
94
95      polygons1.clear()
96      polygons2.clear()
97
98      for i in range(n):
99          for j in range(m):
100             polygons1.append([(0 + j * lengthBG,
101                                0 + i * lengthAE, 0),
102                                (C[0] + j * lengthBG,
103                                C[1] + i * lengthAE, C[2]),
104                                (D[0] + j * lengthBG,
105                                D[1] + i * lengthAE, D[2]),
106                                (A[0] + j * lengthBG,
107                                A[1] + i * lengthAE, A[2])])
108             polygons2.append([(E[0] + j * lengthBG,
109                                E[1] + i * lengthAE, E[2]),
110                                (F[0] + j * lengthBG,
111                                F[1] + i * lengthAE, F[2]),
112                                (C[0] + j * lengthBG,
113                                C[1] + i * lengthAE, C[2]),
114                                (0 + j * lengthBG,
115                                0 + i * lengthAE, 0)])
```

```
            polygons1.append([(F[0] + j * lengthBG,
                               F[1] + i * lengthAE, F[2]),
                               (K[0] + j * lengthBG,
                               K[1] + i * lengthAE, K[2]),
                               (G[0] + j * lengthBG,
                               G[1] + i * lengthAE, G[2]),
                               (C[0] + j * lengthBG,
                               C[1] + i * lengthAE, C[2])])
            polygons2.append([(C[0] + j * lengthBG,
                               C[1] + i * lengthAE, C[2]),
                               (G[0] + j * lengthBG,
                               G[1] + i * lengthAE, G[2]),
                               (L[0] + j * lengthBG,
                               L[1] + i * lengthAE, L[2]),
                               (D[0] + j * lengthBG,
                               D[1] + i * lengthAE, D[2])])

    polygonsline = list()

    for i in range(n):
        for j in range(m):
            polygonsline.append([(0 + j * lengthBG,
                               0 + i * lengthAE, 0),
                               (A[0] + j * lengthBG,
                               A[1] + i * lengthAE, A[2])])
            polygonsline.append([(A[0] + j * lengthBG,
                               A[1] + i * lengthAE, A[2]),
                               (D[0] + j * lengthBG,
                               D[1] + i * lengthAE, D[2])])
            polygonsline.append([(D[0] + j * lengthBG,
                               D[1] + i * lengthAE, D[2]),
                               (C[0] + j * lengthBG,
                               C[1] + i * lengthAE, C[2])])
            polygonsline.append([(C[0] + j * lengthBG,
                               C[1] + i * lengthAE, C[2]),
                               (0 + j * lengthBG,
                               0 + i * lengthAE, 0)])
            polygonsline.append([(0 + j * lengthBG,
                               0 + i * lengthAE, 0),
                               (E[0] + j * lengthBG,
```

```
                                E[1] + i * lengthAE, E[2])])
            polygonsline.append([(E[0] + j * lengthBG,
                                E[1] + i * lengthAE, E[2]),
                                (F[0] + j * lengthBG,
                                F[1] + i * lengthAE, F[2])])
            polygonsline.append([(F[0] + j * lengthBG,
                                F[1] + i * lengthAE, F[2]),
                                (C[0] + j * lengthBG,
                                C[1] + i * lengthAE, C[2])])
            polygonsline.append([(F[0] + j * lengthBG,
                                F[1] + i * lengthAE, F[2]),
                                (K[0] + j * lengthBG,
                                K[1] + i * lengthAE, K[2])])
            polygonsline.append([(K[0] + j * lengthBG,
                                K[1] + i * lengthAE, K[2]),
                                (G[0] + j * lengthBG,
                                G[1] + i * lengthAE, G[2])])
            polygonsline.append([(C[0] + j * lengthBG,
                                C[1] + i * lengthAE, C[2]),
                                (G[0] + j * lengthBG,
                                G[1] + i * lengthAE, G[2])])
            polygonsline.append([(G[0] + j * lengthBG,
                                G[1] + i * lengthAE, G[2]),
                                (L[0] + j * lengthBG,
                                L[1] + i * lengthAE, L[2])])
            polygonsline.append([(D[0] + j * lengthBG,
                                D[1] + i * lengthAE, D[2]),
                                (L[0] + j * lengthBG,
                                L[1] + i * lengthAE, L[2])])

    poly_collection1 = plot3d.art3d.Poly3DCollection(polygons1)
    # rgb(71,81,91) / 255
    poly_collection1.set_color((0.28, 0.32, 0.36, 0.4))

    line_collection = plot3d.art3d.Line3DCollection(
        polygonsline, colors = 'black')

    poly_collection2 = plot3d.art3d.Poly3DCollection(polygons2)
    # rgb(97,179,179) / 255
    poly_collection2.set_color((0.38, 0.7, 0.7, 0.1))
```

```
196
197     # Отрисовка плоскости в программе
198     ax = plt.axes(projection='3d')
199
200     ax.add_collection3d(line_collection)
201     ax.add_collection3d(poly_collection1)
202     ax.add_collection3d(poly_collection2)
203
204     ax.set_xlim((0, 45))
205     ax.set_ylim((-5, 45))
206     ax.set_zlim((0, 6))
207
208     ax.set_xlabel('x')
209     ax.set_ylabel('y')
210     ax.set_zlabel('z')
211
212     # Центрирование отображения окна
213     root = Tk()
214     root.withdraw()
215
216     screen_width = root.winfo_screenwidth()
217     screen_height = root.winfo_screenheight()
218
219     x_old, y_old, width, height =
220         plt.get_current_fig_manager().window.geometry()
221         .getRect()
222     plt.get_current_fig_manager().window.setGeometry(
223         (screen_width - width)/2,
224         (screen_height - height)/2,
225         width, height)
226
227     # Отрисовка поверхности
228     plt.show()
229
230
231 # Формирование файла STL для плоскости
232 def ModelSTL(polygons1, polygons2):
233
234     polygonsAll = list()
235
```

```
    # Раздвигаем параллелограммы
    p = 0
    for i in range(0, n * 2, 2):
        for j in range(0, m * 2, 2):
            # 1
            polygonsAll.append([(polygons1[p][0][0] + j * l,
                                 polygons1[p][0][1] + i * l,
                                 polygons1[p][0][2]),
                                 (polygons1[p][1][0] + j * l,
                                 polygons1[p][1][1] + i * l,
                                 polygons1[p][1][2]),
                                 (polygons1[p][2][0] + j * l,
                                 polygons1[p][2][1] + i * l,
                                 polygons1[p][2][2]),
                                 (polygons1[p][3][0] + j * l,
                                 polygons1[p][3][1] + i * l,
                                 polygons1[p][3][2])])

            # 2
            polygonsAll.append([(polygons2[p][0][0] + j * l,
                                 polygons2[p][0][1] + (i + 1) * l,
                                 polygons2[p][0][2]),
                                 (polygons2[p][1][0] + j * l,
                                 polygons2[p][1][1] + (i + 1) * l,
                                 polygons2[p][1][2]),
                                 (polygons2[p][2][0] + j * l,
                                 polygons2[p][2][1] + (i + 1) * l,
                                 polygons2[p][2][2]),
                                 (polygons2[p][3][0] + j * l,
                                 polygons2[p][3][1] + (i + 1) * l,
                                 polygons2[p][3][2])])

            # 3
            polygonsAll.append([(polygons1[p + 1][0][0]
                                 + (j + 1) * l,
                                 polygons1[p + 1][0][1]
                                 + (i + 1) * l,
                                 polygons1[p + 1][0][2]),
                                 (polygons1[p + 1][1][0]
                                 + (j + 1) * l,
```

```
                                polygons1[p + 1][1][1]
                                + (i + 1) * l,
                                polygons1[p + 1][1][2]),
                                (polygons1[p + 1][2][0]
                                + (j + 1) * l,
                                polygons1[p + 1][2][1]
                                + (i + 1) * l,
                                polygons1[p + 1][2][2]),
                                (polygons1[p + 1][3][0]
                                + (j + 1) * l,
                                polygons1[p + 1][3][1]
                                + (i + 1) * l,
                                polygons1[p + 1][3][2])])

            # 4
            polygonsAll.append([(polygons2[p + 1][0][0]
                                + (j + 1) * l,
                                polygons2[p + 1][0][1] + i * l,
                                polygons2[p + 1][0][2]),
                                (polygons2[p + 1][1][0]
                                + (j + 1) * l,
                                polygons2[p + 1][1][1] + i * l,
                                polygons2[p + 1][1][2]),
                                (polygons2[p + 1][2][0]
                                + (j + 1) * l,
                                polygons2[p + 1][2][1] + i * l,
                                polygons2[p + 1][2][2]),
                                (polygons2[p + 1][3][0]
                                + (j + 1) * l,
                                polygons2[p + 1][3][1] + i * l,
                                polygons2[p + 1][3][2])])

            p = p + 2

    # Создаем элементы трехмерной поверхности
    k = 0

    for i in range(n * m * 4):
        k = CreateParallelogram(polygonsAll[i], k)
```

```
316     # Создаём сетку из треугольников поверхности
317     coordinatesMesh = mesh.Mesh(data, remove_empty_areas = False)
318 
319     # Для всех треугольников сетки добавляем нормали
320     coordinatesMesh.normals
321 
322     # Создаем вектора для доступа к вершинам треугольников
323     coordinatesMesh.v0, coordinatesMesh.v1, coordinatesMesh.v2
324 
325     assert (coordinatesMesh.points[0][0:3]
326             == coordinatesMesh.v0[0]).all()
327     assert (coordinatesMesh.points[0][3:6]
328             == coordinatesMesh.v1[0]).all()
329     assert (coordinatesMesh.points[0][6:9]
330             == coordinatesMesh.v2[0]).all()
331     assert (coordinatesMesh.points[1][0:3]
332             == coordinatesMesh.v0[1]).all()
333 
334     # Сохраняем модель в кодировке ASCII
335     coordinatesMesh.save('DiplomModelSTL.stl',
336                          mode = stl.Mode.ASCII)
337 
338 
339 # Моделирование одного элемента трехмерной плоскости
340 def CreateParallelogram(parallelogram, k):
341 
342     # Сдвиг вниз
343     parallelogram_down = [(parallelogram[0][0],
344                            parallelogram[0][1],
345                            parallelogram[0][2] - e),
346                            (parallelogram[1][0],
347                            parallelogram[1][1],
348                            parallelogram[1][2] - e),
349                            (parallelogram[2][0],
350                            parallelogram[2][1],
351                            parallelogram[2][2] - e),
352                            (parallelogram[3][0],
353                            parallelogram[3][1],
354                            parallelogram[3][2] - e)]
355 
```

```
356     # Заносим исходный и сдвинутый вниз на e
357     data['vectors'][k] = np.array([parallelogram[0],
358                                    parallelogram[3],
359                                    parallelogram[2]])
360     data['vectors'][k + 1] = np.array([parallelogram[0],
361                                    parallelogram[2],
362                                    parallelogram[1]])
363     data['vectors'][k + 2] = np.array([parallelogram_down[0],
364                                    parallelogram_down[2],
365                                    parallelogram_down[3]])
366     data['vectors'][k + 3] = np.array([parallelogram_down[0],
367                                    parallelogram_down[1],
368                                    parallelogram_down[2]])
369
370     k = k + 4
371
372     # Добавление призм по бокам
373     H = (parallelogram[3][0], parallelogram[3][1] - (l / 2),
374         parallelogram[3][2] - (e / 2))
375     H1 = (parallelogram[2][0], parallelogram[2][1] - (l / 2),
376         parallelogram[2][2] - (e / 2))
377
378     data['vectors'][k] = np.array([H, parallelogram[3],
379                                    parallelogram_down[3]])
380     data['vectors'][k + 1] = np.array([H1,
381                                    parallelogram_down[2],
382                                    parallelogram[2]])
383
384     data['vectors'][k + 2] = np.array([H,
385                                    parallelogram_down[2], H1])
386     data['vectors'][k + 3] = np.array([H,
387                                    parallelogram_down[3],
388                                    parallelogram_down[2]])
389
390     data['vectors'][k + 4] = np.array([H1,
391                                    parallelogram[3], H])
392     data['vectors'][k + 5] = np.array([H1, parallelogram[2],
393                                    parallelogram[3]])
394
395     k = k + 6
```

```
396
397     W = (parallelogram[0][0], parallelogram[0][1] + (l / 2),
398         parallelogram[0][2] - (e / 2))
399     W1 = (parallelogram[1][0], parallelogram[1][1] + (l / 2),
400         parallelogram[1][2] - (e / 2))
401
402     data['vectors'][k] = np.array([W, parallelogram_down[0],
403                                    parallelogram[0]])
404     data['vectors'][k + 1] = np.array([W1, parallelogram[1],
405                                    parallelogram_down[1]])
406
407     data['vectors'][k + 2] = np.array([W,
408                                    parallelogram[0], W1])
409     data['vectors'][k + 3] = np.array([W1, parallelogram[0],
410                                    parallelogram[1]])
411
412     data['vectors'][k + 4] = np.array([W1,
413                                    parallelogram_down[1], W])
414     data['vectors'][k + 5] = np.array([W,
415                                    parallelogram_down[1],
416                                    parallelogram_down[0]])
417
418     k = k + 6
419
420     S = (parallelogram[2][0] + (l / 2), parallelogram[2][1],
421         parallelogram[2][2] - (e / 2))
422     S1 = (parallelogram[1][0] + (l / 2), parallelogram[1][1],
423         parallelogram[1][2] - (e / 2))
424
425     data['vectors'][k] = np.array([S, parallelogram[2],
426                                    parallelogram_down[2]])
427     data['vectors'][k + 1] = np.array([S1,
428                                    parallelogram_down[1],
429                                    parallelogram[1]])
430
431     data['vectors'][k + 2] = np.array([S,
432                                    parallelogram_down[2], S1])
433     data['vectors'][k + 3] = np.array([S1,
434                                    parallelogram_down[2],
435                                    parallelogram_down[1]])
```

```
436
437     data['vectors'][k + 4] = np.array([S1,
438                                 parallelogram[1], S])
439     data['vectors'][k + 5] = np.array([S, parallelogram[1],
440                                 parallelogram[2]])
441
442     k = k + 6
443
444     T = (parallelogram[3][0] - (l / 2), parallelogram[3][1],
445         parallelogram[3][2] - (e / 2))
446     T1 = (parallelogram[0][0] - (l / 2), parallelogram[0][1],
447         parallelogram[0][2] - (e / 2))
448
449     data['vectors'][k] = np.array([T, parallelogram_down[3],
450                                 parallelogram[3]])
451     data['vectors'][k + 1] = np.array([T1, parallelogram[0],
452                                 parallelogram_down[0]])
453
454     data['vectors'][k + 2] = np.array([T,
455                                 parallelogram[0], T1])
456     data['vectors'][k + 3] = np.array([T, parallelogram[3],
457                                 parallelogram[0]])
458
459     data['vectors'][k + 4] = np.array([T1,
460                                 parallelogram_down[3], T])
461     data['vectors'][k + 5] = np.array([T1,
462                                 parallelogram_down[0],
463                                 parallelogram_down[3]])
464
465     k = k + 6
466
467     return k
468
469
470 def SaveSTL(event):
471     ModelSTL(polygons1, polygons2)
472
473     # Создание всплывающего окна "The model is saved!"
474     window = Tk()
475     x = (window.winfo_screenwidth() - 300) / 2
```

```
476     y = (window.winfo_screenheight() - 100) / 2
477     window.wm_geometry("+%d+%d" % (x, y))
478     window.title("Information")
479     window.overrideredirect(1)
480     window.geometry('300x100')
481     window.configure(background='#EEF1EF')
482     lbl = Label(window, text="The model is saved!",
483                 font=("Arial Bold", 18),
484                 background = '#EEF1EF')
485     lbl.place(x=11,y=28)
486     window.after(1000, window.destroy)
487     window.mainloop()
488
489 def Reset(event):
490     slider_a.reset()
491     slider_b.reset()
492     slider_Alpha.reset()
493     slider_Psi.reset()
494
495 # Кнопка для сохранения трёхмерной модели
496 resetax = plt.axes([0.05, 0.03, 0.2, 0.07])
497 button = Button(resetax, 'Save STL model', color='#EEF1EF',
498                 hovercolor='#AEB9B3')
499
500 button.on_clicked(SaveSTL)
501
502 # Кнопка, строящая поверхность по изначально заданным параметрам
503 resetaxStart = plt.axes([0.75, 0.03, 0.2, 0.07])
504 buttonStart = Button(resetaxStart, 'Reset', color='#EEF1EF',
505                      hovercolor='#AEB9B3')
506
507 buttonStart.on_clicked(Reset)
508
509 # Обработка изменений значений слайдеров
510 def onChangeValue(val):
511
512     # Атрибут val получает значение слайдеров
513     a = slider_a.val
514     b = slider_b.val
515     Alpha = slider_Alpha.val
```

```
516     Psi = slider_Psi.val
517 
518     A, E, C, F, D, G, K, L, lengthAE, lengthBG
519     = CalkParam(a, b, Alpha, Psi)
520     Draw(A, E, C, F, D, G, K, L, lengthAE, lengthBG)
521 
522 # Место для виджетов
523 plt.subplots_adjust(left=0.1, right=0.85, top=0.95, bottom=0.4)
524 
525 # a
526 axes_slider_a = plt.axes([0.1, 0.3, 0.75, 0.02])
527 slider_a = Slider(axes_slider_a,
528     label = 'a',
529     valmin = 0.0,
530     valmax = 5.0,
531     valinit = a,
532     valfmt = '%1.2f',
533     color = '#5E7367')
534 
535 # Событие on_changed отслеживает, что значение слайдера изменилось
536 # Принимает один параметр, который будет новым значением,
537 # установленным на слайдере
538 slider_a.on_changed(onChangeValue)
539 
540 # b
541 axes_slider_b = plt.axes([0.1, 0.25, 0.75, 0.02])
542 slider_b = Slider(axes_slider_b,
543     label = 'b',
544     valmin = 0.0,
545     valmax = 4.0,
546     valinit = b,
547     valfmt = '%1.2f',
548     color = '#5E7367')
549 
550 slider_b.on_changed(onChangeValue)
551 
552 # Alpha
553 axes_slider_Alpha = plt.axes([0.1, 0.20, 0.75, 0.02])
554 slider_Alpha = Slider(axes_slider_Alpha,
555     label = r'$\alpha$',
```

```
556     valmin = 0.0,
557     valmax = 90.0,
558     valinit = Alpha,
559     valfmt = '%1.2f',
560     color = '#5E7367')
561
562 slider_Alpha.on_changed(onChangeValue)
563
564 # Psi
565 axes_slider_Psi = plt.axes([0.1, 0.15, 0.75, 0.02])
566 slider_Psi = Slider(axes_slider_Psi,
567     label = r'$\psi$',
568     valmin = 0.0,
569     valmax = 179.0,
570     valinit = Psi,
571     valfmt = '%1.2f',
572     color = '#5E7367')
573
574 slider_Psi.on_changed(onChangeValue)
575
576 A, E, C, F, D, G, K, L, lengthAE, lengthBG
577 = CalkParam(a, b, Alpha, Psi)
578 Draw(A, E, C, F, D, G, K, L, lengthAE, lengthBG)
```